Projekt “Restauracje”

Zespół 1. Sylwia Marek, Barbara Wojtarowicz

Przedmiot Podstawy Baz Danych

Spis treści

0. Opis

Projekt dotyczy systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. System jest dostosowany do równoległego użytkowania przez kilka tego typu firm.

Ogólne informacje

W ofercie jest żywność oraz napoje bezalkoholowe. Usługi świadczone są na miejscu oraz na wynos. Zamówienie na wynos może być zlecone na miejscu lub z wyprzedzeniem (z wykorzystaniem formularza WWW i wyboru preferowanej daty i godziny odbioru zamówienia). Firma dysponuje ograniczoną liczbą stolików, w tym miejsc siedzących. Istnieje możliwość wcześniejszej rezerwacji stolika dla co najmniej dwóch osób.

Z uwagi na zmieniające się ograniczenia związane z COVID-19, w poszczególnych dniach może być dostępna ograniczona liczba miejsc (w odniesieniu do powierzchni lokalu), zmienna w czasie.

Klientami są osoby indywidualne oraz firmy, odbierające większe ilości posiłków w porze lunchu lub jako catering (bez dostawy). Dla firm istnieje możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc.

Menu

Menu ustalane jest z dziennym wyprzedzeniem. W firmie panuje zasada, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie, przy czym pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc.

Ponadto, w dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Takie zamówienie powinno być złożone co maksymalnie do poniedziałku poprzedzającego zamówienie.
Istnieje możliwość, że pozycja w menu zostanie usunięta na skutek wyczerpania się półproduktów.

Wcześniejsza rezerwacja zamówienia/stolika

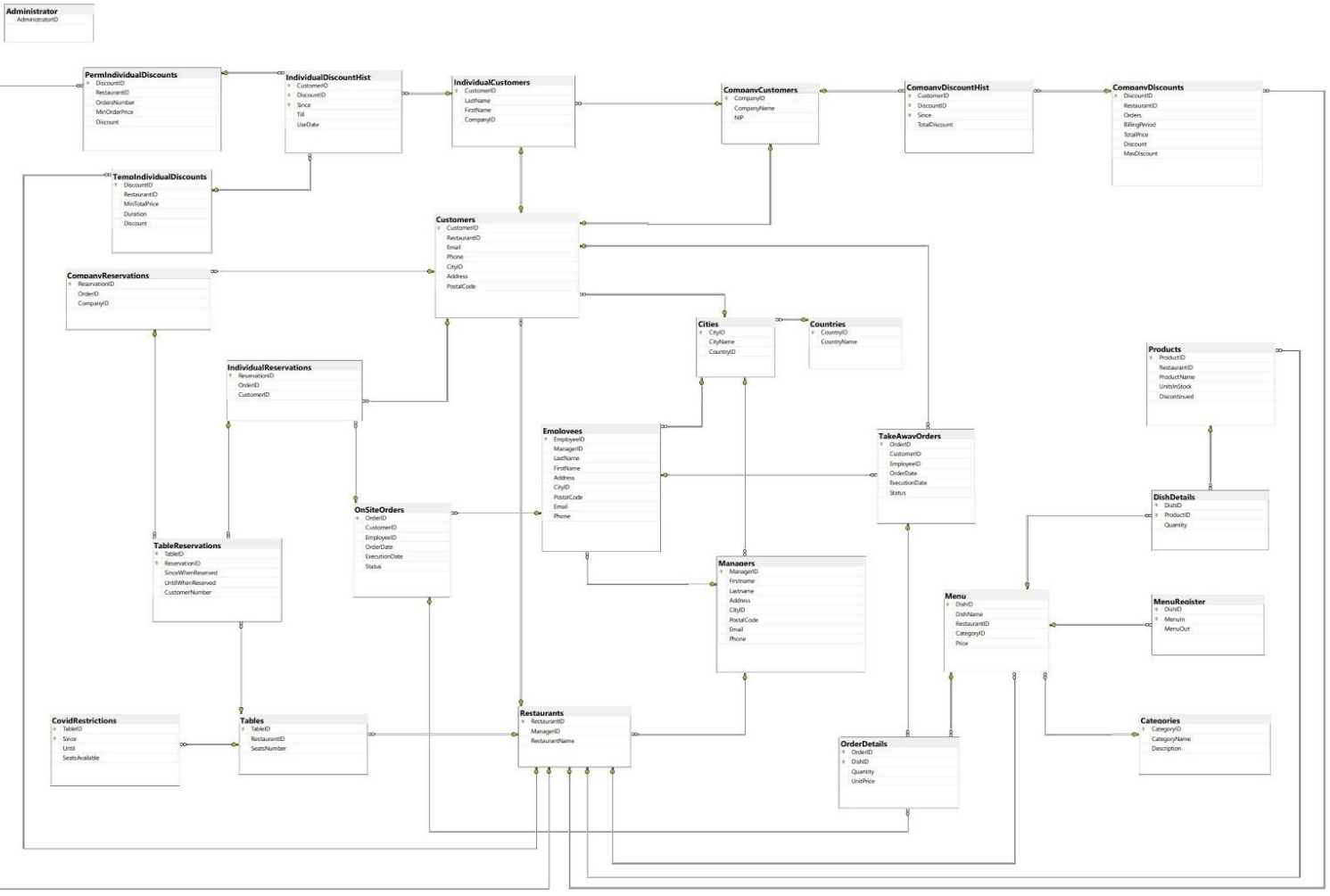
Internetowy formularz umożliwia klientowi indywidualnemu rezerwację stolika, przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu, przy minimalnej wartości zamówienia 50 zł, w przypadku klientów, którzy dokonali wcześniej co najmniej 5 zamówień i/lub mniej, ale w tym przypadku na kwotę co najmniej 200 zł. Informacja wraz z potwierdzeniem zamówienia oraz wskazaniem stolika. Wysyłana jest po akceptacji przez obsługę.

Internetowy formularz umożliwia także rezerwację stolików dla firm, w dwóch opcjach: rezerwacji stolików na firmę i/lub rezerwację stolików dla konkretnych pracowników firmy (imiennie).

Rabaty

System umożliwia realizację programów rabatowych dla klientów indywidualnych (tymczasowych oraz permanentnych) oraz dla klientów firmowych.

1. Schemat bazy danych



2. Opisy tabel wraz z warunkami integralności i kodem je generującym

1. Tabela Administrator

Reprezentuje administratorów w bazie danych. Posiada klucz główny *AdministratorID* (int).

Warunki integralności:

- *AdministratorID* jest unikalne

```
CREATE TABLE [dbo].[Administrator](
      [AdministratorID] [int] NOT NULL
) ON [PRIMARY]
GO
```

2. Tabela Restaurants

Reprezentuje w bazie danych firmy świadczące usługi gastronomiczne. Posiada klucz główny *RestaurantID* (int), numer menedżera reprezentującego tę firmę *ManagerID* (int), który jest kluczem obcym do tabeli Managers oraz nazwę restauracji *RestaurantName* (nvarchar).

Warunki integralności:

- *RestaurantID* jest unikalne

```
CREATE TABLE [dbo].[Restaurants](
      [RestaurantID] [INT] NOT NULL,
      [ManagerID] [INT] NOT NULL,
      [RestaurantName] [NVARCHAR](50) NOT NULL,
 CONSTRAINT [PK_Restaurants] PRIMARY KEY CLUSTERED
(
      [RestaurantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [RestaurantsNameUnique] UNIQUE NONCLUSTERED
(
      [RestaurantName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Restaurants]
WITH CHECK ADD  CONSTRAINT [FK_Restaurants_Managers]
FOREIGN KEY([ManagerID])
REFERENCES [dbo].[Managers] ([ManagerID])
GO
```

```
ALTER TABLE [dbo].[Restaurants]
CHECK CONSTRAINT [FK_Restaurants_Managers]
GO
```

3. Tabela Managers

Reprezentuje w bazie danych menedżerów firm świadczących usługi gastronomiczne. Posiada klucz główny *ManagerID* (int), numer restauracji *RestaurantID* (int), imię, nazwisko, adres menedżera postaci *Firstname* (nvarchar), *Lastname* (nvarchar), *Address* (nvarchar) oraz kod miasta *CityID* (klucz obcy do tabeli Cities) (int), kod pocztowy *PostalCode* (nvarchar), numer telefonu *Phone* (nvarchar).

Warunki integralności:

- *ManagerID* jest unikalne
- *PostalCode* jest postaci XX-XXX, gdzie X to cyfra [0-9]
- *Phone* składa się z cyfr [0-9]

```
CREATE TABLE [dbo].[Managers](
	[ManagerID] [INT] NOT NULL,
	[Firstname] [NVARCHAR](50) NOT NULL,
	[Lastname] [NVARCHAR](50) NOT NULL,
	[Address] [NVARCHAR](50) NOT NULL,
	[CityID] [INT] NOT NULL,
	[PostalCode] [NVARCHAR](50) NOT NULL,
	[Email] [NVARCHAR](50) NOT NULL,
	[Phone] [NVARCHAR](50) NOT NULL,
 CONSTRAINT [PK_Managers] PRIMARY KEY CLUSTERED
(
	[ManagerID] 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].[Managers]
WITH CHECK ADD  CONSTRAINT [FK_Managers_Cities]
FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Managers]
```

```
CHECK CONSTRAINT [FK_Managers_Cities]
GO

ALTER TABLE [dbo].[Managers]
WITH CHECK ADD  CONSTRAINT [CK_Managers_Email]
CHECK  (([Email] LIKE '%@%.%'))
GO

ALTER TABLE [dbo].[Managers]
CHECK CONSTRAINT [CK_Managers_Email]
GO

ALTER TABLE [dbo].[Managers]
WITH CHECK ADD  CONSTRAINT [CK_Managers_Phone]
CHECK  ((ISNUMERIC([Phone])=(1)))
GO

ALTER TABLE [dbo].[Managers]
CHECK CONSTRAINT [CK_Managers_Phone]
GO

ALTER TABLE [dbo].[Managers]
WITH CHECK ADD  CONSTRAINT [CK_Managers_PostalCode]
CHECK  (([PostalCode] LIKE '[0-9][0-9]-[0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Managers]
CHECK CONSTRAINT [CK_Managers_PostalCode]
GO
```

4. Tabela Employees

Reprezentuje w bazie danych pracowników firm świadczących usługi gastronomiczne. Posiada klucz główny *EmployeeID* (int), numer menedżera, któremu ten pracownik podlega - *ManagerID* (int) (klucz obcy do tabeli Managers), imię, nazwisko, adres pracownika postaci *FirstName* (nvarchar), *LastName* (nvarchar), *Address* (nvarchar) oraz numer miasta *CityID* (int) (klucz obcy do tabeli Cities), kod pocztowy *PostalCode* (nvarchar), numer telefonu *Phone* (nvarchar).

Warunki integralności:

- *EmployeeID* jest unikalne
- *PostalCode* jest postaci XX-XXX, gdzie X to cyfra [0-9]
- *Phone* składa się z cyfr [0-9]

```
CREATE TABLE [dbo].[Employees](
	[EmployeeID] [INT] NOT NULL,
	[ManagerID] [INT] NOT NULL,
	[LastName] [NVARCHAR](50) NOT NULL,
	[FirstName] [NVARCHAR](50) NOT NULL,
	[Address] [NVARCHAR](50) NOT NULL,
	[CityID] [INT] NOT NULL,
	[PostalCode] [NVARCHAR](50) NOT NULL,
	[Email] [NVARCHAR](50) NOT NULL,
	[Phone] [NVARCHAR](50) NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

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

ALTER TABLE [dbo].[Employees]
WITH CHECK ADD  CONSTRAINT [FK_Employees_Managers]
FOREIGN KEY([ManagerID])
REFERENCES [dbo].[Managers] ([ManagerID])
GO

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

ALTER TABLE [dbo].[Employees]
WITH CHECK ADD  CONSTRAINT [CK_Employees_Email]
CHECK  (([Email] LIKE '%@%.%'))
GO

ALTER TABLE [dbo].[Employees]
CHECK CONSTRAINT [CK_Employees_Email]
```

```
GO

ALTER TABLE [dbo].[Employees]
WITH CHECK ADD  CONSTRAINT [CK_Employees_Phone]
CHECK  ((ISNUMERIC([Phone])=(1)))
GO

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

ALTER TABLE [dbo].[Employees]
WITH CHECK ADD  CONSTRAINT [CK_Employees_PostalCode]
CHECK  (([PostalCode] LIKE '[0-9][0-9]-[0-9][0-9][0-9]'))
GO

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

5. Tabela Customers

Reprezentuje klientów w bazie danych. Posiada klucz główny *CustomerID* (int), będący identyfikatorem klienta, numer restauracji, której jest on klientem *RestaurantID* (int) (klucz obcy do tabeli Restaurants), adres mailowy *Email* (nvarchar), numer telefonu *Phone* (nvarchar), numer miasta *CityID* (int) (klucz obcy do tabeli Cities), adres *Address* (nvarchar) oraz kod pocztowy *PostalCode* (nvarchar).

Warunki integralności:

- *CustomerID* jest unikalne
- *Email* jest unikalny i zawiera znaki '@' oraz '.'
- *Phone* jest wartością numeryczną
- *PostalCode* jest postaci XX-XXX, gdzie X to cyfra [0-9]

```
CREATE TABLE [dbo].[Customers](
      [CustomerID] [INT] IDENTITY(1,1) NOT NULL,
      [RestaurantID] [INT] NOT NULL,
      [Email] [NVARCHAR](50) NOT NULL,
      [Phone] [NVARCHAR](50) NOT NULL,
      [CityID] [INT] NOT NULL,
      [Address] [NVARCHAR](50) NOT NULL,
      [PostalCode] [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

ALTER TABLE [dbo].[Customers]
WITH CHECK ADD  CONSTRAINT [FK_Customers_Cities]
FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

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

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

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

ALTER TABLE [dbo].[Customers]
WITH CHECK ADD  CONSTRAINT [CK_Customers_Email]
CHECK  (([Email] LIKE '%@%.%'))
GO

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

ALTER TABLE [dbo].[Customers]
WITH CHECK ADD  CONSTRAINT [CK_Customers_Phone]
CHECK  ((ISNUMERIC([Phone])=(1)))
GO

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

ALTER TABLE [dbo].[Customers]
```

```
WITH CHECK ADD  CONSTRAINT [CK_Customers_PostalCode]
CHECK  (([PostalCode] LIKE '[0-9][0-9]-[0-9][0-9][0-9]'))
GO

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

6. Tabela CompanyCustomers

Reprezentuje rejestr klientów-firm w bazie danych. Posiada klucz obcy do tabeli Customers *CompanyID* (int), nazwę firmy *CompanyName* (nvarchar) oraz numer *NIP* (nvarchar).

Warunki integralności:

- *NIP* jest unikalny

```
CREATE TABLE [dbo].[CompanyCustomers](
      [CompanyID] [INT] IDENTITY(1,1) NOT NULL,
      [CompanyName] [NVARCHAR](50) NOT NULL,
      [NIP] [NVARCHAR](50) NOT NULL,
 CONSTRAINT [PK_CompanyCustomers] PRIMARY KEY CLUSTERED
(
      [CompanyID] 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 [NIPUnique] 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].[CompanyCustomers]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyCustomers_Customers] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyCustomers] CHECK CONSTRAINT
[FK_CompanyCustomers_Customers]
GO
```

7. Tabela CompanyDiscountHist

Reprezentuje w bazie danych rejestr przyznawanych klientom-firmom rabatów. Posiada klucz główny będący kombinacją numeru klienta *CustomerID* (int) , numer rabatu *DiscountID* (smallint) oraz daty, w której ten rabat został przyznany *Since* (date).
Posiada całkowitą wartość rabatu *TotalDiscount* (real) w momencie określonym przez *Since.*
Posiada klucze obce: *CustomerID* dla tabeli CompanyCustomers oraz *DiscountID* dla tabeli CompanyDiscounts.

Warunki integralności:

- kombinacja *CustomerID*, *DiscountID* oraz *Since* jest unikalna
- *TotalDiscount* jest z przedziału [0-1]

```
CREATE TABLE [dbo].[CompanyDiscountHist](
      [CustomerID] [INT] IDENTITY(1,1) NOT NULL,
      [DiscountID] [SMALLINT] NOT NULL,
      [Since] [DATE] NOT NULL,
      [TotalDiscount] [REAL] NOT NULL,
 CONSTRAINT [PK_CompanyDiscountHist] PRIMARY KEY CLUSTERED
(
      [CustomerID] ASC,
      [DiscountID] ASC,
      [Since] 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].[CompanyDiscountHist]
WITH CHECK ADD  CONSTRAINT [FK_CompanyDiscountHist_CompanyCustomers]
FOREIGN KEY([CustomerID])
REFERENCES [dbo].[CompanyCustomers] ([CompanyID])
GO

ALTER TABLE [dbo].[CompanyDiscountHist]
CHECK CONSTRAINT [FK_CompanyDiscountHist_CompanyCustomers]
GO

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

ALTER TABLE [dbo].[CompanyDiscountHist]
CHECK CONSTRAINT [FK_CompanyDiscountHist_CompanyDiscounts]
```

```
GO

ALTER TABLE [dbo].[CompanyDiscountHist]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscountHist]
CHECK  (([TotalDiscount]>=(0) AND [TotalDiscount]<=(1)))
GO

ALTER TABLE [dbo].[CompanyDiscountHist]
CHECK CONSTRAINT [CK_CompanyDiscountHist]
GO
```

8. Tabela CompanyDiscounts

Stanowi słownik rabatów przyznawanych klientom-firmom. Posiada klucz główny *DiscountID* (smallint), który jest identyfikatorem rabatu oraz opis warunków, które klient firma musi spełnić, by uzyskać rabat: minimalną liczbę zamówień *Orders* (int), za łączną minimalną kwotę *TotalPrice* (money) w okresie czasu *BillingPeriod* (int) (ciągłość zamówień). Wartości w/w rabatów zawarte są w *Discount* (real), przy czym nie mogą one przekroczyć progu wartości maksymalnej danego rodzaju rabatu *MaxDiscount* (real).

Warunki integralności:

- *DiscountID* jest unikalne
- *Orders* jest liczbą nieujemną
- *BillingPeriod* jest liczbą nieujemną
- *TotalPrice* jest liczbą nieujemną
- *Discount* jest z przedziału [0-1]
- *MaxDiscount* jest z przedziału [0-1]
- *Discount* jest nie większe od *MaxDiscount*

```
CREATE TABLE [dbo].[CompanyDiscounts](
	[DiscountID] [SMALLINT] NOT NULL,
	[RestaurantID] [INT] NOT NULL,
	[Orders] [INT] NOT NULL,
	[BillingPeriod] [INT] NOT NULL,
	[TotalPrice] [MONEY] NOT NULL,
	[Discount] [REAL] NOT NULL,
	[MaxDiscount] [REAL] NOT NULL,
 CONSTRAINT [PK_CompanyDiscounts] PRIMARY KEY CLUSTERED
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO

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

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

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

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

ALTER TABLE [dbo].[CompanyDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscounts_1]
CHECK  (([BillingPeriod]>(0)))
GO

ALTER TABLE [dbo].[CompanyDiscounts]
CHECK CONSTRAINT [CK_CompanyDiscounts_1]
GO

ALTER TABLE [dbo].[CompanyDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscounts_2] CHECK
(([Discount]>(0) AND [Discount]<(1)))
GO

ALTER TABLE [dbo].[CompanyDiscounts]
CHECK CONSTRAINT [CK_CompanyDiscounts_2]
GO

ALTER TABLE [dbo].[CompanyDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscounts_3]
CHECK  (([MaxDiscount]>(0) AND [MaxDiscount]<(1)))
GO

ALTER TABLE [dbo].[CompanyDiscounts]
```

```
CHECK CONSTRAINT [CK_CompanyDiscounts_3]
GO

ALTER TABLE [dbo].[CompanyDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscounts_4]
CHECK  (([TotalPrice]>(0)))
GO

ALTER TABLE [dbo].[CompanyDiscounts]
CHECK CONSTRAINT [CK_CompanyDiscounts_4]
GO

ALTER TABLE [dbo].[CompanyDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_CompanyDiscounts_5]
CHECK  (([MaxDiscount]>=[Discount]))
GO

ALTER TABLE [dbo].[CompanyDiscounts]
CHECK CONSTRAINT [CK_CompanyDiscounts_5]
GO
```

9. Tabela CovidRestrictions

Zawiera obostrzenia związane z COVID-19. Posiada klucz główny będący kombinacją numeru stolika *TableID* (int) (klucz obcy do tabeli Tables), którego dotyczy obostrzenie oraz daty *Since* (date), od kiedy ono obowiązuje. Posiada także informację o liczbie dostępnych miejsc po nałożeniu ograniczeń *SeatsAvailable* (smallint) i datę, do kiedy obowiązuje *Until* (date).

Warunki integralności:

- kombinacja *TableID i Since* jest unikalna
- *Until* nie może nastąpić wcześniej niż *Since*
- *SeatsAvailable* jest liczbą nieujemną

```
CREATE TABLE [dbo].[CovidRestrictions](
      [TableID] [INT] NOT NULL,
      [Since] [DATETIME] NOT NULL,
      [Until] [DATETIME] NOT NULL,
      [SeatsAvailable] [SMALLINT] NOT NULL,
 CONSTRAINT [PK_CovidRestrictions] PRIMARY KEY CLUSTERED
(
      [TableID] ASC,
      [Since] 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].[CovidRestrictions]
WITH CHECK ADD  CONSTRAINT [FK_CovidRestrictions_Tables]
FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[CovidRestrictions]
CHECK CONSTRAINT [FK_CovidRestrictions_Tables]
GO

ALTER TABLE [dbo].[CovidRestrictions]
WITH CHECK ADD  CONSTRAINT [CK_CovidRestrictions]
CHECK  (([SeatsAvailable]>=(0)))
GO

ALTER TABLE [dbo].[CovidRestrictions]
CHECK CONSTRAINT [CK_CovidRestrictions]
GO

ALTER TABLE [dbo].[CovidRestrictions]
WITH CHECK ADD  CONSTRAINT [CK_CovidRestrictions_1]
CHECK  (([Since]<=[Until]))
GO

ALTER TABLE [dbo].[CovidRestrictions]
CHECK CONSTRAINT [CK_CovidRestrictions_1]
GO
```

10. Tabela Tables

Stanowi słownik stolików dostępnych w danej restauracji. Posiada klucz główny *TableID* (int) oraz liczbę miejsc dostępnych przy stoliku *SeatsNumber* (smallint).

Warunki integralności:

- *TableID* jest unikalne
- *SeatsNumber* jest liczbą nieujemną

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

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

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

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

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

11. Tabela TableReservations

Sstanowi rejestr rezerwacji stolików. Posiada klucz główny będący kombinacją identyfikatora stolika *TableID* (int) (klucz obcy do tabeli Tables) oraz identyfikatora rezerwacji *ReservationID* (int) (klucz obcy do tabeli IndividualReservations/CompanyReservations). Zawarte są w niej również: data od kiedy dany stolik jest zarezerwowany *SinceWhenReserved* (datetime), data do kiedy dany stolik jest zarezerwowany *UntilWhenReserved* (datetime) oraz informacja dla ilu osób jest rezerwacja *CustomerNumber*(small).

Warunki integralności:

- kombinacja_*TableID* i *ReservationID* jest unikalna

- *UntilWhenReserved* nie może nastąpić wcześniej niż *SinceWhenReserved*

```
CREATE TABLE [dbo].[TableReservations](
	[TableID] [INT] NOT NULL,
	[ReservationID] [INT] NOT NULL,
	[SinceWhenReserved] [DATETIME] NOT NULL,
	[UntilWhenReserved] [DATETIME] NOT NULL,
	[CustomerNumber] [SMALLINT] NOT NULL,
 CONSTRAINT [PK_TableReservations] PRIMARY KEY CLUSTERED
(
	[TableID] ASC,
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[TableReservations]
CHECK CONSTRAINT [FK_TableReservations_CompanyReservations]
GO

ALTER TABLE [dbo].[TableReservations]
WITH CHECK ADD  CONSTRAINT [FK_TableReservations_IndividualReservations]
FOREIGN KEY([ReservationID])
REFERENCES [dbo].[IndividualReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[TableReservations]
CHECK CONSTRAINT [FK_TableReservations_IndividualReservations]
GO

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

ALTER TABLE [dbo].[TableReservations]
```

```
CHECK CONSTRAINT [FK_TableReservations_Tables]
GO

ALTER TABLE [dbo].[TableReservations]
WITH CHECK ADD  CONSTRAINT [CK_TableReservations]
CHECK  (([SinceWhenReserved]<=[UntilWhenReserved]))
GO

ALTER TABLE [dbo].[TableReservations]
CHECK CONSTRAINT [CK_TableReservations]
GO
```

12. Tabela IndividualReservations

Stanowi rejestr rezerwacji dla klientów indywidualnych. Posiada klucz główny, będący identyfikatorem rezerwacji *ReservationID* (int), a także identyfikator zamówienia *OrderID* (int) (klucz obcy do tabeli OnSiteOrders) i identyfikator danego klienta indywidualnego *CustomerID* (int), będący kluczem obcym do tabeli IndividualCustomers.

Warunki integralności:

- *ReservationID* jest unikalne

```
CREATE TABLE [dbo].[IndividualReservations](
      [ReservationID] [INT] IDENTITY(1,1) NOT NULL,
      [OrderID] [INT] NOT NULL,
      [CustomerID] [INT] NOT NULL,
 CONSTRAINT [PK_IndividualReservations] 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_OnSiteOrders]
FOREIGN KEY([OrderID])
REFERENCES [dbo].[OnSiteOrders] ([OrderID])
GO

ALTER TABLE [dbo].[IndividualReservations]
CHECK CONSTRAINT [FK_IndividualReservations_OnSiteOrders]
GO
```

13. Tabela CompanyReservations

Stanowi rejestr rezerwacji dla klientów firm. Posiada klucz główny, będący identyfikatorem rezerwacji *ReservationID* (int), identyfikator zamówienia *OrderID* int) oraz klucz obcy do tabeli CompanyCustomers - *CompanyID* (int), który jest identyfikatorem klienta firmy składającego daną rezerwację.

Warunki integralności:

- *ReservationID* jest unikalne

```
CREATE TABLE [dbo].[CompanyReservations](
      [ReservationID] [INT] IDENTITY(1,1) NOT NULL,
      [OrderID] [INT] NULL,
      [CompanyID] [INT] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
      [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyReservations]
WITH CHECK ADD  CONSTRAINT [FK_CompanyReservations_Customers]
FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

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

14. Tabela OnSiteOrders

Stanowi rejestr zamówień na miejscu. Posiada klucz główny *OrderID* (int) (numer zamówienia na miejscu) klucz obcy *CustomerID* (int) (identyfikator klienta firmy) do tabeli CompanyCustomers oraz klucz obcy *EmployeeID* (int) (indentyfikator pracownika, który obsługuje dane zamówienie) do tabeli Employees. Zawiera informację o dacie złożenia zamówienia *OrderDate* (datetime), dacie, kiedy zamówienia ma zostać zrealizowane *ExecutionDate* (datetime) oraz statusie danego zamówienia *Status* (char(1)): N - nowe, C - potwierdzone, P - potwierdzone i opłacone, A - anulowane.

Warunki integralności:

- *OrderID* jest unikalne
- *ExecutionDate* nie może nastąpić wcześniej niż *OrderDate*
- *Status* może przyjąć jedną z wartości: {N, C, P, A}

```
CREATE TABLE [dbo].[OnSiteOrders](
      [OrderID] [INT] IDENTITY(1,1) NOT NULL,
      [CustomerID] [INT] NOT NULL,
      [EmployeeID] [INT] NOT NULL,
      [OrderDate] [DATETIME] NOT NULL,
      [ExecutionDate] [DATETIME] NULL,
      [Status] [CHAR](1) NOT NULL,
 CONSTRAINT [PK_OnSiteOrders] 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].[OnSiteOrders]
WITH CHECK ADD  CONSTRAINT [FK_OnSiteOrders_Employees]
FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO

ALTER TABLE [dbo].[OnSiteOrders]
CHECK CONSTRAINT [FK_OnSiteOrders_Employees]
GO

ALTER TABLE [dbo].[OnSiteOrders]
WITH CHECK ADD  CONSTRAINT [CK_OnSiteOrders]
```

```
CHECK  (([Status]='N' OR [Status]='C' OR [Status]='P' OR [Status]='A'))
GO

ALTER TABLE [dbo].[OnSiteOrders]
CHECK CONSTRAINT [CK_OnSiteOrders]
GO

ALTER TABLE [dbo].[OnSiteOrders]
WITH CHECK ADD  CONSTRAINT [CK_OnSiteOrders_1]
CHECK  (([OrderDate]<=[ExecutionDate]))
GO

ALTER TABLE [dbo].[OnSiteOrders]
CHECK CONSTRAINT [CK_OnSiteOrders_1]
GO
```

15. Tabela IndividualCustomers

Reprezentuje rejestr klientów indywidualnych. Posiada klucz obcy do tabeli Customers *CustomerID* (int), nazwisko *LastName* (nvarchar) klienta, jego imię *FirstName* (nvarchar) oraz opcjonalny identyfikator firmy, którą reprezentuje - CompanyID (int), w przypadku, gdy jest to klient indywidualny, ale z ramienia danej firmy. Jest to klucz obcy do tabeli CompanyCustomers.

```
CREATE TABLE [dbo].[IndividualCustomers](
      [CustomerID] [INT] IDENTITY(1,1) NOT NULL,
      [LastName] [NVARCHAR](50) NOT NULL,
      [FirstName] [NVARCHAR](50) NOT NULL,
      [CompanyID] [INT] 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]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualCustomers]
WITH CHECK ADD  CONSTRAINT [FK_IndividualCustomers_CompanyCustomers]
FOREIGN KEY([CompanyID])
REFERENCES [dbo].[CompanyCustomers] ([CompanyID])
GO
```

```
ALTER TABLE [dbo].[IndividualCustomers]
CHECK CONSTRAINT [FK_IndividualCustomers_CompanyCustomers]
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
```

16. Tabela TakeAwayOrders

Stanowi rejestr zamówień na wynos. Posiada klucz główny *OrderID* (int), który jest identyfikatorem zamówienia, klucz obcy *CustomerID* (int) do tabel *CompanyCustomers* i *IndividualCustomers*, który jest identyfikatorem klienta składającego zamówienie, klucz obcy *EmployeeID* (int) do tabeli Employees, który jest identyfikatorem pracownika obsługującego zamówienie. Zawiera informację o dacie złożenia zamówienia *OrderDate* (datetime), dacie, na kiedy ma zostać ono zrealizowane *ExecutionDate* (datetime) oraz status zamówienia *Status* (char): N - nowe, C - potwierdzone, P - potwierdzone i opłacone, A - anulowane.

Warunki integralności:

- *OrderID* jest unikalne
- *ExecutionDate* nie może wystąpić wcześniej niż *OrderDate*
- *Status* może przyjąć jedną z wartości: {N, C, P, A}

```
CREATE TABLE [dbo].[TakeAwayOrders](
      [OrderID] [INT] IDENTITY(1,1) NOT NULL,
      [CustomerID] [INT] NOT NULL,
      [EmployeeID] [INT] NOT NULL,
      [OrderDate] [DATETIME] NOT NULL,
      [ExecutionDate] [DATETIME] NOT NULL,
      [Status] [CHAR](1) NOT NULL,
 CONSTRAINT [PK_TakeAwayOrders] 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].[TakeAwayOrders]
WITH CHECK ADD  CONSTRAINT [FK_TakeAwayOrders_Customers]
FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[TakeAwayOrders]
CHECK CONSTRAINT [FK_TakeAwayOrders_Customers]
GO

ALTER TABLE [dbo].[TakeAwayOrders]
WITH CHECK ADD  CONSTRAINT [FK_TakeAwayOrders_Employees]
FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO

ALTER TABLE [dbo].[TakeAwayOrders]
CHECK CONSTRAINT [FK_TakeAwayOrders_Employees]
GO

ALTER TABLE [dbo].[TakeAwayOrders]
WITH CHECK ADD  CONSTRAINT [CK_TakeAwayOrders]
CHECK  (([Status]='N' OR [Status]='C' OR [Status]='P' OR [Status]='A'))
GO

ALTER TABLE [dbo].[TakeAwayOrders]
CHECK CONSTRAINT [CK_TakeAwayOrders]
GO

ALTER TABLE [dbo].[TakeAwayOrders]
WITH CHECK ADD  CONSTRAINT [CK_TakeAwayOrders_1]
CHECK  (([OrderDate]<=[ExecutionDate]))
GO

ALTER TABLE [dbo].[TakeAwayOrders]
CHECK CONSTRAINT [CK_TakeAwayOrders_1]
GO
```

17. Tabela IndividualDiscountHist

Reprezentuje w bazie danych rejestr przyznawanych klientom indywidualnym rabatów. Posiada klucz główny, będący kombinacją numeru klienta *CustomerID* (klucz obcy do tabeli IndividualCustomers)(int), numer rabatu *DiscountID* (smallint) (klucz obcy do tabeli

PermIndividualDiscounts/TempIndividualDiscounts) oraz daty, w której ten rabat został przyznany *Since* (date). Posiada także informację, do kiedy dany rabat obowiązuje *Till* (date) i kiedy został użyty *UseDate* (date).

Warunki integralności:

- Kombinacja *CustomerID*, *DiscountID* i *Since* jest unikalna
- *Till* nie może wystąpić wcześniej niż *Since*
- *UseDate* nie może wystąpić później niż *Till*

```
CREATE TABLE [dbo].[IndividualDiscountHist](
      [CustomerID] [INT] IDENTITY(1,1) NOT NULL,
      [DiscountID] [SMALLINT] NOT NULL,
      [Since] [DATE] NOT NULL,
      [Till] [DATE] NULL,
      [UseDate] [DATE] NULL,
 CONSTRAINT [PK_IndividualDiscountHist] PRIMARY KEY CLUSTERED
(
      [CustomerID] ASC,
      [DiscountID] ASC,
      [Since] 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].[IndividualDiscountHist]
WITH CHECK ADD  CONSTRAINT
[FK_IndividualDiscountHist_IndividualCustomers]
FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndividualCustomers] ([CustomerID])
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
CHECK CONSTRAINT [FK_IndividualDiscountHist_IndividualCustomers]
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
WITH CHECK ADD  CONSTRAINT
[FK_IndividualDiscountHist_PermIndividualDiscounts]
FOREIGN KEY([DiscountID])
REFERENCES [dbo].[PermIndividualDiscounts] ([DiscountID])
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
CHECK CONSTRAINT [FK_IndividualDiscountHist_PermIndividualDiscounts]
```

```
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
WITH CHECK ADD  CONSTRAINT
[FK_IndividualDiscountHist_TempIndividualDiscounts]
FOREIGN KEY([DiscountID])
REFERENCES [dbo].[TempIndividualDiscounts] ([DiscountID])
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
CHECK CONSTRAINT [FK_IndividualDiscountHist_TempIndividualDiscounts]
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
WITH CHECK ADD  CONSTRAINT [CK_IndividualDiscountHist]
CHECK  (([Since]<=[UseDate] AND [UseDate]<=[Till]))
GO

ALTER TABLE [dbo].[IndividualDiscountHist]
CHECK CONSTRAINT [CK_IndividualDiscountHist]
GO
```

18. Tabela TempIndividualDiscounts

Stanowi słownik rabatów przyznanych na czas określonej liczby dni *Duration* (int). Posiada klucz główny, będący identyfikatorem rabatu *DiscountID* (smallint) oraz informację o wymaganym do jego przyznania warunku, tj. określonej łącznej kwocie zamówień MinTotalPrice (money) i wartość rabatu *Discount* (real).

Warunki integralności:

- *DiscountID* jest unikalne
- *MinTotalPrice* jest liczbą nieujemną
- *Duration* jest liczbą nieujemną
- *Discount* to liczba zmiennoprzecinkowa z zakresu [0,1]

```
CREATE TABLE [dbo].[TempIndividualDiscounts](
      [DiscountID] [SMALLINT] NOT NULL,
      [RestaurantID] [INT] NOT NULL,
      [MinTotalPrice] [MONEY] NOT NULL,
      [Duration] [INT] NOT NULL,
      [Discount] [REAL] NOT NULL,
 CONSTRAINT [PK_TempIndividualDiscounts] 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].[TempIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [FK_TempIndividualDiscounts_Restaurants]
FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
CHECK CONSTRAINT [FK_TempIndividualDiscounts_Restaurants]
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_TempIndividualDiscounts]
CHECK  (([Duration]>=(0)))
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
CHECK CONSTRAINT [CK_TempIndividualDiscounts]
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_TempIndividualDiscounts_1]
CHECK  (([Discount]>=(0) AND [Discount]<=(1)))
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
CHECK CONSTRAINT [CK_TempIndividualDiscounts_1]
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_TempIndividualDiscounts_2]
CHECK  (([MinTotalPrice]>=(0)))
GO

ALTER TABLE [dbo].[TempIndividualDiscounts]
CHECK CONSTRAINT [CK_TempIndividualDiscounts_2]
GO
```

19. Tabela PermIndividualDiscounts

Stanowi słownik rabatów przyznawanych klientom indywidualnym na stałe. Posiada klucz główny *DiscountID* (smallint), który jest identyfikatorem rabatu oraz informacje o warunkach, które klient musi spełnić, by uzyskać rabat: ustaloną liczbę zamówień *OrdersNumber* (int) za minimalną określoną kwotę *MinOrderPrice* (money). Posiada wartość rabatu *Discount* (real).

Warunki integralności:

- *DiscountID* jest unikalne
- *OrdersNumber* jest liczbą nieujemną
- *MinOrdersPrice* jest liczbą nieujemną
- *Discount* to liczba zmiennoprzecinkowa z zakresu [0,1]

```
CREATE TABLE [dbo].[PermIndividualDiscounts](
      [DiscountID] [SMALLINT] IDENTITY(1,1) NOT NULL,
      [RestaurantID] [INT] NOT NULL,
      [OrdersNumber] [INT] NOT NULL,
      [MinOrderPrice] [MONEY] NOT NULL,
      [Discount] [REAL] NOT NULL,
 CONSTRAINT [PK_PermIndividualDiscounts] 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].[PermIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [FK_PermIndividualDiscounts_Restaurants]
FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
CHECK CONSTRAINT [FK_PermIndividualDiscounts_Restaurants]
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_PermIndividualDiscounts]
CHECK  (([OrdersNumber]>=(0)))
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
CHECK CONSTRAINT [CK_PermIndividualDiscounts]
GO
```

```
ALTER TABLE [dbo].[PermIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_PermIndividualDiscounts_1]
CHECK  (([Discount]>=(0) AND [Discount]<=(1)))
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
CHECK CONSTRAINT [CK_PermIndividualDiscounts_1]
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
WITH CHECK ADD  CONSTRAINT [CK_PermIndividualDiscounts_2]
CHECK  (([MinOrderPrice]>=(0)))
GO

ALTER TABLE [dbo].[PermIndividualDiscounts]
CHECK CONSTRAINT [CK_PermIndividualDiscounts_2]
GO
```

20. Tabela OrderDetails

Reprezentuje w bazie szczegółowe informacje o danym zamówieniu. Posiada klucz obcy *OrderID* (int) do tabel *TakeAwayOrders* i *OnSiteOrders*, który jest identyfikatorem zamówienia, klucz obcy *DishID* (int) do tabeli Menu, który jest identyfikatorem dania z menu. Kluczem głównym w tej tabeli jest kombinacja *OrderID* i *DishID*. Posiada również informację o liczbie zamówionych dań *Quantity* (smallint).

Warunki integralności:

- Kombinacja *OrderID* i *DishID* jest unikalna
- *Quantity* jest liczbą nieujemną

```
CREATE TABLE [dbo].[OrderDetails](
      [OrderID] [INT] NOT NULL,
      [DishID] [INT] NOT NULL,
      [Quantity] [SMALLINT] NOT NULL,
      [UnitPrice] [MONEY] 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_OrderDetails_Menu1]
FOREIGN KEY([DishID])
REFERENCES [dbo].[Menu] ([DishID])
GO

ALTER TABLE [dbo].[OrderDetails]
CHECK CONSTRAINT [FK_OrderDetails_Menu1]
GO

ALTER TABLE [dbo].[OrderDetails]
WITH CHECK ADD  CONSTRAINT [FK_OrderDetails_OnSiteOrders1]
FOREIGN KEY([OrderID])
REFERENCES [dbo].[OnSiteOrders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderDetails]
CHECK CONSTRAINT [FK_OrderDetails_OnSiteOrders1]
GO

ALTER TABLE [dbo].[OrderDetails]
WITH CHECK ADD  CONSTRAINT [FK_OrderDetails_TakeAwayOrders1]
FOREIGN KEY([OrderID])
REFERENCES [dbo].[TakeAwayOrders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderDetails]
CHECK CONSTRAINT [FK_OrderDetails_TakeAwayOrders1]
GO

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

ALTER TABLE [dbo].[OrderDetails]
CHECK CONSTRAINT [CK_OrderDetails]
GO
```

21. Tabela Cities

Reprezentuje miasta. Posiada klucz główny *CityID* (int) będący identyfikatorem miasta, a także jego nazwę *CityName* (nvarchar). Zawiera klucz obcy *CountryID* (int) do tabeli Countries, który jest identyfikatorem kraju, w którym dane miasto się znajduje.

Warunki integralności:

- *CityID* jest unikalne

```
CREATE TABLE [dbo].[Cities](
	[CityID] [INT] IDENTITY(1,1) NOT NULL,
	[CityName] [NVARCHAR](50) NOT NULL,
	[CountryID] [INT] NOT NULL,
 CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
	[CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

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

22. Tabela Countries

Reprezentuje kraje. Posiada klucz główny *CountryID* (int), będący identyfikatorem kraju. Posiada też nazwę kraju *CountryName* (nvarchar).

Warunki integralności:

- *CountryID* jest unikalne
- *CountryName* jest unikalne

```
CREATE TABLE [dbo].[Countries](
	[CountryID] [INT] IDENTITY(1,1) NOT NULL,
	[CountryName] [NVARCHAR](50) NOT NULL,
 CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED
(
	[CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [CountryNameUnique] UNIQUE NONCLUSTERED
(
```

```
	[CountryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

23. Tabela Menu

Stanowi słownik wszystkich dań, reprezentuje kartę menu. Posiada klucz główny *DishID* (int), który jest identyfikatorem dania, nazwę dania *DishName* (nvarchar), kategorię, do której danie należy *CategoryID* (np. owoce morza)(int) oraz cenę *Price* (money). Kluczem obcym do tabeli Categories jest *CategoryID*.

Warunki integralności:

- *DishID* jest unikalne
- *DishName* jest unikalne

```
CREATE TABLE [dbo].[Menu](
	[DishID] [INT] IDENTITY(1,1) NOT NULL,
	[DishName] [NVARCHAR](50) NOT NULL,
	[RestaurantID] [INT] NOT NULL,
	[CategoryID] [INT] NOT NULL,
	[Price] [MONEY] NOT NULL,
 CONSTRAINT [PK_Menu] 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_Categories]
FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO

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

ALTER TABLE [dbo].[Menu]
WITH CHECK ADD  CONSTRAINT [FK_Menu_Restaurants]
```

```
FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[Menu]
CHECK CONSTRAINT [FK_Menu_Restaurants]
GO
```

24. Tabela Categories

Stanowi słownik kategorii dań. Posiada klucz główny *CategoryID* (int), będący identyfikatorem kategorii, a także jej nazwę *CategoryName* (nvarchar) i opis *Description* (ntext).

Warunki integralności:

- *CategoryID* jest unikalne
- *CategoryName* jest unikalne

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

25. Tabela MenuRegister

Stanowi rejestr pozycji (z tabeli Menu) wprowadzanych i usuwanych z karty na przestrzeni czasu. Posiada klucz główny będący kombinacją *DishID* (int) - identyfikatora dania oraz *MenuIn* (date) - daty, kiedy danie zostało wprowadzone do karty. Posiada również datę, kiedy pozycja została zdjęta z menu MenuOut (jeśli w ogóle)(date). Posiada klucz obcy DishID do tabeli Menu.

Warunki integralności:

- Kombinacja *DishID* i *MenuIn* jest unikalna
- *MenuOut* nie może wystąpić wcześniej niż *MenuIn*

```
CREATE TABLE [dbo].[MenuRegister](
      [DishID] [INT] IDENTITY(1,1) NOT NULL,
      [MenuIn] [DATE] NOT NULL,
      [MenuOut] [DATE] NULL,
 CONSTRAINT [PK_MenuRegister] PRIMARY KEY CLUSTERED
(
      [DishID] ASC,
      [MenuIn] 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].[MenuRegister]
WITH CHECK ADD  CONSTRAINT [FK_MenuRegister_Menu]
FOREIGN KEY([DishID])
REFERENCES [dbo].[Menu] ([DishID])
GO

ALTER TABLE [dbo].[MenuRegister]
CHECK CONSTRAINT [FK_MenuRegister_Menu]
GO

ALTER TABLE [dbo].[MenuRegister]
WITH CHECK ADD  CONSTRAINT [CK_MenuRegister]
CHECK  (([MenuIn]<=[MenuOut]))
GO

ALTER TABLE [dbo].[MenuRegister]
CHECK CONSTRAINT [CK_MenuRegister]
GO
```

26. Tabela DishDetails

Zawiera szczegóły dotyczące składników dań z menu. Posiada klucz główny, będący kombinacją *DishID* (int) - identyfikatora dania oraz *ProductID* (int) - identyfikatora produktu, który jest składnikiem dania. Zawiera także informację o liczbie potrzebnych do danej potrawy jednostek półproduktów - *Quantity* (int). Posiada klucz obcy DishID do tabeli Menu oraz ProductID do tabeli Products.

Warunki integralności:

- Kombinacja *DishID* i *ProductID* jest unikalna
- *Quantity* jest liczbą nieujemną

```
CREATE TABLE [dbo].[DishDetails](
      [DishID] [INT] NOT NULL,
      [ProductID] [INT] NOT NULL,
      [Quantity] [INT] NOT NULL,
 CONSTRAINT [PK_DishDetails] PRIMARY KEY CLUSTERED
(
      [DishID] ASC,
      [ProductID] 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].[DishDetails]
WITH CHECK ADD  CONSTRAINT [FK_DishDetails_Menu]
FOREIGN KEY([DishID])
REFERENCES [dbo].[Menu] ([DishID])
GO

ALTER TABLE [dbo].[DishDetails]
CHECK CONSTRAINT [FK_DishDetails_Menu]
GO

ALTER TABLE [dbo].[DishDetails]
WITH CHECK ADD  CONSTRAINT [FK_DishDetails_Products1]
FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[DishDetails]
CHECK CONSTRAINT [FK_DishDetails_Products1]
GO

ALTER TABLE [dbo].[DishDetails]
WITH CHECK ADD  CONSTRAINT [CK_DishDetails]
CHECK  (([Quantity]>=(0)))
GO

ALTER TABLE [dbo].[DishDetails]
CHECK CONSTRAINT [CK_DishDetails]
GO
```

27. Tabela Products

Stanowi słownik wszystkich półproduktów, potrzebnych do przyrządzenia dań. Posiada klucz główny *ProductID* (int), będący identyfikatorem produktu, a także jego nazwę *ProductName* (nvarchar), dostępną ilość jednostek półproduktu na stanie - *UnitsInStock* (int) i informację o dostępności - *Discontinued* (bit).

Warunki integralności:

- *ProductID* jest unikalne
- *ProductName* jest unikalne
- *UnitsInStock* jest liczbą nieujemną

```
CREATE TABLE [dbo].[Products](
	[ProductID] [INT] NOT NULL,
	[RestaurantID] [INT] NOT NULL,
	[ProductName] [NVARCHAR](50) NOT NULL,
	[UnitsInStock] [INT] NOT NULL,
	[Discontinued] [BIT] NOT NULL,
 CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
	[ProductID] 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 [ProductNameUnique] UNIQUE NONCLUSTERED
(
	[ProductName] 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].[Products]
WITH CHECK ADD  CONSTRAINT [FK_Products_Restaurants]
FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[Products]
CHECK CONSTRAINT [FK_Products_Restaurants]
GO

ALTER TABLE [dbo].[Products]
```

```
WITH CHECK ADD  CONSTRAINT [CK_Products]
CHECK  (([UnitsInStock]>=(0)))
GO

ALTER TABLE [dbo].[Products]
CHECK CONSTRAINT [CK_Products]
GO
```

3. Widoki

1. **TableReservationsView**

   Przedstawiający rezerwacje stolików

```
CREATE VIEW [dbo].[TableReservationsView]
AS
SELECT   TableID, SinceWhenReserved
FROM         dbo.TableReservations
GROUP BY TableID, SinceWhenReserved
GO
```

2. **CompanyDiscountsView**

   Przedstawiający rabaty przyznane klientom firmowym

```
CREATE VIEW [dbo].[CompanyDiscountsView]
AS
SELECT dbo.CompanyCustomers.CompanyID, dbo.CompanyCustomers.CompanyName,
dbo.CompanyDiscountHist.DiscountID, dbo.CompanyDiscountHist.Since
FROM   dbo.CompanyCustomers INNER JOIN
          dbo.CompanyDiscountHist ON dbo.CompanyCustomers.CompanyID =
dbo.CompanyDiscountHist.CustomerID INNER JOIN
          dbo.CompanyDiscounts ON dbo.CompanyDiscountHist.DiscountID =
dbo.CompanyDiscounts.DiscountID
GROUP BY dbo.CompanyCustomers.CompanyID,
dbo.CompanyCustomers.CompanyName, dbo.CompanyDiscountHist.DiscountID,
dbo.CompanyDiscountHist.Since
GO
```

3. IndividualDiscountsView

Przedstawiający rabaty przyznane klientom indywidualnym

```
CREATE VIEW [dbo].[IndividualDiscountsView]
AS
SELECT dbo.IndividualCustomers.CustomerID,
dbo.IndividualDiscountHist.DiscountID,
dbo.PermIndividualDiscounts.Discount,
dbo.TempIndividualDiscounts.Discount AS Expr1
FROM   dbo.IndividualCustomers INNER JOIN
          dbo.IndividualDiscountHist ON
dbo.IndividualCustomers.CustomerID =
dbo.IndividualDiscountHist.CustomerID INNER JOIN
          dbo.PermIndividualDiscounts ON
dbo.IndividualDiscountHist.DiscountID =
dbo.PermIndividualDiscounts.DiscountID INNER JOIN
          dbo.TempIndividualDiscounts ON
dbo.IndividualDiscountHist.DiscountID =
dbo.TempIndividualDiscounts.DiscountID
GROUP BY dbo.IndividualCustomers.CustomerID,
dbo.IndividualDiscountHist.DiscountID,
dbo.PermIndividualDiscounts.Discount,
dbo.TempIndividualDiscounts.Discount
GO
```

4. MenuView

Przedstawiający, jak zmieniało się Menu na przestrzeni dni

```
CREATE VIEW [dbo].[MenuView]
AS
SELECT dbo.Menu.RestaurantID, dbo.MenuRegister.DishID,
dbo.Menu.DishName, dbo.MenuRegister.MenuIn, dbo.MenuRegister.MenuOut,
dbo.Menu.Price
FROM   dbo.Menu INNER JOIN
          dbo.MenuRegister ON dbo.Menu.DishID = dbo.MenuRegister.DishID
GROUP BY dbo.Menu.RestaurantID, dbo.MenuRegister.DishID,
dbo.Menu.DishName, dbo.MenuRegister.MenuIn, dbo.MenuRegister.MenuOut,
dbo.Menu.Price
GO
```

5. IndividualOnSiteOrdersMenuView

Statystyki dla klientów indywidualnych dot. kwot zamówień na miejscu

```
CREATE VIEW [dbo].[IndividualOnSiteOrdersMoneyView]
AS
SELECT dbo.IndividualCustomers.CustomerID,
dbo.IndividualReservations.OrderID, dbo.OrderDetails.Quantity *
dbo.OrderDetails.UnitPrice AS OrderPrice
FROM   dbo.IndividualReservations INNER JOIN
          dbo.OnSiteOrders ON dbo.IndividualReservations.OrderID =
dbo.OnSiteOrders.OrderID INNER JOIN
          dbo.OrderDetails ON dbo.OnSiteOrders.OrderID =
dbo.OrderDetails.OrderID INNER JOIN
          dbo.IndividualCustomers ON
dbo.IndividualReservations.CustomerID =
dbo.IndividualCustomers.CustomerID
GROUP BY dbo.IndividualCustomers.CustomerID,
dbo.IndividualReservations.OrderID, dbo.OrderDetails.Quantity *
dbo.OrderDetails.UnitPrice
GO
```

6. IndividualTakeAwayOrdersMoneyView

Przedstawiający statystyki dla klientów indywidualnych dot. kwot zamówień na wynos

```
CREATE VIEW [dbo].[IndividualTakeAwayOrdersMoneyView]
AS
SELECT dbo.IndividualCustomers.CustomerID, dbo.TakeAwayOrders.OrderID,
dbo.OrderDetails.UnitPrice * dbo.OrderDetails.Quantity AS OrderPrice
FROM   dbo.OrderDetails INNER JOIN
          dbo.TakeAwayOrders ON dbo.OrderDetails.OrderID =
dbo.TakeAwayOrders.OrderID INNER JOIN
          dbo.IndividualCustomers ON dbo.TakeAwayOrders.CustomerID =
dbo.IndividualCustomers.CustomerID
GROUP BY dbo.IndividualCustomers.CustomerID, dbo.TakeAwayOrders.OrderID,
dbo.OrderDetails.UnitPrice * dbo.OrderDetails.Quantity
GO
```

7. IndividualOnSiteOrdersDatesView

Przedstawiający statystyki dla klientów indywidualnych dot. czasu składania zamówień na miejscu

```
CREATE VIEW [dbo].[IndividualOnSiteOrdersDatesView]
AS
SELECT dbo.IndividualCustomers.CustomerID, dbo.OnSiteOrders.OrderID,
dbo.OnSiteOrders.OrderDate
FROM   dbo.IndividualCustomers INNER JOIN
          dbo.OnSiteOrders ON dbo.IndividualCustomers.CustomerID =
dbo.OnSiteOrders.CustomerID
GROUP BY dbo.IndividualCustomers.CustomerID, dbo.OnSiteOrders.OrderID,
dbo.OnSiteOrders.OrderDate
GO
```

8. IndividualTakeAwayOrdersDatesView

Przedstawiający statystyki dla klientów indywidualnych dot. czasu składania zamówień na wynos

```
CREATE VIEW [dbo].[IndividualTakeAwayOrdersDatesView]
AS
SELECT dbo.IndividualCustomers.CustomerID, dbo.TakeAwayOrders.OrderID,
dbo.TakeAwayOrders.OrderDate
FROM   dbo.IndividualCustomers INNER JOIN
          dbo.TakeAwayOrders ON dbo.IndividualCustomers.CustomerID =
dbo.TakeAwayOrders.CustomerID
GROUP BY dbo.IndividualCustomers.CustomerID, dbo.TakeAwayOrders.OrderID,
dbo.TakeAwayOrders.OrderDate
GO
```

9. CompanyOnSiteOrdersMoneyView

Statystyki dla firm dot. kwot zamówień na miejscu

```
CREATE VIEW [dbo].[CompanyOnSiteOrdersMoneyView]
AS
SELECT dbo.CompanyCustomers.CompanyID, dbo.OnSiteOrders.OrderID,
dbo.OrderDetails.UnitPrice * dbo.OrderDetails.Quantity AS OrderPrice
FROM   dbo.CompanyCustomers INNER JOIN
          dbo.OnSiteOrders ON dbo.CompanyCustomers.CompanyID =
dbo.OnSiteOrders.CustomerID INNER JOIN
          dbo.OrderDetails ON dbo.OnSiteOrders.OrderID =
dbo.OrderDetails.OrderID
GROUP BY dbo.CompanyCustomers.CompanyID, dbo.OnSiteOrders.OrderID,
```

```
dbo.OrderDetails.UnitPrice * dbo.OrderDetails.Quantity
GO
```

10. CompanyTakeAwayOrdersMoneyView

Statystyki dla firm dot. kwot zamówień na wynos

```
CREATE VIEW [dbo].[CompanyTakeAwayOrdersMoneyView]
AS
SELECT   dbo.CompanyCustomers.CompanyID, dbo.TakeAwayOrders.OrderID,
dbo.OrderDetails.Quantity * dbo.OrderDetails.UnitPrice AS OrderPrice
FROM         dbo.CompanyCustomers INNER JOIN
                      dbo.TakeAwayOrders ON
dbo.CompanyCustomers.CompanyID = dbo.TakeAwayOrders.CustomerID INNER
JOIN
                      dbo.OrderDetails ON dbo.TakeAwayOrders.OrderID
= dbo.OrderDetails.OrderID
GROUP BY dbo.CompanyCustomers.CompanyID, dbo.TakeAwayOrders.OrderID,
dbo.OrderDetails.Quantity * dbo.OrderDetails.UnitPrice
GO
```

11. CompanyOnSiteOrdersDatesView

Przedstawiający statystyki dot. czasu składania zamówień na miejscu

```
CREATE VIEW [dbo].[CompanyOnSiteOrdersDatesView]
AS
SELECT dbo.CompanyCustomers.CompanyID, dbo.OnSiteOrders.OrderID,
dbo.OnSiteOrders.OrderDate
FROM   dbo.CompanyCustomers INNER JOIN
          dbo.OnSiteOrders ON dbo.CompanyCustomers.CompanyID =
dbo.OnSiteOrders.CustomerID
GROUP BY dbo.CompanyCustomers.CompanyID, dbo.OnSiteOrders.OrderID,
dbo.OnSiteOrders.OrderDate
GO
```

12. CompanyTakeAwayOrdersDatesView

Przedstawiający statystyki dla firm dot. czasu składania zamówień na wynos

```
CREATE VIEW [dbo].[CompanyTakeAwayOrdersDatesView]
AS
```

```
SELECT dbo.CompanyCustomers.CompanyID, dbo.TakeAwayOrders.OrderID,
dbo.TakeAwayOrders.OrderDate
FROM   dbo.CompanyCustomers INNER JOIN
          dbo.TakeAwayOrders ON dbo.CompanyCustomers.CompanyID =
dbo.TakeAwayOrders.CustomerID
GROUP BY dbo.CompanyCustomers.CompanyID, dbo.TakeAwayOrders.OrderID,
dbo.TakeAwayOrders.OrderDate
GO
```

4. Funkcje zwracające tabele (widoki parametryzowane)

1. GenerateInvoice

Generuje fakturę za zamówienie dla danego klienta - firmy.

```
CREATE FUNCTION [dbo].[GenerateInvoice] (@orderID INT, @companyID INT,
@isTakeAway BIT)
RETURNS @invoice TABLE
(
    param_name VARCHAR(50),
    param_val MONEY
)
AS
BEGIN
    DECLARE @restaurantID INT
    SET @restaurantID = (SELECT RestaurantID FROM dbo.Customers WHERE
CustomerID = @companyID)
    DECLARE @restaurantName VARCHAR(50)
    SET @restaurantName = (SELECT RestaurantName FROM dbo.Restaurants
WHERE RestaurantID = @restaurantID)
    INSERT @invoice
    (
      param_name,
      param_val
    )
    VALUES
    (   CONCAT('Restaurant Name: ', @restaurantName),  -- param_name -
varchar(50)
      NULL -- param_val - money
      )
```

```
    DECLARE @companyName VARCHAR(250)
    SET @companyName = (SELECT CompanyName FROM dbo.CompanyCustomers
WHERE CompanyID = @companyID)
    INSERT @invoice
    (
      param_name,
      param_val
    )
    VALUES
    (   CONCAT('Company Name: ', @companyName),  -- param_name -
varchar(50)
      NULL -- param_val - money
      )
    DECLARE @NIP VARCHAR(50)
    SET @NIP = (SELECT NIP FROM dbo.CompanyCustomers WHERE CompanyID =
@companyID)
    INSERT @invoice
    (
      param_name,
      param_val
    )
    VALUES
    (   CONCAT('NIP: ', @NIP),  -- param_name - varchar(50)
      NULL -- param_val - money
      )
    DECLARE @address VARCHAR(50)
    SET @address = (SELECT Address FROM Customers WHERE CustomerID =
@companyID)
    INSERT @invoice
    (
      param_name,
      param_val
    )
    VALUES
    (   CONCAT('Company Address: ', @address),  -- param_name -
varchar(50)
      NULL -- param_val - money
      )
    DECLARE @email VARCHAR(50)
    SET @email = (SELECT Email FROM dbo.Customers WHERE CustomerID =
@companyID)
    INSERT @invoice
    (
      param_name,
      param_val
    )
```

```
    VALUES
    (   CONCAT('Company email: ', @email),  -- param_name - varchar(50)
      NULL -- param_val - money
      )
    IF (@isTakeAway = 0)
    BEGIN
       DECLARE @orderDate DATETIME
       SET @orderDate = (SELECT OrderDate FROM dbo.OnSiteOrders WHERE
OrderID = @orderID)
       DECLARE @executionDate DATETIME
       SET @executionDate = (SELECT ExecutionDate FROM dbo.OnSiteOrders
WHERE OrderID = @orderID)
       INSERT @invoice
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Order date: ', @orderDate),  -- param_name -
varchar(50)
            NULL -- param_val - money
            )
       INSERT @invoice
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Execution date: ',@executionDate),  -- param_name -
varchar(50)
            NULL -- param_val - money
            )
    END
    ELSE
    BEGIN
       SET @orderDate = (SELECT OrderDate FROM dbo.TakeAwayOrders WHERE
OrderID = @orderID)
       SET @executionDate = (SELECT ExecutionDate FROM
dbo.TakeAwayOrders WHERE OrderID = @orderID)
       INSERT @invoice
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Order date: ', @orderDate),  -- param_name -
```

```
varchar(50)
            NULL -- param_val - money
            )
       INSERT @invoice
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Execution date: ',@executionDate),  -- param_name -
varchar(50)
            NULL -- param_val - money
            )
    END
    DECLARE @dishID INT
    DECLARE @quantity INT
    DECLARE @unitPrice MONEY
    DECLARE @dishName VARCHAR(50)
    DECLARE CUR CURSOR FOR
    SELECT DishID, Quantity, UnitPrice FROM dbo.OrderDetails
    WHERE OrderID = @orderID
    GROUP BY DishID, Quantity, UnitPrice
    OPEN CUR
    FETCH NEXT FROM CUR INTO @dishID, @quantity, @unitPrice
    WHILE @@FETCH_STATUS = 0
    BEGIN
       SET @dishName = (SELECT DishName FROM Menu WHERE DishID =
@dishID)
       INSERT @invoice
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Dish: ', @dishName, ' Quantity: ', @quantity),  --
param_name - varchar(50)
            (@quantity * @unitPrice) -- param_val - money
            )
       FETCH NEXT FROM CUR INTO @dishID, @quantity, @unitPrice
    END
    CLOSE CUR
    DEALLOCATE CUR
    DECLARE @totalOrderValue MONEY
    SET @totalOrderValue = (SELECT dbo.GetOrderValue(@orderID))
    INSERT @invoice
    (
```

```
      param_name,
      param_val
    )
    VALUES
    (   'Total order value: ',  -- param_name - varchar(50)
      @totalOrderValue -- param_val - money
      )
    RETURN
END
GO
```

2. OrderSummary

Generuje podsumowanie danego zamówienia dla danego klienta. Zwraca informację m.in. o danych osobowych klienta, dacie złożenia i realizacji zamówienia oraz typowe informacje zawarte w paragonie: liczba zamówionych sztuk danego dania oraz ich ceny, a także całkowita wartość zamówienia.

```
CREATE FUNCTION [dbo].[OrderSummary](@orderID INT, @customerID INT,
@isTakeAway BIT)
RETURNS @summary TABLE
(
    param_name VARCHAR(250),
    param_val VARCHAR(250)
)
AS
BEGIN
    DECLARE @restaurantID INT
    SET @restaurantID = (SELECT RestaurantID FROM dbo.Customers WHERE
CustomerID = @customerID)
    DECLARE @restaurantName VARCHAR(50)
    SET @restaurantName = (SELECT RestaurantName FROM dbo.Restaurants
WHERE RestaurantID = @restaurantID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Restaurant Name: ',
      @restaurantName
      )
    DECLARE @customerLastname VARCHAR(50)
    SET @customerLastname  = (SELECT LastName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
    INSERT @summary
```

```
    (
      param_name,
      param_val
    )
    VALUES
    (   'Lastname: ',
      @customerLastname
      )
    DECLARE @customerFirstName VARCHAR(50)
    SET @customerFirstName = (SELECT FirstName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Firstname: ',
      @customerFirstName
      )
    DECLARE @address VARCHAR(50)
    SET @address = (SELECT Address FROM Customers WHERE CustomerID =
@customerID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Address:',
      @address
      )
    DECLARE @email VARCHAR(50)
    SET @email = (SELECT Email FROM dbo.Customers WHERE CustomerID =
@customerID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Email: ',
      @email
      )

    IF (@isTakeAway = 0)
```

```
    BEGIN
       DECLARE @orderDate DATETIME
       SET @orderDate = (SELECT OrderDate FROM dbo.OnSiteOrders WHERE
OrderID = @orderID)
       DECLARE @executionDate DATETIME
       SET @executionDate = (SELECT ExecutionDate FROM dbo.OnSiteOrders
WHERE OrderID = @orderID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (    'Order date: ',
            @orderDate
            )
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (    'Execution date: ',
            @executionDate
            )
    END
    ELSE
    BEGIN
       SET @orderDate = (SELECT OrderDate FROM dbo.TakeAwayOrders WHERE
OrderID = @orderID)
       SET @executionDate = (SELECT ExecutionDate FROM
dbo.TakeAwayOrders WHERE OrderID = @orderID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (    'Order date: ',
            @orderDate
            )
       INSERT @summary
       (
            param_name,
            param_val
       )
```

```
       VALUES
       (   'Execution date: ',
            @executionDate
            )
    END
    DECLARE @dishID INT
    DECLARE @quantity INT
    DECLARE @unitPrice MONEY
    DECLARE @dishName VARCHAR(50)
    DECLARE CUR CURSOR FOR
    SELECT DishID, Quantity, UnitPrice FROM dbo.OrderDetails
    WHERE OrderID = @orderID
    GROUP BY DishID, Quantity, UnitPrice
    OPEN CUR
    FETCH NEXT FROM CUR INTO @dishID, @quantity, @unitPrice
    WHILE @@FETCH_STATUS = 0
    BEGIN
       SET @dishName = (SELECT DishName FROM Menu WHERE DishID =
@dishID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (   CONCAT('Dish: ', @dishName, ' x Quantity: ', @quantity),
            CONCAT(@quantity,' x ', @unitPrice,' zł')
            )
       FETCH NEXT FROM CUR INTO @dishID, @quantity, @unitPrice
    END
    CLOSE CUR
    DEALLOCATE CUR
    DECLARE @totalOrderValue MONEY
    SET @totalOrderValue = (SELECT dbo.GetOrderValue(@orderID))
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Total order value: ',
      CONCAT(@totalOrderValue,' zł')
      )
    RETURN
END
GO
```

3. IndividualReservationSummary

Zwraca potwierdzenie rezerwacji indywidualnej ze wskazaniem numeru stolika. Zawiera pełne dane o danej rezerwacji.

```
CREATE FUNCTION [dbo].[IndividualReservationSummary](@reservationID INT,
@customerID INT, @reserevedFor DATETIME)
RETURNS @summary TABLE
(
    param_name VARCHAR(250),
    param_val VARCHAR(250)
)
AS
BEGIN
    DECLARE @restaurantID INT
    SET @restaurantID = (SELECT RestaurantID FROM dbo.TableReservations
INNER JOIN dbo.Tables
    ON Tables.TableID = TableReservations.TableID
    WHERE ReservationID = @reservationID AND SinceWhenReserved =
@reserevedFor)
    DECLARE @restaurantName VARCHAR(50)
    SET @restaurantName = (SELECT RestaurantName FROM dbo.Restaurants
WHERE RestaurantID = @restaurantID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Restaurant Name: ',
      @restaurantName
      )
    DECLARE @customerLastname VARCHAR(50)
    SET @customerLastname  = (SELECT LastName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Lastname: ',
      @customerLastname
      )

    DECLARE @customerFirstName VARCHAR(50)
    SET @customerFirstName = (SELECT FirstName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
```

```
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Firstname: ',
      @customerFirstName
      )
    DECLARE @email VARCHAR(50)
    SET @email = (SELECT Email FROM dbo.Customers WHERE CustomerID =
@customerID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Email: ',
      @email
      )
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Reservation number: ',
      @reservationID
      )
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Reservation status: ',
      'Confirmed'
      )

    DECLARE @till DATETIME
    SET @till = (SELECT UntilWhenReserved FROM dbo.TableReservations
WHERE ReservationID = @reservationID AND SinceWhenReserved =
@reserevedFor)
    INSERT @summary
    (
```

```
      param_name,
      param_val
    )
    VALUES
    (   'Reservation time: ',
      CONCAT(@reserevedFor,' - ',@till)
      )
    DECLARE @tableID INT
    SET @tableID = (SELECT TableID FROM dbo.TableReservations WHERE
ReservationID = @reservationID AND SinceWhenReserved = @reserevedFor)
    DECLARE @guestsNumber INT
    SET @guestsNumber = (SELECT CustomerNumber FROM
dbo.TableReservations WHERE ReservationID = @reservationID AND
SinceWhenReserved = @reserevedFor)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Table number: ',
      @tableID
      )
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Maximum guest number: ',
      @guestsNumber
      )
    RETURN
END
GO
```

4. CompanyReservationSummary

Zwraca potwierdzenie rezerwacji firmowej (imiennej lub zwykłej) ze wskazaniem numeru stolika. Zawiera pełne dane o danej rezerwacji.

```
CREATE FUNCTION [dbo].[CompanyReservationSummary](@reservationID INT,
@customerID INT, @reserevedFor DATETIME)
RETURNS @summary TABLE
(
    param_name VARCHAR(250),
    param_val VARCHAR(250)
```

```
)
AS
BEGIN
    DECLARE @restaurantID INT
    SET @restaurantID = (SELECT RestaurantID FROM dbo.TableReservations
INNER JOIN dbo.Tables
    ON Tables.TableID = TableReservations.TableID
    WHERE ReservationID = @reservationID AND SinceWhenReserved =
@reserevedFor)
    DECLARE @restaurantName VARCHAR(50)
    SET @restaurantName = (SELECT RestaurantName FROM dbo.Restaurants
WHERE RestaurantID = @restaurantID)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Restaurant Name: ',
      @restaurantName
      )
    IF (dbo.IsCompanyCustomer(@customerID) = 1)
    BEGIN
       DECLARE @companyName VARCHAR(50)
       SET @companyName  = (SELECT CompanyName FROM dbo.CompanyCustomers
WHERE CompanyID = @customerID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (   'Company name: ',
            @companyName
            )
       DECLARE @NIP VARCHAR(50)
       SET @NIP  = (SELECT NIP FROM dbo.CompanyCustomers WHERE CompanyID
= @customerID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (   'NIP: ',
            @NIP
```

```
            )
    END
    ELSE
    BEGIN
       DECLARE @customerLastName VARCHAR(50)
       SET @customerLastName = (SELECT LastName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
       INSERT @summary
       (
            param_name,
            param_val
       )
       VALUES
       (   'Lastname: ',
            @customerLastName
            )
       DECLARE @customerFirstName VARCHAR(50)
       SET @customerFirstName = (SELECT FirstName FROM
dbo.IndividualCustomers WHERE CustomerID = @customerID)
       INSERT @summary
       (
             param_name,
             param_val
       )
       VALUES
       (   'Firstname: ',
             @customerFirstName
             )
    END
    DECLARE @email VARCHAR(50)
    SET @email = (SELECT Email FROM dbo.Customers WHERE CustomerID =
@customerID)
       INSERT @summary
       (
             param_name,
             param_val
       )
    VALUES
    (   'Email: ',
      @email
      )
    INSERT @summary
    (
      param_name,
      param_val
    )
```

```
    VALUES
    (   'Reservation number: ',
      @reservationID
      )
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Reservation status: ',
      'Confirmed'
      )

    DECLARE @till DATETIME
    SET @till = (SELECT UntilWhenReserved FROM dbo.TableReservations
WHERE ReservationID = @reservationID AND SinceWhenReserved =
@reserevedFor)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Reservation time: ',
      CONCAT(@reserevedFor,' - ',@till)
      )
    DECLARE @tableID INT
    SET @tableID = (SELECT TableID FROM dbo.TableReservations WHERE
ReservationID = @reservationID AND SinceWhenReserved = @reserevedFor)
    DECLARE @guestsNumber INT
    SET @guestsNumber = (SELECT CustomerNumber FROM
dbo.TableReservations WHERE ReservationID = @reservationID AND
SinceWhenReserved = @reserevedFor)
    INSERT @summary
    (
      param_name,
      param_val
    )
    VALUES
    (   'Table number: ',
      @tableID
      )
    INSERT @summary
    (
      param_name,
```

```
      param_val
    )
    VALUES
    (   'Maximum guest number: ',
      @guestsNumber
      )
    RETURN
END
GO
```

5. GetCompanyDiscounts

   Zwracająca przyznane zniżki dla danej firmy

```
CREATE FUNCTION [dbo].[GetCompanyDiscounts](@companyID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.CompanyDiscountsView
                  WHERE CompanyID = @companyID)
GO
```

6. GetIndividualDiscounts

   Zwracająca przyznane zniżki dla danego klienta indywidualnego

```
CREATE FUNCTION [dbo].[GetIndividualDiscounts] (@customerID INT)
      RETURNS TABLE
AS
      RETURN(SELECT * FROM dbo.IndividualDiscountsView
      WHERE CustomerID = @customerID)
GO
```

7. GetCompanyOrderDates

   Zwracająca daty zamówień dla danej firmy

```
CREATE FUNCTION [dbo].[GetCompanyOrdersDates] (@companyID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.CompanyOnSiteOrdersDatesView
      WHERE CompanyID = @companyID
      UNION
```

```
      SELECT * FROM dbo.CompanyTakeAwayOrdersDatesView
      WHERE CompanyID = @companyID)

GO
```

8. GetIndividualOrdersDates

   Zwracająca daty zamówień dla danego klienta indywidualnego

```
CREATE FUNCTION [dbo].[GetIndividualOrdersDates] (@individualID
INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.IndividualOnSiteOrdersDatesView
            WHERE CustomerID = @individualID
            UNION
            SELECT * FROM dbo.IndividualTakeAwayOrdersDatesView
            WHERE CustomerID = @individualID)
GO
```

9. GetCompanyOrdersMoney

   Zwracająca kwoty zamówień dla danej firmy

```
CREATE FUNCTION [dbo].[GetCompanyOrdersMoney] (@companyID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.CompanyOnSiteOrdersMoneyView
      WHERE CompanyID = @companyID
      UNION
      SELECT * FROM dbo.CompanyTakeAwayOrdersMoneyView
      WHERE CompanyID = @companyID)
GO
```

10. GetIndividualOrdersMoney

    Zwracająca kwoty zamówień dla danego klienta indywidualnego

```
CREATE FUNCTION [dbo].[GetIndividualOrdersMoney] (@individualID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.IndividualOnSiteOrdersMoneyView
            WHERE CustomerID = @individualID
            UNION
            SELECT * FROM dbo.IndividualTakeAwayOrdersMoneyView
            WHERE CustomerID = @individualID)
GO
```

11. GetMenuOfTheDay

Zwracająca menu obowiązujące w danym dniu

```
CREATE FUNCTION [dbo].[GetMenuOfTheDay](@day DATE, @restaurantID
INT)
      RETURNS TABLE
AS
      RETURN (SELECT DishID, DishName, Price, MenuIn, MenuOut FROM
            (SELECT DishID, DishName, Price, MenuIn, MenuOut,
ROW_NUMBER() OVER (PARTITION BY DishName ORDER BY DishID) AS row_no
            FROM dbo.MenuView
            WHERE (RestaurantID = @restaurantID
                  AND MenuIn <= @day
                  AND (MenuOut IS NULL OR MenuOut >= @day))) AS a
            WHERE a.row_no = 1)

GO
```

12. GetMenuInDishes

Zwracająca potrawy wstawione do menu w danym okresie

```
CREATE FUNCTION [dbo].[GetMenuInDishes] (@days INT, @from DATE,
@restaurantID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.MenuView
                  WHERE (RestaurantID = @restaurantID
                  AND (DATEDIFF(DAY, MenuIn, @from) BETWEEN 0 AND
@days)))
```

```
GO
```

13. GetTableReservations

Zwracająca rezerwacje danego stolika w danym okresie

```
CREATE FUNCTION [dbo].[GetTableReservations] (@days INT,@from
DATE, @tableID INT)
      RETURNS TABLE
AS
      RETURN (SELECT * FROM dbo.TableReservationsView
            WHERE (TableID = @tableID AND (DATEDIFF(DAY,
SinceWhenReserved, @from)) BETWEEN 0 AND @days))
GO
```

14. GetFreeTables

Zwracająca wolne stoliki w danym dniu w określonych godzinach (zwracana jest także dostępna ilość miejsc przy stoliku, biorąc pod uwagę obostrzenia covidowe)

```
CREATE FUNCTION [dbo].[GetFreeTables] (@restaurantID INT, @from
DATETIME, @to DATETIME)
      RETURNS TABLE
AS
      RETURN (SELECT TableID, SeatsNumber FROM dbo.Tables
                        WHERE RestaurantID = @restaurantID  AND
TableID NOT IN
      (SELECT TableID FROM dbo.TableReservations
            WHERE (@from BETWEEN SinceWhenReserved AND
UntilWhenReserved)
            OR  (@to BETWEEN SinceWhenReserved AND
UntilWhenReserved)))
GO
```

15. GetCompanyLatestMonthlyDiscount

Zwraca informację o ostatnio wystawionym rabacie miesięcznym dla klienta - firmy. Jeśli klientowi nie został nigdy przyznany rabat miesięczny, zwraca wiersz pusty. Jeśli natomiast posiada on historię przyznanych rabatów miesięcznych, zwracana jest informacja o najświeższym spośród nich. Funkcja ta jest wykorzystywana w procedurze wystawiania rabatów miesięczny dla klienta - firmy,bierze udział w

sprawdzaniu warunków ciągłości zamówień na przestrzeni kolejnych miesięcy (dot. punktu 1. sekcji “Rabaty” z Opisu Zadania).

```
CREATE FUNCTION
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID INT,
@companyID INT, @date DATETIME)
      RETURNS TABLE
AS
      RETURN (SELECT TOP 1 dbo.CompanyDiscounts.DiscountID, Since,
                  TotalDiscount, Orders, BillingPeriod,
TotalPrice,Discount, MaxDiscount
                  FROM dbo.CompanyDiscountHist INNER JOIN
dbo.CompanyDiscounts
                  ON CompanyDiscounts.DiscountID =
CompanyDiscountHist.DiscountID
                  WHERE (RestaurantID = @restaurantID AND
CustomerID = @companyID AND Since <= @date)
                  ORDER BY Since DESC)
GO
```

16. GetCompanyPastAndValidDiscounts

Zwraca rejestr historii rabatów miesięcznych wystawionych klientowi - firmie.

```
CREATE FUNCTION
[dbo].[GetCompanyPastAndValidDiscounts](@restaurantID INT,
@companyID INT)
      RETURNS TABLE
AS
      RETURN (SELECT dbo.CompanyDiscounts.DiscountID, Since,
                  TotalDiscount, Orders, BillingPeriod,
TotalPrice,Discount, MaxDiscount
                  FROM dbo.CompanyDiscountHist INNER JOIN
dbo.CompanyDiscounts
                  ON CompanyDiscounts.DiscountID =
CompanyDiscountHist.DiscountID
                  WHERE (RestaurantID = @restaurantID AND
CustomerID = @companyID))
GO
```

17. GetCurrentPermDiscountForIndividualCustomer

Zwraca informację o rabacie stałym (nieograniczony czas, na wszystkie zamówienia) danego klienta indywidualnego, który aktualnie on posiada. Jeśli klient nie posiada takiego rabatu, zwracany jest wiersz pusty. Funkcja ta jest wykorzystywana w

procedurze wystawiającej rabaty stałe dla klientów indywidualnych (dot. punktów 1. i 2. sekcji “Rabaty” z Opisu Zadania).

```
CREATE FUNCTION
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID
INT, @customerID INT, @date DATETIME)
      RETURNS TABLE
AS
      RETURN (SELECT TOP 1
dbo.PermIndividualDiscounts.RestaurantID, CustomerID,
dbo.IndividualDiscountHist.DiscountID,
      Since, OrdersNumber, MinOrderPrice, Discount, Till, UseDate
      FROM dbo.PermIndividualDiscounts INNER JOIN
dbo.IndividualDiscountHist
      ON IndividualDiscountHist.DiscountID =
PermIndividualDiscounts.DiscountID
      WHERE (RestaurantID = @restaurantID AND CustomerID =
@customerID
            AND Till IS NULL AND Since <= @date)
      GROUP BY dbo.PermIndividualDiscounts.RestaurantID,
CustomerID, dbo.IndividualDiscountHist.DiscountID,
      Since, OrdersNumber, MinOrderPrice, Discount, Till, UseDate
      ORDER BY Since DESC)
GO
```

18. GetCurrentValidTempIndividualDiscounts

Zwraca informacje o rabacie czasowym (jednorazowym) danego klienta indywidualnego, który aktualnie on posiada (rabat zgodny czasowo i jednocześnie niezużyty). Jeśli klient takiego rabatu aktualnie nie posiada, zwracany jest wiersz pusty. Funkcja ta wykorzystywana w procedurze wystawiania rabatów czasowych dla klienta indywidualnego (dot. punktów 3. i 4. sekcji “Rabaty” z Opisu Zadania).

```
CREATE FUNCTION
[dbo].[GetCurrentValidTempIndividualDiscounts](@restaurantID INT,
@customerID INT, @date DATETIME)
      RETURNS TABLE
AS
RETURN (SELECT dbo.TempIndividualDiscounts.DiscountID, CustomerID,
Since, Till, UseDate, MinTotalPrice, Duration, Discount
      FROM dbo.IndividualDiscountHist INNER JOIN
dbo.TempIndividualDiscounts
      ON TempIndividualDiscounts.DiscountID =
IndividualDiscountHist.DiscountID
      WHERE (RestaurantID = @restaurantID AND CustomerID =
```

```
@customerID AND Till IS NOT NULL AND Since <= @date AND UseDate IS
NULL)
      )
GO
```

19. GetDishesWhichCanBePlacedInMenu

Zwraca wszystkie dania, które mogą zostać wstawione do menu, tj.
takie, których nie ma w menu aktualnie oraz zostały zdjęte z menu co najmniej miesiąc temu

```
CREATE FUNCTION [dbo].[GetDishesWhichCanBePlacedInMenu] (
      @day DATE,
      @restaurantID INT
      )
      RETURNS TABLE
AS
RETURN SELECT DISTINCT DishName FROM dbo.Menu
      WHERE RestaurantID = @restaurantID AND
            DishName NOT IN (
            SELECT DishName FROM
dbo.GetMenuOfTheDay(@day,@restaurantID)
            )
            AND DishName NOT IN (
                  SELECT m.DishName FROM menu m
                  INNER JOIN dbo.MenuRegister mr
                  ON mr.DishID = m.DishID
                  WHERE mr.menuout IS NOT NULL AND
                  (DATEDIFF(MONTH, mr.menuout, @day) BETWEEN 0 AND
1) AND
                  m.RestaurantID = @restaurantID
                  )
```

20. GenerateIndividualCustomerReport

Zwraca raport dla danego klienta indywidualnego dot. liczby poszczególnych zamówień (zrealizowanych i anulowanych), łącznej wartości zamówień, liczby rezerwacji, liczby przyznanych zniżek (w danym okresie czasu).

```
CREATE FUNCTION [dbo].[GenerateIndividualCustomerReport](
     @customerID INT,
     @from DATE,
     @to DATE
```

```
      )
RETURNS @report TABLE(
       field NVARCHAR(100),
       field_value NVARCHAR(100)
       )
AS
BEGIN
      DECLARE @firstName NVARCHAR(50)
      SET @firstName = (
            SELECT FirstName FROM dbo.IndividualCustomers
            WHERE CustomerID = @customerID
            )

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'First Name',
          @firstName
          )

      DECLARE @lastName NVARCHAR(50)
      SET @lastName = (
            SELECT LastName FROM dbo.IndividualCustomers
            WHERE CustomerID = @customerID
            )
      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Last Name',
          @lastName
          )

      DECLARE @totalOnSiteOrders INT
      SET @totalOnSiteOrders = (
            SELECT count(*) FROM dbo.OnSiteOrders
            WHERE CustomerID = @customerID
            AND ExecutionDate BETWEEN @from AND @to
            AND Status NOT LIKE 'A'
            )
      INSERT @report
```

```
(
    field,
    field_value
)
VALUES
(   'Executed on site orders',
    @totalOnSiteOrders
    )


DECLARE @cancelledOSOrders INT
SET @cancelledOSOrders = (
      SELECT count(*) FROM dbo.OnSiteOrders
      WHERE CustomerID = @customerID
      AND ExecutionDate BETWEEN @from AND @to
      AND Status LIKE 'A'
      )

INSERT @report
(
    field,
    field_value
)
VALUES
(   'Cancelled on site orders',
    @cancelledOSOrders
    )

DECLARE @totalOSOrdersValue MONEY

SET @totalOSOrdersValue = (
      SELECT SUM(Quantity*UnitPrice) FROM dbo.OnSiteOrders
      INNER JOIN dbo.OrderDetails
      ON OrderDetails.OrderID = OnSiteOrders.OrderID
      INNER JOIN dbo.Menu
      ON Menu.DishID = OrderDetails.DishID
      WHERE CustomerID = @customerID
      AND Status NOT LIKE 'A'
      AND ExecutionDate BETWEEN @from AND @to
      )

IF @totalOSOrdersValue IS NULL
      SET @totalOSOrdersValue = 0

INSERT @report
(
```

```
    field,
    field_value
)
VALUES
(   'Total on site orders value',
    @totalOSOrdersValue
    )

DECLARE @totalTakeAwayOrders INT
SET @totalTakeAwayOrders = (
      SELECT count(*) FROM dbo.TakeAwayOrders
      WHERE CustomerID = @customerID
      AND ExecutionDate BETWEEN @from AND @to
      AND Status NOT LIKE 'A'
      )
INSERT @report
(
    field,
    field_value
)
VALUES
(   'Executed take away orders',
    @totalTakeAwayOrders
    )

DECLARE @cancelledTAOrders INT
SET @cancelledTAOrders = (
      SELECT count(*) FROM dbo.TakeAwayOrders
      WHERE CustomerID = @customerID
      AND ExecutionDate BETWEEN @from AND @to
      AND Status LIKE 'A'
      )
INSERT @report
(
    field,
    field_value
)
VALUES
(   'Cancelled take away orders',
    @cancelledTAOrders
    )

DECLARE @totalTAOrdersValue MONEY
SET @totalTAOrdersValue = (
      SELECT SUM(Quantity*UnitPrice) FROM dbo.TakeAwayOrders
      INNER JOIN dbo.OrderDetails
```

```
            ON OrderDetails.OrderID = TakeAwayOrders.OrderID
            INNER JOIN dbo.Menu
            ON Menu.DishID = OrderDetails.DishID
            WHERE CustomerID = @customerID
            AND Status NOT LIKE 'A'
            AND ExecutionDate BETWEEN @from AND @to
      )

      IF @totalTAOrdersValue IS NULL
            SET @totalTAOrdersValue = 0

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Total take away orders value',
          @totalTAOrdersValue
          )

      DECLARE @reservations INT
      SET @reservations = (
            SELECT COUNT(*) FROM dbo.IndividualReservations
            INNER JOIN dbo.TableReservations
            ON TableReservations.ReservationID =
IndividualReservations.ReservationID
            WHERE CustomerID = @customerID
            AND SinceWhenReserved BETWEEN @from AND @to
            )
      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Reservations number',
          @reservations
          )


      DECLARE @grantedDiscounts INT
      SET @grantedDiscounts  = (
            SELECT COUNT(*) FROM dbo.IndividualDiscountHist
            WHERE CustomerID = @customerID
            AND Since BETWEEN @from AND @to
```

```
            )

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Granted discounts',
          @grantedDiscounts
          )

      RETURN
END


GO
```

21. UseCompanyQuartalDiscount

Generuje paragon dla danego zamówienia, przy którym klient - firma korzysta ze zniżki. Drukuje informacje o cenie do zapłaty przed użyciem rabatu oraz po użyciu rabatu. Informuje pracownika o dacie wykorzystania rabatu.

```
CREATE FUNCTION [dbo].[UseCompanyQuartalDiscount](@restaurantID INT,
@companyID INT, @orderID INT, @currentDate DATETIME)
RETURNS @receipt TABLE
(
    param_name VARCHAR(250),
    param_val VARCHAR(250)
)
AS
BEGIN
    DECLARE @discountSice DATE
    SET @discountSice = (SELECT Since FROM
dbo.GetCompanyLatestQuartalDiscount(@restaurantID,@companyID,'2017-01
-01'))

                              --'2017-01-01' to początek istnienia
danych w bazie
    IF (@discountSice IS NULL)
    BEGIN
       RETURN
    END
    ELSE IF (@currentDate > DATEADD(DAY,90,@discountSice))
```

```
    BEGIN
       RETURN
    END
    ELSE
    BEGIN
       DECLARE @restaurantName VARCHAR(50)
       SET @restaurantName = (SELECT RestaurantName FROM
dbo.Restaurants WHERE RestaurantID = @restaurantID)
       INSERT @receipt
       (
            param_name,
            param_val
       )
       VALUES
       (    'Restaurant name:', -- param_name - varchar(250)
            @restaurantName  -- param_val - varchar(250)
            )
       INSERT @receipt
       (
            param_name,
            param_val
       )
       VALUES
       (    'CustomerID:', -- param_name - varchar(250)
            @companyID  -- param_val - varchar(250)
        )
        INSERT @receipt
       (
            param_name,
            param_val
       )
       VALUES
       (    'Current date:', -- param_name - varchar(250)
            @currentDate -- param_val - varchar(250)
        )
        INSERT @receipt
       (
            param_name,
            param_val
       )
       VALUES
       (    'Receipt for order number: ', -- param_name - varchar(250)
            @orderID  -- param_val - varchar(250)
       )

       DECLARE @discountID SMALLINT
```

```
	DECLARE @discountValue REAL
	SET @discountID = (SELECT DiscountID FROM
dbo.GetCompanyLatestQuartalDiscount(@restaurantID,@companyID,'2017-01
-01'))
	SET @discountValue = (SELECT Discount FROM
dbo.GetCompanyLatestQuartalDiscount(@restaurantID,@companyID,'2017-01
-01'))

	INSERT @receipt
	(
	     param_name,
	     param_val
	)
	VALUES
	(    'Quartal discount value: ', -- param_name - varchar(250)
	     @discountValue  -- param_val - varchar(250)
	)

	DECLARE @normalPricetoPay MONEY
	SET @normalPriceToPay = (SELECT dbo.GetOrderValue(@orderID))
	DECLARE @priceAfterDiscount MONEY
	SET @priceAfterDiscount = @normalPricetoPay *
(1-@discountValue)

	INSERT @receipt
	(
	     param_name,
	     param_val
	)
	VALUES
	(    'Price to pay without discount: ', -- param_name -
varchar(250)
	     @normalPricetoPay  -- param_val - varchar(250)
	     )

	INSERT @receipt
	(
	     param_name,
	     param_val
	)
	VALUES
	(    'Price to pay after discount use: ', -- param_name -
varchar(250)
	     @priceAfterDiscount  -- param_val - varchar(250)
	     )
```

```
        INSERT @receipt
        (
             param_name,
             param_val
        )
        VALUES
        (   CONCAT('Quartal discount number ',@discountID,' used
',@currentDate), -- param_name - varchar(250)
             'STATUS: USED'  -- param_val - varchar(250)
         )
    END
    RETURN
END
GO
```

22. GenerateRestaurantReport

Generuje raport dla danej restauracji w danym przedziale czasowym, dot. liczby stolików, liczby miejsc, łącznej wartości zamówień i ich liczby

```
CREATE FUNCTION [dbo].[GenerateRestaurantReport](
      @restaurantID INT,
      @from DATE,
      @to DATE
      )
RETURNS @report TABLE(
       field NVARCHAR(100),
       field_value NVARCHAR(100)
       )
AS
BEGIN
      DECLARE @restaurantName NVARCHAR(50)
      SET @restaurantName = (
            SELECT RestaurantName FROM dbo.Restaurants
            WHERE RestaurantID = @restaurantID
            )

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
```

```
      (   'Restaurant name',
          @restaurantName
          )

      DECLARE @tables INT
      SET @tables = (
            SELECT COUNT(*) FROM dbo.Tables
            WHERE RestaurantID = @restaurantID
            )

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Tables number',
          @tables
          )

      DECLARE @seatsNumber INT
      SET @seatsNumber = (
            SELECT SUM(SeatsNumber) FROM dbo.Tables
            WHERE RestaurantID = @restaurantID
            )

      INSERT @report
      (
          field,
          field_value
      )
      VALUES
      (   'Total seats number',
          @seatsNumber
          )

      DECLARE @menuDishes INT
      SET @menuDishes = (
            SELECT COUNT(DISTINCT DishName) FROM dbo.MenuRegister
            INNER JOIN dbo.Menu
            ON Menu.DishID = MenuRegister.DishID
            WHERE RestaurantID = @restaurantID AND (MenuIn BETWEEN
@from AND @to)
            )

      INSERT @report
```

```
(
    field,
    field_value
)
VALUES
(   'Dishes number',
      @menuDishes
    )

DECLARE @oSordersNumber INT
SET @oSordersNumber = (
      SELECT COUNT(*) FROM dbo.Customers
      INNER JOIN dbo.OnSiteOrders
      ON OnSiteOrders.CustomerID = Customers.CustomerID
      WHERE RestaurantID = @restaurantID
      AND ExecutionDate BETWEEN @from AND @to
      )
DECLARE @tAOrdersNumber INT
SET @tAOrdersNumber = (
      SELECT COUNT(*) FROM dbo.Customers
      INNER JOIN dbo.TakeAwayOrders
      ON TakeAwayOrders.CustomerID = Customers.CustomerID
      WHERE RestaurantID = @restaurantID
      AND ExecutionDate BETWEEN @from AND @to
      )

INSERT @report
(
    field,
    field_value
)
VALUES
(   'Total orders number',
    @tAOrdersNumber + @oSordersNumber
    )

DECLARE @tAOrdersMoney MONEY
SET @tAOrdersMoney = (
      SELECT SUM(UnitPrice*Quantity) FROM dbo.Customers
      INNER JOIN dbo.TakeAwayOrders
      ON TakeAwayOrders.CustomerID = Customers.CustomerID
      INNER JOIN dbo.OrderDetails
      ON OrderDetails.OrderID = TakeAwayOrders.OrderID
      WHERE RestaurantID = @restaurantID
      AND ExecutionDate BETWEEN @from AND @to
      )
```

```
	IF @tAOrdersMoney IS NULL
		SET @tAOrdersMoney = 0


	DECLARE @oSOrdersMoney MONEY
	SET @oSOrdersMoney = (
		SELECT SUM(Quantity*UnitPrice) FROM dbo.Customers
		INNER JOIN dbo.OnSiteOrders
		ON OnSiteOrders.CustomerID = Customers.CustomerID
		INNER JOIN dbo.OrderDetails
		ON OrderDetails.OrderID = OnSiteOrders.OrderID
		WHERE RestaurantID = @restaurantID
		AND ExecutionDate BETWEEN @from AND @to
		)

	IF @oSOrdersMoney IS NULL
		SET @oSOrdersMoney = 0

	INSERT @report
	(
	    field,
	    field_value
	)
	VALUES
	(   'Total orders money',
		@tAOrdersMoney + @oSOrdersMoney
	    )


	RETURN
END
GO
```

23. GenerateCompanyCustomerReport

```
CREATE FUNCTION GenerateCompanyCustomerReport(
	@customerID INT,
	@from DATE,
	@to DATE
	)

RETURNS @report TABLE(
	 field NVARCHAR(100),
	 field_value NVARCHAR(100)
	 )
AS
```

```
BEGIN
	DECLARE @companyName NVARCHAR(50)
	SET @companyName = (
		SELECT CompanyName FROM dbo.CompanyCustomers
		WHERE CompanyID = @customerID
		)

	INSERT @report
	(
	    field,
	    field_value
	)
	VALUES
	(   'Company Name',
	    @companyName
	    )

	DECLARE @nip NVARCHAR(50)
	SET @nip = (
		SELECT NIP FROM dbo.CompanyCustomers
		WHERE CompanyName = @customerID
		)

	INSERT @report
	(
	    field,
	    field_value
	)
	VALUES
	(   'NIP',
	    @nip
	    )

	DECLARE @orders INT
	SET @orders = (
		SELECT COUNT(*) FROM dbo.OnSiteOrders
		WHERE CustomerID = @customerID
		AND ExecutionDate BETWEEN @from AND @to
		)

	DECLARE @tOrders INT
	SET @tOrders = (
		SELECT COUNT(*) FROM dbo.TakeAwayOrders
		WHERE CustomerID = @customerID
		AND ExecutionDate BETWEEN @from AND @to
		)
```

```
    INSERT @report
    (
        field,
        field_value
    )
    VALUES
    (   'Total orders number',
        @orders + @tOrders
        )
    RETURN

END
```

5. Funkcje zwracające wartości skalarne

1. CheckIfHalfPositionsChanged

   Informująca pracownika, czy co najmniej połowa pozycji aktualnego menu została zmieniona w ciągu ostatnich dwóch tygodni

```
CREATE FUNCTION [dbo].[CheckIfHalfPositionsChanged](@day DATE,
@restaurantID INT)
      RETURNS NVARCHAR(100)
AS
BEGIN
      DECLARE @all INT
      SELECT @all = COUNT(*) FROM GetMenuOfTheDay(@day,
@restaurantID)
      DECLARE @changed INT
      SELECT @changed = COUNT(CASE WHEN DATEDIFF(DAY, MenuIn,
@day) BETWEEN 0 AND 14  THEN 1 ELSE NULL END)
            FROM GetMenuOfTheDay(@day, @restaurantID)
      IF @changed >= CAST(@all AS FLOAT)/2
      BEGIN
            RETURN
            'OK, co najmniej połowa pozycji menu została zmieniona
min 2 tygodnie temu'
      END
      RETURN 'Więcej niż połowa pozycji nie została zmieniona w
ciągu ostatnich 2 tygodni, zmień Menu!'
END
GO
```

2. IsCompanyCustomer

   Zwracająca bit 0/1 w zależności od tego, czy dany klient, jest klientem firmowym

```
CREATE FUNCTION [dbo].[IsCompanyCustomer](@customerID INT)
      RETURNS BIT
AS
BEGIN
      IF ((SELECT CustomerID FROM Customers WHERE (CustomerID IN
(SELECT CompanyID FROM dbo.CompanyCustomers)
            AND CustomerID = @customerID)) = @customerID)
            RETURN 1
      RETURN 0
END
GO
```

3. PastOrdersCount

   Zwracająca łączną liczbę zamówień danego klienta w danym okresie czasu

```
CREATE FUNCTION [dbo].[PastOrdersCount] (@customerID INT, @since
DATETIME, @until DATETIME)
      RETURNS INT
AS
BEGIN
      DECLARE @onSiteOrdersCount INT
            SET @onSiteOrdersCount =
                  (SELECT COUNT(*)
                        FROM Customers INNER JOIN
dbo.IndividualReservations
                        ON IndividualReservations.CustomerID =
Customers.CustomerID
                        INNER JOIN dbo.OnSiteOrders
                        ON OnSiteOrders.OrderID =
IndividualReservations.OrderID
                        INNER JOIN dbo.OrderDetails
                        ON OrderDetails.OrderID =
OnSiteOrders.OrderID
                        WHERE (Customers.CustomerID = @customerID
AND dbo.OnSiteOrders.OrderDate >= @since
                        AND dbo.OnSiteOrders.OrderDate < @until)
                  )
            DECLARE @takeAwayOrdersCount INT
```

```
            SET @takeAwayOrdersCount =
                  (SELECT COUNT(*)
                        FROM Customers INNER JOIN
dbo.TakeAwayOrders
                        ON TakeAwayOrders.CustomerID =
Customers.CustomerID
                        INNER JOIN dbo.OrderDetails
                        ON OrderDetails.OrderID =
TakeAwayOrders.OrderID
                        WHERE (Customers.CustomerID = @customerID
AND dbo.TakeAwayOrders.OrderDate >= @since
                        AND dbo.TakeAwayOrders.OrderDate < @until)
                  )
            IF (@onSiteOrdersCount IS NOT NULL)
                  IF (@takeAwayOrdersCount IS NOT NULL)
                        RETURN (@onSiteOrdersCount +
@takeAwayOrdersCount)
                  ELSE RETURN @onSiteOrdersCount
            ELSE IF (@takeAwayOrdersCount IS NOT NULL)
                  RETURN @takeAwayOrdersCount
            RETURN 0
END
GO
```

4. PastOrdersValue

   Zwracająca łączną wartość zamówień danego klienta w danym okresie czasu

```
CREATE FUNCTION [dbo].[PastOrdersValue](@customerID INT, @date DATETIME)
      RETURNS MONEY
AS
BEGIN
      DECLARE @onSiteOrdersValue INT
      SET @onSiteOrdersValue =
                  (SELECT SUM(UnitPrice * Quantity)
                  FROM Customers INNER JOIN dbo.IndividualReservations
                  ON IndividualReservations.CustomerID =
Customers.CustomerID
                  INNER JOIN dbo.OnSiteOrders
                  ON OnSiteOrders.OrderID =
IndividualReservations.OrderID
                  INNER JOIN dbo.OrderDetails
                  ON OrderDetails.OrderID = OnSiteOrders.OrderID
                  WHERE (Customers.CustomerID = @customerID AND
```

```
dbo.OnSiteOrders.OrderDate <= @date
                  AND dbo.OnSiteOrders.Status != 'A')
                  )

      DECLARE @takeAwayOrdersValue INT
      SET @takeAwayOrdersValue =
                  (SELECT SUM(UnitPrice * Quantity)
                  FROM dbo.Customers INNER JOIN dbo.TakeAwayOrders
                  ON TakeAwayOrders.CustomerID = Customers.CustomerID
                  INNER JOIN dbo.OrderDetails
                  ON OrderDetails.OrderID = TakeAwayOrders.OrderID
                  WHERE (Customers.CustomerID = @customerID AND
dbo.TakeAwayOrders.OrderDate <= @date
                  AND TakeAwayOrders.Status != 'A')
                  )
      IF (@onSiteOrdersValue IS NOT NULL)
            IF (@takeAwayOrdersValue IS NOT NULL)
            RETURN (@onSiteOrdersValue + @takeAwayOrdersValue)
            ELSE RETURN @onSiteOrdersValue
      ELSE IF (@takeAwayOrdersValue IS NOT NULL)
            RETURN @takeAwayOrdersValue
      RETURN 0
END
GO
```

5. PastOrderValueSinceTill

Zwraca wartość wszystkich zamówień w danym okresie czasu.

```
CREATE FUNCTION [dbo].[PastOrdersValueSinceTill](@customerID INT, @since
DATETIME, @till DATETIME)
    RETURNS MONEY
AS
BEGIN
    DECLARE @onSiteOrdersValue INT
    SET @onSiteOrdersValue =
              (SELECT SUM(UnitPrice * Quantity)
              FROM dbo.OnSiteOrders INNER JOIN dbo.OrderDetails
              ON OrderDetails.OrderID = OnSiteOrders.OrderID
              WHERE (dbo.OnSiteOrders.CustomerID = @customerID
              AND dbo.OnSiteOrders.OrderDate >= @since AND
dbo.OnSiteOrders.OrderDate <= @till
              AND dbo.OnSiteOrders.Status != 'A')
              )
```

```
    DECLARE @takeAwayOrdersValue INT
    SET @takeAwayOrdersValue =
             (SELECT SUM(UnitPrice * Quantity)
             FROM dbo.TakeAwayOrders INNER JOIN dbo.OrderDetails
             ON OrderDetails.OrderID = TakeAwayOrders.OrderID
             WHERE (dbo.TakeAwayOrders.CustomerID = @customerID
             AND dbo.TakeAwayOrders.OrderDate >= @since AND
dbo.TakeAwayOrders.OrderDate <= @till
             AND dbo.TakeAwayOrders.Status != 'A')
             )
    IF (@onSiteOrdersValue IS NOT NULL)
       IF (@takeAwayOrdersValue IS NOT NULL)
       RETURN (@onSiteOrdersValue + @takeAwayOrdersValue)
       ELSE RETURN @onSiteOrdersValue
    ELSE IF (@takeAwayOrdersValue IS NOT NULL)
       RETURN @takeAwayOrdersValue
    RETURN 0
END
GO
```

6. GetHighestPermanentIndividualDiscountInARestaurant

Zwraca numer stałej zniżki o wyższej wartości rabatu w danej restauracji.

```
CREATE FUNCTION
[dbo].[GetHighestPermanentIndividualDiscountInARestaurant](@restaurantID
INT)
    RETURNS SMALLINT
AS
BEGIN
    RETURN (SELECT TOP 1 DiscountID FROM dbo.PermIndividualDiscounts
                                          WHERE RestaurantID =
@restaurantID
                                          ORDER BY Discount DESC)
END
GO
```

7. GetLowestPermanentIndividualDiscountInARestaurant

Zwraca numer stałej zniżki o niższej wartości rabatu w danej restauracji.

```
CREATE FUNCTION
[dbo].[GetLowestPermanentIndividualDiscountInARestaurant](@restaurantID
INT)
    RETURNS SMALLINT
AS
BEGIN
```

```
    RETURN (SELECT TOP 1 DiscountID FROM dbo.PermIndividualDiscounts
             WHERE RestaurantID = @restaurantID
             ORDER BY Discount ASC)
END
GO
```

8. GetHighestTempIndividualDiscountInARestaurant

Zwraca numer czasowej zniżki o wyższej wartości rabatu w danej restauracji.

```
CREATE FUNCTION
[dbo].[GetHighestTempIndividualDiscountInARestaurant](@restaurantID INT)
    RETURNS SMALLINT
AS
BEGIN
    RETURN (SELECT TOP 1 DiscountID FROM dbo.TempIndividualDiscounts
             WHERE RestaurantID = @restaurantID
             ORDER BY Discount DESC)
END
GO
```

9. GetLowestTempIndividualDiscountInARestaurant

Zwraca numer czasowej zniżki o niższej wartości rabatu w danej restauracji.

```
CREATE FUNCTION
[dbo].[GetLowestTempIndividualDiscountInARestaurant](@restaurantID INT)
    RETURNS SMALLINT
AS
BEGIN
    RETURN (SELECT TOP 1 DiscountID FROM dbo.TempIndividualDiscounts
             WHERE RestaurantID = @restaurantID
             ORDER BY Discount ASC)
END
GO
```

10. PastCompanyOrdersCountWithinBillingPeriod

Zwraca liczbę zamówień dokonaną w okresie rozliczeniowym danej firmy.

```
CREATE FUNCTION
[dbo].[PastCompanyOrdersCountWithinBillingPeriod](@companyID INT, @since
DATETIME, @billingPeriod INT)
    RETURNS INT
AS
BEGIN
```

```
    DECLARE @onSiteOrdersCount INT
    SET @onSiteOrdersCount = (SELECT COUNT(*)
              FROM OnSiteOrders
              WHERE (OnSiteOrders.CustomerID = @companyID
              AND dbo.OnSiteOrders.OrderDate >= @since
              AND dbo.OnSiteOrders.OrderDate <= DATEADD(DAY,
@billingPeriod, @since)
              AND dbo.OnSiteOrders.Status != 'A'))

    DECLARE @takeAwayOrdersCount INT
    SET @takeAwayOrdersCount = (SELECT COUNT(*)
              FROM dbo.TakeAwayOrders
              WHERE (dbo.TakeAwayOrders.CustomerID = @companyID
              AND dbo.TakeAwayOrders.OrderDate >= @since
              AND dbo.TakeAwayOrders.OrderDate <= DATEADD(DAY,
@billingPeriod, @since)
              AND dbo.TakeAwayOrders.Status != 'A'))

    IF (@onSiteOrdersCount IS NOT NULL)
    BEGIN
       IF (@takeAwayOrdersCount IS NOT NULL)
       BEGIN
             RETURN (@onSiteOrdersCount + @takeAwayOrdersCount)
       END
       ELSE
       BEGIN
             RETURN @onSiteOrdersCount
       END
    END
    ELSE IF (@takeAwayOrdersCount IS NOT NULL)
    BEGIN
       RETURN @takeAwayOrdersCount
    END
    RETURN 0
END
GO
```

11. PastCompanyOrdersValueWithinBillingPeriod

Zwraca wartość zamówień firmowych dokonanych w okresie rozliczeniowych firmy.

```
CREATE FUNCTION
[dbo].[PastCompanyOrdersValueWithinBillingPeriod](@companyID INT, @since
DATETIME, @billingPeriod INT)
    RETURNS INT
AS
BEGIN
```

```
    DECLARE @onSiteOrdersValue MONEY
    SET @onSiteOrdersValue = (SELECT SUM(UnitPrice * Quantity)
              FROM OnSiteOrders
              INNER JOIN dbo.OrderDetails
              ON OrderDetails.OrderID = OnSiteOrders.OrderID
              WHERE (OnSiteOrders.CustomerID = @companyID
              AND dbo.OnSiteOrders.OrderDate >= @since
              AND dbo.OnSiteOrders.OrderDate <= DATEADD(DAY,
@billingPeriod, @since)
              AND dbo.OnSiteOrders.Status != 'A'))

    DECLARE @takeAwayOrdersValue MONEY
    SET @takeAwayOrdersValue = (SELECT SUM(UnitPrice * Quantity)
              FROM TakeAwayOrders
              INNER JOIN dbo.OrderDetails
              ON OrderDetails.OrderID = dbo.TakeAwayOrders.OrderID
              WHERE (dbo.TakeAwayOrders.CustomerID = @companyID
              AND dbo.TakeAwayOrders.OrderDate >= @since
              AND dbo.TakeAwayOrders.OrderDate <= DATEADD(DAY,
@billingPeriod, @since)
              AND dbo.TakeAwayOrders.Status != 'A'))
    IF (@onSiteOrdersValue IS NOT NULL)
    BEGIN
       IF (@takeAwayOrdersValue IS NOT NULL)
       BEGIN
             RETURN (@onSiteOrdersValue + @takeAwayOrdersValue)
       END
       ELSE
       BEGIN
             RETURN @onSiteOrdersValue
       END
    END
    ELSE IF (@takeAwayOrdersValue IS NOT NULL)
    BEGIN
       RETURN @takeAwayOrdersValue
    END
    RETURN 0
END
GO
```

12. PastOrdersCountWithGivenPrice

    Zwracająca łączną liczbę zamówień klienta, o danej minimalnej wartości zamówienia, w danym okresie czasu

```
CREATE FUNCTION [dbo].[PastOrdersCountWithGivenPrice](@minimalCost
```

```
MONEY, @customerID INT, @since DATETIME, @until DATETIME)
 RETURNS INT
AS
BEGIN
      DECLARE @onSiteOrdersCount INT
      SET @onSiteOrdersCount =
            (SELECT COUNT(*)
                  FROM Customers INNER JOIN
dbo.IndividualReservations
                  ON IndividualReservations.CustomerID =
Customers.CustomerID
                  INNER JOIN dbo.OnSiteOrders
                  ON OnSiteOrders.OrderID =
IndividualReservations.OrderID
                  INNER JOIN dbo.OrderDetails
                  ON OrderDetails.OrderID = OnSiteOrders.OrderID
                  WHERE (Customers.CustomerID = @customerID AND
dbo.OnSiteOrders.OrderDate >= @since
                  AND dbo.OnSiteOrders.OrderDate < @until AND
(Quantity * UnitPrice >= @minimalCost))
            )
      DECLARE @takeAwayOrdersCount INT
      SET @takeAwayOrdersCount =
            (SELECT COUNT(*)
                  FROM Customers INNER JOIN dbo.TakeAwayOrders
                  ON TakeAwayOrders.CustomerID =
Customers.CustomerID
                  INNER JOIN dbo.OrderDetails
                  ON OrderDetails.OrderID = TakeAwayOrders.OrderID
                  WHERE (Customers.CustomerID = @customerID AND
dbo.TakeAwayOrders.OrderDate >= @since
                  AND dbo.TakeAwayOrders.OrderDate < @until AND
(Quantity * UnitPrice >= @minimalCost))
            )
      IF (@onSiteOrdersCount IS NOT NULL)
            IF (@takeAwayOrdersCount IS NOT NULL)
                  RETURN (@onSiteOrdersCount +
@takeAwayOrdersCount)
            ELSE RETURN @onSiteOrdersCount
      ELSE IF (@takeAwayOrdersCount IS NOT NULL)
            RETURN @takeAwayOrdersCount
      RETURN 0
END
GO
```

6. Procedury

1. **GivePermanentIndividualDiscount**

   Wystawia klientowi indywidualnemu rabat stały, jeśli spełnia on warunki do jego przyznania.

```
CREATE PROCEDURE [dbo].[GivePermanentIndividualDiscount]
      @restaurantID INT,
      @customerID INT,
      @date DATETIME
AS
BEGIN
      IF ((SELECT [dbo].[IsCompanyCustomer](@customerID)) = 1)
      BEGIN
            ;THROW 52000, 'Wymagany klient indywidualny,
wprowadzono klienta-firmę',1
      END
      IF ((SELECT COUNT(*) FROM
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID,
@customerID,@date)) = 0)
      BEGIN
            DECLARE @givenDiscount SMALLINT
            SET @givenDiscount = (SELECT
[dbo].[GetLowestPermanentIndividualDiscountInARestaurant](@restaur
antID))
            DECLARE @minimalCost money
            SET @minimalCost = (SELECT MinOrderPrice FROM
dbo.PermIndividualDiscounts WHERE DiscountID = @givenDiscount)
            DECLARE @ordersNumber INT
            SET @ordersNumber = (SELECT OrdersNumber FROM
dbo.PermIndividualDiscounts WHERE DiscountID = @givenDiscount)
            DECLARE @pastOrdersCount INT
            SET @pastOrdersCount = (SELECT
[dbo].[PastOrdersCountWithGivenPrice](@minimalCost, @customerID,
'2017-01-01', @date))
            IF (@pastOrdersCount >= @ordersNumber)
            BEGIN
                  SET IDENTITY_INSERT dbo.IndividualDiscountHist
ON
                  INSERT INTO dbo.IndividualDiscountHist
                  (
```

```
                        CustomerID,
                        DiscountID,
                        Since,
                        Till,
                        UseDate
                  )
                  VALUES
                  (   @customerID,
                        @givenDiscount,
                        @date,
                        NULL,
                        NULL
                  )
            END
            ELSE
            BEGIN
            ;THROW 52000, 'Klient w tym momencie nie spełnia
warunków do wystawienia rabatu',1
            END
      END
      ELSE
      BEGIN
            DECLARE @discountID SMALLINT
            SET @discountID = (SELECT DiscountID FROM
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID,
@customerID,@date))
            SET @minimalCost = (SELECT MinOrderPrice FROM
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID,
@customerID,@date))
            SET @ordersNumber = (SELECT OrdersNumber FROM
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID,
@customerID,@date))
            DECLARE @since DATETIME
            SET @since = (SELECT Since FROM
[dbo].[GetCurrentPermDiscountForIndividualCustomer](@restaurantID,
@customerID, @date))
            SET @pastOrdersCount = (SELECT
[dbo].[PastOrdersCountWithGivenPrice](@minimalCost, @customerID,
@since, @date))
            IF (@pastOrdersCount >= 2*@ordersNumber)
            BEGIN
            SET @givenDiscount = (SELECT
[dbo].[GetHighestPermanentIndividualDiscountInARestaurant](@restau
rantID))
            SET IDENTITY_INSERT dbo.IndividualDiscountHist ON
            INSERT INTO dbo.IndividualDiscountHist
```

```
            (
                  CustomerID,
                  DiscountID,
                  Since,
                  Till,
                  UseDate
            )
            VALUES
            (   @customerID,
                  @givenDiscount,
                  @date,
                  NULL,
                  NULL
            )
            END
            ELSE
            BEGIN
            ;THROW 52000, 'Klient posiada już rabat tego rodzaju i
nie spełnia warunków do podniesienia jego wartości',1
            END
      END
END
GO
```

2. GiveTemporalIndividualDiscount

Wystawia klientowi indywidualnemu rabat czasowy (na określoną liczbę dni), jeśli spełnia on warunki do jego przyznania.

```
CREATE PROCEDURE [dbo].[GiveTemporalIndividualDiscount]
      @restaurantID INT,
      @customerID INT,
      @date DATETIME,
      @givenDiscountID SMALLINT OUTPUT
AS
BEGIN
      IF ((SELECT COUNT(*) FROM
[dbo].[GetCurrentValidTempIndividualDiscounts](@restaurantID,
@customerID,@date)) = 0)
      BEGIN
            DECLARE @pastOrdersValue MONEY
            SET @pastOrdersValue = (SELECT
[dbo].[PastOrdersValue](@customerID, @date))
            DECLARE @discountToGive1 SMALLINT
```

```
            SET @discountToGive1 = (SELECT
[dbo].[GetLowestTempIndividualDiscountInARestaurant](@restaurantID)
)
            DECLARE @requiredPastOrdersValue1 MONEY
            SET @requiredPastOrdersValue1 = (SELECT MinTotalPrice
FROM dbo.TempIndividualDiscounts WHERE DiscountID =
@discountToGive1)
            DECLARE @discountToGive2 SMALLINT
            SET @discountToGive2 = (SELECT
[dbo].[GetHighestTempIndividualDiscountInARestaurant](@restaurantID
))
            DECLARE @requiredPastOrdersValue2 MONEY
            SET @requiredPastOrdersValue2 = (SELECT MinTotalPrice
FROM dbo.TempIndividualDiscounts WHERE DiscountID =
@discountToGive2)
            IF (@pastOrdersValue >= @requiredPastOrdersValue1 AND
@pastOrdersValue < @requiredPastOrdersValue2)
            BEGIN
                  DECLARE @duration INT
                  SET @duration = (SELECT Duration FROM
dbo.TempIndividualDiscounts WHERE DiscountID = @discountToGive1)
                  SET IDENTITY_INSERT dbo.IndividualDiscountHist
ON;
                  INSERT INTO dbo.IndividualDiscountHist
                  (
                      CustomerID,
                        DiscountID,
                      Since,
                      Till,
                      UseDate
                  )
                  VALUES
                  (
                        @customerID,
                        @discountToGive1,
                      @date,
                      DATEADD(DAY,@duration,@date),
                      NULL
                      )
                  SET @givenDiscountID = @discountToGive1
            END
            ELSE IF (@pastOrdersValue >= @requiredPastOrdersValue2)
            BEGIN
                  SET @duration = (SELECT Duration FROM
dbo.TempIndividualDiscounts WHERE DiscountID = @discountToGive2)
                  SET IDENTITY_INSERT dbo.IndividualDiscountHist
```

```
ON;
                  INSERT INTO dbo.IndividualDiscountHist
                  (
                      CustomerID,
                        DiscountID,
                      Since,
                      Till,
                      UseDate
                  )
                  VALUES
                  (
                        @customerID,
                        @discountToGive2,
                      @date,
                      DATEADD(DAY,@duration,@date),
                      NULL
                      )
                  SET @givenDiscountID = @discountToGive2
            END
            ELSE
            BEGIN
                  ;THROW 52000, 'Ten klient obecnie nie spełnia
wymagań do przyznania rabatu czasowego',1
            END
      END
      ELSE IF (@pastOrdersValue >= @requiredPastOrdersValue2)
      BEGIN
            SET @duration = (SELECT Duration FROM
dbo.TempIndividualDiscounts WHERE DiscountID = @discountToGive2)
                        SET IDENTITY_INSERT
dbo.IndividualDiscountHist ON;
                        INSERT INTO dbo.IndividualDiscountHist
                        (
                              CustomerID,
                              DiscountID,
                              Since,
                              Till,
                              UseDate
                        )
                        VALUES
                        (
                              @customerID,
                              @discountToGive2,
                              @date,
                              DATEADD(DAY,@duration,@date),
                              NULL
```

```
                              )
                        SET @givenDiscountID = @discountToGive2
      END
      ELSE
      BEGIN
            ;THROW 52000, 'Ten klient obecnie nie spełnia wymagań
do przyznania nowego rabatu czasowego',1
            SET @givenDiscountID = NULL
      END
END

GO
```

3. GiveCompanyMonthlyDiscount

Wystawia klientowi-firmie rabat miesięczny, jeśli spełnia on warunki do jego przyznania.

```
CREATE PROCEDURE [dbo].[GiveCompanyMonthlyDiscount]
      @restaurantID INT,
      @companyID INT,
      @date DATETIME
AS
BEGIN
      IF ((SELECT COUNT(*) FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date)) = 1)
      BEGIN
            DECLARE @discountToContinue SMALLINT
            SET @discountToContinue = (SELECT DiscountID FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
            DECLARE @billingPeriod INT
            SET @billingPeriod = (SELECT BillingPeriod FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
            DECLARE @lastDate DATETIME
            SET @lastDate = (SELECT Since FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
            DECLARE @ordersValueWithinBillingPeriod MONEY
            SET @ordersValueWithinBillingPeriod = (SELECT
```

```
[dbo].[PastCompanyOrdersValueWithinBillingPeriod](@companyID,
@lastDate, @billingPeriod))
            DECLARE @ordersCountWithinBillingPeriod INT
            SET @ordersCountWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersCountWithinBillingPeriod](@companyID,
@lastDate, @billingPeriod))
            DECLARE @requiredOrdersValue MONEY
            SET @requiredOrdersValue = (SELECT TotalPrice FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
            DECLARE @requiredOrdersCount INT
            SET @requiredOrdersCount = (SELECT Orders FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))

            IF ((DATEADD(DAY, @billingPeriod, @lastDate) < @date)
OR (@ordersCountWithinBillingPeriod < @requiredOrdersCount) OR
(@ordersValueWithinBillingPeriod < @requiredOrdersValue))
            BEGIN
                  DECLARE @discountToGive SMALLINT
                  SET @discountToGive = (SELECT TOP 1 DiscountID
FROM dbo.CompanyDiscounts WHERE RestaurantID = @restaurantID AND
BillingPeriod = @billingPeriod ORDER BY Discount ASC)
                  SET @requiredOrdersValue = (SELECT TotalPrice
FROM dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
                  SET @requiredOrdersCount = (SELECT Orders FROM
dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
                  SET @ordersValueWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersValueWithinBillingPeriod](@companyID,
DATEADD(DAY, -1*@billingPeriod, @date), @billingPeriod))
                  SET @ordersCountWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersCountWithinBillingPeriod](@companyID,
DATEADD(DAY, -1*@billingPeriod, @date), @billingPeriod))
                  IF (@ordersCountWithinBillingPeriod >=
@requiredOrdersCount AND @ordersValueWithinBillingPeriod >=
@requiredOrdersValue)
                  BEGIN
                        DECLARE @discount REAL
                        SET @discount = (SELECT Discount FROM
dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
                        SET IDENTITY_INSERT dbo.CompanyDiscountHist
ON;
                        INSERT INTO dbo.CompanyDiscountHist
                        (
                              CustomerID,
                              DiscountID,
```

```
                              Since,
                              TotalDiscount
                        )
                        VALUES
                        (   @companyID,
                              @discountToGive,
                              @date,
                              @discount
                              )
                  END
                  ELSE
                  BEGIN
                        ;THROW 52000, 'Klient nie spełnił wymagań
ciągłości zamówień, jego rabat został anulowany. Nie spełnia również
wymagań do przyznania nowego rabatu za ostatni okres
rozliczeniowy',1
                  END
            END
            ELSE
            BEGIN
                  SET @ordersValueWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersValueWithinBillingPeriod](@companyID,
@lastDate, @billingPeriod))
                  SET @ordersCountWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersCountWithinBillingPeriod](@companyID,
@lastDate, @billingPeriod))
                  SET @requiredOrdersValue = (SELECT TotalPrice
FROM [dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID,
@companyID, @date))
                  SET @requiredOrdersCount = (SELECT Orders FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
                  DECLARE @maxDiscount REAL
                  SET @maxDiscount = (SELECT MaxDiscount FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
                  IF (@ordersValueWithinBillingPeriod >=
@requiredOrdersValue AND @ordersCountWithinBillingPeriod >=
@requiredOrdersCount)
                  BEGIN
                        DECLARE @discountUpdate REAL
                        SET @discountUpdate = ((SELECT
TotalDiscount FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date))
                                                      +
```

```
(SELECT Discount FROM
[dbo].[GetCompanyLatestMonthlyDiscount](@restaurantID, @companyID,
@date)))
                        IF (@discountUpdate <= @maxDiscount)
                        BEGIN
                              INSERT INTO dbo.CompanyDiscountHist
                              (
                                    CustomerID,
                                    DiscountID,
                                    Since,
                                    TotalDiscount
                              )
                              VALUES
                              (   @companyID,
                                    @discountToContinue,
                                    @date,
                                    @discountUpdate
                                    )
                        END
                        ELSE
                        BEGIN
                              ;THROW 52000, 'Klient spełnił
wymagania ciągłości zamówień. Rabat nie ulega zmianom, ponieważ
osiągnięto rabat maksymalny',1
                        END
                  END
            END
      END
      ELSE
      BEGIN
            SET @discountToGive = (SELECT TOP 1 DiscountID FROM
dbo.CompanyDiscounts WHERE RestaurantID = @restaurantID AND
BillingPeriod = @billingPeriod ORDER BY Discount ASC)
            SET @requiredOrdersValue = (SELECT TotalPrice FROM
dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
            SET @requiredOrdersCount = (SELECT Orders FROM
dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
            SET @ordersValueWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersValueWithinBillingPeriod](@companyID,
DATEADD(DAY, -1*@billingPeriod, @date), @billingPeriod))
            SET @ordersCountWithinBillingPeriod = (SELECT
[dbo].[PastCompanyOrdersCountWithinBillingPeriod](@companyID,
DATEADD(DAY, -1*@billingPeriod, @date), @billingPeriod))
            IF (@ordersCountWithinBillingPeriod >=
@requiredOrdersCount AND @ordersValueWithinBillingPeriod >=
@requiredOrdersValue)
```

```
            BEGIN
                  SET @discount = (SELECT Discount FROM
dbo.CompanyDiscounts WHERE DiscountID = @discountToGive)
                  SET IDENTITY_INSERT dbo.CompanyDiscountHist ON;
                  INSERT INTO dbo.CompanyDiscountHist
                  (
                        CustomerID,
                        DiscountID,
                        Since,
                        TotalDiscount
                  )
                  VALUES
                  (   @companyID,
                        @discountToGive,
                        @date,
                        @discount
                        )
            END
            ELSE
            BEGIN
                  ;THROW 52000, 'Klient nie posiadał do tej pory
rabatu firmowego i nie spełnia aktualnie wymagań do jego
przyznania.',1
            END
      END
END

GO
```

4. GiveCompanyQuartalDiscount

Umożliwia przyznanie klientowi - firmie rabatu kwartalnego (dot. punktów 5. i 6. z sekcji “Rabaty” z Opisu Zadania). Procedura sprawdza, czy klient nie stara się o rabat kwartalny zbyt wcześnie (przed końcem upływu kolejnego kwartału), a następnie czy spełnia on warunki do jego wystawienia.

```
CREATE PROCEDURE [dbo].[GiveCompanyQuartalDiscount]
    @restaurantID INT,
    @companyID INT,
    @date DATETIME
AS
BEGIN
    IF ((SELECT COUNT(*) FROM
[dbo].[GetCompanyLatestQuartalDiscount](@restaurantID, @companyID,
@date)) = 1)
    BEGIN
```

```
       DECLARE @quartal INT
       SET @quartal = (SELECT BillingPeriod FROM
[dbo].[GetCompanyLatestQuartalDiscount](@restaurantID, @companyID,
@date))
       DECLARE @since DATETIME
       SET @since = (SELECT Since FROM
[dbo].[GetCompanyLatestQuartalDiscount](@restaurantID, @companyID,
@date))
       IF (DATEDIFF(DAY, @since, @date) < @quartal)
       BEGIN
             ;THROW 52000,'Nie można aktualnie przyznać rabatu
kwartalnego. Od przyznania ostatniego rabatu kwartalnego minęło mniej niż
kwartał',1
       END
    END
    ELSE
    BEGIN
       DECLARE @priceRequirementToMeet MONEY
       SET @quartal = 90
       SET @priceRequirementToMeet = (SELECT TotalPrice FROM
dbo.CompanyDiscounts WHERE RestaurantID = @restaurantID AND
BillingPeriod = @quartal)
       DECLARE @discountToGive REAL
       SET @discountToGive = (SELECT Discount FROM dbo.CompanyDiscounts
WHERE RestaurantID = @restaurantID AND BillingPeriod = @quartal)
       DECLARE @givenDiscountID SMALLINT
       SET @givenDiscountID = (SELECT DiscountID FROM
dbo.CompanyDiscounts WHERE RestaurantID = @restaurantID AND
BillingPeriod = @quartal)
       DECLARE @quartalOrdersValue MONEY
       SET @quartalOrdersValue = (SELECT
[dbo].[PastOrdersValueSinceTill](@companyID, DATEADD(DAY,-91,@date),
@date))
       IF (@quartalOrdersValue >= @priceRequirementToMeet)
       BEGIN
             BEGIN TRY
                   SET IDENTITY_INSERT dbo.CompanyDiscountHist ON;
                   INSERT INTO dbo.CompanyDiscountHist
                   (
                         CustomerID,
                          DiscountID,
                         Since,
                         TotalDiscount
                   )
                   VALUES
                   (   @companyID,
```

```
                        @givenDiscountID,      -- DiscountID - smallint
                        @date, -- Since - date
                        @discountToGive        -- TotalDiscount - real
                        )
            END TRY
            BEGIN CATCH
                  DECLARE @msg VARCHAR(250)
                  SET @msg = 'Wstawianie rabatu nie powiodło się. Error
message: ' + ERROR_MESSAGE()
                  ;THROW 52000,@msg,1
            END CATCH
      END
      ELSE
      BEGIN
            ;THROW 52000,'Klient nie spełnił wymagań, które pozwoliłyby
przyznać mu rabat kwartalny',1
      END
   END
END
GO
```

5. UseIndividualTemporalDiscount

Powoduje odznaczenie rabatu czasowego dla klienta indywidualnego jako zużyty. Uniemożliwia dalsze go używanie.

```
CREATE PROCEDURE [dbo].[UseIndividualTemporalDiscount]
    @restaurantID INT,
    @customerID INT,
    @currentDate DATETIME
AS
BEGIN
    DECLARE @discountID SMALLINT
    SET @discountID = (SELECT TOP 1 DiscountID FROM
[dbo].[GetCurrentValidTempIndividualDiscounts](@restaurantID,
@customerID, @currentDate))
    UPDATE dbo.IndividualDiscountHist SET UseDate = @currentDate
    WHERE DiscountID IN (SELECT TOP 1 DiscountID FROM
[dbo].[GetCurrentValidTempIndividualDiscounts](@restaurantID,
@customerID, @currentDate))
END
GO
```

6. AddRestriction

Dodaje obostrzenie epidemiczne do danego stolika na dany okres czasu.

```
CREATE PROCEDURE [dbo].[AddRestriction]
    @tableID INT,
    @from DATETIME,
    @till DATETIME,
    @seatsAvailable INT
AS
BEGIN
    BEGIN TRY
    IF NOT EXISTS (SELECT TableID FROM Tables WHERE TableID = @tableID)
    BEGIN
       ;THROW 52000,'Stolik nie istnieje',1
    END
    IF (@seatsAvailable < 0 AND @seatsAvailable < (SELECT SeatsNumber
FROM Tables WHERE TableID = @tableID))
    BEGIN
       ;THROW 52000,'Wprowadzono złą liczbę miejsc',1
    END
    INSERT INTO dbo.CovidRestrictions
    (
      TableID,
      Since,
      Until,
      SeatsAvailable
    )
    VALUES
    (   @tableID,       -- TableID - int
      @from, -- Since - datetime
      @till, -- Until - datetime
      @seatsAvailable         -- SeatsAvailable - smallint
      )
    END TRY
    BEGIN CATCH
       DECLARE @msg VARCHAR(250)
       SET @msg = 'Nie udało się dodać restrykcji. Error message: ' +
ERROR_MESSAGE()
       ;THROW 52000, @msg,1
    END CATCH
END
GO
```

7. AddDishToCurrentMenu

   Dodaje danie do aktualnego menu danej restauracji

```
CREATE PROCEDURE [dbo].[AddDishToCurrentMenu] (
      @restaurantID INT,
```

```
      @dishName NVARCHAR(50),
      @menuIn DATE = NULL
      )
AS
BEGIN

      DECLARE @dishID INT
      SET @dishID = (SELECT DishID FROM dbo.Menu
                           WHERE DishName = @dishName
                           AND RestaurantID = @restaurantID)

      IF @menuIn IS NULL
            SET @menuIn = GETDATE()

      IF (@dishID IS NULL)
      BEGIN
            ;THROW 52000,
            'Podane danie nie znajduje się w bazie dań restauracji,
wymagane jego wprowadzenie',1
      END

      IF EXISTS (
            SELECT * FROM dbo.GetMenuOfTheDay(@menuIn,
@restaurantID)
            WHERE DishID = @dishID
            )
            BEGIN
                  ; THROW 52000,
                  'Wprowadzone danie znajduje się już w menu
restauracji', 1
            END

      SET IDENTITY_INSERT dbo.MenuRegister ON
      INSERT INTO dbo.MenuRegister
      (
          DishID,
          MenuIn,
          MenuOut
      )
      VALUES
      (   @dishID,
          @menuIn,
          NULL
          )

END
```

```
GO
```

8. AddNewCompanyCustomer

Dodaje nowego klienta firmowego do bazy

```
CREATE PROCEDURE [dbo].[AddNewCompanyCustomer] (@restaurantID INT,
@nip NVARCHAR(50), @companyName NVARCHAR(50),
      @email NVARCHAR(50) ,@phone NVARCHAR(50), @cityName
NVARCHAR(50),   @countryName NVARCHAR(50),
      @address NVARCHAR(50), @postalCode NVARCHAR(50))
AS
      DECLARE @cityID INT;
      EXEC dbo.FindOrInsertCity @cityName,
                                @countryName,
                                @cityID = @cityID OUTPUT

      INSERT INTO dbo.Customers
      (
          RestaurantID,
          Email,
          Phone,
          CityID,
          Address,
          PostalCode
      )
      VALUES
      (   @restaurantID,
          @email,
          @phone,
          @cityID,
          @address,
          @postalCode
          )
      DECLARE @companyID INT
      SET @companyID = @@IDENTITY

      SET IDENTITY_INSERT dbo.CompanyCustomers ON

      INSERT INTO dbo.CompanyCustomers
      (
            CompanyID,
          CompanyName,
```

```
        NIP
    )
    VALUES
    (   @companyID,
        @companyName,
        @nip
        )


GO
```

9. AddNewDish

Dodaje nowe danie do puli dań danej restauracji

```
CREATE PROCEDURE [dbo].[AddNewDish] (@dishName NVARCHAR(50),
@restaurantID INT, @categoryName NVARCHAR(50), @catDescription
NTEXT, @price MONEY, @dishID INT OUTPUT)
AS
BEGIN
    DECLARE @categoryID INT
    EXEC dbo.FindOrInsertCategory
                @categoryName,
                @catDescription,
                @categoryID = @categoryID OUTPUT
    INSERT INTO dbo.Menu
    (
        DishName,
        RestaurantID,
        CategoryID,
        Price
    )
    VALUES
    (
        @dishName,
        @restaurantID,
        @categoryID,
        @price
        )
    SET @dishID = @@IDENTITY
END


GO
```

10. AddNewIndividualCustomer

Dodaje nowego klienta indywidualnego do bazy

```
CREATE PROCEDURE [dbo].[AddNewIndividualCustomer] (@restaurantID
INT,@email NVARCHAR(50) ,@phone NVARCHAR(50),
      @cityName NVARCHAR(50), @countryName NVARCHAR(50), @address
NVARCHAR(50), @postalCode NVARCHAR(50),
      @lastName NVARCHAR(50), @firstName NVARCHAR(50),
@companyName NVARCHAR(50) = NULL)
AS

      IF @companyName IS NOT NULL
            DECLARE @companyID INT
            SET @companyID = (SELECT CompanyID FROM
dbo.CompanyCustomers WHERE CompanyName = @companyName)
            IF @companyID IS NULL
                  THROW 52000, 'Podana firma nie istnieje',1

      DECLARE @cityID INT;
      EXEC dbo.FindOrInsertCity @cityName,
                                @countryName,
                                @cityID = @cityID OUTPUT
      INSERT INTO dbo.Customers
      (
          RestaurantID,
          Email,
          Phone,
          CityID,
          Address,
          PostalCode
      )
      VALUES
      (   @restaurantID,
          @email,
          @phone,
          @cityID,
          @address,
          @postalCode
          )
      DECLARE @customerID INT
      SET @customerID = @@IDENTITY
```

```
	SET IDENTITY_INSERT dbo.IndividualCustomers ON

	INSERT INTO dbo.IndividualCustomers
	(
		CustomerID,
	    LastName,
	    FirstName,
	    CompanyID
	)
	VALUES
	(   @customerID,
		@lastName,
	    @firstName,
	    @companyID
	    )


GO
```

11. DiscontinueOrPlaceProduct

Wycofuje dany produkt z magazynu restauracji.

```
CREATE PROCEDURE [dbo].[DiscontinueOrPlaceProduct]
    @productID INT,
    @discontinue BIT --1 - discontinue the product, 0 - place the
product back
AS
BEGIN
    IF (@discontinue = 1)
    BEGIN
       UPDATE dbo.Products SET Discontinued = 1 WHERE ProductID =
@productID
    END
    ELSE
    BEGIN
       UPDATE dbo.Products SET Discontinued = 0 WHERE ProductID =
@productID
    END
END
GO
```

12. DeleteDishFromCurrentMenu

Wycofuje danie z aktualnego menu danej restauracji

```
CREATE PROCEDURE [dbo].[DeleteDishFromCurrentMenu] (
      @restaurantID INT,
      @dishName NVARCHAR(50),
      @menuOut DATE = NULL
      )
AS
BEGIN
      DECLARE @dishID INT
            SET @dishID = (SELECT TOP 1 DishID FROM dbo.Menu
                                    WHERE DishName = @dishName
                                    AND RestaurantID = @restaurantID)

            IF @menuOut IS NULL
                  SET @menuOut = GETDATE()

            IF (@dishID IS NULL)
            BEGIN
                  ;THROW 52000, 'Podane danie nie znajduje się w
menu restauracji',1
            END

            UPDATE dbo.MenuRegister
            SET MenuOut = @menuOut
            WHERE DishID = @dishID
            AND (MenuOut IS NULL)
END


GO
```

13. FindOrInsertCategory

Wstawia nową kategorię dań, jeżeli taka nie widnieje w bazie

```
CREATE PROCEDURE [dbo].[FindOrInsertCategory]
      @categoryName VARCHAR(255),
      @description NTEXT,
      @categoryID INT OUTPUT
AS
BEGIN
BEGIN TRY
      BEGIN TRAN FIND_CATEGORY
            SET @categoryID = (SELECT CategoryID FROM dbo.Categories
                                    WHERE CategoryName =
```

```
@categoryName)
            IF (@categoryID IS NULL)
            BEGIN
                  INSERT INTO Categories(CategoryName, Description)
                  VALUES (@categoryName, @description);
                  SET @categoryID = @@IDENTITY;
            END
      COMMIT TRAN FIND_CATEGORY
END TRY
BEGIN CATCH
      ROLLBACK TRAN FIND_CATEGORY
      DECLARE @msg NVARCHAR(2048) = 'Błąd wyszukiwania kategorii: '
+
      CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
END CATCH
END
GO
```

14. FindOrInsertCity

Wstawia nowe miasto do tabeli Cities, jeżeli takie nie widnieje w bazie

```
 CREATE PROCEDURE [dbo].[FindOrInsertCity]
      @cityName VARCHAR(255),
      @countryName VARCHAR(255),
      @cityID INT OUTPUT
AS
BEGIN
      BEGIN TRY
            BEGIN TRAN FIND_CITY
                  SET @cityID = NULL
                  IF((@cityName IS NOT NULL AND @countryName IS
NULL) OR (@cityName IS NULL AND @countryName IS NOT NULL))
                        BEGIN
                              ;THROW 52000,'Nalezy podac nazwe
miasta i nazwe kraju albo zadne z nich ',1;
                        END
                  IF(@cityName IS NOT NULL AND @countryName IS NOT
NULL)
                        BEGIN
                              DECLARE @countryID INT
                              EXEC FindOrInsertCountry @countryName,
@countryID = @countryID OUT
```

```
                              SET @cityID = (SELECT TOP 1 cityID
FROM Cities WHERE (CityName = @cityName AND CountryID =
@countryID))
                              IF(@cityID IS NULL)
                                    BEGIN
                                          INSERT INTO
Cities(CityName,CountryID)VALUES(@cityName,@countryID);
                                          SET @cityID =@@IDENTITY;
                                    END
                        END
            COMMIT TRAN FIND_CITY
      END TRY
      BEGIN CATCH
            ROLLBACK TRAN FIND_CITY
            DECLARE @msg NVARCHAR(2048)='Błąd wyszukiwania
miasta:'+CHAR(13)+CHAR(10)+ ERROR_MESSAGE();
            THROW 52000,@msg,1;
      END CATCH
END
GO
```

15. FindOrInsertCountry

Wstawia nowe państwo do tabeli Countries, jeżeli takie nie widnieje w bazie

```
CREATE PROCEDURE [dbo].[FindOrInsertCountry]
      @countryName VARCHAR(255),
      @countryID INT OUTPUT
AS
BEGIN
BEGIN TRY
      BEGIN TRAN FIND_COUNTRY
            SET @countryID = (SELECT CountryID FROM dbo.Countries
                                    WHERE CountryName =
@countryName)
            IF (@countryID IS NULL)
            BEGIN
                  INSERT INTO Countries(CountryName)
                  VALUES (@countryName);
                  SET @countryID = @@IDENTITY;
            END
      COMMIT TRAN FIND_COUNTRY
END TRY
BEGIN CATCH
      ROLLBACK TRAN FIND_COUNTRY
```

```
        DECLARE @msg NVARCHAR(2048) = 'Błąd wyszukiwania kraju: ' +
        CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
END CATCH
END
GO
```

16. MakeIndividualReservation

Umożliwia klientowi indywidualnemu złożenie rezerwacji, przy jednoczesnym złożeniu zamówienia. Zwraca minimalną wartość zamówienia, przy której obsługa zaakceptuje i potwierdzi tę rezerwację (dot. sekcji “Wcześniejsza rezerwacja zamówienia/stolika” z Opisu Zadania).

```
CREATE PROCEDURE [dbo].[MakeIndividualReservation]
    @restaurantID INT,
    @customerID INT,
    @from DATETIME,
    @paymentOption BIT, --1 - payment straight away, before finalizing
the reservation; 0 - payment after finalizing the reservation
    @orderDate DATETIME,
    @minOrderValue MONEY OUTPUT
AS
BEGIN
    BEGIN TRAN Make_Individual_Reservation
       BEGIN TRY
             DECLARE @ordersCount INT
             SET @ordersCount = (SELECT [dbo].[PastOrdersCount]
(@customerID,'2017-01-01', @orderDate))
             IF (@ordersCount >=5)
             BEGIN
                   SET @minOrderValue = 50
             END
             ELSE
             BEGIN
                   SET @minOrderValue = 200
             END
             SET IDENTITY_INSERT dbo.IndividualReservations ON;
             DECLARE @reservationID INT
             SET @reservationID = ((SELECT TOP 1 ReservationID FROM
dbo.IndividualReservations ORDER BY ReservationID DESC) + 1)
             EXECUTE dbo.AddOnSiteOrder @restaurantID, @customerID,
@orderDate, @from, @paymentOption, NULL
             DECLARE @orderID INT
```

```
            SET @orderID = (SELECT TOP 1 OrderID FROM dbo.OnSiteOrders
WHERE CustomerID = @customerID AND OrderDate = @orderDate AND
ExecutionDate = @from ORDER BY OrderID DESC)
            INSERT INTO dbo.IndividualReservations
            (
                  ReservationID,
                  OrderID,
                  CustomerID
            )
            VALUES
            (
                  @reservationID,
                  @orderID, -- OrderID - int
                  @customerID  -- CustomerID - int
             )
       COMMIT TRAN Make_Individual_Reservation
       END TRY
       BEGIN CATCH
            ROLLBACK TRAN Make_Individual_Reservation
            DECLARE @msg VARCHAR(250)
            SET @msg = 'Błąd w dodawaniu rezerwacji. Error message: ' +
ERROR_MESSAGE()
            ;THROW 52000, @msg,1
       END CATCH
    RETURN @minOrderValue
END
GO
```

17. MakeCompanyReservation

Umożliwia klientowi - firmie złożenie rezerwacji stolika dla firmy w opcji na firmę.

```
CREATE PROCEDURE [dbo].[MakeCompanyReservation]
    @restaurantID INT,
    @companyID INT,
    @from DATETIME,
    @till DATETIME,
    @guestsNumber SMALLINT,
    @givenTableID INT OUTPUT
AS
BEGIN
    DECLARE @designatedTable INT
    SET @designatedTable = 0
    DECLARE @tableID INT
    DECLARE @seatsNumber SMALLINT
```

```
    DECLARE @seatsAvailable SMALLINT
    DECLARE @Since DATETIME
    DECLARE @Until DATETIME
    DECLARE myCursor CURSOR FOR
    SELECT TableID, SeatsNumber, SeatsAvailable, Since, Until FROM
dbo.GetFreeTablesWithCorrespondingCovidRestrictions(@restaurantID,
@guestsNumber, @from, @till)
    ORDER BY SeatsNumber
    OPEN myCursor
    FETCH NEXT FROM myCursor INTO @tableID, @seatsNumber,
@seatsAvailable, @Since, @Until
    WHILE @@FETCH_STATUS = 0
    BEGIN
       IF ((@from NOT BETWEEN @Since AND @Until) AND (@till NOT BETWEEN
@Since AND @Until))
       BEGIN
             SET @designatedTable = @tableID
             BREAK
       END
       ELSE IF (@seatsAvailable >= @guestsNumber)
       BEGIN
             SET @designatedTable = @tableID
             BREAK
       END
       FETCH NEXT FROM myCursor INTO @tableID, @seatsNumber,
@seatsAvailable, @Since, @Until
    END
    PRINT @designatedTable
    SET @givenTableID = @designatedTable
    CLOSE myCursor
    DEALLOCATE myCursor

    IF (@designatedTable = 0)
    BEGIN
       ;THROW 52000,'Brak wolnych stolików na dany termin. Nie można
zrealizować rezerwacji',1
    END
    ELSE
    BEGIN
       BEGIN TRY
             INSERT INTO dbo.CompanyReservations
             (
                  OrderID,
                  CompanyID
             )
             VALUES
```

```
            (   NULL,
                @companyID
                )
      END TRY
      BEGIN CATCH
            DECLARE @msg NVARCHAR(250)
            SET @msg = 'Nie udało się dokonać rezerwacji' +
ERROR_MESSAGE()
            ;THROW 52000,@msg,1
      END CATCH
      BEGIN TRY
            INSERT INTO dbo.TableReservations
            (
                TableID,
                ReservationID,
                SinceWhenReserved,
                UntilWhenReserved,
                CustomerNumber
            )
            VALUES
            (   @designatedTable,        -- TableID - int
                SCOPE_IDENTITY(),        -- ReservationID - int
                @from, -- SinceWhenReserved - datetime
                @till, -- UntilWhenReserved - datetime
                @guestsNumber            -- CustomerNumber - smallint
                )
      END TRY
      BEGIN CATCH
            SET @msg = 'Rezerwacja stolika nie powiodła się' +
ERROR_MESSAGE()
            ;THROW 52000,@msg,1
       END CATCH
     END
END
GO
```

18. ConfirmOrAnullIndividualReservationAsAnEmployee

Umożliwia pracownikowi restauracji potwierdzenie lub anulowanie rezerwacji złożonej przez klienta indywidualnego. Procedura potwierdza rejestrację, jeśli klient spełnił wymagania odpowiedniej minimalnej kwoty zamówienia. Zwracany jest wtedy numer stolika. W przeciwnym przypadku rezerwacja zostaje anulowana.

```
CREATE PROCEDURE [dbo].[ConfirmOrAnullIndividualReservationAsAnEmployee]
    @restaurantID INT,
```

```
    @reservationID INT,
    @requiredOrderValue MONEY,
    @from DATETIME,
    @till DATETIME,
    @guestsNumber SMALLINT,
    @givenTableID INT OUTPUT
AS
BEGIN
    DECLARE @customerID INT
    SET @customerID = (SELECT CustomerID FROM dbo.IndividualReservations
WHERE ReservationID = @reservationID)
    DECLARE @orderID INT
    SET @orderID = (SELECT TOP 1 OrderID FROM dbo.IndividualReservations
WHERE ReservationID = @reservationID)
    DECLARE @actualOrderValue MONEY
    SET @actualOrderValue = (SELECT SUM(Quantity * UnitPrice) FROM
dbo.OrderDetails
    WHERE OrderID = @orderID)
    IF (@actualOrderValue < @requiredOrderValue)
    BEGIN
       DELETE FROM dbo.IndividualReservations WHERE ReservationID =
@reservationID
       EXEC AnullOrderAsAnEmployee @orderID, 0, NULL
       ;THROW 52000,'Rezerwacja klienta jest za kwotę niższą niż
wymagana. Rezerwacja ta została anulowana',1
    END
    ELSE
    BEGIN
       DECLARE @designatedTable INT
       SET @designatedTable = 0
       DECLARE @tableID INT
       DECLARE @seatsNumber SMALLINT
       DECLARE @seatsAvailable SMALLINT
       DECLARE @Since DATETIME
       DECLARE @Until DATETIME
       DECLARE myCursor CURSOR FOR
       SELECT TableID, SeatsNumber, SeatsAvailable, Since, Until FROM
dbo.GetFreeTablesWithCorrespondingCovidRestrictions(@restaurantID,
@guestsNumber, @from, @till)
       ORDER BY SeatsNumber
       OPEN myCursor
       FETCH NEXT FROM myCursor INTO @tableID, @seatsNumber,
@seatsAvailable, @Since, @Until
       WHILE @@FETCH_STATUS = 0
       BEGIN
             IF ((@from NOT BETWEEN @Since AND @Until) AND (@till NOT
```

```
BETWEEN @Since AND @Until))
            BEGIN
                  SET @designatedTable = @tableID
                  BREAK
            END
            ELSE IF (@seatsAvailable >= @guestsNumber)
            BEGIN
                  SET @designatedTable = @tableID
                  BREAK
            END
            FETCH NEXT FROM myCursor INTO @tableID, @seatsNumber,
@seatsAvailable, @Since, @Until
      END
      PRINT @designatedTable
      SET @givenTableID = @designatedTable
      CLOSE myCursor
      DEALLOCATE myCursor
   END

   IF (@designatedTable = 0)
   BEGIN
      DELETE FROM dbo.IndividualReservations WHERE ReservationID =
@reservationID
      EXEC AnullOrderAsAnEmployee @orderID, 0, NULL
      ;THROW 52000,'Brak wolnych stolików na dany termin. Rezerwacja
została anulowana',1
   END
   ELSE
   BEGIN
      BEGIN TRY
            INSERT INTO dbo.TableReservations
            (
                  TableID,
                  ReservationID,
                  SinceWhenReserved,
                  UntilWhenReserved,
                  CustomerNumber
            )
            VALUES
            (   @designatedTable,        -- TableID - int
                  @reservationID,         -- ReservationID - int
                  @from, -- SinceWhenReserved - datetime
                  @till, -- UntilWhenReserved - datetime
                  @guestsNumber           -- CustomerNumber - smallint
                  )
      END TRY
```

```
        BEGIN CATCH
              DECLARE @msg NVARCHAR(250)
              SET @msg = 'Rezerwacja stolika nie powiodła się. Error
message: ' + ERROR_MESSAGE()
              ;THROW 52000,@msg,1
        END CATCH
       END
END
GO
```

AnullOrderAsAnEmployee

Umożliwia pracownikowi anulowanie zamówienia.

```
CREATE PROCEDURE [dbo].[AnullOrderAsAnEmployee]
    @orderID INT,
    @isTakeAway BIT,
    @orderStatus CHAR OUTPUT
AS
BEGIN
    IF (@isTakeAway = 0)
    BEGIN
       UPDATE dbo.OnSiteOrders SET Status = 'A'
       WHERE OrderID = @orderID
    END
    ELSE
    BEGIN
       UPDATE dbo.TakeAwayOrders SET Status = 'A'
       WHERE OrderID = @orderID
    END
    SET @orderStatus = 'A'
END
GO
```

AddPaymentToAnOrder

Umożliwia pracownikowi dodanie płatności za dane zamówienie.

```
CREATE PROCEDURE [dbo].[AddPaymentToAnOrder]
    @orderID INT,
    @isTakeAway BIT,
    @orderStatus CHAR OUTPUT
AS
BEGIN
    IF (@isTakeAway = 0)
    BEGIN
```

```
        UPDATE dbo.OnSiteOrders SET Status = 'P' WHERE OrderID = @orderID
    END
    ELSE
    BEGIN
        UPDATE dbo.TakeAwayOrders SET Status = 'P' WHERE OrderID =
@orderID
    END
    SET @orderStatus = 'P'
END
GO
```

ConfirmOrderAsAnEmployee

Umożliwia pracownikowi restauracji potwierdzenie zamówienia.

```
CREATE PROCEDURE [dbo].[ConfirmOrderAsAnEmployee]
    @employeeID INT,
    @orderID INT,
    @isTakeAway BIT,
    @orderStatus CHAR OUTPUT
AS
BEGIN
    IF (@isTakeAway = 0)
    BEGIN
        UPDATE dbo.OnSiteOrders SET Status = 'C',
                                                EmployeeID = @employeeID
        WHERE OrderID = @orderID
    END
    ELSE
    BEGIN
        UPDATE dbo.TakeAwayOrders SET Status = 'C',
                                                  EmployeeID =
@employeeID
        WHERE OrderID = @orderID
    END
    SET @orderStatus = 'C'
END

GO
```

AddDishToAnOrder

Umożliwia dodanie dania do zamówienia.

```
CREATE PROCEDURE [dbo].[AddDishToAnOrder]
    @dishName NVARCHAR(50),
    @quantity SMALLINT,
    @orderID INT,
    @restaurantID INT,
    @currentDate DATE
AS
BEGIN
    DECLARE @dishID INT
    SET @dishID = (SELECT DishID FROM dbo.GetMenuOfTheDay(@currentDate,
@restaurantID)
                                WHERE DishName = @dishName)
    DECLARE @unitPrice MONEY
    SET @unitPrice = (SELECT Price FROM
dbo.GetMenuOfTheDay(@currentDate, @restaurantID)
                                WHERE DishName = @dishName)
    BEGIN TRY
       INSERT INTO dbo.OrderDetails
       (
             OrderID,
             DishID,
             Quantity,
             UnitPrice
       )
       VALUES
       (   @orderID,   -- OrderID - int
             @dishID,   -- DishID - int
             @quantity,   -- Quantity - smallint
             @unitPrice -- UnitPrice - money
             )
    END TRY
    BEGIN CATCH
       ;THROW 52000,'Nie udało się dodać dania do zamówienia. Error
message: ',1
    END CATCH
END
GO
```

AddOnSiteOrder

Wstawia zamówienie na miejscu do rejestru, z wybraną opcją płatności: od razu bądź po realizacji zamówienia.

```
CREATE PROCEDURE [dbo].[AddOnSiteOrder]
    @restaurantID INT,
    @customerID INT,
    @orderDate DATETIME,
```

```
    @forWhen DATETIME,
    @paymentOption BIT, --1 - payment straight away; 0 - payment to be
confirmed later
    @orderID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    SET IDENTITY_INSERT dbo.OnSiteOrders OFF;
    DECLARE @designateEmployee INT
    SET @designateEmployee = (SELECT TOP 1 EmployeeID FROM dbo.Employees
INNER JOIN dbo.Managers
ON Managers.ManagerID = Employees.ManagerID
INNER JOIN dbo.Restaurants ON Restaurants.ManagerID = Managers.ManagerID
WHERE RestaurantID = @restaurantID)
    IF (@paymentOption = 1)
    BEGIN
    INSERT INTO dbo.OnSiteOrders
    (
      CustomerID,
      EmployeeID,
      OrderDate,
      ExecutionDate,
      Status
    )
    VALUES
    (
       @customerID,
      @designateEmployee,
      @orderDate,
      @forWhen,
      'P'
      )
    SET @orderID = SCOPE_IDENTITY()
    END
    ELSE
    BEGIN
            INSERT INTO dbo.OnSiteOrders
       (
            CustomerID,
            EmployeeID,
            OrderDate,
            ExecutionDate,
            Status
       )
       VALUES
       (
```

```
            @customerID,
            @designateEmployee,
            @orderDate,
            @forWhen,
            'N'
            )
    SET @orderID = SCOPE_IDENTITY()
    END
END
GO
```

AddTakeAwayOrder

Wstawia zamówienie na wynos do rejestru, z wybraną opcją płatności: od razu bądź po realizacji zamówienia.

```
CREATE PROCEDURE [dbo].[AddTakeAwayOrder]
    @restaurantID INT,
    @customerID INT,
    @orderDate DATETIME,
    @forWhen DATETIME,
    @paymentOption BIT, --1 - payment straight away; 0 - payment to be
confirmed later
    @orderID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    SET IDENTITY_INSERT dbo.TakeAwayOrders OFF;
    DECLARE @designateEmployee INT
    SET @designateEmployee = (SELECT TOP 1 EmployeeID FROM dbo.Employees
INNER JOIN dbo.Managers
                                        ON Managers.ManagerID =
Employees.ManagerID
                                        INNER JOIN dbo.Restaurants ON
Restaurants.ManagerID = Managers.ManagerID
                                        WHERE RestaurantID =
@restaurantID)
    IF (@paymentOption = 1)
    BEGIN
    INSERT INTO dbo.TakeAwayOrders
    (
      CustomerID,
      EmployeeID,
      OrderDate,
      ExecutionDate,
```

```
      Status
    )
    VALUES
    (   @customerID,          -- CustomerID - int
      @designateEmployee,           -- EmployeeID - int
      @orderDate, -- OrderDate - datetime
      @forWhen, -- ExecutionDate - datetime
      'P'         -- Status - char(1)
      )
    SET @orderID = SCOPE_IDENTITY()
    END
    ELSE
    BEGIN
       INSERT INTO dbo.TakeAwayOrders
       (
            CustomerID,
            EmployeeID,
            OrderDate,
            ExecutionDate,
            Status
       )
       VALUES
       (   @customerID,       -- CustomerID - int
             @designateEmployee,          -- EmployeeID - int
             @orderDate, -- OrderDate - datetime
             @forWhen, -- ExecutionDate - datetime
             'N'         -- Status - char(1)
             )
    SET @orderID = SCOPE_IDENTITY()
    END
END
GO
```

7. Triggery

1. CheckIfAddingADishIsLegal

   Sprawdza, czy danie dodane do menu danej restauracji zostało poprzednio zdjęte więcej niż miesiąc temu (wedle warunku, że pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc).

```
CREATE TRIGGER [dbo].[CheckIfAddingADishIsLegal]
ON [dbo].[MenuRegister]
AFTER INSERT
AS
```

```
BEGIN
	DECLARE @dishID INT = (SELECT DishID FROM Inserted)

	IF EXISTS (
		SELECT * FROM dbo.MenuRegister
		WHERE DishID = @dishID AND
		MenuOut IS NOT NULL AND
		DATEDIFF(MONTH, MenuOut, GETDATE()) <= 1
		)
	BEGIN
		; THROW 50200,
		'Danie zostało zdjęte z Menu mniej niż miesiąc temu, nie
można go dodać', 1
		ROLLBACK TRANSACTION
	END
END
GO

ALTER TABLE [dbo].[MenuRegister] ENABLE TRIGGER
[CheckIfAddingADishIsLegal]
GO
```

2. NotEnoughProductUnitsInStock

Przy wprowadzaniu szczegółów zamówienia (danie oraz liczba jego zamówionych sztuk) sprawdza, czy starcza składników na realizację dania w danej liczbie sztuk.

```
CREATE TRIGGER [dbo].[NotEnoughtPoductUnitsInStock]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @dishID INT
    SET @dishID = (SELECT DishID FROM Inserted)
    DECLARE @restaurantID INT
    SET @restaurantID = (SELECT RestaurantID FROM Menu WHERE DishID
= @dishID)
    DECLARE @orderedDishQuantity SMALLINT
    SET @orderedDishQuantity = (SELECT Quantity FROM Inserted)
    DECLARE @productUnitsInStock INT
    SET @productUnitsInStock = (SELECT UnitsInStock FROM dbo.Menu
INNER JOIN dbo.DishDetails
    ON DishDetails.DishID = Menu.DishID
    INNER JOIN dbo.Products ON Products.ProductID =
DishDetails.ProductID
```

```
    WHERE Menu.RestaurantID = @restaurantID AND
dbo.DishDetails.DishID = @dishID
    GROUP BY DishDetails.DishID,
      Products.ProductID,
      Quantity,
      UnitsInStock)
    DECLARE @totalNeededProductsQuantity INT
    SET @totalNeededProductsQuantity = ((SELECT Quantity FROM
dbo.Menu INNER JOIN dbo.DishDetails
    ON DishDetails.DishID = Menu.DishID
    INNER JOIN dbo.Products ON Products.ProductID =
DishDetails.ProductID
    WHERE Menu.RestaurantID = @restaurantID AND
dbo.DishDetails.DishID = @dishID
    GROUP BY DishDetails.DishID,
      Products.ProductID,
      Quantity,
      UnitsInStock) * @orderedDishQuantity)
    IF (@totalNeededProductsQuantity > @productUnitsInStock)
    BEGIN
       RAISERROR('Nie starcza półproduktów do realizacji
zamówienia',16,1)
       ROLLBACK TRANSACTION
    END
END

GO

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

3. ReservationForAtLeastTwoPeople

Sprawdza, czy dokonywana rezerwacja stolika jest dla co najmniej dwóch osób.

```
CREATE TRIGGER [dbo].[ReservationForAtLeastTwoPeople]
ON [dbo].[TableReservations]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    IF (SELECT CustomerNumber FROM Inserted) < 2
    BEGIN
       RAISERROR('Rezerwacji stolika można dokonać tylko dla więcej
niż 2 osób',16,1)
       ROLLBACK TRANSACTION
```

```
    END
END
GO

ALTER TABLE [dbo].[TableReservations] ENABLE TRIGGER
[ReservationForAtLeastTwoPeople]
GO
```

4. SeaFoodOrder

Sprawdza, czy zamówienie zawierające owoce morza spełnia warunki, tj., czy zamówienie to jest na czwartek, piątek lub sobotę, a także, czy zostało złożone z odpowiednim wyprzedzeniem (maksymalnie do poniedziałku poprzedzającego zamówienie)

```
CREATE TRIGGER [dbo].[SeaFoodOrder]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
      SET NOCOUNT ON
      DECLARE @orderID INT
      SET @orderID = (SELECT OrderID FROM Inserted)
      DECLARE @dishID INT
      SET @dishID = (SELECT DishID FROM Inserted)
      DECLARE @orderDate DATE
      SET @orderDate = (
            SELECT OrderDate FROM dbo.OnSiteOrders
            WHERE OrderID = @orderID
            )
      DECLARE @executionDate DATE
      SET @executionDate = (
            SELECT ExecutionDate FROM dbo.OnSiteOrders
            WHERE OrderID = @orderID
            )
      IF @executionDate IS NULL
      BEGIN
            SET @executionDate = (
                  SELECT ExecutionDate FROM dbo.TakeAwayOrders
                  WHERE OrderID = @orderID
                  )
            SET @orderDate = (
                  SELECT OrderDate FROM dbo.TakeAwayOrders
                  WHERE OrderID = @orderID
```

```
                  )
      END

      DECLARE @categoryName NVARCHAR(50)
      SET @categoryName = (
            SELECT CategoryName FROM dbo.Menu
            INNER JOIN dbo.Categories
            ON Categories.CategoryID = Menu.CategoryID
            WHERE DishID = @dishID
            )

      IF @categoryName LIKE 'Seafood' AND
            (DATEPART(dw, @executionDate) NOT IN (5,6,7) OR
            DATEDIFF(DAY, @orderDate, @executionDate) < 3)
      BEGIN
            ; THROW 50200,
            'Zamówienie dania z owocami morza nie jest możliwe,
ponieważ nie spełnia ono warunków ', 1
            ROLLBACK TRANSACTION
      END
END
GO

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

8. Generator danych

W celu wygenerowania danych do bazy użyłyśmy generatora SQL Data Generator firmy Red Gate: SQL Data Generator - Data Generator For MS SQL Server Databases . Wygenerowane dane odpowiadają okresowi około trzech lat użytkowania bazy danych. Generator ten sprawdził się dobrze, jedynym problemem wynikającym z jego losowości było przyznanie części klientów indywidualnych tych samych wartości klucza obcego *CustomerID* w tabeli IndividualCustomers, co wartości klucza obcego *CompanyID* w tabeli CompanyCustomers oraz przyznanie dodatkowej puli kluczy głównych *CustomerID* w tabeli Customers, które nie były przeznaczone ani dla klientów indywidualnych, ani dla firmowych. Konieczne było zatem usunięcie tych wartości, by zapewnić zgodność z warunkami integralności i tym samym spójność bazy.

```
DELETE TARGET
FROM dbo.Customers AS TARGET
WHERE (TARGET.CustomerID NOT IN (SELECT CustomerID FROM
dbo.IndividualCustomers)
      AND TARGET.CustomerID NOT IN (SELECT CompanyID FROM
```

```
dbo.CompanyCustomers))


DELETE TARGET
FROM dbo.IndividualCustomers AS TARGET
INNER JOIN dbo.CompanyCustomers ON CompanyCustomers.CompanyID =
TARGET.CustomerID
```

9. Role w systemie

1. Administrator
    a. dodanie nowej restauracji
2. Menedżer restauracji
    a. Dodawanie/usuwanie dań w Menu, aktualizacja Menu danej restauracji
    b. Generowanie statystyk dla danej restauracji, w tym wyszczególnienie najaktywniejszych klientów
    c. Wprowadzanie informacji o obostrzeniach epidemicznych
    d. Wgląd do stanu magazynu
3. Pracownik restauracji
    a. Wgląd do stanu magazynu
    b. Akceptacja zamówień wraz z przydzieleniem do ich realizacji konkretnych pracowników
    c. Wprowadzanie informacji o opłaceniu zamówień
    d. Anulowanie zamówień
    e. Dodawanie Klientów Indywidualnych
    f. Dodawanie Klientów Firmowych
    g. Dodawanie pracowników Klientów Firmowych jako Klientów Indywidualnych
    h. Sprawdzanie aktualności Menu, korzystając z funkcji systemowej (rekomendowane jest, aby w pracownik raz dziennie uruchomił funkcję sprawdzającą aktualność Menu w swojej restauracji)
    i. Aktualizacja Menu: wprowadzanie i zdejmowanie pozycji w Menu
    j. Sprawdzanie aktualnych rabatów dla Klientów Indywidualnych oraz Klientów Firmowych
    k. Sprawdzanie historii przyznanych rabatów dla Klientów Indywidualnych oraz dla Klientów Firmowych
    l. Generowanie faktury dla Klienta Firmowego
4. Klient Indywidualny
    a. Złożenie zamówienia
    b. Złożenie rezerwacji, przy jednoczesnym złożeniu zamówienia oraz wybranie możliwej opcji płatności
    c. Sprawdzanie aktualnych rabatów
    d. Sprawdzanie historii swoich zamówień
5. Klient Firmowy
    a. Złożenie zamówienia

b. Złożenie rezerwacji na firmę
c. Złożenie imiennej rezerwacji z ramienia firmy
d. Sprawdzanie aktualnych rabatów
e. Sprawdzanie historii swoich zamówień

6. Funkcje systemowe
    a. Obsługa systemu rabatów
    b. Wystawianie rabatów dla danego klienta po uprzednim sprawdzeniu spełnienia warunków ich przyznania
    c. Kontrola nad aktualnością Menu
    d. Kontrola nad zasadami związanymi z owocami morza
    e. Kontrola nad stolikami: ich dostępnością ze względu na rezerwacje i aktualne obostrzenia epidemiczne
    f. Kontrola stanu magazynu: automatyczne uaktualnianie stanu półproduktów w magazynie, wynikające ze składanych zamówień
    g. Obliczanie wartości zamówień