# PODSTAWY BAZ DANYCH
# PROJEKT 2022/2023

Małgorzata Krupanek, Aleksandra Sobiesiak, Zofia Burchard
Grupa: 10

# Spis treści

# Użytkownicy systemu

- Klient indywidualny
- Klient firmowy
- Pracownik restauracji

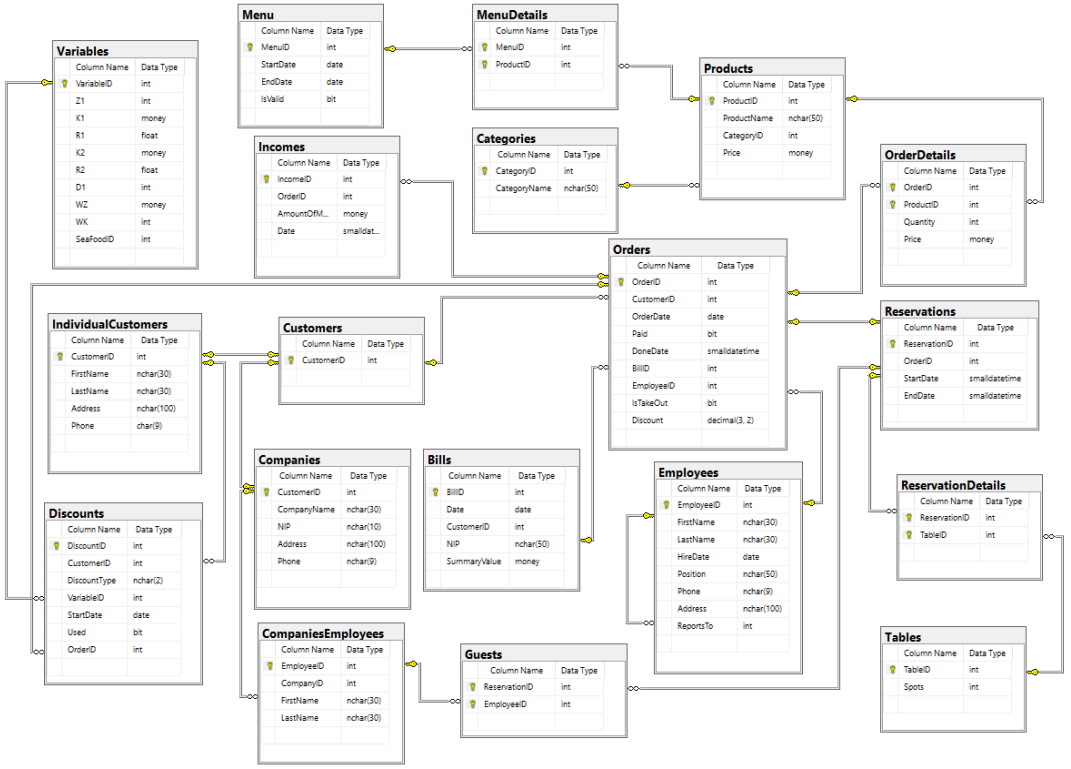
# Dostępne funkcje

- Klient indywidualny:
    - zamawianie jedzenia na miejscu
    - zamawianie jedzenia na wynos na miejscu lub przez formularz www
    - możliwość wyboru daty odbioru zamówienia
    - rezerwacja stolika dla co najmniej dwóch osób
    - możliwość korzystania z rabatów
    - generowanie raportów miesięcznych i tygodniowych dotyczących zamówień i rabatów
    - zmiana zamówienia
    - możliwość zapłaty z góry lub po skorzystaniu z usługi

- Klient firmowy:
    - zamawianie jedzenia na miejscu
    - zamawianie jedzenia na wynos na miejscu lub przez formularz www
    - możliwość wyboru daty odbioru zamówienia
    - możliwość rezerwacji stolików na firmę lub rezerwację stolików dla konkretnych pracowników firmy (imiennie).
    - generowanie raportów miesięcznych i tygodniowych dotyczących zamówień

- Pracownik restauracji:
    - zmiana menu
    - wystawienie faktury
    - przyjmowanie zamówienia
    - anulowanie zamówienia
    - edytowanie zamówienia
    - przyjmowanie rezerwacji
    - dodanie nowego produktu
    - dodanie kategorii
    - zmiana cen produktów
    - podgląd rezerwacji stolików
    - generowanie raportów i statystyk
    - podgląd rabatów danego klienta
    - generowanie raportów miesięcznych i tygodniowych dotyczących rezerwacji stolików, rabatów, menu, czasu składania zamówień, a także statystyk zamówienia
    - generowanie listy produktów wymagających importu

# Diagram


Menu
Column Name Data Type
MenuID int
StartDate date
EndDate date
IsValid bit
MenuDetails
Column Name Data Type
MenuID int
ProductID int
Variables
Column Name Data Type
VariableID int
Z1 int
K1 money
R1 float
K2 money
R2 float
D1 int
WZ money
WK int
SeaFoodID int
Products
Column Name Data Type
ProductID int
ProductName nchar(50)
CategoryID int
Price money
Incomes
Column Name Data Type
IncomeID int
OrderID int
AmountOfM... money
Date smalldat...
Categories
Column Name Data Type
CategoryID int
CategoryName nchar(50)
OrderDetails
Column Name Data Type
OrderID int
ProductID int
Quantity int
Price money
Orders
Column Name Data Type
OrderID int
CustomerID int
OrderDate date
Paid bit
DoneDate smalldatetime
BillID int
EmployeeID int
IsTakeOut bit
Discount decimal(3, 2)
IndividualCustomers
Column Name Data Type
CustomerID int
FirstName nchar(30)
LastName nchar(30)
Address nchar(100)
Phone char(9)
Customers
Column Name Data Type
CustomerID int
Reservations
Column Name Data Type
ReservationID int
OrderID int
StartDate smalldatetime
EndDate smalldatetime
Companies
Column Name Data Type
CustomerID int
CompanyName nchar(30)
NIP nchar(10)
Address nchar(100)
Phone nchar(9)
Bills
Column Name Data Type
BillID int
Date date
CustomerID int
NIP nchar(50)
SummaryValue money
Employees
Column Name Data Type
EmployeeID int
FirstName nchar(30)
LastName nchar(30)
HireDate date
Position nchar(50)
Phone nchar(9)
Address nchar(100)
ReportsTo int
ReservationDetails
Column Name Data Type
ReservationID int
TableID int
Discounts
Column Name Data Type
DiscountID int
CustomerID int
DiscountType nchar(2)
VariableID int
StartDate date
Used bit
OrderID int
CompaniesEmployees
Column Name Data Type
EmployeeID int
CompanyID int
FirstName nchar(30)
LastName nchar(30)
Guests
Column Name Data Type
ReservationID int
EmployeeID int
Tables
Column Name Data Type
TableID int
Spots int

# Tabele

## 1. Products

a. ProductID int - ID produktu, klucz główny
b. ProductName nchar(50) - nazwa produktu
c. CategoryID int - ID kategorii, klucz obcy
d. Price money - cena jednostkowa produktu

```
create table Products
(
    ProductID   int identity
        constraint PK_Products
            primary key,
    ProductName nchar(50) not null,
    CategoryID  int       not null
        constraint FK_CategoryID
            references Categories,
    Price       money     not null
        constraint CK_Products
            check ([Price] > 0)
)
go
```

## 2. Categories

a. CategoryID int - ID kategorii, klucz główny
b. CategoryName nchar(50) - nazwa kategorii

```
create table Categories
(
    CategoryID   int identity
        constraint PK_Categories
            primary key,
    CategoryName nchar(50) not null
)
go
```

## 3. Menu

a. MenuID int - ID menu, klucz główny
b. StartDate date - początek obowiązywania menu
c. EndDate date - koniec obowiązywania menu
d. IsValid bit - czy jest poprawne względem poprzedniego

```
create table Menu
(
    MenuID    int identity
        constraint PK_Menu
            primary key,
    StartDate date not null,
    EndDate   date not null,
    IsValid   bit  not null,
    constraint CK_Menu_Dates
        check ([StartDate] <= [EndDate]
                   and datediff(day,StartDate,EndDate) <= 14)
)
go
```

## 4. MenuDetails

a. MenuID int - ID menu, klucz obcy
b. ProductID int - ID produktu, klucz obcy
a i b - klucz główny złożony

```
create table MenuDetails
(
    MenuID    int not null
        constraint FK_MenuID
            references Menu,
    ProductID int not null
        constraint FK_ProductID
            references Products,
    constraint PK_MenuDetails
        primary key (MenuID, ProductID)
)
go

create index MenuID
    on MenuDetails (MenuID)
go
```

## 5. Orders

a. OrderID int - ID zamówienia, klucz główny
b. CustomerID int - ID klienta, klucz obcy
c. OrderDate date - data zamówienia
d. Paid bit - czy zapłacone
e. DoneDate smalldatetime - data i czas wykonania zamówienia
f. BillID int - ID rachunku, klucz obcy
g. EmployeeID int - ID pracownika obsługującego zamówienie, klucz obcy
h. IsTakeOut (bit) - czy zamówienie jest na wynos
i. Discount decimal(3,2) - wartość procentowa zniżki

```
create table Orders
(
    OrderID    int identity
        constraint PK_Orders
            primary key,
    CustomerID int                                 not null,
    OrderDate  date                                not null,
    Paid       bit                                 not null,
    DoneDate   smalldatetime                       not null,
    BillID     int                                 not null
        constraint FK_BillID
            references Bills,
    EmployeeID int                                 not null
        constraint FK_EmployeeID3
            references Employees,
    IsTakeOut  bit                                 not null,
    Discount   decimal(3, 2)
        constraint DF_Orders_Discount default 0 not null
        constraint CK_Discount
            check ([Discount] <= 1 AND [Discount] >= 0),
    constraint CK_Orders_Dates
        check ([OrderDate] <= [DoneDate])
)
go
```

## 6. Incomes

a. IncomeID - ID wpłaty, klucz główny
b. OrderID - ID zamówienia, klucz obcy
c. AmountOfMoney money - wartość wpłaty
d. Date smalldatetime - data dokonania wpłaty

```
create table Incomes
(
    IncomeID      int identity
        constraint PK_Invoices
            primary key,
    OrderID       int           not null
        constraint [Invoices/Orders]
            references Orders,
    AmountOfMoney money         not null
        constraint CK_Invoices_Amount
            check ([AmountOfMoney] > 0),
    Date          smalldatetime not null
)
go
```

## 7. Bills

a. BillID int - ID rachunku, klucz główny
b. Date date - data wydania rachunku
c. CustomerID int - ID klienta, klucz obcy
d. NIP nchar(10) - NIP firmy (lub null gdy klient indywidualny)
e. SummaryValue money - łączna kwota za zamówienia

```
create table Bills
(
    BillID       int identity
        constraint PK_Bills
            primary key,
    Date         date  not null,
    CustomerID   int   not null,
    NIP          nchar(10),
    SummaryValue money not null
)
go

create index IX_Bills_NIP
    on Bills (NIP)
go
```

## 8. Reservations

a. ReservationID int - ID rezerwacji, klucz główny
b. OrderID int - ID zamówienia, klucz obcy
c. StartDate smalldatetime - początek rezerwacji
d. EndDate smalldatetime - koniec rezerwacji

```
create table Reservations
(
    ReservationID int identity
        constraint PK_Reservations_1
            primary key,
    OrderID       int           not null
        constraint IX_Reservations
            unique
        constraint FK_OrderID1
            references Orders,
    StartDate     smalldatetime not null,
    EndDate       smalldatetime not null,
    constraint CK_Reservations
        check ([EndDate] > [StartDate] AND datediff(day, [StartDate], [EndDate]) = 0)
)
go
```

## 9. OrderDetails

a. OrderID int - ID zamówienia, klucz obcy
b. ProductID int - ID produktu, klucz obcy
c. Quantity int - ilość zamówionego produktu
d. Price money - cena jednostkowa produktu w trakcie dokonywania zamówienia

a i b - klucz główny złożony

```
create table OrderDetails
(
    OrderID   int   not null
        constraint FK_OrderID2
            references Orders,
    ProductID int   not null
        constraint FK_ProductID1
            references Products,
    Quantity  int   not null
        constraint CK_OrderDetails_Quantity
            check ([Quantity] > 0),
    Price     money not null
        constraint CK_OrderDetails_Price
            check ([Price] > 0),
    constraint PK_OrderDetails
        primary key (OrderID, ProductID)
)
go
```

## 10. Customers

a. CustomerID int - ID klienta, klucz główny

```
create table Customers
(
    CustomerID int identity
        constraint PK_Customers
            primary key
)
go
```

## 11. Companies

a. CustomerID int - ID klienta firmowego, klucz obcy
b. CompanyName nchar(30) - nazwa firmy
c. NIP nchar(10) - numer NIP firmy
d. Address nchar(100) - adres firmy
e. Phone nchar(9) - numer telefonu firmy

```
create table Companies
(
    CustomerID  int         not null
        constraint PK_Companies
            primary key
        constraint FK_CustomerID2
            references Customers,
    CompanyName nchar(30)   not null,
    NIP         nchar(10)   not null,
    Address     nchar(100)  not null,
    Phone       nchar(9)    not null
        constraint CK_Companies_Phone
            check ([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
go

create unique index IX_Companies_NIP_UNique
    on Companies (NIP)
go
```

## 12. Companies Employees

a. EmployeeID int - ID pracownika, klucz główny
b. CompanyID int - ID firmy, klucz obcy
c. FirstName nchar(30) - imię pracownika
d. LastName nchar(30) - nazwisko pracownika

```
create table CompaniesEmployees
(
    EmployeeID int identity
        constraint PK_Employees
            primary key,
    CompanyID  int       not null
        constraint FK_EmployeeID
            references Companies,
    FirstName  nchar(30) not null,
    LastName   nchar(30) not null
)
go

create index IX_CompaniesEmployees_CompanyID
    on CompaniesEmployees (CompanyID)
go
```

## 13. IndividualCustomers

a. CustomerID int - ID klienta indywidualnego, klucz obcy 1-1
b. FirstName nchar(30) - imię klienta
c. LastName nchar(30) - nazwisko klienta
d. Address nchar(100) - adres klienta
e. Phone nchar(9) - numer telefonu klienta

```
create table IndividualCustomers
(
    CustomerID int       not null
        constraint PK_IndividualCustomers
            primary key
        constraint FK_CustomerID1
            references Customers,
    FirstName  nchar(30) not null,
    LastName   nchar(30) not null,
    Address    nchar(100),
    Phone      char(9)
        constraint CK_IndividualCustomers_Phone
            check ([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
go
```

## 14. Discounts

a. DiscountID int - ID zniżki, klucz główny
b. CustomerID int - ID klienta, klucz obcy
c. DiscountType nchar(2) - typ zniżki (R1 / R2)
d. VariableID int - ID zmiennych, klucz obcy
e. StartDate date - data początku zniżki
f. Used bit - czy zniżka została użyta
g. OrderID int - ID zamówienia, klucz obcy

```
create table Discounts
(
    DiscountID   int identity
        constraint PK_Discounts
            primary key,
    CustomerID   int      not null
        constraint FK_CustomerID3
            references IndividualCustomers,
    DiscountType nchar(2) not null
        constraint CK_Discounts_DiscountType
            check ([DiscountType] = 'R1' OR [DiscountType] = 'R2'),
    VariableID   int      not null
        constraint FK_Discounts_Variables
            references Variables,
    StartDate    date     not null,
    Used         bit      not null,
    OrderID      int
        constraint FK_OrderID3
            references Orders
)
go
```

## 15. ReservationDetails

a. ReservationID int - ID rezerwacji, klucz obcy
b. TableID int - ID stolika, klucz obcy
a i b - klucz główny złożony

```
create table ReservationDetails
(
    ReservationID int not null
        constraint FK_EmployeeID2
            references Reservations,
    TableID       int not null
        constraint FK_TableID
            references Tables,
    constraint PK_ReservationDetails
        primary key (ReservationID, TableID)
)
go
```

## 16. Tables

a. TableID int - ID stolika, klucz główny
b. Spots int - liczba miejsc przy stoliku

```
create table Tables
(
    TableID int identity
        constraint PK_Tables
            primary key,
    Spots   int not null
        constraint CK_Tables_Spots
            check ([Spots] > 0)
)
go
```

## 17. Guests

a. ReservationID int - ID rezerwacji, klucz obcy
b. EmployeeID int - ID pracownika firmowego - gościa
a i b - klucz główny złożony

```
create table Guests
(
    ReservationID int not null
        constraint FK_ReservationID
            references Reservations,
    EmployeeID    int not null
        constraint FK_EmployeeID1
            references CompaniesEmployees,
    constraint PK_Guests
        primary key (ReservationID, EmployeeID)
)
go

create index IX_Guests
    on Guests (ReservationID)
go
```

## 18. Variables

a. VariableID - ID zmiennych - klucz główny
b. Z1 int - ilość zamówień do zniżki nr 1
c. K1 money - minimalna kwota za każde zamówienie do zniżki nr 1
d. R1 float - procent zniżki nr 1, dostajemy na wszystkie zamówienia
e. K2 int - łączna kwota potrzebna do zniżki nr 2
f. R2 float - procent zniżki nr 2, dostajemy jednorazowo
g. D1 int - na ile dni obowiązuje zniżka nr 2
h. WZ money - minimalna wartość zamówienia przy zamawianiu przez internet
i. WK int - ilość zamówień, które trzeba złożyć, by móc zamawiać przez internet
j. SeaFoodID int - ID kategorii owoców morza

```
create table Variables
(
    VariableID int identity
        constraint PK_Variables
            primary key,
    Z1         int   not null
        constraint CK_Variables_Z1
            check ([Z1] > 0),
    K1         money not null
        constraint CK_Variables_K1
            check ([K1] > 0),
    R1         float not null
        constraint CK_Variables_R1
            check ([R1] > 0 AND [R1] < 1),
    K2         money not null
        constraint CK_Variables_K2
            check ([K2] > 0),
    R2         float not null
        constraint CK_Variables_R2
            check ([R2] > 0 AND [R2] < 1),
    D1         int   not null
        constraint CK_Variables_D1
            check ([D1] > 0),
    WZ         money not null
        constraint CK_Variables_WZ
            check ([WZ] > 0),
    WK         int   not null
        constraint CK_Variables_WK
            check ([WK] > 0),
    SeaFoodID  int   not null
)
go
```

## 19. Employees

a. EmployeeID int - ID pracownika restauracji, klucz główny
b. FirstName nchar(30) - imię pracownika
c. LastName nchar(30) - nazwisko pracownika
d. HireDate date - data zatrudnienia
e. Position nchar(50) - stanowisko pracownika
f. Phone nchar(9) - numer telefonu pracownika
g. Address nchar(100) - adres zamieszkania
h. ReportsTo int - ID przełożonego, klucz obcy

```
create table Employees
(
    EmployeeID int identity
        constraint PK_Employees_1
            primary key,
    FirstName  nchar(30)  not null,
    LastName   nchar(30)  not null,
    HireDate   date       not null
        constraint CK_Employees_HireDate
            check ([HireDate] <= getdate()),
    Position   nchar(50)  not null,
    Phone      nchar(9)   not null
        constraint CK_Employees_Phone
            check ([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Address    nchar(100) not null,
    ReportsTo  int
        constraint FK_ReportsTo
            references Employees
)
go
```

# Widoki

## 1. MonthlyTablesStats

Statystyki rezerwacji stolików na każdy miesiąc. Widok zawiera ilość zarezerwowanych stolików oraz sumaryczną liczbę zarezerwowanych miejsc dla każdego miesiąca.

```
create view dbo.MonthlyTableStats as
select year(OrderDate) as Year,
       month(OrderDate) as Month,
       count(*) as noTables,
       sum(T.Spots) as noSpots
from Orders O
inner join Reservations R on O.OrderID = R.OrderID
inner join ReservationDetails RD on R.ReservationID = RD.ReservationID
inner join Tables T on RD.TableID = T.TableID
group by year(OrderDate), month(OrderDate)
go
```

## 2. WeeklyTablesStats

Statystyki rezerwacji stolików na każdy miesiąc. Widok zawiera ilość zarezerwowanych stolików oraz sumaryczną liczbę zarezerwowanych miejsc dla każdego tygodnia.

```
create view dbo.WeeklyTablesStats as
select year(OrderDate) as Year,
       datepart(week, OrderDate) as Week,
       count(*) as noTables,
       sum(T.Spots) as noSpots
from Orders O
inner join Reservations R on O.OrderID = R.OrderID
inner join ReservationDetails RD on R.ReservationID = RD.ReservationID
inner join Tables T on RD.TableID = T.TableID
group by year(OrderDate), datepart(week, OrderDate)
go
```

## 3. MonthlyDiscountsStats

Widok zawiera sumaryczną wartość wykorzystanych przez klientów zniżek każdego miesiąca.

```
CREATE view dbo.MonthlyDiscountStats as
select year(OrderDate) as Year,
       month(OrderDate) as Month ,
       sum(OD.Quantity * OD.Price * O.Discount) as Value
from Discounts D
inner join Orders O on D.OrderID = O.OrderID
inner join OrderDetails OD on O.OrderID = OD.OrderID
group by year(OrderDate), month(OrderDate)
go
```

## 4. WeeklyDiscountStats

Widok zawiera sumaryczną wartość wykorzystanych przez klientów zniżek każdego tygodnia.

```
CREATE view dbo.WeeklyDiscountStats as
select year(OrderDate) as Year,
       datepart(week, OrderDate) as Week ,
       sum(OD.Quantity * OD.Price * O.Discount) as Value
from Discounts D
inner join Orders O on D.OrderID = O.OrderID
inner join OrderDetails OD on O.OrderID = OD.OrderID
group by year(OrderDate), datepart(week, OrderDate)
go
```

## 5. FutureTakeOuts

Spis zamówień na wynos do zrealizowania. Widok zawiera OrderID, CustomerID, wartość zamówienia, datę dostarczenia oraz czy zamówienie zostało zapłacone.

```
CREATE view dbo.FutureTakeOuts as
    select o.OrderID,
           o.CustomerID,
           sum(od.Quantity * od.Price) * (1 - (o.Discount)) as Value,
           o.DoneDate,
           o.Paid
    from Orders o, OrderDetails od
    where IsTakeOut = 1 and o.DoneDate >= cast(getdate() as smalldatetime)
    group by o.OrderID, o.CustomerID, o.Discount, o.DoneDate, o.Paid
go
```

## 6. IndividualClientsStats

Statystyki zamówień dla klientów indywidualnych. Widok zawiera ID, imię i nazwisko klienta, łączną liczbę jego zamówień, łączną kwotę wydaną na zamówienia, liczbę wykorzystanych rabatów oraz łączną kwotę oszczędności z rabatów.

```
create view dbo.IndividualClientsStats as
select C.CustomerID, IC.FirstName, IC.LastName,
    isnull(count(O.CustomerID),0) as NumberOfOrders,
    isnull(sum(OD.Price*OD.Quantity*(1-O.Discount)), 0) as TotalSpent,
    isnull
        ((select count(D.DiscountID)
        from Discounts D
        where C.CustomerID = D.CustomerID),0) as NumberOfDiscountsUsed,
    isnull((sum(OD.Price*OD.Quantity*O.Discount)), 0) as TotalSavings

from Customers C
join IndividualCustomers IC on C.CustomerID = IC.CustomerID
left join Orders O ON C.CustomerID = O.CustomerID
left join OrderDetails OD on O.OrderID = OD.OrderID
group by C.CustomerID, IC.FirstName, IC.LastName
go
```

## 7. CompaniesStats

Statystyki zamówień dla klientów firmowych. Widok zawiera ID i nazwę firmy klienckiej, łączną liczbę złożonych zamówień przez daną firmę i łączną wartość złożonych zamówień.

```
create view dbo.CompaniesStats as
select C.CustomerID, C.CompanyName,
       count(O.CustomerID) as NumberOfOrders,
       isnull(SUM(OD.Price*OD.Quantity), 0) as TotalValue
from Companies C
join Customers C2 on C.CustomerID = C2.CustomerID
left join Orders O on C2.CustomerID = O.CustomerID
left join OrderDetails OD on O.OrderID = OD.OrderID
group by C.CustomerID, C.CompanyName
go
```

## 8. FreeTablesForToday

Wolne stoliki na dany dzień. Widok zawiera numer ID i liczbę miejsc przy każdym ze stolików wolnych w bieżącym dniu.

```
create view dbo.FreeTablesForToday as
select TableID, Spots
from Tables
where TableID not in(
    select RD.TableID
    from ReservationDetails RD
    join Reservations R on RD.ReservationID = R.ReservationID
    join Orders O on R.OrderID = O.OrderID
    where DoneDate = getdate()
    )
go
```

## 9. OncomingReservations

Informacje o nadchodzących rezerwacjach. Widok dla każdej nadchodzącej rezerwacji zawiera ID rezerwacji, ID zamówienia, jej łączną wartość, datę rezerwacji, liczbę zarezerwowanych stolików i liczbę gości.

```
CREATE view dbo.OncomingReservations as
select R.ReservationID, R.OrderID, O.CustomerID,
       isnull(sum(OD.Quantity*OD.Price*(1-Discount)), 0) as TotalValue,
       O.OrderDate, O.DoneDate as PlannedFor,
       (select count(TableID)
        from ReservationDetails RD
        where RD.ReservationID = R.ReservationID) as NumberOfTables,
        (select count(G.EmployeeID)
         from Guests G
         where G.ReservationID = R.ReservationID) as NumberOfGuests
from Reservations R
join Orders O on O.OrderID = R.OrderID
                    and DoneDate >= cast(getDate() as smalldatetime)
left join OrderDetails OD on O.OrderID = OD.OrderID
group by R.ReservationID, R.OrderID, O.CustomerID, O.OrderDate, O.DoneDate
go
```

## 10. OncomingOrders

Informacje o nadchodzących zamówieniach. Widok dla każdego zamówienia zawiera ID zamówienia, ID zamawiającego klienta, łączną wartość zamówienia, datę realizacji, informację czy zostało opłacone oraz czy jest zamówieniem na wynos.

```
CREATE view dbo.OncomingOrders as
select O.OrderID, O.CustomerID,
       isnull(sum(OD.Quantity*OD.Price*(1-O.Discount)),0) as TotalValue,
       O.DoneDate, O.Paid, O.IsTakeOut
from Orders O
left join OrderDetails OD on O.OrderID = OD.OrderID
where O.DoneDate >= cast(getdate() as smalldatetime)
group by O.OrderID, O.CustomerID, O.DoneDate, O.Paid, O.IsTakeOut
go
```

## 11. CurrentMenu

Informacja na temat aktualnych pozycji w menu, zawiera nazwę produktu, jego cenę oraz kategorię.

```
CREATE VIEW CurrentMenu as
SELECT ProductName,Price,CategoryName
from menu as m
INNER JOIN MenuDetails as md
on md.MenuID=m.MenuID
INNER JOIN products as p
on p.ProductID=md.ProductID
INNER JOIN Categories as c
on p.CategoryID=c.CategoryID
where StartDate<=CAST( GETDATE() AS Date )AND EndDate>=CAST( GETDATE() AS Date)
go
```

## 12. MonthlyMenuStats

Informacje na temat liczby zamówionego  danego produktu przez klientów i ile on przyniósł zysku z podziałem na miesiące. Widok zawiera nazwę produktu, miesiąc, ilość sprzedanych produktów w tym miesiącu oraz zysk.

```
CREATE VIEW dbo.MonthlyMenuStats as
select p.ProductName, sum(od.Quantity) as 'Ilość Zamówień' ,
      sum(Od.Price*od.Quantity) as Przychód,
      month(orderdate) as Miesiąc
from Products as p
INNER JOIN OrderDetails as od
on p.ProductID=od.ProductID
INNER JOIN orders as o
on o.OrderID=od.OrderID
group by p.ProductName,month(OrderDate)
```

## 13. WeeklyMenuStats

Informacje na temat liczby zamówionego danego produktu przez klientów i ile on przyniósł zysku z podziałem na tygodnie. Widok zawiera nazwę produktu, tygodniu, ilość sprzedanych produktów w tym tygodniu oraz zysk.

```
CREATE VIEW dbo.WeeklyMenuStats as
select p.ProductName, isnull(sum(od.Quantity),0) as 'Ilość Zamówień',
       isnull(sum(Od.Price*od.Quantity),0) as Przychód,
       isnull(DATEPART(week,o.orderdate), 0) as Tydzień
from Products as p
LEFT JOIN OrderDetails as od
on p.ProductID=od.ProductID
LEFT JOIN orders as o
on o.OrderID=od.OrderID
group by p.ProductName, DATEPART(week,o.orderdate)
go

grant select on WeeklyMenuStats to manager
go
```

## 14. OrdersStats

Zawiera informacje na temat zamówień, jaki wygenerowały przychód w podziale na lata i miesiące. Widok zawiera miesiąc, rok, ilość zamówień oraz przychód w określonym czasie

```
CREATE VIEW dbo.OrdersStats as
select month(orderDate) as Miesiąc,
     YEAR(o.OrderDate) AS rok,
     SUM((od.Price * od.Quantity) * (1 - o.Discount)) AS Przychód,
     (SELECT COUNT(*)
     FROM  dbo.Orders AS o2
     WHERE (month(o2.OrderDate) = MONTH(o.OrderDate) and year(o2.OrderDate) = year(o.OrderDate)))
         AS 'Liczba zamówień w tym miesiącu i roku'
from orders as o
INNER JOIN OrderDetails as od
on o.OrderID=od.OrderID
GROUP BY MONTH(o.OrderDate), YEAR(o.OrderDate)
go

grant select on OrdersStats to manager
go
```

## 15. ToPay

Zawiera informację o płatnościach klientów. Widok zawiera łączną wartość zamówień, łączną wartość wpłat, bilans i numer klienta.

```
CREATE VIEW dbo.ToPay as
select O.CustomerID,
        (select sum(OD.Quantity*OD.Price*(1-O2.Discount))
        from OrderDetails OD
        join Orders O2 on O2.OrderID = OD.OrderID
        where O.CustomerID = O2.CustomerID) as Orders,
        sum(AmountOfMoney) as Incomes,
       (select sum(OD.Quantity*OD.Price*(1-O2.Discount))
        from OrderDetails OD
        join Orders O2 on O2.OrderID = OD.OrderID
        where O.CustomerID = O2.CustomerID) - sum(AmountOfMoney) as Balance
from Incomes
join Orders O on Incomes.OrderID = O.OrderID
group by O.CustomerID
go
```

# Funkcje

## 1. CompaniesStatsFunction

Funkcja umożliwia przeglądanie zamówień składanych przez firmy  wraz z ich ceną pomiędzy wskazanymi datami.

```
CREATE FUNCTION CompaniesStatsFunction(@StartDate date,@EndDate date)
        RETURNS TABLE AS
        RETURN
        select c.CompanyName,
        (select sum(od.Price*od.Quantity*(1-o.Discount))
        from OrderDetails as od
        INNER JOIN Orders as o on o.OrderID=od.OrderID
        where od.OrderID= o1.OrderID) as "Price"
        from Companies as c
        INNER JOIN orders as o1
        on o1.CustomerID=c.CustomerID
        where o1.OrderDate BETWEEN @StartDate AND @EndDate
```

## 2. IndividualStatsFunction

Funkcja umożliwia przeglądanie zamówień składanych przez klientów indywidualnych wraz z ich cena pomiędzy wskazanymi datami.

```
CREATE FUNCTION IndividualStatsFunction(@StartDate date, @EndDate date)
    RETURNS TABLE AS
    RETURN
    SELECT ic.FirstName, ic.LastName,
    (select sum(od.Price*od.Quantity*(1-o.Discount))
    from OrderDetails as od
    INNER JOIN Orders as o on o.OrderID=od.OrderID
    where od.OrderID= o1.OrderID) as "Price"
    FROM IndividualCustomers as ic
    INNER JOIN Orders as o1
    on o1.CustomerID=ic.CustomerID
    where o1.OrderDate BETWEEN @StartDate AND @EndDate
```

## 3. CompaniesStatsToClientFunction

Funkcja umożliwia przeglądanie zamówień wykonanych pomiędzy wskazanymi datami, złożonych przez konkretnego klienta wraz z ich ceną oraz datą.

```
CREATE FUNCTION CompaniesStatsToCLientFunction(@customerID int,@StartDate date, @EndDate date)
    RETURNS TABLE AS
    RETURN
    select o.OrderDate as "Data",
    (select sum(Price)
    from OrderDetails as od
    where od.OrderID= o.OrderID) as "Price"
    from Companies as c
    INNER JOIN orders as o
    on o.CustomerID=c.CustomerID
    where o.OrderDate BETWEEN @StartDate AND @EndDate
    AND  c.customerID=@customerID
```

## 4. GetMenuProductsByDate

Zwraca informacje o produktach, ID odpowiedniego menu, czas obowiązywania tego menu, zależnie od przekazanej jako argument daty.

```
create function GetMenuProductsByDate(@date date)
returns table as
    return
    select M.MenuID, M.StartDate, M.EndDate, P.ProductID, P.ProductName, P.Price
    from Products P
    join MenuDetails MD on P.ProductID = MD.ProductID
    join Menu M on MD.MenuID = M.MenuID
    where @date between M.StartDate and M.EndDate
```

## 5. GetReservedTablesFromDateToDate

Zwraca informacje o zarezerwowanych stolikach pomiędzy wprowadzonymi datami, a dokładniej: ID stolika, liczbę miejsc przy stoliku, numer rezerwacji danego stolika, numer powiązanego zamówienia i godziny trwania rezerwacji.

```
create function dbo.GetReservedTablesFromDateToDate (@StartDate smalldatetime, @EndDate smalldatetime)
returns table as
    return
    select T.TableID, T.Spots, R.ReservationID, R.OrderID, R.StartDate, R.EndDate
    from Tables T
    join ReservationDetails RD on T.TableID = RD.TableID
    join Reservations R on RD.ReservationID = R.ReservationID
    where R.StartDate >= @StartDate and R.EndDate <= @EndDate
go
```

## 6. GetTakeoutsInfosByDate

Zwraca informacje o zamówieniach na wynos na dany dzień, a dokładniej: ID zamówienia, ID klienta, datę złożenia zamówienia, informację czy zostało opłacone, czas odbioru, ID powiązanego rachunku, ID pracownika obsługującego zamówienie, wysokość rabatu, łączną wartość i zaoszczędzoną dzięki rabatowi kwotę.

```
CREATE function dbo.GetTakeoutsInfosByDate (@date date)
returns table as
    return
    select O.OrderID, O.CustomerID, O.OrderDate, O.Paid, cast(O.DoneDate as time) TakoutTime,
           O.BillID, O.EmployeeID, O.Discount,
           sum(OD.Price*OD.Quantity*(1-O.Discount)) as Value,
           sum(OD.Price*OD.Quantity*O.Discount) as DiscountValue
    from Orders O
    join OrderDetails OD on O.OrderID = OD.OrderID
    where IsTakeOut = 1 and convert(date, O.DoneDate) = @date
    group by O.OrderID, O.CustomerID, O.OrderDate, O.Paid, cast(O.DoneDate as time), O.BillID, O.EmployeeID, O.Discount
go
```

## 7. GetToPayInfoForClient

Zwraca dla danego klienta wysokość debetu, kwoty koniecznej do zapłacenia za zamówienia. Jako argument przyjmuje ID klienta

```
CREATE function GetToPayInfoForClient (@ClientID int)
returns money as
        (select (select sum(OD.Quantity*OD.Price*(1-O2.Discount))
                from OrderDetails OD
                join Orders O2 on O2.OrderID = OD.OrderID
                where O.CustomerID = O2.CustomerID)
                   - sum(I.AmountOfMoney)
        from Orders O
        join Incomes I on O.OrderID = I.OrderID
        where O.CustomerID = @ClientID
        group by CustomerID
        having sum(I.AmountOfMoney) < (select sum(OD.Quantity*OD.Price*(1-O2.Discount))
                from OrderDetails OD
                join Orders O2 on O2.OrderID = OD.OrderID
                where O.CustomerID = O2.CustomerID))
go
```

## 8. GetIsMenuValid

Zwraca dla danego menu czy jest ono poprawne (czy minimum połowa produktów z poprzedniego menu została zmieniona). Zwraca wartość bitową true/false.

```
CREATE function GetIsMenuValid(@currentMenu int)
returns bit as
    begin
        declare @previousID int = (select top 1 MenuID from Menu
                                   where MenuID < @currentMenu order by MenuID desc)
        declare @duplicates int
        set @duplicates =
            (select count(*) from
                ( select MD1.ProductID
            from MenuDetails MD1
            where MD1.MenuID = @previousID
                intersect
            select MD2.ProductID
            from MenuDetails MD2
            where MD2.MenuID = @currentMenu
                ) out)
        declare @amountToBeChanged int
        set @amountToBeChanged =
            (select count(ProductID)
            from MenuDetails
            where MenuID = @previousID) /2
        if @duplicates <= @amountToBeChanged
        begin
            return 1
        end
    return 0
    end
go
```

## 9. FutureOrdersForClient

Funkcja zwraca dla danej firmy nadchodzące zamówienie wraz z datą przyjęcia zamówienia, datą wykonania zamówienia oraz czy zamówienie jest opłacone.

```
CREATE FUNCTION FutureOrdersForClient(@customerid int)
        RETURNS TABLE AS
        RETURN
        select orderID, DoneDate,paid,OrderDate
        from orders as o
        INNER JOIN Customers as c
        on o.CustomerID=c.CustomerID
        where DoneDate>= CAST (GETDATE() as DATE)
        AND c.CustomerID=@customerid
```

## 10. OrdersToPayForCcompanies

Funkcja zwraca wszystkie nieopłacone zamówienia dla wybranego klienta , wyświetlając ich numer,datę wykonania i kwotę do zapłaty

```
CREATE FUNCTION OrdersToPayForCompanies(@customerid int)
         RETURNS TABLE AS
         RETURN
         select o2.orderid,o2.DoneDate,ISNULL(-sum(AmountOfMoney),0)+ISNULL((select sum(od.quantity*od.price)
               from OrderDetails as od
               join orders as o on o.orderId=od.orderId
               where o.OrderID=o2.OrderID),0) as 'Kwota do zapłaty'
         from Incomes
         RIGHT  JOIN orders as o2
          on o2.orderid=Incomes.OrderID
         LEFT JOIN Customers as c
         on o2.CustomerID=c.CustomerID
         where o2.CustomerID=@customerid
         group by o2.OrderID,o2.DoneDate
         having ISNULL(-sum(AmountOfMoney),0)+ISNULL((select sum(od.quantity*od.price)
               from OrderDetails as od
               join orders as o on o.orderId=od.orderId
               where o.OrderID=o2.OrderID),0) >0
```

## 11. MaxIndexFunction

Funkcja zwraca maksymalny indeks istniejący w klientach.

```
CREATE FUNCTION MaxClientIndexFunction()
RETURNS int
AS
BEGIN
    DECLARE @var int
    SELECT @var=MAX(CUSTOMERid)
    from customers
    RETURN @var
END
go
```

## 12. GetOrdersByDate

Funkcja zwraca zamówienia na dany dzień.

```
CREATE function dbo.GetOrdersByDate(@input smalldatetime)
        returns table as
            return
            select * from Orders O
            where O.DoneDate = @input
go
```

## 13. GetNumberOfOrdersK1ByCustomerID

Funkcja zwraca ilość złożonych przez danego klienta zamówień powyżej kwoty K1.

```
CREATE function dbo.GetNumberOfOrdersK1ByCustomerID(@input int)
        returns int as
            begin
                return
                    (select count(*)
                    from (select O.OrderID
                            from Orders O
                            inner join OrderDetails OD on O.OrderID = OD.OrderID
                            where O.CustomerID = @input
                            group by  O.OrderID
                            having sum((OD.Quantity * OD.Price) * (1 - (O.Discount))) >
                                (select top(1) K1 from Variables order by VariableID desc)) as T)
            end
go
```

## 14. GetNumberOfOrdersByCustomerID

Funkcja zwraca ilość zamówień złożonych przez danego klienta.

```
CREATE function dbo.GetNumberOfOrdersByCustomerID(@input int)
        returns int as
            begin
                return
                (select count(*) from Orders
                where CustomerID = @input)
            end
go
```

## 15. GetValidDiscountsByCustomerID

Funkcja zwraca ważne zniżki dla danego klienta.

```
CREATE function dbo.GetValidDiscountsByCustomerID(@input int)
returns table as
    return (select * from Discounts
            where CustomerID = @input and (Used = 0 or DiscountType = 'R1'))
go
```

## 16. GetValueOfOrdersByCustomerID

Funkcja zwraca wartość wszystkich zamówień dla danego klienta po dacie ostatniej dodanej zniżki jednorazowej.

```
CREATE function dbo.GetValueOfOrdersByCustomerID(@CustomerID int)
    returns float as
        begin
            declare @Date date
                set @Date = '1/2/1950'
                if exists(select *
                            from Discounts
                            where CustomerID = @CustomerID and DiscountType = 'R2')
                begin
                    set @Date = (select top 1 StartDate
                        from Discounts
                        where CustomerID = @CustomerID and DiscountType = 'R2'
                        order by StartDate desc)
                end

            return (select sum((OD.Quantity * OD.Price) * (1 - (O.Discount)))
                    from Orders O
                    inner join OrderDetails OD on O.OrderID = OD.OrderID
                    where CustomerID = @CustomerID and OrderDate > @Date)
        end
go
```

## 17. GetIsCustomerIndividual

Funkcja zwraca wartość bitową: true - jeśli podany klient to klient indywidualny, 0 - jeśli firmowy, null - jeśli nie został znaleziony.

```
create function GetIsCustomerIndividual (@CustomerID int)
returns bit as
    begin
        if exists(select * from IndividualCustomers where @CustomerID = CustomerID)
        begin
            return 1
        end
        else
        begin
            if exists(select * from Companies where @CustomerID = CustomerID)
            begin
                return 0
            end
        end
        return null
    end
go
```

# Procedury

## 1. AddProductToOrder

Procedura dodaje nowy produkt do wybranego zamówienia pod warunkiem, że produkt istnieje, znajduje się w aktualnym menu oraz jeśli jest z kategorii owoców morza to zostaje zamówiony na odpowiednie dni i z odpowiednim wyprzedzeniem. Procedura jako argumenty przyjmuje OrderID (ID zamówienia), ProductID lub ProductName (jedno z tych może być null), Quantity (ilość produktu).

```
CREATE procedure AddProductToOrder @OrderID int, @ProductID int, @Quantity int
as
    begin
        begin try
            if not exists(
                select * from Products
                where ProductID = @ProductID)
            begin
                throw 52000, 'Taka potrawa nie istnieje', 1
            end

            if not exists(
                select * from Orders
                where OrderID = @OrderID)
            begin
                throw 52000, N'Takie zamówienie nie istnieje', 1
            end

            declare @DoneDate date = (select convert(date,DoneDate) from Orders
                                      where OrderID = @OrderID)

            if not exists(
                select * from GetMenuProductsByDate ( @date: @DoneDate)
                where ProductID = @ProductID)
            begin
                throw 52000, N'Tego produktu nie ma w menu na ten dzień', 1
            end

            if exists(
                select * from Products P
                where (P.ProductID = @ProductID) and
                    P.CategoryID = (select top 1 SeaFoodID from Variables order by VariableID desc))

            begin
                set datefirst 1;

                declare @OrderDate date = (select OrderDate from Orders
```

```
            where OrderID = @OrderID

            declare @weekday int = datepart(weekday, @DoneDate)

            if @weekday not in (4, 5, 6) or
               @OrderDate < convert(date,dateadd(dd, -1*(datepart(dw, getdate())-1), getdate()))
            begin
                throw 52000, N'Tego dnia nie można złożyć zamówienia na owoce morza', 1
            end
        end

        declare @Price money = (select Price from Products
                                where @ProductID = ProductID)
        insert into OrderDetails(OrderID, ProductID, Quantity, Price)
            values (@OrderID, @ProductID, @Quantity, @Price)
    end try

    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania produktu do zamówienia: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
end
go
```

## 2. AddVariables

Procedura wprowadza nowe zmienne dotyczące wysokości i warunków zniżek oraz rezerwacji, oraz wstawia je do tabeli Variables. Procedura w sytuacji wprowadzenia tylko niektórych zmiennych, dla tych niezdefiniowanych, ustawia takie same wartości jakie były w poprzednim zestawie zmiennych.

```
CREATE procedure AddVariables @Z1 int = null, @K1 money  = null, @R1 float = null, @K2 int = null, @R2 float = null,
@D1 int = null, @WZ money = null, @WK int = null, @SeaFoodID int = null
as
    begin try
        declare @previousID int = (
            select top 1 VariableID from Variables order by VariableID desc)

        if @Z1 is null
        begin
            set @Z1 = (select Z1 from Variables where VariableID = @previousID)
        end
        if @K1 is null
        begin
            set @K1 = (select K1 from Variables where VariableID = @previousID)
        end
        if @R1 is null
        begin
            set @R1 = (select R1 from Variables where VariableID = @previousID)
        end
        if @K2 is null
        begin
            set @K2 = (select K2 from Variables where VariableID = @previousID)
        end
        if @R2 is null
        begin
            set @R2 = (select R2 from Variables where VariableID = @previousID)
        end
        if @D1 is null
        begin
            set @D1 = (select D1 from Variables where VariableID = @previousID)
        end
        if @WZ is null
        begin
            set @WZ = (select WZ from Variables where VariableID = @previousID)
        end
        if @WK is null
        begin
            set @WK = (select WK from Variables where VariableID = @previousID)
        end
        if @SeaFoodID is null
        begin
            set @SeaFoodID = (select SeaFoodID from Variables where VariableID = @previousID)
        end
        insert into Variables(Z1, K1, R1, K2, R2, D1, WZ, WK, SeaFoodID)
        values (@Z1, @K1, @R1, @K2, @R2, @D1, @WZ, @WK, @SeaFoodID)
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania nowych zmiennych' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 3. AddTableToReservation

Procedura dodaje nowy stolik do wybranej rezerwacji, pod warunkiem, że dana rezerwacja oraz stolik istnieją oraz, że ten stolik nie jest już o wybranej godzinie zarezerwowany. Procedura jako argumenty przyjmuje ReservationID (ID rezerwacji) oraz TableID (ID stolika).

```
CREATE procedure AddTableToReservation @ReservationID int, @TableID int
as
    begin try
        if not exists(select * from Tables where TableID = @TableID)
        begin
            throw 52000, N'Taki stolik nie istnieje',1
        end

        if not exists(select * from Reservations
                      where ReservationID = @ReservationID)
        begin
            throw 52000, N'Taka rezerwacja nie istnieje',1
        end

        declare @StartDate smalldatetime = (select StartDate from Reservations
                                            where ReservationID = @ReservationID)
        declare @EndDate smalldatetime = (select EndDate from Reservations
                                            where ReservationID = @ReservationID)

        declare @isTableReserved int = (select TableID from dbo.GetReservedTablesByDate( @date: @StartDate)
                                            where StartTime >= cast(@StartDate as time) and
                                            EndTime <= cast(@EndDate as time) and
                                            TableID = @TableID)

        if @isTableReserved is not null
            begin
                throw 52000, N'Ten stolik już jest zarezerwowany',1
            end

        insert into ReservationDetails(ReservationID, TableID)
        values (@ReservationID, @TableID)
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania stolika do rezerwacji: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

# 4. AddOneTimeDiscountToOrder

Procedura dodaje jednorazową zniżkę do zamówienia pod warunkiem, że takie zamówienie istnieje, jest to klient indywidualny oraz klient posiada odpowiednią, jednorazową, niewykorzystaną i nie przeterminowaną zniżkę. Procedura jako argument przyjmuje OrderID (ID zamówienia).

```
CREATE procedure AddOneTimeDiscountToOrder @OrderID int
as
    begin try
        if not exists(select * from Orders where OrderID = @OrderID)
        begin
            throw 52000, N'Takie zamówienie nie istnieje', 1
        end
        declare @CustomerID int = (select CustomerID from Orders where OrderID = @OrderID)
        if exists(select * from Companies where CustomerID = @CustomerID)
        begin
            throw 52000, N'Klientom firmowym nie przysługują zniżki', 1
        end
        exec dbo.UpdateDiscountsForCustomerSingleUse @CustomerID: @CustomerID
        declare @OrderDate date = (select OrderDate from Orders where OrderID = @OrderID)
        declare @DoneDate date = (select DoneDate from Orders where OrderID = @OrderID)
        declare @DiscountID int = (select top 1 DiscountID from Discounts
                                   join Variables V on Discounts.VariableID = V.VariableID
                                   where CustomerID = @CustomerID and DiscountType = 'R2' and Used = 0 and
                                   ((datediff(day, StartDate, @OrderDate) between 0 and V.D1) or
                                       datediff(day, StartDate, @DoneDate) between 0 and V.D1))
        if @DiscountID is null
        begin
            throw 52000, N'Klient nie posiada odpowiedniej zniżki', 1
        end
        declare @Discount decimal(3,2) = (select V.R2 from Discounts
                        join Variables V on Discounts.VariableID = V.VariableID
                        where @DiscountID = DiscountID)
        update Orders
        set Discount = @Discount
        where OrderID = @OrderID
        update Discounts
        set Used = 1, OrderID = @OrderID
        where DiscountID = @DiscountID
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania jednorazowej zniżki do zamówienia: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 5. InsertToMenu

Procedura dodaje nowe produkty do istniejącego menu. Jako argument przyjmuje MenuID oraz ProductId

```
CREATE PROCEDURE AddProductToMenu
        @menuID int,
        @productID int
    AS
    BEGIN
        IF (NOT EXISTS(Select * from Menu where @menuID=menuID))
            BEGIN
                RAISERROR ('menu not exist',-1,-1)
                RETURN
            END
        IF (NOT EXISTS(Select * from Products where @productID=ProductID))
            BEGIN
                RAISERROR ('product not exist',-1,-1)
                RETURN
            END
        INSERT INTO MenuDetails(MenuID, ProductID)
        VALUES (@menuID,@productID)
        if (dbo.GetIsMenuValid( @currentMenu: @menuid)=1)
        BEGIN
        update Menu
        set IsValid=1
        where menuid=@menuID
        END
    END
go
```

## 6. CreateIndividualClient

Procedura dodaje nowego klienta indywidualnego.Jako argument przyjmuje jego imię, nazwisko, adres oraz numer telefonu

```
CREATE PROCEDURE createIndividualClient
    @fistname nchar(30),
    @lastname nchar(30),
    @address nchar(100),
    @phone nchar(9)
AS
BEGIN
    INSERT INTO Customers(CustomerID)
    VALUES(dbo.MaxIndexFunction()+1)
    INSERT INTO IndividualCustomers(CustomerID,FirstName,LastName,Address,Phone)
    VALUES ( dbo.MaxIndexFunction(),@Fistname,@Lastname,@address,@phone)

END
```

## 7. CreateCompanies

Procedura dodaje nowego klienta firmowego. Jako argument przyjmuje jego nazwę firmy, numer NIP, adres, i telefon.

```
CREATE PROCEDURE createCompanies
    @Companyname nchar(30),
    @NIP nchar(30),
    @address nchar(30),
    @phone nchar(9)
AS
BEGIN
    INSERT INTO Customers(CustomerID)
    VALUES(dbo.MaxIndexFunction()+1)
    INSERT INTO Companies(CustomerID,CompanyName,Nip,Address,Phone)
    VALUES ( dbo.MaxIndexFunction(),@Companyname,@NIP,@address,@phone)

END
```

## 8. CancelOrder

Procedura usuwa całe zamówienie,jeśli data jego wykonania jest nie mniejsza niż aktualna. Argumentem jest OrderId - zamówienie do usunięcia.

```
CREATE PROCEDURE CancelOrder(@orderID int)
    AS
    BEGIN
        IF(NOT EXISTS (SELECT OrderID FROM orders WHERE OrderID= @orderID))
        BEGIN
            RAISERROR ('Order not exist',-1,-1)
            RETURN
        END
        IF( ((SELECT DoneDate from Orders where OrderID=@orderID) < GETDATE()))
        BEGIN
            RAISERROR('Orders is complete',-1,-1)
            RETURN
        END

        DELETE FROM Orders
        Where Orders.OrderID=@orderID

        DELETE FROM OrderDetails
        Where OrderDetails.OrderID=@orderID

        declare @ReservationID int = (select ReservationID from Reservations where @orderID = OrderID)
        exec CancelReservation @reservationID = @ReservationID
    END
go
```

## 9. CancelReservation

Procedura usuwa wybraną rezerwacja oraz gości. Argumentem jest reservationID.

```
CREATE PROCEDURE CancelReservation(@reservationID int)
    AS
    BEGIN
        IF(NOT EXISTS (SELECT ReservationID FROM Reservations WHERE ReservationID= @reservationID))
        BEGIN
            RAISERROR ('Reservation not exist',-1,-1)
            RETURN
        END
        IF( ((SELECT StartDate from Reservations where ReservationID=@reservationID) < GETDATE()))
        BEGIN
            RAISERROR('Reservation is complete',-1,-1)

        END

        DELETE FROM ReservationDetails
        Where ReservationDetails.ReservationID=@reservationID

        DELETE FROM Reservations
        Where Reservations.ReservationID=@reservationID

        DELETE FROM Guests
        Where Guests.ReservationID=@reservationID
    END
go
```

## 10. ChangeIndividualCustomer

Procedura zmienia dane indywidualnego klienta- jego imię,nazwisko, adres,telefon.Przyjmuje te dane wraz z cutomerid jako argument.

```
CREATE PROCEDURE ChangeIndividualCustomer( @firstname nchar(30), @lastname nchar(30),@address nchar(100),@phone char(9),@customerid int)
AS
BEGIN
    IF(NOT EXISTS (SELECT customerId from IndividualCustomers where CustomerID=@customerID))
        BEGIN
            RAISERROR ('customer not exist',-1,-1)
        END
    ELSE
        BEGIN
        BEGIN TRANSACTION tran2
        BEGIN TRY
        UPDATE IndividualCustomers
        SET FirstName=@firstname,
            LastName=@lastname,
            Address=@address,
            phone=@phone
        WHERE CustomerID=@customerid
        COMMIT TRANSACTION Tran2
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION Tran2
        END CATCH
    END
END
```

## 11. ChangeCompanies

Procedura zmienia dane klienta firmowego - nazwę firmy,numer NIP, adres. Przyjmuje te dane wraz z customerid jako argument.

```
CREATE PROCEDURE ChangeCompanies( @Companyname nchar(30), @NIP nchar(10),@address nchar(100),@phone char(9),@customerid int)
AS
BEGIN
    IF(NOT EXISTS (SELECT customerId from Companies where CustomerID=@customerID))
        BEGIN
            RAISERROR ('customer not exist',-1,-1)
        END
    ELSE
        BEGIN
        BEGIN TRANSACTION tran2
        BEGIN TRY
        UPDATE Companies
        SET CompanyName=@Companyname,
            NIP=@NIP,
            Address=@address,
            phone=@phone
        WHERE CustomerID=@customerid
        COMMIT TRANSACTION Tran2
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION Tran2
        END CATCH
    END
END
```

## 12. ChangeEmployees

Procedura zmienia dane pracownika- jego imię, nazwisko. Przyjmuje te dane wraz z employeeID jako argument.

```
CREATE PROCEDURE ChangeEmployees( @firstname nchar(30), @lastname nchar(30),@employeeID int)
AS
BEGIN
    IF(NOT EXISTS (SELECT EmployeeID from CompaniesEmployees where EmployeeID=@employeeID))
        BEGIN
            RAISERROR ('Employee not exist',-1,-1)
        END
    ELSE
        BEGIN
        BEGIN TRANSACTION tran2
        BEGIN TRY
        UPDATE CompaniesEmployees
        SET FirstName=@firstname,
            LastName=@lastname
        WHERE EmployeeID=@employeeID
        COMMIT TRANSACTION Tran2
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION Tran2
        END CATCH
    END
END
```

## 13. ChangeOrder

Procedura usuwa wybrany produkt z istniejącego już zamówienia. Jako argument przyjmuje orderid i productid- produkt który usuwamy z danego zamówienia.

```
CREATE PROCEDURE ChangeOrder(@orderID int,@productid int )
AS
BEGIN
    IF(NOT EXISTS (SELECT OrderID FROM OrderDetails WHERE OrderID= @orderID AND productId=@productid))
    BEGIN
        RAISERROR ('Order not exist',-1,-1)
        RETURN
    END


    DELETE FROM OrderDetails
    Where OrderDetails.OrderID=@orderID
    AND OrderDetails.ProductID=@productid
END
```

## 14. AddForeverDiscountToOrder

Procedura dodaje dożywotnią zniżkę do zamówienia pod warunkiem, że takie zamówienie istnieje, jest to klient indywidualny oraz klient posiada odpowiednią, aktualną, dożywotnią zniżkę. Procedura jako argument przyjmuje OrderID (ID zamówienia).

```
CREATE procedure AddForeverDiscountToOrder @OrderID int                                                    17
as
    begin try
        if not exists(select * from Orders where OrderID = @OrderID)
        begin
            throw 52000, N'Takie zamówienie nie istnieje', 1
        end

        declare @CustomerID int = (select CustomerID from Orders where OrderID = @OrderID)

        if exists(select * from Companies where CustomerID = @CustomerID)
        begin
            throw 52000, N'Klientom firmowym nie przysługują zniżki', 1
        end
        exec UpdateDiscountsForCustomerLongTerm  @CustomerID: @CustomerID
        declare @OrderDate date = (select OrderDate from Orders where OrderID = @OrderID)
        declare @DoneDate date = (select DoneDate from Orders where OrderID = @OrderID)
        declare @DiscountID int = (select top 1 DiscountID from Discounts
                                   where CustomerID = @CustomerID and DiscountType = 'R1' and Used = 0 and
                                   (StartDate <= @OrderDate or StartDate <= @DoneDate))
        if @DiscountID is null
        begin
            throw 52000, N'Klient nie posiada odpowiedniej zniżki', 1
        end

        declare @Discount decimal(3,2) = (select V.R1 from Discounts
                join Variables V on Discounts.VariableID = V.VariableID
                where @DiscountID = DiscountID)
        update Orders
        set Discount = @Discount
        where OrderID = @OrderID
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania dożywotniej zniżki do zamówienia: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 15. AddGuestToReservation

Procedura dodaje nowego gościa do rezerwacji pod warunkiem, że taka rezerwacja i pracownik istnieją, jest to rezerwacja klienta firmowego i do której należy pracownik dodawany, a także, że łączna liczba gości nie przekracza łącznej liczby dostępnych miejsc przy zarezerwowanych stolikach. Procedura jako argumenty przyjmuje ReservationID (ID rezerwacji) oraz EmployeeID lub FirstName i LastName (czyli można wprowadzić ID pracownika bądź jego imię i nazwisko).

```
CREATE procedure AddGuestToReservation @ReservationID int, @EmployeeID int = null,
                                       @FirstName nchar(30) = null, @LastName nchar(30) = null
as
    begin try
        if not exists(select * from Reservations where @ReservationID = ReservationID)
        begin
            throw 52000, N'Taka rezerwacja nie istnieje',1
        end

        if not exists(select * from CompaniesEmployees
                    where @EmployeeID = EmployeeID or (@FirstName = FirstName and @LastName = LastName))
        begin
            throw 52000, N'Taki pracownik nie istnieje',1
        end

        if @EmployeeID is null
        begin
            set @EmployeeID = (select EmployeeID from CompaniesEmployees
                               where @FirstName = FirstName and @LastName = LastName)
        end

        declare @CustomerID int = (select O.CustomerID from Orders O
                        join Reservations R2 on O.OrderID = R2.OrderID
                        and @ReservationID = ReservationID)

        if not exists(select * from Companies where CustomerID = @CustomerID)
        begin
            throw 52000, N'Nie można dodać gości do rezerwacji klienta indywidualnego',1
        end

        if not exists(select * from CompaniesEmployees
                        where @EmployeeID = EmployeeID and @CustomerID = CompanyID)
        begin
            throw 52000, N'Ten pracownik nie należy do firmy rezerwującej',1
        end

        declare @SpotsMax int = (select sum(T.Spots) from Tables T
```

```
                                join ReservationDetails RD on T.TableID = RD.TableID
                                where ReservationID = @ReservationID)

        declare @SpotsTaken int = (select count(*) from Guests
                                where @ReservationID = ReservationID)

        if @SpotsTaken >= @SpotsMax
        begin
            throw 52000, N'Dodano już maksymalną liczbę gości do rezerwacji', 1
        end

        insert into Guests(ReservationID, EmployeeID)
        values (@ReservationID, @EmployeeID)
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania gościa do rezerwacji: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

# 16. AddReservation

Procedura dodaje nową rezerwację, dla istniejącego zamówienia. Procedura dodaje nową rezerwację do Reservations pod warunkiem, że zamówienie (OrderID) oraz klient (CustomerID) istnieją, daty planowanej rezerwacji są nie późniejsze niż aktualna data (StartDate i EndDate). Oprócz tego, odrzuca rezerwację, jeśli klient jest klientem indywidualnym i nie spełnia warunków dokonania rezerwacji. Po przejściu warunków, procedura dodaje stoliki do rezerwacji, zgodnie z określoną liczbą gości, spośród dostępnych w danym terminie.

```
CREATE procedure AddReservation @OrderID int, @StartDate smalldatetime, @EndDate smalldatetime, @NumberOfGuests int
as
    begin try
        if not exists(select * from Orders where @OrderID = OrderID)
        begin
            throw 52000, N'Takie zamówienie nie istnieje', 1
        end

        if @StartDate is null or @StartDate < GETDATE()
        begin
            throw 52000, N'Niepoprawna data początku rezerwacji', 1
        end

        if @EndDate is null or @EndDate < getDate()
        begin
            throw 52000, N'Niepoprawna data końca rezerwacji', 1
        end

        if @StartDate >= @EndDate
        begin
            throw 52000, N'Rezerwacja nie może zacząć się później niż zakończyć', 1
        end

        if exists(select * from Orders where @OrderID = OrderID and IsTakeOut = 1)
        begin
            throw 52000, N'Zamówienie na wynos nie może być rezerwacją', 1
        end

        if @NumberOfGuests is null or @NumberOfGuests < 2
        begin
            throw 52000, N'Niepoprawna liczba gości do rezerwacji', 1
        end

        declare @CustomerID int = (select CustomerID from Orders where @OrderID = OrderID)

        declare @isIndividual bit = (select dbo.GetIsCustomerIndividual( @CustomerID: @CustomerID))
```

```
    if @isIndividual is null
    begin
        throw 52000, N'Nie znaleziono klienta', 1
    end

    if @isIndividual = 1
    begin
        declare @CountOrders int = (select count(*) from Orders where @CustomerID = CustomerID and
                                    OrderDate < (select OrderDate from Orders where OrderID = @OrderID))
        declare @SumOfOrder money = (select sum(OD.Quantity*OD.Price*(1-O.Discount)) from OrderDetails OD
                    join Orders O on OD.OrderID = O.OrderID)

        if @CountOrders < (select top 1 WK from Variables order by VariableID desc) or
           @SumOfOrder < (select top 1 WZ from Variables order by VariableID desc)
        begin
            throw 52000, N'Klient indywidualny nie spełnia warunków dokonania rezerwacji', 1
        end
    end

    insert into Reservations(OrderID, StartDate, EndDate)
    values(@OrderID, @StartDate, @EndDate)

    declare @ReservationID int = (select ReservationID from Reservations where @OrderID = OrderID)

    declare @CountSpots int = 0
    declare @FreeTableID int
    while @CountSpots < @NumberOfGuests
        begin
            set @FreeTableID = (select top 1 TableID from Tables
                                where TableID not in
                                      (select TableID from GetReservedTablesFromDateToDate(
                                          @StartDate: @StartDate, @EndDate: @EndDate)
                                       where StartDate >= @StartDate and
                                       EndDate <= @EndDate))
            if @FreeTableID is not null
                begin
                    exec AddTableToReservation @ReservationID = @ReservationID, @TableID = @FreeTableID
                    set @CountSpots = @CountSpots + (select Spots from Tables where TableID = @FreeTableID)
                end
            else
                begin
                    exec CancelReservation @reservationID = @ReservationID;
                    throw 52000, N'Brak wolnych stolików na ten termin', 1
                end
        end
    update Orders
    set DoneDate = @StartDate
    where OrderID = @OrderID
end try
begin catch
    declare @error nchar(1000) = N'Wystąpił błąd dodania nowej rezerwacji: ' + ERROR_MESSAGE();
    throw 52000, @error, 1
end catch
go
```

## 17. AddOrder

Procedura dodaje nowe zamówienie do Orders, pod warunkiem, że podany klient istnieje oraz data wykonania jest nie wcześniejsza niż aktualna data. Procedura przyjmuje opcjonalny argument - potrzebny przy ustalaniu czy jest to zamówienie na wynos (isTakeOut).

```
CREATE procedure AddOrder @CustomerID int ,@DoneDate smalldatetime, @IsTakeOut bit = 0, @IsPaid bit = 0
as
    begin try
        if not exists(select * from Customers where @CustomerID = CustomerID)
        begin
            throw 52000, N'Taki klient nie istnieje', 1
        end

        if @DoneDate < GETDATE()
        begin
            throw 52000, N'Niepoprawna data wykonania zamówienia', 1
        end

        insert into Orders(CustomerID, OrderDate, Paid, DoneDate, EmployeeID, IsTakeOut)
        values(@CustomerID, getDate(), @IsPaid, @DoneDate, 7, @IsTakeOut)

    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania nowego zamówienia: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 18. AddCategory

Procedura dodaje nową kategorię produktów. Argumentem jest nazwa nowej kategorii.

```
CREATE procedure AddCategory @CategoryName nchar(50)
as
    begin
        begin try
            if exists(
                select *
                from Categories
                where @CategoryName = CategoryName
                )
            begin
                throw 52000, 'Taka kategoria już istnieje', 1
            end

            insert into Categories(CategoryName)
            values (@CategoryName)

        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodania nowej kategorii: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 19. AddCompanyEmployee

Procedura dodaje nowego pracownika dla danej firmy. Argumentem jest nazwa firmy, imię i nazwisko.

```
CREATE procedure AddCompanyEmployee @CompanyID int,
                          @FirstName nchar(30),
                          @LastName nchar(30)
as
    begin
        begin try

            if not exists(select CustomerID from Companies
                                  where CustomerID = @CompanyID)
                begin
                    throw 52000, 'Brak firmy o zadanym id', 1
                end

            insert into dbo.CompaniesEmployees(CompanyID, FirstName, LastName)
            values (@CompanyID, @FirstName, @LastName)
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodania nowego pracownika firmy: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 20. AddTable

Procedura dodaje nowy stolik. Argumentem jest ilość miejsc przy stoliku.

```
CREATE procedure AddTable @Spots int
as
    begin
        begin try
            insert into dbo.Tables(Spots)
            values (@Spots)
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodania nowego stolika: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

# 21. AddProduct

Procedura dodaje nowy produkt. Argumentem jest nazwa nowego produktu, jego kategoria i cena.

```
CREATE procedure AddProduct @ProductName nchar(50),
                            @CategoryName nchar(50),
                            @Price money
as
    begin
        begin try
            if exists(
                select *
                from Products
                where @ProductName = ProductName
                )
            begin
                throw 52000, 'Taki produkt już istnieje', 1
            end

            if not exists(
                select * from Categories
                where @CategoryName = CategoryName
                )
                begin
                    throw 52000, 'Nie ma takiej kategorii', 1
                end

            declare @CategoryID int
            select @CategoryID = (select CategoryID from Categories
                                  where CategoryName = @CategoryName)

            insert into Products(ProductName, CategoryID, Price)
            values (@ProductName, @CategoryID, @Price)
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodania nowego produktu: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 22. AddRestaurantEmployee

Procedura dodaje nowego pracownika restauracji. Argumentami są: imię, nazwisko, data zatrudnienia, stanowisko, telefon, adres i id przełożonego.

```
CREATE procedure AddRestaurantEmployee @FirstName nchar(30),
                           @LastName nchar(30),
                           @HireDate date,
                           @Position nchar(50),
                           @Phone nchar(9),
                           @Address nchar(100),
                           @ReportsTo int

as
    begin
        begin try
            if not exists(
                select * from Employees
                where @ReportsTo = EmployeeID
                )
                begin
                    throw 52000, 'Pracownik o takim id nie istnieje, wiec nie może być szefem', 1
                end

            insert into dbo.Employees(FirstName, LastName, HireDate, Position, Phone, Address, ReportsTo)
            values (@FirstName, @LastName, @HireDate, @Position, @Phone, @Address, @ReportsTo)
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodania nowego pracownika: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 23. CreateNewMenu

Procedura tworzy nowe menu. Argumentami są daty rozpoczęcia i zakończenia obowiązywania menu.

```
CREATE procedure CreateNewMenu @startDate date,
                               @endDate date
as
    begin
        begin try
            declare @previousID int = (select top 1 MenuID from Menu order by MenuID desc)

            if @startDate <= (select EndDate from Menu where MenuID = @previousID)
            begin
                throw 52000, N'Nowe menu nie może się pokrywać datami z poprzednim', 1
            end

            if @endDate < @startDate
            begin
                throw 52000, N'Menu nie może obowiązywać mniej niż 1 dzień', 1
            end

            insert into dbo.Menu(startdate, enddate, isvalid)
            values (@startDate, @endDate, 0)

        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd tworzenia nowego menu: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 24. ModifyProductPrice

Procedura zmienia cenę istniejącego produktu. Argumentami są nazwa produktu i nowa cena.

```
CREATE procedure ModifyProductPrice @ProductName nchar(50) , @NewPrice money
as
    begin
        begin try
            if not exists(select ProductID from Products
                                where ProductName = @ProductName)
                begin
                    throw 52000, 'Nie ma produktu o takiej nazwie', 1
                end

            update Products
                set Price = @NewPrice
                where ProductName = @ProductName
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd zmiany ceny produktu: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 25. AddIncome

Procedura dodaje wpłatę. Argumentami są id zamówienia i ilość pieniędzy.

```
create procedure AddIncome @OrderID int , @AmountOfMoney money
as
    begin
        begin try
            if not exists(select OrderID from Orders
                                where OrderID = @OrderID)
                begin
                    throw 52000, 'Nie ma zamowienia o takim id', 1
                end

            insert into dbo.Incomes(OrderID, AmountOfMoney, Date)
            values (@OrderID, @AmountOfMoney, getdate())

            exec isOrderPaid @OrderID: @OrderID

        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd dodawania nowej wpłaty: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

# 26. IsOrderPaid

Procedura sprawdza czy dane zamówienie zosało opłacone, jeżeli tak to zmienia wartość komórki isPaid w tabeli Orders.

```
CREATE procedure IsOrderPaid @OrderID int
as
    begin
        begin try
            declare @OrderPrice int
            declare @PaidAmount money

            select @OrderPrice = (select sum((OD.Quantity * OD.Price) * (1 - (O.Discount)))
                from Orders O
                inner join OrderDetails OD on O.OrderID = OD.OrderID
                where O.OrderID = @OrderID)

            select @PaidAmount = (select sum(AmountOfMoney) from Incomes
                                  where OrderID = @OrderID)

            if @OrderPrice <= @PaidAmount
            begin
                update Orders
                set Paid = 1
                where OrderID = @OrderID
            end

            else
            begin
                throw 52000, N'Zamówienie nie jest jeszcze opłacone', 1
            end
        end try

        begin catch
            declare @error nchar(1000) = N'Wystąpił błąd przy sprawdzaniu czy zamówienie jest zapłacone: ' + ERROR_MESSAGE();
            throw 52000, @error, 1
        end catch
    end
go
```

## 27. AddBill

Procedura dodaje dla klienta nowy rachunek do tabeli rachunków, pod warunkiem, że argumenty istnieją, zamówienie należy do tego klienta oraz zostało opłacone. Jako argumenty przyjmuje ID klienta (CustomerID) i ID zamówienia, które chcemy dodać do rachunku (OrderID). Procedura dodaje też do odpowiedniego zamówienia w Orders numer stworzonego rachunku.

```
CREATE procedure AddBill @CustomerID int, @OrderID int
as
    begin try
        if not exists(select * from Customers where @CustomerID = @CustomerID)
        begin
            throw 52000, N'Taki klient nie istnieje', 1
        end

        if not exists(select * from Orders where OrderID = @OrderID)
        begin
            throw 52000, N'Takie zamówienie nie istnieje', 1
        end

        if not exists(select * from Orders where OrderID = @OrderID and @CustomerID = CustomerID)
        begin
            throw 52000, N'To zamówienie nie należy do tego klienta', 1
        end

        if exists(select * from Orders where OrderID = @OrderID and BillID is not null)
        begin
            throw 52000, N'To zamówienie ma już przypisany rachunek', 1
        end

        exec IsOrderPaid @OrderID = @OrderID

        declare @SummaryValue money = (select isnull(sum(OD.Price*OD.Quantity*(1-Orders.Discount)),0) from Orders
                                       left join OrderDetails OD on Orders.OrderID = OD.OrderID
                                       where Orders.OrderID = @OrderID)

        declare @NIP nchar(50) = (select NIP from Companies where CustomerID = @CustomerID)

        insert into Bills(Date, CustomerID, NIP, SummaryValue)
        values (getdate(), @CustomerID, @NIP, @SummaryValue)

        declare @BillID int = (select top 1 BillID from Bills order by BillID desc)


         update Orders
         set BillID = @BillID
         where OrderID = @OrderID
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania nowego rachunku: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 28. AddOrderToBill

Procedura umożliwia dodanie kolejnego zamówienia do już istniejącego rachunku w tabeli Bills, pod warunkiem, że argumenty istnieją, zamówienie należy do właściciela rachunku oraz jest już opłacone. Procedura jako argumenty przyjmuje ID rachunku (BillID) oraz ID zamówienia (OrderID). Procedura dodaje też do odpowiedniego zamówienia w Orders numer wybranego rachunku.

```
CREATE procedure AddOrderToBill @BillID int, @OrderID int
as
    begin try
        if not exists(select * from Bills where @BillID = BillID)
        begin
            throw 52000, N'Taki rachunek nie istnieje', 1
        end

        if not exists(select * from Orders where OrderID = @OrderID)
        begin
            throw 52000, N'Takie zamówienie nie istnieje', 1
        end

        if exists(select * from orders where OrderID = @OrderID and BillID is not null)
        begin
            throw 52000, N'To zamówienie ma już przypisany rachunek', 1
        end

        declare @CustomerID int = (select CustomerID from Bills where @BillID = BillID)

        if not exists(select * from Orders where @OrderID = OrderID and @CustomerID = CustomerID)
        begin
            throw 52000, N'To zamówienie nie należy do właściciela rachunku', 1
        end

        exec IsOrderPaid @OrderID = @OrderID

        declare @Value money = (select isnull(sum(OD.Price*OD.Quantity*(1-Orders.Discount)),0) from Orders
                                    left join OrderDetails OD on Orders.OrderID = OD.OrderID
                                    where Orders.OrderID = @OrderID)

        declare @OldValue money = (select SummaryValue from Bills where @BillID = BillID)

        update Bills
        set SummaryValue = @Value + @OldValue
        where @BillID = BillID


        update Orders
        set BillID = @BillID
        where @OrderID = OrderID
    end try
    begin catch
        declare @error nchar(1000) = N'Wystąpił błąd dodania zamówienia do rachunku: ' + ERROR_MESSAGE();
        throw 52000, @error, 1
    end catch
go
```

## 29. UpdateDiscountsForCustomerSingleUse

Procedura dodaje do tabeli Discounts zniżkę jednorazową dla danego klienta jeśli ten na nią zasługuje

```
CREATE procedure UpdateDiscountsForCustomerSingleUse @CustomerID int
    as
        begin
            begin try
                declare @Value money
                set @Value = dbo.GetValueOfOrdersByCustomerID( @CustomerID: @CustomerID)

                declare @K2 money
                set @K2 = (select top 1 K2 from Variables order by VariableID desc)

                declare @VariableID int
                set @VariableID = (select top 1 VariableID from Variables order by VariableID desc)

                if(@Value > @K2)
                begin
                    insert into Discounts(CustomerID, DiscountType, VariableID, StartDate, Used, OrderID)
                    values(@CustomerID, 'R2', @VariableID, getdate(), 0, null)
                end
            end try

            begin catch
                declare @error nchar(1000) = N'Wystąpił błąd dodania nowej znizki jednorazowej klientowi ' + ERROR_MESSAGE();
                throw 52000, @error, 1
            end catch
        end
go
```

## 30. UpdateDiscountsForCustomerLongTerm

Procedura dodaje do tabeli Discounts zniżkę dożywotnią dla danego klienta jeśli ten na nią zasługuje

```
CREATE procedure UpdateDiscountsForCustomerLongTerm @CustomerID int
    as
        begin
            begin try

                declare @Z1 int
                set @Z1 = (select top 1 Z1 from Variables order by VariableID desc)

                declare @VariableID int
                set @VariableID = (select top 1 VariableID from Variables order by VariableID desc)

                if(dbo.GetNumberOfOrdersK1ByCustomerID( @input: @CustomerID) > @Z1)
                begin
                    insert into Discounts(CustomerID, DiscountType, VariableID, StartDate, Used, OrderID)
                    values(@CustomerID, 'R1', @VariableID, getdate(), 0, null)
                end
            end try

            begin catch
                declare @error nchar(1000) = N'Wystąpił błąd dodania nowej znizki dożywotniej klientowi ' + ERROR_MESSAGE();
                throw 52000, @error, 1
            end catch
        end
go
```

# Indeksy

1. PK_Bills

```
create unique clustered index PK_Bills
    on Bills (BillID)
go
```

2. IX_Bills_NIP

```
create index IX_Bills_NIP
    on Bills (NIP)
go
```

3. PK_Categories

```
create unique clustered index PK_Categories
    on Categories (CategoryID)
go
```

4. PK_Companies

```
create unique clustered index PK_Companies
    on Companies (CustomerID)
go
```

5. IX_Companies_NIP_Unique

```
create unique index IX_Companies_NIP_Unique
    on Companies (NIP)
go
```

6. PK_Companies

```
create unique clustered index PK_Employees
    on CompaniesEmployees (EmployeeID)
go
```

7. PK_Customers

```
create unique clustered index PK_Customers
    on Customers (CustomerID)
go
```

8. PK_Discounts

```
create unique clustered index PK_Discounts
    on Discounts (DiscountID)
go
```

9. PK_Employees_1

```
create unique clustered index PK_Employees_1
    on Employees (EmployeeID)
go
```

10. PK_Guests

```
create unique clustered index PK_Guests
    on Guests (ReservationID, EmployeeID)
go
```

11. IX_Guests_ReservationID

```
create index IX_Guests_ReservationID
    on Guests (ReservationID)
go
```

12. PK_Incomes

```
create unique clustered index PK_Incomes
    on Incomes (IncomeID)
go
```

13. PK_IndividualCustomers

```
create unique clustered index PK_IndividualCustomers
    on IndividualCustomers (CustomerID)
go
```

14. PK_Menu

```
create unique clustered index PK_Menu
    on Menu (MenuID)
go
```

15. PK_MenuDetails

```
create unique clustered index PK_MenuDetails
    on MenuDetails (MenuID, ProductID)
go
```

16. IX_MenuDetails_MenuID

```
create index IX_MenuDetails_MenuID
    on MenuDetails (MenuID)
go
```

17. PK_OrderDetails

```
create unique clustered index PK_OrderDetails
    on OrderDetails (OrderID, ProductID)
go
```

18. PK_Orders

```
create unique clustered index PK_Orders
    on Orders (OrderID)
go
```

19. PK_Products

```
create unique clustered index PK_Products
    on Products (ProductID)
go
```

20. PK_ReservationDetails

```
create unique clustered index PK_ReservationDetails
    on ReservationDetails (ReservationID, TableID)
go
```

21. PK_Reservations

```
create unique clustered index PK_Reservations
    on Reservations (ReservationID)
go
```

22. IX_Reservations_OrderID_Unique

```
create unique index IX_Reservations_OrderID_Unique
    on Reservations (OrderID)
go
```

23. PK_Tables

```
create unique clustered index PK_Tables
    on Tables (TableID)
go
```

24. PK_Variables

```
create unique clustered index PK_Variables
    on Variables (VariableID)
go
```

# Role

## 1. Pracownik

Pracownik ma dostęp do przyszłych zamówień, wolnych stolików, rezerwacji i zamówień, aktualnego menu , statystyk dla wskazanego klienta, płatnościach klienta. Jest w stanie dodawać  nowe produktu do zamówienia, stoliki do rezerwacji, tworzyć nowych klientów, zmieniać ich dane, nadawać zniżki, dodawać rachunek, i przyjmować wpłaty, dodawać nowe produkty, stoliki, tworzyć nowe menu, rezerwacje, może dodawać gości do rezerwacji, usuwać zamówienia

```
CREATE ROLE worker
GRANT SELECT ON FutureTakeOuts to worker
GRANT SELECT ON FreeTablesForToday to worker
GRANT SELECT ON OncomingReservations to worker
GRANT SELECT ON OncomingOrders to worker
GRANT SELECT ON Actualmenu to worker
GRANT SELECT ON GetmenuProductsByDate to worker
GRANT SELECT ON GetReservedTablesFromDateToDate to worker
GRANT SELECT ON GetTakeoutsInfosByDate to worker
GRANT EXECUTE ON GetToPayInfoForClient to worker
GRANT EXECUTE ON GetIsMenuValid to worker
GRANT SELECT  ON FutureOrdersForClient to worker
GRANT SELECT ON OrdersToPayForCompanies to worker
GRANT EXECUTE ON MaxindexFunction to worker
GRANT SELECT ON GetOrdersByDate to worker
GRANT EXECUTE ON getNumberOfordersK1ByCustomerId to worker
GRANT EXECUTE ON GetNumberOfOrdersByCustomerID to worker
GRANT SELECT ON GetValidDiscountsByCustomerID to worker
GRANT EXECUTE ON GetValueOfOrdersByCustomerID to worker
GRANT EXECUTE ON GetIsCustomerIndividual to worker
GRANT EXECUTE ON ChangeOrder to worker
GRANT EXECUTE ON AddProductToOrder to worker
GRANT EXECUTE ON AddTableToReservation to worker
GRANT EXECUTE ON AddOneTimeDiscountToOrder to worker
GRANT EXECUTE ON InsertToMenu to worker
GRANT EXECUTE ON CancelReservation to worker
GRANT EXECUTE ON CreateIndividualClient to worker
GRANT EXECUTE ON CreateCompanies to worker
GRANT EXECUTE ON CancelOrder to worker
GRANT EXECUTE ON ChangeIndividualCustomer to worker
GRANT EXECUTE ON ChangeCompanies to worker
GRANT EXECUTE ON AddForeverDiscountToOrder to worker
GRANT EXECUTE ON AddGuestToReservation to worker
GRANT EXECUTE ON AddReservation to worker
GRANT EXECUTE ON AddOrder to worker
GRANT EXECUTE ON AddCompanyEmployee to worker
GRANT EXECUTE ON AddProduct to worker
GRANT EXECUTE ON AddTable to worker
GRANT EXECUTE ON CreateNewMenu to worker
GRANT EXECUTE ON ModifyProductPrice to worker
GRANT EXECUTE ON AddIncome to worker
GRANT EXECUTE ON IsOrderPaid to worker
GRANT EXECUTE ON AddBill to worker
GRANT EXECUTE ON AddOrderToBill to worker
```

## 2. Manager

Manager a te same uprawnienia, dodatkowe ma wgląd do statystyk wszystkich zamówień , rabatów, klientów, ich płatności , zmienia warianty zniżek , zmienia pracowników, dodaje kategorie i wprowadza nowych pracowników restauracji

```
CREATE ROLE manager
GRANT SELECT ON FutureTakeOuts to manager
GRANT SELECT ON FreeTablesForToday to manager
GRANT SELECT ON OncomingReservations to manager
GRANT SELECT ON OncomingOrders to manager
GRANT SELECT ON Actualmenu to manager
GRANT SELECT ON GetmenuProductsByDate to manager
GRANT SELECT ON GetReservedTablesFromDateToDate to manager
GRANT SELECT ON GetTakeoutsInfosByDate to manager
GRANT EXECUTE ON GetToPayInfoForClient to manager
GRANT EXECUTE ON GetIsMenuValid to manager
GRANT SELECT  ON FutureOrdersForClient to manager
GRANT SELECT ON OrdersToPayForCompanies to manager
GRANT EXECUTE ON MaxindexFunction to manager
GRANT SELECT ON GetOrdersByDate to manager
GRANT EXECUTE ON getNumberOfordersK1ByCustomerId to manager
GRANT EXECUTE ON GetNumberOfOrdersByCustomerID to manager
GRANT SELECT ON GetValidDiscountsByCustomerID to manager
GRANT EXECUTE ON GetValueOfOrdersByCustomerID to manager
GRANT EXECUTE ON GetIsCustomerIndividual to manager
GRANT EXECUTE ON ChangeOrder to manager
GRANT EXECUTE ON AddProductToOrder to manager
GRANT EXECUTE ON AddTableToReservation to manager
GRANT EXECUTE ON AddOneTimeDiscountToOrder to manager
GRANT EXECUTE ON InsertToMenu to manager
GRANT EXECUTE ON CancelReservation to manager
GRANT EXECUTE ON CreateIndividualClient to manager
GRANT EXECUTE ON CreateCompanies to manager
GRANT EXECUTE ON CancelOrder to manager
GRANT EXECUTE ON ChangeIndividualCustomer to manager
GRANT EXECUTE ON ChangeCompanies to manager
GRANT EXECUTE ON AddForeverDiscountToOrder to manager
GRANT EXECUTE ON AddGuestToReservation to manager
GRANT EXECUTE ON AddReservation to manager
```

```
GRANT EXECUTE ON AddOrder to manager
GRANT EXECUTE ON AddCompanyEmployee to manager
GRANT EXECUTE ON AddProduct to manager
GRANT EXECUTE ON AddTable to manager
GRANT EXECUTE ON CreateNewMenu to manager
GRANT EXECUTE ON ModifyProductPrice to manager
GRANT EXECUTE ON AddIncome to manager
GRANT EXECUTE ON IsOrderPaid to manager
GRANT EXECUTE ON AddBill to manager
GRANT EXECUTE ON AddOrderToBill to manager
GRANT SELECT ON MonthlyTablesStats to manager
GRANT SELECT ON WeeklyTablesStats to manager
GRANT SELECT ON WeeklyDicountStats to manager
GRANT SELECT ON IndividualClientsStats to manager
GRANT SELECT ON CompaniesStats to manager
GRANT SELECT ON MonthlyMenuStats to manager
GRANT SELECT ON WeeklyMenuStats to manager
GRANT SELECT ON OrdersStats to manager
GRANT SELECT ON ToPay to manager
GRANT SELECT ON CompaniesStatsFunction to manager
GRANT SELECT ON IndividualStatsFunction to manager
GRANT EXECUTE ON AddVariables to manager
GRANT EXECUTE  ON ChangeEmployees to manager
GRANT EXECUTE ON AddCategory to manager
GRANT EXECUTE  ON AddRestaurantEmployee to manager
```