

Raport

Hieronim Koc

Przemysław Rola

Remigiusz Kozicki

Spis treści

Warunki integralności	3
Products:	3
Categories:	3
Orders	3
Discount	3
Customers	3
Companies	3
Menu	3
Reservation	3
Table	3
Opisy i funkcje	4
Schemat baz danych	6
Tabele	7
Categories	7
Products	8
Customers	9
Reservation	10
Receipt	11
Orders	12
Order Details	13
Tables	14
Reservation Details	15
Companies	16
Individual Customers	17
Discount	18
	1

Discount Details	19
ConditionOrders	20
Menu	21
Menu Details	22
Names Reservation	23
Payments	24
Widoki	25
Procedury	44
Funkcje	55
Triggery	60
Indeksy	61
Uprawnienia	62

Warunki integralności

Poza danymi widocznymi na schemacie (klucz, klucze zewnętrzne, notnull)

Products:

PricePerUnit numeric(6,2) CONSTRAINT minprice CHECK PricePerUnit > 0

ProductName UNIQUE

Categories:

CategoryName UNIQUE

Orders

(Reservation, CustomerID, OrderDate) UNIQUE

Reservation może być NULL ! <- wtedy jest na Wynos

Discount

Value TINYINT CHECK >0 AND <1

Customers

Adres NOT NULL

Email UNIQUE CHECK LIKE '%@%.%'

Phone CHECK LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'

Companies

NIP NOT NULL UNIQUE

Menu

CHECK StartDate < EndDate

Reservation

CHECK StartDate < EndDate

Table

Seats NOT NULL CHECK > 0

Opisy i funkcje

Opis użytkowników:

1. Klient indywidualny:
 - rezerwować stoliki,
 - składać zamówienia,
 - płacić z góry za zamówienie lub po realizacji usługi,
 - generować swoje dane (dane, raporty, historie swoich zamówień)
2. Klient-firma:
 - rezerwować stoliki i składać zamówienia w dwóch opcjach: dla firmy i dla poszczególnych pracowników (imienne),
 - płacić z góry za zamówienie lub po realizacji usługi,
 - generować swoje dane (dane, raporty, historie swoich zamówień)
3. Pracownik restauracji:
 - aktualizować menu,
 - tworzyć listę produktów które na należy importować (zamówić),
 - ma dostęp do danych klientów (dane, raporty, historie zamówień),
 - akceptować rezerwacje i zamówienia,
 - ustalać ceny produktów,
 - generować raporty miesięczne, tygodniowe,
 - generować statystyki zamówień,
 - generować raporty zamówień i rabatów dla poszczególnych klientów

Funkcje użytkowników

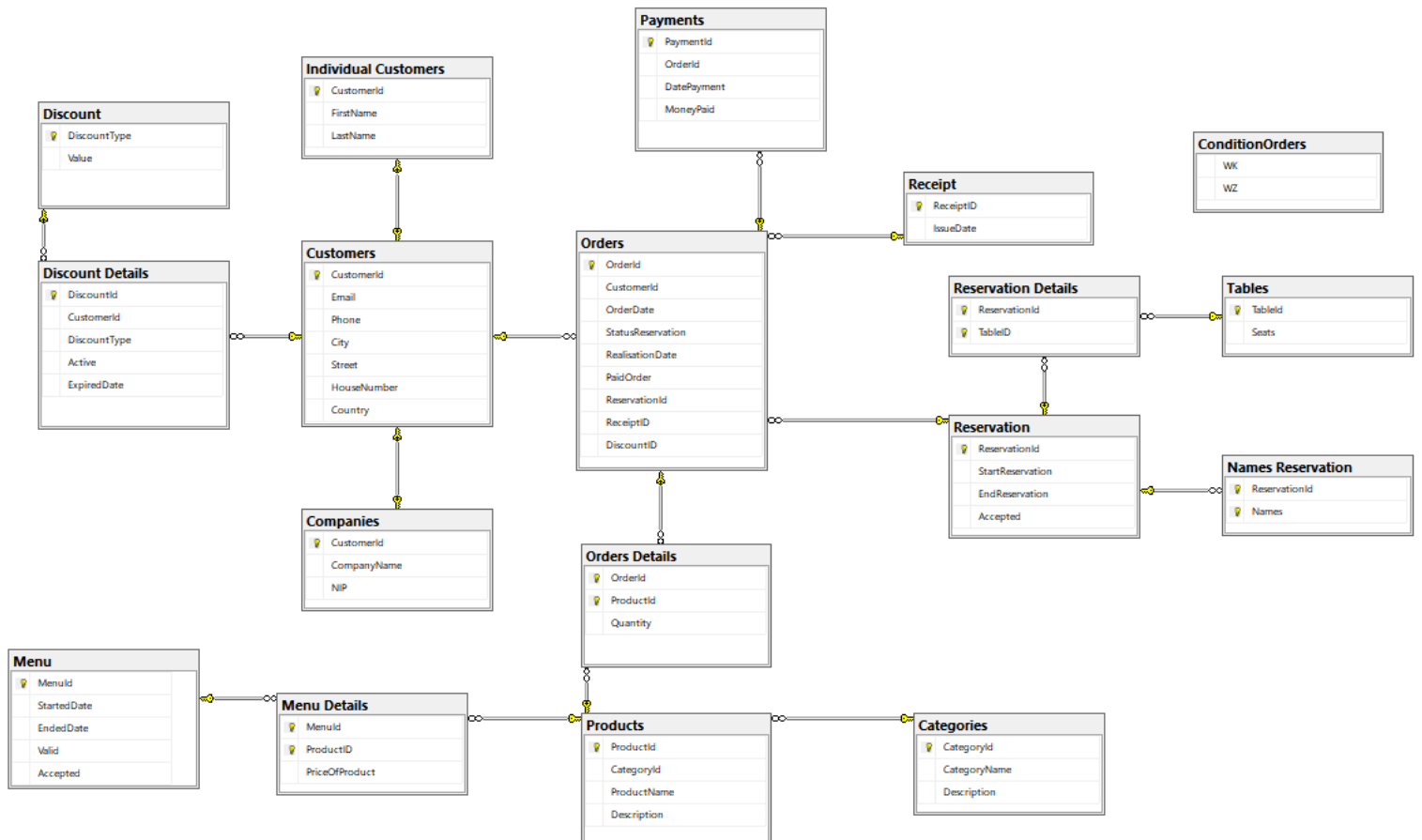
- Klient indywidualny:
 - Składanie zamówienia
 - Rezerwacja stolików
 - Wyświetlić dostępne rabaty
 - Wyświetlić swoją historię zamówień
 - Wygenerować raporty własnych zamówień i rabatów
- Klient-firma:
 - Składanie zamówienia
 - Rezerwacja stolików na firmę i imienne dla pracowników
 - Wyświetlić dostępne rabaty
 - Wyświetlić historię zamówień firmy
 - Wygenerować raporty zamówień i rabatów firmy
- Pracownik restauracji:
 - Wyświetlić rabaty przysługujące klientowi
 - Tworzenie listy produktów wymagających importu (zamówienia)
 - Wyświetlić historię zamówień i rabatów konkretnego klienta

- Wyświetlić historię zamówień i rabatów konkretnej firmy
- Wystawiać faktury dla jednego zamówienia
- Wystawiać faktury zbiorczej raz na miesiąc
- Aktualizowanie menu
- Akceptacja zamówień i rezerwacji
- Wyświetlać aktualny stan zarezerwowania Sali
- Zmiana cen produktów
- Generowanie raportów tygodniowych
- Generowanie raportów miesięcznych
- Generowanie raportów dotyczących stolików
- Generowanie rabatów
- Generowanie statystyk zamówień klientów indywidualnych
- Generowanie statystyk zamówień firm
- Generowanie raportów menu
- Dodawanie produktów i ich kategorii

Funkcje systemowe

- Sprawdzanie dostępności stolików w danym w danym czasie
- Sprawdzanie możliwości złożenia zamówienia przez klienta
- Podliczanie kosztów zamówienia
- Przydzielanie rabatów klientom
- Sprawdzanie poprawności wprowadzonego menu

Schemat baz danych



Tabele

Categories

Zawiera wszystkie kategorie produktów dostępnych w restauracji.

Klucz główny: CategoryID

Nazwa Kategorii: CategoryName

Opis Kategorii: Description

```
CREATE TABLE Categories (  
    CategoryId INT IDENTITY,  
    CategoryName VARCHAR(50) NOT NULL,  
    Description VARCHAR(200),  
    PRIMARY KEY (CategoryId)  
);  
  
ALTER TABLE Categories ADD CONSTRAINT UniqueCategoryName  
    UNIQUE ( CategoryName)
```

Products

Zawiera wszystkie produkty dostępnych w restauracji.

Klucz Główny: ProductID

Kategoria do której należy: CategoryID

Nazwa Produktu: ProductName

```
CREATE TABLE Products (  
    ProductId INT IDENTITY,  
    CategoryId INT NOT NULL,  
    Description VARCHAR(256),  
    ProductName VARCHAR(50) NOT NULL,  
    PRIMARY KEY (ProductId),  
    FOREIGN KEY (CategoryId) REFERENCES Categories(CategoryId)  
);  
  
ALTER TABLE Products ADD CONSTRAINT UniqueProductName UNIQUE (ProductName)
```


Customers

Zawiera wszystkich klientów

Klucz główny: CustomerID

Email Klienta (opcjonalny): Email

Numer telefonu Klienta (opcjonalny): Phone

Adres zamieszkania: Street, HouseNumber, Country

```
CREATE TABLE Customers (  
    CustomerId INT IDENTITY,  
    Email VARCHAR(64),  
    Phone VARCHAR(64),  
    City VARCHAR(64) NOT NULL,  
    Street VARCHAR(64) NOT NULL,  
    HouseNumber VARCHAR(64) NOT NULL,  
    Country VARCHAR(64) NOT NULL,  
    PRIMARY KEY (CustomerId)  
);  
  
ALTER TABLE Customers ADD CONSTRAINT ProperEmail CHECK (Email LIKE '%@%.%')  
  
ALTER TABLE Customers ADD CONSTRAINT ProperPhoneNumber CHECK (Phone LIKE  
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' OR Phone LIKE  
'+[0-9][0-9]-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

Reservation

Zawiera Wszystkie rezerwacje na stoliki w restauracji

Klucz główny: ReservationID

Godzina rozpoczęcia rezerwacji: StartReservation

Godzina zakończenia rezerwacji: EndReservation

Informacja czy zostało zaakceptowane: Accepted

```
CREATE TABLE Reservation (  
    ReservationId INT IDENTITY,  
    StartReservation DATETIME NOT NULL,  
    EndReservation DATETIME NOT NULL,  
    Accepted BIT NOT NULL,  
    PRIMARY KEY (ReservationId)  
);  
  
ALTER TABLE Reservation ADD CONSTRAINT ProperDateReservation CHECK  
(StartReservation < EndReservation)
```

Receipt

zawiera informacje o paragonach wystawionych na zamówienie

Numer paragonu: ReceiptID

Data wydania: IssueDate

```
CREATE TABLE Receipt (  
    ReceiptID INT IDENTITY,  
    IssueDate DATE NOT NULL,  
    PRIMARY KEY (ReceiptID)  
);
```

Orders

Klucz główny: OrderID

Kupujący klient: CustomerID

Data złożenia zamówienia: OrderDate

Status przyjęcia zamówienia: StatusReservation

Data realizacji zamówienia: RelisationDate

Informacja czy zapłacone przy zamówieniu: PaidOrder

Numer Rezerwacji (jeśli jest takowa): ReservationID

Numer Paragonu (jeśli jest wydany): ReceiptID

```
CREATE TABLE Orders (  
    OrderId INT IDENTITY,  
    CustomerId INT NOT NULL,  
    OrderDate DATETIME NOT NULL,  
    StatusReservation BIT NOT NULL,  
    RealisationDate DATETIME,  
    PaidOrder BIT NOT NULL,  
    ReservationId INT,  
    ReceiptID INT,  
    PRIMARY KEY (OrderId),  
    FOREIGN KEY (CustomerId) REFERENCES Customers(CustomerId),  
    FOREIGN KEY (ReservationId) REFERENCES Reservation(ReservationId),  
    FOREIGN KEY (ReceiptID) REFERENCES Receipt(ReceiptID)  
);  
  
ALTER TABLE Orders ADD CONSTRAINT UniqueOrderInfo UNIQUE  
(Reservationid, CustomerID, OrderDate)
```

Order Details

Zawiera informacje jakie produkty i w jakiej ilości zostały zamówione.

Numer zamówienia które dotyczy: OrderID

Numer produktu którego dotyczy: ProductID

Ilość zamówionych sztuk: Quantity

```
CREATE TABLE [Orders Details] (  
    OrderId INT NOT NULL,  
    ProductId INT NOT NULL,  
    Quantity SMALLINT NOT NULL,  
    Discount INT NOT NULL,  
    PRIMARY KEY (OrderId, ProductId),  
    FOREIGN KEY (ProductId) REFERENCES Products(ProductId),  
    FOREIGN KEY (OrderId) REFERENCES Orders(OrderId)  
);
```

Tables

zawiera informacje na temat stolików dostępnych w restauracji

Numer Stolika: TableID

Ilość miejsc przy stoliku: Seats

```
CREATE TABLE [Tables] (  
    TableId INT,  
    Seats TINYINT NOT NULL,  
    PRIMARY KEY (TableId)  
);  
  
ALTER TABLE Tables ADD CONSTRAINT ProperSeatsNumber CHECK (Seats > 0)
```

Reservation Details

informacje na temat jakie stoły zostały zarezerwowane dla danego zamówienia

Numer rezerwacji: ReservationID

Numery stolików: TableID

```
CREATE TABLE [Reservation Details] (  
    ReservationId INT IDENTITY,  
    TableID INT NOT NULL,  
    PRIMARY KEY (ReservationId, TableID),  
    FOREIGN KEY (ReservationId) REFERENCES Reservation (ReservationId),  
    FOREIGN KEY (TableID) REFERENCES [Tables] (TableId)  
);
```

Companies

podtabela klientów, zawierająca opis klientów firmowych

Numer klienta: CustomerID

Nazwa firmy: CompanyName

NIP: NIP

```
CREATE TABLE Companies (  
    CustomerId INT NOT NULL,  
    CompanyName VARCHAR(64) NOT NULL,  
    NIP VARCHAR(10) NOT NULL,  
    PRIMARY KEY (CustomerId),  
    FOREIGN KEY (CustomerId) REFERENCES Customers(CustomerId)  
);  
ALTER TABLE Companies ADD CONSTRAINT UniqueNip UNIQUE (NIP)
```


Individual Customers

podtabela klientów, zawierająca opis klientów indywidualnych

Numer klienta: CustomerID

Imię: FirstName

Nazwisko: LastName

```
CREATE TABLE [Individual Customers] (  
    CustomerId INT NOT NULL,  
    FirstName VARCHAR(64) NOT NULL,  
    LastName VARCHAR(64) NOT NULL,  
    PRIMARY KEY (CustomerId),  
    FOREIGN KEY (CustomerId) REFERENCES Customers(CustomerId)  
);
```

Discount

tabela zawiera zniżki jakie występują w restauracji

Numer zniżki: DiscountId

Wartość zniżki: Value

```
CREATE TABLE Discount (  
    DiscountId INT NOT NULL,  
    Value TINYINT NOT NULL,  
    PRIMARY KEY (DiscountId)  
);  
  
ALTER TABLE Discount ADD CONSTRAINT ProperDiscount CHECK (Value BETWEEN 0  
                                                                AND 1)
```

Discount Details

tabela zawierająca szczegółowe informacje i konkretnym zamówieniu

Klucz główny: DiscountId

ID klienta: CustomerId

ID zniżki/ typ zniżki: DiscountType

Flaga czy zniżka jest aktywna: Active

Data przedawnienia zniżki: ExpiredDate

```
CREATE TABLE [Discount Details] (  
    DiscountId INT NOT NULL,  
    CustomerId INT NOT NULL,  
    Discount INT NOT NULL,  
    Active BIT NOT NULL,  
    ExpiredDate DATETIME,  
    PRIMARY KEY (DiscountId),  
    FOREIGN KEY (Discount) REFERENCES Discount(DiscountId),  
    FOREIGN KEY (CustomerId) REFERENCES Customers(CustomerId)  
);
```

ConditionOrders

tabela stałych wartości

Minimalna wartość zamówienia online: WK

Minimalna ilość zamówień przed zamówieniem online: WZ

Ilość zamówień do otrzymania stałej 3% zniżki: Z1

Minimalna wartość zamówienia uruchamiająca zniżkę 3%: K1

Minimalna łączna kwota po której przyznawana: K2

Ilość zamówień wymagana do otrzymania zniżki jednorazowej: Z3

Łączna kwota wymagana do otrzymania zniżki jednorazowej: K4

```
CREATE TABLE ConditionOrders (  
    WK 50,  
    WZ 3,  
    Z1 5,  
    K1 50,  
    K2 500,  
    Z3 2,  
    K4 200  
);
```

Menu

tabela zawierająca informacje o każdym menu

Klucz główny: MenuId

Początek obowiązywania menu: StartedDate

Koniec obowiązywania menu: EndedDate

Flaga czy menu jest poprawne: Valid

Flaga czy menu jest wprowadzone do użytku: Accepted

```
CREATE TABLE Menu (  
    MenuId INT IDENTITY,  
    StartedDate DATETIME NOT NULL,  
    EndedDate DATETIME NOT NULL,  
    Valid BIT NOT NULL,  
    Accepted BIT NOT NULL,  
    PRIMARY KEY (MenuId)  
);  
  
ALTER TABLE Menu ADD CONSTRAINT ProperDate CHECK (StartedDate < EndedDate)
```

Menu Details

tabela zawierająca informacje o zawartości menu

Klucz główny: MenuId

Klucz główny: ProductID

Cena produktu: PriceOfProduct

```
CREATE TABLE [Menu Details] (  
    MenuId INT NOT NULL,  
    ProductID INT NOT NULL,  
    PriceOfProduct MONEY NOT NULL,  
    PRIMARY KEY (MenuId, ProductID),  
    FOREIGN KEY (MenuId) REFERENCES Menu (MenuId),  
    FOREIGN KEY (ProductID) REFERENCES Products (ProductId)  
);  
  
ALTER TABLE [Menu Details] ADD CONSTRAINT positivePrice CHECK (PriceOfProduct > 0)
```

Names Reservation

tabela z informacjami dotyczącymi rezerwacji

Klucz główny: ReservationId

Klucz główny: Names

```
CREATE TABLE [Names Reservation] (  
    ReservationId INT NOT NULL,  
    Names VARCHAR(64) NOT NULL,  
    PRIMARY KEY (ReservationId),  
    PRIMARY KEY (ReservationId) REFERENCES Reservation(ReservationId)  
);
```

Payments

tabela zawierająca informacje o płatnościach za zamówienia

Klucz główny: PaymentsId

ID zamówienia którego dotyczy płatność: OrderId

Data zapłaty: DatePayments

Kwota którą zapłacono: MoneyPaid

```
CREATE TABLE Payments (  
    PaymentId INT IDENTITY,  
    OrderId INT NOT NULL,  
    DatePayment DATETIME NOT NULL,  
    PRIMARY KEY (PaymentId),  
    FOREIGN KEY (OrderId) REFERENCES Orders (OrderId)  
);  
  
ALTER TABLE Payments ALTER COLUMN MoneyPaid MONEY NOT NULL
```


Widoki

1. **ClientStats** - wyświetla informacje o kliencie razem z ilością zamówień i ich wartości

```
create view ClientStats as
select c.CustomerId,
       c.Country,
       c.City,
       c.Street,
       count(o.OrderId) as [amount of orders],
       isnull(c.Email, 'BRAK') as email,
       isnull(c.Phone, 'BRAK') as phone,
       isnull(sum(ov.value), 0) as [sum of orders]
from Customers c
inner join orders o
  on c.CustomerId = o.CustomerId
left join orderValue ov
  on ov.OrderId = o.OrderId
group by c.CustomerId, c.Country, c.City, c.Street,
         isnull(c.Email, 'BRAK'), isnull(c.Phone, 'BRAK')
go
```

2. **CurrentMenu** – wyświetla produkty w aktualnym menu razem z ich ceną

```
CREATE VIEW CurrentMenu
AS
SELECT ProductName AS Dish, PriceOfProduct [Regular Price] FROM Menu m
JOIN [Menu Details] md ON md.MenuId = m.MenuId AND GETDATE() BETWEEN
    m.StartedDate AND m.EndedDate
JOIN Products p ON p.ProductId = md.ProductID
```

3. **DiscountInfoMonthly** - wyświetla ilość zniżek permanentnych oraz czasowych przyznanych w danym roku oraz miesiącu

```
create view DiscountInfoMonthly
as
SELECT YEAR(OrderDate) year, MONTH(OrderDate) month, count(DiscountId)
'amount of discounts'
from Orders
group by YEAR(OrderDate), MONTH(OrderDate)
```

4. **DiscountInfoWeekly** - Wyświetla ilość zniżek permanentnych oraz czasowych przyznanych w danym roku oraz tygodniu

```
create view DiscountInfoWeekly
as
SELECT YEAR(OrderDate) year, datepart(week , OrderDate) weak,
count(DiscountId) 'amount of discounts'
from Orders
group by YEAR(OrderDate), datepart(week , OrderDate)
```

5. **MealsInfo** - wyświetla informacje o produktach

```
create view MealsInfo as
select p.ProductName ,c.CategoryName,
       isnull(c.Description, 'BRAK OPISU') as categoryDescription,
       isnull(p.Description, 'BRAK OPISU') as productsDescription
from Products p
inner join Categories c
    on p.CategoryId = c.CategoryId
go
```

6. **MealsSoldInfo** - wyświetla ile razy został sprzedany dany produkt

```
create view MealsSoldInfo as
select p.ProductName, count(od.OrderId) as [amount of products]
from Products p
left join [Orders Details] od
    on p.ProductId = od.ProductId
group by p.ProductName
go
```

7. **MealsSoldMonthly** - wyświetla ilość sprzedanych produktów z podziałem na lata i miesiące

```
create view MealsSoldMonthly as
select year(o.OrderDate) as year,
       month(o.OrderDate) as month,
       p.ProductName,
       isnull(count(p.ProductName), 0) as [Amount Of Sold]
from Products p
join [Orders Details] od
    on p.ProductId = od.ProductId
join Orders o
    on od.OrderId = o.OrderId
group by p.ProductName, month(o.OrderDate), year(o.OrderDate)
go
```

8. **MealsSoldWeekly** - wyświetla ilość sprzedanych produktów z podziałem na lata i tygodnie

```
CREATE view MealsSoldWeekly as
select year(o.OrderDate) as year,
       datepart(week, o.OrderDate) as week,
       p.ProductName,
       isnull(count(p.ProductName), 0) as [Amount Of Sold]
from Products p
inner join [Orders Details] od
    on p.ProductId = od.ProductId
inner join Orders o
    on od.OrderId = o.OrderId
group by year(o.OrderDate), datepart(week, o.OrderDate), p.ProductName
go
```


9. **MenuStats** - wyświetla ile razy dany produkt pojawił się w menu, razem z średnią ceną, max i min ceną

```
CREATE VIEW MenuStats
AS
SELECT ProductName AS Dish, COUNT(MenuId) AS [times on menu],
      AVG(PriceOfProduct) [AVG Price], MAX(PriceOfProduct) [MAX Price],
      MIN(PriceOfProduct) [MIN Price]
FROM Products p
LEFT JOIN [Menu Details] md ON md.ProductID = p.ProductId
GROUP BY ProductName
go
```

10. **OrderInfo** - wyświetla informacje o zamówieniu, klienta który je złożył, datę jego złożenia, czy zostało opłacone oraz wartość zamówienia

```
create view OrderInfo
as
SELECT OrderId, (select SUM(OD.Quantity * PriceOfProduct)
                  from [Orders Details] OD
                  inner join Products P on P.ProductId = OD.ProductId
                  inner join [Menu Details] MD on P.ProductId = MD.ProductID
                  where OD.OrderId = O.OrderId
                  GROUP BY OD.OrderId) as 'order value',
CustomerId,
OrderDate,
PaidOrder

from Orders O
go
```

11. OrdersStatsMonthly - ilość zamówień w danym miesiącu oraz ich całkowita wartość

```
CREATE VIEW OrdersStatsMonthly
AS
WITH
    mindate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate),

    maxdate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate DESC),

    rm(dt) AS (
        SELECT (SELECT * FROM mindate) as dt
        UNION all
        SELECT dateadd(mm,1,dt) FROM rm WHERE dt<(SELECT * FROM maxdate)
    ),

    orderValue AS (
        SELECT o.OrderId, sum(Quantity*PriceOfProduct*(1-IsNull(d.value,0))
                                AS value FROM Orders o
        JOIN [Orders Details] od ON o.OrderId = od.OrderId
        JOIN Menu m ON o.OrderDate BETWEEN m.StartedDate AND m.EndedDate
        JOIN [Menu Details] md on m.MenuId = md.MenuId AND md.ProductID =
                                od.ProductId
        LEFT JOIN [Discount Details] dd ON dd.DiscountId = o.DiscountID
        LEFT JOIN Discount d ON d.DiscountType = dd.DiscountType
        GROUP BY o.OrderId
    ),

    t AS
        (SELECT year(orderdate) AS year, month(orderdate) AS month, count(*)
                                AS cnt, sum(ov.value) AS value
        FROM Orders o
        JOIN orderValue ov ON o.OrderId = ov.OrderId
        GROUP BY year(orderdate), month(orderdate)
        )

SELECT year(dt) AS ROK, month(dt) AS MIESIAC , ISNULL(cnt,0) AS
ILOSC_ZAMOWIEN, ISNULL(value,0) AS WARTOSC
FROM rm
LEFT JOIN t ON t.year = year(dt) AND t.month = month(dt);
```

12. **OrdersStatsWeekly** - wyświetla ilość zamówień i ich wartość z podziałem na lata i tygodnie

```
CREATE VIEW OrdersStatsWeekly
AS
WITH
    mindate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate),
    maxdate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate DESC),
    rm(dt) AS (
        SELECT (SELECT * FROM mindate) as dt
        UNION all
        SELECT dateadd(ww,1,dt) FROM rm WHERE dt<(SELECT * FROM maxdate)
    ),

    orderValue AS(
        SELECT o.OrderId, sum(Quantity*PriceOfProduct*(1-IsNull(d.value,0)))
                                                AS value FROM Orders o
        JOIN [Orders Details] od ON o.OrderId = od.OrderId
        JOIN Menu m ON o.OrderDate BETWEEN m.StartedDate AND m.EndedDate
        JOIN [Menu Details] md on m.MenuId = md.MenuId AND md.ProductID =
                                                od.ProductId
        LEFT JOIN [Discount Details] dd ON dd.DiscountId = o.DiscountID
        LEFT JOIN Discount d ON d.DiscountType = dd.DiscountType
        GROUP BY o.OrderId
    ),

    t AS
    (
        SELECT  YEAR(OrderDate) year, DATEPART(ww,orderDate) week, count(*)
                                                AS cnt, sum(ov.value) AS value
        FROM Orders o
        JOIN orderValue ov ON o.OrderId = ov.OrderId
        GROUP BY year(orderdate), DATEPART(ww,orderDate)
    )
SELECT  year(dt) AS ROK, DATEPART(ww,dt) AS TYDZIEN , ISNULL(cnt,0) AS
ILOSC_ZAMOWIEN ,ISNULL(value,0) AS WARTOSC_ZAMOWIEN
FROM _rm
LEFT JOIN t ON t.year = year(dt) AND t.week = DATEPART(ww,dt);
go
```

13. OrdersToPay - wyświetla zamówienia, które wymagają uregulowania należności

```
CREATE VIEW OrdersStatsWeekly
AS
WITH
    mindate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate),
    maxdate AS
        (SELECT TOP 1 Orderdate FROM Orders ORDER BY OrderDate DESC),
    rm(dt) AS (
        SELECT (SELECT * FROM mindate) as dt
        UNION all
        SELECT dateadd(ww,1,dt) FROM rm WHERE dt<(SELECT * FROM maxdate)
    ),
    orderValue AS (
        SELECT o.OrderId, sum(Quantity*PriceOfProduct*(1-IsNull(d.value,0)))
                                                AS value FROM Orders o

        JOIN [Orders Details] od ON o.OrderId = od.OrderId
        JOIN Menu m ON o.OrderDate BETWEEN m.StartedDate AND m.EndedDate
        JOIN [Menu Details] md on m.MenuId = md.MenuId AND md.ProductID =
                                                od.ProductId

        LEFT JOIN [Discount Details] dd ON dd.DiscountId = o.DiscountID
        LEFT JOIN Discount d ON d.DiscountType = dd.DiscountType
        GROUP BY o.OrderId
    ),
    t AS
        (SELECT YEAR(OrderDate) year, DATEPART(ww,orderDate) week, count(*)
                                                AS cnt, sum(ov.value) AS value

        FROM Orders o
        JOIN orderValue ov ON o.OrderId = ov.OrderId
        --where year(orderdate) BETWEEN (SELECT * FROM mindate) AND (SELECT *
FROM maxdate)
        group by year(orderdate), DATEPART(ww,orderDate)
    )
SELECT year(dt) AS ROK, DATEPART(ww,dt) AS TYDZIEN , ISNULL(cnt,0) AS
ILOSC_ZAMOWIEN ,ISNULL(value,0) AS WARTOSC_ZAMOWIEN
FROM rm
LEFT JOIN t ON t.year = year(dt) AND t.week = DATEPART(ww,dt);
go
```

14. **OrderValue** - wyświetla dla danego zamówienia wartość tego zamówienia

```
CREATE VIEW OrderValue
AS
SELECT o.OrderId, sum(Quantity*PriceOfProduct*(1-IsNull(d.value,0))) AS value
FROM Orders o
JOIN [Orders Details] od ON o.OrderId = od.OrderId
JOIN Menu m ON o.OrderDate BETWEEN m.StartedDate AND m.EndedDate
JOIN [Menu Details] md on m.MenuId = md.MenuId AND
                        md.ProductID =od.ProductId
LEFT JOIN [Discount Details] dd ON dd.DiscountId = o.DiscountID
LEFT JOIN Discount d ON d.DiscountType = dd.DiscountType
GROUP BY o.OrderId
go
```

15. **OwingClients** - wyświetla klientów, którzy muszą zapłacić za usługę

```
create view OwingClients
as
SELECT CustomerId, (isnull(Sum(MoneyPaid),0) - sum(OV.value)) as debt
from Orders O
left join Payments P2 on O.OrderId = P2.OrderId
inner join OrderValue OV on O.OrderId = OV.OrderId
where PaidOrder = 0
GROUP BY CustomerId
go
```

16. **PendingReservation** - wyświetla rezerwacje, które nie zostały jeszcze zrealizowane

```
create view PendingReservation as
select r.ReservationId, r.StartReservation, r.EndReservation, o.CustomerId,
o.OrderId, 'In processing' as [accepted?]
from Reservation r
inner join Orders o
    on r.ReservationId = o.ReservationId
where r.Accepted = 0
go
```


17. ReservationInfo - wyświetla informację o rezerwacjach

```
create view ReservationInfo as
select r.ReservationId, r.StartReservation, r.EndReservation,
count(rd.TableID) as [amount of tables]
from Reservation r
inner join [Reservation Details] rd
    on r.ReservationId = rd.ReservationId
where r.Accepted = 1
group by r.ReservationId, r.StartReservation, r.EndReservation
go
```

18. `TablesMonthly` - wyświetla statystyki miesięczne dotyczące rezerwacji stolików

```
create view TablesMonthly as
select year(r.StartReservation) as year,
       month(r.StartReservation) as month,
       t.TableId,
       t.Seats as [size of table],
       count(rd.TableID) as [how many times reserved]
from Tables t
inner join [Reservation Details] rd
    on t.TableId = rd.TableID
inner join Reservation r
    on rd.ReservationId = r.ReservationId
group by year(r.StartReservation), month(r.StartReservation), t.TableId,
t.Seats
go
```

19. **TablesWeekly** - wyświetla statystyki tygodniowe dotyczące rezerwacji stolików

```
create view TablesWeekly as
select year(r.StartReservation) as year,
       datepart(week, r.StartReservation) as week,
       t.TableId,
       t.Seats as [size of table],
       count(rd.TableID) as [how many times reserved]
from Tables t
inner join [Reservation Details] rd
    on t.TableId = rd.TableID
inner join Reservation r
    on rd.ReservationId = r.ReservationId
group by year(r.StartReservation), datepart(week, r.StartReservation),
t.TableId, t.Seats
go
```

Procedury

1. AddCustomer - procedura dodająca customera do bazy danych

```
CREATE PROCEDURE AddCustomer
@email varchar(64) = null,
@phone varchar(64) = null,
@city varchar(64),
@street varchar(64),
@houseNumber varchar(32),
@country varchar(64),
@clientType varchar(1),
@companyName varchar(64) = null,
@nip varchar(10) = null,
@firstName varchar(64) = null,
@lastName varchar(64) = null
as
begin
    set nocount on
    begin try
        if exists(
            select * from Companies
            where CompanyName = @CompanyName
        )
            begin
                ; throw 60000, N'Dana firma istnieje już w bazie', 1
            end
        if exists(
            select * from Companies
            where NIP = @NIP
        )
            begin
                ; throw 60000, N'Dany numer NIP istnieje już w bazie', 1
            end
        declare @CustomerId int
        insert into Customers (Email, Phone, City, Street, HouseNumber,
                               Country)
        values (@Email, @Phone, @City, @Street, @HouseNumber, @Country)
        set @CustomerId = scope_identity()
        if @ClientType = 'C'
            insert into Companies (CustomerId, CompanyName, NIP)
            values (@CustomerId, @CompanyName, @NIP)
        if @ClientType = 'I'
            insert into [Individual Customers] (CustomerId, FirstName,
                                                  LastName)
            values (@CustomerId, @FirstName, @LastName)
    end try
    begin catch
        declare @msg nvarchar(2000) = N'Błąd dodawania klienta do bazy: ' +
                                         ERROR_MESSAGE();
        THROW 60000, @msg, 1;
    end catch
end
go
```

2. AddDiscount - procedura dodająca zniżkę

```
CREATE procedure AddDiscount(
    @DiscountId int,
    @CustomerId int,
    @Active bit = 1
)
as
begin
    set nocount on
    declare @DiscountType as int
    set @DiscountType = 0
    declare @Z1 as int
        set @Z1 = (select Z1 from ConditionOrders)
    declare @K1 as int
        set @K1 = (select K1 from ConditionOrders)
    declare @K2 as int
        set @K2 = (select K2 from ConditionOrders)
    declare @K4 as int
        set @K4 = (select K4 from ConditionOrders)
    declare @Z3 as int
        set @Z3 = (select Z3 from ConditionOrders)

    if ( 1 not in (select DiscountType
        from [Discount Details]
        where CustomerId = @CustomerId)
    )
        begin
            if (select count(O.OrderId)
                from Orders O
                join OrderValue V on O.OrderId = V.OrderId
                where CustomerId = @CustomerId
                and value > @K1 ) >= @Z1
                begin
                    set @DiscountType = 1
                end
            end
        else
            begin
                if ( 2 not in (select DiscountType
                    from [Discount Details]
                    where CustomerId = @CustomerId)
                )
                    begin
                        if (select sum(value)
                            from OrderValue OV
                            join Orders O on OV.OrderId = O.OrderId
                            where CustomerId = @CustomerId) >= @K2
                            begin
                                set @DiscountId = 2
                            end
                        end
                    end
            end

            declare @OldPayment as int
            declare @newPayments as int

            set @newPayments = (select sum(MoneyPaid)
                                from Payments)
```

```

set @OldPayment = (@newPayments - (select top 1 MoneyPaid
                                   from Payments
                                   order by DatePayment desc ))

if ((@newPayments / @K4) - (@OldPayment / @K4) > 1)
begin
    if (select sum(value)
        from OrderValue OV
        join Orders O on OV.OrderId = O.OrderId
        where CustomerId = @CustomerId) >= @K4
    begin
        set @DiscountId = 4
    end
end

if (select count(OrderId)
    from Orders
    where CustomerId = @CustomerId) % @Z3 = 0
    and (select count(OrderId)
        from Orders
        where CustomerId = @CustomerId) > 0
begin
    set @DiscountType = 3
end

if (@DiscountType > 0 )
begin
    insert into [Discount Details] (DiscountId, CustomerId,
                                   DiscountType, Active)
    values (@DiscountId, @CustomerId, @DiscountType, @Active)
end

end
go

```

3. AddDiscountToOrder

```
CREATE PROCEDURE AddDiscountToOrder(  
    @DiscountID AS INT,  
    @OrderID AS INT  
)  
AS  
BEGIN  
    DECLARE @CustomerID AS INTEGER  
  
    SET @CustomerID = (SELECT CustomerID FROM ORders WHERE ORderID = @OrderID)  
    IF @DiscountID IN (SELECT DiscountId FROM [Discount Details] dd  
        JOIN Customers c ON c.CustomerId = dd.CustomerId)  
    BEGIN  
        IF (SELECT Active FROM [Discount Details]) = 1  
        BEGIN  
            Update Orders  
            SET DiscountID = @DiscountID  
            WHERE OrderId = @OrderID  
        END  
    END  
END
```

3. AddOrder - procedura dodająca zamówienie

```
CREATE procedure AddOrder(  
@CustomerId int,  
@PaidOrder bit = 0,  
@RealisationDate datetime= null,  
@StatusReservation bit = 0,  
@DiscountID int = null,  
@OrderDate datetime,  
@ReservationID int = null,  
@ReceiptID int = null  
)  
as  
begin  
    set nocount on  
    INSERT INTO Orders(CustomerId, OrderDate, StatusReservation,  
        RealisationDate, PaidOrder, ReservationId, ReceiptID,  
        DiscountID)  
    VALUES (@CustomerId, @OrderDate,  
        @StatusReservation,@RealisationDate,  
        @PaidOrder,@ReservationID,@ReceiptID,@DiscountID)  
end  
go
```


4. **AddToMenu** - procedura dodająca produkt do menu i sprawdzająca czy menu jest już poprawnie stworzone

```
CREATE PROCEDURE AddToMenu(  
    @menuID AS INTEGER  
    ,@productID AS INTEGER  
    ,@price AS MONEY  
)  
AS  
BEGIN  
    INSERT INTO [Menu Details] (MenuId, ProductID, PriceOfProduct)  
    VALUES (@menuID, @productID, @price)  
  
    DECLARE @valid AS BIT  
    EXEC @valid = dbo.MenuCorrect @id = @menuID  
    IF @valid = 1  
    BEGIN  
        UPDATE Menu  
        SET Valid = 1  
        WHERE Menu.MenuId = @menuID  
    END  
END  
go
```

5. AddToOrder - wstawianie produktów do danego zamówienia

```
CREATE PROCEDURE AddToOrder(
    @OrderID AS INT
    ,@ProductID AS INT
    ,@Quantity AS SMALLINT
)
AS
BEGIN

    IF @ProductID IN
        (SELECT ProductID FROM [Menu Details] md
        JOIN Menu m ON m.MenuId = md.MenuId
        WHERE (SELECT OrderDate FROM Orders WHERE OrderId = @OrderID) BETWEEN
m.StartedDate AND m.EndedDate)
    BEGIN

        IF @ProductID IN (SELECT ProductID FROM Products p
        JOIN Categories c ON c.CategoryId = p.CategoryId
        WHERE CategoryName = 'Egzotyczne')
        BEGIN
            DECLARE @neededspace AS INT
            DECLARE @realisationDate AS DATETIME
            SET @realisationDate = (SELECT RealisationDate FROM Orders WHERE
OrderID = @OrderID)
            SET @neededspace =
                CASE
                    WHEN datename(dw,dateadd(day,-1,@realisationDate))='Monday'
                        THEN 1
                    WHEN datename(dw,dateadd(day,-2,@realisationDate))='Monday'
                        THEN 2
                    WHEN datename(dw,dateadd(day,-3,@realisationDate))='Monday'
                        THEN 3
                    WHEN datename(dw,dateadd(day,-4,@realisationDate))='Monday'
                        THEN 4
                    WHEN datename(dw,dateadd(day,-5,@realisationDate))='Monday'
                        THEN 5
                    WHEN datename(dw,dateadd(day,-6,@realisationDate))='Monday'
                        THEN 6
                    WHEN datename(dw,dateadd(day,-7,@realisationDate))='Monday'
                        THEN 7
                END
            IF @realisationDate - (SELECT Orderdate FROM Orders WHERE OrderID =
@OrderID) >= @neededspace
            BEGIN
                INSERT INTO [Orders Details] (OrderId,ProductId,Quantity) VALUES
(@OrderID,@ProductID,@Quantity)
            END
        END
    ELSE
        BEGIN
            INSERT INTO [Orders Details] (OrderId,ProductId,Quantity) VALUES
(@OrderID,@ProductID,@Quantity)
        END
    END
END
```

6. AddToReservation - wstawianie zamówienia do rezerwacji

```
create procedure AddToReservation(  
    @StartReservation datetime,  
    @EndReservation datetime,  
    @Accepted bit = 0,  
    @OrderId int  
)  
as  
begin  
    set nocount on  
    declare @ResevationId int;  
  
    insert into Reservation(StartReservation, EndReservation, Accepted)  
    VALUES (@StartReservation, @EndReservation, @Accepted)  
  
    set @ResevationId = SCOPE_IDENTITY()  
  
    update Orders  
    set ReservationId = @ResevationId  
    where OrderId = @OrderId  
  
end  
go
```

7. RemoveFromMenu - procedura usuwająca produkt z danego menu

```
CREATE PROCEDURE RemoveFromMenu(  
    @menuID AS INTEGER  
    ,@productID AS INTEGER  
    ,@price AS MONEY  
)  
AS  
BEGIN  
    DELETE FROM [Menu Details]  
    WHERE [Menu Details] .MenuId = @menuID  
    AND ProductID = @productID  
  
    DECLARE @valid AS BIT  
    EXEC @valid = dbo.MenuCorrect @id = @menuID  
    IF @valid = 0  
    BEGIN  
        UPDATE Menu  
        SET Valid = 0  
        WHERE Menu.MenuId = @menuID  
    END  
END  
go
```

8. CheckForDiscounts - procedura sprawdzająca które zniżki należą się klientowi

```
CREATE PROCEDURE CheckForDiscounts(
    @CustomerId AS INT
)
AS
BEGIN
    declare @DiscountType as int
    declare @Z1 as int
        set @Z1 = (select Z1 from ConditionOrders)
    declare @K1 as int
        set @K1 = (select K1 from ConditionOrders)
    declare @K2 as int
        set @K2 = (select K2 from ConditionOrders)
    declare @K4 as int
        set @K4 = (select K4 from ConditionOrders)
    declare @Z3 as int
        set @Z3 = (select Z3 from ConditionOrders)
    DECLARE @Discount1 AS INT
    SET @Discount1 = 0
    DECLARE @Discount2 AS INT
    SET @Discount2 = 0
    DECLARE @Discount3 AS INT
    SET @Discount3 = 0
    DECLARE @Discount4 AS INT
    SET @Discount4 = 0

    if ( 1 not in (select DiscountType
        from [Discount Details]
        where CustomerId = @CustomerId)
    )
    if (select count(O.OrderId)
        from Orders O
        join OrderValue V on O.OrderId = V.OrderId
        where CustomerId = @CustomerId
        and value > @K1 ) >= @Z1
    begin
        set @Discount1 = 1
    end
    else
    if ( 2 not in (select DiscountType
        from [Discount Details]
        where CustomerId = @CustomerId)
    )

    if (select sum(value)
        from OrderValue OV
        join Orders O on OV.OrderId = O.OrderId
        where CustomerId = @CustomerId) >= @K2
    begin
        set @Discount2 = 1
    end

    declare @OldPayment as int
    declare @newPayments as int

    set @newPayments = (select sum(MoneyPaid)
        from Payments)

    set @OldPayment = (@newPayments - (select top 1 MoneyPaid
```

```

                                from Payments
                                order by DatePayment desc ))

if ((@newPayments / @K4) - (@OldPayment / @K4) > 1)

    if (select sum(value)
        from OrderValue OV
        join Orders O on OV.OrderId = O.OrderId
        where CustomerId = @CustomerId) >= @K4
    begin
        set @Discount4 = 1
    end
declare @counter AS INT
set @counter = (select count(OrderId)
                from Orders
                where CustomerId = @CustomerId)
if @counter % @Z3 = 0 AND
    @counter > 0
    begin
        set @Discount3 = 1
    end
SELECT *
FROM (VALUES (@CustomerId, @Discount1, @Discount2, @Discount3, @Discount4))
      t1 (klient, znizka1, znizka2, znizka3, znizka4)
END

```

Funkcje

1. **GetEarnMoneyInPeriod** - funkcja zwraca ilość pieniędzy w okresie od danej daty do danej daty

```
create function GetEarnMoneyInPeriod(@dateFrom datetime, @dateTo datetime)
    returns money
as begin return (
    select sum(MoneyPaid)
    from Payments
    where DatePayment > @dateFrom and DatePayment < @dateTo)
end
go
```

2. **GetMenuStatsByDate** - funkcja zwraca informację o menu i jego produktach z danego okresu czasu podanego jako argumenty

```
CREATE function GetMenuStatsByDate(@dateFrom datetime, @dateTo datetime)
returns table
return
select ProductName AS Dish, COUNT(md.MenuId) AS [times on menu],
       sum(PriceOfProduct) [SUM Price],
       avg(PriceOfProduct) [AVG Price],
       MAX(PriceOfProduct) [MAX Price],
       MIN(PriceOfProduct) [MIN Price]
from Products p
join [Menu Details] md on p.ProductId = md.ProductID
join Menu M on md.MenuId = M.MenuId
where m.StartedDate > @dateFrom and m.EndedDate < @dateTo
group by ProductName
go
```


3. **GetProductsBetweenDates** - funkcja zwraca produkty zamawiane w danym okresie czasu

```
create function GetProductsBetweenDates(@startdate datetime, @enddate
datetime)
returns table as
    return
    select ProductName
    from Products P
    inner join [Orders Details] [O D] on P.ProductId = [O D].ProductId
    inner join Orders O on O.OrderId = [O D].OrderId
    where OrderDate between @startdate and @enddate
go
```

4. **GetTop10Bestsellers** - funkcja zwraca top 10 najlepiej sprzedawanych się produktów w danym okresie czasu

```
CREATE function GetTop10Bestsellers(@startdate datetime, @enddate
datetime)
returns table as
return
select top 10 P.ProductID, ProductName, Sum(Quantity) as 'amount'
from Products P
inner join [Orders Details] [O D] on P.ProductId = [O D].ProductId
inner join Orders O on [O D].OrderId = O.OrderId
where OrderDate between @startdate and @enddate
group by ProductName, P.ProductId
order by COUNT([O D].ProductId) desc
go
```

5. **MenuCorrect** - funkcja zwraca 0/1 w zależności czy dane menu spełnia założenia o poprawnym menu

```
CREATE function MenuCorrect(@id int)
returns bit
AS
BEGIN
    declare @items int
    declare @minProductsToChange int
    declare @previousId int
    set @previousId = (select MenuId from menu where StartedDate =
                      (select max(StartedDate)
from menu where StartedDate < (select StartedDate from menu where MenuId =
@id)))
    set @items = (
        select count(*)
        from (
            select ProductId
            from [Menu Details] md
            where md.MenuId = @id

            except

            select ProductId
            from [Menu Details] md
            where md.MenuId = @previousId) out )

    set @minProductsToChange = (
        select count(*)
        from (
            select *
            from [Menu Details] md
            where md.MenuId = @previousId
            ) out ) / 2

    if @items >= @minProductsToChange
    begin
        return 1
    end
    return 0
END
go
```

Triggery

Baza danych nie potrzebuje triggerów.

Indeksy

1. Ustawienie indexu dla przedziału dat w menu

```
CREATE NONCLUSTERED INDEX DataRangeOfMenu  
ON Menu (StartDate, EndedDate);
```


Uprawnienia

CUSTOMER - klient korzystający z usługi restauracji

EMPLOYEE - zatrudniony pracownik, który pracuje w restauracji

ADMIN - właściciel restauracji

CUSTOMER:

 dbo.AddOrder	Grant: execute
 dbo.AddToOrder	Grant: execute
 dbo.AddToReservation	Grant: execute
 dbo.CurrentMenu	Grant: select
 dbo.GetTop10Bestsellers	Grant: select

EMPLOYEE:

 dbo.AddCustomer	Grant: execute
 dbo.AddDiscount	Grant: execute
 dbo.AddOrder	Grant: execute
 dbo.AddToOrder	Grant: execute
 dbo.AddToReservation	Grant: execute
 dbo.AmountOfTakeaways	Grant: execute
 dbo.CurrentMenu	Grant: select
 dbo.GetMenuStatsByDate	Grant: select
 dbo.GetProductsBetweenDates	Grant: select
 dbo.OrderInfo	Grant: select
 dbo.OrderValue	Grant: select
 dbo.OrdersToPay	Grant: select
 dbo.OwingClients	Grant: select

ADMIN:

Dostęp do wszystkiego co znajduje się w bazie danych

