System zarządzania restauracją

Database model documentation

Spis treści

1. Wymagania funkcjonalne

1.1 Wymagania ze strony restauracji

- dodanie nowego klienta - system umożliwia ręczne dodanie klienta do bazy przez pracownika firmy

1.2 Potrawy

- dodanie nowej potrawy do bazy wszystkich potraw
- dana potrawa jest oznaczana jako niedostępna

1.3 Owoce morza

- sprawdzenie czy data zamówienia jest odpowiednia
- czy jest odpowiedni dzień
    - W dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Z uwagi na indywidualny import takie zamówienie winno być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.

1.4 Menu

- dodanie potrawy do menu
    - przy próbie dodania potrawy do menu sprawdzane jest czy potrawa nie znajduje się obecnie w menu oraz czy wycofano potrawę
- wycofanie potrawy z menu
    - data wycofania potrawy ustawiana jest na aktualną
- zwracanie listy potraw w obecnym menu
    - zwracana jest lista pozycji menu, które mają datę końca pustą lub większą lub równą dzisiejszej i datę początku mniejszą lub równą dzisiejszej
- informowanie o konieczności zmian w menu
    - jeśli istnieje menu wcześniejsze(czyli endDate poprzedniego + 1 dzień = startDate aktualnego menu) od aktualnie dodanego to system sprawdza czy przynajmniej połowa pozycji została zmieniona w aktualnym względem poprzedniego. Tak samo działa to jeśli istnieje menu przyszłe(czyli endDate aktualnego + 1 dzień = startDate przyszłego menu)

1.5 Rezerwacje

- dodanie nowego stolika do bazy stolików
- zwrócenie wolnych stolików w danym terminie
    - w celu otrzymania numerów oraz liczby miejsc wolnych stolików posługujemy się dwiema wartościami:
        - data zajęcia
        - data zwolnienia
        - Jeżeli stolik jest wolny to:
            - Data zajęcia i zwolnienia nie są zdefiniowane lub
            - Data zajęcia jest większa od zadanego terminu, a data zwolnienia jest mniejsza od zadanego terminu.
- kontrola dostępności stolików przy dokonywaniu rezerwacji

    - przy dokonywaniu rezerwacji sprawdzana jest dostępność stolików w danym terminie w sposób jak powyżej z dodatkowym warunkiem sprawdzającym czy sumaryczna liczba miejsc przy wolnych stolikach jest większa lub równa ilości osób w rezerwacji
- oznaczenie stolika jako wolny / zajęty (zapisywana jest odpowiednio data zwolnienia / zajęcia stolika)
- zmiana statusu rezerwacji klienta indywidualnego, w tym możliwość odrzucenia rezerwacji:
    - po dokonaniu rezerwacji przez klienta za pomocą formularza www zostaje ona oznaczona automatycznie jako niepotwierdzona, pracownik restauracji może potwierdzić lub odrzucić rezerwację
    - potwierdzoną rezerwację pracownik może anulować z powodu np. nieprawidłowego statusu opłacenia zamówienia
    - Klient sam może anulować zamówienie na swoje życzenie

1.6 Zamówienia

- zamówienie może być utworzone dla klienta zarejestrowanego w systemie do bazy zamówień dodawane jest nowe zamówienie zawierające wybrane z obecnego menu potrawy, obecną datę jako datę zamówienia, jeżeli zamówienie jest na wynos to zawiera także datę odbioru, jeżeli zamówienie jest na miejscu to wtedy zawiera także stolik
- jeśli zamówienie obejmuje potrawy zawierające owoce morza, to data realizacji zamówienia może wypadać tylko w czwartek, piątek lub sobotę, przy czym data zamówienia musi być najpóźniej poniedziałkiem poprzedzającym datę realizacji (czyli datę rezerwacji dla zamówień na miejscu lub datę odbioru dla zamówień na wynos)

1.7 Rabaty

1. `typ 1` - Po realizacji ustalonej liczby zamówień Z1 (przykładowo Z1=10) za co najmniej określoną kwotę K1 (np. 30 zł każde zamówienie): R1% (np. 3%) zniżki na wszystkie zamówienia;
2. `typ 2` - Po realizacji zamówień za łączną kwotę K2 (np. 1000 zł): jednorazowa zniżka R2% (np. 5%) na zamówienia złożone przez D1 dni (np. D1 = 7), począwszy od dnia przyznania zniżki (zniżki nie łączą się).

- Dodanie/ usunięcie zmiennych dotyczących rabatów
- Trigger sprawdzający czy klient nabył prawo do zniżki
- Zdecydowanie która zniżka jest ważniejsza w hierarchii i wybranie jej
- zapisanie informacji o rabacie przyznanym danemu klientowi
- obliczenie rabatu dla danego klienta w trakcie składania zamówienia

1.8 Raporty

Raporty jako widoki

- generowanie listy potraw do przygotowania w danym czasie
- generowanie raportów (miesięcznych i tygodniowych dotyczących rezerwacji stolików, rabatów, menu, produktów(tutaj jeszcze dziennie))
- generowanie statystyk dla klientów indywidualnych oraz firm
- generowanie raportów dotyczących zamówień dla wszystkich klientów oraz rabatów dla klienta indywidualnego
- generowanie faktury za zamówienie dla klientów

- generowanie raportów dotyczących wydatków w danym roku, miesiącu, tygodniu

1.9 Raporty jako funkcje

- Generowanie raportów dla klientów indywidualnych i firm w podam okresie czasu
- Generowanie raportów o zniżkach dla danego klienta
- Generowanie raportu dla określonego menu
- Generowanie raportu dla określonego zamówienia
- Generowanie raportu dla rezerwacji (miesięcznych i tygodniowych) w podanym okresie czasu
- Generowanie raportu dla stolików (miesięcznych i tygodniowych) w podanym okresie czasu

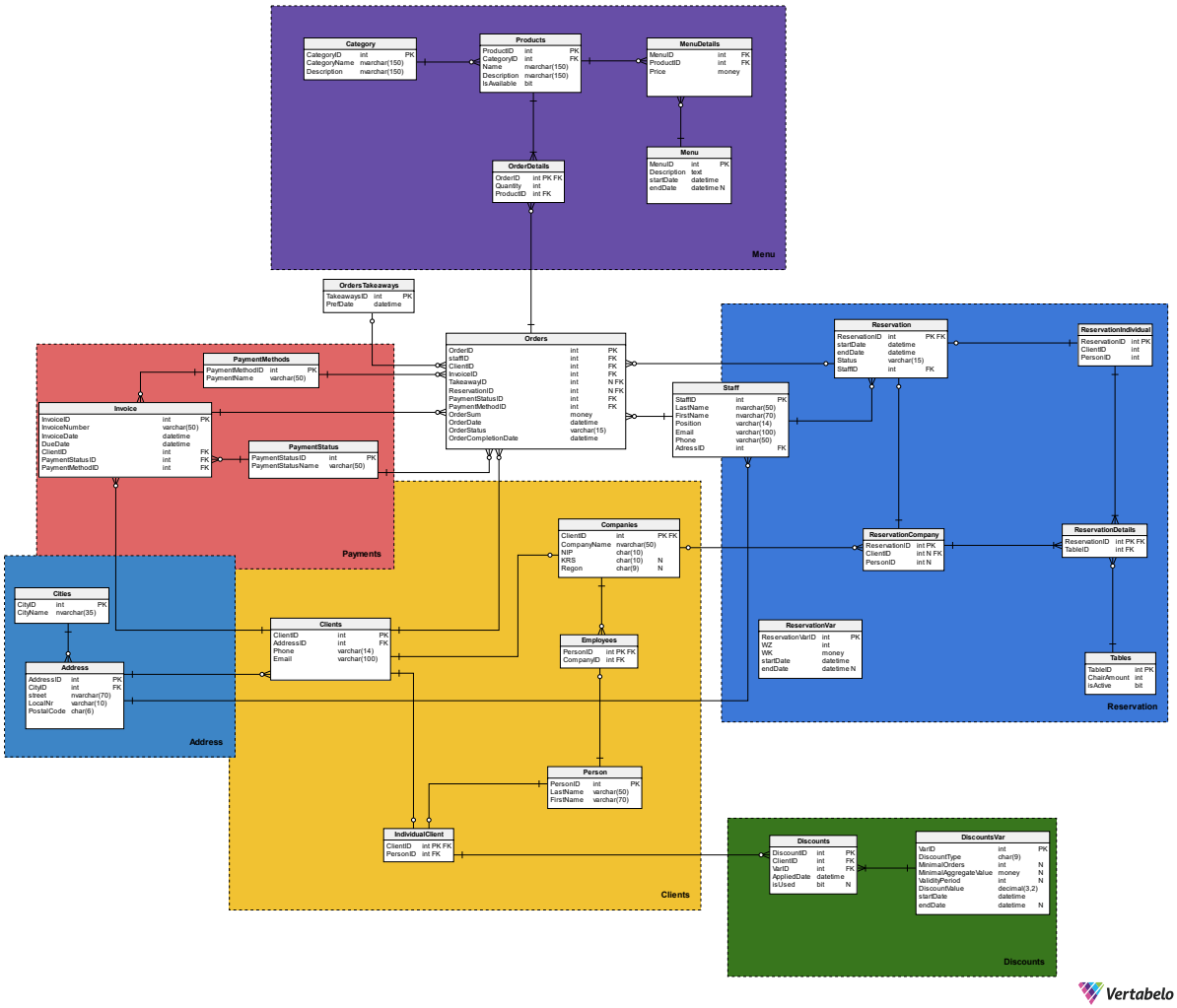
2. Opis modelu

Model name: `System zarządzania restauracją`

Version: `2.4`

Database engine: `Microsoft SQL Server`

2.1. Schemat bazy



3. Opisy Tabel

3.1. Tabela Orders

Przechowuje informacje o zamówieniach.

- Klucz główny: OrderID
- Klucze obce: ClientID (do tabeli Clients), TakeawayID (do tabeli OrdersTakeaways), ReservationID (do tabeli Reservation), PaymenttStatusID (do tabeli PaymentStatus), staffID (do tabeli Staff) , PaymentMethodID(do tabeli PaymentMethods)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
OrderID	int	Not null	Numer ID zamówienia
ClientID	int	Not null	Numer ID klienta
TakeawayID	int	null	Numer ID na wynos
InvoiceID	Int	Null	Numer Faktury do zamówienia
ReservationID	int	null	Numer ID rezerwacji
PaymentStatusID	int	Not null	Numer ID statusu zapłaty
PaymentMethodID	int	Not null	Numer ID metody płatności
staffID	int	Not null	Numer ID personelu restauracji
OrderSum	money	Not null	Wartość zamówienia
OrderDate	datetime	Not null	Data złożenia zamówienia
OrderCompletionDate	datetime	null	Data skompletowania zamówienia
OrderStatus	varchar(15)	Not null	Status zamówienia - Pending - Accepted - Completed - Denied - Picked - Cancelled

```
CREATE TABLE Orders (
    OrderID int  NOT NULL IDENTITY (1,1),
    ClientID int  NOT NULL,
    TakeawayID int  NULL,
    InvoiceID int  NULL,
    ReservationID int  NULL,
    PaymentStatusID int  NOT NULL,
    PaymentMethodID int  NOT NULL,
    staffID int  NOT NULL,
    OrderSum money  NOT NULL check ( OrderSum >= 0 ),
    OrderDate datetime  NOT NULL default getdate(),
    OrderCompletionDate datetime  NULL ,
    OrderStatus varchar(15) NOT NULL check (OrderStatus in ('pending', 'accepted',
'completed', 'denied', 'picked', 'cancelled')),
    CONSTRAINT validDateOrders check ( (OrderCompletionDate is null)),
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);
```

3.2. Tabela OrdersTakeaways

Przechowuje informacje o zamówieniu na wynos

- Klucz główny: TakeawaysID

Nazwa kolumny	Typy Danych	Czy null	Co przechowuje
TakeawaysID	int	Not null	Numer ID zamówienia na wynos
PrefDate	datetime	Not null	Preferowaną date odbioru zamówienia

```
CREATE TABLE OrdersTakeaways (
    TakeawaysID int  NOT NULL IDENTITY (1,1),
    PrefDate datetime  NOT NULL check (PrefDate >= getdate()),
    CONSTRAINT OrdersTakeaways_pk PRIMARY KEY  (TakeawaysID)
);
```

3.3. Tabela Reservation

Przechowuje informacje o aktualnych rezerwacjach

- Klucz główny: ReservationID
- Klucze obce: StaffID (do tabeli Staff)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ReservationID	int	Not null	Numer ID rezerwacji

startDate	datetime	Not null	Data rozpoczęcia rezerwacji
endDate	datetime	Not null	Data zakończenia rezerwacji
Status	Varchar(15)	Not null	Status danej rezerwacji - Pending - Accepted - Denied - Cancelled - Waiting
StaffID	int	Not null	Numer ID personelu

```
CREATE TABLE Reservation (
    ReservationID int  NOT NULL  IDENTITY (1,1),
    startDate datetime  NOT NULL,
    endDate datetime  NOT NULL ,
    Status varchar(15)  NOT NULL default 'waiting',
    StaffID int  NOT NULL,
    constraint validStatus
check (Status in ('pending', 'accepted', 'denied', 'cancelled', 'waiting')),
    CONSTRAINT validDateReservation  check(startDate < endDate),
    CONSTRAINT Reservation_pk PRIMARY KEY  (ReservationID)
);
```

3.4. Tabela ReservationIndividual

Przechowuje informacje o rezerwacjach osób indywidualnych

- Klucz główny: ReservationID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ReservationID	int	Not null	Numer ID rezerwacji
ClientID	int	Not null	Numer ID klienta
PersonID	int	Not null	Numer ID osoby

```
CREATE TABLE ReservationIndividual (
    ReservationID int  NOT NULL,
    ClientID int  NOT NULL,
    PersonID int  NOT NULL,
    CONSTRAINT ReservationIndividual_pk PRIMARY KEY  (ReservationID)
);
```

3.5. Tabela ReservationCompany

Przechowuje informacje o rezerwacjach firmowych

- Klucz główny: ReservationID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ReservationID	int	Not null	Numer ID rezerwacji
ClientID	int	Not null	Numer ID klienta
PersonID	int	Not null	Numer ID osoby

```
CREATE TABLE ReservationCompany (
    ReservationID int  NOT NULL,
    ClientID int  NULL,
    PersonID int  NULL,
    CONSTRAINT ReservationCompany_pk PRIMARY KEY  (ReservationID)
);
```

3.6. Tabela ReservationDetails

Łączy rezerwacje z stolikami dla nich

- Klucz główny: brak
- Klucze obce: TableID (do tabeli Tables)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ReservationID	int	Not null	Numer ID rezerwacjio
TableID	int	Not null	Numer ID stolika

```
CREATE TABLE ReservationDetails (
    ReservationID int  NOT NULL,
    TableID int  NOT NULL
);
```

3.7. Tabela Tables

Przechowuje informacje o stolikach

- Klucz główny: TableID
- Klucze obce: TableID (do tabeli Tables)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
TableID	int	Not null	Numer ID stolika
ChairAmount	int	Not null	Liczbe krzeseł
isActive	bit	Not null	Czy stolik jest aktywny. Czy nie stoi na zapleczu

```
CREATE TABLE Tables (
    TableID int  NOT NULL,
    ChairAmount int  NOT NULL check (ChairAmount >= 2),
    isActive bit  NOT NULL default 1,
    CONSTRAINT Tables_pk PRIMARY KEY  (TableID)
);
```

3.8. Tabela ReservationVar

Przechowuje informacje o zmiennych do rezerwacji

- Klucz główny: ReservationVarID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ReservationVarID	int	Not null	Numer ID zmiennej dotyczącej rezerwacji
WZ	int	Not null	Minimalna liczba zamówień potrzebna do rezerwację
WK	money	Not null	Minimalna kwota potrzebna do rezerwację
startDate	datetime	Not null	Data obowiązywania zmiennej
endDate	datetime	null	Data zakończenia obowiązywania zmiennej

```
CREATE TABLE ReservationVar (
    ReservationVarID int  NOT NULL IDENTITY (1,1),
    WZ int  NOT NULL check ( WZ > 0 ),
    WK money NOT NULL check (WK > 0),
    startDate datetime  NOT NULL,
    endDate datetime  NULL,
    CONSTRAINT check(startDate < endDate or endDate is NULL),
    CONSTRAINT ReservationVar_pk PRIMARY KEY  (ReservationVarID)
);
```

3.9. Tabela Clients

Przechowuje informacje o klientach

- Klucz główny: ClientID
- Klucze obce: AddressID (do tabeli Address)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ClientID	int	Not null	Numer ID klienta
AddressID	int	Not null	Numer ID adressów
Phone	varchar(14)	Not null	Numer telefonu
Email	varchar(100)	Not null	Adres email

```
CREATE TABLE Clients (
    ClientID int  NOT NULL IDENTITY (1,1),
    AddressID int  NOT NULL,
    Phone varchar(14)  NOT NULL UNIQUE
                                check (isnumeric(Phone) = 1 and len(Phone) >= 9),
    Email varchar(100)  NOT NULL UNIQUE check( Email like '%[@]%[.]%'),
    CONSTRAINT Clients_pk PRIMARY KEY  (ClientID)
);
```

3.10. Tabela Companies

Przechowuje informacje o klientach firmowych

- Klucz główny: ClientID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ClientID	int	Not null	Numer ID klienta

CompanyName	nvarchar(50)	Not null	Nazwe firmy
NIP	char(10)	Not null	Numer NIP
KRS	char(10)	null	Numer KRS
Regon	char(9)	null	Numer REGON

```
CREATE TABLE Companies (
    ClientID int  NOT NULL,
    CompanyName nvarchar(50)  NOT NULL UNIQUE,
    NIP char(10)  NOT NULL UNIQUE
        check(NIP like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    KRS char(10)  NULL
        check(KRS like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Regon char(9)  NULL
        check(Regon like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    CONSTRAINT Companies_pk PRIMARY KEY  (ClientID)
);
```

3.11. Tabela Employees

Przechowuje informacje o pracownikach danej firmy

- Klucz główny: PersonID
- Klucze obce: CompanyID (do tabeli Companies), PersonID (do tabeli Employees)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
PersonID	int	Not null	Numer ID osoby
CompanyID	int	Not null	Numer ID Firmy

```
CREATE TABLE Employees (
    PersonID int  NOT NULL,
    CompanyID int  NOT NULL,
    CONSTRAINT Employees_pk PRIMARY KEY  (PersonID)
);
```

3.12. Tabela Person

Przechowuje informacje o osobach

- Klucz główny: PersonID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
PersonID	int	Not null	Numer ID osoby
LastName	varchar(50)	Not null	Imie osoby
FirstName	varchar(70)	Not null	Nazwisko osoby

```
CREATE TABLE Person (
    PersonID int  NOT NULL,
    LastName varchar(50)  NOT NULL,
    FirstName varchar(70)  NOT NULL,
    CONSTRAINT Person_pk PRIMARY KEY  (PersonID)
);
```

3.13. Tabela IndividualClient

Przechowuje informacje o klientach indywidualnych

- Klucz główny: ClientID
- Klucze obce: PersonID (do tabeli Person)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ClientID	int	Not null	Numer ID klienta
PersonID	int	Not null	Numer ID osoby

```
CREATE TABLE IndividualClient (
    ClientID int  NOT NULL,
    PersonID int  NOT NULL,
    CONSTRAINT IndividualClient_pk PRIMARY KEY  (ClientID)
);
```

3.14. Tabela Category

Przechowuje informacje o kategoriach produktów

- Klucz główny: CategoryID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
CategoryID	int	Not null	Numer ID kategorii

CategoryName	nvarchar(50)	Not null	Nazwe kategorii
Description	nvarchar(150)	Not null	Opis Kategorii

```
CREATE TABLE Category (
    CategoryID int  NOT NULL IDENTITY (1,1),
    CategoryName nvarchar(50)  NOT NULL,
    Description nvarchar(150)  NOT NULL,
    CONSTRAINT Category_pk PRIMARY KEY  (CategoryID)
);
```

3.15. Tabela Products

Przechowuje informacje o produktach oferowanych przez restauracje

- Klucz główny: ProductID
- Klucze obce: CategoryID (do tabeli Category)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
ProductID	int	Not null	Numer ID produktu
CategoryID	int	Not null	Numer ID kategorii
Name	nvarchar(50)	Not null	Nazwe produktu
Description	nvarchar(150)	Not null	Opis produktu
IsAvailable	Bit	Not null	Mówimy o tym czy dany produkt jest dostępny. Chodzi o produkty np sezonowe

```
CREATE TABLE Products (
    ProductID int  NOT NULL IDENTITY (1,1),
    CategoryID int  NOT NULL,
    Name nvarchar(50)  NOT NULL,
    Description nvarchar(150)  NOT NULL default 'brak opisu' ,
    IsAvailable bit NOT NULL default 1,
    CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);
```

3.16. Tabela Menu

Przechowuje informacje o menu oferowanym przez restauracje w danym okresie

- Klucz główny: MenuID
- Klucze obce: MenuID (do tabeli MenuDetails)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
MenuID	int	Not null	Numer ID menu
Description	Varchar(max)	Not null	Opis danego menu
startDate	datetime	Not null	Data od kiedy obowiązuje menu
endDate	datetime	null	Data do kiedy obowiązuje menu. Może być tak, że menu nie ma końca obowiązywania

```
CREATE TABLE Menu (
    MenuID int  NOT NULL,
    Description varchar(max)  NOT NULL  DEFAULT 'Brak opisu',
    startDate datetime  NOT NULL DEFAULT getdate(),
    endDate datetime  NULL,
    CONSTRAINT validDateMenu
        check((startDate < endDate and endDate is not null) or endDate is null),
    CONSTRAINT Menu_pk PRIMARY KEY  (MenuID)
);
```

3.17. Tabela OrderDetails

Przechowuje informacje o szczegółach danego zamówienia

- Klucz główny: brak
- Klucze obce: ProductID (do tabeli Products), OrderID (do tabeli Orders)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
OrderID	int	Not null	Numer ID zamówienia
Quantity	int	Not null	Ilość danego produktu w danym zamówieniu
ProductID	int	Not null	Numer ID produktu

```
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    Quantity int  NOT NULL check ( Quantity > 0 ),
    ProductID int  NOT NULL
);
```

3.18. Tabela Discounts

Przechowuje informacje o zniżka dla klientów indywidualnych

- Klucz główny: DiscountID
- Klucze obce: ClientID (do tabeli IndividualClient), VarID (do tabeli DiscountsVar)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
DiscountID	int	Not null	Numer ID zniżki
ClientID	int	Not null	Numer ID klienta
VarID	int	Not null	Numer ID zmiennych dotyczących tego zamówienia
AppliedDate	datetime	Not null	Data od kiedy obowiązuje zniżka
isUsed	Bit	Null	Jeśli jest to jednorazowa zniżka to posiada wartości czy jest już nałożona na klienta

```
CREATE TABLE Discounts (
    DiscountID int  NOT NULL IDENTITY (1,1),
    ClientID int  NOT NULL,
    VarID int  NOT NULL,
    AppliedDate datetime  NOT NULL ,
    isUsed bit NULL default 0,
    CONSTRAINT Discounts_pk PRIMARY KEY  (DiscountID)
);
```

3.19. Tabela DiscountsVar

Przechowuje informacje o zmiennych dotyczących zniżek dla klientów indywidualnych

- Klucz główny: VarID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje

VarID	int	Not null	Numer ID zmiennej
DiscountType	char(9)	Not null	Typ zniżki: - Tymczasowa - Trwała
MinimalOrders	int	null	Najmniższa liczba zamówień aby zniżka zaczęła obowiązywać. Dotyczy zniżki tymczasowej
MinimalAggregateValue	money	null	Najmniższa sumaryczna kwota wydana na zamówienia
ValidityPeriod	int	null	Ilość dni w jakich dotyczy zniżka. Dotyczy zniżki tymczasowej
DiscountValue	decimal(3,2)	Not null	Wartość zniżki
startDate	datetime	Not null	Data od kiedy dane zmienne obowiązywały
endDate	datetime	null	Data kiedy zmienne skończyły obowiązywać

```
CREATE TABLE DiscountsVar (
    VarID int  NOT NULL IDENTITY (1,1),
    DiscountType char(9)  NOT NULL
                        check (DiscountType in ('Permanent', 'Temporary')),
    MinimalOrders int  NULL,
    MinimalAggregateValue money  NULL,
    ValidityPeriod int  NULL,
    DiscountValue decimal(3,2)  NOT NULL
                        CHECK ( DiscountValue >= 0 and DiscountValue <= 1 ),
    startDate datetime  NOT NULL DEFAULT getdate(),
    endDate datetime  NULL check(endDate IS NULL or startDate < endDate),
    CONSTRAINT DiscountsVar_pk PRIMARY KEY  (VarID)
);
```

3.20. Tabela PaymentStatus

Przechowuje informacje o statusie opłat zamówienia

- Klucz główny: PaymentStatusID
- Klucze obce: PaymentMethodID (dotyczy tabeli PaymentMethods)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje

PaymentStatusID	int	Not null	Numer ID statusu płatności zamówienia
PaymentStatusName	varchar(50)	Not null	Nazwa statusu płatności

```
-- Table: PaymentStatus
CREATE TABLE PaymentStatus (
    PaymentStatusID int  NOT NULL IDENTITY (1,1),
    PaymentStatusName varchar(50)  NOT NULL default 'Unpaid',
    CONSTRAINT PaymentStatus_pk PRIMARY KEY  (PaymentStatusID)
);
```

3.21. Tabela Invoice

Przechowuje informacje o fakturach

- Klucz główny: InvoiceID
- Klucze obce: PaymentStatusID (dotyczy tabeli PaymentStatus), ClientID (dotyczy tabeli Clients), PaymentMethodID(do tabeli PaymentMethod)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
InvoiceID	int	Not null	Numer ID fakttury
InvoiceNumber	varchar(50)	Not null	Numer faktury
InvoiceDate	datetime	Not null	Data wystawienia faktury
DueDate	datetime	Not null	Termin zapłaty
ClientID	int	Not null	Numer ID klienta
PaymentStatusID	int	Not null	Numer ID statusu płatności faktury
PaymentMethodID	int	Not null	Numer ID metody płatności faktury

```
CREATE TABLE Invoice (
    InvoiceID int  NOT NULL IDENTITY (1,1),
    InvoiceNumber varchar(50)  NOT NULL UNIQUE,
    InvoiceDate datetime  NOT NULL,
    DueDate datetime  NOT NULL,
    ClientID int  NOT NULL,
    PaymentStatusID int  NOT NULL,
    CONSTRAINT Invoice_pk PRIMARY KEY  (InvoiceID)
);
```

3.22. Tabela Cities

Przechowuje informacje o miastach

- Klucz główny: CityID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
CityID	int	Not null	Numer ID miasta.
CityName	nvarchar(35)	Not null	Nazwa miasta

```
CREATE TABLE Cities (
    CityID INT  NOT NULL IDENTITY (1,1),
    CityName nvarchar(35)  NOT NULL,
    CONSTRAINT Cities_pk PRIMARY KEY  (CityID)
);
```

3.23. Tabela Address

Przechowuje informacje o Adresach

- Klucz główny: AddressID
- Klucze obce: CityID (dotyczy tabeli Cities)

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
AddressID	int	Not null	Numer ID adresu
CityID	char(3)	Not null	Numer ID miasta. Opis w tabeli Cities.
street	nvarchar(70)	Not null	Nazwa ulicy
LocalNr	varchar(10)	Not null	Numer Domu wraz z np. 10A itp.
PostalCode	char(6)	Not null	Kod pocztowy w postaci XX-XXX.

```
CREATE TABLE Address (
    AddressID int  NOT NULL IDENTITY (1,1),
    CityID INT NOT NULL,
    street nvarchar(70)  NOT NULL,
    LocalNr varchar(10)  NOT NULL check(localNr like '[0-9]%'),
    PostalCode char(6)  NOT NULL
                check(PostalCode like '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT Address_pk PRIMARY KEY  (AddressID)
);
```

3.24. Tabela PaymentMethods

Przechowuje informacje o metodach płatności

- Klucz główny: PaymentMethodID

Nazwa kolumny	Typy danych	Czy null	Co przechowuje
PaymentMethodID	int	Not null	Numer ID metody płatności
PaymentName	varchar(50)	Not null	Nazwa metody

```
CREATE TABLE PaymentMethods (
    PaymentMethodID int  NOT NULL IDENTITY (1,1),
    PaymentName varchar(50)  NOT NULL,
    CONSTRAINT PaymentMethods_pk PRIMARY KEY  (PaymentMethodID)
);
```

3.25. Table Staff

Przechowuje informacje o personelu

- Klucz główny: `StaffID`

Column name	Type	Properties	Description
`StaffID`	`int`	Not null	Numer ID pracownika
`LastName`	`nvarchar(50)`	Not null	Nazwisko pracownika
`FirstName`	`nvarchar(70)`	Not null	Imię pracownika
`Position`	`varchar(14)`	Not null	Stanowisko jakie pracownik zajmuje w firmie
`Email`	`varchar(100)`	Not null	Adres email

Phone	varchar(50)	Not null	Numer telefonu
AdressID	int	Not null	Numer ID adresu

```
CREATE TABLE Staff (
    StaffID int  NOT NULL IDENTITY (1,1),
    LastName nvarchar(50)  NOT NULL,
    FirstName nvarchar(70)  NOT NULL,
    Position varchar(50)  NOT NULL,
    Email varchar(100)  NOT NULL UNIQUE check( Email like '%[@]%[.]%'),
    Phone varchar(14)  NOT NULL UNIQUE
         check( isnumeric(Phone) = 1 and len(Phone) >= 9),
    AddressID int  NOT NULL ,
    CONSTRAINT Staff_pk PRIMARY KEY  (StaffID)
);
```

3.26 Table MenuDetails

Przechowuje pomocnicze informacje dla Menu

Klucz główny: `MenuID`

Column name	Type	Properties	Description
MenuID	int	Not null	Numer ID Menu
ProductID	int	Not null	Numer ID Produktu
Price	money	Not null	Cena produktu w danym menu

```
CREATE TABLE MenuDetails (
    MenuID int  NOT NULL,
    ProductID int  NOT NULL,
    Price money  NOT NULL CHECK ( Price > 0 )
);
```

4. Referencje

4.1. Reference Products_Category

Category	0..*	Products
CategoryID	<->	CategoryID

```
alter table Products
    add constraint Products_Category
        foreign key (CategoryID) references Category
            on update cascade
```

4.2. Reference MenuDetails_Products

Products	0..*	MenuDetails
ProductID	<->	ProductID

```
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID)
    ON UPDATE  CASCADE;
```

4.3 Reference MenuDetails_Menu

Menu	0..*	MenuDetails
MenuID	<->	MenuID

```
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Menu
    FOREIGN KEY (MenuID)
    REFERENCES Menu (MenuID)
    ON UPDATE  CASCADE;
```

4.4. Reference OrderDetails_Products

Products	1..*	OrderDetails
ProductID	<->	ProductID

```
alter table OrderDetails
    add constraint OrderDetails_Products
        foreign key (ProductID) references Products
            on update cascade
```

4.5. Reference Orders_OrdersTakeaways

OrdersTakeaways	0..*	Orders
TakeawaysID	<->	TakeawayID

```
alter table Orders
    add constraint Orders_OrdersTakeaways
        foreign key (TakeawayID) references OrdersTakeaways
            on update cascade
```

4.6. Reference OrderDetails_Orders

Orders	0..*	OrderDetails
OrderID	<->	OrderID

```
alter table OrderDetails
    add constraint OrderDetails_Orders
        foreign key (OrderID) references Orders
            on update cascade
```

4.7. Reference Discounts_DiscountsVar

DiscountsVar	1..*	Discounts
VarID	<->	VarID

```
alter table Discounts
    add constraint Discounts_DiscountsVar
        foreign key (VarID) references DiscountsVar
            on update cascade
```

4.8. Reference Discounts_IndividualClient

IndividualClient	0..*	Discounts
ClientID	<->	ClientID

```
alter table Discounts
    add constraint Discounts_IndividualClient
        foreign key (ClientID) references IndividualClient
            on update cascade
```

4.9. Reference Clients_IndividualClient

Clients	0..1	IndividualClient
ClientID	<->	ClientID

```
alter table IndividualClient
    add constraint Clients_IndividualClient
        foreign key (ClientID) references Clients
            on update cascade
            on delete cascade
```

4.10. Reference IndividualClient_Person

Person	0..1	IndividualClient
PersonID	<->	PersonID

```
alter table IndividualClient
    add constraint IndividualClient_Person
        foreign key (PersonID) references Person
            on update cascade
```

4.11. Reference Employees_Person

Person	0..1	Employees

PersonID	<->	PersonID

```
alter table Employees
    add constraint Employees_Person
        foreign key (PersonID) references Person
            on update cascade
```

4.12. Reference Companies_Clients

Clients	0..1	Companies
ClientID	<->	ClientID

```
alter table Companies
    add constraint Companies_Clients
        foreign key (ClientID) references Clients
            on update cascade
            on delete cascade
```

4.13. Reference Employees_Companies

Companies	0..*	Employees
ClientID	<->	CompanyID

```
alter table Employees
    add constraint Employees_Companies
        foreign key (CompanyID) references Companies
            on update cascade
```

4.14. Reference Orders_Clients

Clients	0..*	Orders
ClientID	<->	ClientID

```
alter table Orders
    add constraint Orders_Clients
        foreign key (ClientID) references Clients
            on update cascade
```

4.15. Reference ReservationCompany_Companies

Companies	0..*	ReservationCompany
ClientID	<->	ClientID

```
alter table ReservationCompany
    add constraint ReservationCompany_Companies
        foreign key (ClientID) references Companies
            on update cascade
```

4.16. Reference Reservation_ReservationCompany

ReservationCompany	0..1	Reservation
ReservationID	<->	ReservationID

```
ALTER TABLE ReservationCompany ADD CONSTRAINT ReservationCompany_Reservation
    FOREIGN KEY (ReservationID)
    REFERENCES Reservation (ReservationID);
```

4.17. Reference Orders_Reservation

Reservation	0..*	Orders
ReservationID	<->	ReservationID

```
alter table Orders
    add constraint Orders_Reservation
        foreign key (ReservationID) references Reservation
            on update cascade
```

4.18. Reference ReservationDetails_Tables

Tables	0..*	ReservationDetails
TableID	<->	TableID

```
alter table ReservationDetails
    add constraint ReservationDetails_Tables
        foreign key (TableID) references Tables
            on update cascade
```

4.19. Reference ReservationDetails_ReservationCompany

ReservationCompany	1..*	ReservationDetails
ReservationID	<->	ReservationID

```
alter table ReservationDetails
    add constraint ReservationDetails_ReservationCompany
        foreign key (ReservationID) references ReservationCompany
            on update cascade
```

4.20. Reference ReservationDetails_ReservationIndividual

ReservationIndividual	1..*	ReservationDetails
ReservationID	<->	ReservationID

```
alter table ReservationDetails
    add constraint ReservationDetails_ReservationIndividual
        foreign key (ReservationID) references ReservationIndividual
            on update cascade
```

4.21. Reference Reservation_ReservationIndividual

ReservationIndividual	0..1	Reservation
ReservationID	<->	ReservationID

```
ALTER TABLE ReservationIndividual ADD CONSTRAINT ReservationIndividual_Reservation
    FOREIGN KEY (ReservationID)
    REFERENCES Reservation (ReservationID);
```

4.22. Reference Orders_PaymentStatus

PaymentStatus	0..*	Orders
PaymentStatusID	<->	PaymentStatusID

```
alter table Orders
    add constraint Orders_PaymentStatus
        foreign key (PaymentStatusID) references PaymentStatus
            on update cascade
```

4.23. Reference Invoice_PaymentStatus

PaymentStatus	0..*	Invoice
PaymentStatusID	<->	PaymentStatusID

```
alter table Invoice
    add constraint Invoice_PaymentStatus
        foreign key (PaymentStatusID) references PaymentStatus
            on update cascade
```

4.24. Reference Invoice_Clients

Clients	0..*	Invoice
ClientID	<->	ClientID

```
alter table Invoice
    add constraint Invoice_Clients
        foreign key (ClientID) references Clients
            on update cascade
```

4.25. Reference Clients_Address

Address	0..*	Clients
AddressID	<->	AddressID

```
alter table Clients
    add constraint Clients_Address
        foreign key (AddressID) references Address
            on update cascade
```

4.26. Reference Address_Cities

Cities	0..*	Address
CityID	<->	CityID

```
alter table Address
    add constraint Address_Cities
        foreign key (CityID) references Cities
            on update cascade
```

4.27. Reference Orders_staff

Staff	0..*	Orders
StaffID	<->	staffID

```
alter table Orders
    add constraint Orders_staff
        foreign key (staffID) references Staff
            on update cascade
```

4.28. Reference Staff_Address

Address	0..*	Staff
AddressID	<->	AdressID

```
ALTER TABLE Staff ADD CONSTRAINT Staff_Address
    FOREIGN KEY (AddressID)
    REFERENCES Address (AddressID);
```

4.29. Reference Reservation_Staff

Staff	0..*	Reservation
StaffID	<->	StaffID

```
alter table Reservation
    add constraint Reservation_Staff
        foreign key (StaffID) references Staff
```

4.30. Reference Orders_PaymentMethod

```
ALTER TABLE Orders ADD CONSTRAINT Orders_PaymentMethods
    FOREIGN KEY (PaymentMethodID)
    REFERENCES PaymentMethods (PaymentMethodID);
```

4.31 Reference Invoice_PaymentMethod

```
ALTER TABLE Invoice ADD CONSTRAINT Invoice_PaymentMethods
    FOREIGN KEY (PaymentMethodID)
    REFERENCES PaymentMethods (PaymentMethodID);
```

4.32. Reference Orders_Invoice

```
-- Reference: Orders_Invoice (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Invoice
    FOREIGN KEY (InvoiceID)
    REFERENCES Invoice (InvoiceID);
```

5.Widoki

5.1. CurrentMenu

Aktualne menu wraz z datą obowiązywania.

```
CREATE VIEW dbo.CurrentMenu AS
    SELECT M1.MenuID, Price, Name,
           P.Description AS 'Product Description',
           M2.Description
           AS 'Menu Description', startDate,
           ISNULL(CONVERT(varchar(max), endDate, 120), 'Menu nie ma daty
końca') AS 'endDate'
    FROM MenuDetails M1
        INNER JOIN Products P ON P.ProductID = M1.ProductID
        INNER JOIN Menu M2 ON M1.MenuID = M2.MenuID
    WHERE ((getdate() >= startDate) AND (getdate() <= endDate))
       OR ((getdate() >= startDate) AND endDate IS NULL);
GO
```

5.2. CurrentReservationVars

Aktualnie obowiązujące zmienne opisujące warunki rezerwacji.

```
CREATE VIEW dbo.CurrentReservationVars
AS
    SELECT
        WZ AS [Minimal number of orders],
        WK AS [Minimal value for orders],
        startDate,
        isnull(
            CONVERT(varchar(20), endDate, 120),
            'Obowiązuje zawsze'
        ) AS 'Koniec daty obowiązywania zmiennej'
    FROM
        ReservationVar
    WHERE
        (
            (getdate() >= startDate)
            AND (getdate() <= endDate)
        )
        OR (
            (getdate() >= startDate)
            AND endDate IS NULL
        );
GO
```

5.3. UnPaidInvoicesIndividuals

Nieopłacone faktury klientów indywidualnych.

```
CREATE VIEW dbo.UnPaidInvoicesIndividuals
AS
    SELECT
        InvoiceNumber AS [Numer faktury],
        InvoiceDate AS [Data wystawienia],
        DueDate AS [Data terminu zaplaty],
        concat(LastName, ' ', FirstName) AS [Dane],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode
    FROM Invoice I
        INNER JOIN Clients C ON C.ClientID = I.ClientID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN IndividualClient IC ON C.ClientID = IC.ClientID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
        INNER JOIN Cities C2 ON C2.CityID = A.CityID
        INNER JOIN PaymentStatus PS ON I.PaymentStatusID =
PS.PaymentStatusID
    WHERE
        LOWER(PaymentStatusName) LIKE 'unpaid';
GO
```

5.4. UnPaidInvoicesCompanies

Nieopłacone faktury firm.

```
CREATE VIEW dbo.UnPaidInvoicesCompanies
AS
    SELECT
        InvoiceNumber AS [Numer faktury],
        InvoiceDate AS [Data wystawienia],
        DueDate AS [Data terminu zaplaty],
        CompanyName,
        NIP,
        isnull(KRS, 'Brak') AS [KRS],
        isnull(Regon, 'Brak') AS [Regon],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode
    FROM Invoice
        INNER JOIN Clients C ON C.ClientID = Invoice.ClientID
        INNER JOIN Companies CO ON CO.ClientID = C.ClientID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN Cities C2 ON C2.CityID = A.CityID
        INNER JOIN PaymentStatus PS ON Invoice.PaymentStatusID =
PS.PaymentStatusID
    WHERE
        (LOWER(PaymentStatusName) LIKE 'Unpaid');
GO
```

5.5. WithdrawnProducts

Wycofane produkty.

```
CREATE VIEW dbo.WithdrawnProducts AS
    SELECT
        Name,
        P.Description,
        C.CategoryName
    FROM Products P
        INNER JOIN Category C ON C.CategoryID = P.CategoryID
    WHERE
        P.IsAvailable = 0
GO
```

5.6. ActiveProducts

Aktualnie dostępne produkty.

```
CREATE VIEW dbo.ActiveProducts
AS
    SELECT Name, P.Description, C.CategoryName
    FROM Products P
        INNER JOIN Category C ON C.CategoryID = P.CategoryID
    WHERE
        P.IsAvailable = 1
GO
```

5.7. Activetables

Stoliki dostępne dla klientów.

```
CREATE VIEW dbo.ActiveTables
AS
    SELECT
        TableID,
        ChairAmount
    FROM Tables
    WHERE
        isActive = 1
GO
```

5.8. WithdrawnTables

Stoliki niedostępne dla klientów.

```
CREATE VIEW dbo.[WithdrawnTables]
AS
    SELECT
        TableID,
        ChairAmount
    FROM Tables
        WHERE TableID not in (SELECT TableID FROM ActiveTables)
GO
```

5.9. Not reserved tables

Niezarezerwowane stoliki.

```
CREATE VIEW dbo.[Not reserved Tables]
AS
        SELECT
            TableID,
            ChairAmount
        FROM Tables
        WHERE
            TableID NOT IN(
                SELECT
                    ReservationDetails.TableID
                FROM
                    ReservationDetails
                    INNER JOIN ReservationCompany RC ON RC.ReservationID =
ReservationDetails.ReservationID
                    INNER JOIN Reservation R2 ON RC.ReservationID =
R2.ReservationID
                WHERE
                    (getdate() >= startDate)
                    AND (getdate() <= endDate)
                    AND (
                        STATUS NOT LIKE 'cancelled'
                        AND STATUS NOT LIKE 'denied'
                    )
                    AND isActive = 1
            ) AND isActive = 1
    UNION
        SELECT
            TableID,
            ChairAmount
        FROM Tables
        WHERE
            TableID NOT IN(
                SELECT
                    ReservationDetails.TableID
                FROM
                    ReservationDetails
                    INNER JOIN ReservationIndividual RC ON
```

```
RC.ReservationID = ReservationDetails.ReservationID
                    INNER JOIN Reservation R2 ON RC.ReservationID =
R2.ReservationID
                WHERE
                    (getdate() >= startDate)
                    AND (getdate() <= endDate)
                    AND (
                        STATUS NOT LIKE 'cancelled'
                        AND STATUS NOT LIKE 'denied'
                    )
                    AND isActive = 1
            ) AND isActive = 1
GO
```

5.10 TablesWeekly

Tygodniowy raport o stolikach: ilość rezerwacji konkretnych stolików.

```
CREATE VIEW dbo.TablesWeekly
AS
        SELECT
            YEAR(R2.StartDate) AS year,
            DATEPART(iso_week, R2.StartDate) AS week,
            T.TableID AS table_id,
            T.ChairAmount AS table_size,
            COUNT(RD.TableID) AS how_many_times_reserved
        FROM Tables T
            INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
            INNER JOIN ReservationIndividual RI ON RI.ReservationID =
RD.ReservationID
            INNER JOIN Reservation R2 ON RD.ReservationID =
R2.ReservationID
        WHERE
            (
                LOWER(STATUS) NOT LIKE 'cancelled'
                AND LOWER(STATUS) NOT LIKE 'denied'
            )
        GROUP BY
            YEAR(R2.StartDate),
            DATEPART(iso_week, R2.StartDate),
            T.TableID,
            T.ChairAmount
    UNION
        SELECT
            YEAR(R2.StartDate) AS year,
            DATEPART(iso_week, R2.StartDate) AS week,
            T.TableID AS table_id,
            T.ChairAmount AS table_size,
            COUNT(RD.TableID) AS how_many_times_reserved
        FROM Tables T
            INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
            INNER JOIN ReservationCompany RI ON RI.ReservationID =
RD.ReservationID
            INNER JOIN Reservation R2 ON RD.ReservationID =
```

```
R2.ReservationID
        WHERE
            (
                LOWER(STATUS) NOT LIKE 'cancelled'
                AND LOWER(STATUS) NOT LIKE 'denied'
            )
        GROUP BY
            YEAR(R2.StartDate),
            DATEPART(iso_week, R2.StartDate),
            T.TableID,
            T.ChairAmount
GO
```

5.11. TablesMonthly

Miesięczny raport o stolikach: ilość rezerwacji konkretnych stolików.

```
CREATE VIEW dbo.TablesMonthly
AS
        SELECT
            YEAR(R2.StartDate) AS year,
            DATEPART(MONTH, R2.StartDate) AS MONTH,
            T.TableID AS table_id,
            T.ChairAmount AS table_size,
            COUNT(RD.TableID) AS how_many_times_reserved
        FROM Tables T
            INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
            INNER JOIN ReservationIndividual RI ON RI.ReservationID =
RD.ReservationID
            INNER JOIN Reservation R2 ON RD.ReservationID =
R2.ReservationID
        WHERE
            (
                LOWER(STATUS) NOT LIKE 'cancelled'
                AND LOWER(STATUS) NOT LIKE 'denied'
            )
        GROUP BY
            YEAR(R2.StartDate),
            DATEPART(MONTH, R2.StartDate),
            T.TableID,
            T.ChairAmount
    UNION
        SELECT
            YEAR(R2.StartDate) AS year,
            DATEPART(MONTH, R2.StartDate) AS MONTH,
            T.TableID AS table_id,
            T.ChairAmount AS table_size,
            COUNT(RD.TableID) AS how_many_times_reserved
        FROM
            Tables T
            INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
            INNER JOIN ReservationCompany RI ON RI.ReservationID =
RD.ReservationID
            INNER JOIN Reservation R2 ON RD.ReservationID =
```

```
R2.ReservationID
        WHERE
            (
                LOWER(STATUS) NOT LIKE 'cancelled'
                AND LOWER(STATUS) NOT LIKE 'denied'
            )
        GROUP BY
            YEAR(R2.StartDate),
            DATEPART(MONTH, R2.StartDate),
            T.TableID,
            T.ChairAmount
GO
```

5.12. Takeaways orders not picked Individuals

Nieodebrane zamówienia na wynos dla klientów indywidualnych, które są już gotowe.

```
CREATE VIEW dbo.[Takeaways orders not picked Individuals]
AS
    SELECT
        PrefDate AS [Data odbioru],
        concat(LastName, ' ', FirstName) AS [Dane],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode,
        OrderID,
        OrderDate,
        OrderCompletionDate,
        OrderSum
    FROM OrdersTakeaways OT
        INNER JOIN Orders O ON OT.TakeawaysID = O.TakeawayID
        INNER JOIN Clients C ON O.ClientID = C.ClientID
        INNER JOIN IndividualClient IC ON C.ClientID = IC.ClientID
        INNER JOIN Person P ON IC.PersonID = P.PersonID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN Cities C2 ON A.CityID = C2.CityID
    WHERE
        LOWER(OrderStatus) LIKE 'Completed'
GO
```

5.13. Takeaways orders not picked Companies

Nieodebrane zamówienia na wynos dla firm, które są już gotowe.

```
CREATE VIEW dbo.[Takeaways orders not picked Companies]
AS
    SELECT
        PrefDate AS [Data odbioru],
        CompanyName,
        NIP,
```

```
        isnull(KRS, 'Brak') AS [KRS],
        isnull(Regon, 'Brak') AS [Regon],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode,
        OrderID,
        OrderDate,
        OrderCompletionDate,
        OrderSum
    FROM OrdersTakeaways OT
        INNER JOIN Orders O ON OT.TakeawaysID = O.TakeawayID
        INNER JOIN Clients C ON O.ClientID = C.ClientID
        INNER JOIN Companies CO ON C.ClientID = CO.ClientID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN Cities C2 ON A.CityID = C2.CityID
    WHERE
        LOWER(OrderStatus) LIKE 'Completed'
GO
```

5.14. Takeaways orders Individuals

Aktualnie przygotowywane zamówienia dla klientów indywidualnych wraz z danymi kontaktowymi.

```
CREATE VIEW dbo.[Takeaways orders Individuals]
AS
    SELECT
        PrefDate AS [Data odbioru],
        concat(LastName, ' ', FirstName) AS [Dane],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode,
        OrderID,
        OrderDate,
        OrderCompletionDate,
        OrderStatus,
        OrderSum
    FROM OrdersTakeaways OT
        INNER JOIN Orders O ON OT.TakeawaysID = O.TakeawayID
        INNER JOIN Clients C ON O.ClientID = C.ClientID
        INNER JOIN IndividualClient IC ON C.ClientID = IC.ClientID
        INNER JOIN Person P ON IC.PersonID = P.PersonID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN Cities C2 ON A.CityID = C2.CityID
    WHERE
        (
            (
                (getdate() >= OrderDate)
                AND (getdate() <= OrderCompletionDate)
            )
            OR (
                OrderCompletionDate IS NULL
```

```
                AND (getdate() >= OrderDate)
            )
        )
GO
```

5.15. Takeaways orders comapnies

Aktualnie przygotowywane zamówienia dla firm wraz z danymi kontaktowymi.

```
CREATE VIEW dbo.[Takeaways orders companies]
AS
    SELECT
        PrefDate AS [Data odbioru],
        CompanyName,
        NIP,
        isnull(KRS, 'Brak') AS [KRS],
        isnull(Regon, 'Brak') AS [Regon],
        Phone,
        Email,
        concat(CityName, ' ', street, ' ', LocalNr) AS [Adres],
        PostalCode,
        OrderID,
        OrderDate,
        OrderCompletionDate,
        OrderStatus,
        OrderSum
    FROM OrdersTakeaways OT
        INNER JOIN Orders O ON OT.TakeawaysID = O.TakeawayID
        INNER JOIN Clients C ON O.ClientID = C.ClientID
        INNER JOIN Companies CO ON C.ClientID = CO.ClientID
        INNER JOIN Address A ON C.AddressID = A.AddressID
        INNER JOIN Cities C2 ON A.CityID = C2.CityID
    WHERE
        (
            (
                (getdate() >= OrderDate)
                AND (getdate() <= OrderCompletionDate)
            )
            OR (
                OrderCompletionDate IS NULL
                AND (getdate() >= OrderDate)
            )
        )
GO
```

5.16. ReservationInfo

Informacje o nieodwołanych rezerwacjach.

```
CREATE VIEW ReservationInfo
AS
```

```
    SELECT
        R.ReservationID,
        TableID,
        StartDate,
        EndDate
    FROM
        Reservation R
        LEFT OUTER JOIN ReservationDetails RD ON RD.ReservationID =
R.ReservationID
    WHERE
        LOWER(STATUS) NOT LIKE 'cancelled'
GO
```

5.17. ResarvationDenied

Informacje o odrzuconych rezerwacjach.

```
CREATE VIEW ReservationDenied
AS
    SELECT
        R.ReservationID,
        TableID,
        ClientID,
        StartDate,
        EndDate
    FROM
        Reservation R
        LEFT OUTER JOIN ReservationDetails RD ON RD.ReservationID =
R.ReservationID
        INNER JOIN Orders O ON O.ReservationID = R.ReservationID
    WHERE
        STATUS LIKE 'denied'
GO
```

5.18. PendingReservations

Rezerwacje w toku.

```
CREATE VIEW dbo.PendingReservations AS
    SELECT
        R.ReservationID,
        startDate,
        endDate,
        OrderID,
        OrderSum
    FROM
        Reservation R
        INNER JOIN Orders O ON R.ReservationID = O.ReservationID
    WHERE
        STATUS LIKE 'Pending'
GO
```

5.19. OrdersReport

Roczny, miesięczny, tygodniowy raport o zamówieniach.

```
CREATE VIEW dbo.OrdersReport AS
    SELECT
        isnull(convert(varchar(50), YEAR(O.OrderDate), 120), 'Podsumowanie
po latach') AS [Year],
        isnull(convert(varchar(50),  MONTH(O.OrderDate), 120),
'Podsumowanie po miesiacach') AS [Month],
        isnull(convert(varchar(50),  DATEPART(iso_week , O.OrderDate),
120), 'Podsumowanie po tygodniach') AS [WEEK],
        COUNT(O.OrderID) AS [ilość zamówień],
        SUM(OD.Quantity * M.Price) AS [suma przychodów]
    FROM Orders AS O
        INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
        INNER JOIN Products P ON P.ProductID = OD.ProductID
        INNER JOIN MenuDetails M ON M.ProductID = P.ProductID
    GROUP BY ROLLUP (YEAR(O.OrderDate), MONTH(O.OrderDate),
DATEPART(iso_week, O.OrderDate))
GO
```

5.20. individualClientExpensesReport

Roczny, miesięczny, tygodniowy raport o kwotach wydanych przez klientów indywidualnych

```
CREATE VIEW dbo.IndividualClientExpensesReport
AS
    SELECT
        DISTINCT isnull(
            CONVERT(varchar(50), YEAR(O.OrderDate), 120),
            'Podsumowanie Roku'
        ) AS [Year],
        isnull(
            CONVERT(varchar(50), MONTH(O.OrderDate), 120),
            'Podsumowanie miesiaca'
        ) AS [Month],
        isnull(
            CONVERT(
                varchar(50),
                DATEPART(iso_week, O.OrderDate),
                120
            ),
            'Podsumowanie tygodnia'
        ) AS [WEEK],
        C.ClientID,
        CONCAT(P2.LastName, ' ', P2.FirstName) AS [Dane],
        C.Phone,
        C.Email,
        concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        ISNULL(SUM(O.OrderSum), 0) AS [wydane środki]
    FROM Orders O
```

```
        RIGHT JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
        INNER JOIN Person P2 ON P2.PersonID = IC.PersonID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week, O.OrderDate),
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.ClientID,
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.ClientID,
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            ),
            (
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.ClientID,
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode,
                YEAR(O.OrderDate)
            )
        )
GO
```

5.21 companyExpensesReport

Roczny, miesięczny, tygodniowy raport o kwotach wydanych przez firmy.

```
CREATE VIEW dbo.companyExpensesReport
AS
    SELECT
        YEAR(O.OrderDate) AS [Rok],
        MONTH(O.OrderDate) AS [Miesiąc],
        DATEPART(iso_week , O.OrderDate) AS [Tydzień],
        C.ClientID,
        C2.CompanyName,
        C2.NIP,
        ISNULL(cast(C2.KRS AS varchar), 'Brak') AS [KRS],
```

```
        ISNULL(cast(C2.Regon AS varchar), 'Brak') AS [Regon],
        C.Phone,
        C.Email,
        CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        SUM(O.OrderSum) AS [wydane środki]
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week , O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak'),
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak'),
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak'),
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            )
        )
GO
```

5.22. numberOfIndividualClients

Roczny, miesięczny, tygodniowy raport o ilości obsłużonych klientów indywidualnych.

```
CREATE VIEW dbo.numberOfIndividualClients
AS
    SELECT
        YEAR(O.OrderDate) AS [Rok],
        MONTH(O.OrderDate) AS [Miesiąc],
        DATEPART(iso_week , O.OrderDate) AS [Tydzień],
        COUNT(DISTINCT C.ClientID) AS [Ilość klientów indywidualnych]
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week , O.OrderDate)
            ),
            (YEAR(O.OrderDate), MONTH(O.OrderDate)),
            (YEAR(O.OrderDate))
        )
GO
```

5.23. numberOfCompanies

Roczny, miesięczny, tygodniowy raport o ilości obsłużonych firm.

```
CREATE VIEW dbo.numberOfCompanies
AS
    SELECT
        YEAR(O.OrderDate) AS [Rok],
        MONTH(O.OrderDate) AS [Miesiąc],
        DATEPART(iso_week , O.OrderDate) AS [Tydzień],
        COUNT(DISTINCT C.ClientID) AS [Ilość zamawiających firm]
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week, O.OrderDate)
            ),
            (YEAR(O.OrderDate), MONTH(O.OrderDate)),
            (YEAR(O.OrderDate))
        )
GO
```

5.24. individualClientsNumberOfOrders

Roczny, miesięczny, tygodniowy raport o ilości złożonych zamówień przez klientów indywidualnych.

```
CREATE VIEW dbo.individualClientNumberOfOrders
AS
    SELECT
        YEAR(O.OrderDate) AS [Rok],
        MONTH(O.OrderDate) AS [Miesiąc],
        DATEPART(iso_week , O.OrderDate) AS [Tydzień],
        C.ClientID,
        CONCAT(P2.LastName, ' ', P2.FirstName) AS [Dane],
        C.Phone,
        C.Email,
        concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        COUNT(DISTINCT O.OrderID) AS [Ilość złożonych zamówień]
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
        INNER JOIN Person P2 ON P2.PersonID = IC.PersonID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week , O.OrderDate),
                C.ClientID,
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) ,
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                C.ClientID,
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) ,
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                C.ClientID,
                CONCAT(P2.LastName, ' ', P2.FirstName),
                C.Phone,
                C.Email,
                concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) ,
                A.PostalCode
            )
```

```
        )
GO
```

5.25. companiesNumberOfOrders

Roczny, miesięczny, tygodniowy raport o ilości złożonych zamówień przez firmy.

```
CREATE VIEW dbo.companiesNumberOfOrders
AS
    SELECT
        YEAR(O.OrderDate) AS [Rok],
        MONTH(O.OrderDate) AS [Miesiąc],
        DATEPART(iso_week, O.OrderDate) AS [Tydzień],
        C.ClientID,
        C2.CompanyName,
        C2.NIP,
        ISNULL(cast(C2.KRS AS varchar), 'Brak') AS [KRS],
        ISNULL(cast(C2.Regon AS varchar), 'Brak') AS [Regon],
        C.Phone,
        C.Email,
        CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        COUNT(DISTINCT O.OrderID) AS [Ilość złożonych zamówień]
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C3 on C3.CityID = A.CityID
    GROUP BY
        GROUPING SETS (
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                DATEPART(iso_week, O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak') ,
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                MONTH(O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak') ,
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr),
```

```
                A.PostalCode
            ),
            (
                YEAR(O.OrderDate),
                C.ClientID,
                C2.CompanyName,
                C2.NIP,
                ISNULL(cast(C2.KRS AS varchar), 'Brak') ,
                ISNULL(cast(C2.Regon AS varchar), 'Brak'),
                C.Phone,
                C.Email,
                CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr),
                A.PostalCode

            )
        )
GO
```

5.26. individualClientsWhoNotPayForOrders

Klienci indywidualni, którzy mają nieopłacone zamówienia wraz z kwotą należności.

```
CREATE VIEW dbo.IndividualClientsWhoNotPayForOrders
AS
    SELECT
        C.ClientID,
        CONCAT(P.LastName, ' ', P.FirstName) AS [Dane],
        C.Phone,
        C.Email,
        concat(C2.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        O.OrderDate,
        SUM(O.OrderSum) AS [money to pay]
    FROM Clients C
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
        INNER JOIN Orders O ON O.ClientID = C.ClientID
        INNER JOIN PaymentStatus PS ON PS.PaymentStatusID =
O.PaymentStatusID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C2 ON C2.CityID = A.CityID
    WHERE
        (PS.PaymentStatusName LIKE 'Unpaid')
    GROUP BY
        C.ClientID,
        CONCAT(P.LastName, ' ', P.FirstName),
        C.Phone,
        C.Email,
        concat(C2.CityName, ' ', A.street, ' ', A.LocalNr),
        A.PostalCode,
        O.OrderDate
GO
```

5.27. companiesWhoNotPayForOrders

Firmy, które mają nieopłacone zamówienia wraz z kwotą należności.

```
CREATE VIEW dbo.CompaniesWhoNotPayForOrders
AS
    SELECT
        C.ClientID,
        C2.CompanyName,
        C2.NIP,
        ISNULL(C2.KRS, 'Brak') AS [KRS],
        ISNULL(C2.Regon, 'Brak') AS [Regon],
        C.Phone,
        C.Email,
        CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode,
        SUM(O.OrderSum) AS [money to pay]
    FROM Clients C
        INNER JOIN Orders O ON O.ClientID = C.ClientID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
        INNER JOIN PaymentStatus PS ON PS.PaymentStatusID =
O.PaymentStatusID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C3 ON C3.CityID = A.CityID
    WHERE
        (PS.PaymentStatusName LIKE 'Unpaid')
    GROUP BY
        C.ClientID,
        C2.CompanyName,
        C2.NIP,
        ISNULL(C2.KRS, 'Brak'),
        ISNULL(C2.Regon, 'Brak'),
        C.Phone,
        C.Email,
        CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr),
        A.PostalCode
GO
```

5.28. ordersOnSite

Zamówienia na miejscu, które są przygotowywane.

```
CREATE VIEW dbo.ordersOnSite
AS
    SELECT
        O.OrderID,
        O.ClientID,
        C.Phone,
        C.Email,
        OD.Quantity,
        P.Name
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
        INNER JOIN Products P ON P.ProductID = OD.ProductID
```

```
    WHERE
        (O.TakeawayID IS NULL)
        AND (O.OrderStatus LIKE 'accepted')
GO
```

5.29. takeawayOrdersInProgressIndividual

Zamówienia na wynos, które są przygotowywane dla klientów indywidualnych.

```
CREATE VIEW dbo.takeawayOrdersInProgressIndividual
AS
    SELECT
        O.OrderID,
        O.ClientID,
        C.Phone,
        C.Email,
        concat(P.LastName, ' ', P.FirstName) AS [Dane],
        OD.Quantity,
        P.Name,
        OT.PrefDate
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
        INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
        INNER JOIN Products P ON P.ProductID = OD.ProductID
        INNER JOIN OrdersTakeaways OT ON OT.TakeawaysID = O.TakeawayID
    WHERE
        (O.OrderStatus LIKE 'accepted')
GO
```

5.30. takeawayOrdersInProgressCompanies

Zamówienia na wynos, które są przygotowywane dla firm.

```
CREATE VIEW dbo.takeawayOrdersInProgressCompanies
AS
    SELECT
        O.OrderID,
        O.ClientID,
        C.Phone,
        C.Email,
        C2.CompanyName,
        C2.NIP,
        ISNULL(cast(C2.KRS AS varchar), 'Brak') AS [KRS],
        ISNULL(cast(C2.Regon AS varchar), 'Brak') AS [Regon],
        OD.Quantity,
        P.Name,
        OT.PrefDate
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
```

```
        INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
        INNER JOIN Products P ON P.ProductID = OD.ProductID
        INNER JOIN OrdersTakeaways OT ON OT.TakeawaysID = O.TakeawayID
    WHERE
        (O.OrderStatus LIKE 'accepted')
GO
```

5.31. OrdersInformationIndividualClient

Informacje o zamówieniach dla klientów indywidualnych.

```
CREATE VIEW dbo.OrdersInformationIndividualClient
AS
    SELECT
        O.OrderID,
        O.OrderStatus,
        PS.PaymentStatusName,
        SUM(O.OrderSum) AS [Wartość zamówienia],
        C.Phone,
        C.Email,
        CONCAT(P.LastName, ' ', P.FirstName) AS [Dane],
        CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN IndividualClient IC ON IC.ClientID = C.ClientID
        INNER JOIN PaymentStatus PS ON PS.PaymentStatusID =
O.PaymentStatusID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C2 ON A.CityID = C2.CityID
    GROUP BY
        O.OrderID,
        O.OrderStatus,
        PS.PaymentStatusName,
        C.Phone,
        C.Email,
        CONCAT(P.LastName, ' ', P.FirstName),
        CONCAT(C2.CityName, ' ', A.street, ' ', A.LocalNr),
        A.PostalCode
GO
```

5.32. OrdersInformationCompany

Informacje o zamówieniach dla firm.

```
CREATE VIEW dbo.OrdersInformationCompany
AS
    SELECT
        O.OrderID,
        O.OrderStatus,
        PS.PaymentStatusName,
        SUM(O.OrderSum) AS [Wartość zamówienia],
```

```
        C.Phone,
        C.Email,
        C2.CompanyName,
        C2.NIP,
        ISNULL(cast(C2.KRS AS varchar), 'Brak') AS [KRS],
        ISNULL(cast(C2.Regon AS varchar), 'Brak') AS [Regon],
        CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr) AS [Adres],
        A.PostalCode
    FROM Orders O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN PaymentStatus PS ON PS.PaymentStatusID =
O.PaymentStatusID
        INNER JOIN Companies C2 ON C2.ClientID = C.ClientID
        INNER JOIN Address A ON A.AddressID = C.AddressID
        INNER JOIN Cities C3 ON A.CityID = C3.CityID
    GROUP BY
        O.OrderID,
        O.OrderStatus,
        PS.PaymentStatusName,
        C.Phone,
        C.Email,
        C2.CompanyName,
        C2.NIP,
        ISNULL(cast(C2.KRS AS varchar), 'Brak'),
        ISNULL(cast(C2.Regon AS varchar), 'Brak'),
        CONCAT(C3.CityName, ' ', A.street, ' ', A.LocalNr),
        A.PostalCode
GO
```

5.33. PendingReservationsCompanies

Rezerwacje w toku dla firm.

```
CREATE VIEW dbo.PendingReservationsCompanies
AS
    SELECT
        R.ReservationID,
        startDate,
        endDate,
        OrderID,
        OrderSum
    FROM Reservation R
        INNER JOIN ReservationCompany RC ON RC.ReservationID =
R.ReservationID
        INNER JOIN Orders O ON R.ReservationID = O.ReservationID
    WHERE
        LOWER(STATUS) LIKE 'pending'
GO
```

5.34. PendingReservationIndividual

Rezerwacje w toku dla klientów indywidualnych.

```
CREATE VIEW dbo.PendingReservationsIndividual AS
    SELECT
        R.ReservationID,
        startDate,
        endDate,
        OrderID,
        OrderSum
    FROM Reservation R
        INNER JOIN ReservationIndividual RC ON RC.ReservationID =
R.ReservationID
        INNER JOIN Orders O ON R.ReservationID = O.ReservationID
    WHERE
        LOWER(STATUS) LIKE 'pending'
GO
```

5.35. ReservationAcceptedBy

Kto potwierdził dane zamówienie.

```
CREATE VIEW dbo.ReservationAcceptedBy
AS
    SELECT
        concat(LastName, ' ', FirstName) AS Dane,
        Position,
        Email,
        Phone
    FROM Staff
        INNER JOIN Reservation R2 ON Staff.StaffID = R2.StaffID
    WHERE
        LOWER(STATUS) LIKE 'accepted'
GO
```

5.36. ReservationSummary

Podsumowanie rezerwacji.

```
CREATE VIEW dbo.ReservationSummary
AS
        SELECT
            O.ClientID AS 'Numer clienta',
            startDate,
            endDate,
            CONVERT(TIME, endDate - startDate, 108) AS 'Czas trwania',
            O.OrderSum,
            O.OrderDate,
            O.OrderCompletionDate,
            OD.Quantity,
            RD.TableID
        FROM Reservation
            INNER JOIN Orders O ON Reservation.ReservationID =
O.ReservationID
```

```
            INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
            INNER JOIN ReservationCompany RC ON Reservation.ReservationID
= RC.ReservationID
            INNER JOIN ReservationDetails RD ON RC.ReservationID =
RD.ReservationID
        WHERE
            LOWER(STATUS) NOT LIKE 'denied' AND LOWER(STATUS) NOT LIKE
'cancelled' AND LOWER(O.OrderStatus) NOT LIKE 'denied' AND
LOWER(O.OrderStatus) NOT LIKE 'cancelled'
    UNION
        SELECT
            O.ClientID AS 'Numer clienta',
            startDate,
            endDate,
            CONVERT(TIME, endDate - startDate, 108) AS 'Czas trwania',
            O.OrderSum,
            O.OrderDate,
            O.OrderCompletionDate,
            OD.Quantity,
            RD.TableID
        FROM Reservation
            INNER JOIN Orders O ON Reservation.ReservationID =
O.ReservationID
            INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
            INNER JOIN ReservationIndividual RC ON
Reservation.ReservationID = RC.ReservationID
            INNER JOIN ReservationDetails RD ON RC.ReservationID =
RD.ReservationID
        WHERE
            LOWER(STATUS) NOT LIKE 'denied' AND LOWER(STATUS) NOT LIKE
'cancelled' AND LOWER(O.OrderStatus) NOT LIKE 'denied' AND
LOWER(O.OrderStatus) NOT LIKE 'cancelled'
GO
```

5.37. ProductsSummaryDaily

Raport o ilości zamówionych produktów w danych dniach.

```
CREATE VIEW dbo.ProductsSummaryDaily
AS
    SELECT
        P.Name,
        P.Description,
        cast(O.OrderDate AS DATE) AS 'Dzien',
        count(OD.ProductID) AS 'Liczba zamowionych produktow'
    FROM Products P
        INNER JOIN OrderDetails OD ON P.ProductID = OD.ProductID
        INNER JOIN Orders O ON OD.OrderID = O.OrderID
    WHERE
            LOWER(O.OrderStatus) NOT LIKE 'denied' AND
LOWER(O.OrderStatus) NOT LIKE 'cancelled'
    GROUP BY
        P.Name,
        P.Description,
```

```
        cast(O.OrderDate AS DATE)
GO
```

5.38. ProductsSummaryWeekly

Raport o ilości zamówionych produktów w danych dniach.

```
CREATE VIEW dbo.ProductsSummaryWeekly
AS
    SELECT
        P.Name,
        P.Description,
        DATEPART(iso_week, cast(O.OrderDate AS DATE)) AS 'Tydzien',
        DATEPART(YEAR, cast(O.OrderDate AS DATE)) AS 'Rok',
        count(OD.ProductID) AS 'Liczba produktow'
    FROM Products P
        INNER JOIN OrderDetails OD ON P.ProductID = OD.ProductID
        INNER JOIN Orders O ON OD.OrderID = O.OrderID
    WHERE
        LOWER(O.OrderStatus) NOT LIKE 'denied' AND LOWER(O.OrderStatus)
NOT LIKE 'cancelled'
    GROUP BY
        P.Name,
        P.Description,
        DATEPART(iso_week, cast(O.OrderDate AS DATE)),
        DATEPART(YEAR, cast(O.OrderDate AS DATE))
GO
```

5.39. Products summary monthly

Raport o ilości zamówionych produktów w danych miesiącach.

```
CREATE VIEW dbo.ProductsSummaryMonthly
AS
    SELECT
        P.Name,
        P.Description,
        DATEPART(MONTH, cast(O.OrderDate AS DATE)) AS 'Miesiac',
        DATEPART(YEAR, cast(O.OrderDate AS DATE)) AS 'Rok',
        count(OD.ProductID) AS 'Liczba zamowionych produktow'
    FROM Products P
        INNER JOIN OrderDetails OD ON P.ProductID = OD.ProductID
        INNER JOIN Orders O ON OD.OrderID = O.OrderID
    WHERE
        LOWER(O.OrderStatus) NOT LIKE 'denied' AND LOWER(O.OrderStatus)
NOT LIKE 'cancelled'
    GROUP BY
        P.Name,
        P.Description,
        DATEPART(MONTH, cast(O.OrderDate AS DATE)),
        DATEPART(YEAR, cast(O.OrderDate AS DATE))
GO
```

5.40. WhoIssuedAnOrder

Kto wydał dane zamówienia.

```
CREATE OR ALTER VIEW dbo.WhoIssuedAnOrder
AS
    SELECT
        FirstName + ' ' + LastName AS Name,
        OrderID AS id
    FROM Staff
        INNER JOIN Orders O ON Staff.StaffID = O.staffID
    WHERE
        LOWER(Position) LIKE 'waiter'
        OR LOWER(Position) LIKE 'waitress';
GO
```

5.41. AllTakeaways

Wszystkie zamówienia na wynos.

```
CREATE OR ALTER VIEW dbo.AllTakeaways
AS
    SELECT
        TakeawayID,
        PrefDate,
        OrderID,
        ClientID,
        PaymentStatusID,
        concat(S.LastName, ' ', S.FirstName) AS 'Dane kelnera',
        Position,
        OrderSum,
        OrderDate,
        OrderCompletionDate,
        OrderStatus
    FROM OrdersTakeaways
        JOIN Orders O ON OrdersTakeaways.TakeawaysID = O.TakeawayID
        JOIN Staff S ON O.staffID = S.StaffID
GO
```

5.42. OrdersToPrepare

Aktualnie przygotowywane zamówienia.

```
CREATE OR ALTER VIEW dbo.OrdersToPrepare
AS
    SELECT OrderID, ClientID, ISNULL(CAST(TakeawayID AS varchar), '') as
TakeawayID, ISNULL(CAST(ReservationID AS varchar), '') as ReservationID,
PaymentStatusName, PM.PaymentName,
    CONCAT(S.LastName, ' ',S.FirstName) AS 'Dane kelnera',
    OrderSum, OrderDate
    FROM Orders O
        INNER JOIN PaymentStatus PS ON PS.PaymentStatusID =
```

```
O.PaymentStatusID
        INNER JOIN PaymentMethods PM ON PM.PaymentMethodID =
O.PaymentMethodID
        INNER JOIN Staff S ON O.staffID = S.StaffID
    WHERE (((GETDATE() >= OrderDate) AND (GETDATE() <=
OrderCompletionDate)) OR (OrderCompletionDate IS NULL AND (GETDATE() >=
OrderDate)) AND LOWER(OrderStatus) LIKE 'pending')
GO
```

5.43. CurrentDiscounts

Aktualnie nałożone zniżki na klientów.

```
CREATE OR ALTER VIEW CurrentDiscounts
AS
    SELECT
        FirstName,
        LastName,
        IC.ClientID,
        DiscountID,
        AppliedDate,
        startDate,
        endDate,
        DiscountType,
        DiscountValue,
        MinimalOrders,
        MinimalAggregateValue,
        ValidityPeriod
    FROM DiscountsVar
        INNER JOIN Discounts ON DiscountsVar.VarID = Discounts.VarID
        INNER JOIN IndividualClient IC ON Discounts.ClientID = IC.ClientID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
    WHERE
        (
            (
                (getdate() >= startDate)
                AND (getdate() <= endDate)
            )
            OR (
                (getdate() >= startDate)
                AND (endDate IS NULL)
            )
        )
GO
```

5.44. AllDiscounts

Wszystkie przyznane zniżki.

```
CREATE OR ALTER VIEW AllDiscounts
AS
    SELECT
        IC.PersonID,
        LastName,
        FirstName,
        IC.ClientID,
        DiscountsVar.VarID,
        DiscountType,
        MinimalOrders,
        MinimalAggregateValue,
        ValidityPeriod,
        DiscountValue,
        startDate,
        endDate,
        DiscountID,
        AppliedDate
    FROM DiscountsVar
        INNER JOIN Discounts ON DiscountsVar.VarID = Discounts.VarID
        INNER JOIN IndividualClient IC ON Discounts.ClientID = IC.ClientID
        INNER JOIN Person P ON P.PersonID = IC.PersonID
GO
```

5.45. DishesInProgressTakeaways

Dania na wynos wymagane na dzisiaj.

```
CREATE OR ALTER VIEW DishesInProgressTakeaways
AS
    SELECT
        Name,
        count(Products.ProductID) AS 'Liczba zamowien',
        sum(Quantity) AS 'Liczba sztuk'
    FROM Products
        INNER JOIN OrderDetails OD ON Products.ProductID = OD.ProductID
        INNER JOIN Orders ON OD.OrderID = Orders.OrderID
        INNER JOIN OrdersTakeaways OT ON Orders.TakeawayID =
OT.TakeawaysID
    WHERE
        (
            (
                (getdate() >= OrderDate)
                AND (getdate() <= OrderCompletionDate)
            )
        )
        AND (
            LOWER(Orders.OrderStatus) NOT LIKE 'denied'
            OR LOWER(Orders.OrderStatus) NOT LIKE 'cancelled'
        )
    GROUP BY Name
GO
```

5.46. DishesInProgressReservation

Dania wymagane na dzisiaj w rezerwacji.

```
CREATE OR ALTER VIEW DishesInProgressReservation
AS
    SELECT
        Name,
        count(Products.ProductID) AS 'Liczba zamowien',
        sum(Quantity) AS 'Liczba sztuk'
    FROM Products
        INNER JOIN OrderDetails OD ON Products.ProductID = OD.ProductID
        INNER JOIN Orders ON OD.OrderID = Orders.OrderID
        INNER JOIN Reservation R2 ON Orders.ReservationID =
R2.ReservationID
    WHERE
        (
            (
                (getdate() >= OrderDate)
                AND (getdate() <= OrderCompletionDate)
            )
        )
        AND LOWER(Orders.OrderStatus) NOT LIKE 'denied'
        AND (
            LOWER(R2.Status) NOT LIKE 'denied'
            OR LOWER(R2.Status) NOT LIKE 'cancelled'
        )
    GROUP BY Name
GO
```

5.47. ProductsInformation

Informacje o produktach.

```
CREATE VIEW dbo.ProductsInformation
AS
    SELECT  Name,
            P.Description,
            CategoryName,
            IIF(IsAvailable = 1, 'Aktywne', 'Nieaktywne') AS 'Czy produkt
aktywny',
            IIF(P.ProductID IN (SELECT ProductID FROM MenuDetails M
                                INNER JOIN Menu M2 on M2.MenuID = M.MenuID
                            WHERE ((startDate >= getdate()) AND (endDate
>= getdate()))
                                OR ((startDate >= getdate()) AND endDate
IS NULL) AND P.ProductID = M.ProductID), 'Aktualnie w menu', 'Nie jest w
menu') as 'Czy jest aktualnie w menu',
            count(OD.ProductID) as 'Ilosc zamowien danego produktu'
    FROM Products P
        LEFT JOIN OrderDetails OD ON P.ProductID = OD.ProductID
        INNER JOIN Category C ON C.CategoryID = P.CategoryID
    GROUP BY Name, P.Description, CategoryName, P.ProductID, IsAvailable
GO
```

5.48. MealMenuInfo

Informacje o menu.

```
CREATE VIEW MealMenuInfo
AS
    SELECT
        DISTINCT M.MenuID,
        M2.startDate,
        M2.endDate,
        M.ProductID,
        ISNULL(
            (
                SELECT
                    SUM(Quantity)
                FROM Products P
                    INNER JOIN OrderDetails OD ON P.ProductID =
OD.ProductID AND P.ProductID = M.ProductID
                    INNER JOIN Orders O ON O.OrderID = OD.OrderID
                WHERE
                    (
                        O.OrderDate BETWEEN M2.startDate
                        AND M2.endDate
                    )
                GROUP BY P.Name
            ),
            0
        ) times_sold
    FROM MenuDetails M
        INNER JOIN Menu M2 ON M.MenuID = M2.MenuID
GO
```

5.49. ClientExpensesReport

Raport o rocznych, miesięcznych, tygodniowych wydatkach danego klienta.

```
CREATE VIEW dbo.ClientExpensesReport
AS
    SELECT
        YEAR(O.OrderDate) AS [Year],
        isnull(convert(varchar(50),  MONTH(O.OrderDate), 120),
'Podsumowanie miesiaca') AS [Month],
        isnull(convert(varchar(50),  DATEPART(iso_week , O.OrderDate),
120), 'Podsumowanie tygodnia') AS [Week],
        C.ClientID,
        SUM(O.OrderSum) AS [wydane środki]
    FROM Orders AS O
        INNER JOIN Clients C ON C.ClientID = O.ClientID
        INNER JOIN OrderDetails OD ON OD.OrderID = O.OrderID
        INNER JOIN Products P ON P.ProductID = OD.ProductID
        INNER JOIN MenuDetails M ON M.ProductID = P.ProductID
    GROUP BY GROUPING SETS (
            (C.ClientID, YEAR(O.OrderDate), MONTH(O.OrderDate),
DATEPART(iso_week, O.OrderDate)),
```

```
            (C.ClientID, YEAR(O.OrderDate), MONTH(O.OrderDate)),
            (C.ClientID, YEAR(O.OrderDate))
        )
GO
```

5.50. CurrentDiscountsVars

Aktualnie obowiązujące zniżki

```
CREATE VIEW dbo.CurrentDiscountsVars
AS
    SELECT
        VarID,
        DiscountType,
        ISNULL(CAST(MinimalOrders AS varchar), ' ') AS MinimalOrders,
        ISNULL(CAST(MinimalAggregateValue AS varchar), ' ') AS
MinimalAggregateValue,
        ISNULL(CAST(ValidityPeriod AS varchar), ' ') AS ValidityPeriod,
        DiscountValue,
        startDate,
        endDate
    FROM dbo.DiscountsVar
    WHERE
        (
            (
                (getdate() >= startDate)
                AND (getdate() <= endDate)
            )
        )
GO
```

5.51. ClientsStatistics

Statystyki danych klientów

```
CREATE VIEW ClientStatistics
AS
SELECT  C.ClientID,
            C2.CityName + ' ' + A.street + ' ' + A.LocalNr + ' ' +
A.PostalCode as Address,
            C.Phone,
            C.Email,
            COUNT(O.OrderID) as [times ordered],
            ISNULL((SELECT [value ordered]
                    FROM (SELECT ClientID, SUM(value) [value ordered]
                            FROM (SELECT O.ClientID as ClientID,
O.OrderSum as value
                                    FROM Orders O) OUT
                            GROUP BY ClientID) a
                    WHERE ClientID = C.ClientID), 0) [value ordered]
    FROM Clients C
        LEFT JOIN Orders O ON C.ClientID = O.ClientID
```

```
        INNER JOIN Address A on A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
    GROUP BY C.ClientID, C2.CityName + ' ' + A.street + ' ' + A.LocalNr +
' ' + A.PostalCode, C.Phone, C.Email
GO
```

5.52. ReservationSummaryMonthly

Miesięczny raport o rezerwacjach

```
CREATE VIEW dbo.ReservationSummaryMonthly
AS
    SELECT
        R.ReservationID,
        R.startDate,
        R.endDate,
        R.Status,
        O.ClientID,
        DATEPART(MONTH, cast(O.OrderDate AS DATE)) AS 'Miesiac',
        DATEPART(YEAR, cast(O.OrderDate AS DATE)) AS 'Rok',
        count(OD.ProductID) AS 'Liczba zamowionych produktow'
    FROM Reservation R
        INNER JOIN Orders O on R.ReservationID = O.ReservationID
        INNER JOIN OrderDetails OD on O.OrderID = OD.OrderID
    WHERE
        LOWER(STATUS) NOT LIKE 'denied' AND LOWER(STATUS) NOT LIKE
'cancelled' AND LOWER(O.OrderStatus) NOT LIKE 'denied' AND
LOWER(O.OrderStatus) NOT LIKE 'cancelled'
    GROUP BY
        R.ReservationID,
        R.startDate,
        R.endDate,
        R.Status,
        O.ClientID,
        DATEPART(MONTH, cast(O.OrderDate AS DATE)),
        DATEPART(YEAR, cast(O.OrderDate AS DATE))
GO
```

5.53. ReservationSummaryWeekly

Tygodniowy raport o rezerwacjach

```
CREATE VIEW dbo.ReservationSummaryWeekly
AS
    SELECT
        R.ReservationID,
        R.startDate,
        R.endDate,
        R.Status,
        O.ClientID,
        DATEPART(iso_week, cast(O.OrderDate AS DATE)) AS 'Tydzien',
        DATEPART(YEAR, cast(O.OrderDate AS DATE)) AS 'Rok',
        count(OD.ProductID) AS 'Liczba zamowionych produktow'
```

```
    FROM Reservation R
        INNER JOIN Orders O on R.ReservationID = O.ReservationID
        INNER JOIN OrderDetails OD on O.OrderID = OD.OrderID
    WHERE
        LOWER(STATUS) NOT LIKE 'denied' AND LOWER(STATUS) NOT LIKE
'cancelled' AND LOWER(O.OrderStatus) NOT LIKE 'denied' AND
LOWER(O.OrderStatus) NOT LIKE 'cancelled'
    GROUP BY
        R.ReservationID,
        R.startDate,
        R.endDate,
        R.Status,
        O.ClientID,
        DATEPART(iso_week, cast(O.OrderDate AS DATE)),
        DATEPART(YEAR, cast(O.OrderDate AS DATE))
GO
```

5.54. ShowIndividualClients

Lista wszystkich klientów indywidualnych.

```
CREATE VIEW dbo.ShowIndividualClients
AS
    SELECT
        C.ClientID,
        P.FirstName,
        P.LastName,
        C.Phone,
        C.Email,
        C2.CityName + ' ' + A.street + ' ' + A.LocalNr + ' ' +
A.PostalCode as Address
    FROM Clients C
        INNER JOIN Address A on A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
        INNER JOIN IndividualClient IC on IC.ClientID = C.ClientID
        INNER JOIN Person P on IC.PersonID = P.PersonID
GO
```

5.55. ShowCompanyClients

Lista wszystkich firm.

```
CREATE VIEW dbo.ShowCompanyClients
AS
    SELECT
        C.ClientID,
        CompanyName,
        NIP,
        ISNULL(CAST(KRS AS VARCHAR), '') AS KRS,
        ISNULL(CAST(Regon AS VARCHAR), '') AS Regon,
        C.Phone,
```

```
        C.Email,
        C2.CityName + ' ' + A.street + ' ' + A.LocalNr + ' ' +
A.PostalCode as Address
    FROM Clients C
        INNER JOIN Address A on A.AddressID = C.AddressID
        INNER JOIN Cities C2 on C2.CityID = A.CityID
        INNER JOIN Companies CC on CC.ClientID = C.ClientID
GO
```

5.56. DiscountsSummaryPerClient

Informacje o zniżkach dla danych klientów.

```
CREATE VIEW DiscountsSummaryPerClient
AS
    SELECT
            IC.ClientID,
            CONCAT(P.LastName, ' ', P.FirstName) AS 'Person',
            DiscountID,
            AppliedDate,
            DiscountType,
            DiscountValue,
            ISNULL(CAST(MinimalOrders AS varchar), '') AS 'Minimal Orders
needed',
            ISNULL(CAST(MinimalAggregateValue AS varchar), '') AS 'Minimal
Aggregate Value needed',
            ISNULL(CAST(ValidityPeriod AS varchar), '') AS 'Validity
Period',
            ISNULL(CAST(isUsed AS varchar), 'It is permanent') AS 'is
Used'
    FROM IndividualClient IC
        INNER JOIN Person P on P.PersonID = IC.PersonID
        INNER JOIN Discounts D on IC.ClientID = D.ClientID
        INNER JOIN DiscountsVar DV on D.VarID = DV.VarID
GO
```

6. Procedury

6.1. addCategory

```
CREATE PROCEDURE addCategory @CategoryName nvarchar(50), @Description
nvarchar(150) AS
    BEGIN
        SET NOCOUNT ON
            BEGIN TRY
                IF EXISTS(
                    SELECT * FROM Category WHERE @CategoryName =
CategoryName
                )
                    BEGIN
                        ;
                        THROW 52000, N'Kategoria juz istnieje!', 1
                    END
                INSERT INTO Project.dbo.Category (CategoryName,
Description) VALUES (@CategoryName, @Description)
            END TRY
            BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Blad dodania kategorii: '
+ ERROR_MESSAGE();
                THROW 52000, @msg, 1
            END CATCH
    END
GO
```

6.2. ModifyTableSize

```
CREATE PROCEDURE ModifyTableSize @TableID int, @Size int
AS
    BEGIN
        SET NOCOUNT ON
            BEGIN TRY
                IF NOT EXISTS(SELECT * from Tables where TableID =
@TableID)
                    BEGIN;
                        THROW 52000, N'Nie ma takiego stolika.', 1
                    END
                IF @Size < 2
                    BEGIN;
                        THROW 52000, N'Stolik musi mieć przynajmniej 2
miejsca.', 1
                    END
                IF @Size IS NOT NULL
                    BEGIN
                        UPDATE Tables SET ChairAmount = @Size WHERE
TableID = @TableID
                    END
            END TRY
            BEGIN CATCH
                DECLARE  @msg nvarchar(2048) = N'Bład edytowania stolika:
```

```
' + ERROR_MESSAGE();
                THROW 52000, @msg, 1
            END CATCH
    END
GO
```

6.3. ModifyTableStatus

```
CREATE PROCEDURE ModifyTableStatus @TableID int, @Status bit
AS
    BEGIN
        SET NOCOUNT ON
            BEGIN TRY
                IF NOT EXISTS(SELECT * from Tables where TableID =
@TableID)
                    BEGIN;
                        THROW 52000, N'Nie ma takiego stolika.', 1
                    END
                DECLARE @TableStatus bit
                SELECT @TableStatus = isActive from Tables where TableID =
@TableID
                IF @TableStatus = @Status
                    BEGIN;
                        THROW 52000, N'Stolik ma już taki status!.', 1
                    END
                IF @Status IS NOT NULL
                    BEGIN
                        UPDATE Tables SET isActive = @Status WHERE TableID
= @TableID
                    END
            END TRY
            BEGIN CATCH
                DECLARE  @msg nvarchar(2048) = N'Bład edytowania stolika:
' + ERROR_MESSAGE();
                THROW 52000, @msg, 1
            END CATCH
    END
GO
```

6.4. addTable

```
CREATE PROCEDURE addTable @Size int, @Status bit
AS
    BEGIN
       SET NOCOUNT ON
       BEGIN TRY
           IF @Size < 2
               BEGIN;
                    THROW 52000, N'Stolik musi mieć przynajmniej 2
miejsca.', 1
                END
```

```
            DECLARE @TableID INT
            SELECT @TableID = ISNULL(MAX(TableID), 0) + 1 FROM Tables
            INSERT INTO Tables(TableID, ChairAmount, isActive)
            VALUES (@TableID, @Size, @Status)
        END TRY
        BEGIN CATCH
             DECLARE  @msg nvarchar(2048) = N'Blad edytowania stolika: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.5. removeTable

```
CREATE PROCEDURE removeTable @TableID int
AS
    BEGIN
        SET NOCOUNT ON
            BEGIN TRY
                IF NOT EXISTS(SELECT * from Tables where TableID =
@TableID)
                    BEGIN;
                        THROW 52000, N'Nie ma takiego stolika.', 1
                    END
                DELETE FROM Tables WHERE TableID = @TableID
            END TRY
            BEGIN CATCH
                DECLARE  @msg nvarchar(2048) = N'Blad edytowania stolika:
' + ERROR_MESSAGE();
                THROW 52000, @msg, 1
            END CATCH
    END
GO
```

6.6. addCity

```
CREATE PROCEDURE addCity @CityName varchar(35)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM Cities WHERE CityName = @CityName)
            BEGIN
                THROW 52000, 'Takie miasto już istnieje!', 1
            END
            INSERT INTO Cities(CityName) VALUES (@CityName)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania miasta: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
```

```
    END
GO
```

6.7. addAddress

```
CREATE PROCEDURE addAddress
                @Street nvarchar(70),
                @LocalNr varchar(10),
                @PostalCode char(6),
                @CityName nvarchar(35)
AS
    BEGIN
       SET NOCOUNT ON
       BEGIN TRY
            DECLARE @CityID int
            IF NOT EXISTS(SELECT * FROM Cities WHERE CityName LIKE
@CityName)
                 BEGIN
                     EXEC addCity @CityName
                 END
            SELECT @CityID = CityID FROM Cities WHERE CityName LIKE
@CityName

            IF EXISTS(SELECT * FROM Address WHERE CityID = @CityID AND
PostalCode LIKE @PostalCode AND street LIKE @Street AND LocalNr LIKE
@LocalNr)
                BEGIN
                    THROW 52000, 'Istnieje już dokładnie taki sam adres w
bazie!', 1
                END
            INSERT INTO Address(CityID, street, LocalNr, PostalCode)
            VALUES (@CityID, @Street, @LocalNr, @PostalCode)
       END TRY
       BEGIN CATCH
           DECLARE  @msg nvarchar(2048) = N'Bład dodania adresu: ' +
ERROR_MESSAGE();
           THROW 52000, @msg, 1
       END CATCH
    END
go
```

6.8 removeAdrress

```
CREATE PROCEDURE removeAddress @AddressID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * from Address where AddressID =
@AddressID)
                BEGIN;
                    THROW 52000, N'Nie ma takiego adresu.', 1
                END
            DELETE FROM Address WHERE AddressID = @AddressID
        END TRY
```

```
        BEGIN CATCH
            DECLARE  @msg nvarchar(2048) = N'Blad usuniecia adresu: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.9. addPerson

```
CREATE PROCEDURE addPerson @FirstName varchar(70), @LastName varchar(50)
as
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            INSERT INTO Person(LastName, FirstName)
            VALUES(@LastName, @FirstName)
        end try
        begin catch
            DECLARE  @msg nvarchar(2048) = N'Blad dodania Osoby: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        end catch
    end
GO
```

6.10. removePerson

```
CREATE PROCEDURE removePerson @PersonID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * from Person where PersonID = @PersonID)
                BEGIN;
                    THROW 52000, N'Nie ma takiej osoby.', 1
                END
            DELETE FROM Person WHERE PersonID = @PersonID
        END TRY
        BEGIN CATCH
            DECLARE  @msg nvarchar(2048) = N'Blad usuniecia osoby: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.11. addClient

@ClientType przyjmuje wartość ‘I’ dla klienta indywidualnego lub ‘C’ dla klienta firmowego

```
CREATE PROCEDURE addClient @ClientType varchar(1),
```

```
                                @CityName nvarchar(35) = NULL,
                                @Street nvarchar(70) = NULL,
                                @LocalNr varchar(10) = NULL,
                                @PostalCode char(6) = NULL,
                                @AddressID int = NULL,
                                @Phone varchar(14),
                                @Email varchar(100),
                                @FirstName varchar(50) = NULL,
                                @LastName varchar(70) = NULL,
                                @CompanyName nvarchar(50) = NULL,
                                @NIP char(10) = NULL,
                                @KRS char(10) = NULL,
                                @REGON char(9) = NULL
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF (@ClientType NOT LIKE 'C' AND @ClientType NOT LIKE 'I')
            BEGIN;
                THROW 52000, N'Nie ma takiego typu klienta!', 1
            END

            IF EXISTS(SELECT * FROM Clients WHERE Phone LIKE @Phone)
            BEGIN;
                THROW 52000, N'Numer telefonu jest już w bazie', 1
            END

            IF EXISTS(SELECT * FROM Clients WHERE Email LIKE @Email)
            BEGIN;
                THROW 52000,N'Email jest już w bazie',1
            END

            IF @CompanyName IS NOT NULL AND @ClientType LIKE 'C' AND
EXISTS( SELECT * FROM Companies WHERE CompanyName LIKE @CompanyName)
            BEGIN;
                THROW 52000, N'Firma jest już w bazie', 1
            END

            IF @KRS IS NOT NULL AND @ClientType LIKE 'C' AND EXISTS(
SELECT * FROM Companies WHERE KRS LIKE @KRS)
            BEGIN;
                THROW 52000, N'KRS jest już w bazie', 1
            END

            IF @NIP IS NOT NULL AND @ClientType LIKE 'C' AND EXISTS(
SELECT * FROM Companies WHERE NIP LIKE @NIP)
            BEGIN;
                THROW 52000, N'NIP jest już w bazie', 1
            END

            IF @REGON IS NOT NULL AND @ClientType LIKE 'C' AND EXISTS(
SELECT * FROM Companies WHERE Regon LIKE @REGON)
            BEGIN;
                THROW 52000, N'REGON jest już w bazie', 1
            END

            IF (@ClientType = 'C')
```

```
            BEGIN
                IF(@NIP IS NULL)
                BEGIN
                    THROW 52000, N'Nip musi być określony dla klienta
firmowego!', 1
                END

                IF(@CompanyName IS NULL)
                BEGIN
                    THROW 52000, N'CompanyName musi być określony dla
klienta firmowego!', 1
                END
            END

            IF (@ClientType = 'I')
            BEGIN
                IF(@LastName IS NULL)
                BEGIN
                    THROW 52000, N'Nazwisko musi być określony dla klienta
indywidualnego!', 1
                END

                IF(@FirstName IS NULL)
                BEGIN
                    THROW 52000, N'Imie musi być określony dla klienta
indywidualnego!',1
                END
            END

            IF @Street IS NOT NULL AND @PostalCode IS NULL AND @LocalNr IS
NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street! Musisz podać
jeszcze @CityName, @LocalNr, @PostalCode!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NOT NULL AND @LocalNr IS
NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @PostalCode! Musisz
podać jeszcze @Street, @LocalNr, @CityName!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NULL AND @LocalNr IS NOT
NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @LocalNr. Musisz
podać jeszcze @Street, @CityName, @PostalCode!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NULL AND @LocalNr IS
NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @CityName. Musisz
podać jeszcze @Street, @LocalNr, @PostalCode!', 1
            END
```

```
            IF @Street IS NOT NULL AND @PostalCode IS NOT NULL AND
@LocalNr IS NOT NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @PostalCode,
@LocalNr. Musisz podać jeszcze @CityName!', 1
            END

            IF @Street IS NOT NULL AND @PostalCode IS NOT NULL AND
@LocalNr IS NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @PostalCode,
@CityName. Musisz podać jeszcze @LocalNr!', 1
            END

            IF @Street IS NOT NULL AND @PostalCode IS NULL AND @LocalNr IS
NOT NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @LocalNr,
@CityName. Musisz podać jeszcze @PostalCode!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NOT NULL AND @LocalNr IS
NOT NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @PostalCode,
@LocalNr, @CityName. Musisz podać jeszcze @Street!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NULL AND @LocalNr IS NOT
NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @LocalNr, @CityName.
Musisz podać jeszcze @Street, @PostalCode!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NOT NULL AND @LocalNr IS
NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @PostalCode,
@CityName. Musisz podać jeszcze @Street, @LocalNr!', 1
            END

            IF @Street IS NULL AND @PostalCode IS NOT NULL AND @LocalNr IS
NOT NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @PostalCode,
@LocalNr. Musisz podać jeszcze @Street, @CityName!', 1
            END

            IF @Street IS NOT NULL AND @PostalCode IS NULL AND @LocalNr IS
NULL AND @CityName IS NOT NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @CityName.
Musisz podać jeszcze @PostalCode, @LocalNr!', 1
            END

            IF @Street IS NOT NULL AND @PostalCode IS NULL AND @LocalNr IS
```

```
NOT NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @LocalNr.
Musisz podać jeszcze @PostalCode, @CityName!', 1
            END

            IF @Street IS NOT NULL AND @PostalCode IS NOT NULL AND
@LocalNr IS NULL AND @CityName IS NULL
            BEGIN
                THROW 52000, N'Nie można podać tylko @Street, @PostalCode.
Musisz podać jeszcze @LocalNr, @CityName!', 1
            END


            DECLARE @AddressID_ int;
            IF @Street IS NOT NULL AND @PostalCode IS NOT NULL AND
@LocalNr IS NOT NULL AND @CityName IS NOT NULL
                BEGIN
                    IF NOT EXISTS( SELECT * FROM Address WHERE street LIKE
@Street AND PostalCode LIKE @PostalCode AND LocalNr LIKE @LocalNr)
                        BEGIN
                            EXEC addAddress
@Street,@LocalNr,@PostalCode,@CityName
                        END
                    SELECT @AddressID_ = AddressID FROM Address
                END
            ELSE
                BEGIN
                    IF @AddressID IS NOT NULL
                        BEGIN
                           SET @AddressID_ = @AddressID
                        END
                    ELSE
                        BEGIN
                            THROW 52000, 'Nie można, żeby wszystkie
parametry nie zostały podane tj. @AddressID, @Street, @PostalCode,
@LocalNr, @CityName. Musisz podać @AddressID lub @Street, @PostalCode,
@LocalNr, @CityName!', 1
                        END
                END


            INSERT INTO Clients(AddressID, Phone, Email)
            VALUES(@AddressID_, @Phone, @Email)

            DECLARE @ClientID int;
            SELECT @ClientID = ClientID FROM Clients
            WHERE @AddressID_ = AddressID
                AND Clients.Phone LIKE @Phone
                AND Clients.Email LIKE @Email

            IF (@ClientType = 'C')
            BEGIN
                INSERT INTO Companies(ClientID, CompanyName, NIP, KRS,
Regon)
                VALUES(@ClientID, @CompanyName, @NIP, @KRS, @REGON)
            END
```

```
            IF (@ClientType = 'I')
            BEGIN
                EXEC addPerson @FirstName,@LastName

                DECLARE @PersonID int
                SELECT @PersonID = PersonID FROM Person

                INSERT INTO IndividualClient(ClientID, PersonID)
                VALUES(@ClientID, @PersonID)
            END
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Blad dodania klienta: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.12. addProductToMenu

```
CREATE PROCEDURE addProductToMenu   @Name nvarchar(150),
                                    @MenuID int,
                                    @Price money
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(
                SELECT * FROM Products WHERE Name like @Name
                )
            BEGIN;
                THROW 52000, N'Nie ma takiego produktu', 1
            END
            IF EXISTS(SELECT * FROM Products WHERE Name LIKE @Name AND
IsAvailable = 0)
                BEGIN
                    THROW 52000, N'Ten produkt jest aktualnie
niedostępny!', 1
                END

            IF NOT EXISTS(
                SELECT * FROM Menu WHERE MenuID = @MenuID
                )
            BEGIN;
                THROW 52000, N'Nie ma takiego menu. Dodaj napierw menu aby
dodać produkt!', 1
            END


            DECLARE @ProductID int
            SELECT @ProductID = ProductID from Products WHERE Name like
@Name
```

```
            IF EXISTS(SELECT * FROM MenuDetails WHERE ProductID =
@ProductID AND MenuID = @MenuID)
                BEGIN
                    THROW 52000, N'Produkt już jest w menu!', 1
                END

            INSERT INTO MenuDetails(MenuID, ProductID, Price)
            VALUES (@MenuID, @ProductID, @Price)

        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania potrawy do menu:
' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
go
```

6.13. removeProductFromMenu

```
CREATE PROCEDURE removeProductFromMenu  @Name nvarchar(150),
                                        @MenuID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(
                SELECT * FROM Products WHERE Name like @Name
                )
            BEGIN;
                THROW 52000, N'Nie ma takiej potrawy', 1
            END

            IF NOT EXISTS(
                SELECT * FROM Menu WHERE MenuID = @MenuID
                )
            BEGIN;
                THROW 52000, N'Nie ma takiego menu', 1
            END

            IF NOT EXISTS(
                SELECT * FROM MenuDetails MD
                    INNER JOIN Products P ON P.ProductID = MD.ProductID
                WHERE MenuID = @MenuID AND Name like @Name
                )
            BEGIN;
                THROW 52000, N'Nie ma takiego produktu w menu', 1
            END

            DECLARE @ProductID int
            SELECT @ProductID = ProductID from Products WHERE Name like
@Name
```

```
            DELETE FROM MenuDetails  WHERE MenuID = @MenuID and ProductID
= @ProductID

        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd usunięcia potrawy z menu:
' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.14. addMenu

```
CREATE PROCEDURE addMenu  @StartDate datetime,
                          @EndDate datetime = NULL,
                          @Description varchar(max) = NULL
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM Menu WHERE CAST(startDate AS date) =
CAST(@StartDate AS date))
                BEGIN
                    THROW 52000, N'Menu zaczynające się w ten dzień już
istnieje!', 1
                END
            IF EXISTS(SELECT * FROM Menu WHERE CAST(endDate AS date) =
CAST(@EndDate AS date))
                BEGIN
                    THROW 52000, N'Menu kończące się w ten dzień już
istnieje!', 1
                END
            IF EXISTS(SELECT * FROM Menu WHERE CAST(startDate AS date) =
CAST(@StartDate AS date) AND CAST(endDate AS date) = CAST(@EndDate AS
date))
                BEGIN
                    THROW 52000, N'Menu już istnieje!', 1
                END
            DECLARE @MenuID int
            SELECT @MenuID = ISNULL(MAX(MenuID), 0) + 1 FROM Menu

            IF @Description IS NOT NULL
                INSERT INTO Menu(MenuID,startDate, endDate, Description)
                VALUES(@MenuID, @StartDate, @EndDate, @Description)
            ELSE
                INSERT INTO Menu(MenuID,startDate, endDate)
                VALUES(@MenuID, @StartDate, @EndDate)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania menu: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
```

```
    END
GO
```

6.15. updateMenuDescription

```
CREATE PROCEDURE UpdateMenuDescription(@MenuID int, @Description varchar)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(
                    SELECT * FROM Menu WHERE MenuID = @MenuID
                )
                BEGIN;
                    THROW 52000, 'Nie ma takiego menu!',1
                END
            UPDATE Menu SET Description = @Description WHERE MenuID =
@MenuID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania/zmienienia opisu
menu: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.16. create invoice

```
CREATE PROCEDURE [create invoice] @OrderID int,
  @InvoiceDate datetime,
  @PaymentMethodName varchar(50),
  @PaymentStatusName varchar(50),
  @InvoiceID int output
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(
                SELECT OrderID FROM Orders
                WHERE OrderID = @OrderID
            )
            BEGIN;
                THROW 52000, N'Nie ma takiego zamówienia', 1
            END

            IF NOT EXISTS(
                    SELECT PaymentName FROM PaymentMethods
                    WHERE PaymentName LIKE @PaymentMethodName
            )
            BEGIN;
                THROW 52000, N'Nie ma takiej metody płatności', 1
```

```
            END

            IF NOT EXISTS(
                    SELECT PaymentStatusName FROM PaymentStatus
                    WHERE PaymentStatusName LIKE @PaymentStatusName
            )
            BEGIN;
                THROW 52000, N'Nie ma takiego statusu płatności', 1
            END

            DECLARE @invoiceNum nvarchar(50) = concat('FV/', cast(@OrderID
AS nvarchar(50)), '/', cast(year((SELECT OrderCompletionDate FROM Orders

WHERE OrderID = @OrderID)) AS nvarchar(4)))
            DECLARE @ClientID int = (SELECT ClientID FROM Orders
                                        WHERE OrderID = @OrderID)
            DECLARE @InvoiceIDs TABLE (ID int)
            DECLARE @PaymentMethodID int
            DECLARE @PaymentStatusID int

            SELECT @PaymentMethodID = PaymentMethodID FROM PaymentMethods
WHERE PaymentName LIKE @PaymentMethodName
            SELECT @PaymentStatusID = PaymentStatusID FROM PaymentStatus
WHERE PaymentStatusName LIKE @PaymentStatusName

            INSERT INTO
              Invoice(InvoiceNumber, InvoiceDate, DueDate, ClientID,
PaymentStatusID, PaymentMethodID) OUTPUT inserted.InvoiceID INTO
@InvoiceIDs
                VALUES (@invoiceNum, @InvoiceDate, dateadd(DAY, 12,
GETDATE()), @ClientID, @PaymentStatusID, @PaymentMethodID)
            SELECT @InvoiceID = ID FROM @InvoiceIDs RETURN @InvoiceID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania faktury: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
go
```

6.17. add Payment Status

```
CREATE PROCEDURE [add Payment Status] @PaymentStatusName varchar(50)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            INSERT INTO PaymentStatus(PaymentStatusName) values
(@PaymentStatusName)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania metody płatności
do zamówienia: ' + ERROR_MESSAGE();
```

```
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.18. add Payment Method

```
CREATE PROCEDURE [add Payment Method] @PaymentMethodName varchar(50)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            INSERT INTO PaymentMethods(PaymentName) values
(@PaymentMethodName)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania metody płatności
do zamówienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.19. change payment method for order

```
CREATE PROCEDURE [change payment method for order] @PaymentMethodName
varchar(50), @OrderID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT OrderID FROM Orders WHERE OrderID =
@OrderID)
                BEGIN;
                    THROW 52000, 'Brak takiego zamowienia', 1
                END
            IF NOT EXISTS(SELECT PaymentMethodID FROM PaymentMethods WHERE
PaymentName LIKE @PaymentMethodName)
                BEGIN;
                    THROW 52000, 'Brak takiej metody platnosci', 1
                END

            DECLARE @PaymentMethodID int;

            SELECT @PaymentMethodID =  PaymentMethodID FROM PaymentMethods
WHERE PaymentName LIKE @PaymentMethodName

            UPDATE Orders SET PaymentMethodID = @PaymentMethodID WHERE
OrderID = @OrderID
        END TRY
        BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Błąd zmiany metody: ' +
ERROR_MESSAGE();
                THROW 52000, @msg, 1
```

```
        END CATCH
    END
GO
```

6.20. change payment method for invoice

```
CREATE PROCEDURE [change payment method for invoice] @PaymentMethodName
varchar(50), @InvoiceID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF not EXISTS(SELECT InvoiceID FROM Invoice WHERE InvoiceID =
@InvoiceID)
                BEGIN;
                    THROW 52000, 'Brak takiego zamowienia', 1
                END
            IF not EXISTS(SELECT PaymentMethodID FROM PaymentMethods WHERE
PaymentName LIKE @PaymentMethodName)
                BEGIN;
                    THROW 52000, 'Brak takiej metody platnosci', 1
                END

            DECLARE @PaymentMethodID int;

            SELECT @PaymentMethodID =  PaymentMethodID FROM PaymentMethods
WHERE PaymentName LIKE @PaymentMethodName

            UPDATE Invoice SET PaymentMethodID = @PaymentMethodID WHERE
InvoiceID = @InvoiceID
        END TRY
        BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Błąd zmiany metody: ' +
ERROR_MESSAGE();
                THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.21. change payment status for invoice

```
CREATE PROCEDURE [change payment status for invoice] @PaymentStatusName
varchar(50), @InvoiceID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT InvoiceID FROM Invoice WHERE InvoiceID =
@InvoiceID)
                BEGIN;
                    THROW 52000, 'Brak takiego zamowienia', 1
                END
            IF NOT EXISTS(select PaymentStatusID FROM PaymentStatus WHERE
PaymentStatus.PaymentStatusName LIKE @PaymentStatusName)
```

```
                BEGIN;
                    THROW 52000, 'Brak takiego statusu platnosci', 1
                END
            DECLARE @PaymentStatusID int;

            SELECT @PaymentStatusID = PaymentStatusID FROM PaymentStatus
WHERE PaymentStatus.PaymentStatusName LIKE @PaymentStatusName

            UPDATE Invoice SET PaymentStatusID = @PaymentStatusID WHERE
InvoiceID = @InvoiceID
        END TRY
        BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Błąd zmiany statusu: ' +
ERROR_MESSAGE();
                THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.22. change payment status for order

```
CREATE PROCEDURE [change payment status for order] @PaymentStatusName
varchar(50), @OrderID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT OrderID FROM Orders WHERE OrderID =
@OrderID)
                BEGIN;
                    THROW 52000, 'Brak takiego zamowienia', 1
                END
            IF NOT EXISTS(SELECT PaymentStatusID FROM PaymentStatus WHERE
PaymentStatus.PaymentStatusName LIKE @PaymentStatusName)
                BEGIN;
                    THROW 52000, 'Brak takiego statusu platnosci', 1
                END

            DECLARE @PaymentStatusID int;

            SELECT @PaymentStatusID = PaymentStatusID FROM PaymentStatus
WHERE PaymentStatus.PaymentStatusName LIKE @PaymentStatusName

            UPDATE Orders SET PaymentStatusID = @PaymentStatusID WHERE
OrderID = @OrderID
        END TRY
        BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Błąd zmiany statusu: ' +
ERROR_MESSAGE();
                THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.23. AddOrderInstantPay

@OrderStatus przyjmowane są te które zostały zdefiniowane w tabeli Orders

```
CREATE PROCEDURE AddOrderInstantPay @ClientID int,
                                    @OrderCompletionDate datetime = null,
                                    @PaymentStatusName_ varchar(50),
                                    @PaymentMethodName_ varchar(50),
                                    @OrderStatus varchar(15),
                                    @StaffID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF @OrderCompletionDate IS NOT NULL AND  @OrderCompletionDate <=
GETDATE()
            BEGIN
                THROW 52000, N'Data ukończenia zamówienia musi być większa
od aktualnej!', 1
            END
        IF NOT EXISTS(SELECT PaymentStatusID FROM PaymentStatus WHERE
PaymentStatusName LIKE @PaymentStatusName_)
            BEGIN;
                THROW 52000, 'Nie ma takiego statusu!', 1
            END
        IF NOT EXISTS(select PaymentMethods.PaymentName FROM
PaymentMethods WHERE PaymentMethods.PaymentName LIKE @PaymentMethodName_)
            BEGIN;
                THROW 52000, 'Nie ma takiej metody!', 1
            END
        IF NOT EXISTS(SELECT StaffID FROM Staff WHERE StaffID = @StaffID)
            BEGIN;
                THROW 52000, 'Nie ma takiego pracownika!', 1
            END

        DECLARE @OrderIDTable TABLE
                (
                    Id int
                )

        DECLARE @OrderID int
        DECLARE @PaymentMethodID int
        DECLARE @PaymentStatusID int
        DECLARE @InvoiceID int

        SELECT @PaymentStatusID =  PaymentStatusID FROM PaymentStatus
WHERE PaymentStatusName LIKE @PaymentStatusName_
        SELECT @PaymentMethodID =  PaymentMethods.PaymentMethodID FROM
PaymentMethods WHERE PaymentMethods.PaymentName LIKE @PaymentMethodName_

        INSERT INTO Orders (ClientID, PaymentStatusID, PaymentMethodID,
staffID, OrderSum, OrderCompletionDate, OrderStatus, OrderDate)
        OUTPUT inserted.OrderID INTO @OrderIDTable
        VALUES (@ClientID, @PaymentStatusID, @PaymentMethodID, @StaffID ,
0.0, @OrderCompletionDate, @OrderStatus, GETDATE());
```

```
        SELECT @OrderID = Id FROM @OrderIDTable
        EXEC dbo.[create invoice] @OrderID = @OrderID, @InvoiceDate =
@OrderCompletionDate, @PaymentMethodName = @PaymentMethodName_,
@PaymentStatusName = @PaymentStatusName_, @InvoiceID = @InvoiceID OUTPUT
        UPDATE [Orders] SET InvoiceID= @InvoiceID WHERE OrderID = @OrderID
        RETURN @OrderID
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) = N'Błąd dodania zamowienia: ' +
ERROR_MESSAGE();
            THROW 52000, @msg,1
    END CATCH
END
go
```

6.24. addOrderMonthPay

@OrderStatus przyjmowane są te które zostały zdefiniowane w tabeli Orders

```
CREATE PROCEDURE AddOrderMonthPay   @ClientID int,
                                    @OrderCompletionDate datetime,
                                    @PaymentStatusName_ varchar(50),
                                    @PaymentMethodName_ varchar(50),
                                    @OrderStatus varchar(15),
                                    @StaffID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF @OrderCompletionDate IS NOT NULL AND  @OrderCompletionDate <=
GETDATE()
            BEGIN
                THROW 52000, N'Data ukończenia zamówienia musi być większa
od aktualnej!', 1
            END
        IF NOT EXISTS(SELECT PaymentStatusID FROM PaymentStatus WHERE
PaymentStatusName LIKE @PaymentStatusName_)
            BEGIN;
                THROW 52000, 'Nie ma takiego statusu!', 1
            END
        IF not EXISTS(SELECT PaymentMethods.PaymentName FROM
PaymentMethods WHERE PaymentMethods.PaymentName LIKE @PaymentMethodName_)
            BEGIN;
                THROW 52000, 'Nie ma takiej metody!', 1
            END
        IF NOT EXISTS(SELECT StaffID FROM Staff WHERE StaffID = @StaffID)
            BEGIN;
                THROW 52000, 'Nie ma takiego pracownika!', 1
            END

        Declare @OrderIDTable table
                (
                    Id int
                )
```

```
        Declare @OrderID int
        DECLARE @PaymentMethodID int
        DECLARE @PaymentStatusID int
        DECLARE @startOfMonth datetime = cast(DATEADD(month,
DATEDIFF(month, 0, @OrderCompletionDate) + 1, 0) AS date)
        DECLARE @InvoiceID int

        SELECT @PaymentStatusID =  PaymentStatusID FROM PaymentStatus
WHERE PaymentStatusName LIKE @PaymentStatusName_
        SELECT @PaymentMethodID =  PaymentMethods.PaymentMethodID FROM
PaymentMethods WHERE PaymentMethods.PaymentName LIKE @PaymentMethodName_

        INSERT INTO Orders (ClientID, PaymentStatusID, PaymentMethodID,
staffID, OrderSum, OrderCompletionDate, OrderStatus, OrderDate)
        OUTPUT inserted.OrderID INTO @OrderIDTable
        VALUES (@ClientID, @PaymentStatusID, @PaymentMethodID, @StaffID ,
0.0, @OrderCompletionDate, @OrderStatus, GETDATE());

        SELECT @OrderID = Id FROM @OrderIDTable


        SELECT @InvoiceID = InvoiceID FROM Invoice
        WHERE ClientID = @ClientID
            AND month(InvoiceDate) = month(@startOfMonth)
            AND year(InvoiceDate) = year(@startOfMonth)

        IF @InvoiceID IS NULL
            BEGIN;
                EXEC dbo.[create invoice] @OrderID = @OrderID,
@InvoiceDate = @startOfMonth, @PaymentMethodName = @PaymentMethodName_,
@PaymentStatusName = @PaymentStatusName_, @InvoiceID = @InvoiceID OUTPUT
            END

        UPDATE [Orders] SET InvoiceID= @InvoiceID WHERE OrderID = @OrderID
        RETURN @OrderID
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) = N'Błąd dodania zamowienia: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END;
go
```

6.25. addStaffMember

```
CREATE PROCEDURE addStaffMember @LastName nvarchar(50), @FirstName
nvarchar(70), @Position varchar(50), @Email varchar(100), @Phone
varchar(14), @AddressID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM Staff WHERE Email LIKE @Email)
            BEGIN;
```

```
                THROW 52000, 'Pracownik o takim emailu już istnieje!', 1
            END

            IF NOT EXISTS(SELECT * FROM Address WHERE AddressID =
@AddressID)
            BEGIN;
                THROW 52000, 'Nie ma takiego adresu!', 1
            END
            IF EXISTS(SELECT * FROM Staff WHERE LastName = @LastName AND
FirstName = @FirstName AND Position = @Position AND Email = @Email AND
Phone = @Phone AND AddressID = @AddressID)
            BEGIN;
                THROW 52000, 'Taki pracownik już istnieje!', 1
            END


            INSERT INTO Staff (LastName, FirstName, Position, Email,
Phone, AddressID)
            VALUES (@LastName, @FirstName, @Position, @Email, @Phone,
@AddressID);
        END TRY
        BEGIN CATCH
            DECLARE @msg varchar(2048) = N'Błąd dodania nowego pracownika:
' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.26. addProduct

```
CREATE PROCEDURE addProduct @CategoryID int, @Name nvarchar(50),
@Description nvarchar(150) = NULL, @IsAvailable bit = NULL
AS
    BEGIN
       SET NOCOUNT ON
       BEGIN TRY
           IF EXISTS(SELECT * FROM Products WHERE Name LIKE @Name)
           BEGIN;
                THROW 52000, N'Taki produkt już istnieje!', 1
            END
            IF @Description IS NULL AND @IsAvailable IS NOT NULL
            BEGIN
                INSERT INTO Products(CategoryID, Name, IsAvailable)
                VALUES (@CategoryID, @Name, @IsAvailable)
            END
            ELSE IF @Description IS NOT NULL AND @IsAvailable IS NULL
            BEGIN
                INSERT INTO Products(CategoryID, Name, Description)
                VALUES (@CategoryID, @Name, @Description)
            END
            ELSE
            BEGIN
                INSERT INTO Products(CategoryID, Name, Description,
IsAvailable)
```

```
                VALUES (@CategoryID, @Name, @Description, @IsAvailable)
            END
        END TRY
        BEGIN CATCH
            DECLARE @msg varchar(2048) = N'Błąd dodania nowego produktu: '
+ ERROR_MESSAGE()
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.27. removeCity

```
CREATE PROCEDURE removeCity @CityID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Cities WHERE CityID = @CityID)
            BEGIN
                THROW 52000, 'Nie ma takiego miasta!', 1
            END
            DELETE FROM Cities WHERE CityID = @CityID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd usunięcia miasta: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.28. removeCategory

```
CREATE PROCEDURE removeCategory @CategoryID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Category WHERE CategoryID =
@CategoryID)
            BEGIN
                THROW 52000, 'Nie ma takiej kategorii!', 1
            END

            DELETE FROM Category WHERE CategoryID = @CategoryID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd usunięcia kategorii: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.29. removeProduct

```
CREATE PROCEDURE removeProduct @ProductID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Products WHERE ProductID =
@ProductID)
            BEGIN;
                THROW 52000, 'Nie ma takiego produktu!', 1
            END
            DELETE FROM Products WHERE ProductID = @ProductID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd usunięcia produktu: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.30. changeOrderStatus

```
CREATE PROCEDURE changeOrderStatus @OrderID int, @OrderStatus varchar(15)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
            BEGIN;
                THROW 52000, N'Nie ma takiego zamówienia!', 1
            END
            IF LOWER(@OrderStatus) NOT IN('pending', 'accepted',
'completed', 'denied', 'picked', 'cancelled')
                BEGIN
                    THROW 52000, N'Nie ma takiego statusu zamówienia! ', 1
                END
            IF LOWER(@OrderStatus) LIKE 'picked' AND NOT EXISTS(SELECT  *
FROM Orders INNER JOIN OrdersTakeaways OT ON Orders.TakeawayID =
OT.TakeawaysID WHERE OrderID = @OrderID)
                BEGIN
                    THROW 52000, N'To zamówienie nie jest na wynos!', 1
                END
            UPDATE Orders SET OrderStatus = @OrderStatus WHERE OrderID =
@OrderID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd odrzucenia zamówienia: '
+ ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
```

```
    END
GO
```

6.31. changeEmployeeResponsibleForOrder

```
CREATE PROCEDURE changeEmployeeResponsibleForOrder @OrderID int, @StaffID
int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Orders WHERE ReservationID =
@OrderID)
                BEGIN;
                    THROW 52000, N'Nie ma takiego zamówienia!', 1
                END
            IF NOT EXISTS(SELECT * FROM Staff WHERE StaffID = @StaffID)
                BEGIN;
                    THROW 52000, N'Nie ma takiego pracownika!', 1
                END
            UPDATE Orders SET StaffID = @StaffID WHERE ReservationID =
@OrderID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd w zmianie pracownika: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO;
```

6.32. changeEmployeeResposibleForReservation

```
CREATE PROCEDURE changeEmployeeResponsibleForReservation @ReservationID
int, @StaffID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Reservation WHERE ReservationID =
@ReservationID)
                BEGIN;
                    THROW 52000, N'Nie ma takiej rezerwacji!', 1
                END
            IF NOT EXISTS(SELECT * FROM Staff WHERE StaffID = @StaffID)
                BEGIN;
                    THROW 52000, N'Nie ma takiego pracownika!', 1
                END
            UPDATE Reservation SET StaffID = @StaffID WHERE ReservationID
= @ReservationID
        END TRY
```

```
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd w zmianie pracownika: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO;
```

6.33. showBestDiscount

```
CREATE PROCEDURE showBestDiscount @ClientID int, @DiscountType varchar
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Clients WHERE ClientID =
@ClientID)
            BEGIN;
                THROW 52000, N'Nie ma takiego klienta!', 1
            END
            IF LOWER(@DiscountType) LIKE 'temporary'
                BEGIN;
                    SELECT max(DiscountValue) AS 'Discount Value' FROM
IndividualClient I
                        INNER JOIN Discounts D ON I.ClientID = D.ClientID
                        INNER JOIN DiscountsVar DV ON DV.VarID = D.VarID
                    WHERE DiscountType = 'Temporary'
                        AND I.ClientID = @ClientID
                        AND AppliedDate <= getdate() AND GETDATE() <=
dateadd(DAY, ValidityPeriod, AppliedDate)
                END
            ELSE IF LOWER(@DiscountType) LIKE 'permanent'
                BEGIN;
                    SELECT max(DiscountValue) AS 'Discount Value' FROM
IndividualClient I
                        INNER JOIN Discounts D ON I.ClientID = D.ClientID
                        INNER JOIN DiscountsVar DV ON DV.VarID = D.VarID
                    WHERE DiscountType = 'Permanent'
                        AND I.ClientID = @ClientID
                        AND AppliedDate <= getdate() AND GETDATE() <=
dateadd(DAY, ValidityPeriod, AppliedDate)
                END
            ELSE
                BEGIN
                   THROW 52000, N'Nie ma takiego typu zniżki', 1
                END
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd wyświetlenia zniżki: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.34. changeReservationStatus

```
CREATE PROCEDURE changeReservationStatus @ReservationID int, @Status
varchar(15)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Reservation WHERE ReservationID =
@ReservationID)
                BEGIN;
                    THROW 52000, N'Nie ma takiej rezerwacji!', 1
                END
            UPDATE Reservation SET Status = @Status
            WHERE Reservation.ReservationID = @ReservationID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd edytowania rezerwacji: '
+ ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.35. AddProductToOrder

```
CREATE PROCEDURE AddProductToOrder @OrderID int,
                                   @Quantity int,
                                   @ProductName nvarchar(50)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
        IF NOT EXISTS(SELECT * FROM Products WHERE Name = @ProductName)
          BEGIN;
                THROW 52000, N'Nie ma takiej potrawy', 1
          END
        IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
            BEGIN;
                THROW 52000, 'Nie ma takiego zamowienia', 1
            END
        IF NOT EXISTS(SELECT * FROM CurrentMenu CM WHERE CM.Name like
@ProductName)
        BEGIN;
            THROW 52000, N'Nie mozna zamowic tego produktu, gdyz nie ma go
obecnie w menu', 1
        END

        DECLARE @OrderDate DATE

        SELECT @OrderDate = OrderDate FROM Orders WHERE OrderID = @OrderID
DECLARE @CategoryName nvarchar(50)
        SELECT @CategoryName = CategoryName FROM Products
```

```
            INNER JOIN Category ON Category.CategoryID =
Products.CategoryID
        WHERE
        Products.Name = @ProductName

        DECLARE @ProductID INT
        SELECT @ProductID = ProductID FROM Products WHERE Name =
@ProductName

        IF EXISTS(SELECT * FROM OrderDetails WHERE OrderID = @OrderID AND
ProductID = @ProductID)
        BEGIN;
            THROW 52000, N'Produkt jest już w zamówieniu!', 1
        END

        DECLARE @BasePrice money
        SELECT @BasePrice = Price from CurrentMenu CM where CM.Name LIKE
@ProductName

        DECLARE @CurrentValue money
        DECLARE @ClientID int
        DECLARE @DiscMulti decimal(3, 2);

        SELECT @CurrentValue = OrderSum FROM [Orders] WHERE OrderID =
@OrderId
        SELECT @ClientID = ClientID, @OrderDate = OrderDate FROM Orders
WHERE OrderID = @OrderId
        SELECT @DiscMulti = dbo.calculateDiscountForClient(@ClientID)

        INSERT INTO OrderDetails(OrderID, Quantity, ProductID) VALUES
(@OrderID, @Quantity, @ProductID)

        IF @DiscMulti IS NOT NULL
            BEGIN
                UPDATE Orders SET OrderSum = @CurrentValue + (@BasePrice *
@Quantity * @DiscMulti) WHERE OrderID = @OrderID
            END
        ELSE
            BEGIN
                UPDATE Orders SET OrderSum = @CurrentValue + (@BasePrice *
@Quantity * 1) WHERE OrderID = @OrderID
            END
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania produktu do
zamowienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
go
```

6.36. EmployeeAssignedToTheOrder

```
CREATE PROCEDURE EmployeeAssignedToTheOrder @OrderID int
```

```
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
        IF NOT EXISTS(SELECT * FROM Orders
                        WHERE OrderID = @OrderID
        )
        BEGIN;
            THROW 52000, 'Nie ma takiego zamowienia', 1
        END
        SELECT S.FirstName, S.LastName, S.Position, S.Email, S.Phone,
O.OrderID, O.OrderStatus, O.OrderDate FROM Staff AS S
            INNER JOIN Orders O ON O.StaffID = S.StaffID
        WHERE
            O.OrderID = @OrderID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd wypisywania pracownikow:'
+ ERROR_MESSAGE();
            THROW 52000,@msg,1
        END CATCH
    END
GO
```

6.37. getDishesForDay

```
CREATE PROCEDURE dbo.getDishesForDay @data Date
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            SELECT O.OrderID, cast(OrderCompletionDate AS Date) AS
'OrderCompletionDate', P.Name, sum(OD.Quantity) AS 'Quantity' FROM Orders
O
                INNER JOIN OrderDetails OD ON O.OrderID = OD.OrderID
                INNER JOIN Products P ON OD.ProductID = P.ProductID
            WHERE cast(OrderCompletionDate AS Date) = @data
            GROUP BY O.OrderID, O.OrderCompletionDate, P.Name
        END TRY
        BEGIN CATCH
            DECLARE @msg varchar(2048) = N'Blad wyświetlenia danych: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.38. ClientStaticstics

```
CREATE PROCEDURE Client_Statistics @ClientID int
AS
    BEGIN
        SET NOCOUNT ON
```

```
        BEGIN TRY
            IF NOT EXISTS(SELECT ClientID FROM Clients WHERE ClientID =
@ClientID)
                BEGIN;
                    throw 52000, 'Nie ma takiego klienta!', 1
                END

            DECLARE @PaymentStatusID int

            SELECT @PaymentStatusID = PaymentStatusID FROM PaymentStatus
WHERE PaymentStatusName LIKE 'Paid'

            SELECT O.OrderID, O.OrderDate, O.OrderSum, O.OrderSum -
O2.no_disc AS [discount value], 1 - (O2.no_disc/O.OrderSum) AS [discount
multiplier] FROM Orders O
                INNER JOIN (SELECT O.OrderID, sum(Quantity) AS no_disc
                            FROM Orders O
                                INNER JOIN OrderDetails OD ON O.OrderID =
OD.OrderID
                                INNER JOIN Products P ON OD.ProductID =
P.ProductID
                                INNER JOIN MenuDetails MD ON P.ProductID =
MD.ProductID
                                INNER JOIN Menu M ON MD.MenuID = M.MenuID
                            WHERE M.startDate <= GETDATE() AND (M.endDate
IS NULL OR M.endDate >= getdate())
                            GROUP BY O.OrderID) O2 ON O2.OrderID =
O.OrderID
                WHERE ClientID = @ClientID AND PaymentStatusID =
@PaymentStatusID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd wyswietlenia statystyk o
kliencie: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.39. AddPaymentStatus

```
CREATE PROCEDURE AddPaymentStatus @PaymentStatusName varchar(50)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM PaymentStatus WHERE PaymentStatusName
= @PaymentStatusName)
                BEGIN;
                    THROW 52000, 'Istnieje już taki status płatności', 1
                END
            INSERT INTO PaymentStatus (PaymentStatusName) VALUES
(@PaymentStatusName)
        END TRY
        BEGIN CATCH
```

```
            DECLARE @msg nvarchar(2048) = 'Błąd dodania statusu płatności:
' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.40. RemovePaymentStatus

```
CREATE PROCEDURE RemovePaymentStatus @PaymentStatusID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM PaymentStatus WHERE
PaymentStatusID = @PaymentStatusID)
                BEGIN;
                    THROW 52000, 'Nie ma takiego statusu płatności', 1
                END
            DELETE FROM PaymentStatus WHERE PaymentStatusID =
@PaymentStatusID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd usunięcia statusu
płatności: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.41. RemovePaymentMethod

```
CREATE PROCEDURE RemovePaymentMethod @PaymentMethodID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM PaymentMethods WHERE
PaymentMethodID = @PaymentMethodID)
                BEGIN;
                    THROW 52000, 'Nie ma takiej metody płatności', 1
                END
            DELETE FROM PaymentMethods WHERE PaymentMethodID =
@PaymentMethodID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd usunięcia metody
płatności: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.42. AddPaymentMethod

```
CREATE PROCEDURE AddPaymentMethod @PaymentMethodName varchar(50)
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM PaymentMethods WHERE PaymentName =
@PaymentMethodName)
                BEGIN;
                    THROW 52000, 'Istnieje już taka metoda płatności', 1
                END
            INSERT INTO PaymentMethods (PaymentName) VALUES
(@PaymentMethodName)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania metody płatności:
' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.43. AddTakeAway

```
CREATE PROCEDURE AddTakeaway @PrefDate datetime
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            INSERT INTO OrdersTakeaways (PrefDate) VALUES (@PrefDate)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania zamowienia: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.44. AddTakeawayToOrder

```
CREATE PROCEDURE AddTakeawayToOrder @OrderID int, @PrefDate datetime
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
                BEGIN;
                    THROW 52000, 'Nie ma takiego zamowienia', 1
                END
```

```
            IF @PrefDate < GETDATE()
                BEGIN
                    THROW 52000, N'Data nie może być wcześniejsza niż
dzisiejsza!', 1
                END
            EXEC AddTakeaway @PrefDate
            DECLARE @TakeawayID int;
            SELECT @TakeawayID = MAX(TakeawaysID) FROM OrdersTakeaways

            UPDATE Orders SET TakeawayID = @TakeawayID WHERE OrderID =
@OrderID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania zamowienia do
zamowienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.45. AddReservationToOrder

```
CREATE PROCEDURE AddReservationToOrder @OrderID int, @ReservationID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
                BEGIN;
                    THROW 52000, 'Nie ma takiego zamowienia', 1
                END

            IF NOT EXISTS(SELECT * FROM Reservation WHERE ReservationID =
@ReservationID)
                BEGIN;
                    THROW 52000, 'Nie ma takiego rezerwacji', 1
                END
            DECLARE @ReservationIDAssignmentToOrder int
            SET @ReservationIDAssignmentToOrder = (SELECT ReservationID
FROM Orders WHERE OrderID = @OrderID)
            IF @ReservationIDAssignmentToOrder IS NOT NULL
                BEGIN
                    THROW 52000, N'To zamówienie ma już swoją
rezerwację!', 1
                END

            UPDATE Orders SET ReservationID = @ReservationID WHERE OrderID
= @OrderID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania rezerwacji do
zamowienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
```

```
    END
go
```

6.46. AddReservation

```
CREATE PROCEDURE AddReservation @ClientID int, @OrderID int , @StartDate
datetime, @EndDate datetime, @StaffID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF @StartDate >= @EndDate
                BEGIN
                    THROW 52000, 'Data końca musi być większa od startu!',
1
                END
            IF NOT EXISTS(SELECT * FROM Clients WHERE ClientID =
@ClientID)
                BEGIN;
                        THROW 52000, N'Nie ma takiego klienta', 1
                END

            IF NOT EXISTS(SELECT * FROM Staff WHERE StaffID = @StaffID)
                BEGIN;
                        THROW 52000, N'Nie ma takiego pracownika', 1
                END

            DECLARE @ReservationIDAssignmentToOrder int
            SET @ReservationIDAssignmentToOrder = (SELECT ReservationID
FROM Orders WHERE OrderID = @OrderID)
            IF @ReservationIDAssignmentToOrder IS NOT NULL
                BEGIN
                    THROW 52000, N'To zamówienie ma już swoją
rezerwację!', 1
                END

            DECLARE @ReservationID int
            DECLARE @PersonID int

            SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 FROM
Reservation


            INSERT INTO Reservation(ReservationID, startDate, endDate,
Status, StaffID)
            VALUES (@ReservationID,@StartDate, @EndDate, 'waiting',
@StaffID)

            IF EXISTS(SELECT * FROM Companies WHERE ClientID = @ClientID)
                BEGIN;
                    INSERT INTO ReservationCompany(ReservationID,
ClientID, PersonID)
                    VALUES (@ReservationID, @ClientID, null)
                END
            ELSE
```

```
                BEGIN;
                    SELECT @PersonID=PersonID from IndividualClient WHERE
ClientID = @ClientID
                    INSERT INTO ReservationIndividual(ReservationID,
ClientID, PersonID)
                    VALUES (@ReservationID, @ClientID, @PersonID)
                END
            EXEC AddReservationToOrder @OrderID, @ReservationID
        END TRY
        BEGIN CATCH
                DECLARE @msg nvarchar(2048) = 'Błąd dodania rezerwacji: '
+ ERROR_MESSAGE();
                THROW 52000, @msg, 1
        END CATCH
    END
go
```

6.47 RemoveReservation

```
CREATE PROCEDURE RemoveReservation @ReservationID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Reservation WHERE ReservationID =
@ReservationID)
                BEGIN;
                    THROW 52000, 'Nie ma takiej rezerwacji', 1
                END

            DELETE FROM Reservation WHERE ReservationID = @ReservationID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd usunięcia rezerwacji: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.48. addTableToReservation

```
CREATE PROCEDURE addTableToReservation @ReservationID int, @TableID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM TABLES WHERE TableID = @TableID)
```

```
            BEGIN;
                THROW 52000, 'Nie ma takiego stolika! ', 1
            END

            IF NOT EXISTS(SELECT * FROM Orders WHERE ReservationID =
@ReservationID)
            BEGIN;
                THROW 52000, 'Nie ma takiej rezerwacji! ', 1
            END

            INSERT INTO ReservationDetails(ReservationID, TableID)
            VALUES (@ReservationID, @TableID)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania stolika do
rezerwacji: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.49. removeTableFromReservation

```
CREATE PROCEDURE removeTableFromReservation @ReservationID int, @TableID
int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM TABLES WHERE TableID = @TableID)
            BEGIN;
                THROW 52000, 'Nie ma takiego stolika! ', 1
            END

            IF NOT EXISTS(SELECT * FROM Orders WHERE ReservationID =
@ReservationID)
            BEGIN;
                THROW 52000, 'Nie ma takiej rezerwacji! ', 1
            END
            DELETE FROM ReservationDetails WHERE ReservationID =
@ReservationID AND TableID = @TableID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania stolika do
rezerwacji: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.50. AddReservationVar

```
CREATE PROCEDURE AddReservationVar @WK int, @WZ money, @startDate
```

```
datetime, @endDate datetime = NULL
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM ReservationVar WHERE WZ = @WZ AND WK =
@WK AND startDate = @startDate AND endDate = @endDate)
                BEGIN;
                    THROW 52000, 'Istnieje już taka zmienna dotycząca
rezerwacji', 1
                END
            INSERT INTO ReservationVar (WZ, WK, startDate, endDate) VALUES
(@WZ, @WK, @startDate, @endDate)

        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd dodania zmiennej
dotyczącej rezerwacji: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.51 AddDiscountVar

Jeśli dodajemy zniżke tymczasową to dodajemy @ValidityPeriod, a jeżeli pernamentną to @MinimalOrders

```
CREATE PROCEDURE AddDiscountVar @MinimalOrders int = NULL, @MinimalValue
money, @ValidityPeriod int = NULL, @DiscountValue decimal(3,2), @StartDate
datetime, @EndDate datetime = NULL
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF EXISTS(SELECT * FROM DiscountsVar WHERE ((MinimalOrders =
@MinimalOrders AND ValidityPeriod = @ValidityPeriod) OR
MinimalAggregateValue = @MinimalValue)  AND DiscountValue = @DiscountValue
AND startDate = @StartDate AND endDate = @EndDate)
                BEGIN;
                    THROW 52000, N'Istnieje już taka zmienna dotycząca
rabatu!', 1
                END
            IF @MinimalOrders IS NULL AND @ValidityPeriod IS NULL
                BEGIN;
                    THROW 52000, N'Nie można dodać zmiennych bez warunków!
Podaj @MinimalOrders dla zniżki permanentnej lub @ValidityPeriod dla
zniżki tymczasowej', 1
                END

            IF @ValidityPeriod IS NOT NULL AND @MinimalOrders IS NULL
                BEGIN;
                    INSERT INTO DiscountsVar (DiscountType,MinimalOrders,
MinimalAggregateValue, ValidityPeriod, DiscountValue, startDate, endDate)
VALUES ('Temporary',@MinimalOrders, @MinimalValue, @ValidityPeriod,
```

```
@DiscountValue, @StartDate, @EndDate)
                END

            IF @MinimalOrders IS NOT NULL AND @ValidityPeriod IS NULL
                BEGIN;
                    INSERT INTO DiscountsVar (DiscountType,MinimalOrders,
MinimalAggregateValue, ValidityPeriod, DiscountValue, startDate, endDate)
VALUES ('Permanent',@MinimalOrders, @MinimalValue, @ValidityPeriod,
@DiscountValue, @StartDate, @EndDate)
                END

        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania zmiennej
dotyczącej rabatu: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
go
```

6.52. addDiscount

@DiscountType:

- Permanent
- Temporary

```
-   CREATE PROCEDURE addDiscount @ClientID int, @DiscountType char(9)
-   AS
-       BEGIN
-           BEGIN TRY
-               IF NOT EXISTS(SELECT * FROM IndividualClient WHERE
    ClientID = @ClientID)
-                   BEGIN
-                       THROW 52000, N'Nie ma takiego klienta
    indywidualnego! ', 1
-                   END
-               IF LOWER(@DiscountType) NOT IN('permanent', 'temporary')
-                   BEGIN
-                       THROW 52000, N'Nie ma takiego typu zniżki! ', 1
-                   END
-
-               DECLARE @VarID int
-               SET @VarID = (SELECT VarID FROM DiscountsVar WHERE
    LOWER(DiscountType) LIKE LOWER(@DiscountType)  AND (startDate <=
    GETDATE() AND (endDate IS NULL OR endDate >= GETDATE())))
-
-               IF EXISTS(SELECT * FROM Discounts WHERE ClientID =
    @ClientID AND CAST(AppliedDate AS date) = CAST(GETDATE() AS DATE)
    AND VarID = @VarID)
-                   BEGIN
-                       return
-                   END
-
```

```
-
-               IF @DiscountType LIKE 'Permanent'
                    BEGIN
                        INSERT INTO Discounts(ClientID, VarID,
    AppliedDate, isUsed)
                        VALUES(@ClientID, @VarID, GETDATE(), NULL)
                    END
-               ELSE
                    BEGIN
                        INSERT INTO Discounts(ClientID, VarID,
    AppliedDate)
                        VALUES(@ClientID, @VarID, GETDATE())
                    END
-           END TRY
-           BEGIN CATCH
                DECLARE @msg nvarchar(2048) = N'Błąd dodania zmiennej
    dotyczącej rabatu: ' + ERROR_MESSAGE();
-               THROW 52000, @msg, 1
-           END CATCH
-       END
-   go
-
```

6.53. addEmployeeToCompany

```
CREATE PROCEDURE addEmployeeToCompany @CompanyID int, @PersonID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Companies WHERE ClientID =
@CompanyID)
            BEGIN;
                THROW 52000, N'Nie ma takiej firmy! ', 1
            END

            IF NOT EXISTS(SELECT * FROM Person WHERE PersonID = @PersonID)
            BEGIN;
                THROW 52000, N'Nie ma takiej osoby! ', 1
            END

            INSERT INTO Employees(PersonID, CompanyID)
            VALUES (@PersonID, @CompanyID)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodania pracownika do
firmy: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

6.54. removeEmployeeFromCompany

```
CREATE PROCEDURE removeEmployeeFromCompany @CompanyID int, @PersonID int
AS
    BEGIN
        SET NOCOUNT ON
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Companies WHERE ClientID =
@CompanyID)
            BEGIN;
                THROW 52000, N'Nie ma takiej firmy! ', 1
            END

            IF NOT EXISTS(SELECT * FROM Person WHERE PersonID = @PersonID)
            BEGIN;
                THROW 52000, N'Nie ma takiej osoby! ', 1
            END

            DELETE FROM Employees WHERE PersonID = @PersonID AND CompanyID
= @CompanyID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd usunięcia pracownika z
firmy: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
    END
GO
```

7. Funkcje

7.1. GetAvgPriceOfMenu

```
CREATE FUNCTION GetAvgPriceOfMenu(@MenuID int) RETURNS money AS BEGIN
RETURN (
    SELECT
        AVG(Price)
    FROM
        MenuDetails
    WHERE
        MenuID = @MenuID
) END
GO
```

7.2. GetMinimumPriceOfMenu

```
CREATE FUNCTION GetMinimumPriceOfMenu(@MenuID int) RETURNS money AS BEGIN
RETURN (
        SELECT
            TOP 1 MIN(Price)
        FROM
            MenuDetails
        WHERE
            MenuID = @MenuID
    ) END
GO
```

7.3. GetMaximumPriceOfMenu

```
CREATE FUNCTION GetMaximumPriceOfMenu(@MenuID int) RETURNS money AS BEGIN
RETURN (
        SELECT
            TOP 1 MAX(Price)
        FROM
            MenuDetails
        WHERE
            MenuID = @MenuID
    ) END
GO
```

7.4. getNotReservedTablesOnAParticularDay

```
CREATE FUNCTION getNotReservedTablesOnAParticularDay(@Date datetime)
RETURNS TABLE
    AS
        RETURN (SELECT TableID, ChairAmount FROM Tables
                    WHERE TableID NOT IN(SELECT ReservationDetails.TableID
FROM ReservationDetails
                                                INNER JOIN ReservationCompany
RC ON RC.ReservationID = ReservationDetails.ReservationID
```

```
                                                INNER JOIN Reservation R2 ON
RC.ReservationID = R2.ReservationID
                                            WHERE
                                                (CAST(@Date AS date) =
CAST(startDate AS date))
                                                AND (CAST(@Date AS date) =
CAST(endDate AS date))
                                                AND (STATUS NOT LIKE
'cancelled' AND STATUS NOT LIKE 'denied')
                                                AND isActive = 1
                                            ) AND isActive = 1
                UNION
                SELECT TableID, ChairAmount FROM Tables
                    WHERE TableID NOT IN(SELECT ReservationDetails.TableID
FROM ReservationDetails
                                                INNER JOIN
ReservationIndividual RC ON RC.ReservationID =
ReservationDetails.ReservationID
                                                INNER JOIN Reservation R2 ON
RC.ReservationID = R2.ReservationID
                                             WHERE
                                                   (CAST(@Date AS date) =
CAST(startDate AS date))
                                                   AND (CAST(@Date AS date) =
CAST(endDate AS date))
                                                   AND (
                                                       STATUS NOT LIKE
'cancelled'
                                                       AND STATUS NOT LIKE
'denied'
                                                   )
                                                   AND isActive = 1
                                            ) AND isActive = 1
            )
GO
```

7.5. showTakenTablesFromXToYWithZChairs

```
CREATE FUNCTION showTakenTablesFromXToYWithZChairs(
        @StartDate datetime,
        @EndDate datetime,
        @Chairs int
    ) RETURNS TABLE AS RETURN
SELECT
    T.TableID,
    T.ChairAmount,
    O.ClientID,
    O.OrderID,
    R2.startDate,
    R2.endDate
FROM
    TABLES T
    INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
```

```
    INNER JOIN ReservationCompany RC ON RC.ReservationID =
RD.ReservationID
    INNER JOIN Reservation R2 ON RC.ReservationID = R2.ReservationID
    INNER JOIN Orders O ON R2.ReservationID = O.ReservationID
WHERE
    R2.startDate >= @StartDate
    AND R2.endDate <= @EndDate
    AND T.ChairAmount = @Chairs
UNION
SELECT
    T.TableID,
    T.ChairAmount,
    O.ClientID,
    O.OrderID,
    R2.startDate,
    R2.endDate
FROM
    TABLES T
    INNER JOIN ReservationDetails RD ON T.TableID = RD.TableID
    INNER JOIN ReservationIndividual RC ON RC.ReservationID =
RD.ReservationID
    INNER JOIN Reservation R2 ON RC.ReservationID = R2.ReservationID
    INNER JOIN Orders O ON R2.ReservationID = O.ReservationID
WHERE
    R2.startDate >= @StartDate
    AND R2.endDate <= @EndDate
    AND T.ChairAmount = @Chairs
GO
```

7.6. ShowFreeTablesFromXToYWithZChairs

```
CREATE FUNCTION showFreeTablesFromXToYWithZChairs(
        @StartDate datetime,
        @EndDate datetime,
        @Chairs int
    ) RETURNS TABLE AS RETURN
SELECT
    T.TableID,
    T.ChairAmount
FROM
    TABLES T
WHERE
    T.TableID NOT IN (
        SELECT
            Q.TableID
        FROM
            show_taken_tables_from_x_to_y_with_z_chairs(@StartDate,
@EndDate, @Chairs) Q
    )
    AND T.isActive = 1
    AND ChairAmount = @Chairs
GO
```

7.7. GetBestMeal

```
CREATE FUNCTION GetBestMeal(@input int) RETURNS TABLE AS RETURN
SELECT
    DISTINCT TOP (@input) P.Name,
    MMI.times_sold
FROM
    Products P
    INNER JOIN mealMenuInfo MMI ON P.ProductID = MMI.ProductID
ORDER BY
    MMI.times_sold
GO
```

7.8. GetClientsOrderedMoreThanXTimes

```
CREATE FUNCTION GetClientsOrderedMoreThanXTimes(@amount int) RETURNS TABLE
AS RETURN
SELECT
    *
FROM
    ClientStatistics
WHERE
    [times ordered] > @amount
GO
```

7.9. GetClientsOrderedMoreThanXValue

```
CREATE FUNCTION GetClientsOrderedMoreThanXValue(@value float) RETURNS
TABLE AS RETURN
SELECT
    *
FROM
    ClientStatistics
WHERE
    [value ordered] > @value
go
```

7.10. GetClientsWhoOweMoreThanX

```
CREATE FUNCTION GetClientsWhoOweMoreThanX(@value int) RETURNS TABLE AS
RETURN
SELECT
    ClientID,
    [money to pay]
FROM
    individualClientsWhoNotPayForOrders
WHERE
    [money to pay] > @value
```

```
UNION
SELECT
    ClientID,
    [money to pay]
FROM
    companiesWhoNotPayForOrders
WHERE
    [money to pay] > @value
GO
```

7.11. caluculateBestDiscountTemporary

```
CREATE FUNCTION calculateBestDiscountTemporary(@ClientID int) RETURNS
Table
AS
    RETURN (SELECT DiscountValue, DiscountID FROM
                            (SELECT DiscountValue, DiscountID,
ROW_NUMBER() over (order by DiscountValue DESC ) as 'Row number' FROM
Discounts
                                    INNER JOIN DiscountsVar DV ON DV.VarID
= Discounts.VarID
                            WHERE ClientID = @ClientID
                                    AND DiscountType = 'Temporary'
                                    AND isUsed = 0
                                    AND AppliedDate <= getdate() AND
GETDATE() <= dateadd(DAY, ValidityPeriod, AppliedDate)
                            ) CTE
                        WHERE [Row number] = 1
                        )
GO
```

7.12. calculateBestDiscountPermanent

```
CREATE FUNCTION calculateBestDiscountPermanent(@ClientID int) RETURNS
Table
AS
    RETURN (SELECT DiscountValue, DiscountID FROM
                            (SELECT DiscountValue, DiscountID,
ROW_NUMBER() over (order by DiscountValue DESC ) as 'Row number' FROM
Discounts
                                    INNER JOIN DiscountsVar DV ON DV.VarID
= Discounts.VarID
                            WHERE ClientID = @ClientID
                                    AND DiscountType = 'Permanent'
                            ) CTE
                        WHERE [Row number] = 1
                        )
GO
```

7.13. caluclateDiscountForClient

```
CREATE FUNCTION calculateDiscountForClient(@ClientID int) RETURNS
@Discount Table(ID int, Value decimal(3,2), Type nvarchar(50))
AS
    BEGIN
        DECLARE @BestValue decimal(3, 2);
        DECLARE @DiscountID int
        DECLARE @Permanent int
        DECLARE @Temporary int

        SET @Permanent = (SELECT COUNT(*) FROM
dbo.calculateBestDiscountPermanent(@ClientID))
        SET @Temporary = (SELECT COUNT(*) FROM
dbo.calculateBestDiscountTemporary(@ClientID))
    IF (@Permanent + @Temporary) = 0
        BEGIN
            INSERT @Discount(ID, Value, Type)
            VALUES (
                    NULL,
                    NULL,
                    NULL
                   )
            RETURN
        END

    IF @Permanent = 0 AND @Temporary = 1
        BEGIN
            SELECT @BestValue = DiscountValue, @DiscountID = DiscountID
FROM dbo.calculateBestDiscountTemporary(@ClientID)
            INSERT @Discount(ID, Value, Type)
            values (
                        @DiscountID,
                        @BestValue,
                        'Temporary'
                   )
            RETURN
        END
    IF @Permanent = 1 AND @Temporary = 0
        BEGIN
            SELECT @BestValue = DiscountValue, @DiscountID = DiscountID
FROM dbo.calculateBestDiscountPermanent(@ClientID)
            INSERT @Discount(ID, Value, Type)
            values (
                        @DiscountID,
                        @BestValue,
                        'Permanent'
                   )
            RETURN
        END

    DECLARE @PermanentValue decimal(3,2)
    DECLARE @PermanentID int
    DECLARE @TemporaryValue decimal(3,2)
    DECLARE @TemporaryID int
```

```
    SELECT @TemporaryValue = DiscountValue, @TemporaryID = DiscountID FROM
dbo.calculateBestDiscountTemporary(@ClientID)
    SELECT @PermanentValue = DiscountValue, @PermanentID = DiscountID FROM
dbo.calculateBestDiscountPermanent(@ClientID)

    IF @PermanentValue > @TemporaryValue
        BEGIN
            INSERT @Discount(ID, Value, Type)
            values (
                        @PermanentID,
                        @PermanentValue,
                        'Permanent'
                    )
            RETURN
        END
        INSERT @Discount(ID, Value, Type)
        values (
                    @TemporaryID,
                    @TemporaryValue,
                    'Temporary'
                )
        RETURN

    END
GO
```

7.14. sumOfMoneySpentIn_Month_Year

```
CREATE FUNCTION sumOfMoneySpentIn_Month_Year(@WhichYear int, @WhichMonth
int) RETURNS money AS BEGIN RETURN (
        SELECT
            sum(OrderSum)
        FROM
            dbo.Orders
        WHERE
            @WhichYear = Year(OrderDate)
            AND @WhichMonth = MONTH(OrderDate)
    ) END
GO
```

7.15. GetOrdersDetails

```
CREATE FUNCTION GetOrderDetails(@InputOrderID int) RETURNS TABLE
    AS
        RETURN (
        SELECT
            O.OrderID,
            O.ClientID,
            ISNULL(cast(O.TakeawayID as varchar), 'Order not for
takeaway') as 'TakeAwayID',
            ISNULL(cast(O.ReservationID as varchar), 'Order is not for
reservation') as 'ReservationID',
            ISNULL(cast(O.InvoiceID as varchar), 'Order does not have
invoice.') as 'InvoiceID',
```

```
            PM.PaymentName,
            PS.PaymentStatusName,
            CONCAT(S.LastName, ' ', S.FirstName) as 'Employee',
            O.OrderSum,
            O.OrderDate,
            ISNULL(convert(varchar, O.OrderCompletionDate, 120), 'Order is
pending') as 'OrderCompletionDate',
            O.OrderStatus,
            P.Name,
            (SELECT MD.Price FROM MenuDetails MD INNER JOIN CurrentMenu CM
on MD.MenuID = CM.MenuID WHERE MD.ProductID = OD.ProductID) as 'Product
Price',
            OD.Quantity
        FROM Orders O
            INNER JOIN OrderDetails OD on O.OrderID = OD.OrderID
            INNER JOIN Products P ON P.ProductID = OD.ProductID
            INNER JOIN PaymentMethods PM on O.PaymentMethodID =
PM.PaymentMethodID
            INNER JOIN PaymentStatus PS on O.PaymentStatusID =
PS.PaymentStatusID
            INNER JOIN Staff S on O.staffID = S.StaffID
        WHERE
            O.OrderID = @InputOrderID
    )
GO
```

7.16. OrderProductWithin14days

```
CREATE FUNCTION OrderProductWithin14days (@InputProductName nvarchar(150))
RETURNS INT
AS
    BEGIN
     RETURN (
        SELECT
            SUM([O D].Quantity)
        FROM
            OrderDetails AS [O D]
            INNER JOIN Products P ON P.ProductID = [O D].ProductID
            INNER JOIN Orders O ON O.OrderID = [O D].OrderID
        WHERE
            P.Name LIKE @InputProductName
            AND ABS(DATEDIFF(DAY, O.OrderDate, GETDATE())) <= 14
    )
    END
GO
```

7.17. OrdersMoreExpensiveThanN

```
CREATE FUNCTION OrdersMoreExpensiveThanN (@N int) RETURNS TABLE
AS
    RETURN (
        SELECT
            O.*
        FROM
```

```
            Orders AS O
        WHERE
            O.OrderSum > @N
    )
GO
```

7.18. WhatWasNotInTheMenuOfGivenID

```
CREATE FUNCTION WhatWasNotInTheMenuOfGivenID(@MenuID int)
    RETURNS TABLE AS RETURN
        SELECT P.ProductID, P.Name, P.Description as 'Product
Description', P.IsAvailable, C.CategoryName , C.Description as 'Category
Description' FROM Products P
                INNER JOIN Category C on C.CategoryID = P.CategoryID
            WHERE P.ProductID IN
         (SELECT ProductID
          FROM Products PI
          EXCEPT
          SELECT ProductID
          FROM MenuDetails
          WHERE MenuID=@MenuID) AND P.IsAvailable = 1
GO
```

7.19. WhatWasNotInPreviousAndFollowingMenu

```
CREATE FUNCTION WhatWasNotInThePreviousAndFollowingMenu(@MenuID int)
    RETURNS TABLE AS RETURN
        SELECT P.ProductID, P.Name, P.Description as 'Product
Description', C.CategoryName , C.Description as 'Category Description'
FROM Products P
                INNER JOIN Category C on C.CategoryID = P.CategoryID
            WHERE P.ProductID IN
         (SELECT P.ProductID FROM Products PI
          EXCEPT
             (SELECT ProductID FROM MenuDetails MD
                     INNER JOIN Menu M on M.MenuID = MD.MenuID
                 WHERE MD.MenuID=dbo.GetIdOfFollowingMenu(@MenuID)
                     AND ABS(DATEDIFF(day, (SELECT TOP 1 endDate from Menu
inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.startDate)) <= 1
                 )
          EXCEPT
             (SELECT ProductID FROM MenuDetails
                 WHERE MenuID=dbo.GetIdOfPreviousMenu(@MenuID) )
          ) AND P.IsAvailable = 1
Go
```

7.20. MenuIsCorrect

```
CREATE FUNCTION MenuIsCorrect(@MenuID int) RETURNS @WhereAreDuplicated
Table(Field nvarchar(100), Field_value nvarchar(100))
AS
    BEGIN
        DECLARE @SameItemsPrevious int
        SET @SameItemsPrevious = (
            SELECT
                COUNT(*)
            FROM (
                        SELECT Name, Description FROM
dbo.ShowDuplicatesPreviousMenu (@MenuID)
                    INTERSECT
                        SELECT P.Name, P.Description FROM MenuDetails
INNER JOIN Products P on P.ProductID = MenuDetails.ProductID WHERE MenuID
= @MenuID
                ) OUT
        )

        DECLARE @SameItemsFollowing int
        SET @SameItemsFollowing = (
            SELECT
                COUNT(*)
            FROM (
                        SELECT Name, Description FROM
dbo.ShowDuplicatesFollowingMenu (@MenuID)
                    INTERSECT
                        SELECT P.Name, P.Description FROM MenuDetails
INNER JOIN Products P on P.ProductID = MenuDetails.ProductID WHERE MenuID
= @MenuID
                ) OUT
        )

        DECLARE @minAmountToChangePrevious int
        SET @minAmountToChangePrevious = (
            SELECT
                COUNT(*)
            FROM
                MenuDetails
            WHERE
                MenuID = dbo.GetIdOfPreviousMenu(@MenuID)
        ) / 2

        DECLARE @minAmountToChangeFollowing int
        SET @minAmountToChangeFollowing = (
            SELECT
                COUNT(*)
            FROM
                MenuDetails
            WHERE
                MenuID = dbo.GetIdOfFollowingMenu (@MenuID)
        ) / 2
        IF @SameItemsFollowing > @minAmountToChangeFollowing
```

```
            BEGIN
                INSERT @WhereAreDuplicated(Field, Field_value)
                VALUES (
                        'Following',
                        0
                       )
            END
        ELSE
            BEGIN
                INSERT @WhereAreDuplicated(Field, Field_value)
                VALUES (
                        'Following',
                        1
                       )
            END
        IF @SameItemsPrevious > @minAmountToChangePrevious
            BEGIN
                INSERT @WhereAreDuplicated(Field, Field_value)
                VALUES (
                        'Previous',
                        0
                       )
            END
        ELSE
            BEGIN
                INSERT @WhereAreDuplicated(Field, Field_value)
                VALUES (
                        'Previous',
                        1
                       )
            END
        RETURN
    END
go
```

7.21. GetIdFollowingMenu

```
CREATE FUNCTION GetIdOfFollowingMenu(@MenuID int)
RETURNS int
    AS
        BEGIN
            RETURN (SELECT FollowingID FROM (SELECT MI.MenuID,
LEAD(MenuID) OVER (ORDER BY startDate, endDate) as 'FollowingID' FROM Menu
MI) MO WHERE MO.MenuID = @MenuID)
        END
    GO

CREATE FUNCTION GetIdOfPreviousMenu(@MenuID int)
RETURNS int
    AS
        BEGIN
            RETURN (SELECT PreviousID FROM (SELECT MI.MenuID, LAG(MenuID)
OVER (ORDER BY startDate, endDate) as 'PreviousID' FROM Menu MI) MO WHERE
MO.MenuID = @MenuID)
```

```
        END
    GO
```

7.22. ShowDuplicatesInPreviousAndFollowingMenu

```
CREATE FUNCTION ShowDuplicatesInPreviousAndFollowingMenu(@MenuID int)
RETURNS table
AS
    RETURN SELECT P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
            WHERE MenuID = @MenuID
            INTERSECT
            (SELECT P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
                INNER JOIN Menu M on M.MenuID = MD.MenuID
            WHERE MD.MenuID = dbo.GetIdOfPreviousMenu(@MenuID)
                AND ABS(DATEDIFF(day, (SELECT TOP 1 Menu.startDate from
Menu inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.endDate)) <= 1
            UNION
            SELECT P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
                INNER JOIN Menu M on M.MenuID = MD.MenuID
            WHERE MD.MenuID = dbo.GetIdOfFollowingMenu(@MenuID)
                    AND ABS(DATEDIFF(day, (SELECT TOP 1 endDate from Menu
inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.startDate)) <= 1)
go
```

7.23. ShowDuplicatesInPreviousAndFollowingMenuWithID

```
CREATE FUNCTION ShowDuplicatesInPreviousAndFollowingMenuWithID(@MenuID
int)
RETURNS table
    AS
        RETURN  SELECT P.ProductID ,P.Name, P.Description FROM MenuDetails
MD
                INNER JOIN Products P ON P.ProductID = MD.ProductID
                INNER JOIN Menu M on M.MenuID = MD.MenuID
                WHERE
                    P.Name IN (SELECT  MI.Name FROM
dbo.ShowDuplicatesInPreviousAndFollowingMenu(@MenuID) MI)
                    AND MD.MenuID = dbo.GetIdOfPreviousMenu(@MenuID)
                    AND ABS(DATEDIFF(day, (SELECT TOP 1 Menu.startDate
from Menu inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE
D.MenuID = @MenuID), M.endDate)) <= 1
              UNION
              SELECT P.ProductID, P.Name, P.Description FROM MenuDetails
MD
                    INNER JOIN Products P ON P.ProductID = MD.ProductID
                    INNER JOIN Menu M on M.MenuID = MD.MenuID
```

```
                WHERE
                     P.Name IN (SELECT  MI.Name FROM
dbo.ShowDuplicatesInPreviousAndFollowingMenu(@MenuID) MI)
                     AND MD.MenuID = dbo.GetIdOfFollowingMenu(@MenuID)
                     AND ABS(DATEDIFF(day, (SELECT TOP 1 endDate from Menu
inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.startDate)) <= 1
go
```

7.24. ShowDupliactesFollowingMenu

```
CREATE FUNCTION ShowDuplicatesFollowingMenu(@MenuID int)
RETURNS table
AS
    RETURN SELECT P.ProductID, P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
            WHERE MenuID = @MenuID
    INTERSECT
        (SELECT P.ProductID,P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
                INNER JOIN Menu M on M.MenuID = MD.MenuID
            WHERE MD.MenuID = dbo.GetIdOfFollowingMenu(@MenuID)
                     AND ABS(DATEDIFF(day, (SELECT TOP 1 endDate from Menu
inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.startDate)) <= 1)
go
```

7.25. ShowDuplicatesPreviousMenu

```
CREATE FUNCTION ShowDuplicatesPreviousMenu(@MenuID int)
RETURNS table
AS
    RETURN  SELECT P.ProductID, P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
            WHERE MenuID = @MenuID
    INTERSECT
        (SELECT P.ProductID,P.Name, P.Description FROM MenuDetails MD
                INNER JOIN Products P on P.ProductID = MD.ProductID
                INNER JOIN Menu M on M.MenuID = MD.MenuID
            WHERE MD.MenuID = dbo.GetIdOfPreviousMenu(@MenuID)
                AND ABS(DATEDIFF(day, (SELECT TOP 1 Menu.startDate from
Menu inner join MenuDetails D on Menu.MenuID = D.MenuID WHERE D.MenuID =
@MenuID), M.endDate)) <= 1)
go
```

7.26. GenerateIndividualClientReport

Funkcja generująca raport o danym kliencie indywidualnym na przestrzeni zadanego czasu.

7.27. GenerateCompanyReport

Funkcja generująca raport o danej firmie na przestrzeni zadanego czasu.

7.28. GenerateTableWeeklyReport

Funkcja generująca tygodniowy raport o stolikach na przestrzeni zadanego czasu.

7.29. GenerateTableMonthlyReport

Funkcja generująca miesięczny raport o stolikach na przestrzeni zadanego czasu.

7.30. GenerateReservationReport

```
CREATE FUNCTION GenerateReservationReport(@From Date, @To Date)
RETURNS Table
AS
    RETURN (
        SELECT * FROM ReservationSummary WHERE startDate BETWEEN @From AND
@To
    )
GO
```

7.31. GenerateReservationMonthlyReport

Funkcja generująca miesięczny raport o rezerwacjach na przestrzeni zadanego czasu.

7.32. GenerateReservationWeeklyReport

Funkcja generująca tygodniowy raport o rezerwacjach na przestrzeni zadanego czasu.

7.33. GenerateMenuReport

Funkcja generująca raport o zadanym menu.

7.34. GenerateOrderReport

Funkcja generująca raport o zadanym zamówieniu.

7.35. GenerateDiscountsSummaryForClient

```
CREATE FUNCTION GenerateDiscountsSummaryForClient(@ClientID int)
RETURNS TABLE
    AS
        RETURN (
                SELECT * FROM DiscountsSummary WHERE ClientID = @ClientID
    )
GO
```

7.36. GetInvoice

Funkcja generująca fakturę do danego zamówienia.

7.37. GetClientInvoices

```
CREATE FUNCTION GetClientInvoices(@ClientID int)
RETURNS TABLE
AS
    RETURN (SELECT * FROM Invoice WHERE ClientID = @ClientID)
GO
```

8. Triggery

8.1. SeaFoodCheckMonday

blokuje zamówienia, które ze względu na znajdujące się w nim owoce morza, powinny być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.

```
CREATE TRIGGER SeaFoodCheckMonday
    ON OrderDetails
AFTER INSERT
AS BEGIN
   SET NOCOUNT ON
    DECLARE @CategoryID int
    SELECT @CategoryID = CategoryID FROM Category WHERE
LOWER(CategoryName) LIKE 'sea food'
    IF EXISTS(
        SELECT * FROM inserted AS I
            INNER JOIN Orders AS O ON O.OrderID = I.OrderID
            INNER JOIN dbo.OrderDetails OD ON O.OrderID = OD.OrderID
            INNER JOIN Products P ON OD.ProductID = P.ProductID
        WHERE
            (   DATENAME(WEEKDAY, O.OrderCompletionDate) LIKE 'Thursday'
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, O.OrderCompletionDate) LIKE 'Friday'
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, O.OrderCompletionDate) LIKE 'Saturday'
                AND CategoryID = @CategoryID
            )
            OR
            (   DATENAME(WEEKDAY, O.OrderDate) LIKE 'Thursday'
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, O.OrderDate) LIKE 'Friday'
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, O.OrderDate) LIKE 'Saturday'
                AND CategoryID = @CategoryID
            )
        )
        OR
        EXISTS(
        SELECT * FROM inserted AS I
            INNER JOIN Orders AS O ON O.OrderID = I.OrderID
            INNER JOIN dbo.OrderDetails OD ON O.OrderID = OD.OrderID
            INNER JOIN Products P ON OD.ProductID = P.ProductID
            INNER JOIN Reservation R2 ON O.ReservationID =
```

```
R2.ReservationID
        WHERE
            (   DATENAME(WEEKDAY, R2.startDate) LIKE 'Thursday'
                AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 2
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, R2.startDate) LIKE 'Friday'
                AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 3
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, R2.startDate) LIKE 'Saturday'
                AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 4
                AND CategoryID = @CategoryID
            )
        ) OR EXISTS(
            SELECT * FROM inserted AS I
            INNER JOIN Orders AS O ON O.OrderID = I.OrderID
            INNER JOIN dbo.OrderDetails OD ON O.OrderID = OD.OrderID
            INNER JOIN Products P ON OD.ProductID = P.ProductID
            INNER JOIN OrdersTakeaways OT ON O.TakeawayID = OT.TakeawaysID
        WHERE
            (   DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Thursday'
                AND DATEDIFF(DAY, O.OrderDate, OT.PrefDate) <= 2
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Friday'
                AND DATEDIFF(DAY, O.OrderDate, OT.PrefDate) <= 3
                AND CategoryID = @CategoryID
            )
            OR
            (
                DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Saturday'
                AND DATEDIFF(DAY, O.OrderDate, OT.PrefDate) <= 4
                AND CategoryID = @CategoryID
            )
        )
        BEGIN;
            THROW 50001, N'Takie zamówienie winno być złożone maksylamnie
do poniedziałku poprzedzającego zamówienie.', 1
        END
    END
Go
```

8.2. DeleteOrderDetails

Usuwa szczegóły zamówienia z tabeli OrderDetails, jeżeli powiązana z nim rezerwacja została anulowana przez klienta

```
CREATE TRIGGER DeleteOrderDetails
ON OrderDetails
FOR DELETE
AS
BEGIN
    SET NOCOUNT ON
    DELETE FROM OrderDetails WHERE OrderID IN (
        SELECT O.OrderID FROM Orders O
            INNER JOIN Reservation R2 ON R2.ReservationID =
O.ReservationID
        WHERE LOWER(R2.Status) LIKE 'cancelled' OR LOWER(R2.Status) LIKE
'denied'
    )
    DELETE FROM OrderDetails WHERE OrderID IN (
        SELECT O.OrderID FROM Orders O
        WHERE LOWER(O.OrderStatus) LIKE 'cancelled' OR
LOWER(O.OrderStatus) LIKE 'denied'
    )
END
go
```

8.3. OrderDetailsInsert

Sprawdza czy danie które próbujemy dodać do zamówienia jest zaznaczone jako dostępne.

```
CREATE TRIGGER OrderDetailsInsert
ON OrderDetails
FOR INSERT
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @ProductID int
    DECLARE @OrderID int
    DECLARE @MenuID int
    DECLARE @ProductName nvarchar(200)
    SELECT @MenuID = MAX(MenuID) from Menu

    SELECT @ProductID = ProductID from inserted
    SELECT @OrderID = OrderID from inserted
    SET @ProductName = (SELECT Name FROM Products where Products.ProductID
= @ProductID)
    IF EXISTS(SELECT * FROM Products P WHERE P.ProductID = @ProductID AND
P.IsAvailable = 0)
        BEGIN;
            THROW 50001, 'Niepoprawne ProductID, Jego IsAvailable to 0 w
tabeli Products. ', 1
            ROLLBACK TRANSACTION
        END
    IF NOT EXISTS(SELECT * FROM CurrentMenu where Name like @ProductName)
        BEGIN
            THROW 50001, 'Ten produkt nieznajduje się aktualnie w menu.',
1
            ROLLBACK TRANSACTION
        END
END
```

```
Go
```

8.4. EmployeeInsert

Sprawdzanie, czy pracodawca dodanego pracownika jest firmą.

```
CREATE TRIGGER EmployeeInsert
ON Employees
FOR INSERT
AS
    BEGIN
        DECLARE @ClientID int
        SELECT @ClientID = CompanyID from inserted
        IF NOT EXISTS(SELECT * FROM Companies C where C.ClientID =
@ClientID)
            BEGIN;
                THROW 50001, N'Klient o podanym ID nie jest firmą. Nie
można dodać pracownika!', 1
                ROLLBACK TRANSACTION
            END
    END
GO
```

8.5. addTableToReservationInsertCheck

Sprawdzanie, czy stolik można dodać do rezerwacji.

```
CREATE TRIGGER addTableToReservationInsertCheck
ON ReservationDetails
FOR INSERT
AS
    BEGIN
        DECLARE @ReservationID int = (SELECT ReservationID FROM inserted)
        DECLARE @TableID int = (SELECT TableID FROM inserted)

        DECLARE @StartReservationDate datetime
        DECLARE @EndReservationDate datetime

        SELECT @StartReservationDate = StartDate, @EndReservationDate =
EndDate FROM Reservation R WHERE R.ReservationID = @ReservationID

        DECLARE @TableInUseCountCompany int
        DECLARE @TableInUseCountIndividuals int

        SELECT @TableInUseCountCompany = COUNT(TableID) from Reservation R
            INNER JOIN ReservationCompany RC on R.ReservationID =
RC.ReservationID
            INNER JOIN ReservationDetails RD on RC.ReservationID =
RD.ReservationID
        WHERE (R.startDate >= @StartReservationDate AND R.endDate <=
@EndReservationDate)

        SELECT @TableInUseCountIndividuals = COUNT(TableID) from
```

```
Reservation R
            INNER JOIN ReservationIndividual RI on R.ReservationID =
RI.ReservationID
            INNER JOIN ReservationDetails RD on RI.ReservationID =
RD.ReservationID
        WHERE (R.startDate >= @StartReservationDate AND R.endDate <=
@EndReservationDate)

        IF @TableInUseCountIndividuals > 0 OR @TableInUseCountCompany > 0
        BEGIN;
            THROW 50200, N'Dany stolik jest używany przez inną rezerwację!
', 1
            ROLLBACK TRANSACTION;
        END

        IF EXISTS(SELECT * FROM Tables T WHERE T.TableID = @TableID AND
T.isActive = 0)
            BEGIN;
                THROW 50200, N'Stolik nie jest w użyciu (isActive jest
0)', 1
                ROLLBACK TRANSACTION;
            END
    END
GO
```

8.6. TablesOnDelete

```
CREATE TRIGGER TablesOnDelete
ON ReservationDetails
FOR DELETE, UPDATE
AS
    BEGIN
        SET NOCOUNT ON
        DECLARE @TableInUseCountCompany int
        DECLARE @TableInUseCountIndividuals int

        SELECT @TableInUseCountCompany = COUNT(*) FROM deleted D
            INNER JOIN ReservationCompany RC ON RC.ReservationID =
D.ReservationID
            INNER JOIN Reservation R2 on RC.ReservationID =
R2.ReservationID
        WHERE R2.startDate >= GETDATE()

        SELECT @TableInUseCountIndividuals = COUNT(*) FROM deleted D
            INNER JOIN ReservationIndividual RC ON RC.ReservationID =
D.ReservationID
            INNER JOIN Reservation R2 on RC.ReservationID =
R2.ReservationID
        WHERE R2.startDate >= GETDATE()

        IF @TableInUseCountCompany > 0 OR @TableInUseCountIndividuals > 0
        BEGIN;
            THROW 52000, N'Stolik nie może zostać usunięty lub zmieniony
jego status aktywności jeśli jest zarezerwowany', 1
```

```
            ROLLBACK TRANSACTION;
        END
    END
GO
```

8.7. Z1TestForNewDiscountVariable

Sprawdzanie, czy dodana nowa zmienna zniżki Z1 jest prawidłowa.

```
CREATE TRIGGER Z1TestForNewDiscountVariable
    ON DiscountsVar
    FOR INSERT ,UPDATE
    AS
    BEGIN
        SET NOCOUNT ON
        IF EXISTS(
            SELECT * FROM inserted AS I
            WHERE I.MinimalOrders<=0
            )
        BEGIN
            THROW 52000, N'Nowa zniżka powinna miec dodatni parametr
MinimalOrders', 1
        END
    END
GO
```

8.8. NewMenuIsCorrect

```
CREATE TRIGGER NewMenuIsCorrect
ON MenuDetails
FOR INSERT
AS
    BEGIN
        DECLARE @MenuID int = (SELECT MenuID FROM inserted)
        DECLARE @PreviousCorrect int = (SELECT Field_value FROM
dbo.MenuIsCorrect(@MenuID) WHERE LOWER(Field) LIKE 'previous')
        DECLARE @FollowingCorrect int = (SELECT Field_value FROM
dbo.MenuIsCorrect(@MenuID) WHERE LOWER(Field) LIKE 'following')

        IF(@PreviousCorrect = 0)
            BEGIN
                DECLARE @PreviousMenuItemsCount int

                SET @PreviousMenuItemsCount = (SELECT count(*) FROM
ShowDuplicatesPreviousMenu (@MenuID))

                IF @PreviousMenuItemsCount > 0
                    BEGIN
                        SELECT ProductID, Name, Description FROM
ShowDuplicatesPreviousMenu (@MenuID) ORDER BY ProductID;
                    END;
```

```
                THROW 50001, N'Zmieniono za małą liczbę dań w aktualnym
menu względem wcześniejszego menu!',1
                ROLLBACK TRANSACTION
            END

        IF(@FollowingCorrect = 0)
            BEGIN
                DECLARE @FollowingMenuItemsCount int

                SET @FollowingMenuItemsCount = (SELECT count(*) FROM
ShowDuplicatesFollowingMenu(@MenuID))

                IF @FollowingMenuItemsCount > 0
                    BEGIN
                        SELECT ProductID, Name, Description FROM
ShowDuplicatesFollowingMenu(@MenuID) ORDER BY ProductID;
                    END;

                THROW 50001, N'Zmieniono za małą liczbę dań w aktualnym
menu względem przyszłego menu!',1
                ROLLBACK TRANSACTION
            END
    END
go
```

8.9. CanReservation

```
CREATE TRIGGER CanReservation
    ON Orders
    AFTER UPDATE
AS
    SET NOCOUNT ON
    BEGIN
        DECLARE @LastOrderID int = (SELECT OrderID FROM inserted );
        DECLARE @ClientID int = (SELECT ClientID FROM inserted )
        DECLARE @ReservationID int;

        SELECT @ReservationID = R2.ReservationID FROM Orders
            INNER JOIN Reservation R2 on Orders.ReservationID =
R2.ReservationID
        WHERE OrderID = @LastOrderID


        IF @ReservationID IS NOT NULL
        BEGIN;
            DECLARE @MinimalOrders int
            DECLARE @MinimalValue money
            DECLARE @CategoryID int

            SELECT @CategoryID = CategoryID FROM Category WHERE
LOWER(CategoryName) LIKE 'sea food'
            SELECT @MinimalOrders = [Minimal number of orders],
@MinimalValue = [Minimal value for orders] FROM CurrentReservationVars
```

```
            IF EXISTS(
                SELECT * FROM  Orders AS O
                    INNER JOIN dbo.OrderDetails OD ON O.OrderID =
OD.OrderID
                    INNER JOIN Products P ON OD.ProductID = P.ProductID
                    INNER JOIN Reservation R2 ON O.ReservationID =
R2.ReservationID
                WHERE
                    (   DATENAME(WEEKDAY, R2.startDate) LIKE 'Thursday'
                        AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 2
                        AND CategoryID = @CategoryID AND O.ClientID =
@ClientID AND O.OrderID = @LastOrderID
                    )
                    OR
                    (
                        DATENAME(WEEKDAY, R2.startDate) LIKE 'Friday'
                        AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 3
                        AND CategoryID = @CategoryID AND O.ClientID =
@ClientID AND O.OrderID = @LastOrderID
                    )
                    OR
                    (
                        DATENAME(WEEKDAY, R2.startDate) LIKE 'Saturday'
                        AND DATEDIFF(DAY, O.OrderDate, R2.startDate) <= 4
                        AND CategoryID = @CategoryID AND O.ClientID =
@ClientID AND O.OrderID = @LastOrderID
                    )
            )
                BEGIN
                    THROW 52000, N'Należy odrzucić dane zamówienie i
rezerwację! Klient nie może zamówić w tym dniu owoców morza!', 1
                END


            IF NOT EXISTS(SELECT * FROM
dbo.GetClientsOrderedMoreThanXTimes(@MinimalOrders) WHERE ClientID =
@ClientID)
                BEGIN
                    DECLARE @msg1 nvarchar(2048) = N'Należy odrzucić dane
zamówienie i rezerwację! Klient nie spełnia minimalnej liczby zamówień
wynoszącej: ' + CAST(@MinimalOrders AS nvarchar);
                    THROW 52000, @msg1, 1
                    ROLLBACK TRANSACTION
                END

            IF (SELECT OrderSum FROM inserted WHERE OrderID = @LastOrderID
AND ClientID = @ClientID ) <= @MinimalValue
                BEGIN
                    DECLARE @msg2 nvarchar(2048) = N'Należy odrzucić dane
zamówienie i rezerwację! Klient nie spełnia minimalnej wartości zamówienia
wynoszącej: ' + CAST(@MinimalValue AS nvarchar);
                    THROW 52000, @msg2, 1
                    ROLLBACK TRANSACTION
                END
        END
    END
go
```

8.10. uniqueValuesInCompanies

```
CREATE TRIGGER uniqueValuesInCompanies
ON Companies
FOR INSERT, UPDATE
AS
    BEGIN
        SET NOCOUNT ON
        IF EXISTS(SELECT * FROM inserted I WHERE I.CompanyName IN (SELECT
CompanyName FROM Companies WHERE ClientID <> I.ClientID))
        BEGIN
            THROW 52000, N'Nazwa firmy musi być unikalna!', 1
        END
        DECLARE @KRS varchar = (SELECT KRS FROM inserted)
        IF @KRS IS NOT NULL AND EXISTS(SELECT * FROM inserted I WHERE
I.KRS IN (SELECT KRS FROM Companies WHERE ClientID <> I.ClientID))
        BEGIN
            THROW 52000, N'KRS musi być unikalny!', 1
        END
        DECLARE @Regon varchar = (SELECT Regon FROM inserted)
        IF @Regon IS NOT NULL AND EXISTS(SELECT * FROM inserted I WHERE
I.Regon IN (SELECT Regon FROM Companies WHERE ClientID <> I.ClientID))
        BEGIN
            THROW 52000, N'Regon musi być unikalny!', 1
        END
    END
GO
```

8.11. UpdateUserDiscounts

```
CREATE TRIGGER UpdateUserDiscounts
ON OrderDetails
AFTER INSERT
AS
    BEGIN
        SET NOCOUNT  ON

        DECLARE @ClientID INT
        DECLARE @MinimalOrders INT
        DECLARE @MinimalAggregateValueTemporary MONEY
        DECLARE @MinimalAggregateValuePermanent MONEY
        SET @ClientID = (SELECT O.ClientID FROM inserted INNER JOIN Orders
O ON O.OrderID = inserted.OrderID)

        SELECT @MinimalAggregateValueTemporary = MinimalAggregateValue
FROM DiscountsVar WHERE LOWER(DiscountType) LIKE 'temporary' AND
(startDate <= GETDATE() AND (endDate IS NULL OR endDate >= GETDATE()))
        SELECT @MinimalAggregateValuePermanent=MinimalAggregateValue,
@MinimalOrders = MinimalOrders FROM DiscountsVar WHERE LOWER(DiscountType)
LIKE 'permanent' AND (startDate <= GETDATE() AND (endDate IS NULL OR
endDate >= GETDATE()))
```

```
        DECLARE @ClientCountOrders int

        SET @ClientCountOrders = (SELECT COUNT(*) FROM
OrdersMoreExpensiveThanN(@MinimalAggregateValuePermanent) WHERE ClientID =
@ClientID)

        IF @ClientCountOrders >= @MinimalOrders
            BEGIN
--                Add permanent discount
                EXEC addDiscount @ClientID, 'Permanent'
            END
        IF EXISTS(SELECT * FROM
GetClientsOrderedMoreThanXValue(@MinimalAggregateValueTemporary) WHERE
ClientID = @ClientID)
            BEGIN
--                  Add Temporary discount
                IF NOT EXISTS(
                            SELECT * FROM Discounts
                                INNER JOIN DiscountsVar DV on DV.VarID =
Discounts.VarID
                            WHERE ClientID = @ClientID AND
LOWER(DiscountType) LIKE 'temporary' AND DATEADD(DAY, ValidityPeriod,
AppliedDate) >= GETDATE() AND isUsed = 0
                        )
                BEGIN
                    EXEC addDiscount @ClientID, 'Temporary'
                END
            END
    END
go
```

9.Indeksy

9.1. Index_ClientID

```
CREATE INDEX Index_ClientID
    on Clients (ClientID)
```

9.2. Index_Clients__Phone

```
CREATE INDEX Index_Clients_Phone
    on Clients (Phone)
```

9.3. Index_Clients_Email

```
CREATE INDEX Index_Clients_Email
    on Clients (Email)
```

9.4. Index_Staff_Email

```
CREATE INDEX Index_Staff_Email
    on Staff (Email)
```

9.5. Index_Staff_Phone

```
CREATE INDEX Index_Staff_Phone
    on Staff (Phone)
```

9.6. Index_PersonID

```
CREATE INDEX Index_PersonID
    on Person (PersonID)
```

9.7. Index_PaymentName

```
CREATE INDEX Index_PaymentName
    on PaymentMethods (PaymentName)
```

9.8. Index_PaymentStatusID

```
CREATE INDEX Index_PaymentStatusID
    on PaymentStatus (PaymentStatusID)
```

9.9. Index_InvoiceID

```
CREATE INDEX Index_InvoiceID
    on Invoice (InvoiceID)
```

9.10. Index_InvoiceNumber

```
CREATE INDEX Index_InvoiceNumber
    on Invoice (InvoiceNumber)
```

9.11. Index_MenuID

```
CREATE INDEX Index_MenuID
    on Menu (MenuID)
```

9.12. Index_Price

```
CREATE INDEX Index_Price
    on MenuDetails (Price)
```

9.13. Index_CategoryID

```
CREATE INDEX Index_CategoryID
    on Category (CategoryID)
```

9.14. Index_ProductID

```
CREATE INDEX Index_ProductID
    on Products (ProductID)
```

9.15. Index_Name

```
CREATE INDEX Index_Name
    on Products (Name)
```

9.16. Index_DiscountID

```
CREATE INDEX Index_DiscountID
    on Discounts (DiscountID)
```

9.17. Index_ReservationID

```
CREATE INDEX Index_ReservationID
    on Reservation (ReservationID)
```

9.18. Index_TableID

```
CREATE INDEX Index_TableID
    on Tables (TableID)
```

9.19. Index_AddressID

```
CREATE INDEX Index_AddressID
    on Address (AddressID)
```

9.20. Index_CityID

```
CREATE INDEX Index_CityID
    on Cities (CityID)
```

9.21. Index_DiscountsVar_Information

```
CREATE INDEX Index_DiscountsVar_Information
    on DiscountsVar (discounttype, minimalorders, minimalaggregatevalue,
validityperiod, discountvalue, startdate, enddate)
```

10. Role

10.1 Manager restauracji

- dodawanie stolików,
- wycofywanie stolików,
- dodawanie potraw,
- wycofywanie potraw
- usuwanie potraw
- dodawanie menu
- dodawanie produktów do menu
- usuwanie produktów z menu
- wgląd do faktur
- dodawanie nowych zmiennych dotyczących zniżek
- dodawanie zniżki dla klienta
- sprawdzanie i generowanie raportów
- usuwanie adresu
- usuwanie miasta
- dodawanie kategorii
- usuwanie kategorii
- usuwanie pracownika z firmy
- dodawanie metod płatności
- usuwanie metod płatności
- dodawanie statusu płatności
- usuwanie statusu płatności
- zmiana pracownika odpowiedzialnego za zamówienie
- zmiana pracownika odpowiedzialnego za rezerwację
- sprawdzanie duplikatów w poprzednim i/lub przyszłym menu
- sprawdzanie zarobków w danym roku danego miesiąca
- sprawdzanie minimalnej, średniej i maksymalnej ceny menu
- sprawdzanie najlepiej sprzedającego się dania

10.2 Pracownik restauracji

- dodawanie klientów,
- dodawanie zamówień,
- dodawanie produktu do zamówienia
- zmiana statusu rezerwacji
- dodawanie rezerwacji
- usuwanie rezerwacji
- dodawanie stolika do rezerwacji
- usuwanie stolika z rezerwacji
- dodawanie rezerwacji do zamówienia
- zmiana statusu zamówienia i rezerwacji
- potwierdzanie i anulowanie zamówień
- dodawanie nowego adresu
- dodawanie nowego miasta
- dodawanie klienta
- dodawanie pracownika do firmy
- tworzenie faktur do zamówienia

- sprawdzanie zajętych i/lub wolnych stolików w danym okresie czasu i z daną liczbą miejsc

10.3 Klient

- składanie zamówień,
- dokonywanie rezerwacji,
- anulowanie rezerwacji