Spis treści:

1. Funkcje systemu

1.1 Administrator bazy danych

- ➔ Dodawanie i usuwanie użytkowników z bazy danych (klientów, pracowników).

1.2 Klient

- ➔ Składanie zamówień;
- ➔ Aktualizacja swoich danych w systemie;
- ➔ Składanie rezerwacji (data rezerwacji, stolik) z jednoczesnym złożeniem zamówienia;
- ➔ Możliwość skorzystania z zamówienia owoców morza przy spełnieniu warunku odnośnie dni składania takiego zamówienia.

1.3 Pracownik restauracji

- ➔ Przyjmowanie zamówień;
- ➔ Generowanie faktur;
- ➔ Generowanie e-maili informujących o dostępności rabatu;
- ➔ Przyjmowanie rezerwacji.

1.4 Menedżer

- ➔ Ustalanie kwot rabatu, wskaźników WK, WZ;
- ➔ Decyzja o zamianie pozycji w menu;
- ➔ Zbieranie zamówień owoców morza i aktualizacja ilości zamówionej na dany dzień;
- ➔ Przekazywanie informacji dostawcom;
- ➔ Dodawanie i usuwanie stolików;
- ➔ Modyfikowanie ilości miejsc przy stolikach;
- ➔ Dodawanie i usuwanie dań z menu.

1.5 Pracownicy kuchni

- ➔ Aktualizacja ilości dań dostępnych na stanie oraz powiadomienie o brakujących produktach;
- ➔ Dodawanie nowych dostępnych dań.

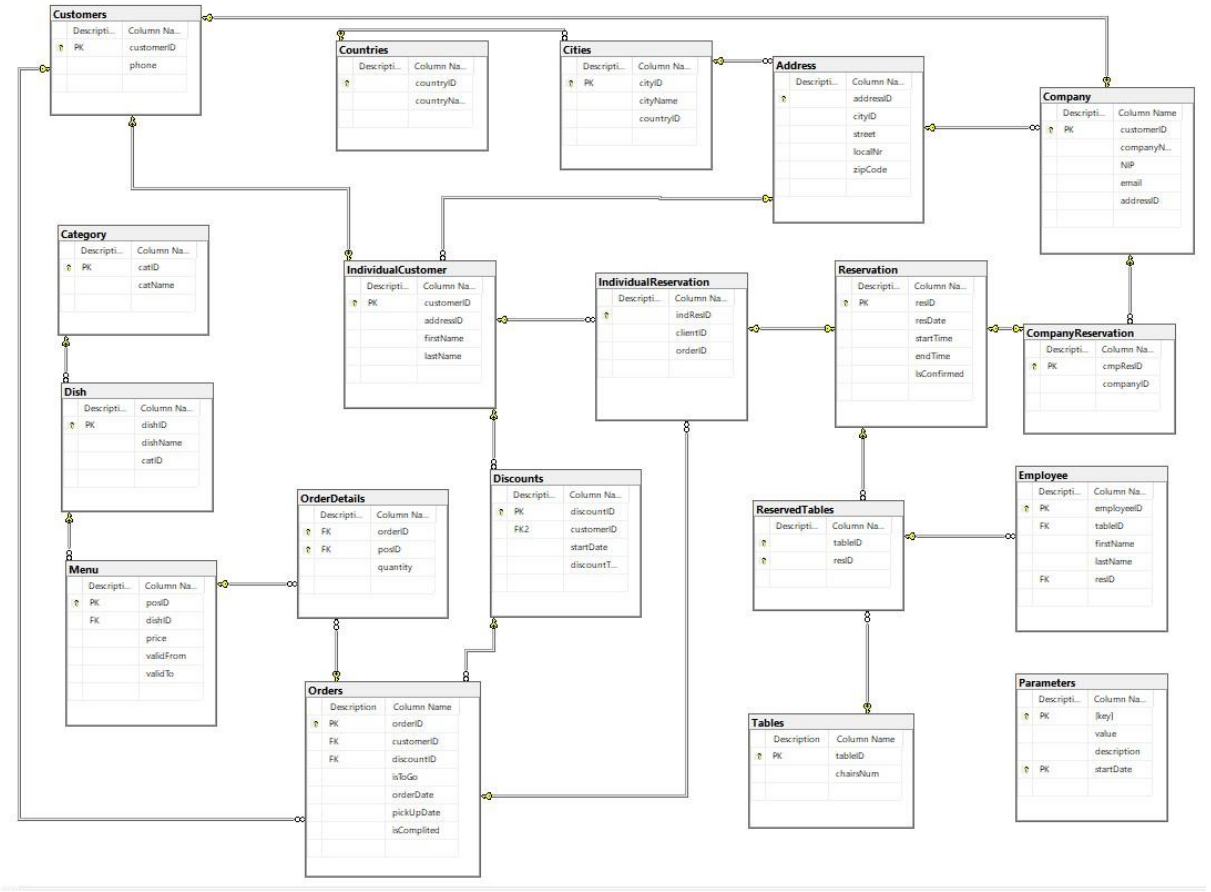
1.6 System

- ➔ Naliczanie rabatów przy zamówieniach.

1.7 Dostawcy

- ➔ Dostęp do listy adresów zamówień na wynos

2. Schemat bazy danych



3. Opisy tabel

3.1 Tabela Address

Tabela Address przechowuje informacje o adresach klientów restauracji - zarówno indywidualnych, jak i firm.

- ☐ klucz główny: addressID
- ☐ klucz obcy: cityID (*do tabeli Cities*)

nazwa kolumny	typ danych	czy null	co przechowuje
addressID	int	not null	numer identyfikacyjny adresu
cityID	int	not null	numer identyfikacyjny miasta
street	nvarchar(50)	not null	nazwa ulicy
localNr	nchar(10)	not null	numer lokalu
zipCode	nchar(10)	not null	kod pocztowy

Warunki integralnościowe:

- Pierwszy znak localNr nie może być literą.
- zipCode składa się z cyfr

```
create table Address
(
    addressID int          not null
        constraint PK_Address
            primary key,
    cityID    int          not null
        constraint FK_Address_Cities
            references Cities,
    street    nvarchar(50) not null,
    localNr   nchar(10)    not null
        check ([localNr] like '[0-9]%'),
    zipCode   nchar(10)    not null
        check (isnumeric([zipCode]) = 1)
)
go
```

3.2 Tabela Category

Tabela Category zawiera informacje o kategoriach, do których należą dania z tabeli Dish.

- ☐ klucz główny: catID

nazwa kolumny	typ danych	czy null	co przechowuje
catID	int	not null	numer identyfikacyjny kategorii
catName	nvarchar(50)	not null	nazwa kategorii

Warunki integralnościowe:

- catName jest unikalne

```
create table Category
(
   catID   int          not null
       constraint PK_Category
           primary key,
   catName nvarchar(50) not null
)
go

create unique index Category_catName_uindex
   on Category (catName)
go
```

3.3 Tabela Cities

Tabela Cities jest słownikiem przechowującym informacje o miastach.

- ☐ klucz główny: cityID
- ☐ klucz obcy: countryID (*do tabeli Countries*)

nazwa kolumny	typ danych	czy null	co przechowuje
cityID	int	not null	numer identyfikacyjny miasta
cityName	nvarchar(50)	not null	nazwa miasta
countryID	int	null	numer identyfikacyjny kraju, w którym znajduje się miasto

Warunki integralnościowe:

- cityName jest unikalne

```
create table Cities
(
   cityID    int       not null
       constraint PK_Cities
           primary key,
   cityName  nchar(50) not null,
   countryID int
       constraint FK_Cities_Countries1
```

```
            references Countries
)
go
create unique index Cities_cityName_uindex
   on Cities (cityName)
go
```

3.4 Tabela Customers

Tabela Customers zawiera informację o klientach restauracji (indywidualnych oraz o firmach będących klientami).

- ☐ klucz główny: customerID

nazwa kolumny	typ danych	czy null	co przechowuje
customerID	int	not null	numer identyfikacyjny klienta restauracji
phone	nvarchar(12)	null	Numer telefonu

```
create table Customers
(
   customerID int not null
       constraint PK_Clients
           primary key,
   phone      varchar(12)
       check (isnumeric([phone]) = 1)

)
go
```

3.5 Tabela Company

Tabela Company przechowuje informacje odnośnie klientów będących firmą.

- ☐ klucz główny: customerID (*jednocześnie klucz obcy do tabeli Customers*)
- ☐ klucz obcy: addressID (*do tabeli Address*)

nazwa kolumny	typ danych	czy null	co przechowuje
customerID	int	not null	numer identyfikacyjny klienta
companyName	nvarchar(50)	not null	nazwa firmy
NIP	int	not null	numer NIP firmy
email	nvarchar(50)	null	adres email firmy
addressID	int	not null	numer identyfikacyjny adresu siedziby firmy

Warunki integralnościowe:

- companyName jest unikalne dla każdej firmy;
- NIP jest unikalny dla każdej firmy;
- phone może składać się wyłącznie z cyfr i jest unikalny;
- email musi zawierać znak “@” i musi być unikalny dla każdej firmy.

```
create table Company
(
    companyID   int          not null
        constraint PK_Company
            primary key
        constraint FK_Company_Clients
            references Customers,
    companyName nvarchar(50) not null,
    NIP         int          not null,
    phone       varchar(12)
        check (isnumeric([phone]) = 1),
    email       nvarchar(50)
        constraint CK__Company__email__10216507
            check ([email] like '%@%'),
    addressID   int          not null
        constraint FK_Company_Adress
            references Address
)
go
create unique index Company_companyName_uindex
    on Company (companyName)
go

create unique index Company_NIP_uindex
    on Company (NIP)
go

create unique index Company_phone_uindex
    on Company (phone)
go

create unique index Company_email_uindex
    on Company (email)
go
```

3.6 Tabela CompanyReservation

Tabela CompanyReservation dotyczy rezerwacji złożonych przez firmy.

- ☐ klucz główny: cmpResID (*jednocześnie klucz obcy do tabeli Reservation*)

☐ klucz obcy: companyID (*do tabeli Company*)

nazwa kolumny	typ danych	czy null	co przechowuje
cmpResID	int	not null	numer identyfikacyjny rezerwacji firmowej
companyID	int	not null	numer identyfikacyjny firmy

```
create table CompanyReservation
(
    cmpResID  int not null
        constraint PK_CompanyReservation
            primary key
        constraint FK_CompanyReservation_Reservation
            references Reservation,
    companyID int not null
        constraint FK_CompanyReservation_Company
            references Company
)
go
```

3.7 Tabela Countries

Tabela Countries zawiera informacje odnośnie krajów, z których pochodzą zamówienia.

☐ klucz główny: countryID

nazwa kolumny	typ danych	czy null	co przechowuje
countryID	int	not null	numer identyfikacyjny kraju
countryName	nvchar(50)	not null	nazwa kraju

Warunki integralnościowe:

- countryName musi być unikalne

```
create table Countries
(
    countryID   int          not null
        constraint PK_Countries
            primary key,
    countryName nvarchar(50) not null
)
go

create unique index Countries_countryName_uindex
    on Countries (countryName)
go
```

3.8 Tabela Discount

Tabela Discount zawiera informacje o rabatach.

- ☐ klucz główny: Para (discountID,customerID)
- ☐ klucze obce: discountID (*do tabeli Discount*), customerID (*do tabeli Customers*)

nazwa kolumny	typ danych	czy null	co przechowuje
discountID	int	not null	Numer identyfikacyjny zniżki
customerID	int	not null	Numer identyfikacyjny klienta, któremu tę zniżkę przyznano
startDate	date	not null	Początek obowiązywania danego kuponu dla klienta
discountType	nvarchar	not null	Informacja czy zniżka jest ograniczona, czy nieograniczona czasowo

Warunki integralnościowe:

- discountType może być tylko “unlimited” lub “limited”

```
create table Discounts
(
   discountID int  not null
       constraint Discounts_pk
           primary key nonclustered
       constraint PK_ActiveDiscount
           unique,
   customerID int  not null
       constraint FK_Discounts_IndividualCustomer
           references IndividualCustomer,
   startDate  date not null,
   constraint CK__ActiveDiscount__11158940
       check (discountType in ('unlimited', 'limited'))
)
go
```

3.9 Tabela Dish

Tabela Dish przechowuje informacje o daniach.

- ☐ klucz główny: dishID
- ☐ klucz obcy: catID (*do tabeli Category*)

nazwa kolumny	typ danych	czy null	co przechowuje
dishID	int	not null	Numer identyfikacyjny dania
dishName	nvarchar(50)	not null	Nazwa dania
catID	int	not null	Numer kategorii do której należy danie

Warunki integralnościowe:

- dishName musi być unikalne

```
create table Dish
(
   dishID   int          not null
       constraint PK_Dish
           primary key,
   dishName nvarchar(50) not null,
   catID    int          not null
       constraint FK_Dish_Category
           references Category
)
go
create unique index Dish_dishName_uindex
   on Dish (dishName)
go
```

3.10 Tabela Employee

Tabela Employee przechowuje informacje o pracownikach firm, dla których te firmy złożyły rezerwacje imienne.

- ☐ klucz główny: employeeID
- ☐ klucz obcy: resID (*do tabeli CompanyReservation*)

nazwa kolumny	typ danych	czy null	co przechowuje
employeeID	Int	not null	Numer identyfikacyjny pracownika
resID	int	not null	Numer rezerwacji, do której przypisany jest pracownik
firstName	nvarchar(10)	not null	Imię pracownika
lastName	nvarchar(50)	not null	Nazwisko pracownika

```
create table Employee
(
   employeeID int          not null
       constraint PK_Employee
           primary key,
   resID      int          not null
       constraint FK_Employee_CompanyReservation
           references CompanyReservation,
   firstName  nchar(10)    not null,
   lastName   nvarchar(50) not null
)
go
```

3.11 Tabela IndividualCustomer

Tabela IndividualCustomer przechowuje informacje o klientach indywidualnych.

- ☐ klucz główny: customerID (*jednocześnie klucz obcy do tabeli Customers*)
- ☐ klucz obcy: addressID (*do tabeli Address)*

nazwa kolumny	typ danych	czy null	co przechowuje
customerID	Int	not null	Numer identyfikacyjny klienta
addressID	Int	not null	Numer identyfikacyjny adresu klienta
firstName	nvarchar(50)	not null	imię klienta
lastName	nvarchar(50)	not null	nazwisko klienta

Warunki integralnościowe:

- phone składa się z cyfr
- phone jest unikalny

```
create table IndividualCustomer
(
   customerID int not null
       constraint PK_IndividualCustomer
           primary key
       constraint FK_IndividualCustomer_Clients
           references Customers,
   addressID  int not null
       constraint FK_IndividualCustomer_Adress
           references Address,
   firstName  nvarchar(50)not null,
   lastName   nvarchar(50),not null
)
go
create unique index IndividualCustomer_phone_uindex
   on IndividualCustomer (phone)
go
```

3.12 Tabela IndividualReservation

Tabela IndividualReservation zawiera informacje o rezerwacjach złożonych przez klientów indywidualnych.

- ☐ klucz główny: indResID (*jednocześnie klucz obcy do tabeli Reservation*)
- ☐ klucze obce: clientID (*do tabeli IndividualCustomer*), tableID (*do tabeli Tables*), orderID (*do tabeli Orders*)

nazwa kolumny	typ danych	czy null	co przechowuje
indResID	int	not null	Numer identyfikacyjny rezerwacji
clientID	int	not null	Numer klienta indywidualnego
tableID	int	not null	Numer zarezerwowanego stolika
orderID	int	not null	Numer złożonego zamówienia

```
create table IndividualReservation
(
   indResID int not null
       constraint PK_IndividualReservation
           primary key
       constraint FK_IndividualReservation_Reservation
           references Reservation,
   clientID int not null
       constraint FK_IndividualReservation_IndividualCustomer
           references IndividualCustomer,
   tableID  int not null
       constraint FK_IndividualReservation_Tables1
           references Tables,
   orderID  int not null
       constraint FK_IndividualReservation_Orders
           references Orders
)
go
```

3.13 Tabela Menu

Tabela Menu zawiera szczegółowe informacje o wszystkich pozycjach w menu, ich ceny, daty obowiązywania.

- ☐ klucz główny: posID
- ☐ klucz obcy: dishID (*do tabeli Dish*)

nazwa kolumny	typ danych	czy null	co przechowuje
posID	int	not null	numer ID pozycji w menu
dishID	int	not null	numer ID dania
price	money	not null	cena
validFrom	date	not null	od kiedy danie jest dostępne
validTo	date	null	do kiedy danie jest dostępne

Warunki integralnościowe:

- price musi być większe od 0
- validFrom musi być wcześniej niż validTo lub validTo jest Null

- validTo może przyjmować wartość null, jeśli danie jest aktualnie dostępne

```
create table Menu
(
   posID      int    not null
       constraint PK_Menu
           primary key,
   dishID     int    not null
       constraint FK_Menu_Dish1
           references Dish,
   price      money not null
       check ([PRICE] > 0),
   validFrom date   not null,
   validTo    date,
   check ([validfrom] < [validTO] OR [validTo] IS NULL)
)
go
```

3.14 Tabela OrderDetails

Tabela OrderDetails zawiera szczegóły dotyczące zamówień.

- ☐ klucz główny: Para (orderID,posID)
- ☐ klucze obce: orderID (*do tabeli Orders*), posID (*do tabeli Menu*)

nazwa kolumny	typ danych	czy null	co przechowuje
orderID	int	not null	numer ID zamówienia
posID	int	not null	numer ID pozycji w menu
quantity	int	not null	zamówiona ilość

Warunki integralnościowe:

- quantity musi być większe od 0

```
create table OrderDetails
(
   orderID  int not null
       constraint FK_OrderDetails_Orders
           references Orders,
   posID    int not null
       constraint FK_OrderDetails_Menu1
           references Menu,
   quantity int not null
       check ([quantity] > 0),
   constraint PK_OrderDetails
       primary key (orderID, posID)
)
go
```

3.15 Tabela Orders

Tabela Orders przechowuje informacje o zamówieniach.

- ☐ klucz główny: orderID
- ☐ klucze obce: customerID (*do tabeli Customers*), discountID (*do tabeli Discount*)

nazwa kolumny	typ danych	czy null	co przechowuje
orderID	int	not null	numer ID zamówienia
customerID	int	not null	numer ID klienta
discountID	int	null	numer ID rabatu
isToGo	bit	not null	informacja czy zamówienie jest na wynos czy nie
orderDate	datetime	null	data złożenia zamówienia
pickUpDate	datetime	null	data odbioru zamówienia
isCompleted	bit	not null	informacja, czy zamówienie jest zakończone

Warunki integralnościowe:

- isToGo - przyjmuje wartość 1, jeśli zamówienie jest na wynos i wartość 0, jeśli zamówienie jest realizowane na miejscu
- orderDate jest wcześniej niż pickUpDate
- discountID jest nullem jeśli nie przyznano żadnego rabatu
- orderDate przyjmuje domyślnie dzisiejszą datę, jeśli nie wprowadzono inaczej
- pickupDate jest Null jeśli zamówienie jest na miejscu

```
create table Orders
(
   orderID    int                          not null
       constraint PK_Orders
           primary key,
   customerID int                          not null
       constraint FK_Orders_Clients1
           references Customers,
   discountID int
       constraint FK_Orders_Discounts
           references Discounts,
   isToGo     bit                          not null,
   orderDate  datetime default getdate() null,
   pickUpDate datetime                     null,
   check ([orderDate] < [pickUpDate])
)
go
```

3.16 Tabela Parameters

Tabela Parameters przechowuje informacje odnośnie wskaźników (np. WK, Z1, K1, D1).

- ☐ klucz główny: [key]

nazwa kolumny	typ danych	czy null	co przechowuje
[key]	varchar(50)	not null	klucz parametru
value	int	not null	wartość wskaźnika
description	nchar(50)	null	opis parametru
startDate	date	not null	data rozpoczęcia

Warunki integralnościowe:

- value musi być większe od zera
- description przyjmuje wartość NULL, jeśli nikt go nie wprowadzi (defaultowa wartość)
- [key] to jedna z wartości : `('D1','R1','R2','K1','K2','Z1','WZ','WK')`

```
create table Parameters
(
   [key]       varchar(50) not null
       constraint Parameters_pk
           primary key nonclustered
       check ([key] = 'WK' OR [key] = 'WZ' OR [key] = 'Z1' OR
[key] = 'K2' OR [key] = 'K1' OR [key] = 'R2' OR
               [key] = 'R1' OR [key] = 'D1'),
   value       int         not null
       check ([value] > 0),
   description nchar(50) default NULL,
   startDate   date        not null)
go
```

3.17 Tabela Reservation

Tabela Reservation przechowuje informacje odnośnie rezerwacji stolików przez klientów indywidualnych oraz klientów-firm.

- ☐ klucz główny: resID

nazwa kolumny	typ danych	czy null	co przechowuje
resID	int	not null	numer ID rezerwacji
orderID	int	not null	numer ID zamówienia
resDate	date	not null	data rezerwacji
startTime	time	not null	godzina rozpoczęcia rezerwacji
endTime	time	not null	godzina zakończenia rezerwacji

IsConfirmed	bit	not null	czy rezerwacja została potwierdzona

Warunki integralnościowe:

- startTime musi być wcześniej niż endTime
- resDate musi być co najmniej na ten sam dzień (nie można złożyć rezerwacji na dzień, który już był)

```
create table Reservation
(
   resID     int  not null
       constraint PK_Reservation
           primary key,
   orderID   int  not null
       constraint FK_Reservation_Orders
           references Orders,
   resDate   date not null
       check ([resDate] >= getdate()),
   startTime time not null,
   endTime   time not null,
   check ([startTime] < [endTime])
)
go
```

3.18 Tabela Tables

Tabela Tables przechowuje informacje na temat stolików w restauracji:

- ☐ klucz główny: tableID
- ☐ klucz obcy: resID (*do tabeli CompanyReservation lub Reservation - w zależności czy rezerwacji dokonała firma, czy klient indywidualny*)

nazwa kolumny	typ danych	czy null	co przechowuje
tableID	int	not null	numer ID stolika
chairsNum	int	not null	liczba krzeseł przy stoliku

Warunki integralnościowe:

- chairsNum większe od 0

```
create table Tables
(
   tableID   int not null
       constraint PK_Tables
           primary key,
   chairsNum int not null
       check ([chairsNum] > 0),
   resID     int
       constraint FK_Tables_CompanyReservation
```

```
            references CompanyReservation
        constraint FK_Tables_Reservation
            references Reservation
)
go
```

3.19 Tabela ReservedTables

Tabela Tables przechowuje informacje na temat stolików w restauracji:

- ☐ klucz główny i obcy: tableID (*do tabeli Tables*)
- ☐ klucz główny i obcy: resID (*do tabeli Reservation*)

nazwa kolumny	typ danych	czy null	co przechowuje
resID	int	not null	numer ID rezerwacji
tableID	int	not null	numer ID stolika

Warunki integralnościowe:

- chairsNum większe od 0

```
create table ReservedTables
(
   resID int not null,
      constraint PK_Reservation
          primary key,
   tableID   int not null
       constraint PK_Tables
           primary key,
   chairsNum int not null
       check ([chairsNum] > 0),
   resID     int
       constraint FK_ReservedTables_Tables
           references Tables
       constraint FK_ReservedTables_Reservation
           references Reservation
)
go
```

4. Widoki

4.1 Widok CurrentMenu

Przedstawia aktualnie dostępne w Menu dania wraz z cenami:

```
create view CurrentMenu as
   SELECT catName, dishName, price
```

```
   FROM MENU M
   INNER JOIN Dish D on D.dishID = M.dishID
   INNER JOIN Category C on C.catID = D.catID
   WHERE  validFrom <= GETDATE() AND (validTo >= GETDATE() OR
validTo IS NULL)
go
```

4.2 Widok CompanyGuestsList

Przedstawia listę ID rezerwacji, imion i nazwisk pracowników firmy, nazwę firmy oraz numer stolika.

```
create view CompanyGuestsList as
   SELECT R.resID, C.companyName, E.firstName + ' ' + E.lastName
AS NAME, T.tableID
   FROM Employee E
   INNER JOIN CompanyReservation CR on CR.cmpResID = E.resID
   INNER JOIN Company C on C.customerID = CR.companyID
   INNER JOIN Reservation R on R.resID = CR.cmpResID
   INNER JOIN Tables T on R.resID = T.resID
go
```

4.3 Widok OrderTotalValues

Dla każdego klienta przedstawia sumę pieniędzy, którą musi zapłacić za dane zamówienie.

```
create view OrderTotalValues as
SELECT O.orderID,C.customerID,SUM(OD.quantity * M.price) *
(1-ISNULL(
    (SELECT TOP 1 CAST(P.value AS FLOAT)/100
    FROM Parameters P
    WHERE p.[key] =
       (CASE
          WHEN D.discountType = 'limited' THEN 'R1'
          ELSE 'R2'
       END)
    AND P.startDate <= D.startDate
    ORDER BY P.startDate DESC),0)) AS TotalValue
 FROM Customers C
 INNER JOIN Orders O ON C.customerID = O.customerID
 INNER JOIN OrderDetails OD ON O.orderID = OD.orderID
 INNER JOIN Menu M ON OD.posID = M.posID
 LEFT JOIN Discounts D ON O.discountID = D.discountID
 GROUP BY C.customerID, O.orderID, D.startDate, D.discountType
go
```

4.4 Widok DishesRank

Widok przedstawia dla każdego dania ilość zamówień w przeciągu ostatnich dwóch tygodni. (co najmniej raz na dwa tygodnie trzeba aktualizować menu, więc można sprawdzić na przykład, które danie jest zamawiane najrzadziej).

```
CREATE VIEW DishesRank as
      SELECT dishName, COUNT(O.OrderID) as OrdersQuantity
      FROM ORDERS O
      INNER JOIN OrderDetails OD on O.orderID = OD.orderID
      INNER JOIN Menu M on M.posID = OD.posID
      INNER JOIN Dish D on D.dishID = M.dishID
      WHERE DATEDIFF(DAY,GETDATE(),orderDate) <= 14
      GROUP BY M.dishID,dishName
go
```

4.5 Widok TodaysReservations

Widok przedstawia informacje odnośnie rezerwacji zamówionych na dzień dzisiejszy (informacja dla pracowników, którzy muszą wcześniej przygotować stoły).

```
CREATE VIEW TodaysReservations AS
   SELECT R.resID,R.startTime,
IIF(cmpResID IS NULL,'INDIVIDUAL','COMPANY') AS RESERVATIONTYPE
   FROM Reservation R
   LEFT JOIN CompanyReservation CR on R.resID = CR.cmpResID
   WHERE resDate = CONVERT(DATE, GETDATE())
go
```

4.6 Widok TakeoutOrdersAdresses

Widok przedstawia listę zamówień na wynos wraz z informacją pod jaki adres mają być one dostarczone i kiedy(informacja dla osób dowożących jedzenie).

```
CREATE VIEW TakeoutOrdersAddresses AS
   SELECT orderID, countryName,
cityName + ' ' + street + ' ' + localNr AS ADDRESS, pickUpDate
   FROM ORDERS O
   INNER JOIN Customers C on C.customerID = O.customerID
   INNER JOIN IndividualCustomer IC on C.customerID =
IC.customerID
   INNER JOIN Address A on A.addressID = IC.addressID
   INNER JOIN Cities C2 on C2.cityID = A.cityID
   INNER JOIN Countries C3 on C2.countryID = C3.countryID
   WHERE O.isToGo = 1
   UNION
```

```
   SELECT orderID, countryName,
cityName + ' ' + street + ' ' + localNr AS ADDRESS, pickUpDate
   FROM ORDERS O1
   INNER JOIN Customers C4 on C4.customerID = O1.customerID
   INNER JOIN Company C5 on C4.customerID = C5.customerID
   INNER JOIN Address A2 on C5.addressID = A2.addressID
   INNER JOIN Cities C6 on C6.cityID = A2.cityID
   INNER JOIN Countries C7 on C6.countryID = C7.countryID
   WHERE O1.isToGo = 1
go
```

4.7 Widok ClientDiscount

Pokazuje jakie zniżki są przyznane poszczególnym klientom.

```
CREATE VIEW ClientDiscount as
   SELECT C.customerID, (CASE WHEN D.discountType = 'limited' THEN
'R1' ELSE 'R2' END) AS TYPE
   FROM Customers C
   INNER JOIN Orders O on C.customerID = O.customerID
   INNER JOIN OrderDetails OD on O.orderID = OD.orderID
   INNER JOIN Discounts D on D.discountID = O.discountID
go
```

4.8 Widok CurrentParameters

Przedstawia aktualne wartości parametrów.

```
CREATE VIEW CurrentParameters AS
   SELECT [KEY], VALUE
   FROM Parameters P
   WHERE  P.startDate <= GETDATE()
go
```

4.9 Widok NonCompletedOrders

Przedstawia listę zamówień które jeszcze nie zostały zrealizowane.

```
CREATE VIEW NonCompletedOrders AS
   SELECT orderID
   FROM Orders
   WHERE isComplited = 0;
go
```

4.10 Widok NumberOfOrders

Przedstawia liczbę złożonych zamówień dla każdego klienta.

```
CREATE VIEW NumberOfOrders AS
   SELECT C.customerID, COUNT(orderID) AS NUMBER_OF_ORDERS
   FROM Customers C
   INNER JOIN Orders O on C.customerID = O.customerID
   GROUP BY C.customerID
go
```

4.11 Widok UnconfirmedReservations

Przedstawia listę rezerwacji, które oczekują na potwierdzenie.

```
create view UncomfirmedReservations AS
   SELECT resID, resDate
   FROM Reservation R
   WHERE IsConfirmed = 0
go
```

4.12 Widok ParameterDates

Przedstawia listę aktualnych obecnie Parametrów z pełnymi datami obowiązywania.

```
CREATE VIEW ParametersDates AS
  SELECT P.[key], P.value, P.startDate, ISNULL(MIN(P2.startDate),
GETDATE()) AS 'endDate'
  FROM Parameters P
  LEFT JOIN Parameters P2 ON P.[key] = P2.[key] AND P2.startDate >
P.startDate
  GROUP BY P.[key], P.startDate, P.value
go
```

4.13 Widok TODAYSORDERS

Przedstawia numery identyfikacyjne zamówień na dzisiejszy dzień.

```
CREATE VIEW TODAYSORDERS AS
   SELECT orderID FROM Orders WHERE pickUpDate = GETDATE()
go
```

5. Indeksy

5.1 Customers

- ix_customer_name - indeks wykorzystujący dane kluczowe do identyfikacji klienta: jego nazwisko i imię.

```
CREATE INDEX ix_customers_name ON Customers(lastName, firstName);
```

5.2 Orders

- ix_orders_dates - indeksowanie dat złożenia i odebrania zamówienia przez klienta.

```
CREATE INDEX ix_orders_dates ON Orders(orderDate, pickUpDate);
```

5.3 Menu

- ix_menu_dates - indeks wspomagający wyszukiwanie dań w menu w danym przedziale czasowym, w szczególności dań z aktualnego menu.

```
CREATE INDEX ix_menu_dates ON Menu(validFrom, validTo);
```

- ix_menu_prices - indeksowanie cen dań w celu szybszego znajdowania dań w określonym zakresie cenowym.

```
CREATE INDEX ix_menu_prices ON Menu(Price);
```

5.4 OrderDetails

- ix_orderDetails_orders - indeksowanie ID zamówień w tabeli OrderDetails, dzięki czemu można szybciej uzyskać informacje o zamówionych pozycjach dla wybranego zamówienia.

```
CREATE INDEX ix_orderDetails_orders ON OrderDetails(orderID,
posID);
```

5.5 Discounts

- ix_discounts_dates - indeks początku obowiązywania zniżki.

```
CREATE INDEX ix_discounts_dates ON Discounts(startDate);
```

5.6 Reservation

- ix_reservation_dates - indeks daty i godziny rezerwacji.

```
CREATE INDEX ix_reservation_dates ON  Reservation(resDate,
startTime, endTime);
```

5.7 Parameters

- ix_parameters_dates - indeks początków obowiązywania parametrów, pozwalający wyszukać najnowsze dane.

```
CREATE INDEX ix_parameters_dates ON Parameters(startDate);
```

6. Procedury

6.1 Procedura GetCityId

```
CREATE PROCEDURE GetCityId(
  @CityName nvarchar(50),
  @CountryName nvarchar(50),
  @CityID INT OUTPUT
) AS
BEGIN
  DECLARE @CountryID INT =
      (SELECT countryID
      FROM Countries
      WHERE countryName = @CountryName);

  IF @@ROWCOUNT = 0

  BEGIN
      INSERT INTO Countries(countryName)
      VALUES (@CountryName)
      SET @CountryID = SCOPE_IDENTITY()
  END


  SELECT @CityID =
      (SELECT cityId
      FROM Cities
      WHERE cityName = @CityName AND countryID = @CountryID)

  IF @@ROWCOUNT = 0
  BEGIN
      INSERT INTO Cities(cityName, countryID)
      VALUES (@CityName, @CountryID)
      SET @CityID = SCOPE_IDENTITY()
  END
END
GO
```

6.2 Procedura AddIndividualCustomer

Procedura dodaje do bazy danych klienta indywidualnego wraz z informacjami o nim.

```
CREATE PROCEDURE AddIndividualCustomer(
  @Phone nvarchar(12),
  @FirstName nvarchar(50),
  @LastName nvarchar(50),
  @City nvarchar(50),
  @Country nvarchar(50),
  @Street nvarchar(50),
  @LocalNr nchar(10),
  @ZipCode nchar(10)
)
AS BEGIN
  INSERT INTO Customers(Phone)
  VALUES (@Phone);

  DECLARE @CustomerId INT = SCOPE_IDENTITY()

  DECLARE @CityID INT;

  EXEC GetCityId @CityName=@City, @CountryName=@Country,
@CityID=@CityID

  INSERT INTO Address(cityID, street, localNr, zipCode)
  VALUES(@CityID, @Street, @LocalNr, @ZipCode);

  DECLARE @AddressId INT = SCOPE_IDENTITY()

  INSERT INTO IndividualCustomer(customerID, firstName, lastName,
addressID)
  VALUES(@CustomerId, @FirstName, @LastName, @AddressId);
END
GO
```

6.3 Procedura AddCompany

Procedura wprowadza do bazy danych informacje o firmie.

```
CREATE PROCEDURE AddCompany(
  @Phone nvarchar(12),
  @Name nvarchar(50),
  @NIP int,
  @Email nvarchar(50),
  @City nvarchar(50),
  @Country nvarchar(50),
  @Street nvarchar(50),
  @LocalNr nchar(10),
  @ZipCode nchar(10)
)
AS BEGIN
  INSERT INTO Customers(Phone)
  VALUES (@Phone)
```

```
  DECLARE @CustomerId INT = SCOPE_IDENTITY()

  DECLARE @CityID INT;

  EXEC GetCityId @CityName=@City, @CountryName=@Country,
@CityID=@CityID


  INSERT INTO Address(cityID, street, localNr, zipCode)
  VALUES(@CityID, @Street, @LocalNr, @ZipCode);

  DECLARE @AddressId INT = SCOPE_IDENTITY()

  INSERT INTO Company(customerID, companyName, NIP, email,
addressID)
  VALUES(@CustomerId, @Name, @NIP, @Email, @AddressId);
END
GO
```

6.4 Procedura ChangeChairsNum

Procedura zmienia liczbę krzeseł przy stoliku o podanym numerze identyfikacyjnym.

```
CREATE PROCEDURE ChangeChairsNum(
@NewChairsNum int,
@TableNumber int
)
AS BEGIN
   UPDATE Tables SET chairsNum = @NewChairsNum WHERE tableID =
@TableNumber
END
GO
```

6.5 Procedura AddTables

Procedura dodaje nowy stolik do bazy danych, w razie wystąpienia błędu wysyła jego treść.

```
Create PROCEDURE AddTables(
@tableID int,
@NumberOfChairs int,
@ReservationID int = NULL
)
as begin
   begin try
       insert into Tables(tableID, chairsNum, resID)
       values (@tableID,@NumberOfChairs,@ReservationID)
   end try
   begin catch
       select ErrorMessage = ERROR_MESSAGE();
   end catch
end
go
```

6.6 Procedura DeleteTables

Procedura usuwa stolik o podanym numerze identyfikacyjnym z bazy danych.

```
CREATE PROCEDURE DeleteTables(
@tableID int
)
AS BEGIN
   DELETE FROM Tables WHERE tableID = @tableID
end
go
```

6.7 Procedura ConfirmOrCancelReservation

Procedura potwierdza lub odwołuje rezerwację o podanym numerze identyfikacyjnym rezerwacji.

```
create PROCEDURE ConfirmOrCancelReservation(
@reservationNumber int,
@isConfirmed bit
)
as begin
   update Reservation set IsConfirmed = @isConfirmed where resID =
@reservationNumber;
end
go
```

6.8 Procedura AddCountry

Procedura dodaje kraj o podanej nazwie

```
create procedure AddCountry(
@CountryName nvarchar(50)
)
as begin
   insert into Countries(countryName) values (@CountryName);
end
go
```

6.9 Procedura DeleteCountry

Procedura usuwa kraj o podanym numerze ID.

```
create PROCEDURE DeleteCountry(
@countryNumber int
)
as begin
   delete from Countries where countryID = @countryNumber
end
```

```
go
```

6.10 Procedura AddCity

Procedura dodaje miasto o podanej nazwie.

```
create procedure AddCity(
@CityName nvarchar(50),
@COUNTRYNAME NVARCHAR(50)
)
as begin
   DECLARE @CountryID INT = (SELECT countryID FROM Countries WHERE
countryName = @COUNTRYNAME);
   insert into Cities(cityName, countryID) VALUES
(@CityName,@CountryID)
end
go
```

6.11 Procedura DeleteCity

Procedura usuwa miasta o podanym numerze ID.

```
create procedure DeleteCity(
@CityID int
)
as begin
   delete from Cities where cityID = @CityID
end
go
```

6.12 Procedura AddMenu

Procedura dodaje nową pozycję do menu, której wcześniej nie było.

```
create procedure AddMenu(
@CategoryName nvarchar(50),
@Dishname nvarchar(50),
@Price money,
@Validfrom date,
@ValidTo date = NULL
)
as begin
   DECLARE @CategoryID INT  = (SELECT catID FROM Category WHERE
catName = @CategoryName)
   IF @CategoryID IS NULL
   BEGIN
       INSERT INTO Category(catName) VALUES (@CategoryName)
       set @CategoryID = SCOPE_IDENTITY()
   end
   DECLARE @DISHid INT  = (SELECT dishID FROM dish WHERE dishName=
@Dishname)
   IF @DISHid IS NULL
```

```
   BEGIN
       INSERT INTO Dish(dishName, catID) VALUES
(@Dishname,@CategoryID)
       SET @DISHid = SCOPE_IDENTITY()
   end

   DECLARE @posID INT  = (SELECT posID from Menu where dishID =
@DISHid)
   IF @posID IS NULL
   BEGIN
       INSERT INTO Menu(dishID, price, validFrom, validTo) values
(@DISHid,@Price,@Validfrom,@ValidTo)
       SET @posID = SCOPE_IDENTITY()
   end
end
go
```

6.13 Procedura UpdateAvailability

Procedura zmienia dostępność dań na null, jeśli są dostępne, lub na datę zakończenia obowiązywania tej pozycji w menu.

```
create procedure UpdateAvailability(
@posID int,
@Availability date = Null
)
as begin
   update Menu set validTo = @Availability where posID = @posID
end
go
```

6.14 Procedura AddOrder

Procedura wstawia do tabeli Orders nowe zamówienie dla klienta o wskazanym ID.

```
CREATE PROCEDURE AddOrder(
  @CustomerID INT = 0,
  @IsToGo BIT = 0,
  @PickUpDate DATE = NULL
) AS
BEGIN
  DECLARE @DiscountID INT;
  DECLARE @Today DATE = CONVERT(DATE, GETDATE());

  SET @DiscountID = (
     SELECT discountID
     FROM CustomersDiscountExtended
     WHERE customerID = @CustomerID AND startDate <= @Today AND
```

```
            @Today <= ISNULL(endDate, @Today)
            AND value =
                (SELECT MAX(CD.value)
                FROM CustomersDiscountExtended CD
                WHERE CD.customerID = @CustomerID AND CD.startDate
<= @Today AND
                    @Today <= ISNULL(CD.endDate, @Today)
                GROUP BY CD.customerID)
  );

  INSERT INTO Orders(customerID, discountID, isToGo, orderDate,
pickUpDate)
  VALUES (
     @CustomerID, @DiscountID, @IsToGo, @Today, @PickUpDate
  )

END
go
```

6.15 Procedura AddToOrder

Procedura AddToOrder dodaje nowe pozycje do istniejącego zamówienia.

```
CREATE PROCEDURE AddToOrder (
  @OrderID INT,
  @PosID INT,
  @Quantity INT = 1
) AS
BEGIN
  IF @OrderID IN (SELECT OrderId FROM Orders)
  BEGIN
     DECLARE @PrevQuantity INT =
         (SELECT OrderId
        FROM OrderDetails
        WHERE OrderId = @OrderID AND posID = @PosID);
     IF @PrevQuantity IS NOT NULL
     BEGIN
        UPDATE OrderDetails
        SET quantity = (@PrevQuantity + @Quantity)
        WHERE OrderId = @OrderID;
     END
     ELSE INSERT INTO OrderDetails(orderId, posId, quantity)
          VALUES(@OrderID, @PosID, @Quantity)
  END
END
go
```

6.16 Procedura GrantUnlimitedDiscount

Procedura dodaje zniżkę klientowi (nielimitowaną), jeśli spełnił wymagane warunki.

```
@customerID int,
@DAY DATE
)
as begin
   declare @minimumValue float = (SELECT value FROM
ParametersDates WHERE startDate <= @DAY AND endDate>= @DAY

and [key] = 'K1')
   DECLARE @ORDNUMBER INT =(SELECT value FROM ParametersDates
WHERE startDate <= @DAY AND endDate>= @DAY AND [key]='Z1')
   IF dbo.MinValueOrdersNumber(@customerID,@minimumValue) >=
@ORDNUMBER
   BEGIN
       DECLARE @CUSTOMERIDENDITY INT = (SELECT customerID FROM
IndividualCustomer WHERE customerID = @customerID)
       IF @CUSTOMERIDENDITY IS NOT NULL
       BEGIN
          insert into Discounts(customerID, startDate,
discountType) values (@customerID,@DAY,'unlimited')
       END
   end
end
go
```

6.17 Procedura GrantLimitedDiscount

Procedura dodaje zniżkę klientowi (limitowaną), jeśli spełnił wymagane warunki.

```
create procedure GrantLimitedDiscount(
@customerID int,
@DAY DATE
)
as begin
   declare @minimumValue float = (SELECT value FROM
ParametersDates WHERE startDate <= @Day
   AND endDate>= @day and [key] = 'K2')

   IF dbo.AllOrdersValue(@customerID) >= @minimumValue
   BEGIN
       DECLARE @CUSTOMERIDENDITY INT = (SELECT customerID FROM
IndividualCustomer WHERE customerID = @customerID)
       IF @CUSTOMERIDENDITY IS NOT NULL
       BEGIN
          insert into Discounts(customerID, startDate,
discountType) values (@customerID,@DAY,'limited')
       END
   end
```

```
end
go
```

6.18 Procedura AddIndividualReservation

Procedura dodaje do bazy danych rezerwacje dla klienta indywidualnego.

```
CREATE PROCEDURE AddIndividualReservation(
@OrderID INT,
@Date DATE,
@NumOfPeople INT,
@StartTime TIME,
@EndTime TIME = NULL
) AS
BEGIN
  IF @EndTime IS NULL SET @EndTime = DATEADD(HOUR, 4,
@StartTime);
  DECLARE @Today DATE = CONVERT(DATE, GETDATE());
  DECLARE @CustomerID INT;
  SET @CustomerID =
      (SELECT @CustomerID
      FROM IndividualCustomer
      INNER JOIN Orders ON Orders.customerID =
IndividualCustomer.customerID
      WHERE Orders.orderID = @OrderID);
  IF @CustomerID IS NOT NULL
  AND
      (SELECT TotalValue
      FROM OrderTotalValues
      WHERE orderID = @OrderID)
      >
      (SELECT [value]
      FROM ParametersDates
      WHERE [key] = 'WZ' AND startDate <= @Today AND endDate >=
@Today)
  AND
      (SELECT NUMBER_OF_ORDERS
      FROM NumberOfOrders
      WHERE @CustomerID = customerID)
      >=
      (SELECT [value]
      FROM ParametersDates
      WHERE [key] = 'WK' AND startDate <= @Today AND endDate >=
@Today)
  AND
      (SELECT COUNT(*)
```

```
      FROM FreeTableList(@Date, @StartTime, @EndTime)
      WHERE chairsNum >= @NumOfPeople) > 0
  BEGIN
      DECLARE @TableID INT =
             (SELECT TOP 1 tableID
             FROM FreeTableList(@StartTime, @EndTime, @Date)
             WHERE chairsNum =
                   (SELECT MIN(chairsNum)
                   FROM FreeTableList(@StartTime, @EndTime, @Date)
                   WHERE chairsNum >= @NumOfPeople))
       INSERT INTO Reservation(resDate, startTime, endTime,
IsConfirmed)
       VALUES (@Date, @StartTime, @EndTime, 0)

       INSERT INTO IndividualReservation(indResID, clientID,
orderID)
       VALUES(@@IDENTITY, @CustomerID, @OrderID)

      DECLARE @resID INT = (SELECT indResID FROM
IndividualReservation WHERE orderID=@OrderID)
       INSERT INTO ReservedTables(resID, tableID)
       VALUES(@resID, @TableID)
    END
END
go
```

7. Funkcje

7.1 Funkcja FreeTables

Funkcja wyświetla liczbę stolików, które są wolne danego dnia i w podanym przedziale czasowym.

```
CREATE FUNCTION FreeTables(
@Day date,
@startTime time,
@endTime time
)
returns int
as begin
   return (SELECT COUNT(tableID) FROM TABLES WHERE tableID NOT IN
   (SELECT tableID
   FROM Tables
   INNER JOIN Reservation R on R.resID = Tables.resID
```

```
   WHERE resDate=@Day AND
   ((startTime <= @startTime AND endTime >= @endTime)
   OR (startTime >= @startTime AND endTime >= @endTime AND
startTime < @endTime)
   OR (startTime <= @startTime AND endTime <= @endTime AND
@startTime < endTime)
   OR (startTime > @startTime AND endTime < @endTime))));
end
go
```

7.2 Funkcja FreeTableList

Funkcja zwraca w postaci listy numery identyfikacyjne stolików, które danego dnia w podanym przedziale czasowym są wolne i można je zarezerwować.

```
create function FreeTableList(
@startTime time,
@endTime time,
@Day date
)
returns table as return
SELECT tableID,chairsNum FROM TABLES WHERE tableID NOT IN
  (SELECT tableID
  FROM Tables
   INNER JOIN Reservation R on R.resID = RT.resID
   INNER JOIN Tables T ON RT.tableID = T.tableID
  WHERE resDate=@Day AND
  ((startTime <= @startTime AND endTime >= @endTime)
  OR (startTime >= @startTime AND endTime >= @endTime AND
startTime < @endTime)
  OR (startTime <= @startTime AND endTime <= @endTime AND
@startTime < endTime)
  OR (startTime > @startTime AND endTime < @endTime)))
go
```

7.3 Funkcja OrderValue

Funkcja zwraca wartość zamówienia o podanym identyfikatorze.

```
CREATE FUNCTION OrderValue(
@OrderIdentityNumber int
)
returns float
as begin
   return (SELECT SUM(OD.quantity * M.price) * (1-ISNULL(
    (SELECT TOP 1 CAST(P.value AS FLOAT)/100
    FROM Parameters P
    WHERE p.[key] =
       (CASE
          WHEN D.discountType = 'limited' THEN 'R1'
          ELSE 'R2'
```

```
        END)
    AND P.startDate <= D.startDate
    ORDER BY P.startDate DESC),0)) AS TotalValue
 FROM Customers C
 INNER JOIN Orders O ON C.customerID = O.customerID
 INNER JOIN OrderDetails OD ON O.orderID = OD.orderID
 INNER JOIN Menu M ON OD.posID = M.posID
 LEFT JOIN Discounts D ON O.discountID = D.discountID
 where O.orderID = @OrderIdentityNumber
 GROUP BY C.customerID, O.orderID, D.startDate, D.discountType)
end
go
```

7.4 Funkcja HowManyDishOrders

Funkcja dla danego dania zwraca ilość jego zamówień w przeciągu ostatnich dwóch tygodni.

```
CREATE FUNCTION HowManyDishOrders(
@DishName nvarchar(50)
)
returns int
as begin
   declare @quantity int = (SELECT COUNT(O.OrderID) as
OrdersQuantity
   FROM ORDERS O
   INNER JOIN OrderDetails OD on O.orderID = OD.orderID
   INNER JOIN Menu M on M.posID = OD.posID
   INNER JOIN Dish D on D.dishID = M.dishID
   WHERE DATEDIFF(DAY,GETDATE(),orderDate) <= 14 and dishName =
@DishName
   GROUP BY M.dishID,dishName)
   return isnull(@quantity,0)
end
go
```

7.5 Funkcja GenerateInvoice

Funkcja dla danego zamówienia generuje fakturę w postaci tabeli - szczegóły zamówienia (co zamówiono, w jakiej ilości, ile kosztowało przed zniżką, czy przyznano zniżkę, ile kosztowało ze zniżką).

```
create function GenerateInvoice(
@OrderNumber int
)
returns TABLE as
   return SELECT D1.dishName, OD.quantity, M.price AS PriceForOne,
(OD.quantity * M.price) AS ValueWithoutDiscount,ISNULL(
    (SELECT TOP 1 CAST(P.value AS FLOAT)/100
    FROM Parameters P
    WHERE p.[key] =
```

```
        (CASE
           WHEN D.discountType = 'limited' THEN 'R1'
           ELSE 'R2'
        END)
    AND P.startDate <= D.startDate
    ORDER BY P.startDate DESC),0) AS DISCOUNT,(OD.quantity *
M.price) * (1-ISNULL(
    (SELECT TOP 1 CAST(P.value AS FLOAT)/100
    FROM Parameters P
    WHERE p.[key] =
        (CASE
           WHEN D.discountType = 'limited' THEN 'R1'
           ELSE 'R2'
        END)
    AND P.startDate <= D.startDate
    ORDER BY P.startDate DESC),0)) AS ValueWithDiscount
 FROM Customers C
 INNER JOIN Orders O ON C.customerID = O.customerID
 INNER JOIN OrderDetails OD ON O.orderID = OD.orderID
 INNER JOIN Menu M ON OD.posID = M.posID
 INNER JOIN Dish D1 on D1.dishID = M.dishID
 LEFT JOIN Discounts D ON O.discountID = D.discountID
 WHERE O.orderID = @OrderNumber
 GROUP BY C.customerID, O.orderID, D.startDate,
D.discountType,quantity,price,M.posID,dishName
go
```

7.6 Funkcja AllOrdersValue

Funkcja zwraca wartość wszystkich zamówień, jakie do tej pory złożył klient.

```
create function AllOrdersValue(
@CustomerIdentityNumber int
)
returns float
as begin
   return (select sum(dbo.OrderValue(orderID))
from Customers C
inner join Orders O on C.customerID = O.customerID
where C.customerID=@CustomerIdentityNumber)
end
go
```

7.7 Funkcja MinValueOrders

Funkcja zwraca liczbę zamówień dla danego klienta, które kosztowały więcej lub tyle samo co podana liczba.

```
create function MinValueOrdersNumber(
@CustomerIdentityNumber int,
@MinValue float
```

```
)
returns int
as begin
   return (select count(*)
from OrderTotalValues
where customerID = @CustomerIdentityNumber and TotalValue >=
@MinValue)
end
go
```

7.8 Funkcja NumberOfReservations

Funkcja zwraca tabelę przedstawiającą dla danego roku miesiące oraz liczbę rezerwacji w danym miesiącu.

```
CREATE FUNCTION NumberOfReservations(
@Year int
)
returns table as return
(SELECT MONTH(resDate) as Month, COUNT(*) AS NumberOfReservations
from Reservation
where year(resDate) = @Year
group by MONTH(resDate))
go
```

7.9 Funkcja MonthlyIncome

Funkcja zwraca łączną sumę wartości zamówień w danym miesiącu i roku.

```
create function MonthlyIncome(
@Year int,
@month int
)
returns float
as begin
   return (select sum(TotalValue)
from OrderTotalValues
inner join Orders O on OrderTotalValues.customerID = O.customerID
where month(orderDate)= @month and year(orderDate)= @Year)
end
go
```

7.10 Funkcja OrderContent

Zawiera zawartość zamówienia o danym numerze identyfikacyjnym czyli to, jakie danie zostało zamówione, w jakiej ilości i na kiedy).

```
create function OrderContent(
@OrderNumber int
)
```

```
returns TABLE as
 return SELECT D1.dishName, OD.quantity, O.pickUpDate
 FROM Customers C
 INNER JOIN Orders O ON C.customerID = O.customerID
 INNER JOIN OrderDetails OD ON O.orderID = OD.orderID
 INNER JOIN Menu M ON OD.posID = M.posID
 INNER JOIN Dish D1 on D1.dishID = M.dishID
 LEFT JOIN Discounts D ON O.discountID = D.discountID
 WHERE O.orderID = @OrderNumber
 GROUP BY C.customerID, O.orderID, D.startDate,
D.discountType,quantity,price,M.posID,dishName, O.pickUpDate
go
```

8. Triggery

8.1 Trigger TablesOnDelete

Nie można usunąć stolika, jeżeli jest on zarezerwowany w przyszłości.

```
CREATE TRIGGER TablesOnDelete ON ReservedTables
AFTER DELETE, UPDATE AS
BEGIN
   SET NOCOUNT ON;
  IF (SELECT COUNT(*)
     FROM deleted D
     INNER JOIN Reservation R ON D.resID = R.resID
     WHERE R.resDate >= CONVERT(DATE, GETDATE())) > 0
  BEGIN
     ROLLBACK;
       RAISERROR ('The table cannot be removed as it is reserved',
17, -1);
  END
END;
GO
```

8.2 Trigger ReservedTablesOnInsert

Nie można dodać stolika do rezerwacji indywidualnej, jeśli już jakiś jest.

```
CREATE TRIGGER ReservedTablesOnInsert ON ReservedTables
AFTER INSERT, UPDATE AS
BEGIN
   SET NOCOUNT ON;
  IF (SELECT RT.resID
     FROM ReservedTables RT
     WHERE RT.resID IN
        (SELECT resID
        FROM inserted I
        INNER JOIN IndividualReservation IR ON I.resID =
IR.indResID)
     GROUP BY RT.resID
```

```
      HAVING COUNT(*) > 1) IS NOT NULL
  BEGIN
     ROLLBACK;
       RAISERROR ('The individual reservation already has an
assigned table', 17, -1);
  END
END;
go
```

8.3 Trigger MenuOnChangeValidity

Nie można usunąć pozycji z menu (zmienić daty ważności na wcześniejszą), jeżeli jest na zamówieniu, które będzie zrealizowane w przyszłości

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

    IF (SELECT TOP 1 I.posID
       FROM inserted I
       INNER JOIN OrderDetails OD ON OD.posID = I.posID
       INNER JOIN NonCompletedOrders NCO ON OD.orderID =
NCO.orderID
       ) IS NOT NULL
    BEGIN
       ROLLBACK;
       RAISERROR ('The dish is included in an unfinished order.',
17, -1);
    END
END
GO
```

8.4 Trigger EmployeeOnInsert

Nie można dodać pracownika do zarezerwowanego stolika, jeżeli jest już przy nim tylu klientów ile krzeseł.

```
CREATE TRIGGER EmployeeOnInsert ON Employee
AFTER INSERT, UPDATE AS
BEGIN
   SET NOCOUNT ON;
  IF (SELECT I.resID
     FROM inserted I
     INNER JOIN Employee E ON I.resID = E.resID AND I.tableID =
E.tableID
     INNER JOIN Tables T ON I.tableID = T.tableID
     GROUP BY I.resID, I.tableID, T.chairsNum
```

```
      HAVING COUNT(E.employeeID) > T.chairsNum) IS NOT NULL
  BEGIN
      ROLLBACK;
      RAISERROR ('More people at the table than there are seats',
17, -1);
  END
END;
go
```

8.5 Trigger DiscountIDOnInsert

Nie można dodać numeru zniżki do zamówienia, jeżeli klient nie jest indywidualny.

```
CREATE TRIGGER DiscountIDOnInsert ON Orders
AFTER INSERT, UPDATE AS
BEGIN
      SET NOCOUNT ON;
  IF (SELECT I.orderID
      FROM inserted I
      INNER JOIN Discounts D ON I.discountID = D.discountID
      WHERE I.customerID <> D.customerID) IS NOT NULL
  BEGIN
      ROLLBACK;
      RAISERROR ('Incorrect discount number', 17, -1);
  END
END
go
```

8.6 Trigger MenuOnInsert

Nie można dodać pozycji menu, jeżeli aktualnie jest ta sama dostępna.

```
CREATE TRIGGER MenuOnInsert ON Menu
AFTER INSERT, UPDATE AS
BEGIN
  SET NOCOUNT ON;
  IF (SELECT I.posID
      FROM inserted I
      INNER JOIN Menu M ON I.dishID = M.dishID AND I.posID <>
M.posID
      WHERE ((M.validTo IS NULL AND I.validTo IS NULL)
          OR (M.validFrom < ISNULL(I.validTo, M.validTo) AND
M.validTo >= I.validFrom)
          OR (I.validFrom < ISNULL(M.validTo, I.validTo) AND
I.validTo >= M.validFrom))) IS NOT NULL
  BEGIN
      ROLLBACK;
      RAISERROR ('The dish cannot appear twice on the menu', 17,
-1);
  END
END
go
```

9. Uprawnienia

9.1 Rola Supplier

Rola przyznawana osobom, które zajmują się dostarczaniem zamówień na wynos.

```
CREATE ROLE Supplier
```

Uprawnienia:

★ Odczyt Widoku TakeoutOrderAdresses

```
GRANT SELECT ON TakeoutOrdersAddresses TO Supplier
```

9.2 Rola IndividualCustomer

Rola przyznawana klientowi indywidualnemu.

```
CREATE ROLE IndividualCustomer
```

Uprawnienia:

★ Odczyt widoku CurrentMenu

```
GRANT SELECT ON CurrentMenu TO IndividualCustomer
```

★ Składanie rezerwacji indywidualnej

```
GRANT EXECUTE ON AddIndividualReservation TO
IndividualCustomer
```

★ Składanie zamówienia

```
GRANT EXECUTE ON AddOrder TO IndividualCustomer
```

★ Dodawanie nowych pozycji do istniejącego zamówienia

```
GRANT EXECUTE ON AddToOrder TO IndividualCustomer
```

★ Możliwość sprawdzenia, ile trzeba zapłacić za zamówienie o podanym ID

```
GRANT EXECUTE ON OrderValue TO IndividualCustomer
```

★ Możliwość sprawdzenia, jakie stoliki są dostępne do rezerwacji danego dnia w podanym przedziale czasowym

```
GRANT SELECT ON FreeTableList TO IndividualCustomer
```

★ Możliwość wglądu w aktualne parametry potrzebne do zniżek

```
GRANT SELECT ON CurrentParameters TO IndividualCustomer
```

9.3 Rola CompanyCustomer

Rola przyznawana firmie, która jest klientem.

```
CREATE ROLE CompanyCustomer
```

Uprawnienia:

- ★ Odczyt widoku CurrentMenu

```
GRANT SELECT ON CurrentMenu TO CompanyCustomer
```

- ★ Składanie rezerwacji firmowej

```
GRANT EXECUTE ON AddCompanyReservation TO CompanyCustomer
```

- ★ Składanie zamówienia

```
GRANT EXECUTE ON AddOrder TO CompanyCustomer
```

- ★ Dodawanie nowych pozycji do istniejącego zamówienia

```
GRANT EXECUTE ON AddToOrder TO CompanyCustomer
```

- ★ Możliwość sprawdzenia, ile trzeba zapłacić za zamówienie o podanym ID

```
GRANT EXECUTE ON OrderValue TO CompanyCustomer
```

- ★ Możliwość sprawdzenia, jakie stoliki są dostępne do rezerwacji danego dnia w podanym przedziale czasowym

```
GRANT SELECT ON FreeTableList TO CompanyCustomer
```

- ★ Możliwość wglądu w aktualne parametry potrzebne do zniżek

```
GRANT SELECT ON CurrentParameters TO CompanyCustomer
```

9.4 Rola RestaurantService

Rola przyznawana osobom pracującym w restauracji.

```
CREATE ROLE RestaurantService
```

Uprawnienia:

- ★ Akceptacja rezerwacji

```
GRANT EXECUTE ON ConfirmOrCancelReservation TO
RestaurantService
```

- ★ Widok na dzisiejsze rezerwacje

```
GRANT SELECT ON TodaysReservations TO RestaurantService
```

- ★ Widok na niepotwierdzone rezerwacje

```
GRANT SELECT ON UncomfirmedReservations TO RestaurantService
```

- ★ Generowanie faktur (na życzenie klienta)

```
GRANT SELECT ON GenerateInvoice TO RestaurantService
```

★ Składanie zamówień (dla klientów zamawiających na miejscu)

```
GRANT EXECUTE ON AddOrder TO RestaurantService
```

★ Dodawanie nowych pozycji do istniejącego zamówienia

```
GRANT EXECUTE ON AddToOrder TO RestaurantService
```

9.5 Rola Manager

Przyznawana menadżerowi zajmującemu się działaniem i optymalizacją zysków firmy.

```
CREATE ROLE Manager
```

Uprawnienia:

★ Edycja parametrów

```
GRANT INSERT, UPDATE, DELETE,SELECT ON PARAMETERS TO Manager
```

★ Możliwość wglądu w spis ilości zamówień poszczególnych dań w przeciągu ostatnich dwóch tygodni

```
GRANT SELECT ON DishesRank TO Manager
```

★ Możliwość aktywowania dania w menu lub jego dezaktywacji

```
GRANT EXECUTE ON UpdateAvailability TO Manager
```

★ Dodawanie i usuwanie stolików oraz modyfikacja liczby krzeseł

```
GRANT EXECUTE ON AddTables TO Manager
GRANT EXECUTE ON DeleteTables TO Manager
GRANT EXECUTE ON ChangeChairsNum TO Manager
```

★ Możliwość sprawdzenia miesięcznego dochodu firmy

```
GRANT EXECUTE ON MonthlyIncome TO Manager
```

★ Możliwość dodawania nowych dań do Menu

```
GRANT EXECUTE ON AddMenu  TO Manager
```

★ Możliwość sprawdzenia liczby zamówień dla danego dania

```
GRANT EXECUTE ON HowManyDishOrders TO Manager
```

★ Możliwość wyświetlenia obecnego Menu

```
GRANT SELECT ON CurrentMenu TO Manager
```

★ Możliwość usunięcia dania z Menu

```
GRANT DELETE ON Menu TO Manager
```

9.6 Rola DBAdministator

Przyznawana administratorowi bazy danych.

```
CREATE ROLE DBAdministrator
```

Uprawnienia:

★ Dodawanie i usuwanie miast z bazy danych

```
GRANT EXECUTE ON AddCity TO DBAdministrator
GRANT EXECUTE ON DeleteCity TO DBAdministrator
```

★ Dodawanie klientów indywidualnych i firm do bazy danych

```
GRANT EXECUTE ON AddIndividualCustomer TO DBAdministrator
GRANT EXECUTE ON AddCompany TO DBAdministrator
```

★ Dodawanie i usuwanie krajów z bazy danych

```
GRANT EXECUTE ON AddCountry TO DBAdministrator
GRANT EXECUTE ON DeleteCountry TO DBAdministrator
```

★ Dodawanie i usuwanie stolików oraz modyfikowanie liczby miejsc

```
GRANT EXECUTE ON AddTables TO DBAdministrator
GRANT EXECUTE ON DeleteTables TO DBAdministrator
GRANT EXECUTE ON UpdateAvailability TO DBAdministrator
```

9.7 Rola KitchenService

Przyznawana pracownikom kuchni.

```
CREATE ROLE KitchenService
```

Uprawnienia:

★ Widok na aktualne Menu

```
GRANT SELECT ON CurrentMenu TO KitchenService
```

★ Widok na dzisiejsze zamówienia (OrderID)

```
GRANT SELECT ON TODAYSORDERS TO KitchenService
```

★ Widok na szczegóły zamówienia o podanym ID (żeby wiedzieć, co przygotować i w jakiej ilości)

```
GRANT SELECT ON OrderContent TO KitchenService
```

★ Możliwość modyfikowania dostępności dań w Menu (gdyby jakiś składnik się skończył i niemożliwe byłoby przygotowanie dania)

```
GRANT EXECUTE ON UpdateAvailability TO KitchenService
```

★ Dodawanie nowych dań do Menu

```
GRANT EXECUTE ON AddMenu TO KitchenService
```