

Bazy Danych

# **Dokumentacja projektu Restauracja**

---

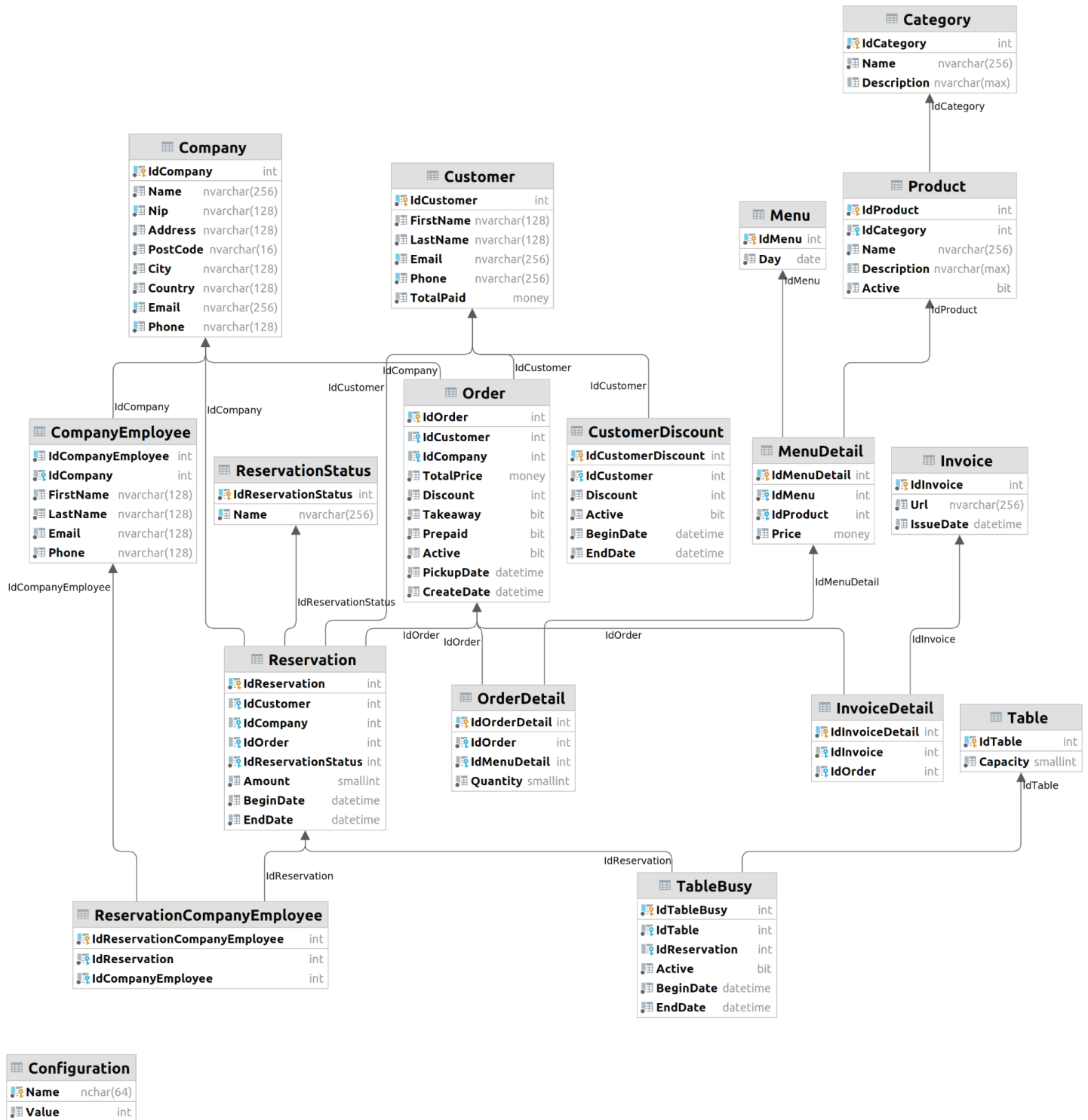
Adrian Chrobot

Mateusz Furga

Szymon Banyś

Grudzień, 2021

# Schemat bazy danych



## Definicje i opisy tabel

**Tabela Category** - przechowuje informacje o kategoriach produktów.

Kolumny:

1. IdCategory (int) - klucz główny
2. Name (nvarchar(256)) - nazwa kategorii
3. Description (nvarchar(max)) - opis kategorii

Warunki integralnościowe:

1. Name - pole unikalne

```
create table Category
(
    IdCategory int not null
        constraint Category_pk
            primary key nonclustered,
    Name nvarchar(256) not null,
    Description nvarchar(max)
)
```

**Tabela Company** - przechowuje informacje o firmach.

Kolumny:

1. IdCompany (int) - klucz główny
2. Name (nvarchar(256)) - nazwa firmy
3. Nip (nvarchar(128)) - NIP firmy
4. Address (nvarchar(128)) - adres siedziby firmy
5. PostCode (nvarchar(16)) - kod pocztowy siedziby firmy
6. City (nvarchar(128)) - miejscowość siedziby firmy
7. Country (nvarchar(128)) - kraj siedziby firmy
8. Email (nvarchar(256)) - email kontaktowy firmy
9. Phone (nvarchar(128)) - telefon kontaktowy firmy

Warunki integralnościowe:

1. Name - pole unikalne
2. Nip - pole unikalne
3. PostCode - string postaci XX-XXX, gdzie X to cyfra [0..9]

#### 4. Email - pole unikalne

```
create table Company
(
    IdCompany int not null
        constraint Company_pk
            primary key nonclustered,
    Name nvarchar(256) not null,
    Nip nvarchar(128) not null,
    Address nvarchar(128) not null,
    PostCode nvarchar(16) not null,
    City nvarchar(128) not null,
    Country nvarchar(128) not null,
    Email nvarchar(256) not null,
    Phone nvarchar(128) not null
)
```

**Tabela CompanyEmployee** - przechowuje informacje o pracownikach firm.

Kolumny:

1. IdCompanyEmployee (int) - klucz główny
2. IdCompany (int) - klucz obcy do tabeli Company
3. FirstName (nvarchar(128)) - Imię pracownika
4. LastName (nvarchar(128)) - Nazwisko pracownika
5. Email (nvarchar(128)) - Adres email kontaktowy pracownika
6. Phone (nvarchar(128)) - Telefon kontaktowy pracownika

Warunki integralnościowe:

1. Email - pole unikalne

```

create table CompanyEmployee
(
    IdCompanyEmployee int not null,
    IdCompany int not null
    constraint CompanyEmployee_Company_IdCompany_fk
    references Company,
    FirstName nvarchar(128) not null,
    LastName nvarchar(128) not null,
    Email nvarchar(128) not null,
    Phone nvarchar(128) not null
)

```

**Tabela Configuration** - przechowuje informacje o stałych konfiguracyjnych.

Kolumny:

1. Name (nvarchar(64)) - klucz główny, nazwa stałej konfiguracyjnej
2. Value (int) - wartość stałej konfiguracyjnej

```

create table Configuration
(
    Name nvarchar(64) not null
    constraint Configuration_pk
    primary key nonclustered,
    Value int not null
)

```

**Tabela Customer** - przechowuje informacje o klientach indywidualnych.

Kolumny:

1. IdCustomer (int) - klucz główny
2. FirstName (nvarchar(128)) - imię klienta indywidualnego
3. LastName (nvarchar(128)) - nazwisko klienta indywidualnego
4. Email (nvarchar(128)) - adres email kontaktowy klienta indywidualnego
5. Phone (nvarchar(128)) - telefon kontaktowy klienta indywidualnego
6. TotalPaid (money) - suma opłaconych zamówień liczona do uzyskania rabatu typu R2 oraz zerowana w momencie wykorzystania rabatu

Warunki integralnościowe:

1. Email - pole unikalne

2. TotalPaid - większe od lub równe zero

```
create table Customer
(
    IdCustomer int not null
        constraint Customer_pk
            primary key nonclustered,
    FirstName nvarchar(128) not null,
    LastName nvarchar(128) not null,
    Email nvarchar(256) not null,
    Phone nvarchar(256) not null,
    DiscountSum money not null
)
```

**Tabela CustomerDiscount** - przechowuje informacje o zniżkach przydzielonym klientom indywidualnym.

Kolumny:

1. IdCustomerDiscount (int) - klucz główny
2. IdCustomer (int) - Klucz obcy do tabeli Customer
3. Discount (int) - Rabat w procentach
4. Active (bit) - Czy dany rabat jest aktywny
5. BeginDate (datetime) - Początek ważności rabatu
6. EndDate (datetime) - Koniec ważności rabatu

Warunki integralnościowe:

1. Email - pole unikalne
2. TotalPaid - większe od lub równe zero

```
create table CustomerDiscount
(
    IdCustomerDiscount int not null
        constraint CustomerDiscount_pk
            primary key nonclustered,
    IdCustomer int not null
        constraint CustomerDiscount_Customer_IdCustomer_fk
            references Customer,
    Name nchar(64) not null
        constraint CustomerDiscount_Configuration_Name_fk
```

```
references Configuration,
BeginDate          datetime not null,
EndDate            datetime not null
)
```

**Tabela Invoice** - przechowuje wystawione faktury na klientów.

Kolumny:

1. IdInvoice (int) - klucz główny
2. Url (nvarchar(256)) - adres url do faktury
3. IssueDate (datetime) - data wystawienia

Warunki integralnościowe:

1. Url - pole unikalne

```
create table Invoice
(
    IdInvoice int not null
        constraint Invoice_pk
            primary key nonclustered,
    Url nvarchar(256) not null,
    IssueDate datetime not null
)
```

**Tabela InvoiceDetail** - przechowuje informacje o zamówieniach przypisanych do faktur.

Kolumny:

1. IdInvoiceDetail (int) - klucz główny
2. IdInvoice (int) - klucz obcy do faktury
3. IdOrder (int) - klucz obcy do zamówienia

```

create table InvoiceDetail
(
    IdInvoiceDetail int not null
        constraint InvoiceDetail_pk
            primary key nonclustered,
    IdInvoice int not null
        constraint InvoiceDetail_Invoice_IdInvoice_fk
            references Invoice,
    IdOrder int not null
        constraint InvoiceDetail_Order_IdOrder_fk
            references [Order]
)

```

**Tabela Menu** - wyróżnia dzień na jaki menu jest określone.

Kolumny:

1. IdMenu (int) - klucz główny
2. Day (date) - data menu

Warunki integralnościowe:

1. Day - pole unikalne

```

create table Menu
(
    IdMenu int not null
        constraint Menu_pk
            primary key nonclustered,
    Day date not null
)

```

**Tabela MenuDetail** - przydziela produkty do menu na dany dzień i określa ich cenę w danym menu.

Kolumny:

1. IdMenuDetail (int) - klucz główny
2. IdMenu (int) - klucz obcy do menu na dany dzień
3. IdProduct (int) - klucz obcy do produktu
4. Price (money) - cena za produkt w danym menu

Warunki integralnościowe:

1. Price - pole o dodatniej wartości bądź równej 0.



```
create table InvoiceDetail
(
    IdInvoiceDetail int not null
        constraint InvoiceDetail_pk
            primary key nonclustered,
    IdInvoice int not null
        constraint InvoiceDetail_Invoice_IdInvoice_fk
            references Invoice,
    IdOrder int not null
        constraint InvoiceDetail_Order_IdOrder_fk
            references [Order]
)
```

**Tabela Order** - przechowuje zamówienia wraz z danymi.

Kolumny:

1. IdOrder (int) - klucz główny
2. IdCustomer (int) - klucz obcy do klienta indywidualnego
3. IdCompany (int) - klucz obcy do firmy
4. TotalPrice (money) - całkowita cena za zamówienie
5. Discount (int) - zniżka
6. Takeaway (bit) - oznacza czy zamówienie jest na wynos
7. Prepaid (bit) - oznacza czy zamówienie jest opłacone
8. Active (bit) - oznacza zamówienie zaplanowane z zamiarem wykonania lub wykonywane
9. PickupDate (datetime) - data odebrania zamówienia
10. CreateDate (datetime) - data stworzenia zamówienia

Warunki integralnościowe:

1. TotalPrice - pole o dodatniej wartości lub równej 0
2. Discount - pole o dodatniej wartości
3. PickupDate - data musi być późniejsza niż data dla CreateDate

```

create table [Order]
(
    IdOrder      int      not null
        constraint Order_pk
            primary key nonclustered,
    IdCustomer   int
        constraint Order_Customer_IdCustomer_fk
            references Customer,
    IdCompany    int
        constraint Order_Company_IdCompany_fk
            references Company,
    TotalPrice   money    not null,
    Discount     int      not null,
    Takeaway     bit      not null,
    Prepaid      bit      not null,
    Active       bit      not null,
    PickupDate   datetime not null,
    CreateDate   datetime not null
)

```

**Tabela OrderDetail** - tabela powiązująca dane zamówienie z produktami do tego zamówienia wraz z ich ilością.

Kolumny:

1. IdOrderDetail (int) - klucz główny
2. IdOrder (int) - klucz obcy do zamówienia
3. IdMenuDetail (int) - klucz obcy do menu z produktem
4. Quantity (smallint) - pole określające ilość produktu w zamówieniu

Warunki integralnościowe:

1. Quantity - pole o dodatniej wartości

```
create table OrderDetail
(
    IdOrderDetail int not null
        constraint OrderDetail_pk
            primary key nonclustered,
    IdOrder int not null
        constraint OrderDetail_Order_IdOrder_fk
            references [Order]
            on update cascade on delete cascade,
    IdMenuDetail int not null
        constraint OrderDetail_MenuDetail_IdMenuDetail_fk
            references MenuDetail,
    Quantity smallint not null
)
```

**Tabela Product** - przechowuje nazwy produktów wraz z opisami.

Kolumny:

1. IdProduct (int) - klucz główny
2. IdCategory (int) - klucz obcy
3. Name (nvarchar(256)) - nazwa produktu
4. Description (nvarchar(max)) - opis produktu
5. Active (bit) - określenie czy produkt znajduje się w menu

Warunki integralnościowe:

1. Name - unikalne

```
create table Product
(
    IdProduct int not null
        constraint Product_pk
            primary key nonclustered,
    IdCategory int not null
        constraint Product_Category_IdCategory_fk
            references Category,
    Name nvarchar(256) not null,
    Description nvarchar(max)
)
```

**Tabela Reservation** - przechowuje dane rezerwacji.

Kolumny:

1. IdReservation (int) - klucz główny
2. IdCustomer (int) - klucz obcy
3. IdCompany (int) - klucz obcy
4. IdOrder (int) - klucz obcy
5. IdReservationStatus - klucz obcy
6. Amount (smallint) - ilość osób w rezerwacji
7. BeginDate (datetime) - początek rezerwacji
8. EndDate (datetime) - koniec rezerwacji

Warunki integralnościowe:

1. BeginDate - nie może być większe od EndDate
2. Amount - większa od zera

```
create table Reservation
(
    IdReservation      int      not null
        constraint Reservation_pk
            primary key nonclustered,
    IdCustomer         int
        constraint Reservation_Customer_IdCustomer_fk
            references Customer,
    IdCompany          int
        constraint Reservation_Company_IdCompany_fk
            references Company,
    IdOrder            int
        constraint Reservation_Order_IdOrder_fk
            references [Order],
    IdReservationStatus int      not null
        constraint Reservation_ReservationStatus_IdReservationStatus_fk
            references ReservationStatus,
    Amount             smallint not null,
    BeginDate          datetime not null,
    EndDate            datetime not null
)
```

**Tabela ReservationCompanyEmployee** - przechowuje informacje o rezerwacjach na firmę.

Kolumny:

1. IdReservationCompanyEmployee (int) - klucz główny
2. IdReservation (int) - klucz obcy
3. IdCompanyEmployee (int) - klucz obcy

```
create table ReservationCompanyEmployee
(
    IdReservationCompanyEmployee int not null
        constraint ReservationCompanyEmployee_pk
            primary key nonclustered,
    IdReservation int not null
        constraint ReservationCompanyEmployee_Reservation_IdReservation_fk
            references Reservation,
    IdCompanyEmployee int not null
        constraint
ReservationCompanyEmployee_CompanyEmployee_IdCompanyEmployee_fk
            references CompanyEmployee (IdCompanyEmployee)
)
```

**Tabela ReservationStatus** - przechowuje dane o statusach rezerwacji.

Kolumny:

1. IdReservationStatus (int) - klucz główny
2. Name (nvarchar(256)) - nazwa statusu rezerwacji

```
create table ReservationStatus
(
    IdReservationStatus int IDENTITY (1,1) not null
        constraint ReservationStatus_pk
            primary key nonclustered,
    Name nvarchar(256) not null
)
```

**Tabela Table** - przechowuje dane o stolikach:

Kolumny:

1. IdTable (int) - klucz główny
2. Capacity (smallint) - ilość miejsc przy stoliku

Warunki integralnościowe:

1. Capacity - większe od lub równe zero

```

create table [Table]
(
    IdTable int IDENTITY (1,1) not null
        constraint Table_pk
            primary key nonclustered,
    Capacity smallint not null
)

```

**Tabela TableBusy** - przechowuje dane o zajętych stolikach:

Kolumny:

1. IdTableBusy (int) - klucz główny
2. IdTable (int) - klucz obcy
3. IdReservation (int) - klucz obcy
4. Active (bit) - czy jest aktywna
5. BeginDate (datetime) - data zajęcia stolika
6. EndDate (datetime) - data zwolnienia stolika

Warunki integralnościowe:

1. BeginDate - mniejsze od EndDate

```

create table TableBusy
(
    IdTableBusy int IDENTITY (1,1) not null
        constraint TableBusy_pk
            primary key nonclustered,
    IdTable int not null
        constraint TableBusy_Table_IdTable_fk
            references [Table],
    IdReservation int
        constraint TableBusy_Reservation_IdReservation_fk
            references Reservation,
    Active bit not null,
    BeginDate datetime not null,
    EndDate datetime not null
)

```

# Dostępne funkcjonalności

PROCEDURY 32x

FUNKCJE 11x

WIDOKI 32x

TRIGGER 3x

## Zamówienia:

1. Utworzenie zamówienia złożonego przez klienta na wynos (pracownik)
2. Utworzenie zamówienia złożonego przez klienta na miejscu wraz z zajęciem stolika (bez wykonania wcześniejszej rezerwacji w systemie) (pracownik)
3. Anulowanie zamówienia na wynos (pracownik)
4. Dodanie pozycji do zamówienia
5. Dodanie rabatu R1 dla klienta indywidualnego według ustalonych reguł
6. Dodanie rabatu R2 dla klienta indywidualnego według ustalonych reguł
7. Wyświetlenie informacji o danym zamówieniu (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data) (klient, pracownik)
8. Wyświetlenie informacji o zamówieniach (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data) (klient, pracownik)
9. Wyświetlenie listy zrealizowanych zamówień (pracownik)
10. Wyświetlenie listy zamówień złożonych (pracownik)
11. Wyświetlenie listy zamówień złożonych przez danego klienta (pracownik)
12. Wyświetlenie sumarycznego kosztu danego zamówienia po rabatach (pracownik)
13. Wyświetlenie listy przyszłych zamówień do wykonania (pracownik, manager)
14. Wyświetlanie statystyk z zamówień z ostatniego tygodniowo (manager)
15. Wyświetlanie statystyk z zamówień z ostatniego miesiąca (manager)
16. Wyświetlanie statystyk z zamówień z ostatniego roku (manager)

## Rezerwacja:

1. Utworzenie rezerwacji wraz z zamówieniem (klient indywidualny)
2. Utworzenie rezerwacji (bez lub z zamówieniem) (firma)
3. Akceptacja danej rezerwacji wraz z zajęciem stolików (manager)
4. Odrzucenie danej rezerwacji (manager)
5. Anulowanie zaakceptowanej rezerwacji (klient, manager)
6. Wyświetlenie informacji o rezerwacjach firmowych, pracownikach imiennie oraz firmach (klient, manager)
7. Wyświetlenie listy przyszłych rezerwacji (manager)
8. Wyświetlenie listy zrealizowanych rezerwacji (manager)

9. Wyświetlanie statystyk z rezerwacji z obecnego tygodnia (manager)
10. Wyświetlanie statystyk z rezerwacji z obecnego miesiąca (manager)
11. Wyświetlanie statystyk z rezerwacji z obecnego roku (manager)

#### Menu:

1. Utworzenie menu z produktów oraz ustalenie ich cen na dany dzień (manager)
2. Wyświetlenie menu wraz z cenami oraz kategoriami produktów (klient, pracownik)
3. Wyświetlenie menu wraz z cenami oraz kategoriami produktów na dany dzień (klient, pracownik)
4. Wyświetlanie statystyk / raportu z menu z ostatniego tygodnia (manager)
5. Wyświetlanie statystyk / raportu z menu z ostatniego miesiąca (manager)
6. Sprawdzenia warunków rotacyjnych menu

#### Produkt:

1. Dodanie nowego produktu (manager)
2. Edycja danych produktu (nazwa, opis, kategoria) (manager)
3. Aktywacja / dezaktywacja danego produktu (manager)
4. Wyświetlenie informacji o produktach (klient, pracownik)

#### Kategoria produktu:

1. Utworzenie nowej kategorii produktu (manager)
2. Edycja danej kategorii produktu (nazwa, opis) (manager)
3. Wyświetlanie produktów wraz z kategoriami (pracownik)
4. Wyświetlanie produktów danej kategorii (pracownik)
5. Wyświetlanie wszystkich kategorii (pracownik)

#### Stoliki:

1. Dodanie nowego stolika do bazy (pracownik)
2. Zmiana ilości miejsca dla danego stolika (pracownik)
3. Oznaczenie danego stolika jako zajętego (manager)
4. Zwolnienie danego stolika (manager)
5. Wyświetlanie aktualnie dostępnych/zajętych stolików (manager)
6. Wyświetlanie stolików z ostatniego tygodnia (manager)
7. Wyświetlanie stolików z ostatniego miesiąca (manager)

#### Firma:

1. Dodanie nowej firmy (pracownik)
2. Edycja danych istniejącej firmy (pracownik)
3. Dodanie nowego pracownika do firmy (pracownik)
4. Edycja danych pracownika firmy (pracownik)
5. Wyświetlanie zapisanych firm (pracownik)
6. Wyświetlenie wszystkich pracowników (pracownik)



## 7. Wyświetlenie pracowników danej firmy (pracownik)

Klient indywidualny:

1. Dodanie nowego klienta indywidualnego (pracownik)
2. Edycja danych istniejącego klienta indywidualnego (pracownik)
3. Wyświetlanie informacji o klientach indywidualnych, ilości złożonych zamówień, sumarycznej kwoty za zamówienia, ilości wykonanych rezerwacji (pracownik)

Zniżki dla klientów indywidualnych:

1. Udzielenie zniżki krótkoterminowej dla klienta indywidualnego (pracownik)
2. Udzielenie stałej zniżki dla klienta indywidualnego (pracownik)
3. Anulowanie danej zniżki dla klienta indywidualnego (pracownik)
4. Wyświetlenie informacji o zniżkach (pracownik)
5. Wyświetlanie statystyk / raportu z zniżek dla klientów indywidualnych tygodniowo (pracownik)
6. Wyświetlanie statystyk / raportu z zniżek dla klientów indywidualnych miesięcznie (pracownik)
7. Wyświetlenie informacji o zniżkach dla danego klienta indywidualnego (pracownik)

Konfiguracja:

1. Edycja danej zmiennej konfiguracyjnej (np. edycja wartości zniżek) (manager)
2. Wyświetlanie wartości danej zmiennej konfiguracyjnej (manager)
3. Wyświetlanie wartości zmiennych konfiguracyjnych (manager)

Faktury:

1. Wystawienie faktury jednorazowej na zamówienie (manager)
2. Wystawienie faktury zbiorczej na zamówienia danej firmy (manager)
3. Wystawienie faktury zbiorczej na zamówienia danego klienta (manager)
4. Wyświetlanie faktury na dane zamówienie (manager)
5. Wyświetlanie listy wystawionych faktur na danego klienta indywidualnego (manager)
6. Wyświetlanie listy wystawionych faktur na daną firmę (manager)
7. Wyświetlanie listy wystawionych faktur klientów indywidualnych (manager)
8. Wyświetlanie listy wystawionych faktur na firmy (manager)
9. Wyświetlanie historii wystawionych faktur na wszystkich klientów (manager)

## Widoki

Nazwa: VReservationFuture

Opis: Wyświetlenie listy przyszłych rezerwacji wraz z informacjami o nich.

```
CREATE VIEW VReservation AS
SELECT
    r.IdReservation,
    CASE
        WHEN r.IdCompany IS NULL THEN 'Customer'
        ELSE 'Company'
    END AS 'ReservationFor',
    rs.Name AS 'ReservationStatus',
    r.IdOrder,
    r.Amount,
    r.BeginDate,
    r.EndDate
FROM Reservation r
LEFT JOIN Customer cu ON (cu.IdCustomer = r.IdCustomer)
LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
WHERE r.BeginDate >= GETDATE()
```

Nazwa: VProduct

Opis: Wyświetlenie informacji o produktach wraz z kategoriami.

```
CREATE VIEW VProduct AS
SELECT
    p.IdProduct AS 'IdProduct',
    p.Name AS 'ProductName',
    p.Description AS 'ProductDescription',
    c.Name AS 'CategoryName',
    c.Description AS 'CategoryDescription',
    p.Active AS 'ProductActive'
FROM Product p
LEFT JOIN Category c ON (c.IdCategory = p.IdProduct)
```

Nazwa: VCompletedReservation

Opis: Wyświetlenie listy zrealizowanych rezerwacji wraz z informacjami o nich.

```
CREATE VIEW VCompletedReservation AS
SELECT
    r.IdReservation,
    CASE
        WHEN r.IdCompany IS NULL THEN 'Customer'
        ELSE 'Company'
    END AS 'ReservationFor',
    rs.Name AS 'ReservationStatus',
    r.IdOrder,
    r.Amount,
    r.BeginDate,
    r.EndDate
FROM Reservation r
LEFT JOIN Customer cu ON (cu.IdCustomer = r.IdCustomer)
LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
WHERE r.IdReservationStatus = 1 AND r.EndDate <= GETDATE()
```

Nazwa: VMenu

Opis: Wyświetlenie menu wraz z cenami oraz kategoriami produktów na dany dzień.

```
CREATE VIEW VMenu AS
SELECT
    m.IdMenu AS 'IdMenu',
    m.Day AS 'Day',
    p.Name AS 'ProductName',
    p.Description AS 'ProductDescription',
    c.Name AS 'CategoryName',
    md.Price AS 'ProductPrice',
    p.Active AS 'ProductActive'
FROM Menu m
LEFT JOIN MenuDetail md ON (md.IdMenu = m.IdMenu)
LEFT JOIN Product p ON (p.IdProduct = md.IdProduct)
LEFT JOIN Category c ON (c.IdCategory = p.IdCategory)
```

Nazwa: VMenuStatisticsLastWeek

Opis: Wyświetla raport menu z obecnego tygodnia

```
CREATE VIEW VMenuStatisticsLastWeek AS
SELECT p.Name, m.Day,
(
    SELECT COUNT(*) FROM [Order] o1
    LEFT JOIN OrderDetail od1
        ON o1.IdOrder = od1.IdOrder
    LEFT JOIN MenuDetail md1
        ON od1.IdMenuDetail = md1.IdMenuDetail
    LEFT JOIN Product p1
        ON md1.IdProduct = p1.IdProduct
    WHERE DATEPART(DAYOFYEAR ,o1.CreateDate) = DATEPART(DAYOFYEAR,
m.Day) AND YEAR(o1.CreateDate) = YEAR(M.Day) AND
        p1.Name = p.Name
    ) AS 'HowManyTimesOrdered'
FROM Product p
INNER JOIN MenuDetail md on p.IdProduct = md.IdProduct
INNER JOIN Menu M on md.IdMenu = m.IdMenu
WHERE DATEPART(WEEK, GETDATE()) = DATEPART(WEEK, m.Day) AND
YEAR(GETDATE()) = YEAR(m.Day)
GROUP BY M.Day, p.Name
```

Nazwa: VMenuStatisticsLastMonth

Opis: Wyświetla raport menu z obecnego miesiąca

```
CREATE VIEW VMenuStatisticsLastMonth AS
SELECT p.Name, m.Day,
(
    SELECT COUNT(*) FROM [Order] o1
    LEFT JOIN OrderDetail od1
        ON o1.IdOrder = od1.IdOrder
    LEFT JOIN MenuDetail md1
        ON od1.IdMenuDetail = md1.IdMenuDetail
    LEFT JOIN Product p1
        ON md1.IdProduct = p1.IdProduct
    WHERE DATEPART(DAYOFYEAR ,o1.CreateDate) = DATEPART(DAYOFYEAR,
m.Day) AND YEAR(o1.CreateDate) = YEAR(M.Day) AND
        p1.Name = p.Name
    ) AS 'HowManyTimesOrdered'
FROM Product p
INNER JOIN MenuDetail md on p.IdProduct = md.IdProduct
```

```
INNER JOIN Menu M on md.IdMenu = m.IdMenu
WHERE MONTH(GETDATE()) = MONTH(m.Day) AND YEAR(GETDATE()) = YEAR(m.Day)
GROUP BY p.Name, M.Day
```

Nazwa: VReservationStatisticsLastWeek

Opis: Wyświetlanie statystyk z rezerwacji z obecnego tygodnia.

```
CREATE VIEW VReservationStatisticsLastWeek AS
SELECT
    DATEPART(WEEKDAY, x.BeginDate),
    x.ReservationFor,
    AVG(x.Amount) AS 'AverageAmount',
    COUNT(CASE WHEN x.IdReservationStatus = 1 THEN 1 END) AS 'Accepted',
    COUNT(*) AS 'Total'
FROM (
    SELECT
        r.IdReservation,
        CASE
            WHEN r.IdCompany IS NULL THEN 'Customer'
            ELSE 'Company'
        END AS 'ReservationFor',
        r.IdReservationStatus,
        rs.Name AS 'ReservationStatus',
        r.Amount,
        r.IdOrder,
        r.BeginDate
    FROM Reservation r
    LEFT JOIN Customer cu ON (cu.IdCustomer = r.IdCustomer)
    LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
    LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
) x
WHERE (x.BeginDate >= DATEADD(day, -7, GETDATE()) AND x.BeginDate <=
GETDATE())
GROUP BY DATEPART(WEEKDAY, x.BeginDate), x.ReservationFor
```

Nazwa: VReservationStatisticsLastMonth

Opis: Wyświetlanie statystyk z rezerwacji z obecnego miesiąca.

```
CREATE VIEW VReservationStatisticsLastMonth AS
SELECT
    YEAR(x.BeginDate) AS 'Year',
    MONTH(x.BeginDate) AS 'Month',
    x.ReservationFor,
    AVG(x.Amount) AS 'AverageAmount',
    COUNT(CASE WHEN x.IdReservationStatus = 1 THEN 1 END) AS 'Accepted',
    COUNT(*) AS 'Total'
FROM (
    SELECT
        r.IdReservation,
        CASE
            WHEN r.IdCompany IS NULL THEN 'Customer'
            ELSE 'Company'
        END AS 'ReservationFor',
        r.IdReservationStatus,
        rs.Name AS 'ReservationStatus',
        r.Amount,
        r.IdOrder,
        r.BeginDate
    FROM Reservation r
    LEFT JOIN Customer cu ON (cu.IdCustomer = r.IdCustomer)
    LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
    LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
) x
GROUP BY YEAR(x.BeginDate), MONTH(x.BeginDate), x.ReservationFor
HAVING YEAR(x.BeginDate) = YEAR(GETDATE()) AND MONTH(x.BeginDate) =
MONTH(GETDATE())
```

Nazwa: VReservationStatisticsLastYear

Opis: Wyświetlanie statystyk z rezerwacji z obecnego roku.

```
CREATE VIEW VReservationStatisticsLastYear AS
SELECT
    MONTH(x.BeginDate) AS 'Month',
    x.ReservationFor,
    AVG(x.Amount) AS 'AverageAmount',
    COUNT(CASE WHEN x.IdReservationStatus = 1 THEN 1 END) AS 'Accepted',
    COUNT(*) AS 'Total'
FROM (
    SELECT
        r.IdReservation,
        CASE
            WHEN r.IdCompany IS NULL THEN 'Customer'
            ELSE 'Company'
        END AS 'ReservationFor',
        r.IdReservationStatus,
        rs.Name AS 'ReservationStatus',
        r.Amount,
        r.IdOrder,
        r.BeginDate
    FROM Reservation r
    LEFT JOIN Customer cu ON (cu.IdCustomer = r.IdCustomer)
    LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
    LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
) x
WHERE YEAR(x.BeginDate) = YEAR(GETDATE())
GROUP BY MONTH(x.BeginDate), x.ReservationFor
```

Nazwa: VCustomer

Opis: Wyświetlanie informacji o klientach indywidualnych, ilości złożonych zamówień, sumarycznej kwoty za zamówienia, ilości wykonanych rezerwacji.

```
CREATE VIEW VCustomer AS
SELECT
    c.IdCustomer,
    c.FirstName,
    c.LastName,
    c.Email,
    c.Phone,
    c.TotalPaid,
    o.Sum AS 'OrderSum',
    o.Count AS 'OrderCount',
    r.Count AS 'ReservationCount'
FROM Customer c
LEFT JOIN (
    SELECT
        o.IdCustomer,
        SUM(o.TotalPrice - (o.Discount / 100 * o.TotalPrice)) AS 'Sum',
        COUNT(*) AS 'Count'
    FROM [Order] o
    WHERE o.IdCustomer IS NOT NULL
    GROUP BY o.IdCustomer
) o ON (o.IdCustomer = c.IdCustomer)
LEFT JOIN (
    SELECT
        r.IdCustomer,
        COUNT(*) AS 'Count'
    FROM Reservation r
    WHERE r.IdCustomer IS NOT NULL
    GROUP BY r.IdCustomer
) r ON (r.IdCustomer = c.IdCustomer)
```



Nazwa: VCompanyReservation

Opis: Wyświetlenie informacji o rezerwacjach firmowych, pracownikach imiennie oraz firmach.

```
CREATE VIEW VCompanyReservation AS
SELECT
    r.IdReservation AS 'IdReservation',
    rs.Name AS 'ReservationStatus',
    r.IdOrder AS 'IdOrder',
    r.Amount AS 'Amount',
    ce.FirstName AS 'EmployeeFirstName',
    ce.LastName AS 'EmployeeLastName',
    co.Name AS 'CompanyName',
    co.Nip AS 'CompanyNip',
    r.BeginDate AS 'BeginDate',
    r.EndDate AS 'EndDate'
FROM Reservation r
LEFT JOIN ReservationStatus rs ON (rs.IdReservationStatus =
r.IdReservationStatus)
LEFT JOIN ReservationCompanyEmployee rce ON (rce.IdReservation =
r.IdReservation)
LEFT JOIN Company co ON (co.IdCompany = r.IdCompany)
LEFT JOIN CompanyEmployee ce ON (ce.IdCompanyEmployee =
rce.IdCompanyEmployee)
WHERE r.IdCompany IS NOT NULL
```

Nazwa: VSavedCompanies

Opis: Tworzy widok zapisanych firm w bazie wraz z ich danymi.

```
CREATE VIEW VSavedCompanies AS
SELECT
    c.IdCompany AS "IdCompany",
    c.Name AS "Company Name",
    c.Nip AS "Company Nip",
    c.Address AS "Company Address",
    c.PostCode AS "Company Postcode",
```

```
c.City AS "Company City",
c.Country AS "Company Country",
c.Email AS "Company Email",
c.Phone AS "Company Phone Number"
FROM Company c;
```

Nazwa: VCompanyEmployees

Opis: Tworzy widok wszystkich pracowników firm korzystających z restauracji.

```
CREATE VIEW VCompanyEmployees AS
SELECT
    CE.IdCompanyEmployee AS "IdCompanyEmployee",
    CE.FirstName AS "First Name of Employee",
    CE.LastName AS "Last Name of Employee",
    CE.Email AS "Employee Email",
    CE.Phone AS "Employee Phone Number",
    C.Name AS "Company Name",
    C.Nip AS "Company Nip"
FROM CompanyEmployee CE
INNER JOIN Company C ON CE.IdCompany = C.IdCompany;
```

Nazwa: VCustomersDiscounts

Opis: Tworzy widok wszystkich zniżek dla klientów.

```
CREATE VIEW VCustomersDiscounts AS
SELECT
    CD.IdCustomerDiscount AS "IdCustomerDiscount",
    CD.Discount AS "Customer Discount",
    CD.Active AS "Discount Status",
    CD.BeginDate AS "Discount Begin Date",
    CD.EndDate AS "Discount End Date",
    C.FirstName AS "Customer First Name",
    C.LastName AS "Customer Last Name",
    C.TotalPaid AS "Customer Bill"
FROM CustomerDiscount CD
INNER JOIN Customer C ON CD.IdCustomer = C.IdCustomer;
```

Nazwa: VConfigurationVariables

Opis: Tworzy widok wszystkich zmiennych konfiguracyjnych.

```
CREATE VIEW VConfigurationVariables AS
SELECT
    C.Name as "Variable Name",
    C.Value AS "Variable value"
FROM Configuration C;
```

Nazwa: VAvailableTables

Opis: Tworzy widok z listą dostępnych stolików.

```
CREATE VIEW VAvailableTables AS
SELECT
    T.IdTable AS "IdTable",
    T.Capacity AS "Amount of seats"
FROM [Table] T
WHERE T.IdTable NOT IN
    (
        SELECT DISTINCT TB.IdTable
        FROM TableBusy TB
        INNER JOIN [Table] T ON T.IdTable = TB.IdTable
        INNER JOIN Reservation R2 ON R2.IdReservation = TB.IdReservation
        WHERE (
            R2.IdReservationStatus != 2 AND
            TB.Active != 'false' AND
            TB.BeginDate <= GETDATE() AND
            TB.EndDate >= GETDATE()
        )
    );
```

Nazwa: VBusyTable

Opis: Tworzy widok z listą zajętych stolików.

```

CREATE VIEW VBusyTable AS
SELECT DISTINCT
    TB.IdTable AS "IdTable",
    TB.Active AS "Table Status",
    TB.BeginDate AS "Begin of Table Reservation",
    TB.EndDate AS "End of Table Reservation",
    T.Capacity AS "Amount of Seats",
    R2.IdReservation AS "IdReservation",
    R2.IdCustomer AS "IdCustomer",
    R2.IdCompany AS "IdCompany",
    R2.IdOrder AS "IdOrder",
    RS.Name AS "Status of Reservation",
    R2.Amount AS "Reservation Amount"
FROM TableBusy TB
INNER JOIN [Table] T ON T.IdTable = TB.IdTable
INNER JOIN Reservation R2 ON R2.IdReservation = TB.IdReservation
INNER JOIN ReservationStatus RS ON R2.IdReservationStatus =
RS.IdReservationStatus
WHERE (
    R2.IdReservationStatus != 2 AND
    TB.Active != 'false' AND
    TB.BeginDate <= GETDATE() AND
    TB.EndDate >= GETDATE()
);

```

Nazwa: VInvoicesCustomers

Opis: Tworzy widok z wystawionymi fakturami dla klientów indywidualnych.

```

CREATE VIEW VInvoicesCustomers AS
SELECT
    I.IdInvoice AS "IdInvoice",
    I.Url AS "Address of Invoice",
    I.IssueDate AS "Issue Date of Invoice",
    C.FirstName AS "Customer First Name",
    C.LastName AS "Customer Last Name",
    C.Email AS "Customer Email",
    O.TotalPrice AS "Order Bill"
FROM Invoice I
INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice
INNER JOIN [Order] O ON O.IdOrder = ID.IdOrder

```

```
INNER JOIN Customer C ON O.IdCustomer = C.IdCustomer
WHERE (
    O.IdCustomer IS NOT NULL AND
    O.IdCompany IS NULL
);
```

Nazwa: VInvoicesCompanies

Opis: Tworzy widok z wystawionymi fakturami dla firm.

```
CREATE VIEW VInvoicesCompanies AS
SELECT
    I.IdInvoice AS "IdInvoice",
    I.Url AS "Address of Invoice",
    I.IssueDate AS "Issue Date of Invoice",
    C.Name AS "Company Name",
    C.Nip AS "Company Nip",
    C.Email AS "Company Email",
    O.TotalPrice AS "Order Bill"
FROM Invoice I
INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice
INNER JOIN [Order] O ON O.IdOrder = ID.IdOrder
INNER JOIN Company C ON O.IdCompany = C.IdCompany
WHERE (
    O.IdCustomer IS NULL AND
    O.IdCompany IS NOT NULL
);
```

Nazwa: VInvoices

Opis: Tworzy widok z wszystkimi wystawionymi fakturami.

```
CREATE VIEW VInvoices AS
SELECT
    I.IdInvoice AS "IdInvoice",
    I.Url AS "Address of Invoice",
    I.IssueDate AS "Issue Date of Invoice",
    O.IdCompany AS "CompanyID",
    O.IdCustomer AS "CustomerID",
    O.TotalPrice AS "Order Bill"
FROM Invoice I
INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice
```

```
INNER JOIN [Order] O ON O.IdOrder = ID.IdOrder;
```

Nazwa: VOrdersDetails

Opis: Wyświetlenie informacji o zamówieniach (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data) (klient, pracownik)

```
CREATE VIEW VOrdersDetails AS
SELECT
    o.IdOrder,
    p.Name,
    od.Quantity,
    md.Price,
    o.Discount,
    o.CreateDate,
    o.PickupDate,
    dbo.OrderValue(o.IdOrder) AS 'OrderValue'
FROM [Order] o
INNER JOIN OrderDetail od
ON o.IdOrder = od.IdOrder
INNER JOIN MenuDetail md
ON od.IdMenuDetail = md.IdMenuDetail
INNER JOIN Product p
ON md.IdProduct = p.IdProduct
```

Nazwa: VCompletedOrders

Opis: Wyświetlenie listy zrealizowanych zamówień

```
CREATE VIEW VCompletedOrders as
SELECT IdOrder FROM [Order]
WHERE PickupDate < GETDATE()
AND Active = 0
```

Nazwa: VOrdersByCustomer

Opis: Wyświetlanie zamówień złożonych przez poszczególnych klientów

```
CREATE VIEW VOrdersByCustomer AS
SELECT IdOrder,
```

```

CASE
    WHEN IdCompany IS NULL THEN IdCustomer
    ELSE IdCompany
END AS 'CustomerID',
CASE
    WHEN IdCompany IS NULL THEN 'Private Customer'
    ELSE 'Company'
END AS 'Company or private customer'
FROM [Order]

```

Nazwa: VFutureOrders

Opis: Wyświetlanie przyszłych zamówień do wykonania

```

CREATE VIEW VFutureOrders AS
SELECT IdOrder
FROM [Order]
WHERE Active = 1 AND PickupDate > GETDATE()

```

Nazwa: VOrderStatisticsLastWeek

Opis: Wyświetlanie statystyk zamówień z obecnego tygodnia

```

CREATE VIEW VOrderStatisticsLastWeek AS
SELECT
    YEAR(x.CreateDate) AS 'YEAR',
    DATEPART(WEEK,X.CreateDate) AS 'WEEK',
    x.OrderFor,
    AVG(x.TotalPrice) AS 'AveragePrice',
    COUNT(CASE WHEN x.Active = 0 THEN 0 END) AS 'Completed',
    COUNT(*) AS 'Total',
    (
        SELECT TOP 1 p.Name FROM [Order] o1
        LEFT JOIN OrderDetail od1
            ON o1.IdOrder = od1.IdOrder
        LEFT JOIN MenuDetail md1
            ON od1.IdMenuDetail = md1.IdMenuDetail
        LEFT JOIN Product p
            ON md1.IdProduct = p.IdProduct
        WHERE DATEPART(WEEK,GETDATE()) = DATEPART(WEEK,X.CreateDate) AND
        YEAR(o1.CreateDate) = YEAR(x.CreateDate)
        GROUP BY p.Name
        ORDER BY COUNT(*) DESC
    )

```

```

) AS 'TopProduct'
FROM (
    SELECT
        o.IdOrder,
        CASE
            WHEN o.IdCompany IS NULL THEN 'Customer'
            ELSE 'Company'
        END AS 'OrderFor',
        o.Active,
        o.CreateDate,
        o.TotalPrice
    FROM [Order] o
    LEFT JOIN OrderDetail od
    ON o.IdOrder = od.IdOrder
) x
GROUP BY YEAR(x.CreateDate), DATEPART(WEEK,X.CreateDate), x.OrderFor
HAVING YEAR(x.CreateDate) = YEAR(GETDATE()) AND DATEPART(WEEK,GETDATE())
= DATEPART(WEEK,X.CreateDate)

```

Nazwa: VOrderStatisticsLastMonth

Opis: Wyświetlanie statystyk zamówień z obecnego miesiąca

```

CREATE VIEW VOrderStatisticsLastMonth AS
SELECT
    YEAR(x.CreateDate) AS 'YEAR',
    MONTH(x.CreateDate) AS 'MONTH',
    x.OrderFor,
    AVG(x.TotalPrice) AS 'AveragePrice',
    COUNT(CASE WHEN x.Active = 0 THEN 0 END) AS 'Completed',
    COUNT(*) AS 'Total',
    (
        SELECT TOP 1 p.Name FROM [Order] o1
        LEFT JOIN OrderDetail od1
        ON o1.IdOrder = od1.IdOrder
        LEFT JOIN MenuDetail md1
        ON od1.IdMenuDetail = md1.IdMenuDetail
        LEFT JOIN Product p
        ON md1.IdProduct = p.IdProduct
        WHERE MONTH(o1.CreateDate) = MONTH(x.CreateDate) AND
        YEAR(o1.CreateDate) = YEAR(x.CreateDate)
    )

```



```

        GROUP BY p.Name
        ORDER BY COUNT(*) DESC
    ) AS 'TopProduct'
FROM (
    SELECT
        o.IdOrder,
        CASE
            WHEN o.IdCompany IS NULL THEN 'Customer'
            ELSE 'Company'
        END AS 'OrderFor',
        o.Active,
        o.CreateDate,
        o.TotalPrice
    FROM [Order] o
    LEFT JOIN OrderDetail od
    ON o.IdOrder = od.IdOrder
    ) x
GROUP BY YEAR(x.CreateDate), MONTH(x.CreateDate), x.OrderFor
HAVING YEAR(x.CreateDate) = YEAR(GETDATE()) AND MONTH(x.CreateDate) =
MONTH(GETDATE())

```

Nazwa: VOrderStatisticsLastYear

Opis: Wyświetlanie statystyk zamówień z obecnego roku

```

CREATE VIEW VOrderStatisticsLastYear AS
SELECT
    MONTH(x.CreateDate) AS 'MONTH',
    x.OrderFor,
    AVG(x.TotalPrice) AS 'AveragePrice',
    COUNT(CASE WHEN x.Active = 0 THEN 0 END) AS 'Completed',
    COUNT(*) AS 'Total',
    (
        SELECT TOP 1 p.Name FROM [Order] o1
        LEFT JOIN OrderDetail od1
        ON o1.IdOrder = od1.IdOrder
        LEFT JOIN MenuDetail md1
        ON od1.IdMenuDetail = md1.IdMenuDetail
        LEFT JOIN Product p
        ON md1.IdProduct = p.IdProduct
        WHERE MONTH(o1.CreateDate) = MONTH(x.CreateDate) AND

```

```

YEAR(o1.CreateDate) = 2021
    GROUP BY p.Name
    ORDER BY COUNT(*) DESC
) AS 'TopProduct'
FROM (
    SELECT
        o.IdOrder,
    CASE
        WHEN o.IdCompany IS NULL THEN 'Customer'
        ELSE 'Company'
    END AS 'OrderFor',
    o.Active,
    o.CreateDate,
    o.TotalPrice
    FROM [Order] o
    LEFT JOIN OrderDetail od
    ON o.IdOrder = od.IdOrder
) x
WHERE YEAR(x.CreateDate) = 2021
GROUP BY MONTH(x.CreateDate), x.OrderFor

```

Nazwa: VCategoryProducts

Opis: Wyświetlanie produktów po kategoriach

```

CREATE VIEW VCategoryProducts AS
SELECT
    p.Name AS 'ProductName',
    c.Name AS 'CategoryName'
FROM Product p
INNER JOIN Category C ON C.IdCategory = P.IdCategory

```

Nazwa: VCategory:

Opis: Wyświetlanie wszystkich kategorii produktów z opisami

```

CREATE VIEW VCategory AS
SELECT
    IdCategory as 'CategoryID',
    Name as 'CategoryName',
    Description as 'CategoryDescription'
FROM Category

```

Nazwa: VTableWeek

Opis: Wyświetlenie stolików z ostatniego tygodnia.

```
CREATE VIEW VTableWeek AS
SELECT DISTINCT
    TB.IdTable AS "IdTable",
    TB.Active AS "Table Status",
    TB.BeginDate AS "Begin of Table Reservation",
    TB.EndDate AS "End of Table Reservation",
    T.Capacity AS "Amount of Seats",
    R2.IdReservation AS "IdReservation",
    R2.IdCustomer AS "IdCustomer",
    R2.IdCompany AS "IdCompany",
    R2.IdOrder AS "IdOrder",
    RS.Name AS "Status of Reservation",
    R2.Amount AS "Reservation Amount"
FROM TableBusy TB
INNER JOIN [Table] T ON T.IdTable = TB.IdTable
INNER JOIN Reservation R2 ON R2.IdReservation = TB.IdReservation
INNER JOIN ReservationStatus RS ON R2.IdReservationStatus =
RS.IdReservationStatus
WHERE (
    TB.BeginDate <= GETDATE() AND
    (DATEDIFF(WEEK ,TB.BeginDate, GETDATE()) < 1)
);
```

Nazwa: VTableMonth

Opis: Wyświetlenie stolików z ostatniego miesiąca.

```
CREATE VIEW VTableMonth AS
SELECT DISTINCT
    TB.IdTable AS "IdTable",
    TB.Active AS "Table Status",
    TB.BeginDate AS "Begin of Table Reservation",
    TB.EndDate AS "End of Table Reservation",
    T.Capacity AS "Amount of Seats",
    R2.IdReservation AS "IdReservation",
```

```

R2.IdCustomer AS "IdCustomer",
R2.IdCompany AS "IdCompany",
R2.IdOrder AS "IdOrder",
RS.Name AS "Status of Reservation",
R2.Amount AS "Reservation Amount"
FROM TableBusy TB
INNER JOIN [Table] T ON T.IdTable = TB.IdTable
INNER JOIN Reservation R2 ON R2.IdReservation = TB.IdReservation
INNER JOIN ReservationStatus RS ON R2.IdReservationStatus =
RS.IdReservationStatus
WHERE (
    TB.BeginDate <= GETDATE() AND
    (DATEDIFF(MONTH ,TB.BeginDate, GETDATE()) < 1)
);

```

Nazwa: VCustomersDiscountsLastWeek

Opis: Wyświetlanie statystyk z zniżek dla klientów indywidualnych z ostatniego tygodnia.

```

CREATE VIEW VCustomersDiscountsLastWeek AS
SELECT
    CD.IdCustomerDiscount AS "IdCustomerDiscount",
    CD.Discount AS "Customer Discount",
    CD.Active AS "Discount Status",
    CD.BeginDate AS "Discount Begin Date",
    CD.EndDate AS "Discount End Date",
    C.FirstName AS "Customer First Name",
    C.LastName AS "Customer Last Name",
    C.TotalPaid AS "Customer Bill"
FROM CustomerDiscount CD
INNER JOIN Customer C ON CD.IdCustomer = C.IdCustomer
WHERE (
    CD.BeginDate <= GETDATE() AND
    (DATEDIFF(WEEK,CD.BeginDate, GETDATE()) < 1)
);

```

Nazwa: VCustomersDiscountsLastMonth

Opis: Wyświetlanie statystyk z zniżek dla klientów indywidualnych z ostatniego miesiąca.

```

CREATE VIEW VCustomersDiscountsLastMonth AS
SELECT
    CD.IdCustomerDiscount AS "IdCustomerDiscount",
    CD.Discount AS "Customer Discount",

```

```
CD.Active AS "Discount Status",
CD.BeginDate AS "Discount Begin Date",
CD.EndDate AS "Discount End Date",
C.FirstName AS "Customer First Name",
C.LastName AS "Customer Last Name",
C.TotalPaid AS "Customer Bill"
FROM CustomerDiscount CD
INNER JOIN Customer C ON CD.IdCustomer = C.IdCustomer
WHERE (
    CD.BeginDate <= GETDATE() AND
    (DATEDIFF(MONTH,CD.BeginDate, GETDATE()) < 1)
);
```

## Funkcje

Nazwa: FGetEmployeesByCompany

Opis: Wyświetla pracowników danej firmy.

```
CREATE FUNCTION FGetEmployeesByCompany
(
    @IdCompany int
)
RETURNS TABLE AS RETURN
(
    SELECT
        ce.FirstName AS 'FirstName',
        ce.LastName AS 'LastName',
        ce.Email AS 'Email',
        ce.Phone AS 'Phone'
    FROM CompanyEmployee ce
    WHERE ce.IdCompany = @IdCompany
)
```

Nazwa: FGetProductsByCategory

Opis: Wyświetla produkty należące do danej kategorii.

```
CREATE FUNCTION FGetProductsByCategory
(
    @IdCategory int
)
RETURNS TABLE AS RETURN
(
    SELECT
        p.IdProduct AS 'IdProduct',
        p.Name AS 'ProductName',
        p.Description AS 'ProductDescription',
        c.Name AS 'CategoryName',
        c.Description AS 'CategoryDescription',
        p.Active AS 'ProductActive'
    FROM Product p
    LEFT JOIN Category c ON (c.IdCategory = p.IdProduct)
    WHERE c.IdCategory = @IdCategory
)
```

Nazwa: FGetMenuProductsByDay

Opis: Wyświetla produkty menu z cenami na dany dzień.

```
CREATE FUNCTION FGetMenuProductsByDay
(
    @Date date
)
RETURNS TABLE AS RETURN
(
    SELECT m.IdMenu AS 'IdMenu',
           m.Day AS 'Day',
           p.Name AS 'ProductName',
           p.Description AS 'ProductDescription',
           c.Name AS 'CategoryName',
           md.Price AS 'ProductPrice',
           p.Active AS 'ProductActive'
    FROM Menu m
    LEFT JOIN MenuDetail md ON (md.IdMenu = m.IdMenu)
    LEFT JOIN Product p ON (p.IdProduct = md.IdProduct)
    LEFT JOIN Category c ON (c.IdCategory = p.IdCategory)
    WHERE m.Day = @Date
)
```

Nazwa: FGetCompanyInvoices

Opis: Funkcja zwraca listę faktur na daną firmę od id równym @ComapnyId.

```
CREATE FUNCTION FGetCompanyInvoices
(
    @ComapnyId int
)
RETURNS TABLE AS RETURN
(
    SELECT
        I.Url AS 'InvoiceUrl',
        I.IssueDate AS 'DateOfInvoice',
        O.IdOrder AS 'IdOrder',
        O.IdCompany AS 'IdCompany'
    FROM Invoice I
    INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice
    INNER JOIN [Order] O ON O.IdOrder = ID.IdOrder
    WHERE IdCompany = @ComapnyId
)
```

```
)  
GO
```

Nazwa: FGetCustomerInvoices

Opis: Funkcja zwraca listę faktur na danego klienta indywidualnego od id równym @CustomerId

```
CREATE FUNCTION FGetCustomerInvoices  
(  
    @CustomerId int  
)  
RETURNS TABLE AS RETURN  
(  
    SELECT  
        I.Url AS 'InvoiceUrl',  
        I.IssueDate AS 'DateOfInvoice',  
        O.IdOrder AS 'IdOrder',  
        O.IdCustomer AS 'IdCustomer'  
    FROM Invoice I  
    INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice  
    INNER JOIN [Order] O ON O.IdOrder = ID.IdOrder  
    WHERE O.IdCustomer = @CustomerId  
)  
GO
```

Nazwa: FGetInvoiceOfOrder

Opis: Funkcja zwraca dane faktury na dane zamówienie.

```
CREATE FUNCTION FGetInvoiceOfOrder  
(  
    @OrderId int  
)  
RETURNS TABLE AS RETURN  
(  
    SELECT  
        I.Url AS 'InvoiceUrl',  
        I.IssueDate AS 'DateOfInvoice'  
    FROM Invoice I  
    INNER JOIN InvoiceDetail ID ON I.IdInvoice = ID.IdInvoice
```



```
WHERE ID.IdOrder = @OrderId
)
GO
```

Nazwa: FGetValueByConfiguratedVariable

Opis: Funkcja zwraca wartość zmiennej konfiguracyjnej o nazwie @VariableName

```
CREATE FUNCTION FGetValueByConfiguratedVariable
(
    @VariableName nchar(64)
)
RETURNS INT AS
BEGIN
    DECLARE @returnValue INT;
    SELECT @returnValue =
    (
        SELECT
            C.Value
        FROM Configuration C
        WHERE C.Name = @VariableName
    );
    RETURN @returnValue;
END
```

Nazwa: FGetDiscountByCustomer

Opis: Funkcja zwraca tabelę z zniżkami dla danego klienta o id: CustomerId.

```
CREATE FUNCTION FGetDiscountByCustomer
(
    @CustomerID int
)
RETURNS TABLE AS RETURN
(
    SELECT
        CD.Discount AS 'CustomerDiscount',
        CD.Active AS 'DiscountStatus',
        CD.BeginDate AS 'DiscountBeginDate',
        CD.EndDate AS 'DiscountEndDate'
    FROM CustomerDiscount CD
    WHERE IdCustomer = @CustomerID
)
```

Nazwa: FOrderValue

Opis: Funkcja zwraca wartość zamówienia

```
create function FOrderValue(@Order_ID INT)
returns float
as
begin
return (
    select (sum(md1.Price * od1.Quantity) *
        (100 - (select top 1 Discount from [Order]
            o where o.IdOrder = o1.IdOrder))) / 100
    from [Order] o1
    inner join OrderDetail od1
    on o1.IdOrder = od1.IdOrder
    inner join MenuDetail md1
    on od1.IdMenuDetail = md1.IdMenuDetail
    where o1.IdOrder = @Order_ID
    group by o1.IdOrder
    )
end
```

Nazwa: FOrderByClient

Opis: Funkcja zwraca listę zamówień złożonych przez danego klienta indywidualnego bądź firmę

```
CREATE FUNCTION FOrderByClient
(
    @IdCustomer int = NULL,
    @IdCompany int = NULL
)
RETURNS TABLE AS RETURN
(
    select o.IdOrder,
        CASE
            WHEN @IdCompany IS NULL THEN @IdCustomer
            ELSE @IdCompany
        END AS 'CustomerID',
        CASE
            WHEN IdCompany IS NULL THEN 'Private Customer'
```

```
        ELSE 'Company'
    END AS 'Company or private customer'
    FROM [Order] o where @IdCustomer = o.IdCustomer or @IdCompany =
o.IdCompany
)
```

Nazwa: FOrderDetail

Opis: Wyświetlenie szczegółów o danym zamówieniu

```
CREATE FUNCTION FOrderDetail
(
    @IdOrder int
)
RETURNS TABLE AS RETURN
(
    select o.IdOrder, p.Name, od.Quantity, md.Price,
o.Discount,o.CreateDate, o.PickupDate, dbo.FOrderValue(o.IdOrder) as
'OrderValue'
    from [Order] o
    inner join OrderDetail od
    on o.IdOrder = od.IdOrder
    inner join MenuDetail md
    on od.IdMenuDetail = md.IdMenuDetail
    inner join Product p
    on md.IdProduct = p.IdProduct
    WHERE o.IdOrder = @IdOrder
)
```

## Procedure

Nazwa: PCreateProductCategory

Opis: Tworzy nową kategorię produktu.

```
CREATE PROCEDURE PCreateProductCategory
(
    @Name nvarchar(256),
    @Description nvarchar(max)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (SELECT * FROM Category WHERE Name = @Name) BEGIN ; THROW
52000, 'Category with given name already exists.', 1 END
        INSERT INTO Category (Name, Description) VALUES (@Name,
@Description)
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot create the category. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END
```

Nazwa: PUpdateProductCategory

Opis: Edytuje daną kategorię produktu.

```
CREATE PROCEDURE PUpdateProductCategory
(
    @IdCategory int,
    @Name nvarchar(256),
    @Description nvarchar(max)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Category WHERE IdCategory =
```

```

@IdCategory) BEGIN ; THROW 52000, 'Category with given id does not exist.',
1 END
    UPDATE Category SET Name = @Name, Description = @Description WHERE
IdCategory = @IdCategory
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot update the category. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateProduct

Opis: Tworzy nowy produkt.

```

CREATE PROCEDURE PCreateProduct
(
    @IdCategory int,
    @Name nvarchar(256),
    @Description nvarchar(max),
    @Active bit
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (SELECT * FROM Product WHERE Name = @Name) BEGIN ; THROW
52000, 'Product with given name already exists.', 1 END
        INSERT INTO Product (IdCategory, Name, Description, Active) VALUES
(@IdCategory, @Name, @Description, @Active)
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot create the product. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PUpdateProduct

Opis: Edytuje dany produkt.

```

CREATE PROCEDURE PUpdateProduct
(

```

```

        @IdProduct int,
        @IdCategory int,
        @Name nvarchar(256),
        @Description nvarchar(max),
        @Active bit
    )
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Product WHERE IdProduct = @IdProduct)
        BEGIN ; THROW 52000, 'Product with given id does not exist.', 1 END
        UPDATE Product SET IdCategory = @IdCategory, Name = @Name,
        Description = @Description, Active = @Active WHERE IdProduct = @IdProduct
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot update the product. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PUpdateProductStatus

Opis: Zmienia status danego produktu na aktywny lub nieaktywny.

```

CREATE PROCEDURE PUpdateProductStatus
(
    @IdProduct int,
    @Active bit
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Product WHERE IdProduct = @IdProduct)
        BEGIN ; THROW 52000, 'Product with given id does not exist.', 1 END
        UPDATE Product SET Active = @Active WHERE IdProduct = @IdProduct
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot update the product. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

```
END CATCH
END
```

Nazwa: PCreateTable

Opis: Tworzy nowy stolik o danej pojemności.

```
CREATE PROCEDURE PCreateTable
(
    @Capacity int
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO [Table] (Capacity) VALUES (@Capacity)
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot create the table. ERROR: '
+ ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END
```

Nazwa: PUpdateTableCapacity

Opis: Zmienia pojemność danego stolika.

```
CREATE PROCEDURE PUpdateTableCapacity
(
    @IdTable int,
    @Capacity int
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM [Table] WHERE IdTable = @IdTable)
        BEGIN ; THROW 52000, 'Table with given id does not exist.', 1 END
        UPDATE [Table] SET Capacity = @Capacity WHERE IdTable = @IdTable
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot update the table. ERROR: '
+ ERROR_MESSAGE();
```

```
        THROW 52000, @error, 1
    END CATCH
END
```

Nazwa: PMarkTableBusy

Opis: Oznacza dany stolik jako zajęty na zadany przedział czasu.

```
CREATE PROCEDURE PMarkTableBusy
(
    @IdTable int,
    @BeginDate datetime,
    @EndDate datetime
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (
            SELECT * FROM TableBusy WHERE IdTable = @IdTable
                                AND Active = 1
                                AND BeginDate <= @EndDate
                                AND EndDate >= @BeginDate
        ) BEGIN ; THROW 52000, 'Table is already busy.', 1 END
        INSERT INTO TableBusy (IdTable, IdReservation, Active, BeginDate,
EndDate) VALUES
                                (@IdTable, NULL, 1, @BeginDate, @EndDate)

    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot occupy the table. ERROR: '
+ ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END
```

Nazwa: PFreeTable

Opis: Zwalnia obecnie zajmowany stolik.

```
CREATE PROCEDURE PFreeTable
(
    @IdTable int
)
AS BEGIN
    SET NOCOUNT ON
```



```

BEGIN TRY
    IF NOT EXISTS (
        SELECT * FROM TableBusy WHERE IdTable = @IdTable
                                AND Active = 1
                                AND BeginDate <= GETDATE()
                                AND EndDate >= GETDATE()
    ) BEGIN ; THROW 52000, 'Table is already free.', 1 END
    UPDATE TableBusy SET Active = 0 WHERE IdTable = @IdTable AND
BeginDate <= GETDATE() AND EndDate >= GETDATE()
END TRY
BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'Cannot update the table. ERROR: '
+ ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCreateCompany

Opis: Tworzy nową firmę.

```

CREATE PROCEDURE PCreateCompany
(
    @Name nvarchar(256),
    @Nip nvarchar(128),
    @Address nvarchar(128),
    @PostCode nvarchar(16),
    @City nvarchar(128),
    @Country nvarchar(128),
    @Email nvarchar(256),
    @Phone nvarchar(128)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (SELECT * FROM Company WHERE Nip = @Nip) BEGIN ; THROW
52000, 'Company with given nip already exists.', 1 END
        INSERT INTO Company (Name, Nip, Address, PostCode, City, Country,
Email, Phone) VALUES
                                (@Name, @Nip, @Address, @PostCode, @City,
@Country, @Email, @Phone)
    END TRY
    BEGIN CATCH

```

```

        DECLARE @error nvarchar(2048) = 'Cannot create the company. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PUpdateCompany

Opis: Edytuje informacje o danej firmie.

```

CREATE PROCEDURE PUpdateCompany
(
    @IdCompany int,
    @Name nvarchar(256),
    @Nip nvarchar(128),
    @Address nvarchar(128),
    @PostCode nvarchar(16),
    @City nvarchar(128),
    @Country nvarchar(128),
    @Email nvarchar(256),
    @Phone nvarchar(128)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (SELECT * FROM Company WHERE Nip = @Nip AND IdCompany !=
@IdCompany) BEGIN ;
            THROW 52000, 'Company with given nip already exists.', 1 END

        IF NOT EXISTS (SELECT * FROM Company WHERE IdCompany = @IdCompany)
BEGIN ;
            THROW 52000, 'Company with given id does not exist.', 1 END

        UPDATE Company SET
            Name = @Name,
            Nip = @Nip,
            Address = @Address,
            PostCode = @PostCode,
            City = @City,
            Country = @Country,
            Email = @Email,

```

```

        Phone = @Phone
    WHERE IdCompany = @IdCompany
END TRY
BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'Cannot update the company. ERROR:
' + ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PAddEmployeeToCompany

Opis: Dodaje pracownika firmy będącej klientem restauracji do tabeli CompanyEmployee.

```

CREATE PROCEDURE PAddEmployeeToCompany
(
    @IdCompany INT,
    @NameCompany NVARCHAR(256),
    @FirstName NVARCHAR(128),
    @LastName NVARCHAR(128),
    @Email NVARCHAR(128),
    @Phone NVARCHAR(128)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS
        (
            SELECT *
            FROM CompanyEmployee CE
            WHERE CE.FirstName = @FirstName AND
                CE.LastName = @LastName AND
                CE.Email = @Email
        ) BEGIN; THROW 52000, 'Employee already exists.',1 END
        IF NOT EXISTS
        (
            SELECT *
            FROM Company c
            WHERE C.IdCompany = @IdCompany AND
                C.Name = @NameCompany
        ) BEGIN; THROW 52000, 'Company does not exist in the
database.', 1 END
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot update the company. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

```

        INSERT INTO CompanyEmployee (IdCompany, FirstName, LastName, Email,
Phone) VALUES (@IdCompany, @FirstName, @LastName, @Email, @Phone)
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot add the employee to
database. ERROR: ' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PUpdateCompanyEmployee

Opis: Zmienia wartości dla danego pracownika firmy.

```

CREATE PROCEDURE PUpdateCompanyEmployee
(
    @IdCompanyEmployee INT,
    @IdCompany INT,
    @FirstName NVARCHAR(128),
    @LastName NVARCHAR(128),
    @Email NVARCHAR(128),
    @Phone NVARCHAR(128)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM CompanyEmployee CE WHERE
CE.IdCompanyEmployee = @IdCompanyEmployee) BEGIN; THROW 52000, 'Employee
does not exist.',1 END
        UPDATE CompanyEmployee SET IdCompany = @IdCompany, FirstName =
@FirstName, LastName = @LastName, Email = @Email, Phone = @Phone WHERE
IdCompanyEmployee = @IdCompanyEmployee
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot update the employee in
database. ERROR: ' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateCustomer

Opis: Tworzy nowego klienta indywidualnego w bazie danych.

```

CREATE PROCEDURE PCreateCustomer
(
    @FirstName NVARCHAR(128),
    @LastName NVARCHAR(128),
    @Email NVARCHAR(128),
    @Phone NVARCHAR(128),
    @TotalPaid MONEY
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS (SELECT * FROM Customer C WHERE C.FirstName = @FirstName
AND C.LastName = @LastName AND C.Email = @Email) BEGIN; THROW 52000,
'Customer already exists.',1 END
        INSERT INTO Customer (FirstName, LastName, Email, Phone, TotalPaid)
VALUES (@FirstName, @LastName, @Email, @Phone, @TotalPaid)
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot create the customer. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PUpdateCustomer

Opis: Zmienia wartości dla danego klienta indywidualnego.

```

CREATE PROCEDURE PUpdateCustomer
(
    @IdCustomer INT,
    @FirstName NVARCHAR(128),
    @LastName NVARCHAR(128),
    @Email NVARCHAR(128),
    @Phone NVARCHAR(128),
    @TotalPaid MONEY
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Customer C WHERE C.IdCustomer =
@IdCustomer) BEGIN; THROW 52000, 'Customer does not exist.',1 END
        UPDATE Customer SET FirstName = @FirstName, LastName = @LastName,

```

```

Email = @Email, Phone = @Phone, TotalPaid = @TotalPaid WHERE IdCustomer =
@IdCustomer
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot update the customer. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateTemporaryDiscountForCustomer

Opis: Tworzy zniżkę krótkoterminową dla istniejącego klienta indywidualnego w bazie danych.

```

CREATE PROCEDURE PCreateTemporaryDiscountForCustomer
(
    @IdCustomer INT,
    @BeginDate DATETIME,
    @EndDate DATETIME
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Customer C WHERE C.IdCustomer =
@IdCustomer) BEGIN; THROW 52000, 'Customer does not exists of this id.',1
    END

    INSERT INTO CustomerDiscount(IdCustomer, Discount, Active,
BeginDate, EndDate)
        VALUES (@IdCustomer, (SELECT
[dbo].[FGetValueByConfiguredVariable]('R2')), 1, @BeginDate, @EndDate)
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot create the discount for
customer. ERROR: ' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreatePermanentDiscountForCustomer

Opis: Tworzy stałą zniżkę dla istniejącego klienta indywidualnego w bazie danych.

```

CREATE PROCEDURE PCreatePermanentDiscountForCustomer
(

```

```

        @IdCustomer INT,
        @BeginDate DATETIME
    )
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @Discount INT = (SELECT
[dbo].[FGetValueByConfiguredVariable]('R1'));
        IF NOT EXISTS (SELECT * FROM Customer C WHERE C.IdCustomer =
@IdCustomer) BEGIN; THROW 52000, 'Customer does not exists of this id.',1
END
        IF (SELECT Discount FROM CustomerDiscount CD WHERE CD.IdCustomer =
@IdCustomer AND CD.EndDate >= '9999-12-31 23:59:59.000') = @Discount BEGIN;
THROW 52000, 'Customer has a discount.',1 END
        INSERT INTO CustomerDiscount(IdCustomer, Discount, Active,
BeginDate, EndDate)
            VALUES (@IdCustomer, @Discount, 1, @BeginDate, '9999-12-31
23:59:59.000');
        END TRY
        BEGIN CATCH
            DECLARE @error NVARCHAR(2048) = 'Cannot create the discount for
customer. ERROR: ' + ERROR_MESSAGE();
            THROW 52000, @error, 1
        END CATCH
    END
END

```

Nazwa: PUpdateConfigurationValue  
 Opis: Zmienia wartość zmiennej konfiguracyjnej.

```

CREATE PROCEDURE PUpdateConfigurationValue
(
    @Name NCHAR(64),
    @Value INT
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Configuration C WHERE C.Name = @Name)
BEGIN; THROW 52000, 'Configuration variable does not exists of this
name.',1 END
        UPDATE Configuration SET Value = @Value WHERE Name = @Name;
    END TRY

```

```

        BEGIN CATCH
            DECLARE @error NVARCHAR(2048) = 'Cannot update the value. ERROR: '
+ ERROR_MESSAGE();
            THROW 52000, @error, 1
        END CATCH
    END
END

```

Nazwa: PUpdateDiscount

Opis: Zmiana wartości danej zniżki.

```

CREATE PROCEDURE PUpdateDiscount
(
    @IdCustomerDiscount INT,
    @Discount INT,
    @Active BIT,
    @BeginDate DATETIME,
    @EndDate DATETIME
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM CustomerDiscount CD WHERE
CD.IdCustomerDiscount = @IdCustomerDiscount) BEGIN; THROW 52000, 'Discount
does not exist.',1 END
        UPDATE CustomerDiscount SET Discount = @Discount, Active = @Active,
BeginDate = @BeginDate, EndDate = @EndDate WHERE IdCustomerDiscount =
@IdCustomerDiscount;
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot update the discount. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateSingleInvoice

Opis: Dodaje fakturę na pojedyncze zamówienie.

```

CREATE PROCEDURE PCreateSingleInvoice
(
    @IdOrder INT,
    @Ur1 NVARCHAR(256)
)

```



```

AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @IssueDate DATETIME = GETDATE();
        IF NOT EXISTS (SELECT * FROM [Order] O WHERE O.IdOrder = @IdOrder)
BEGIN; THROW 52000, 'Order does not exists of this id.',1 END
        IF EXISTS (SELECT * FROM InvoiceDetail ID WHERE ID.IdOrder =
@IdOrder) BEGIN; THROW 52000, 'Invoice exists for this order.',1 END
        INSERT INTO Invoice(url, issuedate) VALUES (@Url, @IssueDate)
        INSERT INTO InvoiceDetail(IdInvoice, IdOrder) VALUES ( (SELECT
IdInvoice FROM Invoice I WHERE I.Url = @Url AND I.IssueDate = @IssueDate),
@IdOrder)
    END TRY
    BEGIN CATCH
        DECLARE @error NVARCHAR(2048) = 'Cannot create the invoice. ERROR:
' + ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateCollectiveInvoiceForCompany

Opis: Dodaje zbiorczą fakturę na firmę.

```

CREATE PROCEDURE PCreateCollectiveInvoiceForCompany
(
    @IdCompany INT,
    @Url NVARCHAR(256)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @IssueDate DATETIME = GETDATE();
        IF NOT EXISTS (SELECT * FROM [Company] C WHERE C.IdCompany =
@IdCompany) BEGIN; THROW 52000, 'Company does not exists of this id.',1 END
        DECLARE @Cursor CURSOR;
        DECLARE @IdOrder INT;
        DECLARE @IdInvoiceVar INT;

        BEGIN
            SET @Cursor = CURSOR FOR
            SELECT IdOrder from FGetCompanyInvoices(@IdCompany)

            OPEN @Cursor

```

```

        FETCH NEXT FROM @Cursor
        INTO @IdOrder

        WHILE @@FETCH_STATUS = 0
        BEGIN
            SET @IdInvoiceVar = (SELECT IdInvoice FROM InvoiceDetail
WHERE IdOrder = @IdOrder)
            IF (SELECT Count(IdOrder) FROM InvoiceDetail WHERE IdInvoice
= @IdInvoiceVar) > 1 AND (DATEDIFF(MONTH,(SELECT IssueDate FROM Invoice
WHERE IdInvoice = @IdInvoiceVar), GETDATE())) < 1)
            BEGIN;
                CLOSE @Cursor ;
                DEALLOCATE @Cursor;
                THROW 52000, 'Company has an collective invoice in last
month.',1
            END

            FETCH NEXT FROM @Cursor
            INTO @IdOrder
        END;

        CLOSE @Cursor ;
        DEALLOCATE @Cursor;
    END;

    IF (SELECT COUNT(IdOrder) FROM [Order] O WHERE O.IdCompany =
@IdCompany AND IdOrder NOT IN (SELECT IdOrder FROM
FGetCompanyInvoices(@IdCompany))) = 0
    BEGIN;
        THROW 52000, 'Company has not got an order without invoice.',1
    END

    INSERT INTO Invoice(url, issuedate) VALUES (@Url, @IssueDate)
    BEGIN
        SET @Cursor = CURSOR FOR
        SELECT IdOrder FROM [Order] O WHERE O.IdCompany = @IdCompany
AND IdOrder NOT IN (SELECT IdOrder FROM FGetCompanyInvoices(@IdCompany))

        OPEN @Cursor
        FETCH NEXT FROM @Cursor
        INTO @IdOrder

```

```

        WHILE @@FETCH_STATUS = 0
        BEGIN
            INSERT INTO InvoiceDetail(IdInvoice, IdOrder) VALUES (
                (SELECT IdInvoice FROM Invoice I WHERE I.Url = @Url AND I.IssueDate =
                @IssueDate), @IdOrder)
            FETCH NEXT FROM @Cursor
            INTO @IdOrder
        END;

        CLOSE @Cursor ;
        DEALLOCATE @Cursor;
    END;

END TRY
BEGIN CATCH
    DECLARE @error NVARCHAR(2048) = 'Cannot create the invoice. ERROR:
' + ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCreateCollectiveInvoiceForCustomer

Opis: Dodaje zbiorczą fakturę na klienta indywidualnego.

```

CREATE PROCEDURE PCreateCollectiveInvoiceForCustomer
(
    @IdCustomer INT,
    @Url NVARCHAR(256)
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @IssueDate DATETIME = GETDATE();
        IF NOT EXISTS (SELECT * FROM [Customer] C WHERE C.IdCustomer =
        @IdCustomer) BEGIN; THROW 52000, 'Customer does not exists of this id.',1
    END

    DECLARE @Cursor CURSOR;
    DECLARE @IdOrder INT;
    DECLARE @IdInvoiceVar INT;

    BEGIN
        SET @Cursor = CURSOR FOR
    
```

```

SELECT IdOrder from FGetCustomerInvoices(@IdCustomer)

OPEN @Cursor
FETCH NEXT FROM @Cursor
INTO @IdOrder

WHILE @@FETCH_STATUS = 0
BEGIN
    SET @IdInvoiceVar = (SELECT IdInvoice FROM InvoiceDetail
WHERE IdOrder = @IdOrder)
    IF (SELECT Count(IdOrder) FROM InvoiceDetail WHERE IdInvoice
= @IdInvoiceVar) > 1 AND (DATEDIFF(MONTH,(SELECT IssueDate FROM Invoice
WHERE IdInvoice = @IdInvoiceVar), GETDATE()) < 1)
    BEGIN;
        CLOSE @Cursor ;
        DEALLOCATE @Cursor;
        THROW 52000, 'Customer has an collective invoice in
last month.',1
    END

    FETCH NEXT FROM @Cursor
    INTO @IdOrder
END;

CLOSE @Cursor ;
DEALLOCATE @Cursor;
END;

IF (SELECT COUNT(IdOrder) FROM [Order] O WHERE O.IdCustomer =
@IdCustomer AND IdOrder NOT IN (SELECT IdOrder FROM
FGetCustomerInvoices(@IdCustomer))) = 0
BEGIN;
    THROW 52000, 'Customer has not got an order without invoice.',1
END

INSERT INTO Invoice(url, issuedate) VALUES (@Ur1, @IssueDate)
BEGIN
    SET @Cursor = CURSOR FOR
    SELECT IdOrder FROM [Order] O WHERE O.IdCustomer = @IdCustomer
AND IdOrder NOT IN (SELECT IdOrder FROM FGetCustomerInvoices(@IdCustomer))

```

```

        OPEN @Cursor
        FETCH NEXT FROM @Cursor
        INTO @IdOrder

        WHILE @@FETCH_STATUS = 0
        BEGIN
            INSERT INTO InvoiceDetail(IdInvoice, IdOrder) VALUES (
                (SELECT IdInvoice FROM Invoice I WHERE I.Url = @Url AND I.IssueDate =
                @IssueDate), @IdOrder)
            FETCH NEXT FROM @Cursor
            INTO @IdOrder
        END;

        CLOSE @Cursor ;
        DEALLOCATE @Cursor;
    END;

END TRY
BEGIN CATCH
    DECLARE @error NVARCHAR(2048) = 'Cannot create the invoice. ERROR:
' + ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCreateOrderTakeout

Opis: Tworzenie zamówienia na wynos

```

CREATE PROCEDURE PCreateOrderTakeout
(
    @IdCustomer int = NULL,
    @IdCompany int = NULL,
    @TotalPrice money = NULL,
    @Discount int = 0,
    @Takeaway bit = 1,
    @Prepaid bit = 0,
    @Active bit = 1,
    @PickupDate datetime,
    @CreateDate datetime = NULL
)
AS BEGIN
    SET @CreateDate = GETDATE()

```

```

SET NOCOUNT ON
BEGIN TRY
    IF @IdCustomer IS NOT NULL and @IdCompany IS NOT NULL BEGIN ; THROW
52000, 'Wrong data for IdCustomer and IdCompany', 1 END
    INSERT INTO [ORDER] (IdCustomer, IdCompany, TotalPrice, Discount,
Takeaway, Prepaid, Active, PickupDate, CreateDate)
    VALUES (@IdCustomer, @IdCompany, @TotalPrice, @Discount, @Takeaway,
@Prepaid, @Active, @PickupDate, @CreateDate)
END TRY
BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'Cannot create the Order, ERROR: ' +
ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCreateOrderOnSite

Opis: Tworzenie zamówienia na miejscu

```

CREATE PROCEDURE PCreateOrderOnSite
(
    @IdCustomer int = NULL,
    @IdCompany int = NULL,
    @TotalPrice money = 0,
    @Discount int = 0,
    @Takeaway bit = 0,
    @Prepaid bit = 0,
    @Active bit = 1,
    @PickupDate datetime,
    @CreateDate datetime = NULL,
)
AS BEGIN
    SET @CreateDate = GETDATE()
    SET NOCOUNT ON
    BEGIN TRY
        IF @IdCustomer IS NOT NULL and @IdCompany IS NOT NULL BEGIN ; THROW
52000, 'Wrong data for IdCustomer and IdCompany', 1 END
        INSERT INTO [ORDER] (IdCustomer, IdCompany, TotalPrice, Discount,
Takeaway, Prepaid, Active, PickupDate, CreateDate)
        VALUES (@IdCustomer, @IdCompany, @TotalPrice, @Discount, @Takeaway,
@Prepaid, @Active, @PickupDate, @CreateDate)
    END TRY

```

```

BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'Cannot create the Order, ERROR: ' +
ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCancelOrderTakeout

Opis: Anulowanie zamówienia na wynos

```

CREATE PROCEDURE PCancelOrderTakeOut
(
    @IdOrder int,
    @Active bit = 0
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM [Order] WHERE IdOrder = @IdOrder) BEGIN
; THROW 52000, 'Order for given IdOrder does not exist.', 1 END
        UPDATE [Order] SET Active = @Active where IdOrder = @IdOrder
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot cancel the Order, ERROR: ' +
ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PAddProductToOrder

Opis: Dodawanie produktów do zamówienia

```

CREATE PROCEDURE PAddProductToOrder
(
    @IdOrder int,
    @IdMenuDetail int,
    @Quantity smallint
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(SELECT * FROM [Order] WHERE IdOrder = @IdOrder) BEGIN

```

```

; THROW 52000, 'Order with given ID does not exist', 1 END
    IF NOT EXISTS(SELECT * FROM MenuDetail INNER JOIN Menu M on
MenuDetail.IdMenu = M.IdMenu WHERE
    Year(M.Day) = YEAR(GETDATE()) AND MONTH(M.Day) = MONTH(GETDATE())
AND DAY(M.Day) = DAY(GETDATE()) AND IdMenuDetail = @IdMenuDetail)
        BEGIN ; THROW 52000, 'Product with given Id is not one the menu
right now, choose something else!', 1 END

    IF (DATEPART(DW,GETDATE()) NOT IN (5,6,7)) AND (SELECT P.IdCategory
from MenuDetail md inner join Product P on md.IdProduct = P.IdProduct where
md.IdMenuDetail = @IdMenuDetail) = 7
        BEGIN ; THROW 52000, 'Cannot order seafood at this time of the
week', 1 END

    INSERT INTO OrderDetail (IdOrder, IdMenuDetail, Quantity) VALUES
(@IdOrder, @IdMenuDetail, @Quantity)
    DECLARE @total_price INT = (SELECT SUM(Price*Quantity) FROM
OrderDetail od INNER JOIN MenuDetail D ON od.IdMenuDetail = D.IdMenuDetail
    INNER JOIN Menu m ON D.IdMenu = m.IdMenu
    WHERE YEAR(M.Day) = YEAR(GETDATE()) AND MONTH(M.Day) =
MONTH(GETDATE()) AND DAY(M.Day) = DAY(GETDATE()) AND
    od.IdOrder = @IdOrder
    GROUP BY IdOrder)
    UPDATE [Order] SET TotalPrice = @total_price where IdOrder =
@IdOrder
    IF EXISTS(SELECT * FROM Customer C INNER JOIN [Order] O ON
C.IdCustomer = O.IdCustomer WHERE O.IdOrder = @IdOrder) BEGIN ;
    DECLARE @total_paid INT = (SELECT C.TotalPaid FROM Customer C INNER
JOIN [Order] O ON C.IdCustomer = O.IdCustomer WHERE O.IdOrder = @IdOrder)
    DECLARE @cu_id INT = (SELECT C.IdCustomer FROM Customer C INNER JOIN
[Order] O ON C.IdCustomer = O.IdCustomer WHERE O.IdOrder = @IdOrder)
    UPDATE Customer SET TotalPaid = @total_paid + @total_price WHERE
IdCustomer = @cu_id END
END TRY
BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'ERROR: ' + ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCreateReservationIndividualClient



Opis: Stworzenie rezerwacji dla klienta indywidualnego wraz z zamówieniem

```
CREATE PROCEDURE PCreateReservationIndividualClient
(
    @IdCustomer1 int = NULL,
    @IdCompany1 int = NULL,
    @IdReservationStatus bit = 0,
    @Amount smallint,
    @BeginDate datetime = NULL,
    @HowLong int = 2,
    @Discount1 int = 0
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF @IdCustomer1 IS NULL and @IdCompany1 IS NOT NULL BEGIN ; THROW
52000, 'It is a reservation for and individual client, try using one for a
Company', 1 END
        DECLARE @End_Date datetime = DATEADD(HOUR,@HowLong, @BeginDate)
        EXEC PCreateOrderOnSite @IdCustomer = @IdCustomer1, @IdCompany =
@IdCompany1, @Discount = @Discount1, @CreateDate = @BeginDate, @PickupDate
= @End_Date
        INSERT INTO Reservation(IdCustomer, IdCompany, IdOrder,
IdReservationStatus, Amount, BeginDate, EndDate)
        VALUES (@IdCustomer1, @IdCompany1, (SELECT TOP 1 IdOrder from
[Order] order by IdOrder desc), @IdReservationStatus, @Amount, @BeginDate,
@End_Date)
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot create reservation, ERROR: '
+ ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END
```

Nazwa: PCreateReservationCompany

Opis: Stworzenie rezerwacji dla firmy z zamówieniem bądź bez

```
CREATE PROCEDURE PCreateReservationCompany
(
    @IdCustomer1 int = NULL,
    @IdCompany1 int = NULL,
    @IdReservationStatus bit = 0,
```

```

    @Amount smallint,
    @BeginDate datetime = NULL,
    @HowLong int = 2,
    @Discount1 int = 0,
    @WithOrder bit = 0
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF @IdCustomer1 IS NOT NULL and @IdCompany1 IS NULL BEGIN ; THROW
52000, 'It is a reservation for a Company, try using one for an individual
client', 1 END
        DECLARE @End_Date datetime = DATEADD(HOUR,@HowLong, @BeginDate)
        IF @WithOrder = 1 BEGIN ; EXEC PCreateOrderOnSite @IdCustomer =
@IdCustomer1, @IdCompany = @IdCompany1, @Discount = @Discount1, @CreateDate
= @BeginDate, @PickupDate = @End_Date
        END
        INSERT INTO Reservation(IdCustomer, IdCompany, IdOrder,
IdReservationStatus, Amount, BeginDate, EndDate)
        VALUES (@IdCustomer1, @IdCompany1, (SELECT TOP 1 IdOrder from
[Order] order by IdOrder desc), @IdReservationStatus, @Amount, @BeginDate,
@End_Date)
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot create reservation, ERROR: '
+ ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PAcceptReservationWithTables

Opis: Zaakceptowanie rezerwacji z rezerwacją stolików

```

CREATE PROCEDURE PAcceptReservationWithTables
(
    @IdReservation int
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Reservation WHERE IdReservation =
@IdReservation) BEGIN ; THROW 52000,

```

```

        'Reservation with given ID does not exist!', 1 END
    IF (SELECT IdReservationStatus from Reservation where @IdReservation
= IdReservation) = 1 BEGIN ; THROW 52000
        , 'Reservation has already been accepted!', 1 END
    IF EXISTS (SELECT top 1 IdTable FROM VAvailableTables
        where [Amount of seats] = (SELECT Amount FROM Reservation WHERE
IdReservation = @IdReservation))
        BEGIN ;
        DECLARE @TABLE_ID INT = (SELECT top 1 IdTable FROM
VAvailableTables
            where [Amount of seats] = (SELECT Amount FROM Reservation WHERE
IdReservation = @IdReservation))
        DECLARE @Res_Start datetime = (SELECT BeginDate FROM Reservation
WHERE IdReservation = @IdReservation)
        DECLARE @End_Res datetime = (SELECT EndDate FROM Reservation
WHERE IdReservation = @IdReservation)
        EXEC PMarkTableBusy @IdTable = @TABLE_ID, @BeginDate =
@Res_Start, @EndDate = @End_Res
        UPDATE TableBusy SET IdReservation = @IdReservation WHERE
IdTable = @TABLE_ID
        END
        UPDATE Reservation SET IdReservationStatus = 1 where IdReservation =
@IdReservation
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot accept reservation. ERROR: '
+ ERROR_MESSAGE();
        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCancelReservation

Opis: Odrzucenie rezerwacji

```

CREATE PROCEDURE PRejectReservation
(
    @IdReservation int
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Reservation WHERE IdReservation =

```

```

@IdReservation) BEGIN ; THROW 52000,
    'Reservation with given ID does not exist!', 1 END
    IF (SELECT IdReservationStatus from Reservation where @IdReservation
= IdReservation) = 2 BEGIN ; THROW 52000
    , 'Reservation has already been rejected!', 1 END
    IF (SELECT IdReservationStatus from Reservation where @IdReservation
= IdReservation) != 0 BEGIN ; THROW 52000
    , 'Use PCancelReservation to cancel this reservation!', 1 END
    UPDATE Reservation SET IdReservationStatus = 2
END TRY
BEGIN CATCH
    DECLARE @error nvarchar(2048) = 'Cannot reject reservation. ERROR: '
+ ERROR_MESSAGE();
    THROW 52000, @error, 1
END CATCH
END

```

Nazwa: PCancelReservation

Opis: Anulowanie zaakceptowanej rezerwacji

```

CREATE PROCEDURE PCancelReservation
(
    @IdReservation int
)
AS BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Reservation WHERE IdReservation =
@IdReservation) BEGIN ; THROW 52000,
        'Reservation with given ID does not exist!', 1 END
        IF (SELECT IdReservationStatus from Reservation where @IdReservation
= IdReservation) = 2 BEGIN ; THROW 52000
        , 'Reservation has already been cancelled!', 1 END
        IF (SELECT IdReservationStatus from Reservation where @IdReservation
= IdReservation) != 1 BEGIN ; THROW 52000
        , 'Use PRejectReservation to cancel this reservation!', 1 END
        UPDATE Reservation SET IdReservationStatus = 2
    END TRY
    BEGIN CATCH
        DECLARE @error nvarchar(2048) = 'Cannot cancel reservation. ERROR: '
+ ERROR_MESSAGE();

```

```

        THROW 52000, @error, 1
    END CATCH
END

```

Nazwa: PCreateMenu

Opis: Tworzenie i edycja menu na dany dzień

```

CREATE PROCEDURE PCreateMenu
(
    @Day datetime = NULL,
    @IdProduct int,
    @Price money
)
AS BEGIN
    SET NOCOUNT ON
    SET @day = GETDATE()
    BEGIN TRY
        IF EXISTS (SELECT md.IdProduct FROM Menu M INNER JOIN MenuDetail MD
on M.IdMenu = MD.IdMenu
        WHERE Year(M.Day) = YEAR(GETDATE()) AND MONTH(M.Day) =
MONTH(GETDATE()) AND DAY(M.Day) = DAY(GETDATE())
        AND @IdProduct = MD.IdProduct)
        BEGIN ; THROW 52000, 'Product already in the Menu!', 1 END
        IF NOT EXISTS (SELECT * FROM Menu M WHERE Year(M.Day) =
YEAR(GETDATE()) and MONTH(M.Day) = MONTH(GETDATE())
        and DAY(M.Day) = DAY(GETDATE()))
        BEGIN ; INSERT INTO Menu (Day) VALUES (@Day) END
        IF NOT EXISTS (SELECT * FROM Product WHERE IdProduct = @IdProduct)
        BEGIN ;
            THROW 52000, 'Product with given Id does not exist!', 1 END
            INSERT INTO MenuDetail (IdMenu, IdProduct, Price) VALUES ((SELECT
TOP 1 IdMenu FROM Menu M WHERE
            YEAR(M.Day) = YEAR(GETDATE()) AND MONTH(M.Day) = MONTH(GETDATE())
            AND DAY(M.Day) = DAY(GETDATE())),
            @IdProduct, @Price)
        END TRY
        BEGIN CATCH
            DECLARE @error nvarchar(2048) = 'Cannot add product to the menu.
ERROR: ' + ERROR_MESSAGE();
            THROW 52000, @error, 1
        END CATCH
    END
END

```

# Triggery

Nazwa: TAssignDiscountR1

Opis: Przydziela rabat R1 dla klienta indywidualnego, który złożył co najmniej Z1 zamówień za co najmniej K1 każde.

```
CREATE TRIGGER TAssignDiscountR1 ON [Order] AFTER INSERT AS
BEGIN
    DECLARE @IdCustomer INT;
    DECLARE @TotalPrice MONEY;
    DECLARE @R1 INT;
    DECLARE @Z1 INT;
    DECLARE @K1 INT;
    SET @IdCustomer = (SELECT i.IdCustomer FROM inserted i)
    SET @TotalPrice = (SELECT i.TotalPrice FROM inserted i)
    SET @Z1 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'Z1')
    SET @K1 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'K1')
    SET @R1 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'R1')
    IF (@IdCustomer IS NULL)
    BEGIN
        RETURN
    END
    IF (SELECT DISTINCT 1 FROM CustomerDiscount cd WHERE cd.IdCustomer =
@IdCustomer AND cd.Discount = @R1) IS NOT NULL
    BEGIN
        RETURN
    END
    IF (SELECT DISTINCT 1 FROM [Order] o WHERE o.IdCustomer = @IdCustomer
AND o.TotalPrice >= @K1 GROUP BY @IdCustomer HAVING COUNT(*) >= @Z1) IS
NULL
    BEGIN
        RETURN
    END
    INSERT INTO CustomerDiscount VALUES (@IdCustomer, @R1, 1, GETDATE(),
'9999-12-31 23:59:59.000')
END
```

Nazwa: TAssignDiscountR2

Opis: Przydziela rabat R2 dla klienta indywidualnego, który wydał co najmniej K2 za złożone zamówienia.

```
CREATE TRIGGER TAssignDiscountR2 ON [Order] AFTER INSERT AS
BEGIN
    DECLARE @IdCustomer INT;
    DECLARE @TotalPaid MONEY;
    DECLARE @R2 INT;
    DECLARE @D1 INT;
    DECLARE @K2 INT;
    SET @IdCustomer = (SELECT i.IdCustomer FROM inserted i)
    SET @D1 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'D1')
    SET @K2 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'K2')
    SET @R2 = (SELECT c.Value FROM Configuration c WHERE c.Name = 'R2')
    IF (@IdCustomer IS NULL)
    BEGIN
        RETURN
    END
    SET @TotalPaid = (SELECT c.TotalPaid FROM Customer c WHERE c.IdCustomer = @IdCustomer)
    IF (@TotalPaid >= @K2)
    BEGIN
        INSERT INTO CustomerDiscount VALUES (@IdCustomer, @R2, 1,
        GETDATE(), DATEADD(day, @D1, GETDATE()))
        UPDATE Customer SET TotalPaid = @TotalPaid - @K2 WHERE IdCustomer = @IdCustomer
    END
END
```

Nazwa: TCheckMenu

Opis: Trigger sprawdzający czy w menu co najmniej połowa produktów jest inna niż przed 14 dniami

```
CREATE TRIGGER TCheckMenu on MenuDetail AFTER INSERT
AS
BEGIN
    DECLARE @ActualMenuCountDistinct INT;
    DECLARE @MenuCount INT;
    SET @ActualMenuCountDistinct =
        (SELECT COUNT(*) FROM (SELECT IdProduct FROM MenuDetail INNER JOIN
```

```

        Menu M ON MenuDetail.IdMenu = M.IdMenu WHERE Day = GETDATE()
    EXCEPT
        SELECT IdProduct FROM MenuDetail INNER JOIN Menu M ON M.IdMenu =
MenuDetail.IdMenu
        WHERE Day = convert(varchar(15),DATEADD(DAY,-14,GETDATE()),110)) x)
    SET @MenuCount =
        (SELECT count(*) / 2 FROM MenuDetail INNER JOIN
        Menu M ON MenuDetail.IdMenu = M.IdMenu WHERE Day =
CONVERT(VARCHAR(15),GETDATE(),110))
    IF @ActualMenuCountDistinct < @MenuCount
        BEGIN
            RAISERROR('Add some products to the menu so it is more different
than the one from 2 weeks ago ;)',-1,-1)
        END
    END
END

```



## Indeksy

Zdefiniowaliśmy następujące indeksy, które zwiększają szybkość operacji wyszukiwania w bazie danych.

```
CREATE UNIQUE INDEX Category_IdCategory_uindex ON Category (IdCategory)
```

```
CREATE UNIQUE INDEX Company_IdCompany_uindex ON Company (IdCompany)
```

```
CREATE UNIQUE INDEX Company_Nip_uindex ON Company (Nip)
```

```
CREATE UNIQUE INDEX Company_Email_uindex ON Company (Email)
```

```
CREATE UNIQUE INDEX Company_Phone_uindex ON Company (Phone)
```

```
CREATE UNIQUE INDEX CompanyEmployee_IdCompanyEmployee_uindex ON  
CompanyEmployee (IdCompanyEmployee)
```

```
CREATE UNIQUE INDEX Configuration_Name_uindex ON Configuration (Name)
```

```
CREATE UNIQUE INDEX Customer_IdCustomer_uindex ON Customer (IdCustomer)
```

```
CREATE UNIQUE INDEX Customer_Email_uindex ON Customer (Email)
```

```
CREATE UNIQUE INDEX Customer_Phone_uindex ON Customer (Phone)
```

```
CREATE UNIQUE INDEX CustomerDiscount_IdCustomerDiscount_uindex ON  
CustomerDiscount (IdCustomerDiscount)
```


```
CREATE UNIQUE INDEX Invoice_IdInvoice_uindex ON Invoice (IdInvoice)
```

```
CREATE UNIQUE INDEX Menu_IdMenu_uindex ON Menu (IdMenu)
```

```
CREATE UNIQUE INDEX InvoiceDetail_IdInvoiceDetail_uindex ON InvoiceDetail  
(IdInvoiceDetail)
```

```
CREATE UNIQUE INDEX Order_IdOrder_uindex ON [Order] (IdOrder)
```

```
CREATE UNIQUE INDEX MenuDetail_IdMenuDetail_uindex ON MenuDetail  
(IdMenuDetail)
```



```
CREATE UNIQUE INDEX OrderDetail_IdOrderDetail_uindex ON OrderDetail
(IdOrderDetail)
```

```
CREATE UNIQUE INDEX Product_IdProduct_uindex ON Product (IdProduct)
```

```
CREATE UNIQUE INDEX Reservation_IdReservation_uindex ON Reservation
(IdReservation)
```

```
CREATE UNIQUE INDEX
ReservationCompanyEmployee_IdReservationCompanyEmployee_uindex ON
ReservationCompanyEmployee (IdReservationCompanyEmployee)
```

```
CREATE UNIQUE INDEX ReservationStatus_IdReservationStatus_uindex ON
ReservationStatus (IdReservationStatus)
```

```
CREATE UNIQUE INDEX ReservationStatus_Name_uindex ON ReservationStatus
(Name)
```

```
CREATE UNIQUE INDEX Table_IdTable_uindex ON [Table] (IdTable)
```

```
CREATE UNIQUE INDEX TableBusy_IdTableBusy_uindex ON TableBusy (IdTableBusy)
```

## Uprawnienia

**Administrator:** Ma dostęp do całej bazy danych. Może przeglądać wszystkie tabele i modyfikować ich dane.

**Manager:** Ma mniejsze uprawnienia niż administrator bazy. Zarządza całą restauracją, dlatego w bazie ma dostęp do wszystkich najważniejszych danych. Do jego odpowiedzialności należy:

- Zamówienia
  - Wyświetlenie listy przyszłych zamówień do wykonania
    - VFutureOrders
  - Wyświetlanie statystyk z zamówień z ostatniego tygodnia
    - VReservationStatisticsLastWeek
  - Wyświetlanie statystyk z zamówień z ostatniego miesiąca
    - VOrderStatisticsLastMonth
  - Wyświetlanie statystyk z zamówień z ostatniego roku
    - VOrderStatisticsLastYear
- Zarządzanie rezerwacjami:
  - Akceptacja danej rezerwacji wraz z zajęciem stolików
    - PAcceptReservationWithTables
  - Odrzucenie danej rezerwacji
    - PCancelReservation
  - Anulowanie zaakceptowanej rezerwacji
    - PCancelReservation
  - Wyświetlenie informacji o rezerwacjach firmowych, pracownikach imiennie oraz firmach
    - VCompanyReservation
  - Wyświetlenie listy przyszłych rezerwacji
    - VReservationFuture
  - Wyświetlenie listy zrealizowanych rezerwacji
    - VCompletedReservation
  - Wyświetlanie statystyk z zamówień z obecnego tygodnia
    - VReservationStatisticsLastWeek
  - Wyświetlanie statystyk z rezerwacji z obecnego miesiąca
    - VReservationStatisticsLastMonth
  - Wyświetlanie statystyk z rezerwacji z obecnego roku
    - VReservationStatisticsLastYear
- Zarządzanie menu i produktami:
  - Utworzenie menu z produktów oraz ustalenie ich cen na dany dzień
    - PCreateMenu
  - Wyświetlanie statystyk / raportu z menu z ostatniego tygodnia
  - Wyświetlanie statystyk / raportu z menu z ostatniego miesiąca
  - Dodanie nowego produktu

- PCreateProduct
  - Edycja danych produktu (nazwa, opis, kategoria)
    - PUpdateProduct
  - Aktywacja / dezaktywacja danego produktu
    - PUpdateProductStatus
  - Utworzenie nowej kategorii produktu
    - PCreateProductCategory
  - Edycja danej kategorii produktu (nazwa, opis)
    - PUpdateProductCategory
- Zarządzanie stolikami:
  - Oznaczenie danego stolika jako zajętego
    - PMarkTableBusy
  - Zwolnienie danego stolika
    - PFreeTable
  - Wyświetlanie aktualnie dostępnych/zajętych stolików
    - VBusyTable
  - Wyświetlanie stolików z ostatniego tygodnia
    - VTableWeek
  - Wyświetlanie stolików z ostatniego miesiąca
    - VTableMonth
- Zarządzanie fakturami:
  - Wystawienie faktury jednorazowej na zamówienie
    - PCreateSingleInvoice
  - Wystawienie faktury zbiorczej na zamówienia danej firmy
    - PCreateCollectiveInvoiceForCompany
  - Wystawienie faktury zbiorczej na zamówienia danego klienta
    - PCreateCollectiveInvoiceForCustomer
  - Wyświetlanie faktury na dane zamówienie
    - FGetInvoiceOfOrder
  - Wyświetlanie listy wystawionych faktur na danego klienta indywidualnego
    - FGetCustomerInvoices
  - Wyświetlanie listy wystawionych faktur na daną firmę
    - FGetCompanyInvoices
  - Wyświetlanie listy wystawionych faktur klientów indywidualnych
    - VInvoicesCustomers
  - Wyświetlanie listy wystawionych faktur na firmy
    - VInvoicesCompanies
  - Wyświetlanie historii wystawionych faktur na wszystkich klientów
    - VInvoices
- Zmienne konfiguracyjne:
  - Edycja danej zmiennej konfiguracyjnej (np. edycja wartości zniżek)
    - PUpdateConfigurationValue

- Wyświetlanie wartości danej zmiennej konfiguracyjnej
  - FGetValueByConfiguredVariable
- Wyświetlanie wartości zmiennych konfiguracyjnych
  - VConfigurationVariables

**Pracownik:** Możliwości pracownika zostały dopasowane do jego obowiązków obsługi klientów na co dzień w restauracji. Ma mniejsze uprawnienia niż manager.

- Zamówienia:
  - Utworzenie zamówienia złożonego przez klienta na wynos
    - PCreateOrderTakeout
  - Utworzenie zamówienia złożonego przez klienta na miejscu wraz z zajęciem stolika ( bez wykonania wcześniejszej rezerwacji w systemie)
    - PCreateOrderOnSite
  - Anulowanie zamówienia na wynos
    - PCancelOrderTakeout
  - Dodanie pozycji do zamówienia
    - PAddProductToOrder
  - Wyświetlenie informacji o danym zamówieniu (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data)
    - FOrderDetail
  - Wyświetlenie informacji o zamówieniu (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data)
    - VOrdersDetails
  - Wyświetlenie listy zrealizowanych zamówień
    - VCompletedOrders
  - Wyświetlenie listy zamówień złożonych
    - VOrdersByCustomer
  - Wyświetlenie listy zamówień złożonych przez danego klienta
    - FOrderByClient
  - Wyświetlenie sumarycznego kosztu danego zamówienia po rabatach
    - FOrderValue
  - Wyświetlenie listy przyszłych zamówień do wykonania
    - VFutureOrders
- Menu:
  - Wyświetlenie menu wraz z cenami oraz kategoriami produktów
    - VMenu
  - Wyświetlenie menu wraz z cenami oraz kategoriami produktów na dany dzień
    - FGetMenuProductsByDay

- Wyświetlenie informacji o produktach
  - VProduct
- Wyświetlanie produktów wraz z kategoriami
  - VCategoryProducts
- Wyświetlanie produktów danej kategorii
  - FGetProductsByCategory
- Wyświetlanie wszystkich kategorii
  - VCategory
- Stoliki:
  - Dodanie nowego stolika do bazy
    - PCreateTable
  - Zmiana ilości miejsca dla danego stolika
    - PUpdateTableCapacity
- Obsługa klientów:
  - Dodanie nowej firmy
    - PCreateCompany
  - Edycja danych istniejącej firmy
    - PUpdateCompany
  - Dodanie nowego pracownika do firmy
    - PAddEmployeeToCompany
  - Edycja danych pracownika firmy
    - PUpdateCompanyEmployee
  - Wyświetlanie zapisanych firm
    - VSavedCompanies
  - Wyświetlenie wszystkich pracowników
    - VCompanyEmployees
  - Wyświetlenie pracowników danej firmy
    - FGetEmployeesByCompany
  - Dodanie nowego klienta indywidualnego
    - PCreateCustomer
  - Edycja danych istniejącego klienta indywidualnego
    - PUpdateCustomer
  - Wyświetlanie informacji o klientach indywidualnych, ilości złożonych zamówień, sumarycznej kwoty za zamówienia, ilości wykonanych rezerwacji
    - VCustomer
  - Udzielenie zniżki krótkoterminowej dla klienta indywidualnego
    - PCreateTemporaryDiscountForCustomer
  - Udzielenie stałej zniżki dla klienta indywidualnego
    - PCreatePermanentDiscountForCustomer
  - Anulowanie danej zniżki dla klienta indywidualnego
    - PUpdateDiscount

- Wyświetlenie informacji o zniżkach
  - VCustomersDiscounts
- Wyświetlanie statystyk / raportu z zniżek dla klientów indywidualnych z ostatniego tygodnia
  - VCustomersDiscountsLastWeek
- Wyświetlanie statystyk / raportu z zniżek dla klientów indywidualnych z ostatniego miesiąca
  - VCustomersDiscountsLastMonth
- Wyświetlenie informacji o zniżkach dla danego klienta indywidualnego
  - FGetDiscountByCustomer

**Klient:** Firma oraz klient indywidualny. Ma najmniejsze możliwości w bazie danych. Zakres dostępu został obniżony do koniecznego użycia bazy przy zamawianiu, rezerwowaniu i korzystaniu z restauracji:

- Zamówienia:
  - Wyświetlenie informacji o danym zamówieniu (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data)
    - FOrderDetail
  - Wyświetlenie informacji o zamówieniach (co i ile zamówiono, cena jednostkowa produktów, rabaty, koszt zamówienia, koszt zamówienia po odjęciu rabatów, data)
    - VOrdersDetails
- Rezerwacja:
  - Utworzenie rezerwacji wraz z zamówieniem
    - PCreateReservationIndividualClient
  - Utworzenie rezerwacji (bez lub z zamówieniem)
    - PCreateReservationCompany
  - Wyświetlenie informacji o rezerwacjach firmowych, pracownikach imiennie oraz firmach
    - VCompanyReservation
- Menu:
  - Wyświetlenie menu wraz z cenami oraz kategoriami produktów
    - VMenu
  - Wyświetlenie menu wraz z cenami oraz kategoriami produktów na dany dzień
    - FGetMenuProductsByDay
  - Wyświetlenie informacji o produktach
    - VProduct

## Raporty

System umożliwia generowanie następujących raportów dostępnych jako widoki w bazie danych:

1. Raport dotyczący zamówień złożonych w ostatnim tygodniowo
2. Raport dotyczący zamówień złożonych w ostatnim miesiącu
3. Raport dotyczący zamówień złożonych w ostatnim roku
4. Raport dotyczący rezerwacji złożonych w ostatnim tygodniowo
5. Raport dotyczący rezerwacji złożonych w ostatnim miesiącu
6. Raport dotyczący rezerwacji złożonych w ostatnim roku
7. Raport dotyczący pozycji w menu z ostatniego tygodnia
8. Raport dotyczący pozycji w menu z ostatniego miesiąca
9. Raport dotyczący zajętych stolików z ostatniego tygodnia
10. Raport dotyczący zajętych stolików z ostatniego miesiąca
11. Raport dotyczący przyznanych zniżek dla klientów indywidualnych w ostatnim tygodniu
12. Raport dotyczący przyznanych zniżek dla klientów indywidualnych w ostatnim miesiącu

## Informacje o generowanych danych

Dane zostały wygenerowane za pomocą napisanych programów w pythonie. Poszczególne informacje były odpowiednio dobrane lub generowane w sposób pseudolosowy. Również użyto pomocniczej strony internetowej:

<https://generator danych.testery.pl/?fbclid=IwAR0ZFRw-B2rZXmPAruaNEbTSVGYTgncW6T8Fc1eDpZizg9G9s2HcfDnryxM>.