Autorzy:

Bartłomiej Kiszka

Kamil Kurowski

Prowadzący:

dr. inż. Anna Zygmunt

PODSTAWY BAZ DANYCH |

PROJEKT

KATEDRA INFORMATYKI WIEIT

AGH W KRAKOWIE

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

Styczeń 2020 r.

0.Analiza wymagań

0.1 Opis

Celem projektu jest stworzenie bazy danych, wspomagającej działalność firmy gastronomicznej świadczącej usługi dla klientów indywidualnych oraz firm.

0.2 Użytkownicy

1. Administrator systemu
2. Pracownik firmy świadczącej usługi gastronomiczne
3. Klient - osoba indywidualna
4. Klient – firma

0.3 Struktura bazy danych:

1. Oferta restauracji składa się z konkretnych dań [MenuItem], będą one podstawą tworzenia Menu a także będą elementami zamówienia
    a. każde danie ma swój unikalny identyfikator (ItemID), swoją nazwę (Name) oraz koszt dania (Cost)
    b. ponadto każde z dań posiada identyfikator kategorii do jakiej należy (CategoryID)

2. Dania podzielone są na kategorie [Category]
    a. Kategoria to unikalny identyfikator kategorii (CategoryID) oraz nazwa kategorii (CategoryName)

3. Dania wytwarza są ze składników - półproduktów [Ingredients]
    a. Każdy składnik zawiera swój unikalny identyfikator (IngredientID) oraz nazwę (Name)
    b. z uwagi na różnorodność składników przechowujemy informacje o tym w jakiej formie przetrzymujemy półprodukty (Unit) np.: dla wody będzie to litr a dla mąki kilogram.

    c. wyróżniamy 2 informacje o ilości danych składników - (UnitsInStock) informuje nas ile aktualnie jednostek znajduje się w magazynie (stan rzeczywisty magazynu), natomiast (UnitsReseved) to ilość jednostek jaka będzie potrzebna do realizacji zamówień już złożonych, pozwala to uniknąć sytuacji gdy nagle zabraknie składników na zamówienia złożone wcześniej oraz sytuacji gdy stan magazynu w bazie danych różni się od stanu fizycznego
    d. naturalnie nie możemy dopuścić do sytuacji gdy UnitsReseved > UnitsInStock, istnieje jednak możliwość wystąpienia takiego przypadku , np. gdy zamawiamy danie należące do kategorii “Owoce morza”, wtedy (UnitsReseved) będzie informacją ile podproduktów zamówić. Informacja o tym czy dany składnik może posiadać taką opcję znajduje się w pole (OnRequest)

4. Informacja o tym jakie półprodukty składają się na konkretnie danie znajduję się [Recipe]
    a. przechowujemy informację, że dla dania (ItemID) potrzeba półproduktu (IngredientID) w ilości (Amount)

5. Oferta restauracji w danym dniu to [Menu]
    a. do każdej pozycji Menu (PositionID) przyporządkowujemy jedno z dań (ItemID)
    b. zapisujemy datę (BeginDate) w jakiej konkretne danie zaczęło obowiązywać w menu

6. W celu spełnienia wymagań dot. ustalania Menu (np. pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc) zapisujemy historie Menu (tj. dań które występowały w menu w określonych interwałach czasowych) na przestrzeni czasu [MenuHistory]
    a. każdy element tabeli ma unikalny identyfikator (HistoryID)
    b. informacje o tym jakie danie to było (ItemID)
    c. datę pojawienia się w Menu (DateBegin) oraz datę zniknięcia z menu (DateEnd)

7. Zamówienia złożone w restauracji są przechowywane w [Orders]
    a. Każde zamówienie ma unikalny identyfikator (OrderID)
    b. Dodatkowo przechowywany jest identyfikator zamawiającego (CustomerID)
    c. Samo zamówienie przechowuję również datę złożenia (OrderDate), oraz datę zrealizowania (RealizationDate)
    d. Zamówienia realizowane być mogą na miejscu, lub na wynos (Type)
    e. Płatność za zamówienia może być dokonana z góry, po wykonaniu usługi, lub dla firm w formie faktury (PaymentType)

f. Zamówienie ma również swoją wartość pieniężną (Sum), oraz czas w którym uiszczono opłatę (PaymentDate)

8. Szczegóły zamówień są przechowywane w specjalnej tablicy [Order Details]
    a. Każdy szczegół zamówienia posiada odnośnik do rodzimego zamówienia (OdrerID), oraz do produktu zamawianego (ItemID), te wartości tworzą razem unikatową parę. Dodatkowo określona jest ilość zamawianego produktu (Quantity)

9. Przechowujemy informacje o rezerwacjach indywidualnych [ReservationsIndywidual], oraz o rezerwacjach firm [ReservationCompany]
    a. Dla klientów indywidualnych:
        i. Przechowujemy numer ich zamówienia (OrderID), numer rezerwacji (ReservationID), ramy czasowe rezerwacji (StartTime) (EndTime), oraz numer stołu (TableID)
    b. Dla klientów firmowych
        i. przechowywane są informacje o Rezerwacji (ReservationID), przyporządkowaniu klienta (CustomerID), nieobowiązkowe jest dokonanie wcześniejszego zamówienia (OrderID). Oprócz tego pamiętamy numer stolika (TableID), oraz ramy czasowe rezerwacji (StartTime) (EndTime)

10. Informacje o stolikach [Tables]
    a. Każdy stolik ma unikalny numer (TableID), oraz określoną pojemność (Places)

11. O restrykcjach informuje [Restrictions]
    a. Tam każdemu stolikowi (TableID) od określonej daty(BeginDate) przyporządkowywane jest ograniczenie miejsc(PlacesAvilable)

12. Szczegóły rezerwacji przechowujemy w [Reservation Individual Details] oraz [Reservation Company Details]
    a. Przyporządkowuje ona rezerwacji (ReservationID) numer stolika (TableID), identyfikator osoby na którą jest rezerwacja (EmployeeID/CustomerID), oraz ilością miejsc do zarezerwowania(People)

13. Dane o klientach [Customers]
    a. Każdemu klientowi przysługuje unikatowy identyfikator [CustomerID]
    b. Klienta opisuje również jego adres (Adress) oraz kontaktowy numer telefonu (PhoneNumber)

c. Dla porządku klienci są podzieleni względem tego czy reprezentują osobę fizyczną, czy firmę.

12. Informacje o firmach będących klientami [CompanyCustomer]
    a. Każdy z nich ma unikalny identyfikator ze zbiorczej tablicy z klientami (CustomerID)
    b. Firmę opisuje dodatkowo jej nazwa (CompanyName), oraz numer identyfikacji podatkowej (NIP)

13. Informacje o klientach indywidualnych [IndividualCustomer]
    a. Każdy z nich ma unikalny identyfikator ze zbiorczej tablicy z klientami (CustomerID)
    b. Dodatkowo opisujemy klientów indywidualnych ich imieniem (FirstName) i nazwiskiem (LastName)

14. Informacji o pracownikach danych firm udziela [CompanyEmployees]
    a. Przyporządkowuje ona pracowników(EmployeeID) do firm w jakich pracują (CompanyID)

15. Informacje o rabatach uwzględnianych przez naszą firmę dla każdego z klientów [CustomerDiscounts]
    a. zawiera ID klienta do którego został przypisany rabatach
    b. zawiera ID rabaty który został przypisany
    c. date początku oraz końca (wygaśnięcia)
    d. wartość zniżki

16. Typy rabatów oraz przypisane do nich wartości
    a. IndyvidualDiscountOnce - pojedyczne rabaty dla klientów indywividualnych
    b. IndyviudalDiscountsConst - stałe, kumulujące się rabaty dla klientów indywidualnych
    c. CompanyDiscountQuarter - kwartalne rabaty dla firm
    d. CompanyDiscountMOnthly - miesięczne rabaty dla firm

4. Funkcje bazy danych

1. Funkcje dostępne dla administratora bazy danych
    a. dodawanie/usuwanie pracowników, edycja bazy danych
2. Wybieranie Menu
    a. gdy pracownik będzie chciał wygenerować nowe Menu (z co najmniej dziennym wyprzedzeniem) baza danych zwróci mu elementy [MenuItem] które mogą się w tym czasie znaleźć w Menu

b. wybieranie elementów odbywa się na podstawie [MenuHistory] i [Menu] oraz założeń jakie zostały podstawione tj.
    - pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc
    - co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie.
c. element który trafia do Menu jest zapisywany z (DateBegin)
d. element który zostaje zdjęty z Menu trafia do MenuHistory z (DateBegin) tj. datą w której pojawił się w menu oraz z datą zdjęcia (DateEnd)
e. element może być zdjęty z menu również w przypadku wyczerpania się półproduktów (o czym pracownik restauracji zostanie poinformowany)
f. pracownik może również dodać zupełnie nowy element do [MenuItem]

3. Składanie zamówienia
    a. klient może złożyć zamówienie na dowolną pozycję (i w określonej ilości) z obecnego Menu
    b. jeśli klient składa zamówienie po raz pierwszy pracownik dodaje go do bazy danych
    c. klient składając zamówienie musi wyspecyfikować dane zamówienia
        - Typ zamówienia (Type) - na miejscu lub na wynos
        - Typ płatności (PaymentType) - z góry, przy odbiorze, oraz w przypadku firm jako faktury
        - Datę na które klient składa zamówienie (mając na myśli datę oraz godzinę) - obowiązuje zasada że zamówienie może zostać złożone z max. tygodniowym wyprzedzeniem, w przypadku zamówienia na miejscu jest to aktualna data i godzina
    d. W przypadku zamawiania owoców morza następuje weryfikacja czy zamówienie jest składane w odpowiedni dzień tygodnia (czwartek/piątek/sobota) i na odpowiednią datę, tj. na datę poprzedzającą kolejny poniedziałek
    e. Klient może zarezerwować konkretny stolik na konkretny interwał czasu, jeśli jest on wtedy wolny oraz jeśli spełnił założenia:
        - minimalna wartości zamówienia 50 zł jeślli dokonał wcześniej co najmniej 5 zamówień
        - w przeciwnym przypadku zamówienie na kwotę co najmniej 200 zł
        - w przypadku firm możliwość złożenia zamówienia jako firma lub w imieniu konkretnego pracownika
    f. Klient może skorzystać z jednego z dostępnych dla niego rabatów, jeśli data jego wykorzystania nie wygasła

g. Nowe rabaty naliczane są w momencie realizacji zamówienia, do każdego z klientów przypisane jest typ rabatu (jego wysokość) oraz data początku i data końca ważności, w przypadku rabatów jednorazowych data końca ważności to data wykorzystania
h. Naliczanie rabatów, oraz sprawdzanie warunków potrzebnych do rezerwacji stolika odbywa się poprzez przeglądnięcie wszystkich zamówień (tych już zrealizowanych) z danym CustomerID w tabeli [Orders]

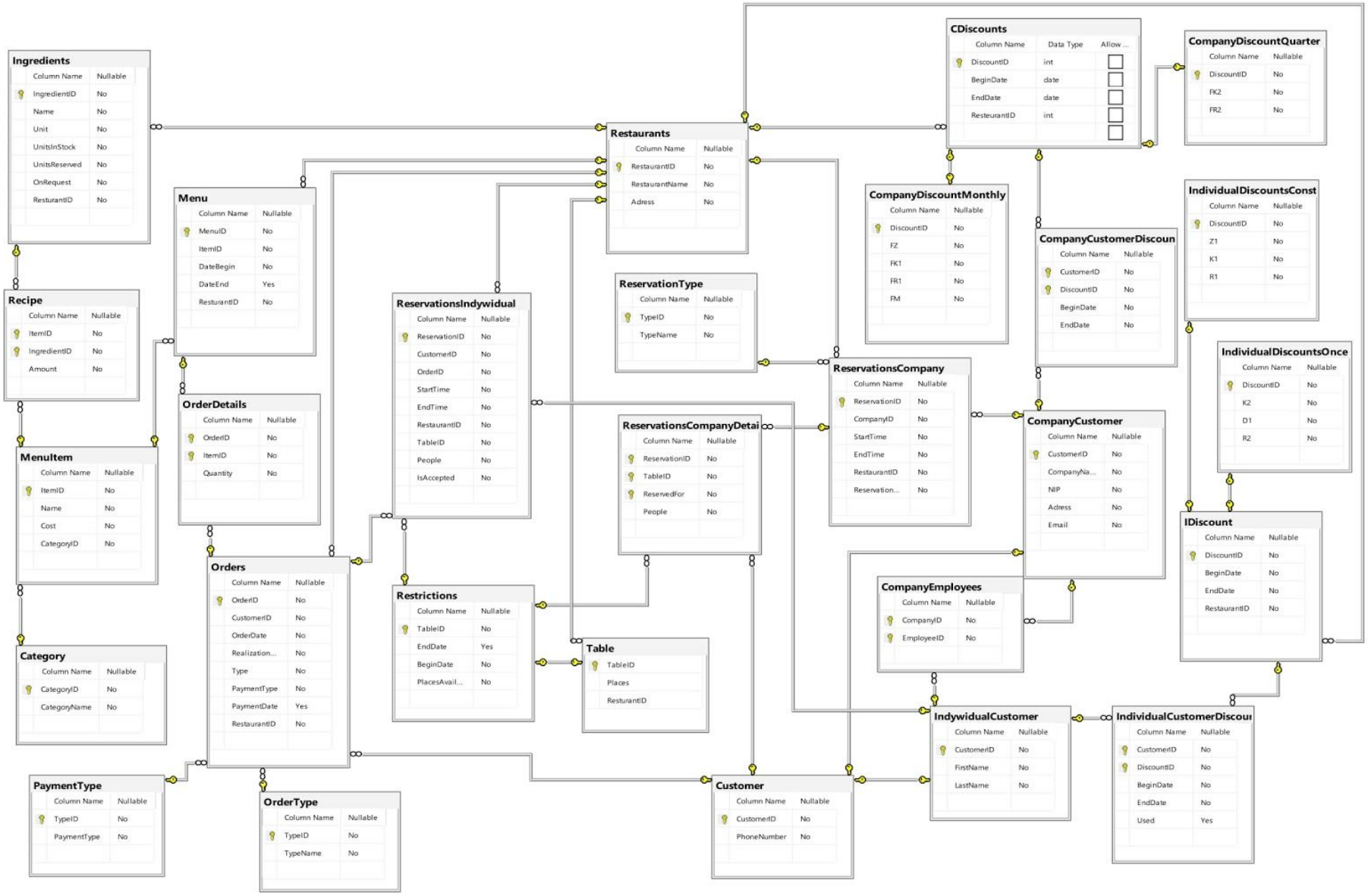
4. Monitorowanie stanu magazynu
    a. podczas składania zamówienia weryfikowana jest ilość półproduktów potrzebnych do realizacji oraz stan magazynu
    b. gdy ilość składników jest zadowalająca po złożeniu zamówienia zwiększany jest rekord (UnitsReserved) w tabeli [Ingredients]
    c. w momencie realizacji zamówienia zmniejszana jest ilość (UnitsReserved) oraz (UnitsInStock)
    d. dbamy o to aby zawsze (UnitsReserved) <= (UnitsInStock) wyjątkiem są pola oznaczone (OnRequest) - np. Owoce Morza.
    e. gdy po złożonym zamówieniu (UnitsReserved) osiągnie taką wartości, że niemożliwe będzia realizacja nawet jednej sztuki jakiegoś Dania z Menu, pracownik zostaje poinformowany o konieczności usunięcia tego dania z Menu
    f. pracownik w wybrane dni zostaje poinformowany o ilości składników (OnRequest) jakie należy zamówić może zostać poinformowany o niskim stanie magazynu niektórych składników

5. Sprawdzanie obostrzeń
    a. Funkcja monitoruje strony rządowe czekając na wyjście nowych obostrzeń. W momencie wyjścia nowego obostrzenia doda je do tabeli [Restrictions] i przejrzy rezerwacje [Reservations] i jeśli przekroczony został limit sali, to będzie anulowała najwcześniejsze zamówienia informując o tym zamawiającego i samą firmę (zasada kto pierwszy - ten lepszy).
    b. Kiedy podjęta zostanie próba rezerwacji stolika, która przekroczyłaby limit osób (obecne obostrzenia w tabeli [Restrictions]), to poinformuje składającego rezerwację o niemożności jej dokonania w tym terminie

6. Generowanie faktur
    a. Pod koniec okresu rozliczeniowego funkcja przejrzy zamówienia z tabeli [Orders], które w polu (Type) są oznaczone jako faktura. Pogrupuje je względem firm, oraz zsumuje. Następnie określi rabat należny firmie i przygotuje fakturę

7. Generowanie raportów
    a. Funkcja co miesiąc przejrzy zamówienia z tabeli [Orders], oraz szczegóły zamówień z tabeli [OrderDetails]. Policzy ilość zamówionych sztuk poszczególnych pozycji menu, sprawdzi aktywność klientów, określi łączny przychód ze sprzedaży.

1. Schemat bazy danych


Ingredients
Column Name Nullable
IngredientID No
Name No
Unit No
UnitsInStock No
UnitsReserved No
OnRequest No
ResturantID No
Recipe
Column Name Nullable
ItemID No
IngredientID No
Amount No
MenuItem
Column Name Nullable
ItemID No
Name No
Cost No
CategoryID No
Category
Column Name Nullable
CategoryID No
CategoryName No
Menu
Column Name Nullable
MenuID No
ItemID No
DateBegin No
DateEnd Yes
ResturantID No
OrderDetails
Column Name Nullable
OrderID No
ItemID No
Quantity No
Orders
Column Name Nullable
OrderID No
CustomerID No
OrderDate No
Realization... No
Type No
PaymentType No
PaymentDate Yes
RestaurantID No
PaymentType
Column Name Nullable
TypeID No
PaymentType No
OrderType
Column Name Nullable
TypeID No
TypeName No
Restaurants
Column Name Nullable
RestaurantID No
RestaurantName No
Adress No
ReservationsIndywidual
Column Name Nullable
ReservationID No
CustomerID No
OrderID No
StartTime No
EndTime No
RestaurantID No
TableID No
People No
IsAccepted No
ReservationType
Column Name Nullable
TypeID No
TypeName No
ReservationsCompanyDetai
Column Name Nullable
ReservationID No
TableID No
ReservedFor No
People No
Restrictions
Column Name Nullable
TableID No
EndDate Yes
BeginDate No
PlacesAvail... No
Table
TableID
Places
ResturantID
Customer
Column Name Nullable
CustomerID No
PhoneNumber No
CDiscounts
Column Name Data Type Allow ...
DiscountID int
BeginDate date
EndDate date
ResteurantID int
CompanyDiscountMonthly
Column Name Nullable
DiscountID No
FZ No
FK1 No
FR1 No
FM No
ReservationsCompany
Column Name Nullable
ReservationID No
CompanyID No
StartTime No
EndTime No
RestaurantID No
Reservation... No
CompanyDiscountQuarter
Column Name Nullable
DiscountID No
FK2 No
FR2 No
CompanyCustomerDiscoun
Column Name Nullable
CustomerID No
DiscountID No
BeginDate No
EndDate No
CompanyCustomer
Column Name Nullable
CustomerID No
CompanyNa... No
NIP No
Adress No
Email No
CompanyEmployees
Column Name Nullable
CompanyID No
EmployeeID No
IndywidualCustomer
Column Name Nullable
CustomerID No
FirstName No
LastName No
IndividualDiscountsConst
Column Name Nullable
DiscountID No
Z1 No
K1 No
R1 No
IndividualDiscountsOnce
Column Name Nullable
DiscountID No
K2 No
D1 No
R2 No
IDiscount
Column Name Nullable
DiscountID No
BeginDate No
EndDate No
RestaurantID No
IndividualCustomerDiscou
Column Name Nullable
CustomerID No
DiscountID No
BeginDate No
EndDate No
Used Yes

2. Opis Tabel

2.1 Category

Tabela przechowuje informacje o kategoriach dań

1. **CategoryID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem typu zamówienia

2. **CategoryName** pole typu nchar przechowujące typ zamówienia, nie może być wartością pustą

```
CREATE TABLE [dbo].[Category](
	[CategoryID] [int] IDENTITY(1,1) NOT NULL,
	[CategoryName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Category] 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 [Unique_CategoryName] UNIQUE NONCLUSTERED
(
	[CategoryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Category]  WITH CHECK ADD  CONSTRAINT
[CK_CategoryName_NotEmpty] CHECK  ((NOT [CategoryName] like ''))
GO

ALTER TABLE [dbo].[Category] CHECK CONSTRAINT [CK_CategoryName_NotEmpty]
GO
```

2.2 CDiscount

Tabela przechowuje informacje o rabatach, funkcjonuje jako słownik

1. **DiscountID** (Klucz główny) wartość typu int, identyfikator zniżki
2. **BeginDate** wartość typu date, pokazuje datę wprowadzenia rabatu
3. **EndDate** wartość typu date, pokazuje datę zakończenia rabatu
4. **RestaurantID** wartość typu int, określa identyfikator restauracji

```
CREATE TABLE [dbo].[CDiscounts](
	[DiscountID] [int] IDENTITY(1,1) NOT NULL,
	[BeginDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
	[ResteurantID] [int] NOT NULL,
 CONSTRAINT [PK_CDiscounts] 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].[CDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_CDiscounts_Restaurants] FOREIGN KEY([ResteurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[CDiscounts] CHECK CONSTRAINT [FK_CDiscounts_Restaurants]
GO

ALTER TABLE [dbo].[CDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_CDiscounts_Date] CHECK  (([BeginDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[CDiscounts] CHECK CONSTRAINT [CK_CDiscounts_Date]
GO
```

2.3 CompanyCustomer

Tabela przechowuje informacje o klientach firmowych.

1. **CustomerID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem klienta.
2. **CompanyName** pole typu varchar nazwę firmy

3. **NIP** pole typu varchar przechowujące NIP firmy, musi być 10-cio cyfrowym napisem

4. **Adress** pole typu varchar przechowujące informacje o adresie firmy, musi być niepustym napisaem który zawierza kods pocztowy w postaci XX-XXX, gdzie X to cyfra
5. **Email** pole typu varchar przechowujące adres e-mailowy

```
CREATE TABLE [dbo].[CompanyCustomer](
	[CustomerID] [int] NOT NULL,
	[CompanyName] [varchar](50) NOT NULL,
	[NIP] [varchar](50) NOT NULL,
	[Adress] [varchar](50) NOT NULL,
	[Email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_CompanyCustomer] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Unique_Email] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Unique_NIP] UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyCustomer]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyCustomer_Customer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customer] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyCustomer] CHECK CONSTRAINT
[FK_CompanyCustomer_Customer]
GO

ALTER TABLE [dbo].[CompanyCustomer]  WITH CHECK ADD  CONSTRAINT
[CK_Adress] CHECK  (([Adress] like '%[0-9][0-9]-[0-9][0-9][0-9]%'))
GO

ALTER TABLE [dbo].[CompanyCustomer] CHECK CONSTRAINT [CK_Adress]
GO

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

```
ALTER TABLE [dbo].[CompanyCustomer] CHECK CONSTRAINT [CK_Email]
GO

ALTER TABLE [dbo].[CompanyCustomer]  WITH CHECK ADD  CONSTRAINT
[NIP_10_Numbers] CHECK  (([NIP] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[CompanyCustomer] CHECK CONSTRAINT [NIP_10_Numbers]
GO
```

2.4 CompanyCustomerDiscount

Tabela przechowuje informacje o rabatach dla klientów firmowych.

1. **CustomerID** (Klucz Główny) wartość typu int będąca identyfikatorem klienta
2. **DiscountID** (Klucz Główny) wartość typu int jest identyfikatorem zniżki
3. **BeginDate** wartość typu date określająca początek obowiązywania rabatu dla pewnego klienta, musi być wcześniejszy niż EndDate
4. **EndDate** wartość typu date określa koniec obowiązywania rabatu

```
CREATE TABLE [dbo].[CompanyCustomerDiscount](
	[CustomerID] [int] NOT NULL,
	[DiscountID] [int] NOT NULL,
	[BeginDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
 CONSTRAINT [PK_CompanyCustomerDiscount] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC,
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyCustomerDiscount_CDiscounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[CDiscounts] ([DiscountID])
GO

ALTER TABLE [dbo].[CompanyCustomerDiscount] CHECK CONSTRAINT
[FK_CompanyCustomerDiscount_CDiscounts]
GO

ALTER TABLE [dbo].[CompanyCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyCustomerDiscount_CompanyCustomer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[CompanyCustomer] ([CustomerID])
GO
```

```
ALTER TABLE [dbo].[CompanyCustomerDiscount] CHECK CONSTRAINT
[FK_CompanyCustomerDiscount_CompanyCustomer]
GO

ALTER TABLE [dbo].[CompanyCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[CK_Dates] CHECK  (([BeginDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[CompanyCustomerDiscount] CHECK CONSTRAINT [CK_Dates]
GO
```

2.5 CompanyDiscountMonthly

Tabela przechowuje informacje o rabatach comiesięcznych

1. **DiscountID** (Klucz Główny) wartość typu int będąca identyfikatorem zniżki

2. **FZ** wartość typu int określająca ilość zamówień koniecznych do otrzymania zniżki, musi być większa, lub równa 0

3. **FK1** wartość typu money określająca minimalną wartość zamówienia liczącego się do zniżki, musi być większa od 0

4. **FR1** wartość typu float określająca procentową wartość zniżki musi być większa od 0 ale mniejsza niż 1

5. **FM** wartość typu float określająca maksymalny rabat procentowy musi być większa od 0 ale mniejsza niż 1

```
CREATE TABLE [dbo].[CompanyDiscountMonthly](
	[DiscountID] [int] NOT NULL,
	[FZ] [int] NOT NULL,
	[FK1] [money] NOT NULL,
	[FR1] [float] NOT NULL,
	[FM] [float] NOT NULL,
 CONSTRAINT [PK_CompanyDiscountMonthly] 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].[CompanyDiscountMonthly]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyDiscountMonthly_CDiscounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[CDiscounts] ([DiscountID])
GO
```

```
ALTER TABLE [dbo].[CompanyDiscountMonthly] CHECK CONSTRAINT
[FK_CompanyDiscountMonthly_CDiscounts]
GO

ALTER TABLE [dbo].[CompanyDiscountMonthly]  WITH CHECK ADD  CONSTRAINT
[CK_Parameters] CHECK  (([FZ]>=(0) AND [FK1]>=(0) AND ([FR1]>=(0) AND [FR1]<=(1))
AND ([FM]>=(0) AND [FM]<=(1))))
GO

ALTER TABLE [dbo].[CompanyDiscountMonthly] CHECK CONSTRAINT
[CK_Parameters]
GO
```

2.6 CompanyDiscountQuarter

Tabela przechowuje informacje o

1. **DiscountID** (Klucz Główny) wartość typu int będąca identyfikatorem zniżki
2. **FK2** wartość typu money określająca sumaryczną ilość pieniędzy potrzebna do otrzymania rabatu, musi być większa od 0
3. **FR2** wartość typu float określająca wartość rabatu procentową musi być większa od 0 ale mniejsza niż 1

```
CREATE TABLE [dbo].[CompanyDiscountQuarter](
	[DiscountID] [int] NOT NULL,
	[FK2] [money] NOT NULL,
	[FR2] [float] NOT NULL,
 CONSTRAINT [PK_CompanyDiscountQuarter] 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].[CompanyDiscountQuarter]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyDiscountQuarter_CDiscounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[CDiscounts] ([DiscountID])
GO

ALTER TABLE [dbo].[CompanyDiscountQuarter] CHECK CONSTRAINT
[FK_CompanyDiscountQuarter_CDiscounts]
GO

ALTER TABLE [dbo].[CompanyDiscountQuarter]  WITH CHECK ADD  CONSTRAINT
[CK_CompanyDiscountQuarter_Parameters] CHECK  (([FK2]>(0) AND ([FR2]>=(0) AND
[FR2]<=(1))))
GO
```

```
ALTER TABLE [dbo].[CompanyDiscountQuarter] CHECK CONSTRAINT
[CK_CompanyDiscountQuarter_Parameters]
GO
```

2.7 CompanyEmployees

Tabela przechowuje informacje o pracownikach danej firmy.

1. **CompanyID** (Klucz Główny) pole typu int zawierające identyfikator firmy

2. **EmployeeID** (Klucz Główny) pole typu int zawierające identyfikator pracownika

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

ALTER TABLE [dbo].[CompanyEmployees]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyEmployees_CompanyCustomer1] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[CompanyCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyEmployees] CHECK CONSTRAINT
[FK_CompanyEmployees_CompanyCustomer1]
GO

ALTER TABLE [dbo].[CompanyEmployees]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyEmployees_IndywidualCustomer] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[IndywidualCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyEmployees] CHECK CONSTRAINT
[FK_CompanyEmployees_IndywidualCustomer]
GO
```

2.8 Customer

Tabela przechowuje informacje o wszystkich klientach.

1. **CustomerID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem klienta

2. **PhoneNumber** pole typu varchar przechowujące numer telefonu klienta, musi być numerem w formacie „+” 11 cyfr

```
CREATE TABLE [dbo].[Customer](
	[CustomerID] [int] IDENTITY(1,1) NOT NULL,
	[PhoneNumber] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Customer] 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].[Customer]  WITH CHECK ADD  CONSTRAINT
[CK_Customer_PhoneNumber] CHECK  (([PhoneNumber] like
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Customer] CHECK CONSTRAINT [CK_Customer_PhoneNumber]
GO
```

2.9 IDiscount

Tabela funkcjonująca jako słownik rabatów indywidualnych

1. **DiscountID** (Klucz główny) wartość typu int, identyfikator zniżki

2. **BeginDate** wartość typu date, pokazuje datę wprowadzenia rabatu

3. **EndDate** wartość typu date, pokazuje datę zakończenia rabatu

4. **RestaurantID** wartość typu int, identyfikuje restaurację

```
CREATE TABLE [dbo].[IDiscount](
	[DiscountID] [int] IDENTITY(1,1) NOT NULL,
	[BeginDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
	[RestaurantID] [int] NOT NULL,
 CONSTRAINT [PK_IDiscount] 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].[IDiscount]  WITH CHECK ADD  CONSTRAINT
[FK_IDiscount_Restaurants] FOREIGN KEY([RestaurantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[IDiscount] CHECK CONSTRAINT [FK_IDiscount_Restaurants]
GO
```

2.10 IndividualCustomerDiscount

Tabela przechowuje informacje o rabatach dla klientów indywidualnych.

1. **CustomerID** (Klucz Główny) wartość typu int będąca identyfikatorem klienta
2. **DiscountID**(Klucz Główny) wartość typu int jest identyfikatorem zniżki
3. **BeginDate** wartość typu date określająca początek obowiązywania rabatu dla pewnego klienta, jest wcześniejsze niż EndDate
4. **EndDate** wartość typu date określa koniec obowiązywania rabatu
5. **RestaurantID** (Klucz Obcy) wartość typu int będąca unikalną reprezentacją restauracji
6. **Used** pole typu int pokazuje czy dany rabat jest wykorzystany

```
CREATE TABLE [dbo].[IndividualCustomerDiscount](
	[CustomerID] [int] NOT NULL,
	[DiscountID] [int] NOT NULL,
	[BeginDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
	[Used] [int] NULL,
 CONSTRAINT [PK_IndividualCustomerDiscount_1] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC,
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualCustomerDiscount_IDiscount1] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[IDiscount] ([DiscountID])
GO

ALTER TABLE [dbo].[IndividualCustomerDiscount] CHECK CONSTRAINT
[FK_IndividualCustomerDiscount_IDiscount1]
GO
```

```
ALTER TABLE [dbo].[IndividualCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualCustomerDiscount_IndywidualCustomer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndywidualCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[IndividualCustomerDiscount] CHECK CONSTRAINT
[FK_IndividualCustomerDiscount_IndywidualCustomer]
GO

ALTER TABLE [dbo].[IndividualCustomerDiscount]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualCustomerDiscount_Date] CHECK  (([BeginDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[IndividualCustomerDiscount] CHECK CONSTRAINT
[CK_IndividualCustomerDiscount_Date]
GO
```

2.11 IndividualDiscountsConst

Tabela przechowuje informacje o

1. **DiscountID** (Klucz Główny) wartość typu int będąca identyfikatorem zniżki
2. **Z1** wartość typu int określająca ilość zamówień, większe od 0
3. **K1** wartość typu money określająca minimalną kwotę zamówienia, większa lub równa 0
4. **R1** wartość typu float określa wartość rabatu, musi być pomiędzy 0 a 1

```
CREATE TABLE [dbo].[IndividualDiscountsConst](
        [DiscountID] [int] NOT NULL,
        [Z1] [int] NOT NULL,
        [K1] [money] NOT NULL,
        [R1] [float] NOT NULL,
 CONSTRAINT [PK_IndividualDiscountsConst] 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].[IndividualDiscountsConst]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualDiscountsConst_IDiscount1] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[IDiscount] ([DiscountID])
GO

ALTER TABLE [dbo].[IndividualDiscountsConst] CHECK CONSTRAINT
[FK_IndividualDiscountsConst_IDiscount1]
GO
```

```
ALTER TABLE [dbo].[IndividualDiscountsConst]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualDiscountsConst_Parameters] CHECK  (([Z1]>(0) AND [K1]>=(0) AND
([R1]>=(0) AND [R1]<=(1))))
GO

ALTER TABLE [dbo].[IndividualDiscountsConst] CHECK CONSTRAINT
[CK_IndividualDiscountsConst_Parameters]
GO
```

2.12 IndividualDiscountsOnce

Tabela przechowuje informacje o

1. **DiscountID** (Klucz Główny) wartość typu int będąca identyfikatorem zniżki

2. **K2** wartość typu money określająca sumaryczną kwotę potrzebną do uzyskania rabatu, większe lub równe 0

3. **D1** wartość typu int określająca ilość dni przez które będzie obowiązywała dana zniżka, musi być większe od 0

4. **R2** wartość typu float określająca wartość zniżki,musi być to wartość pomiędzy 0 a 1

```
CREATE TABLE [dbo].[IndividualDiscountsOnce](
        [DiscountID] [int] NOT NULL,
        [K2] [money] NOT NULL,
        [D1] [int] NOT NULL,
        [R2] [float] NOT NULL,
 CONSTRAINT [PK_IndyvidualDiscountsOnce] 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].[IndividualDiscountsOnce]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualDiscountsOnce_IDiscount] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[IDiscount] ([DiscountID])
GO

ALTER TABLE [dbo].[IndividualDiscountsOnce] CHECK CONSTRAINT
[FK_IndividualDiscountsOnce_IDiscount]
GO

ALTER TABLE [dbo].[IndividualDiscountsOnce]  WITH CHECK ADD  CONSTRAINT
[CK_IndyvidualDiscountsOnce] CHECK  (([K2]>=(0) AND [D1]>(0) AND ([R2]>=(0) AND
[R2]<=(1))))
```

```
GO

ALTER TABLE [dbo].[IndividualDiscountsOnce] CHECK CONSTRAINT
[CK_IndyvidualDiscountsOnce]
GO
```

2.13 IndividualCustomer

Tabela przechowuje informacje o klientach indywidualnych.

1. **CustomerID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem klienta.

2. **FirstName** pole typu varchar imię klienta, nie może być pusty, nie może zawierać cyfr

3. **LastName** pole typu varchar nazwisko klienta

```
CREATE TABLE [dbo].[IndywidualCustomer](
        [CustomerID] [int] NOT NULL,
        [FirstName] [varchar](50) NOT NULL,
        [LastName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_IndywidualCustomer] 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].[IndywidualCustomer]  WITH CHECK ADD  CONSTRAINT
[FK_IndywidualCustomer_Customer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customer] ([CustomerID])
GO

ALTER TABLE [dbo].[IndywidualCustomer] CHECK CONSTRAINT
[FK_IndywidualCustomer_Customer]
GO

ALTER TABLE [dbo].[IndywidualCustomer]  WITH CHECK ADD  CONSTRAINT
[CK_IndywidualCustomer] CHECK  ((NOT [FirstName] like '%[0-9]%' AND NOT
[LastName] like '%[0-9]%'))
GO

ALTER TABLE [dbo].[IndywidualCustomer] CHECK CONSTRAINT
[CK_IndywidualCustomer]
GO
```

2.14 Ingredients

Tabela przechowuje informacje o pojedycznym składniku (połprodukcie) z jakiego tworzone jest danie a także o stanie magazynu.

1. **IngredientID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem zamówienia.

2. **Name** pole typu varchar nazwę półproduktu, nie może być puste

3. **Unit** pole typu varchar będące informacją o tym w jakiej formie, jednostce przechowywane jest dany półprodukt, nie może być puste

4. **UnitsInStock** pole typu int będąca informacją o jednostkach aktuanie znajdujących się w magazynie, nie może być mniejsze niż 0

5. **UnitsReserved** pole typu int będące informacją o jednostkach potrzebnych do realizacji zamówień już złożonych, musi być większe, lub równe 0, musi być zawsze mniejsze bądź równe UnitsInStock, chyba że OnRequest jest równe 1

6. **OnRequest** pole bit informujące o tym czy dany produkt jest zamawiany na specjalne zamówienie (np. owoce morza)

7. **RestaurantID** (Klucz Obcy) pole typu int będąca identyfikatorem restauracji.

```
CREATE TABLE [dbo].[Ingredients](
	[IngredientID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](50) NOT NULL,
	[Unit] [varchar](50) NOT NULL,
	[UnitsInStock] [int] NOT NULL,
	[UnitsReserved] [int] NOT NULL,
	[OnRequest] [bit] NOT NULL,
	[ResturantID] [int] NOT NULL,
 CONSTRAINT [PK_Ingredients] PRIMARY KEY CLUSTERED
(
	[IngredientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Ingredients]  WITH CHECK ADD  CONSTRAINT
[FK_Ingredients_Restaurants] FOREIGN KEY([ResturantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
GO

ALTER TABLE [dbo].[Ingredients] CHECK CONSTRAINT [FK_Ingredients_Restaurants]
GO
```

```
ALTER TABLE [dbo].[Ingredients]  WITH CHECK ADD  CONSTRAINT [CK_Ingredients]
CHECK  ((NOT [Unit] like '' AND NOT [Name] like '' AND [UnitsInStock]>=(0) AND
[UnitsReserved]>=(0) AND ([OnRequest]=(1) OR [UnitsReserved]<=[UnitsInStock])))
GO

ALTER TABLE [dbo].[Ingredients] CHECK CONSTRAINT [CK_Ingredients]
GO
```

2.15 Menu

Tabela przechowuje informacje o menu.

1. **MenuID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem menu.

2. **ItemID** (Klucz Obcy) pole typu int będąca identyfikatorem dania

3. **DateBegin** pole typu date będąca datą pojawienia się pozycji w menu, musi poprzedzać DateEnd, chyba że DateEnd to null

4. **DateEnd** pole typu date będąca datą wycofania pozycji z menu (gdy danie znajduje się w menu pole przyjmuje wartość null)

5. **RestaurantID** (Klucz Obcy) pole typu int będąca identyfikatorem restauracji w której dane menu obowiązuje.

```
CREATE TABLE [dbo].[Menu](
	[MenuID] [int] IDENTITY(2000,1) NOT NULL,
	[ItemID] [int] NOT NULL,
	[DateBegin] [date] NOT NULL,
	[DateEnd] [date] NULL,
	[ResturantID] [int] NOT NULL,
 CONSTRAINT [PK_MenuHistory] PRIMARY KEY CLUSTERED
(
	[MenuID] 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_MenuItem]
FOREIGN KEY([ItemID])
REFERENCES [dbo].[MenuItem] ([ItemID])
GO

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

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [FK_Menu_Restaurants]
FOREIGN KEY([ResturantID])
REFERENCES [dbo].[Restaurants] ([RestaurantID])
```

```
GO

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

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [CK_Menu] CHECK
(([DateEnd] IS NULL OR [DateBegin]<=[DateEnd]))
GO

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

2.16 MenuItem

Tabela przechowuje szczegółowe informacje o pojedynczym daniu (produkcie) jakie może znaleźć się w ofercie restauracji.

1. **ItemID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem dania

2. **Name** pole typu varchar będące nazwą dania, nie jest puste, pole jest unikalne

3. **Cost** pole typu money będące ceną dania, większy lub równy 0

4. **CategoryID** (Klucz Obcy) pole typu int będąca identyfikatorem kategorii do której danie należy

```
CREATE TABLE [dbo].[MenuItem](
	[ItemID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](50) NOT NULL,
	[Cost] [money] NOT NULL,
	[CategoryID] [int] NOT NULL,
 CONSTRAINT [PK_MenuItem] PRIMARY KEY CLUSTERED
(
	[ItemID] 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].[MenuItem]  WITH CHECK ADD  CONSTRAINT
[FK_MenuItem_Category] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Category] ([CategoryID])
GO

ALTER TABLE [dbo].[MenuItem] CHECK CONSTRAINT [FK_MenuItem_Category]
GO
```

```
ALTER TABLE [dbo].[MenuItem]  WITH CHECK ADD  CONSTRAINT [CK_MenuItem]
CHECK  (([Cost]>=(0) AND NOT [Name] like ''))
GO

ALTER TABLE [dbo].[MenuItem] CHECK CONSTRAINT [CK_MenuItem]
GO
```

2.17 OrderDetails

Tabela przechowuje informacje o daniach jakie składają się na zamówienie

1. **OrderID** (Klucz Główny)pole typu int zawierające identyfikator zamówienia

2. **ItemID** (Klucz Główny)pole typu int zawierające identyfikator produktu

3. **Quantity** pole typu int zawierające liczbę sztuk zamówionego dania, musi być większe od 0

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

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

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

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

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

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

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

2.18 Orders

Tabela przechowuje szczegółowe informacje o wszystkich zamówieniach.

1. **OrderID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem zamówienia.

2. **CustomerID** (Klucz Obcy) pole typu int będąca identyfikatorem klienta składającego zamówienie

3. **OrderDate** pole typu date będąca datą złożenia zamówienia, musi poprzedzać RealizationDate (wartość defaultowa)

4. **RealizationDate** pole typu date będąca datą planowanej realizacji zamówienia, musi być większe lub równe obecnej dacie

5. **Type** (Klucz Obcy) pole typu int będąca identyfikatorem typu zamówienia

6. **PaymentType** (Klucz Obcy) pole typu int będąca identyfikatorem typu płatności

7. **PaymentDate** pole typu datę będąca datą płatności, musi być większa, lub równa OrderDate, może przyjmować wartość null

8. **RestaurantID** (Klucz Obcy) pole typu int będąca identyfikatorem restauracji w której zamówienie zostało złożone.

```
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] IDENTITY(1,1) NOT NULL,
	[CustomerID] [int] NOT NULL,
	[OrderDate] [date] NOT NULL,
	[RealizationDate] [datetime] NOT NULL,
	[Type] [int] NOT NULL,
	[PaymentType] [int] NOT NULL,
	[PaymentDate] [date] NULL,
	[RestaurantID] [int] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

```
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Customer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customer] ([CustomerID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_OrderType] FOREIGN KEY([Type])
REFERENCES [dbo].[OrderType] ([TypeID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_PaymentType] FOREIGN KEY([PaymentType])
REFERENCES [dbo].[PaymentType] ([TypeID])
GO

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

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

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

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

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

2.19 OrderType

Tabela przechowuje szczegółowe informacje o wszystkich zamówieniach.

1. **TypeID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem typu zamówienia

2. **TypeName** pole typu nchar przechowujące typ zamówienia, nie jest pusty

```
CREATE TABLE [dbo].[OrderType](
        [TypeID] [int] NOT NULL,
        [TypeName] [nchar](10) NOT NULL,
 CONSTRAINT [PK_OrderType] PRIMARY KEY CLUSTERED
```

```
(
	[TypeID] 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 [Unique_TypeName] UNIQUE NONCLUSTERED
(
	[TypeName] 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].[OrderType]  WITH CHECK ADD  CONSTRAINT [CK_OrderType]
CHECK  ((NOT [TypeName] like ''))
GO

ALTER TABLE [dbo].[OrderType] CHECK CONSTRAINT [CK_OrderType]
GO
```

2.20 PaymentType

Tabela przechowująca informacje o trybie płatności za zamówienie

1. **TypeID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem typu płatności

2. **PaymentType** pole typu nchar przechowujące typ płatności, nie jest pusty, wartość unikalna

```
CREATE TABLE [dbo].[PaymentType](
	[TypeID] [int] NOT NULL,
	[PaymentType] [nchar](10) NOT NULL,
 CONSTRAINT [PK_PaymentType] PRIMARY KEY CLUSTERED
(
	[TypeID] 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 [Unique_PaymentTypeName] UNIQUE NONCLUSTERED
(
	[PaymentType] 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].[PaymentType]  WITH CHECK ADD  CONSTRAINT
[CK_PaymentType] CHECK  ((NOT [PaymentType] like ''))
GO
```

```
ALTER TABLE [dbo].[PaymentType] CHECK CONSTRAINT [CK_PaymentType]
GO
```

2.21 Recipe

Tabela przechowuje informacje o składnikach z jakich składa się dane danie.

1. **IngredientID** (Klucz Główny) pole typu int zawierające identyfikator półproduktu

2. **ItemID**(Klucz Główny) pole typu int zawierające identyfikator produktu

3. **Amount** pole typu int zawierające liczbę jednostek danego półproduktu, musi być większe od 0

```
CREATE TABLE [dbo].[Recipe](
	[ItemID] [int] NOT NULL,
	[IngredientID] [int] NOT NULL,
	[Amount] [int] NOT NULL,
 CONSTRAINT [PK_Recipe] PRIMARY KEY CLUSTERED
(
	[ItemID] ASC,
	[IngredientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Recipe]  WITH CHECK ADD  CONSTRAINT
[FK_Recipe_Ingredients1] FOREIGN KEY([IngredientID])
REFERENCES [dbo].[Ingredients] ([IngredientID])
GO

ALTER TABLE [dbo].[Recipe] CHECK CONSTRAINT [FK_Recipe_Ingredients1]
GO

ALTER TABLE [dbo].[Recipe]  WITH CHECK ADD  CONSTRAINT
[FK_Recipe_MenuItem1] FOREIGN KEY([ItemID])
REFERENCES [dbo].[MenuItem] ([ItemID])
GO

ALTER TABLE [dbo].[Recipe] CHECK CONSTRAINT [FK_Recipe_MenuItem1]
GO

ALTER TABLE [dbo].[Recipe]  WITH CHECK ADD  CONSTRAINT [CK_Recipe] CHECK
(([Amount]>(0)))
GO

ALTER TABLE [dbo].[Recipe] CHECK CONSTRAINT [CK_Recipe]
GO
```

2.22 ReservationsCompany

Tabela przechowuje informacje o rezerwacjach składanych przez klientów firmowych.

1. **ReservationID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem rezerwacji.

2. **CompanyID** (Klucz Obcy) pole typu int będąca identyfikatorem firmy

3. **StartTime** pole typu datetime będąca czasem początku obowiązywania rezerwacji, musi poprzedzać EndTime

4. **EndTime** pole typu datetime będąca czasem zakończenia rezerwacji

5. **RestaurantID** (Klucz Obcy) pole typu int będąca identyfikatorem restauracji w której dana rezerwacja została złożona.

6. **ReservationType** wartość typu int identyfikująca typ rezerwacji

```
CREATE TABLE [dbo].[ReservationsCompany](
	[ReservationID] [int] IDENTITY(1,1) NOT NULL,
	[CompanyID] [int] NOT NULL,
	[StartTime] [datetime] NOT NULL,
	[EndTime] [datetime] NOT NULL,
	[RestaurantID] [int] NOT NULL,
	[ReservationType] [int] NOT NULL,
 CONSTRAINT [PK_ReservationsCompany_1] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservationsCompany]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsCompany_CompanyCustomer] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[CompanyCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[ReservationsCompany] CHECK CONSTRAINT
[FK_ReservationsCompany_CompanyCustomer]
GO

ALTER TABLE [dbo].[ReservationsCompany]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsCompany_ReservationType] FOREIGN KEY([ReservationType])
REFERENCES [dbo].[ReservationType] ([TypeID])
GO

ALTER TABLE [dbo].[ReservationsCompany] CHECK CONSTRAINT
[FK_ReservationsCompany_ReservationType]
GO
```

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

ALTER TABLE [dbo].[ReservationsCompany] CHECK CONSTRAINT
[FK_ReservationsCompany_Restaurants]
GO

ALTER TABLE [dbo].[ReservationsCompany]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationsCompany] CHECK  (([StartTime]<=[EndTime]))
GO

ALTER TABLE [dbo].[ReservationsCompany] CHECK CONSTRAINT
[CK_ReservationsCompany]
GO
```

2.23 ReservationsCompanyDetails

Tabela przechowuje informacje o szczegółach dotyczących rezerwacji dla firm.

1. **ReservationID** (Klucz Główny) unikalna wartość typu int będące identyfikatorem rezerwacji.

2. **TableID** (Klucz Główny)pole typu int będące identyfikatorem stolika

3. **EmployeeID** (Klucz Główny) pole typu int będące identyfikatorem pracownika na którego składana jest rezerwacja lub firmy jeśli jest to rezerwacja „na firmę”

4. **People** pole typu int przechowujące liczbę osób, musi być większe od 0

```
CREATE TABLE [dbo].[ReservationsCompanyDetails](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NOT NULL,
	[ReservedFor] [int] NOT NULL,
	[People] [int] NOT NULL,
 CONSTRAINT [PK_ReservationsCompantDetails_1] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[TableID] ASC,
	[ReservedFor] 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].[ReservationsCompanyDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsCompantDetails_ReservationsCompany1] FOREIGN
KEY([ReservationID])
```

```
REFERENCES [dbo].[ReservationsCompany] ([ReservationID])
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] CHECK CONSTRAINT
[FK_ReservationsCompantDetails_ReservationsCompany1]
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsCompantDetails_Restrictions] FOREIGN KEY([TableID])
REFERENCES [dbo].[Restrictions] ([TableID])
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] CHECK CONSTRAINT
[FK_ReservationsCompantDetails_Restrictions]
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsCompanyDetails_Customer] FOREIGN KEY([ReservedFor])
REFERENCES [dbo].[Customer] ([CustomerID])
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] CHECK CONSTRAINT
[FK_ReservationsCompanyDetails_Customer]
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationsCompanyDetails] CHECK  (([People]>(0)))
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] CHECK CONSTRAINT
[CK_ReservationsCompanyDetails]
GO
```

2.24 ReservationsIndividualDetails

Tabela przechowuje informacje o szczegółach dotyczących rezerwacji dla klientów indywidualnych.

1. **ReservationID**(Klucz Główny) unikalna wartość typu int będąca identyfikatorem rezerwacji.

2. **TableID**(Klucz Główny) pole typu int będące identyfikatorem stolika

3. **CustomerID**(Klucz Główny) pole typu int będące identyfikatorem klienta

4. **People** pole typu int przechowujące liczbę osób, musi być większe od 0

```
CREATE TABLE [dbo].[ReservationsIndyvidualDetails](
        [ReservationID] [int] NOT NULL,
        [TableID] [int] NOT NULL,
```

```
	[CustomerID] [int] NOT NULL,
	[People] [int] NOT NULL,
 CONSTRAINT [PK_ReservationsDetails] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[TableID] ASC,
	[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].[ReservationsIndyvidualDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsDetails_IndywidualCustomer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndywidualCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[ReservationsIndyvidualDetails] CHECK CONSTRAINT
[FK_ReservationsDetails_IndywidualCustomer]
GO

ALTER TABLE [dbo].[ReservationsIndyvidualDetails]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsDetails_Restrictions] FOREIGN KEY([TableID])
REFERENCES [dbo].[Restrictions] ([TableID])
GO

ALTER TABLE [dbo].[ReservationsIndyvidualDetails] CHECK CONSTRAINT
[FK_ReservationsDetails_Restrictions]
GO

ALTER TABLE [dbo].[ReservationsIndyvidualDetails]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationsIndyvidualDetails] CHECK  (([People]>(0)))
GO

ALTER TABLE [dbo].[ReservationsIndyvidualDetails] CHECK CONSTRAINT
[CK_ReservationsIndyvidualDetails]
GO
```

2.25 ReservationsIndividual

Tabela przechowuje informacje o rezerwacjach składanych przez klientów indywidualnych.

1. **ReservationID** (Klucz Główny) unikalna wartość typu int będąca identyfikatorem rezerwacji.

2. **OrderID** (Klucz Obcy) pole typu int będąca identyfikatorem zamówienia

3. **StartTime** pole typu datetime będąca czasem początku obowiązywania rezerwacji, musi poprzedzać EndTime

4. **EndTime** pole typu datetime będąca czasem zakończenia rezerwacji

5. **RestaurantID** (Klucz Obcy) pole typu int będąca identyfikatorem restauracji w której dana rezerwacja została złożona.

6. **TableID** wartość typu int, identyfikuje stolik

7. **People** wartość typu int pokazuje ilość ludzi dla których rezerwujemy miejsca

8. **IsAccepted** wartość typu bit, informuje o tym, czy rezerwacja jest zaakceptowana

```
CREATE TABLE [dbo].[ReservationsIndywidual](
	[ReservationID] [int] IDENTITY(1,1) NOT NULL,
	[CustomerID] [int] NOT NULL,
	[OrderID] [int] NOT NULL,
	[StartTime] [datetime] NOT NULL,
	[EndTime] [datetime] NOT NULL,
	[RestaurantID] [int] NOT NULL,
	[TableID] [int] NOT NULL,
	[People] [int] NOT NULL,
	[IsAccepted] [bit] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[ReservationsIndywidual] CHECK CONSTRAINT
[FK_Reservations_Orders]
GO

ALTER TABLE [dbo].[ReservationsIndywidual]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsIndywidual_IndywidualCustomer] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndywidualCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[ReservationsIndywidual] CHECK CONSTRAINT
[FK_ReservationsIndywidual_IndywidualCustomer]
GO

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

```
ALTER TABLE [dbo].[ReservationsIndywidual] CHECK CONSTRAINT
[FK_ReservationsIndywidual_Restaurants]
GO

ALTER TABLE [dbo].[ReservationsIndywidual]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationsIndywidual_Restrictions] FOREIGN KEY([TableID])
REFERENCES [dbo].[Restrictions] ([TableID])
GO

ALTER TABLE [dbo].[ReservationsIndywidual] CHECK CONSTRAINT
[FK_ReservationsIndywidual_Restrictions]
GO

ALTER TABLE [dbo].[ReservationsIndywidual]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationsIndywidual] CHECK  (([StartTime]<=[EndTime]))
GO

ALTER TABLE [dbo].[ReservationsIndywidual] CHECK CONSTRAINT
[CK_ReservationsIndywidual]
GO
```

2.26 ReservationType

Tabela przechowująca możliwe rodzaje rezerwacji

1. **TypeID** pole o wartości int, identyfikator rodzaju rezerwacji
2. **TypeName** pole varchar(50) z nazwą rodzaju rezerwacji

```
CREATE TABLE [dbo].[ReservationType](
        [TypeID] [int] NOT NULL,
        [TypeName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_ReservationType] PRIMARY KEY CLUSTERED
(
        [TypeID] 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
```

2.27 Restaurants

Tabela przechowuje informacje na temat restauracji (firm) które równolegle korzystają z bazy danych.

1. **RestaurantID** (Klucz Główny) unikalna wartość typu int będąca dentyfikatorem restauracji.

2. **RestaurantName** pole typu varchar przechowujące nazwę restauracji, nie może być puste

3. **Adress** pole typu varchar przechowujące dokładny adres restauracji, musi być niepustym napisem który zawierza kod pocztowy w postaci XX-XXX, gdzie X to cyfra

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

ALTER TABLE [dbo].[Restaurants]  WITH CHECK ADD  CONSTRAINT [CK_Restaurants]
CHECK  (([Adress] like '%[0-9][0-9]-[0-9][0-9][0-9]%' AND NOT [RestaurantName] like ''))
GO

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

2.28 Restrictions

Tabela przechowuje informacje o restrykcjach związanych z COVID-19.

1. **TableID** (Klucz Główny) wartość typu int określająca numer stolika

2. **BeginDate** wartość typu date określająca początek obowiązywania obostrzenia, musi poprzedzać EndDate

3. **EndDate** wartość typu date określająca koniec obowiązywania obostrzenia

4. **PlacesAvailable** wartość typu int określająca ile miejsc jest teraz dostępnych przy danym stoliku, musi być większe od 0

```
CREATE TABLE [dbo].[Restrictions](
	[TableID] [int] NOT NULL,
	[EndDate] [date] NULL,
	[BeginDate] [date] NOT NULL,
	[PlacesAvailable] [int] NOT NULL,
 CONSTRAINT [PK_Restrictions] 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].[Restrictions]  WITH CHECK ADD  CONSTRAINT
[FK_Restrictions_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Table] ([TableID])
GO

ALTER TABLE [dbo].[Restrictions] CHECK CONSTRAINT [FK_Restrictions_Tables]
GO

ALTER TABLE [dbo].[Restrictions]  WITH CHECK ADD  CONSTRAINT [CK_Restrictions]
CHECK  (([BeginDate]<=[EndDate] AND [PlacesAvailable]>=(0)))
GO

ALTER TABLE [dbo].[Restrictions] CHECK CONSTRAINT [CK_Restrictions]
GO
```

2.29 Table

Tabela przechowuje informacje o stolikach.

1. **TableID** (Klucz Główny) wartość typu int określająca numer stolika

2. **Places** wartość typu int określająca ilość miejsc przy stoliku, musi być większe od 0

3. **RestaurantID** (Klucz Obcy) wartość typu int określająca numer restauracji

```
CREATE TABLE [dbo].[Table](
	[TableID] [int] IDENTITY(1,1) NOT NULL,
	[Places] [int] NOT NULL,
	[ResturantID] [int] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
	[TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Table] CHECK CONSTRAINT [FK_Tables_Restaurants]
GO
```

```
ALTER TABLE [dbo].[Table]  WITH CHECK ADD  CONSTRAINT [CK_Tables] CHECK
(([Places]>=(0)))
GO

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

3. Widoki

3.1 vActualPlacesWithRestrictions

Widok pokazujący miejsca uwzględniając ograniczenia

```
CREATE VIEW [dbo].[vActualPlacesWithRestrictions]
AS
SELECT          TOP (100) PERCENT dbo.Restaurants.RestaurantName, dbo.[Table].TableID,
dbo.Restrictions.PlacesAvailable, dbo.[Table].Places, dbo.Restrictions.BeginDate,
dbo.Restrictions.EndDate
FROM            dbo.Restrictions INNER JOIN
                dbo.[Table] ON dbo.Restrictions.TableID = dbo.[Table].TableID INNER JOIN
                dbo.Restaurants ON dbo.[Table].ResturantID = dbo.Restaurants.RestaurantID
WHERE           (dbo.Restrictions.BeginDate <= GETDATE()) AND (dbo.Restrictions.EndDate >=
GETDATE())
ORDER BY dbo.Restaurants.RestaurantName
GO
```

3.2 vAllMenu

Widok pokazujący obecne menu

```
CREATE VIEW [dbo].[vAllMenu]
AS
SELECT          TOP (100) PERCENT dbo.Restaurants.RestaurantName, dbo.MenuItem.Name,
dbo.MenuItem.Cost, dbo.Category.CategoryName, dbo.Menu.DateBegin, dbo.Menu.DateEnd
FROM            dbo.Menu INNER JOIN
                dbo.MenuItem ON dbo.Menu.ItemID = dbo.MenuItem.ItemID INNER JOIN
                dbo.Restaurants ON dbo.Menu.ResturantID = dbo.Restaurants.RestaurantID
INNER JOIN
                dbo.Category ON dbo.MenuItem.CategoryID = dbo.Category.CategoryID
WHERE           (dbo.Menu.DateEnd IS NULL)
ORDER BY dbo.Restaurants.RestaurantName
GO
```

3.3 vCategoryItems

Widok pokazujący kategorie dań

```
CREATE VIEW [dbo].[vCategoryItems]
AS
SELECT          TOP (100) PERCENT dbo.Category.CategoryName, dbo.MenuItem.Name
```

```
FROM            dbo.Category INNER JOIN
                dbo.MenuItem ON dbo.Category.CategoryID = dbo.MenuItem.CategoryID
ORDER BY dbo.Category.CategoryName, dbo.MenuItem.Name
GO
```

3.4 vCompanyEmployees

Widok pokazujący pracowników firmy

```
CREATE VIEW [dbo].[vCompanyEmployees]
AS
SELECT          TOP (100) PERCENT dbo.CompanyCustomer.CompanyName,
dbo.IndywidualCustomer.FirstName, dbo.IndywidualCustomer.LastName
FROM            dbo.CompanyCustomer INNER JOIN
                dbo.CompanyEmployees ON dbo.CompanyCustomer.CustomerID =
dbo.CompanyEmployees.CompanyID INNER JOIN
                dbo.IndywidualCustomer ON dbo.CompanyEmployees.EmployeeID =
dbo.IndywidualCustomer.CustomerID
ORDER BY dbo.CompanyCustomer.CompanyName, dbo.IndywidualCustomer.LastName,
dbo.IndywidualCustomer.FirstName
GO
```

3.5 vIndividualDiscounts

Widok pokazujący zniżki klienta indywidualnego

```
CREATE VIEW [dbo].[vIndividualDiscounts]
AS
SELECT          dbo.Customer.CustomerID, dbo.IDiscount.DiscountID,
dbo.IndividualCustomerDiscount.BeginDate AS assinged,
dbo.IndividualCustomerDiscount.EndDate AS expire, dbo.IDiscount.BeginDate AS start,
                dbo.IDiscount.EndDate AS [end], dbo.IDiscount.RestaurantID
FROM            dbo.IndividualCustomerDiscount INNER JOIN
                dbo.IDiscount ON dbo.IndividualCustomerDiscount.DiscountID =
dbo.IDiscount.DiscountID INNER JOIN
                dbo.Customer ON dbo.IndividualCustomerDiscount.CustomerID =
dbo.Customer.CustomerID
WHERE           (dbo.IDiscount.BeginDate <= GETDATE()) AND
(dbo.IndividualCustomerDiscount.BeginDate <= GETDATE()) AND
(dbo.IndividualCustomerDiscount.EndDate >= GETDATE()) AND (dbo.IDiscount.EndDate >=
GETDATE())
GO
```

3.6 vIngredientsToOrder

Widok pokazujący składniki potrzebne do zamówienia

```
CREATE VIEW [dbo].[vIngredientsToOrder]
AS
SELECT          TOP (100) PERCENT dbo.Restaurants.RestaurantName, dbo.Ingredients.Name,
dbo.Ingredients.UnitsInStock, dbo.Ingredients.UnitsReserved, dbo.Ingredients.OnRequest
```

```
FROM            dbo.Ingredients INNER JOIN
                dbo.Restaurants ON dbo.Ingredients.ResturantID =
dbo.Restaurants.RestaurantID
WHERE           (dbo.Ingredients.OnRequest = 1)
ORDER BY dbo.Restaurants.RestaurantName, dbo.Ingredients.Name
GO
```

3.7 vMostPopularDishes

Widok pokazujący najbardziej popularne dania

```
CREATE VIEW [dbo].[vMostPopularDishes]
AS
SELECT          TOP (100) PERCENT dbo.MenuItem.Name, dbo.Category.CategoryName,
SUM(dbo.OrderDetails.Quantity) AS DishOrdered
FROM            dbo.Category INNER JOIN
                dbo.MenuItem ON dbo.Category.CategoryID = dbo.MenuItem.CategoryID INNER
JOIN
                dbo.OrderDetails ON dbo.MenuItem.ItemID = dbo.OrderDetails.ItemID INNER
JOIN
                dbo.Orders ON dbo.OrderDetails.OrderID = dbo.Orders.OrderID
GROUP BY dbo.MenuItem.Name, dbo.Category.CategoryName
ORDER BY DishOrdered DESC
GO
```

3.8 vStockStatus

Widok pokazujący stan magazynu

```
CREATE VIEW [dbo].[vStockStatus]
AS
SELECT          TOP (100) PERCENT dbo.Restaurants.RestaurantName, dbo.Ingredients.Name,
dbo.Ingredients.Unit, dbo.Ingredients.UnitsInStock, dbo.Ingredients.UnitsReserved,
                dbo.Ingredients.UnitsInStock - dbo.Ingredients.UnitsReserved AS Remain
FROM            dbo.Ingredients INNER JOIN
                dbo.Restaurants ON dbo.Ingredients.ResturantID =
dbo.Restaurants.RestaurantID
ORDER BY dbo.Restaurants.RestaurantName, dbo.Ingredients.Name
GO
```

3.9 vUnpaidOrders

Widok pokazujący niezapłacone zamówienia

```
CREATE VIEW [dbo].[vUnpaidOrders]
AS
SELECT          OrderID, CustomerID, PaymentDate
FROM            dbo.Orders
WHERE           (PaymentDate >= GETDATE())
GO
```

3.10 vUnrealisedOrders

Widok pokazujący niezrealizowane zamówienia

```
CREATE VIEW [dbo].[vUnrealizedOrders]
AS
SELECT      OrderID, CustomerID, RealizationDate
FROM        dbo.Orders
WHERE       (RealizationDate >= GETDATE())
GO
```

4. Procedury

4.1 AddCompanyCustomer

Procedura dodająca klienta będącego firmą

```
CREATE PROCEDURE [dbo].[AddCompanyCustomer]
        @CompanyName varchar(50),
        @NIP varchar(50),
        @Adress varchar(50),
        @Email varchar(50),
        @PhoneNumber varchar(50)
AS
BEGIN
        BEGIN TRY
        INSERT INTO Customer(PhoneNumber)
        VALUES (@PhoneNumber)

        DECLARE @CustomerID int;
        SELECT @CustomerID = scope_identity();
        INSERT INTO CompanyCustomer(CustomerID, CompanyName, NIP, Adress, Email)
        VALUES (@CustomerID, @CompanyName, @NIP, @Adress, @Email)

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

4.2 AddCompanyCustomerDiscount

Procedura przypisująca zniżkę klientowi firmowemu

```
CREATE PROCEDURE [dbo].[AddCompanyCustomerDiscount]
	@CustomerID money,
	@DiscountID int,
	@BeginDate date,
	@EndDate date
AS
BEGIN
	BEGIN TRY

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

	IF NOT EXISTS
	(
		SELECT * FROM CompanyDiscountMonthly
		WHERE CompanyDiscountMonthly.DiscountID = @DiscountID
	)
	BEGIN
		; THROW 52000 , 'Discount does not exist .' ,1
	END
	IF NOT EXISTS
	(
		SELECT * FROM CompanyDiscountQuarter
		WHERE CompanyDiscountQuarter.DiscountID = @DiscountID
	)
	BEGIN
		; THROW 52000 , 'Discount does not exist .' ,1
	END

	INSERT INTO CompanyCustomerDiscount(
		CustomerID,
		DiscountID,
		BeginDate,
		EndDate
		)
	VALUES (
		@CustomerID,
		@DiscountID,
		@BeginDate,
		@EndDate
	)
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot add Discount . Error message : '
```

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

4.3 AddCompanyDiscountMonthly

Procedura dodająca nową zniżkę miesięczną dla klientów firmowych

```
CREATE PROCEDURE [dbo].[AddCompanyDiscountMonthly]
        @FZ int,
        @FK1 money,
        @FR1 float,
        @FM float,
        @BeginDate date,
        @EndDate date,
        @RestaurantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
                WHERE Restaurants.RestaurantID = @RestaurantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END

        INSERT INTO dbo.CDiscounts
        (
          BeginDate,
          EndDate,
          ResteurantID
        )
        VALUES
        (  @BeginDate,
          @EndDate,
          @RestaurantID
        )

        DECLARE @DiscountID int;
        SELECT @DiscountID = scope_identity();

        INSERT INTO dbo.CompanyDiscountMonthly
        (
          DiscountID,
          FZ,
          FK1,
          FR1,
          FM
        )
        VALUES
```

```
        (       @DiscountID,
          @FZ,
          @FK1,
          @FR1,
          @FM
        )


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

4.4 AddCompanyDiscountQuarter

Procedura dodająca nowy rabat firmowy kwartalny

```
CREATE PROCEDURE [dbo].[AddCompanyDiscountQuarter]
        @FK2 money,
        @FR2 float,
        @BeginDate date,
        @EndDate date,
        @RestaurantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
                WHERE Restaurants.RestaurantID = @RestaurantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END

        INSERT INTO dbo.CDiscounts
        (
          BeginDate,
          EndDate,
          ResteurantID
        )
        VALUES
        (  @BeginDate,
          @EndDate,
          @RestaurantID
        )

        DECLARE @DiscountID int;
        SELECT @DiscountID = scope_identity();
```

```
        INSERT INTO dbo.CompanyDiscountQuarter
        (
          DiscountID,
          FK2,
          FR2
        )
        VALUES
        (
                @DiscountID,
          @FK2,
          @FR2
          )



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

4.5 AddCompanyEmployee

Procedura dodająca Pracownika (Klienta Indywidualnego) do Firmy

```
CREATE PROCEDURE [dbo].[AddCompanyEmployee]
        @CompanyID int,
        @EmployeeID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM CompanyCustomer
                WHERE CompanyCustomer.CustomerID = @CompanyID
        )
        BEGIN
                ; THROW 52000 , 'Company does not exist .' ,1
        END

        IF NOT EXISTS
        (
                SELECT * FROM IndywidualCustomer
                WHERE IndywidualCustomer.CustomerID = @EmployeeID
        )
        BEGIN
                ; THROW 52000 , 'Employee does not exist .' ,1
        END

        INSERT INTO CompanyEmployees(CompanyID, EmployeeID)
        VALUES (@CompanyID, @EmployeeID)
```

```
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot and Company Employee . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.6 AddDishToOrder

Procedura dodająca danie do zamówienia

```
CREATE PROCEDURE [dbo].[AddDishToOrder]
        @OrderID int,
        @ItemID int,
        @Quantity int
AS

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

        IF NOT EXISTS
        (
                SELECT * FROM Menu
                WHERE Menu.ItemID = @ItemID AND Menu.DateEnd is null
        )
        BEGIN
                ; THROW 52000 , 'Menu Item does not exist .' ,1
        END


        DECLARE @NewUnits int
        DECLARE @UnitsToReserve int
        DECLARE @MyCursor CURSOR
        DECLARE @IngredientToRemove int
        BEGIN
        SET @MyCursor = CURSOR FOR
        SELECT IngredientID
        FROM DishRecipe(@ItemID)


        OPEN @MyCursor
        FETCH NEXT FROM @MyCursor
        INTO @IngredientToRemove
```

```
        WHILE @@FETCH_STATUS = 0
        BEGIN

        SELECT @UnitsToReserve = dbo.Recipe.Amount * @Quantity
        FROM dbo.Recipe
        WHERE dbo.Recipe.IngredientID = @IngredientToRemove AND dbo.Recipe.ItemID =
@ItemID

                PRINT(@IngredientToRemove)
                PRINT(@UnitsToReserve)
        EXEC ReserveIngredient @IngredientToRemove, @UnitsToReserve


        FETCH NEXT FROM @MyCursor
        INTO @IngredientToRemove
        end

        CLOSE @MyCursor
        DEALLOCATE @MyCursor
        end

        INSERT INTO OrderDetails(
                OrderID,
                ItemID,
                Quantity
                )
        VALUES (
                @OrderID,
                @ItemID,
                @Quantity
        )



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

4.7 AddIndividualCustomer

Procedura dodająca klienta indywidualnego

```
CREATE PROCEDURE [dbo].[AddIndividualCustomer]
        @FirstName varchar(50),
        @LastName varchar(50),
        @PhoneNumber varchar(50)
AS
```

```
BEGIN

	DECLARE @CustomerID int;
	BEGIN TRY
	INSERT INTO Customer(PhoneNumber)
	VALUES (@PhoneNumber)


	SELECT @CustomerID = scope_identity();
	INSERT INTO IndywidualCustomer(CustomerID, FirstName, LastName)
	VALUES (@CustomerID, @FirstName, @LastName)

	END TRY
	BEGIN CATCH
	DELETE FROM Customer
		WHERE Customer.CustomerID = @CustomerID

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

4.8 AddIndividualCustomerDiscount

Procedura dodająca zniżkę do klienta

```
CREATE PROCEDURE [dbo].[AddIndividualCustomerDiscount]
	@CustomerID money,
	@DiscountID int,
	@BeginDate date,
	@EndDate date
AS
BEGIN
	BEGIN TRY
	BEGIN
		; THROW 52000 , 'Restaurant does not exist .' ,1
	END

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

	IF NOT EXISTS
	(
		SELECT * FROM dbo.IndividualDiscountsOnce
```

```
                WHERE IndividualDiscountsOnce.DiscountID = @DiscountID
        )
        BEGIN
                ; THROW 52000 , 'Discount does not exist .' ,1
        END
        IF NOT EXISTS
        (
                SELECT * FROM dbo.IndividualDiscountsConst
                WHERE IndividualDiscountsConst.DiscountID = @DiscountID
        )
        BEGIN
                ; THROW 52000 , 'Discount does not exist .' ,1
        END

        INSERT INTO CompanyCustomerDiscount(
                CustomerID,
                DiscountID,
                BeginDate,
                EndDate
                )
        VALUES (
                @CustomerID,
                @DiscountID,
                @BeginDate,
                @EndDate
        )
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add Discount . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.9 AddIndividualDiscountConst

Procedura dodająca zniżkę stałą dla klientów indywidualnych

```
CREATE PROCEDURE [dbo].[AddIndividualDiscountConst]
        @Z1 int,
        @K1 money,
        @R1 float,
        @BeginDate date,
        @EndDate date,
        @RestaurantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
```

```
                WHERE Restaurants.RestaurantID = @RestaurantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END

        INSERT INTO dbo.IDiscount
        (
           BeginDate,
           EndDate,
           RestaurantID
        )
        VALUES
        (  @BeginDate,
           @EndDate,
           @RestaurantID
        )

        DECLARE @DiscountID int;
        SELECT @DiscountID = scope_identity();

        INSERT INTO dbo.IndividualDiscountsConst
        (
           DiscountID,
           Z1,
           K1,
           R1
        )
        VALUES
        (  @DiscountID,
           @Z1,
           @K1,
           @R1
        )

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

4.10 AddIndividualDiscountOnce

Procedura dodająca jednorazową zniżkę dla klienta

```
CREATE PROCEDURE [dbo].[AddIndividualDiscountOnce]
        @D1 int,
        @K2 money,
        @R2 float,
        @BeginDate date,
```

```
        @EndDate date,
        @RestaurantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
                WHERE Restaurants.RestaurantID = @RestaurantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END

        INSERT INTO dbo.IDiscount
        (
          BeginDate,
          EndDate,
          RestaurantID
        )
        VALUES
        (   @BeginDate,
          @EndDate,
          @RestaurantID
        )

        DECLARE @DiscountID int;
        SELECT @DiscountID = scope_identity();

        INSERT INTO dbo.IndividualDiscountsOnce
        (
          DiscountID,
          K2,
          D1,
          R2
        )
        VALUES
        (
                @DiscountID,
          @K2,
          @D1,
          @R2
         )


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

4.11 AddNewCategory

Procedura dodająca nową kategorię dań

```
CREATE PROCEDURE [dbo].[AddNewCategory]
	@CategoryName varchar(60)
AS
BEGIN
	INSERT INTO Category (CategoryName)
	VALUES (@CategoryName)
END
GO
```

4.12 AddNewIngredient

Procedura dodająca nowy składnik

```
CREATE PROCEDURE [dbo].[AddNewIngredient]
	@Name varchar(50),
	@Unit varchar(50),
	@UnitsInStock int = 0,
	@UnitsReserved int = 0,
	@OnRequest bit = 0,
	@RestaurantID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM Restaurants
		WHERE RestaurantID = @RestaurantID
	)
	BEGIN
		; THROW 52000 , 'Restaurant does not exist .' ,1
	END

	IF @UnitsInStock < 0 OR  @UnitsInStock < 0
	BEGIN
		; THROW 52000 , 'Units number can not be negative.' ,1
	END

	INSERT INTO Ingredients(Name, Unit, UnitsInStock, UnitsReserved, OnRequest,
ResturantID)
	VALUES (@Name, @Unit, @UnitsInStock, @UnitsReserved, @OnRequest,
@RestaurantID)
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot and new Ingredient . Error message : '
	+ ERROR_MESSAGE() ;
```

```
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.13 AddNewMenuItem

Procedura dodająca nowe danie

```
CREATE PROCEDURE [dbo].[AddNewMenuItem]
        @Name varchar(50),
        @Cost money,
        @CategotyID int,
        @RestaurantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
                WHERE RestaurantID = @RestaurantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END
        IF NOT EXISTS
        (
                SELECT * FROM Category
                WHERE CategoryID = @CategotyID
        )
        BEGIN
                ; THROW 52000 , 'Category does not exist .' ,1
        END

        IF @Cost < 0
        BEGIN
                ; THROW 52000 , 'Cost can not be negative.' ,1
        END

        INSERT INTO MenuItem(Name, Cost, CategoryID, ResturantID)
        VALUES (@Name, @Cost, @CategotyID, @RestaurantID)
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot and new MenuItem . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.14 AddNewReipe

Procedura dodająca nowy składnik do przepisu na danie

```
CREATE PROCEDURE [dbo].[AddNewRecipe]
	@IngredientID int,
	@ItemID int,
	@Amount int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM Ingredients
		WHERE IngredientID = @IngredientID
	)
	BEGIN
		; THROW 52000 , 'Ingredient does not exist .' ,1
	END

	IF NOT EXISTS
	(
		SELECT * FROM MenuItem
		WHERE ItemID = @ItemID
	)
	BEGIN
		; THROW 52000 , 'Menu Item does not exist .' ,1
	END

	IF @Amount < 0
	BEGIN
		; THROW 52000 , 'Amount can not be negative.' ,1
	END

	INSERT INTO Recipe(IngredientID, ItemID, Amount)
	VALUES (@IngredientID, @ItemID, @Amount)
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot and new Recipe . Error message : '
	+ ERROR_MESSAGE() ;
	; THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

4.15 AddOrder

Procedura dodająca nowe zamówienie

```
CREATE PROCEDURE [dbo].[AddOrder]
	@CustomerID money,
	@OrderDate date,
	@RealizationDate datetime,
	@TypeID int,
	@PaymentType int,
	@PaymentDate date,
	@RestaurantID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM Restaurants
		WHERE Restaurants.RestaurantID = @RestaurantID
	)
	BEGIN
		; THROW 52000 , 'Restaurant does not exist .' ,1
	END

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

	IF NOT EXISTS
	(
		SELECT * FROM OrderType
		WHERE OrderType.TypeID = @TypeID
	)
	BEGIN
		; THROW 52000 , 'Order type does not exist .' ,1
	END

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

	INSERT INTO Orders(
		CustomerID,
		OrderDate,
		RealizationDate,
		Type,
		PaymentType,
		PaymentDate,
		RestaurantID
```

```
        )
    VALUES (
        @CustomerID,
        @OrderDate,
        @RealizationDate,
        @TypeID,
        @PaymentType,
        @PaymentDate,
        @RestaurantID
    )
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
    = 'Cannot add Order . Error message : '
    + ERROR_MESSAGE() ;
    ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

4.16 AddReservationForCompany

Procedura dodająca rezerwację firmową

```
CREATE PROCEDURE [dbo].[AddReservationForCompany]
    @CompanyID int,
    @StartTime datetime,
    @EndTime datetime,
    @RestaurantID int,
    @ReservationType varchar(50)
AS
BEGIN
    BEGIN TRY
    IF NOT EXISTS
    (
        SELECT * FROM dbo.Restaurants
        WHERE RestaurantID = @RestaurantID
    )
    BEGIN
        ; THROW 52000 , 'Restaurant does not exist .' ,1
    END

    IF NOT EXISTS
    (
        SELECT * FROM ReservationType
        WHERE ReservationType.TypeID = @ReservationType
    )
    BEGIN
        ; THROW 52000 , 'Reservation Type does not exist .' ,1
    END

    INSERT INTO dbo.ReservationsCompany
    (
      CompanyID,
```

```
    StartTime,
    EndTime,
    RestaurantID,
        ReservationType
)
VALUES
(   @CompanyID,        -- ReservationID - int
    @StartTime,            -- CompanyID - int
    @EndTime, -- EndTime - datetime
    @RestaurantID, -- StartTime - datetime
    @ReservationType      -- RestaurantID - int
    )
END TRY
BEGIN CATCH
DECLARE @errorMsg nvarchar (2048)
= 'Cannot add new Reservation . Error message : '
+ ERROR_MESSAGE() ;
; THROW 52000 , @errorMsg ,1;
END CATCH
END
GO
```

4.17 AddReservationForIndividual

Procedura dodająca rezerwację dla klienta indywidualnego

```
CREATE PROCEDURE [dbo].[AddReservationForIndividual]
        @CustomerID int,
        @OrderID int,
        @StartTime DATETIME,
        @EndTime DATETIME,
        @RestaurantID int,
        @TableID int,
        @People int,
        @IsAccepted bit
AS
BEGIN
        BEGIN TRY
        INSERT dbo.ReservationsIndywidual
        (
            CustomerID,
            OrderID,
            StartTime,
            EndTime,
            RestaurantID,
                TableID,
                People,
                IsAccepted
        )
        VALUES
        (   @CustomerID,
        @OrderID,
        @StartTime,
```

```
        @EndTime,
        @RestaurantID,
        @TableID,
        @People,
        @IsAccepted
          )


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

4.18 AddRestaurant

Procedura dodająca nową restaurację

```
CREATE PROCEDURE [dbo].[AddRestaurant]
        @RestaurantName varchar(50),
        @Adress varchar(50)
AS
BEGIN
        SET NOCOUNT ON;
        INSERT INTO Restaurants (RestaurantName, Adress)
        VALUES (@RestaurantName, @Adress)
END
GO
```

4.19 AddRestriction

Procedura dodająca nową restrykcję pandemiczną

```
CREATE PROCEDURE [dbo].[AddRestriction]
        @TableID int,
        @EndDate date,
        @BeginDate date,
        @PlacesAvailable int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM [Table]
                WHERE [Table].TableID = @TableID
        )
        BEGIN
                ; THROW 52000 , 'Table does not exist .' ,1
        END
```

```
        IF @PlacesAvailable < 0 and
                @PlacesAvailable < (SELECT Places from [Table] WHERE TableID = @TableID)
        BEGIN
                ; THROW 52000 , 'Wrong places number.' ,1
        END

        INSERT INTO Restrictions(TableID, EndDate, BeginDate, PlacesAvailable)
        VALUES (@TableID, @EndDate, @BeginDate, @PlacesAvailable)
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add Restriction . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.20 AddTable

Procedura dodająca nowy stolik do restauracji

```
CREATE PROCEDURE [dbo].[AddTable]
        @Places int,
        @ResturantID int
AS
BEGIN
        BEGIN TRY

        IF @Places < 0
        BEGIN
                ; THROW 52000 , 'Places number can not be negative.' ,1
        END

        INSERT INTO [Table](Places, ResturantID)
        VALUES (@Places, @ResturantID)
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add new Table . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.21 AddToMenu

Procedura dodająca danie do obecnego menu

```
CREATE PROCEDURE [dbo].[AddToMenu]
        @ItemID int,
        @DateBegin date = null,
        @DateEnd date = null,
```

```
        @ResturantID int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM MenuItem
                WHERE MenuItem.ItemID = @ItemID
        )
        BEGIN
                ; THROW 52000 , 'Menu Item does not exist .' ,1
        END

        IF NOT EXISTS
        (
                SELECT * FROM Restaurants
                WHERE Restaurants.RestaurantID = @ResturantID
        )
        BEGIN
                ; THROW 52000 , 'Restaurant does not exist .' ,1
        END

        IF @DateBegin is null
        BEGIN
                SET @DateBegin = getdate()
        END

        INSERT INTO Menu(ItemID, DateBegin, DateEnd, ResturantID)
        VALUES (@ItemID, @DateBegin, @DateEnd, @ResturantID)
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add item to the menu. Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.22 AssignDiscountsToCompany

Procedura przyporządkowująca rabaty firmom

```
CREATE PROCEDURE [dbo].[AssingDiscountsToCompany]
        @OrderID INT --order after which we counting discounts
AS
BEGIN
        BEGIN TRY
        DECLARE @RestaurantID INT = (
        SELECT RestaurantID FROM dbo.Orders
        WHERE @OrderID = OrderID
        )

        DECLARE @CustomerID INT = (
        SELECT CustomerID FROM dbo.Orders
```

```
WHERE @OrderID = OrderID
)

DECLARE @PDate date = (
SELECT PaymentDate FROM dbo.Orders
WHERE @OrderID = OrderID
)

DECLARE @DiscountsAvailable TABLE
(
DiscountID int
)

INSERT INTO @DiscountsAvailable
(
   DiscountID
)
VALUES
(
(
        SELECT DiscountID FROM dbo.CDiscounts
        WHERE @RestaurantID = ResteurantID AND
        @PDate BETWEEN BeginDate AND EndDate
)
)

DECLARE @CurrentDiscount INT

DECLARE DISC CURSOR FOR
(
SELECT * FROM @DiscountsAvailable
)

OPEN DISC
FETCH NEXT FROM DISC INTO @CurrentDiscount
WHILE @@FETCH_STATUS = 0
BEGIN

IF @CurrentDiscount IS NOT NULL
BEGIN
IF EXISTS
(
        SELECT * FROM dbo.CompanyCustomerDiscount
        WHERE DiscountID = @CurrentDiscount AND
        @PDate BETWEEN BeginDate AND EndDate
)
BEGIN
        IF EXISTS --znika miesieczna
        (
        SELECT * FROM dbo.CompanyDiscountMonthly
        WHERE DiscountID = @CurrentDiscount
        )
        BEGIN
        IF
        (
```

```
		SELECT COUNT(*) FROM dbo.Orders
		WHERE dbo.Orders.CustomerID = @CustomerID AND
		dbo.Orders.PaymentDate >=
		(
		SELECT TOP 1 BeginDate FROM dbo.CDiscounts
		WHERE dbo.CDiscounts.DiscountID = @CurrentDiscount
		) AND
		RestaurantID = @RestaurantID
		AND
		dbo.OrderCost(OrderID) >=
		(
		SELECT TOP 1 FK1 FROM dbo.CompanyDiscountMonthly
		WHERE dbo.CompanyDiscountMonthly.DiscountID = @CurrentDiscount
		) AND
		DATEDIFF(MONTH, dbo.Orders.PaymentDate, @PDate) <= 1
		) <
(
		SELECT TOP 1 FZ FROM dbo.CompanyDiscountMonthly
		WHERE dbo.CompanyDiscountMonthly.DiscountID = @CurrentDiscount
)
BEGIN
		UPDATE dbo.CompanyCustomerDiscount
		SET BeginDate = @PDate
		WHERE DiscountID = @CurrentDiscount AND
		CustomerID = @CustomerID
END
END

IF EXISTS --znika kwartalne
(
SELECT * FROM dbo.CompanyDiscountQuarter
WHERE DiscountID = @CurrentDiscount
)
BEGIN
IF
(
		SELECT SUM(dbo.OrderCost(OrderID)) FROM dbo.Orders
		WHERE dbo.Orders.CustomerID = @CustomerID AND
		dbo.Orders.PaymentDate >=
		(
		SELECT TOP 1 BeginDate FROM dbo.CDiscounts
		WHERE dbo.CDiscounts.DiscountID = @CurrentDiscount
		) AND RestaurantID = @RestaurantID AND
		DATEDIFF(QUARTER, dbo.Orders.PaymentDate, @PDate) <= 1
) >=
(
		SELECT TOP 1 FK2 FROM dbo.CompanyDiscountQuarter
		WHERE dbo.CompanyDiscountQuarter.DiscountID = @CurrentDiscount
)
BEGIN
		UPDATE dbo.CompanyCustomerDiscount
		SET BeginDate = @PDate
		WHERE DiscountID = @CurrentDiscount AND
		CustomerID = @CustomerID
END
```

```
                END
        END
        ELSE
        BEGIN
                INSERT dbo.CompanyCustomerDiscount
                (
                        CustomerID,
                        DiscountID,
                        BeginDate,
                        EndDate
                )
                VALUES
                (   @CustomerID,            -- CustomerID - int
                        @CurrentDiscount,          -- DiscountID - int
                        @PDate, -- BeginDate - date
                        (
                        SELECT TOP 1 EndDate FROM dbo.CDiscounts
                        WHERE DiscountID = @CurrentDiscount
                )  -- EndDate - date
                        )
                END
                END

        FETCH NEXT FROM DISC INTO @CurrentDiscount

        END

        CLOSE DISC
        DEALLOCATE DISC



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

4.23 AssignDiscountsToCIndividual

Procedura przyporządkowująca rabaty klientom indywidualnym

```
        CREATE PROCEDURE [dbo].[AssingDiscountsToIndividual]
        @OrderID INT --order after which we counting discounts
AS
BEGIN
        BEGIN TRY
        DECLARE @RestaurantID INT = (
        SELECT RestaurantID FROM dbo.Orders
```

```
WHERE @OrderID = OrderID
)

DECLARE @CustomerID INT = (
SELECT CustomerID FROM dbo.Orders
WHERE @OrderID = OrderID
)

DECLARE @PDate date = (
SELECT PaymentDate FROM dbo.Orders
WHERE @OrderID = OrderID
)

DECLARE @DiscountsAvailable TABLE
(
DiscountID int
)

INSERT INTO @DiscountsAvailable
(
   DiscountID
)
VALUES
(
(
        SELECT DiscountID FROM dbo.IDiscount
        WHERE @RestaurantID = RestaurantID AND
        @PDate BETWEEN BeginDate AND EndDate
)
)

DECLARE @CurrentDiscount INT

DECLARE DISC CURSOR FOR
(
SELECT * FROM @DiscountsAvailable
)

OPEN DISC
FETCH NEXT FROM DISC INTO @CurrentDiscount
WHILE @@FETCH_STATUS = 0
BEGIN
IF NOT EXISTS
(
        SELECT * FROM dbo.IndividualCustomerDiscount
        WHERE DiscountID = @CurrentDiscount AND
        @PDate BETWEEN BeginDate AND EndDate
)
BEGIN
        IF EXISTS --znika jednorazowa
        (
        SELECT * FROM dbo.IndividualDiscountsConst
        WHERE DiscountID = @CurrentDiscount
        )
        BEGIN
```

```
IF
(
        SELECT COUNT(*) FROM dbo.Orders
        WHERE dbo.Orders.CustomerID = @CustomerID AND
        dbo.Orders.PaymentDate >=
        (
        SELECT BeginDate FROM dbo.IDiscount
        WHERE dbo.IDiscount.DiscountID = @CurrentDiscount
        ) AND
        RestaurantID = @RestaurantID
        AND
        dbo.OrderCost(OrderID) >=
        (
        SELECT K1 FROM dbo.IndividualDiscountsConst
        WHERE dbo.IndividualDiscountsConst.DiscountID = @CurrentDiscount
        )
) >=
(
        SELECT Z1 FROM dbo.IndividualDiscountsConst
        WHERE dbo.IndividualDiscountsConst.DiscountID = @CurrentDiscount
)
BEGIN
        INSERT dbo.IndividualCustomerDiscount
        (
                CustomerID,
                DiscountID,
                BeginDate,
                EndDate,
                Used
        )
        VALUES
        (   @CustomerID,           -- CustomerID - int
                @CurrentDiscount,          -- DiscountID - int
                @PDate, -- BeginDate - date
        (
                SELECT EndDate FROM dbo.IDiscount
                WHERE dbo.IDiscount.DiscountID = @CurrentDiscount
        ), -- EndDate - date
                NULL    -- Used - bit
                )
END
END

IF EXISTS --znika ciagla
(
SELECT * FROM dbo.IndividualDiscountsOnce
WHERE DiscountID = @CurrentDiscount
)
BEGIN
IF
(
        SELECT SUM(dbo.OrderCost(OrderID)) FROM dbo.Orders
        WHERE dbo.Orders.CustomerID = @CustomerID AND
        dbo.Orders.PaymentDate >=
        (
```

```
                        SELECT BeginDate FROM dbo.IDiscount
                        WHERE dbo.IDiscount.DiscountID = @CurrentDiscount
                        ) AND
                        RestaurantID = @RestaurantID
                ) >=
                (
                        SELECT K2 FROM dbo.IndividualDiscountsOnce
                        WHERE dbo.IndividualDiscountsOnce.DiscountID = @CurrentDiscount
                )
                BEGIN
                        INSERT dbo.IndividualCustomerDiscount
                        (
                                CustomerID,
                                DiscountID,
                                BeginDate,
                                EndDate,
                                Used
                        )
                        VALUES
                        (   @CustomerID,            -- CustomerID - int
                                @CurrentDiscount,          -- DiscountID - int
                                @PDate, -- BeginDate - date
                                DATEADD(DAY,
                        (
                                SELECT D1 FROM dbo.IndividualDiscountsOnce
                                WHERE dbo.IndividualDiscountsOnce.DiscountID =
@CurrentDiscount
                        )
                        , @PDate), -- EndDate - date
                                NULL     -- Used - bit
                                )
                END
                END
        END


        FETCH NEXT FROM DISC INTO @CurrentDiscount
        END

        CLOSE DISC
        DEALLOCATE DISC



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

4.23 DelIngredient

Procedura usuwająca składnik

```
CREATE PROCEDURE [dbo].[DelIngredient]
	@IngredientID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM Ingredients
		WHERE Ingredients.IngredientID = @IngredientID
	)
	BEGIN
		; THROW 52000 , 'Ingredient does not exist .' ,1
	END
	DELETE FROM Ingredients
		WHERE Ingredients.IngredientID = @IngredientID
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot delete Ingredient . Error message : '
	+ ERROR_MESSAGE() ;
	; THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
CREATE PROCEDURE [dbo].[DelIngredient]
	@IngredientID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM Ingredients
		WHERE Ingredients.IngredientID = @IngredientID
	)
	BEGIN
		; THROW 52000 , 'Ingredient does not exist .' ,1
	END
	DELETE FROM Ingredients
		WHERE Ingredients.IngredientID = @IngredientID
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot delete Ingredient . Error message : '
	+ ERROR_MESSAGE() ;
	; THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

4.24 DelMenuItem

Procedura usuwająca danie

```
CREATE PROCEDURE [dbo].[DelMenuItem]
	@ItemID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM MenuItem
		WHERE MenuItem.ItemID = @ItemID
	)
	BEGIN
		; THROW 52000 , 'Item does not exist .' ,1
	END
	DELETE FROM MenuItem
		WHERE MenuItem.ItemID = @ItemID
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot delete MenuItem . Error message : '
	+ ERROR_MESSAGE() ;
	; THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

4.25 RemoveFromMenu

Procedura usuwająca danie z menu

```
CREATE PROCEDURE [dbo].[RemoveFromMenu]
	@MenuID int
AS
BEGIN
	BEGIN TRY
	IF NOT EXISTS
	(
		SELECT * FROM dbo.Menu
		WHERE dbo.Menu.MenuID = @MenuID AND dbo.Menu.DateEnd IS NULL
	)
	BEGIN
		; THROW 52000 , 'MenuID does not exist or was removed.' ,1
	END


	UPDATE dbo.Menu
		SET DateEnd = GETDATE()
		WHERE Menu.MenuID = @MenuID
	END TRY
	BEGIN CATCH
```

```
        DECLARE @errorMsg nvarchar (2048)
        = 'Can not Remove Menu. Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.26 ReserveIngredient

Procedura rezerwująca składniki do dania

```
CREATE PROCEDURE [dbo].[ReserveIngredient]
        @IngredientID int,
        @Value int
AS
BEGIN
        BEGIN TRY
        IF NOT EXISTS
        (
                SELECT * FROM Ingredients
                WHERE IngredientID = @IngredientID
        )
        BEGIN
                ; THROW 52000 , 'Ingredient does not exist .' ,1
        END


        IF @Value < 0
        BEGIN
                ; THROW 52000 , 'VValue can not be negative.' ,1
        END

        UPDATE dbo.Ingredients
                SET UnitsReserved = UnitsReserved + @Value
                WHERE IngredientID = @IngredientID
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot Reserve Ingedient . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.27 UpdateIngredient

Procedura uaktualniająca składnik w magazynie

```
CREATE PROCEDURE [dbo].[UpdateIngredient]
```

```
        @IngredientID int,
        @Name varchar(50),
        @Unit varchar(50),
        @UnitsInStock int,
        @UnitsReserved int,
        @OnRequest bit
AS
BEGIN
        SET NOCOUNT ON

        BEGIN TRY


        IF NOT EXISTS
        (
                SELECT * FROM Ingredients
                WHERE IngredientID = @IngredientID
        )
        BEGIN
                ; THROW 52000 , 'Ingredient does not exist .' ,1
        END

        IF @UnitsInStock < 0 OR  @UnitsInStock < 0
        BEGIN
                ; THROW 52000 , 'Units number can not be negative.' ,1
        END

        IF @UnitsInStock IS NOT NULL
        BEGIN
                UPDATE Ingredients
                SET UnitsInStock = @UnitsInStock
                WHERE Ingredients.IngredientID = @IngredientID
        END

        IF @UnitsReserved IS NOT NULL
        BEGIN
                UPDATE Ingredients
                SET UnitsReserved = @UnitsReserved
                WHERE Ingredients.IngredientID = @IngredientID
        END

        IF @OnRequest IS NOT NULL
        BEGIN
                UPDATE Ingredients
                SET OnRequest = @OnRequest
                WHERE Ingredients.IngredientID = @IngredientID
        END
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot update Ingredient . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
```

```
GO
```

4.29 UpdateMenu

Procedura uaktualniająca menu (poprzez wskazanie dań wycofanych)

```
CREATE PROCEDURE [dbo].[UpdateMenu]
	@MenuID int,
	@DateEnd date
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY

	IF NOT EXISTS
	(
		SELECT * FROM Menu
		WHERE Menu.MenuID = @MenuID
	)
	BEGIN
		; THROW 52000 , 'Menu does not exist .' ,1
	END


	IF @DateEnd IS NOT NULL
	BEGIN
		UPDATE Menu
		SET DateEnd = @DateEnd
		WHERE Menu.MenuID = @MenuID
	END
	END TRY
	BEGIN CATCH
	DECLARE @errorMsg nvarchar (2048)
	= 'Cannot update Manu . Error message : '
	+ ERROR_MESSAGE() ;
	; THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

4.30 UpdateRestriction

Procedura uaktualniająca Obostrzenia

```
CREATE PROCEDURE [dbo].[UpdateRestriction]
	@TableID int,
	@EndDate date,
	@BeginDate date,
	@PlacesAvailable int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
```

```
        IF @TableID is null or NOT EXISTS
        (
                SELECT * FROM [Table]
                WHERE [Table].TableID = @TableID
        )
        BEGIN
                ; THROW 52000 , 'Table does not exist .' ,1
        END

        IF @PlacesAvailable < 0 and
                @PlacesAvailable < (SELECT Places from [Table] WHERE TableID = @TableID)
        BEGIN
                ; THROW 52000 , 'Wrong places number.' ,1
        END

        IF @PlacesAvailable IS NOT NULL
        BEGIN
                IF @PlacesAvailable < 0 and
                @PlacesAvailable < (SELECT Places from [Table] WHERE TableID = @TableID)
                BEGIN
                        ; THROW 52000 , 'Wrong places number.' ,1
                END
                UPDATE Restrictions
                SET PlacesAvailable = @PlacesAvailable
                WHERE Restrictions.TableID = @TableID
        END

        IF @EndDate IS NOT NULL
        BEGIN
                UPDATE Restrictions
                SET EndDate = @EndDate
                WHERE Restrictions.TableID = @TableID
        END

        IF @BeginDate IS NOT NULL
        BEGIN
                UPDATE Restrictions
                SET BeginDate = @BeginDate
                WHERE Restrictions.TableID = @TableID
        END
        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot update Restrictions . Error message : '
        + ERROR_MESSAGE() ;
        ; THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

4.31 UseIndividualDiscountOnce

Procedura wykorzystująca rabat jednorazowy

```
CREATE PROCEDURE [dbo].[UseIndividualDiscountOnce]
	@CustomerID int,
	@DiscountID int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY

	IF @CustomerID IS NULL OR @DiscountID IS NULL
	BEGIN
		; THROW 52000 , 'Customer does not have this discount .' ,1
	END

	IF NOT EXISTS
	(
		SELECT * FROM dbo.IndividualCustomerDiscount
		WHERE @CustomerID = CustomerID
		AND @DiscountID = DiscountID
	)
	BEGIN
		; THROW 52000 , 'Customer does not have this discount .' ,1
	END
	UPDATE dbo.IndividualCustomerDiscount
		SET Used = 1
		WHERE @CustomerID = CustomerID
		AND @DiscountID = DiscountID


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

5. Funkcje zwracające tabele (widoki parametryzowane)

5.1 ActiveCustomersDiscounts

Funkcja zwracająca rabaty przynależne klientowi

```
CREATE FUNCTION [dbo].[ActiveCustomerDiscounts]
```

```
(
	@CustomerID INT,
	@RestaurantID INT,
	@Date date
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT dbo.IDiscount.DiscountID FROM dbo.IndividualCustomerDiscount
	INNER JOIN dbo.IDiscount ON IDiscount.DiscountID =
IndividualCustomerDiscount.DiscountID
	WHERE
	CustomerID = @CustomerID AND
	RestaurantID = @RestaurantID AND
	IDiscount.BeginDate <= @Date AND
	dbo.IDiscount.EndDate >= @Date AND
	IndividualCustomerDiscount.BeginDate <= @Date AND
	IndividualCustomerDiscount.EndDate >= @Date

	UNION

	SELECT dbo.CDiscounts.DiscountID FROM dbo.CompanyCustomerDiscount
	INNER JOIN dbo.CDiscounts ON CDiscounts.DiscountID =
CompanyCustomerDiscount.DiscountID
	WHERE
	CustomerID = @CustomerID AND
	ResteurantID = @RestaurantID AND
	CDiscounts.BeginDate <=@Date AND
	dbo.CDiscounts.EndDate >= @Date AND
	CompanyCustomerDiscount.BeginDate <= @Date AND
	CompanyCustomerDiscount.EndDate >= @Date
	)
GO
```

5.2 CustomerOrderHistory

Funkcja zwracająca historię zamówień klienta

```
	CREATE FUNCTION [dbo].[CustomerOderHistory]
	(
	@CustomerID int
	)
	RETURNS TABLE
	AS
		RETURN
	(
		SELECT * FROM dbo.Orders
		WHERE dbo.Orders.CustomerID = @CustomerID
	)
	GO
```

5.3 DishRecipe

Funkcja zwracająca przepis na danie

```
CREATE FUNCTION [dbo].[DishRecipe]
(
	@ItemID int
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT * FROM dbo.Recipe
	WHERE dbo.Recipe.ItemID = @ItemID
	)
GO
```

5.4 MenuInRestaurant

Funkcja zwraca menu w restauracji

```
CREATE FUNCTION [dbo].[MenuInRestaurant]
(
	@RestaurantID INT
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT MenuItem.ItemID, dbo.MenuItem.Name, DateBegin, DateEnd
	FROM dbo.MenuItem
	INNER JOIN dbo.Menu
	ON Menu.ItemID = MenuItem.ItemID
	WHERE
	Menu.ResturantID = @RestaurantID AND
	DateEnd IS NULL
	)
GO
```

5.5 PositionPossibleToBeInMenu

Funkcja zwracająca dania, które można umieścić w nowym menu

```
CREATE FUNCTION [dbo].[PositionPossibleToBeInMenu]
(
	@RestaurantID INT,
	@DateOfMenuChange DATE
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT DISTINCT A.ItemID, A.Name
	FROM dbo.MenuItem AS A
	INNER JOIN dbo.Menu AS B
```

```
ON B.ItemID = A.ItemID
WHERE
B.ResturantID = @RestaurantID AND
NOT EXISTS
(
	SELECT * FROM dbo.Menu
	WHERE dbo.Menu.ItemID = A.ItemID
	AND  dbo.Menu.ResturantID = @RestaurantID
	AND DateEnd IS NULL
)
AND DATEDIFF(MONTH,
 (
	SELECT TOP 1 DateEnd
	FROM dbo.MenuItem
	INNER JOIN dbo.Menu
	ON Menu.ItemID = MenuItem.ItemID
	WHERE Menu.ItemID = MenuItem.ItemID AND
	Menu.ResturantID = @RestaurantID AND
	DateEnd IS NOT NULL
	ORDER BY DateEnd DESC
), @DateOfMenuChange) >= 1
)

GO
```

5.6 PositionToDelFromMenu

Funkcja zwraca pozycje jakie należy usunąć z menu

```
CREATE FUNCTION [dbo].[PositionToDelFromMenu]
(
	@RestaurantID INT,
	@DateOfMenuChange DATE
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT DISTINCT A.ItemID, A.Name
	FROM dbo.MenuItem AS A
	INNER JOIN dbo.Menu
	ON Menu.ItemID =A.ItemID
	WHERE
	DateEnd IS NULL
	AND DATEDIFF(WEEK,
	(
		DateEnd
	), @DateOfMenuChange) >= 2
	AND @RestaurantID = dbo.Menu.ResturantID
	)
GO
```

5.7 TablesInRestaurant

Funkcja zwraca stoliki dla restauracji

```
CREATE FUNCTION [dbo].[TablesInRestaurant]
(
	@RestaurantID int
)
RETURNS TABLE
AS
	RETURN
	(
	SELECT * FROM dbo.[Table]
	WHERE dbo.[Table].ResturantID = @RestaurantID
	)
GO
```

6. Funkcje zwracające wartości skalarne

6.1 ActiveCustomerDiscountsValue

Funkcja zwraca wartość rabatów przysługujących klientowi w danej restauracji przy danym zamówieniu

```
CREATE FUNCTION [dbo].[ActiveCustomerDiscountsValue]
(
	@CustomerID INT,
	@RestaurantID INT,
	@OrderID INT
)
RETURNS FLOAT
AS
	BEGIN
	DECLARE @discount FLOAT
	DECLARE @allID TABLE
	(
		DiscountID int
	)

	INSERT INTO @allID (DiscountID)
	(SELECT * FROM  dbo.ActiveCustomerDiscounts(@CustomerID, @RestaurantID,
	(
		SELECT PaymentDate FROM Orders
		WHERE OrderID = @OrderID
	)
	))


	SET @discount = 0

	IF EXISTS --individual customer
	(
```

```
        SELECT * FROM dbo.IndywidualCustomer
        WHERE CustomerID = @CustomerID
)
BEGIN
        SET @discount = @discount + ISNULL(
        (
        SELECT SUM(R1) FROM @allD
        INNER JOIN dbo.IndividualDiscountsConst
        ON IndividualDiscountsConst.DiscountID = [@allD].DiscountID
        ), 0)
        SET @discount = @discount + ISNULL(
        (
        SELECT SUM(R2) FROM @allD
        INNER JOIN dbo.IndividualDiscountsOnce
        ON IndividualDiscountsOnce.DiscountID = [@allD].DiscountID

        ), 0)
END
ELSE
BEGIN

        DECLARE @MonthDiscountID INT =
        (
        SELECT  [@allD].DiscountID FROM @allD
        INNER JOIN dbo.CompanyDiscountMonthly
        ON CompanyDiscountMonthly.DiscountID = [@allD].DiscountID
        WHERE [@allD].DiscountID IS NOT NULL
        )
        DECLARE @QuarterDiscountID INT =
        (
        SELECT  [@allD].DiscountID FROM @allD
        INNER JOIN dbo.CompanyDiscountQuarter
        ON CompanyDiscountQuarter.DiscountID = [@allD].DiscountID
        WHERE [@allD].DiscountID IS NOT NULL
        )

        DECLARE @MonthDiscountValue FLOAT = ISNULL(
        (
        DATEDIFF(MONTH,
                (
                        SELECT BeginDate FROM dbo.CompanyCustomerDiscount
                        WHERE CustomerID = @CustomerID
                        AND DiscountID = @MonthDiscountID
                ),
                (
                        SELECT PaymentDate FROM dbo.Orders
                        WHERE @OrderID = OrderID

                )) *
                (
                        SELECT FR1 FROM dbo.CompanyDiscountMonthly
                        WHERE DiscountID = @MonthDiscountID
                )
        ), 0)
```

```
                DECLARE @QuarterDiscountValue FLOAT = ISNULL(
                (
                DATEDIFF(QUARTER,
                        (
                                SELECT BeginDate FROM dbo.CompanyCustomerDiscount
                                WHERE CustomerID = @CustomerID
                                AND DiscountID = @MonthDiscountID
                        ),
                        (
                                SELECT PaymentDate FROM dbo.Orders
                                WHERE @OrderID = OrderID

                        )) *
                        (
                                SELECT FR2 FROM dbo.CompanyDiscountQuarter
                                WHERE DiscountID = @MonthDiscountID
                        )
                ), 0)

                SET @discount = @QuarterDiscountValue
                IF @MonthDiscountValue <=
                ISNULL((
                SELECT FM FROM dbo.CompanyDiscountMonthly
                WHERE DiscountID = @MonthDiscountID
                ), 0)
                BEGIN
                SET @discount = @discount + @MonthDiscountValue
                END
                ELSE
                BEGIN
                SET @discount = @discount +
                ISNULL((
                        SELECT FM FROM dbo.CompanyDiscountMonthly
                        WHERE DiscountID = @MonthDiscountID
                ), 0)
                END


        END

        RETURN @discount
        END
GO
```

6.2 DishInOrderCost

Funkcja obliczająca koszt dania z zamówienia

```
CREATE FUNCTION [dbo].[DishInOrderCost]
(
        @OrderID INT,
        @ItemID INT
)
RETURNS MONEY
AS
BEGIN
```

```
RETURN
	(
	SELECT sum(MenuItem.Cost * OrderDetails.Quantity *
	(1 - dbo.ActiveCustomerDiscountsValue(dbo.Orders.CustomerID,
dbo.Orders.RestaurantID, @OrderID)))
	FROM Orders
	INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
	INNER JOIN MenuItem ON OrderDetails.ItemID = MenuItem.ItemID
	WHERE Orders.OrderID = @OrderId AND MenuItem.ItemID = @ItemID
	)
END
GO
```

6.3 OrderCost

Funkcja zwracająca koszt zamówienia po uwzględnieniu rabatów

```
CREATE FUNCTION [dbo].[OrderCost]
(
	@OrderID INT
)
RETURNS MONEY
AS
BEGIN

RETURN
	(
	(SELECT sum(MenuItem.Cost * OrderDetails.Quantity
	)
	FROM Orders
	INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
	INNER JOIN MenuItem ON OrderDetails.ItemID = MenuItem.ItemID
	WHERE Orders.OrderID = @OrderId
	) * (1 - dbo.ActiveCustomerDiscountsValue(

	(SELECT CustomerID FROM dbo.Orders WHERE OrderID = @OrderID)
	, (SELECT RestaurantID FROM dbo.Orders WHERE OrderID = @OrderID)
	, @OrderID))

	)
END
GO
```

6.4 OrderCostWithoutDiscount

Funkcja zwracająca wartość zamówienia bez rabatów

```
CREATE FUNCTION [dbo].[OrderCostWithoutDiscount]
	(
	@OrderId int
	)
	RETURNS money
AS
```

```
BEGIN
	RETURN
	(
	SELECT sum(MenuItem.Cost * OrderDetails.Quantity)
	FROM Orders
	INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
	INNER JOIN MenuItem ON OrderDetails.ItemID = MenuItem.ItemID
	WHERE Orders.OrderID = @OrderId
	)
END
GO
```

7. Triggery

7.1 NotEnoughIngredientsInStock

Trigger informujący, że magazyn się wyczerpuje

```
CREATE TRIGGER [dbo].[NotEnoughIngredientsInStock]
ON [dbo].[Ingredients]
AFTER INSERT
AS
BEGIN
	SET NOCOUNT ON;
	IF EXISTS
	(
	SELECT * FROM Inserted
	WHERE
	Inserted.OnRequest = 0 AND Inserted.UnitsInStock < Inserted.UnitsReserved
	)
	BEGIN
	; THROW 50001 , 'Not enough Igredients' ,1

	END
END
GO

ALTER TABLE [dbo].[Ingredients] ENABLE TRIGGER [NotEnoughIngredientsInStock]
GO
```

7.2 MenuItemAddingToMenuConditons

Trigger sprawdzający warunki dodania dania do menu

```
CREATE TRIGGER [dbo].[MenuItemAddingToMenuConditons]
ON [dbo].[Menu]
AFTER INSERT
AS
BEGIN
	SET NOCOUNT ON;
```

```
		DECLARE @Date DATE
		SELECT @Date = DATEADD(DAY, 1, GETDATE());

		DECLARE @Restaurant INT
		SET @Restaurant = (
		SELECT Inserted.ResturantID FROM Inserted
		)

		DECLARE @ItemID INT
		SET @ItemID = (
		SELECT Inserted.ItemID FROM Inserted
		)

		IF EXISTS
		(SELECT * FROM dbo.Menu
		WHERE dbo.Menu.ItemID = @ItemID
		AND  dbo.Menu.ResturantID = @Restaurant
		AND dbo.Menu.DateEnd IS NULL
		AND dbo.Menu.MenuID <> (SELECT MenuID FROM Inserted)
		)
		BEGIN
		; THROW 50001 , 'Dish allready in MENu' ,1
		END

		IF NOT EXISTS
		(
		SELECT * FROM dbo.Menu
		WHERE dbo.Menu.ItemID = @ItemID
		AND  dbo.Menu.ResturantID = @Restaurant
		AND dbo.Menu.DateEnd IS NOT NULL
		AND DATEDIFF(MONTH, dbo.Menu.DateEnd, @Date) >= 1
		AND dbo.Menu.MenuID <> (SELECT MenuID FROM Inserted)
		)
		BEGIN
		; THROW 50001 , 'Dish did not fulfil conditions' ,1
		END


END
GO

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

7.3 OrderingDishAvailableInMenu

Trigger sprawdzający czy danie jest dostępne w aktualny menu

```
CREATE TRIGGER [dbo].[OrderingDishAvailableInMenu]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
	SET NOCOUNT ON;
```

```
		DECLARE @OrderID INT
		SET @OrderID = (
		SELECT Inserted.OrderID FROM Inserted
		)
		DECLARE @ItemID INT
		SET @ItemID = (
		SELECT Inserted.ItemID FROM Inserted
		)

		IF NOT EXISTS
		(
		SELECT * FROM dbo.Menu
		WHERE ResturantID =
		(
			SELECT RestaurantID FROM dbo.Orders
			WHERE dbo.Orders.OrderID = @OrderID
		)
		AND ItemID = @ItemID
		AND DateEnd IS NULL
		)
		BEGIN
		; THROW 50001 , 'Dish not in menu' ,1
		END

END
GO

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

7.4 OrderingSeafood

Trigger sprawdzający, czy można zamówić owoce morza

```
CREATE TRIGGER [dbo].[OrderingSeaFood]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
		SET NOCOUNT ON;
		IF EXISTS
		(
		SELECT * FROM Inserted
		INNER JOIN dbo.MenuItem ON
		MenuItem.ItemID = Inserted.ItemID
		INNER JOIN dbo.Category ON
		Category.CategoryID = MenuItem.CategoryID
		INNER JOIN dbo.Orders ON
		Orders.OrderID = Inserted.OrderID
		WHERE Category.CategoryID LIKE('Seafood') AND(
		DATEPART(dw, OrderDate) NOT IN(4, 5, 6)
		)

		)
```

```
        BEGIN
        ; THROW 50001 , 'Wrong day to order seafood' ,1

        END
END
GO
```

7.5 ReservationCompanyDetailAddingOverlapping

Trigger sprawdzający, czy rezerwacje nie kolidują ze sobą

```
CREATE TRIGGER [dbo].[ReservationCompanyDetailAddingOverlapping]
ON [dbo].[ReservationsCompanyDetails]
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @ReservationID INT
        SET @ReservationID = (
        SELECT Inserted.ReservationID FROM Inserted
        )

        DECLARE @TableID INT
        SET @TableID = (
        SELECT Inserted.TableID FROM Inserted
        )

        DECLARE @Restaurant INT
        SET @Restaurant = (
        SELECT RestaurantID FROM dbo.ReservationsCompany
        WHERE ReservationsCompany.ReservationID = @ReservationID
        )

        DECLARE @BeginTime DATETIME
        SET @BeginTime = (
        SELECT StartTime FROM dbo.ReservationsCompany
        WHERE ReservationsCompany.ReservationID = @ReservationID
        )

        DECLARE @EndTime DATETIME
        SET @EndTime = (
        SELECT EndTime FROM dbo.ReservationsCompany
        WHERE ReservationsCompany.ReservationID = @ReservationID
        )


        IF EXISTS
        (
        SELECT * FROM dbo.ReservationsCompany
        INNER JOIN dbo.ReservationsCompanyDetails
        ON ReservationsCompanyDetails.ReservationID = ReservationsCompany.ReservationID
        WHERE
                dbo.ReservationsCompany.ReservationID <> @ReservationID AND
```

```
                ReservationsCompany.RestaurantID = @Restaurant AND
                ReservationsCompanyDetails.TableID = @TableID AND
                (
                ReservationsCompany.StartTime BETWEEN @BeginTime AND @EndTime OR
                ReservationsCompany.EndTime BETWEEN @BeginTime AND @EndTime
                )
        )
        BEGIN
        ; THROW 50001 , 'Overlapping Reservations' ,1
        END

        IF EXISTS
        (
        SELECT * FROM dbo.ReservationsIndywidual
        WHERE
        RestaurantID = @Restaurant AND
        dbo.ReservationsIndywidual.TableID = @TableID AND
                (
                ReservationsIndywidual.StartTime BETWEEN @BeginTime AND @EndTime OR
                ReservationsIndywidual.EndTime BETWEEN @BeginTime AND @EndTime
                )

        )
        BEGIN
        ; THROW 50001 , 'Overlapping Reservations' ,1
        END


END
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] ENABLE TRIGGER
[ReservationCompanyDetailAddingOverlapping]
GO
```

7.6 ReservationForCompanyEmployee

Trigger sprawdzający, czy pracownik faktycznie należy do firmy

```
CREATE TRIGGER [dbo].[ReservationForCompanyEmployee]
ON [dbo].[ReservationsCompanyDetails]
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON;

        IF ( SELECT People FROM Inserted ) < 2
        BEGIN
        ; THROW 50001 , 'Not Enough Places Reserved' ,1
        END

        DECLARE @ReservedFor INT = (SELECT Inserted.ReservedFOR FROM Inserted)
        DECLARE @ReservationID INT = (SELECT Inserted.ReservationID FROM Inserted)

        IF EXISTS
```

```
        (
        SELECT * FROM dbo.CompanyCustomer
        WHERE CustomerID = @ReservedFor
        )
        BEGIN
        IF
        (
                SELECT CompanyID FROM dbo.ReservationsCompany
                WHERE ReservationID = ReservationID
        ) <> @ReservedFor
        BEGIN
                ; THROW 50001 , 'You can not reserve for different Company' ,1
        END
        END

        IF EXISTS
        (
        SELECT * FROM dbo.IndywidualCustomer
        WHERE CustomerID = @ReservedFor
        )
        BEGIN
        IF NOT EXISTS
        (
                SELECT * FROM dbo.CompanyEmployees
                WHERE EmployeeID = @ReservationID AND
                CompanyID = (
                SELECT CompanyID FROM dbo.ReservationsCompany
                WHERE ReservationID = ReservationID
                )
        )
        BEGIN
                ; THROW 50001 , 'This man do not work in this Company' ,1
        END
        END

END
GO

ALTER TABLE [dbo].[ReservationsCompanyDetails] ENABLE TRIGGER
[ReservationForCompanyEmployee]
GO
```

7.7 ReservationForAtLeastTwo

Trigger sprawdzający, czy rezerwacja jest co najmniej na dwie osoby

```
CREATE TRIGGER [dbo].[ReservationForAtLeastTwo]
ON [dbo].[ReservationsIndywidual]
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON;

        IF ( SELECT People FROM Inserted ) < 2
        BEGIN
```

```
        ; THROW 50001 , 'Not Enough Places Reserved' ,1
        END

END
GO

ALTER TABLE [dbo].[ReservationsIndywidual] ENABLE TRIGGER [ReservationForAtLeastTwo]
GO
```

7.8 ReservationIndividualAddingOverlapping

Trigger sprawdzający, czy rezerwacje indywidualne nie nachodzą na siebie

```
CREATE TRIGGER [dbo].[ReservationIndividualAddingOverlapping]
ON [dbo].[ReservationsIndywidual]
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @ReservationID INT
        SET @ReservationID = (
        SELECT Inserted.ReservationID FROM Inserted
        )

        DECLARE @TableID INT
        SET @TableID = (
        SELECT Inserted.TableID FROM Inserted
        )

        DECLARE @Restaurant INT
        SET @Restaurant = (
        SELECT Inserted.RestaurantID FROM Inserted
        )

        DECLARE @BeginTime DATETIME
        SET @BeginTime = (
        SELECT Inserted.StartTime FROM Inserted
        )

        DECLARE @EndTime DATETIME
        SET @EndTime = (
        SELECT Inserted.EndTime FROM Inserted
        )


        IF EXISTS
        (
        SELECT * FROM dbo.ReservationsCompany
        INNER JOIN dbo.ReservationsCompanyDetails
        ON ReservationsCompanyDetails.ReservationID = ReservationsCompany.ReservationID
        WHERE
                ReservationsCompany.RestaurantID = @Restaurant AND
                ReservationsCompanyDetails.TableID = @TableID AND
```

```
                (
                ReservationsCompany.StartTime BETWEEN @BeginTime AND @EndTime OR
                ReservationsCompany.EndTime BETWEEN @BeginTime AND @EndTime
                )
        )
        BEGIN
        ; THROW 50001 , 'Overlapping Reservations' ,1
        END

        IF EXISTS
        (
        SELECT * FROM dbo.ReservationsIndywidual
        WHERE
        ReservationID <> @ReservationID AND
        RestaurantID = @Restaurant AND
        dbo.ReservationsIndywidual.TableID = @TableID AND
                (
                ReservationsIndywidual.StartTime BETWEEN @BeginTime AND @EndTime OR
                ReservationsIndywidual.EndTime BETWEEN @BeginTime AND @EndTime
                )

        )
        BEGIN
        ; THROW 50001 , 'Overlapping Reservations' ,1
        END


END
GO

ALTER TABLE [dbo].[ReservationsIndywidual] ENABLE TRIGGER
[ReservationIndividualAddingOverlapping]
GO
```

7.9 ReservationIndividualConditions

Trigger sprawdzający, czy zamówienie kwalifikuje się na rezerwację

```
CREATE TRIGGER [dbo].[ReservationIndividualConditions]
ON [dbo].[ReservationsIndywidual]
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @Restaurant INT
        SET @Restaurant = (
        SELECT Inserted.RestaurantID FROM Inserted
        )

        DECLARE @OrderID INT
        SET @OrderID = (
        SELECT Inserted.OrderID FROM Inserted
```

```
        )

        DECLARE @CustomerID INT
        SET @CustomerID = (
        SELECT Inserted.CustomerID FROM Inserted
        )

        DECLARE @OrderCost INT
        SET @OrderCost = dbo.OrderCost(@OrderID);

        IF @OrderCost < 50
        BEGIN
        ; THROW 50001 , 'Too low Order Value or not enough orders' ,1
        END


        DECLARE @Orders int
        SET @Orders =
        (
        SELECT COUNT(*) FROM dbo.Orders
        WHERE dbo.Orders.CustomerID = @CustomerID
        AND dbo.Orders.OrderID <> @OrderID
        AND dbo.Orders.RestaurantID = @Restaurant

        )

        IF @OrderCost < 2000  AND @Orders < 5
        BEGIN
        ; THROW 50001 , 'Too low Order Value or not enough orders' ,1
        END

END
GO

ALTER TABLE [dbo].[ReservationsIndywidual] ENABLE TRIGGER
[ReservationIndividualConditions]
GO
```

8.Generowanie faktury

8.1 GenerateInvoice

Funkcja generująca dane do faktury

```
CREATE FUNCTION [dbo].[GenerateInvoice]
(
        @CustomerID INT,
        @RestaurantID INT
)
RETURNS @invoice TABLE
(
```

```
        param_name VARCHAR(50),
        param_val money
)
AS
        BEGIN
        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(50)
        SET @CompanyName =
        (
                SELECT CompanyName FROM dbo.CompanyCustomer
                WHERE CustomerID = @CustomerID
        )
        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.CompanyCustomer
                WHERE CustomerID = @CustomerID
        )
        INSERT @invoice
        (
          param_name,
          param_val
        )
        VALUES
        (  CONCAT('NIP: ', @NIP),  -- param_name - varchar(50)
          NULL -- param_val - money
        )


        DECLARE @Adress VARCHAR(50)
```

```
SET @Adress =
(
        SELECT Adress FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @invoice
(
   param_name,
   param_val
)
VALUES
(  CONCAT('Company Adress: ', @Adress),  -- param_name - varchar(50)
   NULL -- param_val - money
)


DECLARE @Mail VARCHAR(50)
SET @Mail =
(
        SELECT Email FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @invoice
(
   param_name,
   param_val
)
VALUES
(  CONCAT('email: ', @Mail),  -- param_name - varchar(50)
   NULL -- param_val - money
)


DECLARE @OrdID INT
DECLARE @Cost MONEY = 0

DECLARE IT CURSOR FOR
(
        SELECT OrderID FROM dbo.Orders
        WHERE CustomerID = @CustomerID AND
        RestaurantID = @RestaurantID AND
        DATEDIFF(DAY, OrderDate, GETDATE()) <= 31
)

OPEN IT
FETCH NEXT FROM IT INTO @OrdID
WHILE @@FETCH_STATUS = 0
BEGIN
   INSERT @invoice
   (
        param_name,
        param_val
   )
   VALUES
   (  '',  -- param_name - varchar(50)
```

```
        NULL -- param_val - money
    )

    INSERT @invoice
    (
        param_name,
        param_val
    )
    VALUES
    (   CONCAT('Order nr: ', @OrdID),  -- param_name - varchar(50)
        NULL -- param_val - money
    )

    DECLARE @OrdDate DATE
    SET @OrdDate =
    (
    SELECT OrderDate FROM dbo.Orders
    WHERE OrderID = @OrdID
    )

    INSERT @invoice
    (
        param_name,
        param_val
    )
    VALUES
    (   CONCAT('Order Date: ', @OrdDate),  -- param_name - varchar(50)
        NULL -- param_val - money
    )

    DECLARE @Dish INT

    DECLARE DET CURSOR FOR
(
    SELECT ItemID FROM dbo.OrderDetails
    WHERE @OrdID = OrderID
    )

    OPEN DET
    FETCH NEXT FROM DET INTO @Dish
    WHILE @@FETCH_STATUS = 0
    BEGIN
    DECLARE @DishName VARCHAR(50)
    SET @DishName =
    (
        SELECT Name FROM dbo.MenuItem
        WHERE dbo.MenuItem.ItemID = @Dish
    )

    DECLARE @Amount INT
    SET @Amount =
    (
        SELECT Quantity FROM dbo.OrderDetails
        WHERE OrderID = @OrdID AND
        ItemID = @Dish
```

```
                )

                INSERT @invoice
                (
                        param_name,
                        param_val
                )
                VALUES
                (   CONCAT('Dish: ', @DishName, ', quantity: ', @Amount, ', cost: '),  --
param_name - varchar(50)
                        dbo.DishInOrderCost(@OrdID, @Dish) -- param_val - money
                 )


                FETCH NEXT FROM DET INTO @Dish
                END
                CLOSE DET
                DEALLOCATE DET

                INSERT @invoice
                (
                        param_name,
                        param_val
                )
                VALUES
                (   'Order Value',  -- param_name - varchar(50)
                        dbo.OrderCost(@OrdID) -- param_val - money
                 )
                 SET @Cost = @Cost + dbo.OrderCost(@OrdID)

                FETCH NEXT FROM IT INTO @OrdID
        END

        CLOSE IT
        DEALLOCATE IT

        INSERT @invoice
        (
          param_name,
          param_val
        )
        VALUES
        (   '',  -- param_name - varchar(50)
          NULL -- param_val - money
          )

        INSERT @invoice
        (
          param_name,
          param_val
        )
        VALUES
        (   'Total Cost: ',  -- param_name - varchar(50)
          @Cost -- param_val - money
          )
```

```
        RETURN
        END
GO
```

8.2 GenerateOrderInvoice

Funkcja generująca dane do faktury za jedno konkretne zamówienie

```
CREATE FUNCTION [dbo].[GenerateOrderInvoice]
(
        @OrderID int
)
RETURNS @invoice TABLE
(
        param_name VARCHAR(50),
        param_val MONEY
)
AS
        BEGIN
        DECLARE @CustomerID int
        SET @CustomerID =
        (
                SELECT CustomerID FROM dbo.Orders
                WHERE dbo.Orders.OrderID = @OrderID
        )

        DECLARE @RestaurantID int
        SET @RestaurantID =
        (
                SELECT RestaurantID FROM dbo.Orders
                WHERE dbo.Orders.OrderID = @OrderID
        )

        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(50)
        SET @CompanyName =
        (
                SELECT CompanyName FROM dbo.CompanyCustomer
```

```
        WHERE CustomerID = @CustomerID
)
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.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @invoice
(
   param_name,
   param_val
)
VALUES
(  CONCAT('NIP: ', @NIP),  -- param_name - varchar(50)
   NULL -- param_val - money
)


DECLARE @Adress VARCHAR(50)
SET @Adress =
(
        SELECT Adress FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @invoice
(
   param_name,
   param_val
)
VALUES
(  CONCAT('Company Adress: ', @Adress),  -- param_name - varchar(50)
   NULL -- param_val - money
)


DECLARE @Mail VARCHAR(50)
SET @Mail =
(
        SELECT Email FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @invoice
(
   param_name,
```

```
    param_val
)
VALUES
(   CONCAT('email: ', @Mail),  -- param_name - varchar(50)
    NULL -- param_val - money
)



DECLARE @OrdID INT
SET @OrdID = @OrderID

    INSERT @invoice
    (
            param_name,
            param_val
    )
    VALUES
    (   '',  -- param_name - varchar(50)
            NULL -- param_val - money
     )

             INSERT @invoice
             (
                     param_name,
                     param_val
             )
             VALUES
             (   CONCAT('Order nr: ', @OrdID),  -- param_name - varchar(50)
                     NULL -- param_val - money
             )

             DECLARE @OrdDate DATE
             SET @OrdDate =
             (
            SELECT OrderDate FROM dbo.Orders
            WHERE OrderID = @OrdID
             )

             INSERT @invoice
             (
                     param_name,
                     param_val
             )
             VALUES
             (   CONCAT('Order Date: ', @OrdDate),  -- param_name - varchar(50)
                     NULL -- param_val - money
             )

             DECLARE @Dish INT

            DECLARE DET CURSOR FOR
(
            SELECT ItemID FROM dbo.OrderDetails
            WHERE @OrdID = OrderID
```

```
                )

                OPEN DET
                FETCH NEXT FROM DET INTO @Dish
                WHILE @@FETCH_STATUS = 0
                BEGIN
                DECLARE @DishName VARCHAR(50)
                SET @DishName =
                (
                        SELECT Name FROM dbo.MenuItem
                        WHERE dbo.MenuItem.ItemID = @Dish
                )

                DECLARE @Amount INT
                SET @Amount =
                (
                        SELECT Quantity FROM dbo.OrderDetails
                        WHERE OrderID = @OrdID AND
                        ItemID = @Dish
                )

                INSERT @invoice
                (
                        param_name,
                        param_val
                )
                VALUES
                (   CONCAT('Dish: ', @DishName, ', quantity: ', @Amount, ', cost: '),  --
param_name - varchar(50)
                        dbo.DishInOrderCost(@OrdID, @Dish) -- param_val - money
                 )


                FETCH NEXT FROM DET INTO @Dish
                END
                CLOSE DET
                DEALLOCATE DET

                INSERT @invoice
                (
                        param_name,
                        param_val
                )
                VALUES
                (   'Order Value',  -- param_name - varchar(50)
                        dbo.OrderCost(@OrdID) -- param_val - money
                 )


        RETURN
        END
GO
```

9.Generowanie raportów

9.1 GenerateReportForCompanyCustomer

Funkcja generująca raport dla klientów firmowych

```
CREATE FUNCTION [dbo].[GenerateReportForCompanyCustomer]
(
	@CustomerID INT,
	@DateBegin DATE,
	@DateEnd DATE
)
RETURNS @report TABLE
(
	param_name VARCHAR(50),
	param_val VARCHAR(50)
)
AS
	BEGIN
	DECLARE @CompanyName VARCHAR(50)
	SET @CompanyName =
	(
		SELECT CompanyName FROM dbo.CompanyCustomer
		WHERE CompanyCustomer.CustomerID = @CustomerID
	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'Customer Company name: ',  -- param_name - varchar(50)
	  @CompanyName -- param_val - money
	)

	DECLARE @NIP VARCHAR(50)
	SET @NIP =
	(
		SELECT NIP FROM dbo.CompanyCustomer
		WHERE CustomerID = @CustomerID
	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'NIP: ',  -- param_name - varchar(50)
	  @NIP -- param_val - money
	)


	DECLARE @Adress VARCHAR(50)
	SET @Adress =
```

```
(
        SELECT Adress FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Company Adress: ',  -- param_name - varchar(50)
   @Adress -- param_val - money
)


DECLARE @Mail VARCHAR(50)
SET @Mail =
(
        SELECT Email FROM dbo.CompanyCustomer
        WHERE CustomerID = @CustomerID
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'email: ',  -- param_name - varchar(50)
   @Mail -- param_val - money
)



DECLARE @RestaurantsUSed int
SET @RestaurantsUSed =
(
        SELECT COUNT(DISTINCT RestaurantID) FROM dbo.Orders
        INNER JOIN dbo.CompanyCustomer
        ON CompanyCustomer.CustomerID = Orders.CustomerID
        WHERE CompanyCustomer.CustomerID = @CustomerID AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Number of choosed restaurants: ',  -- param_name - varchar(50)
   @RestaurantsUSed -- param_val - money
)

DECLARE @ResevationNumber int
SET @ResevationNumber =
(
```

```
		SELECT COUNT(*) FROM dbo.ReservationsCompany
		WHERE dbo.ReservationsCompany.CompanyID = @CustomerID
		AND dbo.ReservationsCompany.StartTime BETWEEN @DateBegin AND
@DateEnd

	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'Total Reservation Number: ',  -- param_name - varchar(50)
	  @ResevationNumber -- param_val - money
	)

	DECLARE @ReservedPlaces int
	SET @ReservedPlaces =
	(
		(SELECT SUM(People) FROM dbo.ReservationsCompany
		INNER JOIN dbo.ReservationsCompanyDetails
		ON ReservationsCompanyDetails.ReservationID =
ReservationsCompany.ReservationID
		WHERE dbo.ReservationsCompany.CompanyID = @CustomerID
		AND dbo.ReservationsCompany.StartTime BETWEEN @DateBegin AND
@DateEnd
		)

	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'Total Reserved Places: ',  -- param_name - varchar(50)
	  @ReservedPlaces -- param_val - money
	)

	DECLARE @DiscountsAssignedNumber INT

	SET @DiscountsAssignedNumber =
	(
		SELECT COUNT(*) FROM dbo.CompanyCustomerDiscount
		INNER JOIN dbo.CDiscounts
		ON CDiscounts.DiscountID = CompanyCustomerDiscount.DiscountID
		WHERE dbo.CompanyCustomerDiscount.CustomerID = @CustomerID
		AND dbo.CDiscounts.BeginDate BETWEEN @DateBegin AND @DateEnd

	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
```

```
	VALUES
	(  'Total Discount assigned: ',  -- param_name - varchar(50)
	   @DiscountsAssignedNumber -- param_val - money
	)

	DECLARE @CompanyOrders int
	SET @CompanyOrders =
	(
		SELECT COUNT(*) FROM dbo.Orders
		INNER JOIN dbo.CompanyCustomer
		ON CompanyCustomer.CustomerID = Orders.CustomerID
		WHERE CompanyCustomer.CustomerID = @CustomerID AND
		dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total number of orders: ',  -- param_name - varchar(50)
	   @CompanyOrders -- param_val - money
	)

	DECLARE @CompanyOrdersSUM MONEY
	SET @CompanyOrdersSUM =
	(
		SELECT SUM(dbo.OrderCost(dbo.Orders.OrderID)) FROM dbo.Orders
		INNER JOIN dbo.CompanyCustomer
		ON CompanyCustomer.CustomerID = Orders.CustomerID
		WHERE CompanyCustomer.CustomerID = @CustomerID  AND
		dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total orders cost sum:',  -- param_name - varchar(50)
	   CAST(@CompanyOrdersSUM AS VARCHAR(50))  -- param_val - money
	)


	RETURN
	END
GO
```

9.2 GenerateReportForIndividualCustomer

Funkcja generująca raport dla klienta indywidualnego

```
CREATE FUNCTION [dbo].[GenerateReportForIndividualCustomer]
(
	@CustomerID INT,
```

```
        @DateBegin DATE,
        @DateEnd DATE
)
RETURNS @report TABLE
(
        param_name VARCHAR(50),
        param_val VARCHAR(50)
)
AS
        BEGIN
        DECLARE @CustomerFirstName VARCHAR(50)
        SET @CustomerFirstName =
        (
                SELECT FirstName FROM dbo.IndywidualCustomer
                WHERE IndywidualCustomer.CustomerID = @CustomerID
        )
        INSERT @report
        (
           param_name,
           param_val
        )
        VALUES
        (  'Customer First Name: ',  -- param_name - varchar(50)
           @CustomerFirstName -- param_val - money
        )

        DECLARE @CustomerLastName VARCHAR(50)
        SET @CustomerLastName =
        (
                SELECT LastName FROM dbo.IndywidualCustomer
                WHERE IndywidualCustomer.CustomerID = @CustomerID
        )
        INSERT @report
        (
           param_name,
           param_val
        )
        VALUES
        (  'Customer Last Name: ',  -- param_name - varchar(50)
           @CustomerLastName -- param_val - money
        )

        DECLARE @RestaurantsUSed int
        SET @RestaurantsUSed =
        (
                SELECT COUNT(DISTINCT RestaurantID) FROM dbo.Orders
                INNER JOIN dbo.IndywidualCustomer
                ON IndywidualCustomer.CustomerID = Orders.CustomerID
                WHERE IndywidualCustomer.CustomerID = @CustomerID AND
                dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
        )
        INSERT @report
        (
           param_name,
           param_val
```

```
	)
	VALUES
	(  'Number of choosed restaurants: ',  -- param_name - varchar(50)
	   @RestaurantsUSed -- param_val - money
	)

	DECLARE @ResevationNumber int
	SET @ResevationNumber =
	(
		SELECT COUNT(*) FROM dbo.ReservationsIndywidual
		WHERE dbo.ReservationsIndywidual.CustomerID = @CustomerID
		AND dbo.ReservationsIndywidual.StartTime BETWEEN @DateBegin AND
@DateEnd

	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total Reservation Number: ',  -- param_name - varchar(50)
	   @ResevationNumber -- param_val - money
	)

	DECLARE @ReservedPlaces int
	SET @ReservedPlaces =
	(
		(SELECT SUM(People) FROM dbo.ReservationsIndywidual
		WHERE dbo.ReservationsIndywidual.CustomerID = @CustomerID
		AND dbo.ReservationsIndywidual.StartTime BETWEEN @DateBegin AND
@DateEnd
		)

	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total Reserved Places: ',  -- param_name - varchar(50)
	   @ReservedPlaces -- param_val - money
	)

	DECLARE @DiscountsAssignedNumber INT

	SET @DiscountsAssignedNumber =
	(
		SELECT COUNT(*) FROM dbo.IndividualCustomerDiscount
		INNER JOIN dbo.IDiscount
		ON IDiscount.DiscountID = IndividualCustomerDiscount.DiscountID
		WHERE dbo.IndividualCustomerDiscount.CustomerID = @CustomerID
		AND dbo.IDiscount.BeginDate BETWEEN @DateBegin AND @DateEnd
```

```
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total Discount assigned: ',  -- param_name - varchar(50)
   @DiscountsAssignedNumber -- param_val - money
)

DECLARE @IndividualOrders int
SET @IndividualOrders =
(
        SELECT COUNT(*) FROM dbo.Orders
        INNER JOIN dbo.IndywidualCustomer
        ON IndywidualCustomer.CustomerID = Orders.CustomerID
        WHERE IndywidualCustomer.CustomerID = @CustomerID AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total number of orders: ',  -- param_name - varchar(50)
   @IndividualOrders -- param_val - money
)

DECLARE @IndividualOrdersSUM MONEY
SET @IndividualOrdersSUM =
(
        SELECT SUM(dbo.OrderCost(dbo.Orders.OrderID)) FROM dbo.Orders
        INNER JOIN dbo.IndywidualCustomer
        ON IndywidualCustomer.CustomerID = Orders.CustomerID
        WHERE IndywidualCustomer.CustomerID = @CustomerID  AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total orders cost sum:',  -- param_name - varchar(50)
   CAST(@IndividualOrdersSUM AS VARCHAR(50))  -- param_val - money
)


RETURN
END
GO
```

9.3 GenerateReportForRestaurant

Funkcja generująca raport dla restauracji

```
CREATE FUNCTION [dbo].[GenerateReportForRestaurant]
(
	@RestaurantID INT,
	@DateBegin DATE,
	@DateEnd DATE
)
RETURNS @report TABLE
(
	param_name VARCHAR(50),
	param_val VARCHAR(50)
)
AS
	BEGIN
	DECLARE @RestaurantName VARCHAR(50)
	SET @RestaurantName =
	(
		SELECT RestaurantName FROM dbo.Restaurants
		WHERE RestaurantID = @RestaurantID
	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(   'Restaurant Name: ',  -- param_name - varchar(50)
	   @RestaurantName -- param_val - money
	)

	DECLARE @Adress VARCHAR(50)
	SET @Adress =
	(
		SELECT Adress FROM dbo.Restaurants
		WHERE @RestaurantID = RestaurantID
	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(   'Restaurant Addres: ', -- param_name - varchar(50)
	   @Adress  -- param_val - varchar(50)
	 )

	DECLARE @TablesNumber int
	SET @TablesNumber =
	(
		SELECT COUNT(*) FROM dbo.[Table]
		WHERE dbo.[Table].ResturantID = @RestaurantID
	)
```

```
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total tables Number: ',  -- param_name - varchar(50)
	   @TablesNumber -- param_val - money
	)

	DECLARE @PlacesNumber int
	SET @PlacesNumber =
	(
		SELECT SUM(places) FROM dbo.[Table]
		WHERE dbo.[Table].ResturantID = @RestaurantID
	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total place Number: ',  -- param_name - varchar(50)
	   @PlacesNumber -- param_val - money
	)

	DECLARE @ResevationNumber int
	SET @ResevationNumber =
	(
		(
		SELECT COUNT(*) FROM dbo.ReservationsIndywidual
		WHERE dbo.ReservationsIndywidual.RestaurantID = @RestaurantID
		AND dbo.ReservationsIndywidual.StartTime BETWEEN @DateBegin AND
@DateEnd
		)


		+
		(SELECT COUNT(*) FROM dbo.ReservationsCompany
		WHERE dbo.ReservationsCompany.RestaurantID = @RestaurantID
		AND dbo.ReservationsCompany.StartTime BETWEEN @DateBegin AND
@DateEnd
		)

	)
	INSERT @report
	(
	   param_name,
	   param_val
	)
	VALUES
	(  'Total Reservation Number: ',  -- param_name - varchar(50)
	   @ResevationNumber -- param_val - money
	)
```

```
	DECLARE @ReservedPlaces int
	SET @ReservedPlaces =
	(
		(SELECT SUM(People) FROM dbo.ReservationsIndywidual
		WHERE dbo.ReservationsIndywidual.RestaurantID = @RestaurantID
		AND dbo.ReservationsIndywidual.StartTime BETWEEN @DateBegin AND
@DateEnd
		)

		+
		(
		SELECT SUM(People) FROM dbo.ReservationsCompany
		INNER JOIN dbo.ReservationsCompanyDetails
		ON ReservationsCompanyDetails.ReservationID =
ReservationsCompany.ReservationID
		WHERE dbo.ReservationsCompany.RestaurantID = @RestaurantID
		AND dbo.ReservationsCompany.StartTime BETWEEN @DateBegin AND
@DateEnd
		)

	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'Total Reserved Places: ',  -- param_name - varchar(50)
	  @ReservedPlaces -- param_val - money
	)

	DECLARE @DiscountsNumber int
	SET @ReservedPlaces =
	(
		(SELECT COUNT(*) FROM dbo.IDiscount
		WHERE dbo.IDiscount.RestaurantID = @RestaurantID
		AND dbo.IDiscount.BeginDate BETWEEN @DateBegin AND @DateEnd
		)
		+
		(
		SELECT  COUNT(*) FROM dbo.CDiscounts
		WHERE dbo.CDiscounts.ResteurantID = @RestaurantID AND
		dbo.CDiscounts.BeginDate BETWEEN @DateBegin AND @DateEnd
		)

	)
	INSERT @report
	(
	  param_name,
	  param_val
	)
	VALUES
	(  'Total DiscountNumber: ',  -- param_name - varchar(50)
	  @ReservedPlaces -- param_val - money
	)
```

```
DECLARE @DiscountsAssignedNumber int
SET @DiscountsAssignedNumber =
(
        (SELECT COUNT(*) FROM dbo.IndividualCustomerDiscount
        INNER JOIN dbo.IDiscount
        ON IDiscount.DiscountID = IndividualCustomerDiscount.DiscountID
        WHERE dbo.IDiscount.RestaurantID = @RestaurantID
        AND dbo.IDiscount.BeginDate BETWEEN @DateBegin AND @DateEnd
        )
        +
        (
        SELECT  COUNT(*) FROM dbo.CompanyCustomerDiscount
        INNER JOIN dbo.CDiscounts
        ON CDiscounts.DiscountID = CompanyCustomerDiscount.DiscountID
        WHERE dbo.CDiscounts.ResteurantID = @RestaurantID AND
        dbo.CDiscounts.BeginDate BETWEEN @DateBegin AND @DateEnd
        )

)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total Discount Number assigned to customers: ',  -- param_name - varchar(50)
   @DiscountsAssignedNumber -- param_val - money
)

DECLARE @DishesNumber int
SET @DishesNumber =
(
        SELECT COUNT(DISTINCT ItemID) FROM dbo.Menu
        WHERE ResturantID = @RestaurantID AND
        dbo.Menu.DateBegin BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total Number of Distinct Dishes in Menu: ',  -- param_name - varchar(50)
   @DishesNumber -- param_val - money
)

DECLARE @IndividualOrders int
SET @IndividualOrders =
(
        SELECT COUNT(*) FROM dbo.Orders
        INNER JOIN dbo.IndywidualCustomer
        ON IndywidualCustomer.CustomerID = Orders.CustomerID
        WHERE RestaurantID = @RestaurantID AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
```

```
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total Number orders made by individual Customers: ',  -- param_name - varchar(50)
   @IndividualOrders -- param_val - money
)

DECLARE @IndividualOrdersSUM MONEY
SET @IndividualOrdersSUM =
(
        SELECT SUM(dbo.OrderCost(dbo.Orders.OrderID)) FROM dbo.Orders
        INNER JOIN dbo.IndywidualCustomer
        ON IndywidualCustomer.CustomerID = Orders.CustomerID
        WHERE RestaurantID = @RestaurantID AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total orders cost sum made by individual:',  -- param_name - varchar(50)
   CAST(@IndividualOrdersSUM AS VARCHAR(50))  -- param_val - money
)

DECLARE @CompanyOrders int
SET @CompanyOrders =
(
        SELECT COUNT(*) FROM dbo.Orders
        INNER JOIN dbo.CompanyCustomer
        ON CompanyCustomer.CustomerID = Orders.CustomerID
        WHERE RestaurantID = @RestaurantID AND
        dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
)
INSERT @report
(
   param_name,
   param_val
)
VALUES
(  'Total Number orders made by Compoany Customers: ',  -- param_name - varchar(50)
   @CompanyOrders -- param_val - money
)

DECLARE @CompanyOrdersSUM MONEY
SET @CompanyOrdersSUM =
(
        SELECT SUM(dbo.OrderCost(dbo.Orders.OrderID)) FROM dbo.Orders
        INNER JOIN dbo.CompanyCustomer
        ON CompanyCustomer.CustomerID = Orders.CustomerID
```

```
                WHERE RestaurantID = @RestaurantID AND
                dbo.Orders.OrderDate BETWEEN @DateBegin AND @DateEnd
        )
        INSERT @report
        (
          param_name,
          param_val
        )
        VALUES
        (  'Total orders cost sum made by company:',  -- param_name - varchar(50)
          CAST(@CompanyOrdersSUM AS VARCHAR(50))  -- param_val - money
        )

        INSERT @report
        (
          param_name,
          param_val
        )
        VALUES
        (  'Total orders cost sum', -- param_name - varchar(50)
           CAST(ROUND(@IndividualOrdersSUM + @CompanyOrdersSUM, 2) AS
VARCHAR(50))  -- param_val - varchar(50)
         )



        RETURN
        END
GO
```

10.Indeksy

10.1 Discount_C_Mothly

Indeks pokazujący zniżki miesięczne

```
CREATE NONCLUSTERED INDEX [Discount_C_Mothly] ON [dbo].[CompanyDiscountMonthly]
(
        [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.2 CustomerID

Indeks pokazujący ID klienta

```
CREATE NONCLUSTERED INDEX [C_Customer_ID] ON [dbo].[CompanyCustomer]
(
        [CustomerID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.3 Discount_C_Quarter

Indeks pokazujący zniżki kwartalne

```
CREATE NONCLUSTERED INDEX [Discount_C_Quarter] ON [dbo].[CompanyDiscountQuarter]
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.4 Discount_I_Const

Indeks pokazujący zniżki stałe

```
CREATE NONCLUSTERED INDEX [Discount_I_Const] ON [dbo].[IndividualDiscountsConst]
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.5 Restaurant_ID

Indeks pokazujący ID restauracji

```
CREATE NONCLUSTERED INDEX [Restaurant_ID] ON [dbo].[Restaurants]
(
	[RestaurantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.6 Discount_I_Once

Indeks pokazujący zniżki jednorazowe

```
CREATE NONCLUSTERED INDEX [Discount_I_Once] ON [dbo].[IndividualDiscountsOnce]
(
	[DiscountID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

10.7 I_Customer_ID

Indeks pokazujacy klientów indywidualnych

```
CREATE NONCLUSTERED INDEX [I_Customer_ID] ON [dbo].[IndywidualCustomer]

(

    [CustomerID] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]

GO
```

11.Rolę w systemie

1. Admin
    - Dodanie nowej Restauracji
2. Menedżer Restauracji
    - Dodawanie/Aktualizacja/Usuwanie rabatów
    - Dodanie Dania
    - Sprawdzenie produktów które należy zamówić
    - Generowanie statystyk dla restauracji
    - Wgląd do stanu magazynu
    - Wprowadzanie stanu aktualnych obostrzeń
3. Pracownik Restauracji
    - Wgląd do stanu magazynu
    - Dodawanie Klientów Indywidualnych
    - Dodawanie Klientów Firmowych
    - Przyjęcie zamówienia / Akceptacja
4. Klient Indywidualny
    - Sprawdzenie historii zamówień
    - Sprawdzenie statystyk
    - Sprawdzenie aktualnych rabatów
    - Złożenie zamówienia
    - Złożenie rezerwacji
5. Klient Firmowy
    - Sprawdzenie historii zamówień
    - Sprawdzenie statystyk

- Sprawdzenie aktualnych rabatów
- Generowanie faktury za ostatni miesiąc
- Generowanie faktury za dane zamówienie
- Dodanie pracownika jako klient indywidualnego
- Złożenie zamówienia
- Złożenie rezerwacji

6. Funkcje systemowe
    - Automatyczne zwiększenie lub zmniejszenie stanu półproduktu w magazynie
    - Obliczenie rabatów dla danego klienta
    - Obsługa systemu rabatów
    - Obliczenie wartości zamówienia
    - Kontrola dostępności stolików przy rezerwacjach
    - Kontrola zasad związanych z Menu
    - Kontrola zasad związanych z owocami morza

12.Generator danych

Dane zostały wygenerowane przy pomocy oprogramowania REDGATE a także przy pomocy generatora MOCKAROO. Dane odpowiadają okresowi około trzech lat użytkowania danych. Dane dotyczące nazw firm, produktów, danych klientów, adresów itp są rzeczywistymi danymi pobranymi z datasetów dostępnych w REDGATE. Dane złożonych zamówień, rabatów itp są silnie randomizowane.

13. Testowanie funkcjonalności

Poniższy kod pozwala w pełni przetestować funkcje, widoki, i generowanie faktur oraz statystyk. Funkcje są parametryzowane dlatego dobraliśmy je w takisposób, aby pokazać ich rzeczywiste działanie (tj. zapewniamy, że gdy funkcja przyjmuje ID klienta - A i ID zamówienia - B to A na pewno złożył zamówienie B itp.)

```
--TESTING TABLE FUNCTIONS

SELECT * FROM dbo.ActiveCustomerDiscounts(2, 3, GETDATE())

SELECT * FROM dbo.ActiveCustomerDiscounts(28, 3, GETDATE())

SELECT * FROM dbo.CustomerOderHistory(2)

SELECT * FROM dbo.DishRecipe(3)

SELECT * FROM dbo.MenuInRestaurant(2)

SELECT * FROM dbo.PositionPossibleToBeInMenu(2, GETDATE())

SELECT * FROM dbo.PositionToDelFromMenu(2, GETDATE())
```

```sql
SELECT * FROM dbo.TablesInRestaurant(3)



--TESTING SCALAR FUNCTIONS


SELECT dbo.ActiveCustomerDiscountsValue(28, 3, 186)

SELECT dbo.OrderCost(186)

SELECT dbo.OrderCostWithoutDiscount(186)

SELECT dbo.DishInOrderCost(186, 466)


--TESTING INVOICES

SELECT * FROM GenerateOrderInvoice(186)

SELECT * FROM GenerateOrderInvoice(422);

SELECT * FROM GenerateInvoice(79, 9);

SELECT * FROM GenerateInvoice(642, 71);


--TESTING RAPORTS

SELECT * FROM GenerateReportForRestaurant(71, '2018-01-01',GETDATE())

SELECT * FROM GenerateReportForIndividualCustomer(2, '2018-01-01',GETDATE())

SELECT * FROM GenerateReportForCompanyCustomer(11, '2018-01-01',GETDATE())


--TESTING VIEWS

SELECT * FROM dbo.vActualPlacesWithRestrictions ORDER BY RestaurantName

SELECT * FROM dbo.vAllMenu ORDER BY RestaurantName

SELECT * FROM dbo.vCategoryItems ORDER BY CategoryName

SELECT * FROM dbo.vCompanyEmployees ORDER BY CompanyName

SELECT * FROM dbo.vIndividualDiscounts

SELECT * FROM dbo.vIngredientsToOrder
```

```
SELECT * FROM dbo.vMostPopularDishes ORDER BY DishOrdered DESC

SELECT * FROM dbo.vStockStatus

SELECT * FROM dbo.vUnpaidOrders

SELECT * FROM dbo.vUnrealizedOrders
```