Projekt Bazy Restauracja

Podstawy Baz Danych 2022/2023

Autorzy: Adam Górka, Adrian Stahl, Piotr Wiercigroch

Opis funkcji systemu wraz z informacją, co jaki użytkownik może wykonywać w systemie

System

- Udostępnia menu dla wszystkich
- Sczytuje zlecone zamówienia na wynos przez formularz WWW dla danego klienta
- Przydzieli dany stolik

Zarząd

- Edytowanie współczynników rabatów [Z1, K1, R1 \ K2, R2, D1]
- Edytowanie współczynników minimalnej wartości zamówienia WK i minimalnej ilości zamówień WZ
- Edycja listy stolików
- Ustalenie, zmiana menu
- Zamawianie jedzenia (2)
- Zmiana cen dań/produktów
- Generowanie raportów miesięcznych i tygodniowych dotyczących rabatów, rezerwacji stolików, menu i statystyk zamówień

Pracownik

- wpisywanie rezerwacji [na miejscu]
- Akceptacja rezerwacji (1)
- Wystawianie faktury
- Wstawianie zamówień stacjonarnych, wypełnienie informacji o nich
- Możliwość sprawdzenia ilości wolnych/zajętych stolików
- Możliwość dodania zamówienia na wynos w przypadku zlecenia go na miejscu

Klient (ogólnie)

- Możliwość złożenia zamówienia na wynos (?)
- Generowanie statystyk zamówienia dla klientów indywidualnych oraz firm

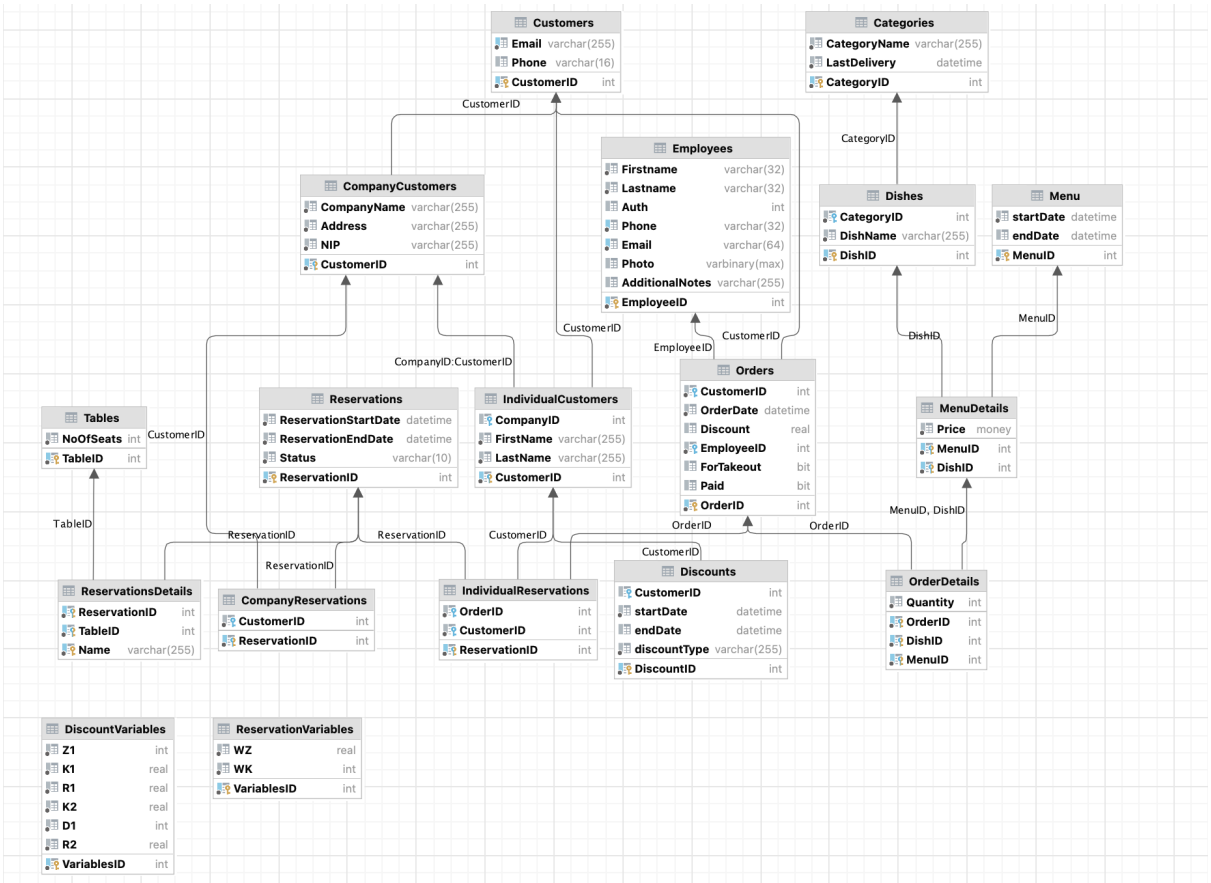
Klient (firma)

- Rezerwacja stolika na firmę (bez żadnych wymagań wstępnych)
- Rezerwacja stolika imiennie dla pracowników

Klient (prywatny)

- Możliwość sprawdzenia informacji o rabacie
- Rezerwacja stolika imiennie, na co najmniej dwie osoby oraz muszą posiadać minimalną wartość zamówienia WZ, przy dokonaniu wcześniej WK zamówień

Schemat i opis tabeli



Customers

Kolumny

- CustomerID - klucz główny klienta
- Phone - telefon do klienta indywidualnego lub do firmy
- Email - email klienta indywidualnego lub firmy

Warunki integralnościowe

- [Phone] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
- [Email] like '%@%'

Kod SQL

```
create table Customers
(
    CustomerID int          not null
        constraint Customers_pk
            primary key,
    Email      varchar(255) not null
        constraint email_unique
            unique
        constraint validateEmail
            check ([Email] like '%@%'),
    Phone      varchar(16)
        constraint validatePhone
            check ([Phone] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
```

CompanyCustomers

- CustomerID - klucz główny firmy
- CompanyName - Nazwa firmy
- Address - Adres firmy
- NIP

Warunki integralnościowe

- [NIP] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'

Kod SQL

```
create table CompanyCustomers
(
    CustomerID  int          not null
        constraint CompanyCustomers_pk
            primary key
        constraint CompanyCustomers_Customers_null_fk
            references Customers,
    CompanyName varchar(255) not null,
    Address     varchar(255) not null,
    NIP         varchar(255) not null
        constraint validateNIP
            check ([NIP] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
```

DiscountVariables

- VariablesID
- Z1
- K1
- R1
- K2
- D1
- R2

Warunki integralnościowe

- [Z1] > 0
- [K1] > 0
- [R1] > 0 and [R1] <= 1
- [K2] > 0
- [D1] > 0
- [R2] > 0 and [R2] <= 1

Kod SQL

```
create table DiscountVariables
(
    Z1          int  not null
        constraint validateZ1
            check ([Z1] > 0),
    K1          real not null
        constraint validateK1
            check ([K1] > 0),
    R1          real not null
        constraint validateR1
            check ([R1] > 0 AND [R1] <= 1),
    K2          real not null
        constraint validateK2
            check ([K2] > 0),
    D1          int  not null
        constraint validateD1
            check ([D1] > 0),
    R2          real not null
        constraint validateR2
            check ([R2] > 0 AND [R2] <= 1),
    VariablesID int  not null
        constraint DiscountVariables_pk
            primary key
)
go
```

IndividualCustomers

- CustomerID - Klucz główny
- CompanyID - ID firmy będącej klientem do której należy klient indywidualny
- FirstName - Imię klienta
- LastName - Nazwisko klienta

Kod SQL

```
create table IndividualCustomers
(
    CustomerID int          not null
        constraint IndividualCustomers_pk
            primary key
        constraint IndividualCustomers_Customers_null_fk
            references Customers,
    CompanyID  int
        constraint IndividualCustomers_CompanyCustomers_null_fk
            references CompanyCustomers,
    FirstName  varchar(255) not null,
    LastName   varchar(255) not null
)
```

Employees

- EmployeeID - klucz główny
- FirstName - Imię pracownika
- LastName - Nazwisko pracownika
- Auth - ID pracownika będącego szefem pracownika
- Phone - telefon pracownika
- Email - email pracownika
- Photo - zdjęcie pracownika
- AdditionalNotes - dodatkowe informacje na temat pracownika

Warunki integralnościowe

- [Phone] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
- [Email] like '%@%'

Kod SQL

```
create table Employees
(
    EmployeeID      int         not null
        constraint Employees_pk
            primary key,
    Firstname       varchar(32) not null,
    Lastname        varchar(32) not null,
    Auth            varchar(32),
    Phone           varchar(32) not null
        constraint validatePhoneEmployees
            check ([Phone] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Email           varchar(64) not null
        constraint validateEmailEmployees
            check ([Email] like '%@%'),
    Photo           varbinary(max),
    AdditionalNotes varchar(255)
)
```

Orders

- OrderID - klucz główny
- CutromerID - ID klienta którego jest zamówienie
- OrderDate - Data wykonania zamówienia
- Discount - Zniżka nałożona na zamówienie
- EmployeeID - ID pracownika obsługującego zamówienie
- ForTakeout - informacja czy zamówienie jest na wynos (1-tak, 0-nie)

Warunki integralnościowe

- [Discount] >= 0 and [Discount] <=1
- ForTakeout - defaultowo przyjmuje 1

Kod SQL

```
create table Orders
(
    OrderID    int      not null
        constraint Orders_pk
            primary key,
    CustomerID int      not null
        constraint Orders_Customers_null_fk
            references Customers,
    OrderDate  datetime not null,
    Discount   real
        constraint validateDiscount
            check ([Discount] <= 1 AND [Discount] >= 0),
    EmployeeID int      not null
        constraint Orders_Employees_null_fk
            references Employees,
    ForTakeout bit
        constraint forTakeoutDefault default 1
)
```

OrderDetails

- OrderID - ID zamówienia do którego należy opis
- DishID - ID zakupionego dania
- MenuID - ID menu z którego zostało zamówione danie
- Quantity - ilość porcji zamówionego dania

Warunki integralnościowe

- [Quantity] > 0

Kod SQL

```
create table OrderDetails
(
    OrderID  int not null
        constraint OrderDetails_Orders_null_fk
            references Orders,
    DishID   int not null,
    MenuID   int not null,
    Quantity int not null
        constraint validateQuantityOrderDetails
            check ([Quantity] > 0),
    constraint OrderDetails_pk
        primary key (OrderID, MenuID, DishID),
    constraint OrderDetails_MenuDetails_null_null_fk
        foreign key (MenuID, DishID) references MenuDetails
)
```

ReservationsDetails

- ReservationID - klucz główny
- CustomerID - ID klienta na którego została złożona rezerwacja (nie musi się zgadzać z ID w tabeli Reservations)
- TableID - ID stolika na który jest rezerwacja

Kod SQL

```
create table ReservationsDetails
(
    ReservationID int not null
        constraint ReservationsDetails_pk
            primary key
        constraint ReservationsDetails_CompanyReservations_null_fk
            references CompanyReservations
        constraint ReservationsDetails_IndividualReservations_null_fk
            references IndividualReservations,
    TableID       int not null
        constraint ReservationsDetails_Tables_null_fk
            references Tables,
    CustomerID    int
)
```

ReservationVariables

- VariablesID - klucz główny
- WZ - minimalna wartość zamówienia
- WK - minimalna liczba zamówień

Warunki integralnościowe

- [WZ] > 0
- [WK] > 0

Kod SQL

```
create table ReservationVariables
(
    VariablesID int  not null
        constraint ReservationVariables_pk
            primary key,
    WZ          real not null
        constraint validateWZ
            check ([WZ] > 0),
    WK          int  not null
        constraint validateWK
            check ([WK] > 0)
)
```

Tables

- TableID - klucz główny
- NoOfSeats - liczba miejsc siedzących przy stoliku

Warunki integralnościowe

- [NoOfSeats] > 0

Kod SQL

```
create table Tables
(
    TableID   int not null
        constraint Tables_pk
            primary key,
    NoOfSeats int not null
        constraint validatenoOfSeats
            check ([NoOfSeats] > 0)
)
```

Dishes

- DishID - klucz główny
- DishName - Nazwa dania
- CategoryID - ID kategorii do której należy dane danie

Kod SQL

```
create table Dishes
(
    DishID     int          not null
        constraint Dishes_pk
            primary key,
    CategoryID int          not null
        constraint Dishes_Categories_null_fk
            references Categories,
    DishName   varchar(255) not null
)
```

Categories

- CategoryID - klucz główny
- CategoryName - Nazwa kategorii
- LastDelivery - data ostatniej dostawy artykułów z tej kategorii

Warunki integralnoświowe

- [LastDelivery] < GETDATE()

Kod SQL

```
create table Categories
(
    CategoryID   int          not null
        constraint Categories_pk
            primary key,
    CategoryName varchar(255) not null,
    LastDelivery datetime     not null
        constraint validateLastDelivery
            check ([LastDelivery] < getdate())
)
```

Menu

- MenuID - klucz główny
- StartDate - Data rozpoczęcia obowiązywania menu
- EndDate - Data zakończenia obowiązywania menu

Warunki integralnościowe

- [EndDate] > [StartDate] OR [EndDate] IS NULL

Kod SQL

```
create table Menu
(
    MenuID    int      not null
        constraint Menu_pk
            primary key,
    startDate datetime not null,
    endDate   datetime,
    constraint endDateValidateMenu
        check ([endDate] > [Menu].[startDate] OR [endDate] IS NULL)
)
```

MenuDetails

- MenuID - ID menu do którego należy danie
- DishID - ID dania
- Price - Cena dania z danego menu

Warunki integralnościowe

- [Price] >= 0

Kod SQL

```
create table MenuDetails
(
    MenuID int   not null
        constraint MenuDetails_Menu_null_fk
            references Menu,
    DishID int   not null
        constraint MenuDetails_Dishes_null_fk
            references Dishes,
    Price  money not null
        constraint validatePriceMenu
            check ([Price] >= 0),
    constraint MenuDetails_pk
        primary key (MenuID, DishID)
)
```

Discounts

- DiscountID - klucz główny
- CustomerID - ID klienta do którego należy promocja
- startDate - data rozpoczęcia promocji
- endDate - data zakończenia promocji
- discountType - typ przyznanej promocji

Warunki integralnościowe

- [endDate] > [startDate] OR [endDate] IS NULL

Kod SQL

```
create table Discounts
(
    CustomerID   int
        constraint Discounts_IndividualCustomers_null_fk
            references IndividualCustomers,
    DiscountID   int          not null
        constraint Discounts_pk
            primary key,
    startDate    datetime     not null,
    endDate      datetime,
    discountType varchar(255) not null,
    constraint validateEndDateDiscounts
        check ([endDate] > [startDate] OR [endDate] IS NULL)
)
```

CompanyReservations

- ReservationID - ID rezerwacji
- CustomerID - ID klienta

Kod SQL

```
create table CompanyReservations
(
    ReservationID int not null
        constraint CompanyReservations_pk
            primary key
        constraint CompanyReservations_Reservations_null_fk
            references Reservations,
    CustomerID    int not null
        constraint CompanyReservations_CompanyCustomers_null_fk
            references CompanyCustomers
)
go
```

IndividualReservations

- ReservationID - ID rezerwacji
- OrderID - ID zamówienia
- CustomerID - ID klienta

Kod SQL

```
create table IndividualReservations
(
    ReservationID int not null
        constraint IndividualReservations_pk
            primary key
        constraint IndividualReservations_Reservations_null_fk
            references Reservations,
    OrderID       int not null
        constraint IndividualReservations_Orders_null_fk
            references Orders,
    CustomerID    int not null
        constraint IndividualReservations_IndividualCustomers_null_fk
            references IndividualCustomers
)
go
```

Reservations

- ReservationID - ID rezerwacji
- ReservationStartDate - data rozpoczęcia rezerwacji
- ReservationEndDate - data zakończenia rezerwacji

Kod SQL

```
create table Reservations
(
    ReservationID   int      not null
        constraint Reservations_pk
            primary key,
    RealizationDate datetime not null
)
go
```

Widoki

ActiveClientsDiscounts

Widok pokazujący aktywne zniżki dla klientów

```
create view dbo.ActiveClientsDiscounts as
select IC.CustomerID, IC.FirstName, IC.LastName, discountType
from Discounts
join IndividualCustomers IC on Discounts.CustomerID = IC.CustomerID
where endDate >= getdate()
go
```

CompanyStatistics

Wyświetla informacje o firmach będących klientami - ID klienta, nazwę, email, telefon, adres, NIP, liczba złożonych zamówień oraz wartość zamówionych produktów

```
create view dbo.CompanyStatistics as
select Customers.CustomerID,
       CompanyName,
       Email,
       Phone,
       Address,
       NIP,
       count(O.OrderID) as OrdersAmmount,
       isnull(sum(Quantity * Price), 0) as ValueOrdered

from Customers
join CompanyCustomers CC on Customers.CustomerID = CC.CustomerID
left join Orders O on Customers.CustomerID = O.CustomerID
left join OrderDetails OD on O.OrderID = OD.OrderID
left join MenuDetails MD on OD.DishID = MD.DishID
group by Customers.CustomerID, CompanyName, Email, Phone, Address, NIP
go
```

ConfirmedReservations

Widok potwierdzonych rezerwacji które trzeba wykonać

```
CREATE view dbo.ConfirmedReservations as
select Reservations.ReservationID, ReservationStartDate, FirstName + ' ' + LastName as ForWho, TableID
from Reservations
join IndividualReservations IR on Reservations.ReservationID = IR.ReservationID
join IndividualCustomers IC on IR.CustomerID = IC.CustomerID
join ReservationsDetails RD on Reservations.ReservationID = RD.ReservationID
where ReservationStartDate >= getdate() and Status = 'Accepted'
union
select Reservations.ReservationID, ReservationStartDate, CompanyName as ForWho, TableID
from Reservations
join CompanyReservations CR on Reservations.ReservationID = CR.ReservationID
join CompanyCustomers CC on CC.CustomerID = CR.CustomerID
join ReservationsDetails RD on Reservations.ReservationID = RD.ReservationID
where ReservationStartDate >= getdate() and Status = 'Accepted'
go
```

CurrentMenu

Wyświetla listę dań aktualnie znajdujących się w menu wraz z cenami

```
create view dbo.CurrentMenu as
select DishName, Price from MenuDetails
join Dishes D on D.DishID = MenuDetails.DishID
join Menu M on M.MenuID = MenuDetails.MenuID
where getdate() between M.startDate and isnull(M.endDate, getdate()+1)
go
```

DiscountInfo

Wyświetla informacje o wykorzystanych i aktualnych zniżkach - na jakiego klienta, jaki rodzaj zniżki i data obowiązywania.

```
create view dbo.discountInfo as
select DiscountID, Discounts.CustomerID,
       FirstName + ' ' + LastName as Fullname,
       discountType, startDate, endDate
from Discounts
join IndividualCustomers IC on Discounts.CustomerID = IC.CustomerID
go
```

DiscountUsedMonthly

Wyświetla informacje dla ilu zamówień w danym miesiącu została wykorzystana zniżka.

```
create view dbo.DiscountUsedMonthly as
select year(orderdate) as Year, month(OrderDate) as Month, count(Discount) as AmountOfDiscounts
from Orders
group by year(orderdate), month(OrderDate)
go
```

DiscountUsedWeekly

Wyświetla informacje dla ilu zamówień w danym tygodniu roku została wykorzystana zniżka.

```
create view dbo.DiscountUsedWeekly as
select year(orderdate) as Year, datepart(week, orderdate) as Week, count(Discount) as AmountOfDiscounts
from Orders
group by year(orderdate), datepart(week, orderdate)
go
```

freeTables

Pokazuje aktualnie wolne stoliki

```
CREATE view dbo.freeTables as
select TableID as "Id wolnych stolikow"
 from Tables WHERE TableID not in
    (SELECT TableID FROM ReservationsDetails
     JOIN Reservations R2 on R2.ReservationID = ReservationsDetails.ReservationID
     WHERE R2.ReservationStartDate <= getDate()
       AND getDate() <= R2.ReservationEndDate
       AND R2.Status = 'Accepted')
 go
```

IndividualClientStatistics

Wyświetla informacje o klientach - jego ID, imię i nazwisko, email, telefon, liczba złożonych zamówień oraz wartość zamówionych produktów

```
create view dbo.IndividualClientsStatistics as
select Customers.CustomerID,
       FirstName + ' ' + LastName as Fullname,
       Email,
       Phone,
       count(OrderID) as AmmountOfOrders,
       isnull((select sum(Price * Quantity)
        from Customers C1
        join IndividualCustomers IC2 on C1.CustomerID = IC2.CustomerID
        join Orders on C1.CustomerID = Orders.CustomerID
        join OrderDetails OD on Orders.OrderID = OD.OrderID
        join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
        where C1.CustomerID = Customers.CustomerID), 0) as OrderedValue

from Customers
join IndividualCustomers IC on Customers.CustomerID = IC.CustomerID
left join Orders O on Customers.CustomerID = O.CustomerID
group by Customers.CustomerID, FirstName + ' ' + LastName, Email, Phone
go
```

MealsInfo

Wyświetla informacje jakie danie należy do której kategorii

```
create view dbo.MealsInfo as
select DishName,
       CategoryName
from Dishes
join Categories C on Dishes.CategoryID = C.CategoryID
go
```

OrdersInfo

Wyświetla historię złożonych zamówień - na kogo, data zamówienia, czy zostało zapłacone oraz wartość zamówienia

```
CREATE view dbo.OrdersInfo as
SELECT CompanyName as 'Name', O.OrderDate, O.Paid as 'IsPaid', (select sum(Quantity * Price)
        from Orders O2
        join OrderDetails OD on O2.OrderID = OD.OrderID
        join MenuDetails MD on MD.MenuID = OD.MenuID and MD.DishID = OD.DishID
        where O.OrderID = O2.OrderID) * (1-isnull(Discount,0)) as OrderedValue
    FROM CompanyCustomers
    JOIN Orders O on CompanyCustomers.CustomerID = O.CustomerID
UNION
SELECT FirstName + ' ' + LastName as 'Name', O.OrderDate, O.Paid as 'IsPaid',
       (select sum(Quantity * Price)
        from Orders O2
        join OrderDetails OD on O2.OrderID = OD.OrderID
        join MenuDetails MD on MD.MenuID = OD.MenuID and MD.DishID = OD.DishID
        where O.OrderID = O2.OrderID) * (1-isnull(Discount,0)) as OrderedValue
    FROM IndividualCustomers
    JOIN Orders O on IndividualCustomers.CustomerID = O.CustomerID
go
```

OrdersInvoicesForCompany

Widok faktur dla zamówień firmowych

```
create view dbo.ordersInvoicesForCompanies as
select OrderDate, CompanyName, DishName, Quantity, sum(Quantity * MD.Price) as valueOfDish,
       (select sum(Quantity * Price)
        from OrderDetails
        join MenuDetails MD2 on MD2.MenuID = OrderDetails.MenuID and MD2.DishID = OrderDetails.DishID
        join Orders O on OrderDetails.OrderID = O.OrderID
        where O.OrderID = Orders.OrderID
        ) as valueOfOrder
from Orders
join OrderDetails OD on Orders.OrderID = OD.OrderID
join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
join MenuDetails M on M.MenuID = OD.MenuID and M.DishID = OD.DishID
join Dishes D on D.DishID = M.DishID
join Customers C on C.CustomerID = Orders.CustomerID
join CompanyCustomers CC on C.CustomerID = CC.CustomerID
group by Orders.OrderID, OrderDate, CompanyName, DishName, Quantity
go
```

OrdersInvoicesForIndividualCustomers

Widok faktur dla zamówień osób prywatnych

```
create view dbo.OrdersInvoicesForIndividualCustomers as
select OrderDate, FirstName + ' ' + LastName as 'Firstname and LastName' ,
       DishName, Quantity, sum(Quantity * MD.Price) as valueOfDish,
       (select sum(OrderDetails.Quantity * MD2.Price)
        from OrderDetails
        join MenuDetails MD2 on MD2.MenuID = OrderDetails.MenuID and MD2.DishID = OrderDetails.DishID
        join Orders O on OrderDetails.OrderID = O.OrderID
        where O.OrderID = Orders.OrderID
        ) as valueOfOrderWithoutDiscount,
        (select sum(OD2.Quantity * MD3.Price * convert(money,(1 - (isnull(O2.Discount, 0)))))
        from OrderDetails OD2
        join MenuDetails MD3 on MD3.MenuID = OD2.MenuID and MD3.DishID = OD2.DishID
        join Orders O2 on OD2.OrderID = O2.OrderID
        where O2.OrderID = Orders.OrderID
        ) as valueOfOrderWithDiscount
from Orders
join OrderDetails OD on Orders.OrderID = OD.OrderID
join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
join MenuDetails M on M.MenuID = OD.MenuID and M.DishID = OD.DishID
join Dishes D on D.DishID = M.DishID
join Customers C on C.CustomerID = Orders.CustomerID
join IndividualCustomers IC on C.CustomerID = IC.CustomerID
group by Orders.OrderID, OrderDate, DishName, Quantity, FirstName, LastName
go
```

OrdersStatisticsMonthly

Wyświetla ilość złożonych zamówień z podziałem na lata i miesiące

```
CREATE view dbo.OrderStatisticsMonthly as
select year(orderdate) as Year, month(orderdate) as Month, count(OrderID) as Amount_Of_Orders
from Orders
group by year(orderdate), month(orderdate)
go
```

OrdersStatisticsWeekly

Wyświetla ilość złożonych zamówień z podziałem na lata i tygodnie

```
create view dbo.OrderStatisticsWeekly as
select year(orderdate) as Year,
       datepart(week, orderdate) as Week,
       count(OrderID) as Ammount_of_orders
from Orders
group by year(orderdate), datepart(week, orderdate)
go
```

OrdersToday

Widok dzisiejszych zamówień

```
CREATE view dbo.OrdersToday as
select *
from Orders
where day(OrderDate) = day(getdate()) and month(OrderDate) = month(getdate()) and year(OrderDate) = year(getdate())
go
```

pendingReservations

pokazuje które rezerwacje nie zostały jeszcze zaakceptowane

```
create view dbo.PendingReservations as
    select ReservationID, ReservationStartDate, ReservationEndDate FROM Reservations
    where Status = 'Pending'
go
```

SeaFoodDeliveries

Pokazuje wszystkie dostawy owoców morza

```
create view dbo.seaFoodDeliveries as
select DishName, LastDelivery
 from Categories
 join Dishes D on Categories.CategoryID = D.CategoryID
where CategoryName = 'Sea food'
 go
```

SoldMealsInfoMonthly

Wyświetla ilość sprzedanych potraw z podziałem na lata i miesiące

```
create view dbo.soldMealsInfoMonthly as
select year(OrderDate) as year,
       month(OrderDate) as month,
       Dishname,
       isnull(count(OD.DishID), 0) as times_sold
from Orders
left join OrderDetails OD on Orders.OrderID = OD.OrderID
left join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
left join Dishes D on MD.DishID = D.DishID
group by year(OrderDate), month(OrderDate), DishName
go
```

SoldMealsInfoWeekly

Wyświetla ilość sprzedanych potraw z podziałem na lata i tygodnie

```
create view dbo.soldMealsInfoWeekly as
select year(OrderDate) as year,
       datepart(week,OrderDate) as week,
       Dishname,
       isnull(count(OD.DishID), 0) as times_sold
from Orders
left join OrderDetails OD on Orders.OrderID = OD.OrderID
left join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
left join Dishes D on MD.DishID = D.DishID
group by year(OrderDate), datepart(week,OrderDate), DishName
go
```

TakeawayOrders

Widok zamówień na wynos do zrealizowania

```
create view dbo.takeAwayOrders as
select Orders.OrderID, Orders.CustomerID, FirstName + IC.LastName as "Imie lub Nazwa", phone, Email
from Orders
join Customers C on Orders.CustomerID = C.CustomerID
join IndividualCustomers IC on C.CustomerID = IC.CustomerID
where ForTakeout = 1 and OrderDate is null
union
select Orders.OrderID, Orders.CustomerID, CompanyName, phone, Email
from Orders
join Customers C on Orders.CustomerID = C.CustomerID
join CompanyCustomers CC on C.CustomerID = CC.CustomerID
where ForTakeout = 1 and OrderDate is null
go
```

UsedTablesMonthly

Wyświetla ilość rezerwacji z podziałem na lata i miesiące

```
create view dbo.usedTablesMonthly as
select year(RealizationDate) as year,
       month(RealizationDate) as month,
       count(TableID) as ammount_of_reserved_tables
from Reservations
join ReservationsDetails RD on Reservations.ReservationID = RD.ReservationID
group by year(RealizationDate), month(RealizationDate)
go
```

UsedTablesWeekly

Wyświetla ilość rezerwacji z podziałem na lata i tygodnie

```
create view dbo.usedTablesWeekly as
select year(RealizationDate) as year,
       datepart(week,RealizationDate) as week,
       count(TableID) as ammount_of_reserved_tables
from Reservations
join ReservationsDetails RD on Reservations.ReservationID = RD.ReservationID
group by year(RealizationDate),datepart(week,RealizationDate)
go
```

Pokazuje wszystkie

WZWKshow

Pokazuje aktualne współczynniki WZ i WK

```
CREATE view dbo.WZWKshow as
select WZ, WK
from ReservationVariables
go
```

Procedury

AcceptReservation

```
CREATE procedure acceptReservation
@ReservationID int,
@TableID int
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Reservations WHERE ReservationID=@ReservationID)
        begin;
            throw 52000, N'Invalid Reservation ID', 1
        end

         if ((SELECT TOP 1 Status FROM Reservations WHERE ReservationID=@ReservationID) = 'Accepted')
         begin;
             throw 52000, N'Reservation is already accepted', 1
         end

        if not exists(SELECT * FROM Tables WHERE TableID=@TableID)
        begin;
            throw 52000, N'Invalid Table ID', 1
        end

        update Reservations set Status = 'Accepted' WHERE ReservationID = @ReservationID
        update ReservationsDetails set TableID=@TableID WHERE ReservationID = @ReservationID

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding customer: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
```

addCategory

Przyjmuje: Nazwę kategorii
Opis: Dodaje nową kategorie do tabeli Categories

```
create procedure addCategory
@CategoryName varchar(255)
AS
BEGIN
    set nocount ON
    begin try
        if exists(SELECT * FROM Categories WHERE @CategoryName = CategoryName)
            begin;
                throw 52000, N'Categry alredy exists', 1
            end
        declare @CategoryID int

        SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
        FROM Categories

        insert into Categories(CategoryID, CategoryName, LastDelivery)
        values(@CategoryID, @CategoryName, getdate());
    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding category: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addCompanyCustomer

Przyjmuje: email, telefon, nazwę firmy(opcjonalne), adres(opcjonalne), nip(opcjonalne)
Opis: Procedura dodaje nowego klienta firmowego.

```
CREATE procedure addCompanyCustomer
@Email varchar(255),
@Phone varchar(16),
@CompanyName varchar(255),
@Address varchar(255),
@NIP varchar(255)
AS
BEGIN
    set nocount ON
    begin try

        if exists(SELECT * FROM Customers WHERE Email = @Email or Phone = @Phone)
        begin;
            throw 52000, N'Email/Phone is alredy taken', 1
        end

        declare @CustomerID int
        SELECT @CustomerID = ISNULL(MAX(CustomerID), 0) + 1 FROM Customers

        insert into Customers(CustomerID, Email, Phone) values(@CustomerID, @Email, @Phone)

        insert into CompanyCustomers(customerid, companyname, address, nip)
        values(@CustomerID, @CompanyName, @Address, @NIP)

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding company customer: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addIndividualCustomer

Przyjmuje: email, telefon, Imię, Nazwisko, (Opcjonalnie) ID firmy do której należy klient
Opis: Dodaje klienta indywidualnego do bazy danych

```
CREATE procedure addIndividualCustomer
@Email varchar(255),
@Phone varchar(16),
@FirstName varchar(255),
@LastName varchar(255),
@CompanyID int
AS
BEGIN
    set nocount ON
    begin try

        if exists(SELECT * FROM Customers WHERE Email = @Email or Phone = @Phone)
        begin;
            throw 52000, N'Email/Phone is alredy taken', 1
        end

        declare @CustomerID int
        SELECT @CustomerID = ISNULL(MAX(CustomerID), 0) + 1 FROM Customers

        insert into Customers(CustomerID, Email, Phone) values(@CustomerID, @Email, @Phone)

        insert into IndividualCustomers(customerid, companyid, firstname, lastname)
        values(@CustomerID, @CompanyID, @FirstName, @LastName)

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding customer: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addDiscount

Procedura dodająca zniżkę dla danego klienta

```
CREATE procedure addDiscount
@CustomerID int,
@startDate datetime,
@endDate datetime,
@type varchar(255)
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Customers WHERE CustomerID=@CustomerID)
        begin;
            throw 52000, N'Invalid customer ID', 1
        end

        if not (@type = 'Stala' OR @type = 'Jednorazowa')
        begin;
             throw 52000, N'Invalid discount type', 1
        end

        if @CustomerID in (SELECT CustomerID FROM CompanyCustomers)
        begin;
             throw 52000, N'Customer cannot be an Company Customer', 1
        end

        declare @DiscountID int
        SELECT @DiscountID = ISNULL(MAX(DiscountID), 0)+1 FROM Discounts

        insert into Discounts(CustomerID, DiscountID, startDate, endDate, discountType)
             values(@CustomerID, @DiscountID, @startDate, @endDate, @type)

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding discount: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addDish

Przyjmuje: Nazwę nowego dania, kategorie
Opis: Procedura dodaje nowe danie do zbioru wszystkich dań

```
create procedure addDish
@DishName varchar(255),
@CategoryName varchar(255)
AS
BEGIN
    set nocount ON
    begin try
        if not exists(SELECT * FROM Categories WHERE @CategoryName = CategoryName)
            begin;
                throw 52000, N'Invalid category name', 1
            end

        if exists(SELECT * FROM Dishes WHERE @DishName = DishName)
            begin;
                throw 52000, N'Dish alredy exists', 1
            end

        declare @CategoryID int
        SELECT @CategoryID = CategoryID FROM Categories WHERE CategoryName = @CategoryName

        declare @DishID int
        SELECT @DishID = ISNULL(MAX(DishID), 0) + 1 FROM Dishes

        insert into Dishes(DishID, CategoryID, DishName) values(@DishID, @CategoryID, @DishName);
    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding dish: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addDishToMenu

Przyjmuje: ID Dania, ID Menu, Cenę Dania
Opis: Dodaje danie o sprecyzowanym ID do menu o sprecyzowanym ID z Ceną która zostanie podana.

```
CREATE procedure addDishToMenu
@DishID int,
@MenuID int,
@Price real
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Menu WHERE MenuID = @MenuID)
            begin;
                throw 52000, N'Menu with given ID doesnt exist', 1
            end

        if not exists(SELECT * FROM Dishes WHERE DishID = @DishID)
            begin;
                throw 52000, N'Dish with given ID doesnt exist', 1
            end

        insert into MenuDetails(menuid, dishid, price) values(@MenuID, @DishID, @Price)
    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding dish to menu: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addDishToOrder

Przyjmuje: ID zamówienia, ID Dania, Ilość danego dania
Opis: Procedura dodaje do zamówienia Danie o sprecyzowanym ID z odpowiednią ilością. Menu ID jest znajdowane poprzez znalezienie menu które było używane podczas gdy zamówienie było robione.

```
CREATE procedure addDishToOrder
@OrderID int,
@DishID int,
@Quantity int = 1
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Orders WHERE OrderID=@OrderID)
        begin;
            throw 52000, N'Invalid order ID', 1
        end

        if not exists(SELECT * FROM Dishes WHERE DishID=@DishID)
        begin;
            throw 52000, N'Invalid dish ID', 1
        end

        declare @Date datetime
        SELECT @Date = (SELECT OrderDate FROM Orders WHERE OrderID=@OrderID)

        if @DishID in (SELECT DishID FROM Dishes
                                    JOIN Categories C on C.CategoryID = Dishes.CategoryID
                                    WHERE C.CategoryID = 1)
        begin;
            if (DATEPART(WEEKDAY, @Date) != 5
                        AND DATEPART(WEEKDAY, @Date) != 6
                        AND DATEPART(WEEKDAY, @Date) != 7)
            begin;
                throw 52000, N'Cannot add seafood dish to this order', 1
            end
        end
```

```
    -- declare menu and check if order date is correct (or if menu exists)
    declare @MenuID int
    SELECT @MenuID = (
        SELECT TOP 1 MenuID FROM Menu WHERE
            startDate <= @Date
            AND (endDate >= @Date OR endDate IS NULL)
        ORDER BY startDate)

    if @MenuID is null
    begin;
        throw 52000, N'Invalid Order date', 1
    end

    if @DishID not in (SELECT DishID FROM MenuDetails WHERE MenuID = @MenuID)
    begin;
        declare @msg_1 nvarchar(2048) = N'Cannot add dish with ID ' + CONVERT(nvarchar, @DishID) +
                                        ' to order because it is not in menu';
        throw 52000, @msg_1, 1;
    end

    insert into OrderDetails(OrderID, DishID, MenuID, Quantity) values(@OrderID, @DishID, @MenuID, @Quantity)

end try
begin catch
    declare @msg_2 nvarchar(2048) = N'Error while adding dish to order: ' + ERROR_MESSAGE();
    throw 52000, @msg_2, 1;
end catch
end
go
```

addEmployee

Przyjmuje: Imię, Nazwisko, ID Szefa, telefon, email, zdjęcie, dodatkowe informacje
Opis: Dodaje pracownika do tabeli Employees

```
CREATE procedure addEmployee
@FirstName varchar(32),
@LastName varchar(32),
@Auth int = null,
@Phone varchar(32),
@Email varchar(64),
@Photo varbinary(max) = null,
@AdditionalNotes varchar(255) = 'Pracownik'
AS
BEGIN
    set nocount ON
    begin try

        if exists(SELECT * FROM Employees WHERE Email = @Email OR Phone = @Phone)
            begin;
                throw 52000, N'Employee with such phone/email exists', 1
            end

        declare @EmployeeID int
        SELECT @EmployeeID = ISNULL(MAX(EmployeeID), 0) + 1 FROM Employees

        insert into Employees(EmployeeID, Firstname, Lastname, Auth, Phone, Email, Photo, AdditionalNotes)
            values (@EmployeeID, @FirstName, @LastName, @Auth, @Phone, @Email, @Photo, @AdditionalNotes)
    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding employee: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addMenu

Przyjmuje: start date, end date, tabela z detalami menu
Opis: Dodaje nowe menu do bazy danych

```
CREATE procedure addMenu
@StartDate datetime,
@EndDate datetime = null,
@MenuDetails MenuDetailsType READONLY
AS
BEGIN
    set nocount ON
    begin try

        declare @MenuID int
        SELECT @MenuID = ISNULL(MAX(MenuID), 0) + 1 FROM Menu


        DECLARE @RowCountcheck INT = (SELECT COUNT(*) FROM @MenuDetails)
        declare @DishIDcheck int

        WHILE @RowCountcheck > 0 BEGIN
            SELECT @DishIDcheck = DishID
            FROM @MenuDetails
            ORDER BY DishID DESC
            OFFSET @RowCountcheck - 1 ROWS FETCH NEXT 1 ROWS ONLY

            if not exists(SELECT * FROM Dishes WHERE DishID = @DishIDcheck)
            begin
                declare @msg_1 nvarchar(2048) = N'Dish with ID ' + @DishIDcheck+ 'doesnt exist';
                throw 52000, @msg_1, 1;
            end

            SET @RowCountcheck -= 1;
        END


        insert into Menu(MenuID, startDate, endDate) values(@MenuID, @StartDate, @EndDate)
```

```
        DECLARE @RowCount INT = (SELECT COUNT(*) FROM @MenuDetails)
        declare @DishID int
        declare @Price money

        WHILE @RowCount > 0 BEGIN
            SELECT @DishID = DishID, @Price = Price
            FROM @MenuDetails
            ORDER BY DishID DESC
            OFFSET @RowCount - 1 ROWS FETCH NEXT 1 ROWS ONLY

            Exec dbo.addDishToMenu @DishID = @DishID, @MenuID = @MenuID, @Price = @Price

            SET @RowCount -= 1;
        END

    end try
    begin catch
        declare @msg_2 nvarchar(2048) = N'Error while adding menu: ' + ERROR_MESSAGE();
        throw 52000, @msg_2, 1;
    end catch
end
go
```

addOrder

Przyjmuje: ID klienta, Date zamówienia, Zniżke, ID Pracownika obsługującego zamówienie, bit mówiący czy zamówienie jest na wynos, bit mówiący czy zamówienie zostało z góry opłacone, Tabele typu OrderDetailsType posiadającą szczegóły zamówienia
Opis: Dodaje nowe zamówienie oraz obsługuje dodawanie zniżek

```
CREATE procedure addOrder @CustomerID int,
                          @OrderDate datetime,
                          @Discount real,
                          @EmployeeID int,
                          @ForTakeout bit,
                          @Paid bit,
                          @OrderDetails OrderDetailsType READONLY
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Customers WHERE CustomerID = @CustomerID)
            begin
                throw 52000, N'Invalid customer ID', 1
            end

        if not exists(SELECT * FROM Employees WHERE EmployeeID = @EmployeeID)
            begin
                throw 52000, N'Invalid employee ID', 1
            end

--      This part checks if @OrderDetails table is correct
        DECLARE @RowCountCheck INT = (SELECT COUNT(*) FROM @OrderDetails)
        declare @DishIDcheck int
        declare @Quantitycheck int

        WHILE @RowCountCheck > 0 BEGIN
            SELECT @DishIDcheck = DishID, @Quantitycheck = Quantity
            FROM @OrderDetails
            ORDER BY DishID DESC
            OFFSET @RowCountCheck - 1 ROWS FETCH NEXT 1 ROWS ONLY

            -- check for seafood products
            if @DishIDcheck in (SELECT DishID
                                FROM Dishes
                                        JOIN Categories C on C.CategoryID = Dishes.CategoryID
                                WHERE C.CategoryID = 1)
                begin
                    declare @OrderSubDate DATE =
                        DATEFROMPARTS(YEAR(@OrderDate), MONTH(@OrderDate), DAY(@OrderDate))
                    declare @PreviousMonday DATETIME =
                        DATEADD(DD, -(DATEPART(WEEKDAY, @OrderSubDate)+5)%7, @OrderSubDate)
```

```
            if ((DATEPART(WEEKDAY, @OrderDate) != 5
                    AND DATEPART(WEEKDAY, @OrderDate) != 6
                    AND DATEPART(WEEKDAY, @OrderDate) != 7)
                    OR GETDATE() > @PreviousMonday)
                begin
                    throw 52000, N'Cannot add seafood dish to this order today', 1
                end
        end

    -- declare menu and check if order date is correct (or if menu exists)
    declare @MenuID int
    SELECT @MenuID = (SELECT TOP 1 MenuID
                      FROM Menu
                      WHERE startDate <= @OrderDate
                        AND (endDate >= @OrderDate OR endDate IS NULL)
                      ORDER BY startDate)

    if @MenuID is null
        begin
            throw 52000, N'Invalid Order date', 1
        end

    if @DishIDcheck not in (SELECT DishID FROM MenuDetails WHERE MenuID = @MenuID)
        begin
            declare @msg_1 nvarchar(2048) = N'Cannot add dish with ID ' +
                                            CONVERT(nvarchar, @DishIDcheck) +
                                            ' to order because it is not in menu';
            throw 52000, @msg_1, 1;
        end

    SET @RowCountCheck -= 1
END

-- exec discout check just to throw new errors now not after adding additional data
EXEC dbo.checkNewDiscountAfterOrder @CustomerID = @CustomerID

--          CHECKS OVER -> INSERTING INTO TABLES

declare @OrderID int
SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1 FROM Orders

insert into Orders(OrderID, CustomerID, OrderDate, Discount, EmployeeID, ForTakeout, Paid)
values (@OrderID, @CustomerID, @OrderDate, @Discount, @EmployeeID, @ForTakeout, @Paid)

--  This part adds order details to the order

DECLARE @RowCount INT = (SELECT COUNT(*) FROM @OrderDetails)
declare @DishID int
declare @Quantity int
```

```
    WHILE @RowCount > 0 BEGIN
        SELECT @DishID = DishID, @Quantity = Quantity
        FROM @OrderDetails
        ORDER BY DishID DESC
        OFFSET @RowCount - 1 ROWS FETCH NEXT 1 ROWS ONLY

        Exec dbo.addDishToOrder @OrderID = @OrderID, @DishID = @DishID, @Quantity = @Quantity
        SET @RowCount -= 1;
    END

--  This part checks, only if the customer is Individual, if an discount can be added

    declare @DiscountCheck int
    declare @DiscountID int

    if @CustomerID in (SELECT CustomerID FROM IndividualCustomers)
    begin

        EXEC @DiscountCheck=dbo.checkNewDiscountAfterOrder @CustomerID = @CustomerID

        if (@DiscountCheck = 1)
        begin
            -- add permanent discount

            SELECT @DiscountID=ISNULL(MAX(DiscountID), 0)+1 FROM Discounts

            insert into Discounts(CustomerID, DiscountID, startDate, endDate, discountType)
                values (@CustomerID, @DiscountID, @OrderDate, null, 'Stala')
        end

        if (@DiscountCheck = 2)
        begin
            -- add temporary discount

            SELECT @DiscountID=ISNULL(MAX(DiscountID), 0)+1 FROM Discounts

            insert into Discounts(CustomerID, DiscountID, startDate, endDate, discountType)
                values (@CustomerID, @DiscountID, @OrderDate,
                        DATEADD(day, (SELECT TOP 1 D1 FROM DiscountVariables),
                            @OrderDate), 'Jednorazowa')
        end

        if (@DiscountCheck = 3)
        begin
            -- add both discounts

            SELECT @DiscountID=ISNULL(MAX(DiscountID), 0)+1 FROM Discounts

            insert into Discounts(CustomerID, DiscountID, startDate, endDate, discountType)
                values (@CustomerID, @DiscountID, @OrderDate, null, 'Stala')
```

```
            insert into Discounts(CustomerID, DiscountID, startDate, endDate, discountType)
                values (@CustomerID, @DiscountID, @OrderDate,
                        DATEADD(day, (SELECT TOP 1 D1 FROM DiscountVariables),
                            @OrderDate), 'Jednorazowa')
        end
    end

    return @OrderID

end try
begin catch
    declare @msg nvarchar(2048) = N'Error while adding Order: ' + ERROR_MESSAGE();
    throw 52000, @msg, 1;
end catch
end
go
```

addPersonToReservation

Przyjmuje: ID rezerwacji, Imię i nazwisko
Opis: Dodaje osobę do istniejącej rezerwacji (musi być to rezerwacja firmowa)

```
CREATE procedure addPersonToReservation
@ReservationID int,
@Name varchar(128)
AS
BEGIN
   set nocount ON
   begin try

       if not exists(SELECT * FROM Reservations WHERE ReservationID=@ReservationID)
       begin;
           throw 52000, N'Invalid Reservation ID', 1
       end

       if @ReservationID not in (SELECT ReservationID FROM CompanyReservations)
       begin;
           throw 5200, N'Not a Company Reservation', 1
       end

       declare @TableID int
       SELECT @TableID = T.TableID FROM Reservations
           JOIN ReservationsDetails RD on Reservations.ReservationID = RD.ReservationID
           JOIN Tables T on T.TableID = RD.TableID
           WHERE RD.ReservationID = @ReservationID

       insert into ReservationsDetails(ReservationID, TableID, Name) values(@ReservationID, @TableID, @Name)

   end try
   begin catch
       declare @msg nvarchar(2048) = N'Error while adding person to reservation: ' + ERROR_MESSAGE();
       throw 52000, @msg, 1;
   end catch
end
go
```

addCompanyReservation

Przyjmuje: startDate, endDate, ID stolika, ID firmy, Lista imion typu NamesListType
Opis: dodaje rezerwacje firmową dla osób sprecyzowanych w liście imion

```
CREATE procedure addCompanyReservation
@StartDate datetime,
@EndDate datetime,
@tableID int,
@CustomerID int,
@NamesList CompanyReservationNamesType READONLY
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Tables WHERE TableID = @tableID)
            begin
                throw 52000, N'Invalid Table ID', 1
            end

        if @CustomerID not in (SELECT CustomerID FROM CompanyCustomers)
        begin
            throw 52000, N'CustomerID is not an CompanyID', 1
        end

        declare @ReservationID int
        SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 FROM Reservations

        insert into Reservations(ReservationID, ReservationStartDate, ReservationEndDate, Status)
        values (@ReservationID, @StartDate, @EndDate, 'Pending')

        insert into CompanyReservations(ReservationID, CustomerID) values (@ReservationID, @CustomerID)

        DECLARE @RowCount INT = (SELECT COUNT(*) FROM @NamesList)
        declare @Name varchar(255)

        WHILE @RowCount > 0 BEGIN
            SELECT @Name = Name
            FROM @NamesList
            ORDER BY Name DESC
            OFFSET @RowCount - 1 ROWS FETCH NEXT 1 ROWS ONLY

            insert into ReservationsDetails(ReservationID, TableID, Name) values (@ReservationID, @tableID, @Name)

            SET @RowCount -= 1;
        END

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding reservation: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addIndividualReservation

Przyjmuje: startDate, endDate, ID stolika, ID klienta, Dane Do zamówienia: Zniżka, ID pracownika, bit decydujący czy zamówienie zostało z góry opłacone, Tabele z detalami
Opis: Dodaje rezerwacje indywidualną oraz powiązane z nią zamówienie

```
CREATE procedure addIndividualReservation
@StartDate datetime,
@EndDate datetime,
@tableID int,
@CustomerID int,
@Discount real,
@EmployeeID int,
@Paid bit,
@OrderDetails OrderDetailsType READONLY
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Tables WHERE TableID = @tableID)
            begin
                throw 52000, N'Invalid Table ID', 1
            end

        if @CustomerID not in (SELECT CustomerID FROM IndividualCustomers)
            begin
                throw 52000, N'CustomerID is not an IndividualID', 1
            end

        if dbo.canClientMakeReservation( @input: @CustomerID) = 0
            begin
                throw 52000, N'Cannot make reservation, insufficient number of orders', 1
            end

        declare @ReservationID int
        SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 FROM Reservations

        declare @OrderID int
        EXEC @OrderID=dbo.addOrder @CustomerID = @CustomerID,
            @OrderDate = @StartDate,
            @Discount = @Discount,
            @EmployeeID = @EmployeeID,
            @ForTakeout = 0,
            @Paid = @Paid,
            @OrderDetails = @OrderDetails

        insert into Reservations(ReservationID, ReservationStartDate, ReservationEndDate, Status)
        values (@ReservationID, @StartDate, @EndDate, 'Pending')
```

```
        insert into ReservationsDetails(ReservationID, TableID, Name)
            values (@ReservationID, @tableID,
                (SELECT Firstname + ' ' + LastName FROM IndividualCustomers WHERE CustomerID = @CustomerID))

        insert into IndividualReservations(ReservationID, OrderID, CustomerID)
            values (@ReservationID, @OrderID, @CustomerID)

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding reservation: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

addTable

Przyjmuje: Ilość miejsc przy stoliku
Opis: Dodaje nowy stolik do tabeli Tables

```
CREATE procedure addTable
@NoOfSeats int
AS
BEGIN
    set nocount ON
    begin try

        declare @TableID int
        select @TableID = isnull(MAX(TableID), 0) + 1 from Tables

        insert into Tables(TableID, NoOfSeats)
        Values (@TableID, @NoOfSeats);

    END try
    BEGIN CATCH
        declare @msg nvarchar(2048)=N'Error while adding table: ' + ERROR_MESSAGE();
        THROW 5200, @msg, 1
    end catch
end
go
```

ModifyOrderPaidStatus

Przyjmuje: ID zamówienia
Opis: Zmienia bit ‘Paid’ w zamówienio o danym ID na 1

```
CREATE procedure modifyOrderPaidStatus
@OrderID int
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Orders WHERE OrderID=@OrderID)
        begin;
            throw 52000, N'Invalid order ID', 1
        end

        update Orders set Paid = 1 WHERE OrderID = @OrderID

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding customer: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

checkNewDiscountsAfterOrder

Przyjmuje: ID klienta
Opis: Procedura wewnętrzna zwracająca inta według którego procedura addOrder decyduje jakie zniżki dodać klientowi indywidualnemu po złożonym zamówieniu

```
CREATE procedure checkNewDiscountAfterOrder
@CustomerID int
as
begin
    set nocount ON
    begin try
        -- 0 no discount
        -- 1 only permanent discount
        -- 2 only temporary discount
        -- 3 both discounts

        if not exists(SELECT CustomerID FROM Customers WHERE CustomerID=@CustomerID)
        begin
            throw 52000, N'Invalid CustomerID', 1
        end

        if @CustomerID not in (SELECT CustomerID FROM IndividualCustomers)
        begin
            throw 52000, N'Cannot check Discounts for non Individual Customers', 1
        end

        declare @DiscountType int = 0

        -- check for permanent discount
        if ((SELECT COUNT(OrderID) FROM Orders WHERE
                    CustomerID=@CustomerID
                    AND dbo.GetOrderValue( @input: OrderID)
                >= (SELECT TOP 1 K1 FROM DiscountVariables))
            >= (SELECT TOP 1 Z1 FROM DiscountVariables)
            AND not EXISTS(SELECT * FROM Discounts
                WHERE CustomerID=@CustomerID
                AND discountType='Stala'))
        begin
            SELECT @DiscountType=@DiscountType+1
        end

        -- check for temporary discount
        -- first declare the last temporary discount date
        declare @LastDiscount DATETIME
        SELECT TOP 1 @LastDiscount=endDate FROM Discounts
            WHERE discountType='Jednorazowa'
            AND CustomerID=@CustomerID
            ORDER BY endDate DESC
```

```
        if ((SELECT SUM(dbo.GetOrderValue( @input: OrderID)) FROM Orders
                WHERE OrderDate >= ISNULL(@LastDiscount, cast(-53690 as datetime))
                AND CustomerID=@CustomerID)
            >= (SELECT K2 FROM DiscountVariables))
        begin
            SELECT @DiscountType=@DiscountType+1
        end

        return @DiscountType

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error in checkNewDiscountAfterOrder procedure: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

ModifyReservationStatus

Przyjmuje: ID rezerwacji
Opis: Zmienia status rezerwacji z ‘Pending’ na ‘Accepted’

```
CREATE procedure modifyReservationStatus
@ReservationID int
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Reservations WHERE ReservationID=@ReservationID)
        begin;
            throw 52000, N'Invalid Reservation ID', 1
        end

        update Reservations set Status = 'Accepted' WHERE ReservationID = @ReservationID

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while adding customer: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

DeclineReservation

```
CREATE procedure declineReservation
@ReservationID int
AS
BEGIN
    set nocount ON
    begin try

        if not exists(SELECT * FROM Reservations WHERE ReservationID=@ReservationID)
        begin;
            throw 52000, N'Invalid Reservation ID', 1
        end

        if ((SELECT TOP 1 Status FROM Reservations WHERE ReservationID=@ReservationID) = 'Accepted')
         begin;
            throw 52000, N'Reservation is already accepted', 1
         end

        delete Reservations WHERE ReservationID=@ReservationID

    end try
    begin catch
        declare @msg nvarchar(2048) = N'Error while declineing reservation: ' + ERROR_MESSAGE();
        throw 52000, @msg, 1;
    end catch
end
go
```

Funkcje

AvailableCustomerDiscounts

```
CREATE function avilableCustomerDiscounts(@CustomerID int)
returns varchar(24)
as
    begin
        declare @ind bit
        SELECT @ind = 0
        declare @return varchar(24) = 'Brak zniżki'

        if exists (SELECT * FROM Discounts WHERE CustomerID=@CustomerID AND discountType='Stala')
        begin
            SELECT @ind=@ind+1
        end
        if exists (SELECT * FROM Discounts WHERE CustomerID=@CustomerID AND discountType='Jednorazowa')
        begin
            SELECT @ind=@ind+2
        end

        if (@ind=1)
        begin
            SELECT @return='Stała'
        end
        if (@ind=2)
        begin
            SELECT @return='Jednorazowa'
        end
        if (@ind=3)
        begin
            SELECT @return='Stała i Jednorazowa'
        end
        return @return
    end
```

CanClientMakeReservation

```
CREATE function canClientMakeReservation(@input int)
    returns bit
as
begin

    declare @return bit
    SELECT @return = 0

    IF (isnull((select count(*)
        from Orders
            where CustomerID = @input
            and dbo.getOrderValue( @input: OrderID) >
                (select top 1 WZ from ReservationVariables order by ReservationVariables.VariablesID)),0)
            > (SELECT TOP 1 WK FROM ReservationVariables ORDER BY ReservationVariables.VariablesID))
        begin
            SELECT @return = 1
        end

    return @return
end
go
```

getAvgMenuPrice

Przyjmuje MenuID jako parametr i dla niego wyświetla jego średnią cenę jego produktów

```
create function GetAvgMenuPrice(@MenuID int)
    returns money
as
begin
    return (select avg(price) from MenuDetails where MenuID = @MenuID)
end
go
```

GetClientsWhoOrderedForMoreValueThanGivenValue

Przyjmuje liczbę całkowitą jako parametr i wyświetli wszystkich klientów którzy złożyli zamówienie za co najmniej podaną kwotę

```
CREATE function getClientsWhoOrderedForMoreValueThanGivenValue (@input int)
returns table as
    return select distinct CompanyName, OrderedValue
           from CompanyStatistics where CompanyStatistics.OrderedValue >= @input
    union select distinct IndividualClientsStatistics.Fullname, OrderedValue
          from IndividualClientsStatistics where IndividualClientsStatistics.OrderedValue >= @input
    order by OrderedValue DESC
go
```

GetClientsWhoOrderedMoreThanGivenValue

Funkcja zwraca klientów którzy wykonali więcej zamówień niż podany parametr całkowitoliczbowy

```
create function getClientsWhoOrderedMoreThanGivenValue(@input int)
returns table as
    return select Orders.CustomerID, CompanyName as customer
    from Orders
    join Customers C on C.CustomerID = Orders.CustomerID
    join CompanyCustomers CC on C.CustomerID = CC.CustomerID
    group by Orders.CustomerID, CompanyName
    having sum(OrderID) >= @input
    union
    select O.CustomerID, FirstName + ' ' + LastName as customer
    from Orders O
    join Customers C2 on C2.CustomerID = O.CustomerID
    join IndividualCustomers IC on C2.CustomerID = IC.CustomerID
    group by O.CustomerID, FirstName + ' ' + LastName
    having sum(OrderID) >= @input
go
```

GetEmployeesOfGivenCompany

Zwraca dla przekazanego CompanyID liste jego pracowników

```
create function getEmployeesOfGivenCompany(@input int)
returns table as
    return select FirstName, LastName
    from IndividualCustomers
    where CompanyID = @input
go
```

GetMealsSoldWithMinimalTimes

Zwraca pozycje dań które zostały zamówione co najmniej podaną wartość razy.

```
create function getMealsSoldWithMinimalTimes(@input int)
    returns table as
        return select DishName, sum(Quantity) as times_sold
        from OrderDetails
        join MenuDetails MD on OrderDetails.MenuID = MD.MenuID and OrderDetails.DishID = MD.DishID
        join Dishes D on MD.DishID = D.DishID
        group by DishName
        having sum(Quantity) > @input
go
```

GetMenuIDByDate

Funkcja zwraca menu które było aktualne dla przekazanego parametru date

```
create function getMenuIDByDate(@Date datetime)
    returns int
as
begin
    return (
        SELECT TOP 1 MenuID FROM Menu WHERE
            startDate <= @Date
            AND (endDate >= @Date OR endDate IS NULL)
        ORDER BY startDate
    )
end
go
```

GetMenuItemsByMenuID

Funkcja zwraca Dania dla przekazanego menu

```
CREATE function GetMenuItemsByMenuID(@input int)
    returns table as
        return select Menu.MenuID, startDate, endDate, D.DishName, Price
        from Menu
        join MenuDetails MD on Menu.MenuID = MD.MenuID
        join Dishes D on D.DishID = MD.DishID
        where (Menu.MenuID = @input)
go
```

GetOrdersAboveGivenValue

Funkcja zwraca wszystkie zamowienia które były większe od przekazanej wartości całkowitoliczbowej

```
create function GetOrdersAboveGivenValue(@input int)
    returns table as
        return
        select Orders.OrderID, CustomerID, sum(price * Quantity) as orderValue
        from Orders
        join OrderDetails OD on Orders.OrderID = OD.OrderID
        join MenuDetails MD on OD.MenuID = MD.MenuID and OD.DishID = MD.DishID
        group by Orders.OrderID, CustomerID
        having sum(price * Quantity) > @input
go
```

GetOrderValue

Funkcja zwraca wartość zamówienia dla przekazanego ID zamówienia

```
CREATE function GetOrderValue(@input int)
returns money
as
    begin
        return CONVERT(money,(select sum(price * quantity * (1 - isnull(discount, 0)))
        from OrderDetails
        join MenuDetails MD on MD.MenuID = OrderDetails.MenuID and MD.DishID = OrderDetails.DishID
        join Orders O on OrderDetails.OrderID = O.OrderID
        where O.OrderID = @input))
    end
go
```

GetTopSellingMeals

Funkcja zwraca najbardziej popularne dania (na podstawie ilości ich zamówień). Zwróci ilość dań w zależności od przekazanej wartości

```
create function getTopSellingMeals(@input int)
returns table as
    return select distinct top (@input) DishName, sum(Quantity) as soldTimes
    from Dishes
join MenuDetails MD on Dishes.DishID = MD.DishID
join OrderDetails OD on MD.MenuID = OD.MenuID and MD.DishID = OD.DishID
group by DishName
order by sum(Quantity)
go
```

GetValueOfOrdersOnDay

Funkcja zwraca wartość wszystkich zamówień dla przekazanego dnia (uw. trzeba w argumencie podać właściwy format!)

```
create function GetValueOfOrdersOnDay(@date date)
    returns money
as
begin
    return (select sum(Quantity * Price * (1 - isnull(discount, 0)))
            from OrderDetails
            join MenuDetails MD on OrderDetails.MenuID = MD.MenuID and OrderDetails.DishID = MD.DishID
            join Orders O on OrderDetails.OrderID = O.OrderID
            where year(@date) = year(OrderDate)
            and month(@date) = month(OrderDate)
            and day(@date) = day(OrderDate)
            )
end
go
```

GetValueOfOrdersOnWeek

Funkcja zwraca wartość wszystkich zamówień dla przekazanego tygodnia

```
create function getValueOfOrdersOnWeek(@year int, @month int, @week int)
    returns money
as
begin
    return (select sum(Quantity * Price * convert(money,(1 - isnull(discount, 0)))) as value
            from OrderDetails
            join MenuDetails MD on OrderDetails.MenuID = MD.MenuID and OrderDetails.DishID = MD.DishID
            join Orders O on OrderDetails.OrderID = O.OrderID
            where @year = year(OrderDate)
            and @month = month(OrderDate)
            and @week = datepart(week, OrderDate)
            )
end
go
```

GetValueOfOrdersOnMonth

Funkcja zwraca wartość wszystkich zamówień dla przekazanego miesiąca (rok, miesiac)

```
create function getValueOfOrdersOnMonth(@year int, @month int)
    returns money
as
begin
    return (select sum(Quantity * Price * convert(money,(1 - isnull(discount, 0)))) as value
            from OrderDetails
            join MenuDetails MD on OrderDetails.MenuID = MD.MenuID and OrderDetails.DishID = MD.DishID
            join Orders O on OrderDetails.OrderID = O.OrderID
            where @year = year(OrderDate)
            and @month = month(OrderDate)
            )
end
go
```

GetValueOfOrdersOnYear

Funkcja zwraca wartość wszystkich zamówień dla przekazanego roku

```
create function getValueOfOrdersOnYear(@year int)
    returns money
as
begin
    return (select sum(Quantity * Price * convert(money,(1 - isnull(discount, 0)))) as value
            from OrderDetails
            join MenuDetails MD on OrderDetails.MenuID = MD.MenuID and OrderDetails.DishID = MD.DishID
            join Orders O on OrderDetails.OrderID = O.OrderID
            where @year = year(OrderDate)
            )
end
go
```

menuCoverage

```
CREATE function menuCoverage(@NewMenuDetails MenuDetailsType READONLY)
returns real
as
    begin
        declare @prevMenuID int
        declare @bDate datetime = DATEADD(day, -14, GETDATE())
        SELECT @prevMenuID=(SELECT TOP 1 MenuID FROM Menu WHERE isnull(
            endDate, GETDATE()) > @bDate AND startDate < @bDate)

        declare @newMenuLen int = (SELECT COUNT(*) FROM @NewMenuDetails)
        declare @repeating int = 0

        SELECT @repeating = (SELECT COUNT(*) FROM @NewMenuDetails WHERE DishID in (
        SELECT DishID FROM MenuDetails WHERE MenuID = @prevMenuID))

        return CAST(@repeating AS real)/CAST(@newMenuLen AS real)
    end
go
```

Triggery

checkDiscountVariables

Opis: Sprawdza czy Współczynniki do zniżek wstawiane są pojedynczo

```
create trigger checkDiscountVariables on DiscountVariables
 for insert
 as
BEGIN
    if ((SELECT COUNT(*) FROM inserted)>1)
    begin
        RAISERROR('Współczynniki powinny być dodawane pojedynczo', 16, 1)
        rollback transaction
    end
END
 go
```

checkReservationsVariables

Opis: Sprawdza czy współczynniki rezerwacji wstawiane są pojedynczo

```
create trigger checkReservationsVariables on ReservationVariables
for insert
as
BEGIN
    if ((SELECT COUNT(*) FROM inserted)>1)
    begin
        RAISERROR('Współczynniki powinny być dodawane pojedynczo', 16, 1)
        rollback transaction
    end
END
go
```

menuDelete

Opis: po usunięciu menu z tabeli Menu usuwane są też wiersze dotyczące tego menu z tabeli MenuDetails

```
create trigger menuDelete on Menu
instead of delete
as
BEGIN
    declare @DeletedID int
    SELECT @DeletedID=MenuID FROM deleted

    DELETE MenuDetails WHERE MenuID = @DeletedID
    DELETE Menu WHERE MenuID = @DeletedID
END
go
```

orderDelete

Opis: Po usunięciu zamówienia z tabeli Orders usuwane są też powiązane wiersze z tabeli OrderDetails

```
create trigger ordersDeletion on Orders
instead of delete
as
BEGIN
    declare @DeletedID int
    SELECT @DeletedID=OrderID FROM deleted

    DELETE OrderDetails WHERE OrderID = @DeletedID
    DELETE Orders WHERE OrderID = @DeletedID
END
go
```

menuCoverageCheck

```
CREATE trigger menuCoverageCheck ON Menu
 after insert
 as
BEGIN
     declare @MenuID int = (SELECT MenuID FROM inserted)

     declare @coverage real
     exec @coverage = dbo.menuCoverage  @NewMenuID: @MenuID

     if @coverage < 0.5
     begin
         rollback transaction
     end
END
 go
```

Role

Manager

```
use u_wiercigr
CREATE ROLE Manager

GRANT EXECUTE ON addCategory TO Manager
GRANT EXECUTE ON addDish TO Manager
GRANT EXECUTE ON addDishToMenu TO Manager
GRANT EXECUTE ON addEmployee TO Manager
GRANT EXECUTE ON addMenu TO Manager
GRANT EXECUTE ON addTable TO Manager
GRANT EXECUTE ON declineReservation TO Manager
GRANT EXECUTE ON acceptReservation TO Manager

GRANT SELECT ON TakeawayOrders TO Manager
GRANT SELECT ON DiscountUsedMonthly TO Manager
GRANT SELECT ON DiscountUsedWeekly TO Manager
GRANT SELECT ON UsedTablesMonthly TO Manager
GRANT SELECT ON UsedTablesWeekly TO Manager
GRANT SELECT ON pendingReservations TO Manager
GRANT SELECT ON SoldMealsInfoMonthly TO Manager
GRANT SELECT ON SoldMealsInfoWeekly TO Manager
GRANT SELECT ON CurrentMenu TO Manager
GRANT SELECT ON SeaFoodDeliveries to Manager
GRANT SELECT ON CompanyStatistics TO Manager

GRANT SELECT, INSERT, DELETE, UPDATE ON DiscountVariables TO Manager
GRANT SELECT, INSERT, DELETE, UPDATE ON ReservationVariables TO Manager
GRANT SELECT, DELETE, UPDATE ON Tables TO Manager
GRANT SELECT, DELETE, UPDATE ON MenuDetails TO Manager
GRANT SELECT, DELETE, UPDATE ON Menu TO Manager
GRANT SELECT, DELETE, UPDATE ON Dishes TO Manager
GRANT SELECT, DELETE, UPDATE ON Categories TO Manager
```

Employee

```
use u_wiercigr
CREATE ROLE Employee
GRANT SELECT ON Tables to Employee
GRANT SELECT ON freeTables to Employee
GRANT SELECT ON OrdersInfo to Employee
GRANT SELECT ON ActiveClientsDiscounts to Employee
GRANT SELECT ON CurrentMenu to Employee
GRANT SELECT ON ConfirmedReservations to Employee
GRANT SELECT ON MealsInfo to Employee
GRANT SELECT ON ConfirmedReservations to Employee
GRANT SELECT ON ordersInvoicesForCompanies to Employee
GRANT SELECT ON OrdersInvoicesForIndividualCustomers to Employee
GRANT SELECT ON OrdersToday to Employee
GRANT SELECT ON pendingReservations to Employee
GRANT SELECT ON SeaFoodDeliveries to Employee
GRANT SELECT ON TakeAwayOrders to Employee


GRANT EXECUTE ON addCompanyReservation to Employee
GRANT EXECUTE ON addIndividualReservation to Employee
GRANT EXECUTE ON addPersonToReservation to Employee
GRANT EXECUTE ON ModifyReservationStatus to Employee
GRANT EXECUTE ON ModifyOrderPaidStatus to Employee
GRANT EXECUTE ON addOrder to Employee
GRANT EXECUTE ON addCompanyCustomer to Employee
GRANT EXECUTE ON addIndividualCustomer to Employee
```

Customer

```
use u_wiercigr
CREATE ROLE Customer

GRANT EXECUTE ON GetOrderValue to Customer
GRANT EXECUTE ON addOrder to Customer
```

IndividualCustomer

```
use u_wiercigr
CREATE ROLE IndividualCustomer

GRANT EXECUTE ON avilableCustomerDiscounts to IndividualCustomer
GRANT EXECUTE ON addIndividualReservation to IndividualCustomer
```

CompanyCustomer

```
use u_wiercigr
CREATE ROLE CompanyCustomer

GRANT EXECUTE ON addCompanyReservation to CompanyCustomer
GRANT EXECUTE ON addPersonToReservation to CompanyCustomer
```

Admin

```
use u_wiercigr
CREATE ROLE Admin
grant all privileges ON u_wiercigr.dbo TO admin
```

Indeksy

Customers_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Customers_pk
ON Customers (CustomerID)
```

CompanyCustomers_pk

```
use u_wiercigr
CREATE UNIQUE INDEX CompanyCustomers_pk
ON CompanyCustomers (CustomerID)
```

DiscountVariables_pk

```
use u_wiercigr
CREATE UNIQUE INDEX DiscountVariables_pk
ON DiscountVariables (VariablesID)
```

IndividualCustomers_pk

```
use u_wiercigr
CREATE UNIQUE INDEX IndividualCustomers_pk
ON IndividualCustomers (CustomerID)
```

Employees_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Employees_pk
ON Employees (EmployeeID)
```

Orders_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Orders_pk
ON Orders (OrderID)
```

OrderDetails_pk

```
use u_wiercigr
CREATE UNIQUE INDEX OrderDetails_pk
ON OrderDetails (OrderID, DishID, MenuID)
```

ReservationsDetails_pk

```
use u_wiercigr
CREATE UNIQUE INDEX ReservationsDetails_pk
ON ReservationsDetails (ReservationID)
```

ReservationVariables_pk

```
use u_wiercigr
CREATE UNIQUE INDEX ReservationVariables_pk
ON ReservationVariables (VariablesID)
```

Tables_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Tables_pk
ON Tables (TableID)
```

Dishes_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Dishes_pk
ON Dishes (DishID)
```

Categories_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Categories_pk
ON Categories (CategoryID)
```

Menu_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Menu_pk
ON Menu (MenuID)
```

MenuDetails_pk

```
use u_wiercigr
CREATE UNIQUE INDEX MenuDetails_pk
ON MenuDetails (MenuID, DishID)
```

Discounts_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Discounts_pk
ON Discounts (DiscountID)
```

CompanyReservations_pk

```
use u_wiercigr
CREATE UNIQUE INDEX CompanyReservations_pk
ON CompanyReservations (ReservationID)
```

IndividualReservations_pk

```
use u_wiercigr
CREATE UNIQUE INDEX IndividualReservations_pk
ON IndividualReservations (ReservationID)
```

Reservations_pk

```
use u_wiercigr
CREATE UNIQUE INDEX Reservations_pk
ON Reservations (ReservationID)
```

IndividualCustomers_LastName_index

```
create index IndividualCustomers_LastName_index
    on dbo.IndividualCustomers (LastName)
go
```

UniqueCompanyNip

```
create unique index uniqueCompanyNIP
    on dbo.CompanyCustomers (NIP)
go
```

email_Unique

```
create unique index email_Unique
    on dbo.Customers (Email)
go
```

Phone_Unique

```
create unique index phone_Unique
    on dbo.Customers (Phone)
go
```

UniqueDishName

```
create unique index uniqueDishName
    on dbo.Dishes (DishName)
go
```

Unique_EmailEmployee

```
create unique index Unique_EmailEmployee
    on dbo.Employees (Email)
go
```

Unique_PhoneEmployee

```
create unique index Unique_PhoneEmployee
    on dbo.Employees (Phone)
go
```

uniqueCateogryName

```
create unique index uniqueCateogryName
    on dbo.Categories (CategoryName)
go
```

CompanyCustomers_CompanyName_Index

```
create index CompanyCustomers_CompanyName_Index
    on dbo.CompanyCustomers (CompanyName)
go
```

IndividualCustomers_LastName_index

```
create index IndividualCustomers_LastName_index
    on dbo.IndividualCustomers (LastName)
go
```

MenuDetails_Price_index

```
create index MenuDetails_Price_index
    on dbo.MenuDetails (Price)
go
```