Projekt Bazy Danych Restauracji

Błażej Nowicki, Krzysztof Gwiazda, Wojciech Dróżdż

Spis Treści

Role:

- Administrator
- Klient Indywidualny
- Klient firma
    - rezerwacja na firmę
    - rezerwacja na konkretnego pracownika firmy
- Pracownik Restauracji

Administrator:

- Posiada dostęp do danych użytkowników
- Może tworzyć, edytować konta klientów indywidualnych, firm oraz pracowników restauracji
- Może tworzyć, edytować oraz usuwać rezerwacje

Klient Indywidualny

- Podaje dane osobowe podczas rejestracji
- Po rejestracji ma możliwość zmiany niektórych danych osobowych (np. Adres)
- Posiada dostęp tylko do własnych danych
- Może tworzyć rezerwacje z opcją płatności przed lub po zamówieniu
    - Podczas tworzenia rezerwacji może odczytać informacje o dostępności stolików
- Odczyt aktualnego menu
- Może przeglądać informacje na temat własnych rezerwacji
- Może złożyć zamówienie na miejscu
- Może złożyć zamówienie na wynos na miejscu
- Może złożyć zamówienie na wynos w wykorzystaniem formularza WWW
- Może anulować rezerwację

Klient Firma:

- Może dokonać rejestracji konta firmy
- Po rejestracji może zmieniać niektóre dane firmy
- Może dokonać rezerwacji na firmę
- Może dokonać rezerwacji na pracownika firmy
- Może odczytywać aktualne menu
- Może przeglądać informacje na temat własnych rezerwacji
- Zamówienie dużych ilości posiłków jako catering. Klient odbiera takie zamówienie samodzielnie w porze lunchu
- Generowanie raportu dotyczącego kwot oraz czasu składania zamówień
- Może anulować rezerwację

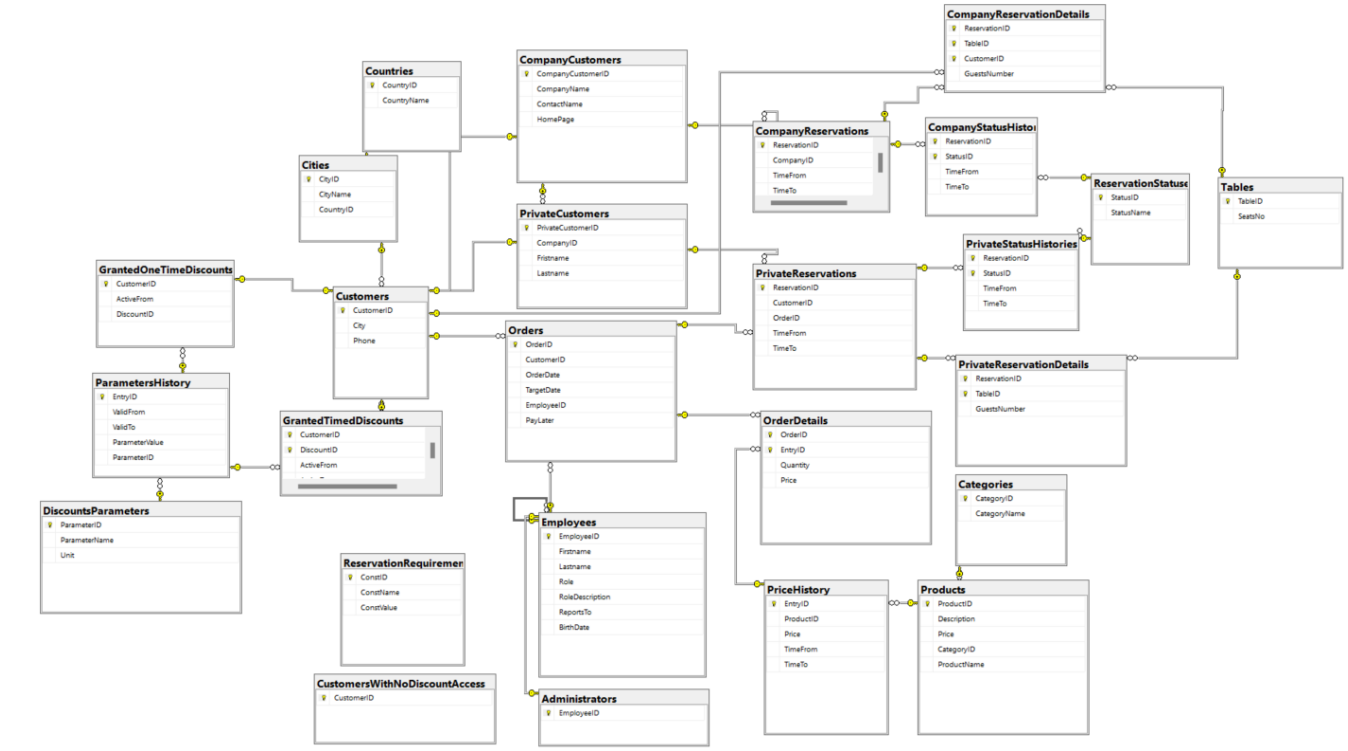
Pracownik Restauracji

- Przyjmuje zamówienia klientów na miejscu
- Akceptacja płatności klientów
- Akceptacja informacji o rezerwacji zamówienia/stolika
- Generowanie raportów odnośnie rezerwacji, rabatów, menu, statystyk zamówienia

Opis funkcji systemu:

- dodawanie zamówienia do bazy danych na miejscu (przez pracownika)
- Formularz WWW umożliwiający zlecenie zamówienia przez klienta
- Możliwość rezerwacji wolnego stolika
    - Dla klienta Indywidualnego:
        - Stolik musi być zarezerwowany dla przynajmniej dwóch osób
        - Przy rezerwacji stolika należy złożyć zamówienie
        - Istnieje określona minimalna wartość zamówienia *WZ*
        - Aby móc złożyć rezerwację klient musiał wcześniej dokonać przynajmniej *WK* zamówień.
        - Rezerwacja stolika musi zostać zatwierdzona przez obsługę.
    - Dla firmy:
        - Rezerwacja stolików może zostać stworzona na firmę
        - Rezerwacja stolików dla konkretnych pracowników firmy
- Ustalanie Menu
    - Menu musi być ustalone z dziennym wyprzedzeniem
    - Co najmniej połowa porcji jest zmieniana raz na dwa tygodnie
    - Owoce morza można wcześniej zamawiać w czwartki, piątki i soboty.
    - Wszystkie decyzje dotyczące menu podejmuje manager
- System posiada funkcję rabatów dla klientów indywidualnych
    - Po realizacji *Z1* zamówień za co najmniej *K1* zł pojawia się *R1*% zniżki na wszystkie zamówienia
    - Po realizacji zamówienia za łączną kwotę *K2* zostaje przyznana jednorazowa zniżka *R2*% na wszystkie zamówienia złożone przez *D1* dni od dnia przyznania zniżki
    - Zniżki są przyznawane automatycznie przez system
    - Wymogi oraz kwoty zniżek ustala manager
- Generowanie raportów miesięcznych i tygodniowych
    - Raporty dotyczące rezerwacji stolików
    - Raporty dotyczące rabatów
    - Raporty dotyczące menu
    - Dla klientów indywidualnych oraz firm raporty dotyczące kwot oraz czasu składania zamówień
    - Raporty może generować tylko manager, administrator, właścicie

Schemat Bazy Danych:



Tabele:

Jeśli nie napisano inaczej watrość ma ustawiony parametr not null

OneTimeDiscountsParameters

Zawiera ona parametry opisujące ograniczenia i wartości zniżek dożywotnich.

- ParameterID - int - PK - ID parametru
- ParameterName - varchar - Zawiera nazwę parametru
- Unit - varchar - nullable - Zawiera jednostkę parametru, jeśli null to jest traktowany jako zwykła liczba

Warunki integralnościowe:

- Nazwa parametru musi być unikalna:
  UNIQUE (ParameterName)

```
CREATE TABLE [dbo].[OneTimeDiscountsParameters](
	[ParameterID] [int] NOT NULL,
	[ParameterName] [varchar](50) NOT NULL,
	[Unit] [varchar](50) NULL,
 CONSTRAINT [PK_OneTimeDiscounts] PRIMARY KEY CLUSTERED
(
	[ParameterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

OneTimeDiscountsHistory

Zawiera informacje o wartościach parametrów zniżek dożywotnich w określonych odstępach czasu

- OneTimeDiscountID - PK - int - ID zmiany parametru
- ValidFrom - date - DEFAULT getdate() - data od której obowiązuje parametr
- ValidTo - date - nullable - data, do której obowiązuje parametr; jeśli null lub > getdate() to nadal obowiązuje.
- ParameterValue - int - wartość parametru
- ParameterID - FK→OneTimeDiscountsParameters.ParameterID - int - ID parametru, którego dotyczy historia

Warunki integralnościowe:

- początkowa data obowiązywania zniżki musi być późniejsza niż 01.01.1900. Data rozpoczęcia obowiązywania musi być wcześniejsza niż zakończenia;
  CHECK ([ValidFrom] > '01-01-1900' AND [ValidTo] > [ValidFrom])

- data ważności zniżki musi być późniejsza niż aktualna data;
  CHECK ([ValidTo]>getdate())
- Wartość każdego z parametrów powinna być większa bądź równa od 0;
  CHECK([ParameterValue] >= 0)

```
CREATE TABLE [dbo].[OneTimeDiscountsHistory](
	[OneTimeDiscountID] [int] NOT NULL,
	[ValidFrom] [date] DEFAULT getdate() NOT NULL,
	[ValidTo] [date] NULL,
	[ParameterValue] [int] NOT NULL,
	[ParameterID] [int] NOT NULL,
 CONSTRAINT [PK_OneTimeDiscountsHistory] PRIMARY KEY CLUSTERED
(
	[OneTimeDiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

Categories

Słownik kategorii.

- CategoryID - PK - int - ID kategorii
- CategoryName - nvarchar(50) - nazwa kategorii

Warunki integralnościowe :

- Nazwa kategorii musi być unikalna;
  UNIQUE (CategoryName)

```
CREATE TABLE [dbo].[Categories](
	[CategoryID] [int] NOT NULL,
	[CategoryName] [nvarchar](50) NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
	[CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Products

Zawiera informacje o daniach i produktach dostępnych w restauracji (niekoniecznie w aktualnym menu)

- ProductID - PK - int - ID produktu/ dania
- CategoryID - FK →Categories.CategoryID - int - ID kategorii
- Description - nvarchar(50) - opis produktu/ dania
- Price - money -DEFAULT 0 - aktualna cena produktu/dania

Warunki integralnościowe:

- Cena musi być nieujemna;
  CHECK ([Price]>=(0))

```
CREATE TABLE [dbo].[Products](
	[ProductID] [int] NOT NULL,
	[Description] [varchar](1000) NULL,
	[Price] [money] DEFAULT 0 NOT NULL,
	[CategoryID] [int] NOT NULL,
 CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
	[ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

PriceHistory

Zawiera informacje o tym jakie pozycje są w aktualnym menu oraz historię pozycji, które były wcześniej.

- EntryID - PK - int - ID pozycji w menu
- ProductID - FK → Products.ProductID - int - ID produktu
- Price - int - DEFAULT 0 - cena pozycji w danym okresie występowania w menu
- TimeFrom - date - DEFAULT getdate() - data wstawienia pozycji do menu
- TimeTo - date - data końcowa występowania pozycji w menu

Warunki integralnościowe:

- Cena musi być nieujemna;
  CHECK ([Price]>=(0))
- data wstawienia pozycji do menu musi być późniejsza niż aktualna data;
  CHECK ([TimeFrom] >= getdate())
- data końcowa musi być późniejsza niż data początkowa;
  CHECK([TimeTo]>[TimeFrom])

```
CREATE TABLE [dbo].[PriceHistory](
	[EntryID] [int] NOT NULL,
	[ProductID] [int] NOT NULL,
	[Price] [money] DEFAULT 0 NOT NULL,
	[TimeFrom] [date] DEFAULT getdate() NOT NULL,
	[TimeTo] [date] NULL,
 CONSTRAINT [PK_PriceHistory] PRIMARY KEY CLUSTERED
(
	[EntryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
```

```
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

OrderDetails

Zawiera informacje o szczegółach zamówienia.

- OrderID - PK → Orders.OrderID - int - ID zamówienia
- EntryID - PK → PriceHistory.EntryID - int - ID pozycji w menu
- Quantity - int - DEFAULT 1 - ilość zamówionych pozycji tego rodzaju
- Price - money - cena za powyższe pozycje

Warunki integralnościowe:

- Cena musi być nieujemna;
  CHECK ([Price]>=(0))

```
CREATE TABLE [dbo].[OrderDetails](
	[OrderID] [int] NOT NULL,
	[EntryID] [int] NOT NULL,
	[Quantity] [float] DEFAULT 1 NOT NULL,
	[Price] [money] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC,
	[EntryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO
```

Orders

Zawiera informacje o zamówieniach.

- OrderID - PK - int - ID zamówienia
- CustomerID - FK→ Customers.CustomerID - int - ID klienta
- EmployeeID - FK → Employees.EmployeeID - int - ID pracownika realizującego zamówienie
- OrderDate - date - DEFAULT getdate() - data złożenia zamówienia
- TargetDate - date - DEFAULT getdate() - ustalony czas odebrania zamówienia
- PayLater - bit - DEFAULT 0 - czy klient płaci przy odbiorze zamówienia

Warunki integralnościowe :

- Daty zamówienia i odbioru nie mogą być z wcześniej niż XX wieku i data odbioru musi być późniejsza niż data złożenia zamówienia;
  CHECK ([OrderDate]>'01-01-1900' AND [TargetDate]>[OrderDate])

```
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] NOT NULL,
	[ClustomerID] [int] NOT NULL,
	[OrderDate] [datetime] DEFAULT getdate() NOT NULL,
	[TargetDate] [datetime] DEFAULT getdate() NOT NULL,
	[EmployeeID] [int] NOT NULL,
	[PayLater] [bit] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

Employees

Zawiera informacje o pracownikach restauracji.

- EmployeeID - PK - int - ID pracownika
- FirstName - nvarchar(50) - imię pracownika
- LastName - nvarchar(50) - nazwisko pracownika
- Role - nvarchar(50) - stanowisko sprawowane
- RoleDescription - nvarchar(1000) - opis stanowiska, obowiązków
- ReportTo - FK → Employees.EmployeeID - int - nullable - ID pracownika przełożonego
- BirthDate - date - data urodzenia pracownika

Warunki Integralnościowe:

- Data urodzenia nie może być z wcześniej niż 1900 roku i nie może być urodzony w przyszłości;
  CHECK ([BirthDate]>'01-01-1900' AND [BirthDate]<getdate())

```
CREATE TABLE [dbo].[Employees](
	[EmployeeID] [int] NOT NULL,
	[Firstname] [nvarchar](50) NOT NULL,
	[Lastname] [nvarchar](50) NOT NULL,
	[Role] [nvarchar](50) NOT NULL,
	[RoleDescription] [nvarchar](1000) NULL,
	[ReportsTo] [int] NULL,
	[BirthDate] [date] NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Administrators

Zawiera informacje o pracownikach, którzy są administratorami.

- EmployeeID - PK - int - ID pracownika

```
CREATE TABLE [dbo].[Administrators](
	[EmployeeID] [int] NOT NULL,
 CONSTRAINT [PK_Administrators] PRIMARY KEY CLUSTERED
(
	[EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

PrivateReservations

Zawiera informacje o rezerwacjach dokonanych przez klientów indywidualnych.

- ReservationID - PK - int - ID rezerwacji
- CustomerID - FK → PrivateCustomers.CustomerID- int - ID klienta, dokonującego rezerwacji
- OrderID - FK → Orders.OrderID - int - ID zamówienia
- TimeFrom - datetime - data i godzina początku rezerwacji
- TimeTo - datetime -data i godzina końca rezerwacji

Warunki integralnościowe:

- Data rezerwacji może być ustalona tylko na przyszłą datę, data zakończenia rezerwacji musi być późniejsza niż data rozpoczęcia;
  CHECK ([TimeFrom]>getdate() AND [TimeFrom]>[TimeTo])

```
CREATE TABLE [dbo].[PrivateReservations](
	[ReservationID] [int] NOT NULL,
	[CustomerID] [int] NOT NULL,
	[OrderID] [int] NOT NULL,
	[TimeFrom] [datetime] NOT NULL,
	[TimeTo] [datetime] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

PrivateReservationDetails

Zawiera informacje o szczegółach rezerwacji indywidualnych.

- ReservationID - PK - int - ID rezerwacji
- TableID - FK → Tables.TableID - int - ID stolika przypisanego do rezerwacji
- GuestsNumber - int - ilość gości do danej rezerwacji

Warunki integralnościowe:

- Liczba gości, którzy są planowani na rezerwację (nie licząc rezerwującego) musi być większa, bądź równa 0;
  CHECK([GuestsNumber] >= (0))

```
CREATE TABLE [dbo].[PrivateReservationDetails](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NOT NULL,
	[GuestsNumber] [int] NOT NULL,
 CONSTRAINT [PK_PrivateReservationDetails] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

PrivateStatusHistories

Zawiera historię statusów rezerwacji składanych przez klientów indywidualnych.

- ReservationID - PK - int - ID rezerwacji
- StatusID - FK → ReservationStatuses.StatusID - int - ID statusu rezerwacji
- TimeFrom - datetime - DEFAULT getdate() - data i czas od kiedy dany status był aktualny
- TimeTo - datetime - data i czas do kiedy dany status był aktualny

Warunki integralnościowe:

- Nie istnieją rekordy statusów rezerwacji sprzed 1900 roku oraz status rezerwacji musi się kończyć po tym jak się rozpoczął
  CHECK ([TimeFrom]>'01-01-1900' AND [TimeTo]>[TimeFrom])

```
CREATE TABLE [dbo].[PrivateStatusHistories](
	[ReservationID] [int] NOT NULL,
	[StatusID] [int] NOT NULL,
	[TimeFrom] [datetime] NOT NULL DEFAULT getdate() ,
	[TimeTo] [datetime] NULL,
 CONSTRAINT [PK_PrivateStatusHistories] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[StatusID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tables

Zawiera informacje o stolikach, słownik stolików.

- TableID - PK - int - ID stolika
- SeatsNo - int - ilość miejsc siedzących przy danym stoliku

Warunki integralnościowe:

- Ilość siedzeń przy stoliku musi być większa od 0
  CHECK ([SeatsNo]>(0))

```
CREATE TABLE [dbo].[Tables](
	[TableID] [int] NOT NULL,
	[SeatsNo] [int] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
	[TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

ReservationStatuses

Zawiera informacje o statusach rezerwacji, słownik statusów.

- StatusID - PK - int - ID statusu
- StatusName - nvarchar(50) - nazwa statusu

Warunki integralnościowe:

- Nazwa statusu musi być unikalna;
  CHECK (UNIQUE (StatusName))

```
CREATE TABLE [dbo].[ReservationStatuses](
	[StatusID] [int] NOT NULL,
	[StatusName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_ReservationStatuses] PRIMARY KEY CLUSTERED
(
	[StatusID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

CompanyStatusHistories

Zawiera informacje o historii statusów dla rezerwacji firmowych.

- ReservationID - PK - int - ID rezerwacji
- StatusID - FK → ReservationStatuses.StatusID - int - ID statusu rezerwacji
- TimeFrom - datetime - DEFAULT getdate() - data i czas od kiedy dany status był aktualny
- TimeTo - datetime - data i czas do kiedy dany status był aktualny

Warunki integralnościowe:

- data początkowa musi być późniejsza niż aktualna data;
  CHECK ([TimeFrom] >= getdate()
- data końcowa musi być późniejsza niż data rozpoczęcia;
  CHECK ([TimeTo]>[TimeFrom])

```
CREATE TABLE [dbo].[CompanyStatusHistories](
	[ReservationID] [int] DEFAULT getdate() NOT NULL ,
	[StatusID] [int] NOT NULL,
	[TimeFrom] [datetime] NOT NULL,
	[TimeTo] [datetime] NULL,
 CONSTRAINT [PK_CompanyStatusHistories] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[StatusID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

CompanyReservations

Zawiera informacje o rezerwacjach dokonanych przez klientów firmowych.

- ReservationID - PK - int - ID rezerwacji
- CompanyID - FK → CompanyCustomers.CompanyCustomerID - int - ID klienta firmowego
- TimeFrom - datetime - data i czas początku rezerwacji
- TimeTo - datetime - data i czas zakończenia rezerwacji

Warunki integralnościowe:

- data początku rezerwacji musi być późniejsza niż aktualna data;
  CHECK ([TimeFrom]>getdate())
- data zakończenia rezerwacji musi być późniejsza niż data rozpoczęcia;
  CHECK ([TimeTo]>[TimeFrom])

```
CREATE TABLE [dbo].[CompanyReservations](
	[ReservationID] [int] NOT NULL,
	[CompanyID] [int] NOT NULL,
```

```
	[TimeFrom] [datetime] NOT NULL,
	[TimeTo] [datetime] NOT NULL,
 CONSTRAINT [PK_CompanyReservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

CompanyReservationDetails

Zawiera informacje o szczegółach rezerwacji firmowych.

- ReservationID - PK - int - ID rezerwacji
- TableID - FK → Tables.TableID int - ID stolika
- CustomerID - FK - int - ID klienta

Warunki integralnościowe : brak

```
CREATE TABLE [dbo].[CompanyReservationDetails](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NOT NULL,
	[CustomerID] [int] NULL,
 CONSTRAINT [PK_CompanyReservationDetails_1] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

CompanyCustomers

Zawiera informacje o klientach firmowych.

- CompanyCustomerID - PK - int - ID klienta firmowego
- CompanyName - nvarchar(50) - nazwa firmy
- ContactName - nvarchar(50) - reprezentant firmy, osoba kontaktowa
- HomePage - nvarchar(50) - adres URL strony internetowej

Warunki integralnościowe

- Nazwa firmy musi być unikalna;
  UNIQUE (CompanyName)

```
CREATE TABLE [dbo].[CompanyCustomers](
	[CompanyCustomerID] [int] NOT NULL,
	[CompanyName] [nvarchar](50) NULL,
	[ContactName] [nvarchar](50) NULL,
	[HomePage] [nvarchar](50) NULL,
 CONSTRAINT [PK_CompanyCustomers] PRIMARY KEY CLUSTERED
(
	[CompanyCustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

PrivateCustomers

Zawiera informacje o klientach indywidualnych.

- PrivateCustomerID - PK - int - ID prywatnego klienta
- CompanyID - FK -> Customer.CustomerID - int - ID firmy, w której pracuje dana osoba
- Firstname - nvarchar(50) - imię klienta
- Lastname - nvarchar(50) - nazwisko klienta

```
CREATE TABLE [dbo].[PrivateCustomers](
	[PrivateCustomerID] [int] NOT NULL,
	[CompanyID] [int] NULL,
	[Fristname] [nvarchar](50) NOT NULL,
	[Lastname] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_PrivateCustomers] PRIMARY KEY CLUSTERED
(
	[PrivateCustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Customers

Zawiera informacje o wszystkich klientach.

- CustomerID - PK - int - ID klienta
- Address - nvarchar(50) - adres klienta
- Phone - nvarchar(50) - numer telefonu klienta

Warunki integralnościowe :

- UNIQUE (Phone)

```
CREATE TABLE [dbo].[Customers](
	[CustomerID] [int] NOT NULL,
	[Address] [int] NOT NULL,
	[Phone] [varchar](15) NOT NULL,
 CONSTRAINT [PK_Customer] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Cities

Zawiera informacje o miastach, słownik miast.

- CityID - PK - int - ID miasta
- CountryID - FK → Countries.CountryID - int - ID kraju
- CityName - nvarchar(50) - nazwa miasta

Warunki integralnościowe :

- Nazwa miasta musi być unikalna;
  UNIQUE (CityName)

```
CREATE TABLE [dbo].[Cities](
	[CityID] [int] NOT NULL,
	[CityName] [nvarchar](100) NOT NULL,
	[CountryID] [int] NOT NULL,
 CONSTRAINT [PK_City] PRIMARY KEY CLUSTERED
(
	[CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Countries

Zawiera informacje o krajach, słownik krajów.

- CountryID - PK - int - ID kraju
- CountryName - nvarchar(50) - nazwa kraju

Warunki integralnościowe :

- Nazwa kraju musi być unikalna;
  UNIQUE (CountryName)

```
CREATE TABLE [dbo].[Countries](
	[CountryID] [int] NOT NULL,
	[CountryName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Country] PRIMARY KEY CLUSTERED
(
	[CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

TimedDiscountsParameters

Zawiera ona parametry opisujące ograniczenia i wartości zniżek dożywotnich.

- ParameterID - int - PK - ID parametru
- ParameterName - varchar - Zawiera nazwę parametru
- Unit - varchar - Nullable - Zawiera jednostkę parametru, jeśli null to jest traktowany jako zwykła liczba

Warunki integralnościowe:

- Nazwa parametru musi być unikalna;
  UNIQUE (ParameterName)

```
CREATE TABLE [dbo].[TimedDiscountsParameters](
	[ParameterID] [int] NOT NULL,
	[ParameterName] [nvarchar](50) NULL,
	[Unit] [nchar](10) NULL,
 CONSTRAINT [PK_TimedDiscounts] PRIMARY KEY CLUSTERED
(
	[ParameterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

TimeDiscountsHistory

Zawiera informacje o wartościach parametrów zniżek dożywotnich w określonych odstępach czasu

- OneTimeDiscountID - PK - int - ID zmiany parametru
- ValidFrom - date -DEFAULT getdate()- data, od której obowiązuje parametr
- ValidTo - date - nullable - data, do której obowiązuje parametr. Jeśli null lub > getdate() to nadal obowiązuje
- ParameterValue - int - wartość parametru
- ParameterID - FK →TimedDiscountsParameters.ParameterID - int - ID parametru, którego dotyczy historia

Warunki integralnościowe:

- data początkowa musi być późniejsza niż 01.01.1900;
  CHECK ([ValidFrom]>'01-01-1900')
- data ważności zniżki musi późniejsza niż aktualna data;
  CHECK ([ValidTo]>getdate())

```
CREATE TABLE [dbo].[TimedDiscountsHistory](
	[TimedDiscountID] [int] DEFAULT getdate() NOT NULL,
	[ValidFrom] [date] NOT NULL,
	[ValidTo] [date] NULL,
	[ParameterValue] [int] NOT NULL,
	[ParameterID] [int] NOT NULL,
 CONSTRAINT [PK_TimedDiscountsHistory] PRIMARY KEY CLUSTERED
(
	[TimedDiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

GrantedOneTimeDiscounts

Zawiera informacje o przyznanych zniżkach dożywotnich.

- CustomerID - PK - int - ID klienta, dla którego przyznana jest zniżka
- OneTimeDiscountID- FK -> OneTimeDiscountsParameters.ParameterID - int - ID parametru przyznanej zniżki
- ActiveFrom - datetime -DEFAULT getdate()- data i czas aktywacji zniżki dożywotniej

Warunki integralnościowe:

- data aktywacji musi być późniejsza niż 01.01.1900;
  CHECK ([ActiveFrom]>='01-01-1900')

```
CREATE TABLE [dbo].[GrantedOneTimeDiscounts](
	[CustomerID] [int] NOT NULL,
	[ActiveFrom] [datetime]  DEFAULT getdate() NOT NULL,
	[OneTimeDiscountID] [int] NOT NULL,
 CONSTRAINT [PK_GrantedOneTimeDiscounts] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

ReservationRequirements

Zawiera wymagania, które trzeba spełnić, żeby móc dokonać rezerwacji.

- ConstID - PK - int - ID stałej
- ConstName - nvarchar(50) - nazwa stałej
- ConstValue - int - wartość stałej

Warunki integralnościowe:

- Nazwa stałej musi być unikalna;
  UNIQUE(ConstName)

```
CREATE TABLE [dbo].[ReservationRequirements](
	[ConstID] [int] NOT NULL,
	[ConstName] [varchar](50) NOT NULL,
	[ConstValue] [int] NOT NULL,
 CONSTRAINT [PK_ReservationRequirements] PRIMARY KEY CLUSTERED
(
	[ConstID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

CustomersWithNoDiscountAccess

Zawiera klientów, którzy nie mają dostępu do żadnej zniżki.

- CustomerID - PK - int - ID klienta który nie może mieć przyznanej zniżki. np. stworzony klient na którego będą zapisywane wszystkie zamówienia niezarejestrowanych klientów

```
CREATE TABLE [dbo].[CustomersWithNoDiscountAccess](
	[CustomerID] [int] NOT NULL,
 CONSTRAINT [PK_CustomersWithNoDiscountAccess] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
```

Widoki:

1. ALL_EMPLOYEES - Pracownicy, ich role oraz przełożeni

```
CREATE VIEW ALL_EMPLOYEES
as
select Firstname, Lastname, Role, (
    select Firstname + ' ' + Lastname from Employees E2 where
E1.ReportsTo = E2.EmployeeID
    ) as ReportsTo from Employees E1
```

2. ALL_SUPERIORS - Lista podwładnych dla każdego z pracowników

```
CREATE VIEW ALL_SUPERIORS
as
SELECT m.FirstName + ' ' +  m.LastName as Supervisor, s.FirstName + ' '
+ S.LastName Subordinate From Employees m
    INNER JOIN Employees s on m.EmployeeID = s.ReportsTo
```

3. CURRENT_MENU - produkty ich ceny oraz kategorie dla produktów aktualnie dostępnych w menu

```
CREATE VIEW ALL_EMPLOYEES
as
select Firstname, Lastname, Role, (
    select Firstname + ' ' + Lastname from Employees E2 where
E1.ReportsTo = E2.EmployeeID
    ) as ReportsTo from Employees E1
```

4. MENU_CHANGES - Zmiany w menu w ciągu ostatnich dwóch tygodni - ID zmiany, Podukt, Kategoria, Cena oraz data obowiązywania

```
CREATE VIEW MENU_CHANGES
AS
SELECT DISTINCT * from MENU_HISTORY
where DATEDIFF(day, TimeFrom, getdate()) <= 14 or (DATEDIFF(day, TimeTo,
getdate()) <= 14 and TimeTo IS NOT NULL)
```

5. PRIVATE_RESERVATIONS - Wszystkie rezerwacje prywatne, ich statusy oraz daty wraz ze stolikami oraz nr klienta dla klientów indywidualnych.

```
CREATE VIEW PRIVATE_RESERVATIONS
as
SELECT DISTINCT PR.ReservationID,
     PR.TimeFrom as 'Reservation Start',
     PR.TimeTo as 'Reservation End',
     PSH.TimeFrom as 'Status Start',
     PSH.TimeTo as 'Status End',
     RS.StatusName as 'Status',
     PC.Fristname + ' ' + PC.Lastname as 'Person Assigned',
     P.TableID as 'Assigned Table' from PrivateReservations PR
INNER JOIN PrivateStatusHistories PSH on PR.ReservationID =
PSH.ReservationID
INNER JOIN ReservationStatuses RS on PSH.StatusID = RS.StatusID
INNER JOIN PrivateReservationDetails PRD on PR.ReservationID =
PRD.ReservationID
INNER JOIN PrivateReservationDetails P on PR.ReservationID =
P.ReservationID
Inner JOIN PrivateCustomers PC on PR.CustomerID = PC.PrivateCustomerID
```

6. CURRENT_PRIVATE_RESERVATIONS - Wszystkie aktualne (przyszłe) rezerwacje prywatne, ich statusy oraz datę wraz ze stolikami oraz nr klienta dla klientów indywidualnych

```
CREATE VIEW CURRENT_PRIVATE_RESERVATIONS
as
SELECT ReservationID, [Reservation Start], [Reservation End], Status,
[Person Assigned], [Assigned Table]
FROM PRIVATE_RESERVATIONS
where [Status End] IS NULL and [Reservation End] < GETDATE()
```

7. COMPANY_RESERVATIONS Wszystkie rezerwacje firmowe, ich status oraz data wraz z nr. pracowników i stolikami

```
CREATE VIEW dbo.COMPANY_RESERVATIONS
AS
SELECT CR.ReservationID,
      CR.TimeFrom AS [Reservation Start],
      CR.TimeTo AS [Reservation End],
      RS.StatusName AS [Status],
      CSH.TimeFrom AS [Status Start],
      CSH.TimeTo AS [Status End],
      PC.Fristname + ' ' + PC.Lastname AS [Person Assigned],
      C.TableID AS [Assigned Table]
FROM     dbo.CompanyReservations AS CR INNER JOIN
        dbo.CompanyStatusHistories AS CSH ON CR.ReservationID =
CSH.ReservationID INNER JOIN
        dbo.ReservationStatuses AS RS ON CSH.StatusID = RS.StatusID
INNER JOIN
        dbo.CompanyReservationDetails AS CRD ON CR.ReservationID =
CRD.ReservationID INNER JOIN
        dbo.CompanyReservationDetails AS C ON CR.ReservationID =
C.ReservationID INNER JOIN
                 dbo.PrivateCustomers AS PC ON CRD.CustomerID =
PC.PrivateCustomerID
```

8. CURRENT_COMPANY_RESERVATIONS Wszystkie aktualne (przyszłe) rezerwacje firmowe, ich status oraz datę wraz z nr. pracowników i stolikami

```
CREATE VIEW CURRENT_COMPANY_RESERVATIONS
as
SELECT ReservationID, [Reservation Start], [Reservation End], Status,
[Person Assigned], [Assigned Table]
FROM COMPANY_RESERVATIONS
where [Status End] IS NULL and [Reservation End] < GETDATE()
```

9. PARAMETERS_VIEW - Obowiązujące aktualnie parametry rabatów

```
CREATE VIEW PARAMETERS_VIEW
AS
SELECT EntryID, ParametersHistory.ParameterID, ParameterValue,
ParameterName from ParametersHistory
INNER JOIN DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
where ValidTo IS NULL or ValidTo > getdate()
```

10. RESERVATION_REQUIREMENTS_VIEW - Obowiązujące wymagania do rezerwacji

```
CREATE VIEW RESERVATION_REQUIREMENTS_VIEW
AS
SELECT ConstName, ConstValue from ReservationRequirements
```

11. DISCOUNTS_VIEW - Widok na klientów wszystkich klientów i przyznane im rabaty

```
CREATE VIEW DISCOUNTS_VIEW
AS
SELECT C.CustomerID, PH.ParameterValue as 'Discount Percentage',
ActiveFrom, ActiveTo from Customers C
left join GrantedTimedDiscounts GTD on C.CustomerID = GTD.CustomerID
left join ParametersHistory PH on GTD.DiscountID = PH.EntryID
UNION
SELECT C.CustomerID, PH.ParameterValue as 'Discount Percentage',
ActiveFrom, 'Active Forever' from Customers C
left join GrantedOneTimeDiscounts GOTD on C.CustomerID = GOTD.CustomerID
left join ParametersHistory PH on GOTD.DiscountID = PH.EntryID
```

12. Widok na raporty miesięczny i tygodniowy dotyczące rezerwacji stolików, rabatów, menu, a także statystyk zamówienia – dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień - W PUNKTACH 11, 13, 5, 7

13. ORDER_VALUES - Raport dotyczący zamówień (klient - wartości konkretnych zamówień)

```
create view dbo.ORDER_VALUES as
select Name as 'Customer Name', O.OrderID, OrderDate as
'Order Date', sum(Price*Quantity) as 'Order Value' from
(select CompanyName as 'Name', CompanyCustomerID as
'CustomerID' from CompanyCustomers
union
```

```
    select Fristname+' '+Lastname as 'Name', PrivateCustomerID
    from PrivateCustomers) as CST inner join Customers C on
    CST.CustomerID = C.CustomerID
    inner join Orders O on C.CustomerID = O.ClustomerID inner
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by Name, O.OrderID, OrderDate
    go
```

14. ORDERS_SUMMARY - Raport dotyczący zamówień (łączna wartość i liczba zamówień danego klienta)

```
CREATE view dbo.ORDERS_SUMMARY as
select [Customer Name], sum([Order Value]) as 'Order Value
Combined', count([OrderID]) as 'Number of orders' from
ORDER_VALUES
group by [Customer Name]
go
```

15. PLANNED_MENU_CHANGES - Widok planowanych zmian dostępności w menu

```
create view dbo.PLANNED_MENU_CHANGES(ProductName, TimeFrom,
TimeTo, Description, CategoryName) as
select ProductName, TimeFrom, TimeTo, Description, CategoryName
from Menu M
        inner join Products P on M.ProductID = P.ProductID
        inner join Categories C on C.CategoryID = P.CategoryID
where TimeFrom > getdate()
  or TimeTo is null
  or TimeTo >= getdate()
go
```

16. CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS - Klienci indywidualni którzy mogą robić rezerwacje stolika z zamówieniem

```
create view dbo.CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS as
select PrivateCustomerID, Fristname, Lastname
from PrivateCustomers PC
   inner join Customers C on PC.PrivateCustomerID = C.CustomerID
   inner join Orders O on C.CustomerID = O.ClustomerID
group by PrivateCustomerID, Fristname, Lastname
having count(OrderID) >= (SELECT ConstValue from ReservationRequirements
where ConstName = 'WK'
```

17. MENU_HISTORY - Historia zmian w menu

```
CREATE VIEW MENU_HISTORY
AS
SELECT DISTINCT PH.EntryID, ProductName, C.CategoryName, PH.Price,
PH.TimeFrom, PH.TimeTo from Products
inner join Categories C on C.CategoryID = Products.CategoryID
inner join PriceHistory PH on Products.ProductID = PH.ProductID
```

18. TOP_DISCOUNTS - Najczęściej przyznawane zniżki

```
CREATE VIEW TOP_DISCOUNTS
AS
Select top 10 EntryID as 'Discount ID', [Discount Percentage],  Count(*)
as 'GrantedTimes', Type from DISCOUNTS_VIEW
where EntryID is not NULL
group by EntryID, [Discount Percentage], Type
order by 2 desc
go
```

19. PRODUCT_SUMMARY - Sprzedane produkty w każdym z zamówień (nazwa, kategoria, data, ilość)

```
create view dbo.PRODUCT_SUMMARY as
select ProductName, CategoryName, OrderDate, sum(Quantity) as 'Total Quantity'
from Products P
   inner join PriceHistory PH on P.ProductID = PH.ProductID
   inner join Categories C on P.CategoryID = C.CategoryID
   inner join OrderDetails OD on PH.EntryID = OD.EntryID
   inner join Orders O on OD.OrderID = O.OrderID
group by P.ProductID, ProductName, CategoryName, OrderDate
go
```

20. TOP_PRODUCTS_QUANTITY - Wyświetla 10 najpopulatniejszych produktów według liczby sprzedanych jednostek

```
CREATE VIEW TOP_PRODUCTS_QUANTITY
AS
SELECT TOP 10 P.ProductID, P.ProductName, SUM(Quantity) as 'Total
Quantity' from OrderDetails
INNER JOIN PriceHistory PH on PH.EntryID = OrderDetails.EntryID
INNER JOIN Products P on P.ProductID = PH.ProductID
group by P.ProductID, P.ProductName
order by 3 desc
```

21. TOP_PRODUCTS_VALUE - Wyświetla 10 najpopulatniejszych produktów według wartości sprzedaży

```
CREATE VIEW TOP_PRODUCTS_VALUE
AS
SELECT TOP 10 P.ProductID, P.ProductName,
SUM(Quantity*OrderDetails.Price) as 'Total Quantity' from OrderDetails
INNER JOIN PriceHistory PH on PH.EntryID = OrderDetails.EntryID
INNER JOIN Products P on P.ProductID = PH.ProductID
group by P.ProductID, P.ProductName
order by 3 desc
```

22. CURRENT_DISCOUNTS - Widok przedstawiający aktualnie przyznane zniżki

```
CREATE VIEW CURRENT_DISCOUNTS
AS
SELECT PH.EntryID, C.CustomerID, PH.ParameterValue as 'Discount
Percentage', ActiveFrom, ActiveTo from Customers C
inner join GrantedTimedDiscounts GTD on C.CustomerID = GTD.CustomerID
inner join ParametersHistory PH on GTD.DiscountID = PH.EntryID
where ActiveTo IS Null or ActiveTo > getdate()
UNION
SELECT PH.EntryID, C.CustomerID, PH.ParameterValue as 'Discount
Percentage', ActiveFrom, Null from Customers C
inner join GrantedOneTimeDiscounts GOTD on C.CustomerID =
GOTD.CustomerID
inner join ParametersHistory PH on GOTD.DiscountID = PH.EntryID
```

23. AWAITING_APPROVAL - Rezerwacje oczekujące na zatwierdzenie przez pracownika

```
CREATE VIEW AWAITING_APPROVAL
AS
Select ReservationID, [Reservation Start], [Reservation End], Status
from PRIVATE_RESERVATIONS where Status='Placed' and [Status End] is NULL
UNION
Select ReservationID, [Reservation Start], [Reservation End], Status
from COMPANY_RESERVATIONS where Status='Placed' and [Status End] is NULL
```

24. WEEK_REPORT - Łączne wartości zamówień z podziałem na tygodnie

```
CREATE VIEW dbo.WEEK_REPORT as
select year as Year, week as Week, sum([total order value]) as [Order Value]
from (select datepart(year, [OrderDate]) as year, datepart(week, [OrderDate]) as
'week', [Order Value] as 'total order value' from ORDER_VALUES) as T group by
T.year, T.week
go
```

25. ORDER_VALUES_MONTH_REPORT - Łączne wartości zamówień z podziałem na miesiące

```
CREATE VIEW dbo.ORDER_VALUES_MONTH_REPORT as
select datepart(year, [OrderDate])  as Year,
       datepart(month, [OrderDate]) as Month,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES
group by datepart(year, [OrderDate]), datepart(month, [OrderDate])
```

26. ORDER_VALUES_MONTH_REPORT_COMPANY - Łączne wartości zamówień z podziałem na miesiące dla firm

```
CREATE VIEW dbo.ORDER_VALUES_MONTH_REPORT_COMPANY as
select datepart(year, [OrderDate])  as Year,
       datepart(month, [OrderDate]) as Month,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES_COMPANY
group by datepart(year, [OrderDate]), datepart(month, [OrderDate])
```

27. ORDER_VALUES_MONTH_REPORT_PRIVATE - Łączne wartości zamówień z podziałem na miesiące dla klientów prywatnych

```
CREATE VIEW dbo.ORDER_VALUES_MONTH_REPORT_PRIVATE as
select datepart(year, [OrderDate])  as Year,
       datepart(month, [OrderDate]) as Month,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES_PRIVATE
group by datepart(year, [OrderDate]), datepart(month, [OrderDate])
```

28. ORDER_VALUES_REPORT_PRIVATE- Łączne wartości zamówień z podziałem na miesiące i tygodnie dla klientów prywatnych

```
CREATE VIEW dbo.ORDER_VALUES_REPORT_PRIVATE as
select datepart(year, [OrderDate])  as Year,
```

```
       datepart(month, [OrderDate]) as Month,
       datepart(week, [OrderDate])  as Week,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES_PRIVATE
group by datepart(year, [OrderDate]), datepart(month, [OrderDate]),
datepart(week, [OrderDate])
```

29. ORDER_VALUES_REPORT_COMPANY- Łączne wartości zamówień z podziałem na miesiące i tygodnie dla klientów firmowych

```
CREATE VIEW dbo.ORDER_VALUES_REPORT_COMPANY as
select datepart(year, [OrderDate])  as Year,
       datepart(month, [OrderDate]) as Month,
       datepart(week, [OrderDate])  as Week,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES_COMPANY
group by datepart(year, [OrderDate]), datepart(month, [OrderDate]),
datepart(week, [OrderDate])
```

30. ORDER_VALUES_REPORT - Łączne wartości zamówień z podziałem na miesiące i tygodnie dla klientów prywatnych

```
CREATE VIEW dbo.ORDER_VALUES_REPORT as
select datepart(year, [OrderDate])  as Year,
       datepart(month, [OrderDate]) as Month,
       datepart(week, [OrderDate])  as Week,
       sum([Order Value])           as [Order Value]
from ORDER_VALUES
group by datepart(year, [OrderDate]), datepart(month, [OrderDate]),
datepart(week, [OrderDate])
```

31. ORDER_VALUES_WEEK_REPORT - Łączne wartości zamówień z podziałem na tygodnie

```
CREATE VIEW dbo.ORDER_VALUES_WEEK_REPORT as
select datepart(year, [OrderDate]) as Year, datepart(week, [OrderDate])
as Week, sum([Order Value]) as [Order Value]
from ORDER_VALUES
group by datepart(year, [OrderDate]), datepart(week, [OrderDate])
```

32. ORDER_VALUES_WEEK_REPORT_COMPANY - Łączne wartości zamówień z podziałem na tygodnie dla firm

```
CREATE VIEW dbo.ORDER_VALUES_WEEK_REPORT_COMPANY as
select datepart(year, [OrderDate]) as Year, datepart(week, [OrderDate])
as Week, sum([Order Value]) as [Order Value]
from ORDER_VALUES_COMPANY
group by datepart(year, [OrderDate]), datepart(week, [OrderDate]
```

33. ORDER_VALUES_WEEK_REPORT_PRIVATE - Łączne wartości zamówień z podziałem na tygodnie dla klientów prywatnych

```
CREATE VIEW dbo.ORDER_VALUES_WEEK_REPORT_PRIVATE as
select datepart(year, [OrderDate]) as Year, datepart(week, [OrderDate])
as Week, sum([Order Value]) as [Order Value]
from ORDER_VALUES_PRIVATE
group by datepart(year, [OrderDate]), datepart(week, [OrderDate])
```

34. PRIVATE_RESERVATIONS_REPORT_WEEK

```
Create VIEW PRIVATE_RESERVATIONS_REPORT_WEEK
as
SELECT YEAR(TimeFrom) as Year, DATEPART(week, TimeFrom) as Week,
count(ReservationID) as 'Reservations Number', AVG(DATEDIFF(minute
,TimeFrom, TimeTo))/60.0 as 'Average Duration'  from PrivateReservations
group by  YEAR(TimeFrom), DATEPART(week, TimeFrom)
```

35. PRIVATE_RESERVATIONS_REPORT_MONTH

```
Create VIEW PRIVATE_RESERVATIONS_REPORT_MONTH
as
SELECT YEAR(TimeFrom) as Year, DATEPART(month, TimeFrom) as Month,
count(ReservationID) as 'ReservationsNumber', AVG(DATEDIFF(minute
,TimeFrom, TimeTo))/60.0 as 'AverageDuration' from PrivateReservations
group by  YEAR(TimeFrom), DATEPART(month , TimeFrom)
```

36. COMPANY_RESERVATIONS_REPORT_WEEK

```
SELECT YEAR(TimeFrom) as Year, DATEPART(week, TimeFrom) as Week,
count(ReservationID) as 'ReservationsNumber', AVG(DATEDIFF(minute
,TimeFrom, TimeTo))/60.0 as 'AverageDuration' from CompanyReservations
```

```
group by  YEAR(TimeFrom), DATEPART(week, TimeFrom)
```

37. COMPANY_RESERVATIONS_REPORT_MONTH

```
Create VIEW COMPANY_RESERVATIONS_REPORT_MONTH
as
SELECT YEAR(TimeFrom) as Year, DATEPART(month, TimeFrom) as Month,
count(ReservationID) as 'ReservationsNumber', AVG(DATEDIFF(minute
,TimeFrom, TimeTo))/60.0 as 'AverageDuration' from CompanyReservations
group by  YEAR(TimeFrom), DATEPART(month , TimeFrom)
```

38. TIMED_DISCOUNTS_VIEW - Widok na przyznane zniżki czasowe

```
CREATE VIEW TIMED_DISCOUNTS_VIEW
AS
SELECT C.CustomerID, PH.EntryID, PH.ParameterValue as 'Discount
Percentage', ActiveFrom, ActiveTo from Customers C
left join GrantedTimedDiscounts GTD on C.CustomerID = GTD.CustomerID
left join ParametersHistory PH on GTD.DiscountID = PH.EntryID
```

39. ONE_TIME_DISCOUNTS_VIEW - Widok na przyznane zniżki dożywotnie

```
CREATE VIEW ONE_TIME_DISCOUNTS_VIEW
AS
SELECT C.CustomerID, PH.EntryID, PH.ParameterValue as 'Discount
Percentage', ActiveFrom, Null as ActiveTo from Customers C
left join GrantedOneTimeDiscounts GOTD on C.CustomerID = GOTD.CustomerID
left join ParametersHistory PH on GOTD.DiscountID = PH.EntryID
```

40. TIMED_DISCOUNTS_REPORT_MONTH- Miesięczny raport ilości przyznanych zniżek czasowych

```
CREATE VIEW TIMED_DISCOUNTS_REPORT_MONTH
as
Select YEAR(ActiveFrom) as Year, MONTH(ActiveFrom) as Month, EntryID as
'Discount ID', [Discount Percentage],  Count(*) as 'Granted times' from
TIMED_DISCOUNTS_VIEW
where EntryID is not NULL
group by EntryID, YEAR(ActiveFrom), MONTH(ActiveFrom), [Discount
Percentage]
```

41. TIMED_DISCOUNTS_REPORT_WEEK - Tygodniowy raport ilości przyznanych zniżek czasowych

```
CREATE VIEW TIMED_DISCOUNTS_REPORT_WEEK
as
Select YEAR(ActiveFrom) as Year, DATEPART(WEEK, ActiveFrom) as Week,
EntryID as 'Discount ID', [Discount Percentage],  Count(*) as 'Granted
times' from TIMED_DISCOUNTS_VIEW
where EntryID is not NULL
group by EntryID, YEAR(ActiveFrom), DATEPART(WEEK, ActiveFrom),
[Discount Percentage]
```

42. ONE_TIME_DISCOUNTS_REPORT_MONTH - Miesięczny raport ilości przyznanych zniżek dożywotnich

```
CREATE VIEW ONE_TIME_DISCOUNTS_REPORT_MONTH
as
Select YEAR(ActiveFrom) as Year, MONTH(ActiveFrom) as Month, EntryID as
'Discount ID', [Discount Percentage],  Count(*) as 'Granted times' from
ONE_TIME_DISCOUNTS_VIEW
where EntryID is not NULL
group by EntryID, YEAR(ActiveFrom), MONTH(ActiveFrom), [Discount
Percentage]
```

43. ONE_TIME_DISCOUNTS_REPORT_WEEK- Tygodniowy raport ilości przyznanych zniżek dożywotnich

```
CREATE VIEW ONE_TIME_DISCOUNTS_REPORT_WEEK
as
Select YEAR(ActiveFrom) as Year, DATEPART(WEEK, ActiveFrom) as Week,
EntryID as 'Discount ID', [Discount Percentage],  Count(*) as 'Granted
times' from ONE_TIME_DISCOUNTS_VIEW
where EntryID is not NULL
group by EntryID, YEAR(ActiveFrom), DATEPART(WEEK, ActiveFrom),
[Discount Percentage]
```

44. PRIVATE_CUSTOMER_VALUES_REPORT_MONTH - Raport przedstawiający miesięczne wydatki dla każdego klienta indywidualnego

```
CREATE VIEW PRIVATE_CUSTOMER_VALUES_REPORT_MONTH
AS
Select YEAR(OrderDate) as Year, MONTH(OrderDate) as Month, CustomerID,
PC.Fristname + ' ' + PC.Lastname as Name,  SUM([Order Value]) as
'TotalValue',
      COUNT(OrderID) 'OrderCount' from ORDER_VALUES_PRIVATE
inner join PrivateCustomers PC on ORDER_VALUES_PRIVATE.CustomerID =
```

```
PC.PrivateCustomerID
group by CustomerID, PC.Fristname, PC.Lastname, YEAR(OrderDate),
MONTH(OrderDate)
```

45. ORDERS_WORKHORSE - Widok tylko do użytku przez inne widoki, wygodnie zrbiera wszystkie informacje o zamówieniach do jednego widoku z którego można je wyciągać w innych widokach

```
Create VIEW ORDERS_WORKHORSE
as
select O.OrderID,
      P.ProductID,
      ProductName,
      O.CustomerID,
      O.EmployeeID,
      OrderDate,
      Quantity,
      OrderDetails.Price,
      ISNULL(IIF(PH.ParameterValue < PH1.ParameterValue,
PH1.ParameterValue, PH.ParameterValue), 0) as 'DiscountPercentage',
      (1-ISNULL(IIF(PH.ParameterValue < PH1.ParameterValue,
PH1.ParameterValue, PH.ParameterValue)*0.01, 0)) * OrderDetails.Price as
'DiscountedPrice',
      IIF(PH.ParameterValue < PH1.ParameterValue, PH1.ParameterID,
PH.ParameterID) as 'DiscountID'
      from OrderDetails
Inner Join Orders O on O.OrderID = OrderDetails.OrderID
inner join Customers C on C.CustomerID = O.CustomerID
inner join PriceHistory PrH on OrderDetails.EntryID = PrH.EntryID
inner join Products P on PrH.ProductID = P.ProductID
left join GrantedOneTimeDiscounts GOTD on C.CustomerID = GOTD.CustomerID
left join GrantedTimedDiscounts GTD on C.CustomerID = GTD.CustomerID and
GTD.ActiveTo >= O.OrderDate  and GTD.ActiveFrom <= O.OrderDate
left join ParametersHistory PH on GOTD.DiscountID = PH.EntryID
left join ParametersHistory PH1 on GTD.DiscountID = PH1.EntryID
```

46. PRODUCTS_VALUES_MONTH - Widok zawierający informacje o tym ile dany produkt zarobił z podziałem na miesiące i lata

```
CREATE VIEW PRODUCTS_VALUES_MONTH
as
SELECT Year(OrderDate) as Year, MONTH(OrderDate) as Month, ProductID,
ProductName,
      SUM(DiscountedPrice) as TotalValue, COUNT(ProductID) as TimesSold
from ORDERS_WORKHORSE
```

```
GROUP BY Year(OrderDate), MONTH(OrderDate), ProductID, ProductName
```

47. PRODUCTS_VALUES_WEEK - Widok zawierający informacje o tym ile dany produkt zarobił z podziałem na miesiące i lata

```
PRODUCTS_VALUES_WEEK
CREATE VIEW PRODUCTS_VALUES_WEEK
as
SELECT Year(OrderDate) as Year, DATEPART(Week,OrderDate) as Week,
ProductID, ProductName,
      SUM(DiscountedPrice) as TotalValue, COUNT(ProductID) as TimesSold
from ORDERS_WORKHORSE
GROUP BY Year(OrderDate), DATEPART(WEEK, OrderDate), ProductID,
ProductName
```

48. COMPANY_CUSTOMER_VALUES_REPORT_MONTH - Raport przedstawiający miesięczne wydatki dla każdej firmy będącej klientem

```
CREATE VIEW COMPANY_CUSTOMER_VALUES_REPORT_MONTH
AS
Select YEAR(OrderDate) as Year,
MONTH(OrderDate) as Month,
CustomerID,
CC.CompanyName as Name,
SUM([Order Value]) as 'TotalValue',
COUNT(OrderID) 'OrderCount'
from ORDER_VALUES_COMPANY
inner join CompanyCustomers CC on ORDER_VALUES_COMPANY.CustomerID =
CC.CompanyCustomerID
group by CustomerID, CC.CompanyName, YEAR(OrderDate), MONTH(OrderDate)
```

49. PRIVATE_CUSTOMER_VALUES_REPORT_WEEK - Raport przedstawiający tygodniowe wydatki dla każdego klienta indywidualnego

```
CREATE VIEW PRIVATE_CUSTOMER_VALUES_REPORT_WEEK
AS
Select YEAR(OrderDate) as Year, DATEPART(week, OrderDate) as Week,
CustomerID, PC.Fristname + ' ' + PC.Lastname as Name,  SUM([Order
Value]) as 'TotalValue',
     COUNT(OrderID) 'OrderCount' from ORDER_VALUES_PRIVATE
inner join PrivateCustomers PC on ORDER_VALUES_PRIVATE.CustomerID =
PC.PrivateCustomerID
```

```
group by CustomerID, PC.Fristname, PC.Lastname, YEAR(OrderDate),
DATEPART(week, OrderDate)
```

50. COMPANY_CUSTOMER_VALUES_REPORT_WEEK - Raport przedstawiający tygodniowe wydatki dla każdej firmy będącej klientem

```
CREATE VIEW COMPANY_CUSTOMER_VALUES_REPORT_WEEK
AS
Select YEAR(OrderDate) as Year,
DATEPART(week, OrderDate) as WEEK,
CustomerID,
CC.CompanyName as Name,
SUM([Order Value]) as 'TotalValue',
COUNT(OrderID) 'OrderCount'
from ORDER_VALUES_COMPANY
inner join CompanyCustomers CC on ORDER_VALUES_COMPANY.CustomerID =
CC.CompanyCustomerID
group by CustomerID, CC.CompanyName, YEAR(OrderDate), DATEPART(week,
OrderDate)
```

51. PRODUCTS_ORDERED_REPORT_MONTH - raport ile razy w miesiącu zamówiono dany produkt

```
create view dbo.PRODUCTS_ORDERED_REPORT_MONTH as
select year(TargetDate) as YEAR, month(TargetDate) as Month, ProductName,
SUM(Quantity) as TotalQauntityOrdered from Orders O inner join OrderDetails OD
on O.OrderID = OD.OrderID
inner join Products P on OD.Price = P.Price group by year(TargetDate),
month(TargetDate), ProductName
go
```

52. PRODUCTS_ORDERED_REPORT_WEEK - raport ile razy w tygodniu zamówiono dany produkt

```
create view dbo.PRODUCTS_ORDERED_REPORT_MONTH as
select year(TargetDate) as YEAR, month(TargetDate) as Month, ProductName,
SUM(Quantity) as TotalQauntityOrdered from Orders O inner join OrderDetails OD
on O.OrderID = OD.OrderID
inner join Products P on OD.Price = P.Price group by year(TargetDate),
month(TargetDate), ProductName
go
```

Procedury

1. AddPrivateCustomer - Dodaje Osobę prywatną

```
CREATE PROCEDURE AddPrivateCustomer
@firstname nvarchar(50),
@lastname nvarchar(50),
@phone varchar(15),
@city int,
@companyId int
as
BEGIN
   DECLARE @customerID int;
   SET NOCOUNT ON;
   Begin TRY
       Insert Into Customers(City, Phone) values (@city, @phone)
       SET @customerID = @@IDENTITY
       Insert INTO PrivateCustomers(PrivateCustomerID, CompanyID,
Fristname, Lastname)
       values(
              @customerID,
              @companyId,
              @firstname,
              @lastname
             );
   end TRY
   BEGIN CATCH
       DECLARE @message NVARCHAR(1000) = N'Nie udało dodać się klienta
prywatnego: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch
end
go
```

2. AddCompanyCustomer - Dodaje Klienta-Firmę

```
CREATE PROCEDURE AddCompanyCustomer
@companyName nvarchar(50),
@phone varchar(15),
@city int,
@contactName nvarchar(50),
@homePage nvarchar(50)
as
BEGIN
   DECLARE @customerID int;
   SET NOCOUNT ON;
   Begin TRY
       Insert Into Customers(City, Phone) values (@city, @phone)
       SET @customerID = @@IDENTITY
       Insert INTO CompanyCustomers(companycustomerid, companyname,
contactname, homepage)
       values(
              @customerID,
              @companyName,
              @contactName,
              @homePage
             );
   end TRY
   BEGIN CATCH
       DECLARE @message NVARCHAR(1000) = N'Nie udało dodać się klienta
firmowego: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch

end
```

3. AddCountry - Dodaje Kraj

```
CREATE PROCEDURE AddCountry @countryName varchar(50)
as
BEGIN
```

```
   begin try
       insert into Countries values (@countryName)
   end try
   begin catch
       declare @message nvarchar(1000) = N'Nie udało się dodać kraju: ' +
ERROR_MESSAGE();
       throw 52000, @message, 1;
   end catch
end
go
```

4. AddCity - Dodaje miasto do istniejącego Kraju

```
create procedure AddCity @cityName nvarchar(100),
                         @countryID int
as
begin
   begin try
       insert into Cities values (@cityName, @countryID)
   end try
   begin catch
       DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać miasta: ' +
ERROR_MESSAGE();
       THROW 52000, @message, 1;
   end catch
end
go
```

5. ChangePrivateReservationStatus - kończy aktualny status reserwacji prywatnej i ustala ją na nowy

```
Create PROCEDURE ChangePrivateReservationStatus
@privateReservationID int,
@newStatusID int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
       UPDATE PrivateStatusHistories SET TimeTo=GETDATE() where
ReservationID=@privateReservationID;
       INSERT INTO PrivateStatusHistories (ReservationID, StatusID,
TimeFrom, TimeTo)
       VALUES (@privateReservationID, @newStatusID, getdate(), null)
   END TRY
   BEGIN CATCH
       DECLARE @message NVARCHAR(1000) = N'Nie udało się zmienić statusu
rezerwacji prywatnej: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message, 1;
```

```
   end catch
end
```

6. ChangeCompanyReservationStatus- kończy aktualny status reserwacji firmwej i ustala ją na nowy

```
Create PROCEDURE ChangeCompanyReservationStatus
@companyReservationID int,
@newStatusID int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
       UPDATE CompanyStatusHistories SET TimeTo=GETDATE() where
ReservationID=@companyReservationID;
       INSERT INTO CompanyStatusHistories (ReservationID, StatusID,
TimeFrom, TimeTo)
       VALUES (@companyReservationID, @newStatusID, getdate(), null)
   END TRY
   BEGIN CATCH
       DECLARE @message NVARCHAR(1000) = N'Nie udało się zmienić statusu
rezerwacji firmowej: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message, 1;
   end catch
end
go
```

7. AddEmployee - Dodaje Pracownika

```
create procedure AddEmployee @firstName nvarchar(50),
                             @lastname varchar(50),
                             @role varchar(50),
                             @roleDescription text = null,
                             @reportsTo int = null,
                             @birthDate date
as
begin
      begin try
      insert into Employees values (@firstName, @lastname, @role,
@roleDescription, @reportsTo, @birthDate)
      end try
      begin catch
      DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać pracownika:
' + ERROR_MESSAGE();
```

```
      THROW 52000, @message, 1;
      end catch
end
go
```

8. GrantTimedDiscount - Przyznaje danemu użytkownikowi zniżkę o podanym parametrze czasu i wartości

```
CREATE PROCEDURE GrantTimedDiscount
@CustomerID int
AS
BEGIN
   DECLARE @discountID int = (SELECT TOP 1 EntryID from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'TimedR%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL))
   DECLARE @duration int = (SELECT TOP 1 ParameterValue from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'TimedD%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL))
   DECLARE @ordersRequiredValue int = (SELECT TOP 1 ParameterValue from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'TimedK%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL));
   DECLARE @orderMaxValue money = (SELECT MAX (la.total) from (SELECT
sum(Price) as total from ORDERS_WORKHORSE where CustomerID=@CustomerID
group by OrderID) as la);

   If @orderMaxValue < @ordersRequiredValue
       BEGIN
           Throw 52000, 'The customer doesnt meet the current timed
discount parameters!',1;
       END
   ELSE
       BEGIN
           BEGIN TRY
               Insert Into GrantedTimedDiscounts (CustomerID,
DiscountID, ActiveFrom, ActiveTo)
```

```
                values (@CustomerID, @discountID, getdate(), DATEADD(day,
@duration, getdate()));
            END TRY
            BEGIN CATCH
                DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać
zniżki czasowej: ' + char(13) + char(10) + ERROR_MESSAGE();
                THROW 52000, @message, 1;
            end catch
        END

END
go
```

9. GrantLifetimeDiscount - Przyzjaje danemu klientowi zniżkę dożywotnią aktualnym podanym parametrze wartości

```
CREATE PROCEDURE GrantLifetimeDiscount
@CustomerID int
AS
BEGIN
   DECLARE @discountID int = (SELECT TOP 1 EntryID from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'ConstR%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL))
   DECLARE @requiredOrders int = (SELECT TOP 1 ParameterValue from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'ConstZ%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL))
  DECLARE @ordersCount int = (SELECT Count(*) from (Select distinct
OrderID from ORDERS_WORKHORSE where CustomerID = @CustomerID
   group by OrderID
   HAVING Sum(Price) >
       (SELECT TOP 1 ParameterValue from ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName like 'ConstK%' and ValidFrom < getdate()
and (ValidTo > getdate() or ValidTo is NULL))) as la);
   If @ordersCount < @requiredOrders
       BEGIN
           Throw 52000, 'The customer doesnt meet the current lifetime
discount parameters!',1;
```

```
        END

    BEGIN TRY
        Insert Into GrantedOneTimeDiscounts (CustomerID, ActiveFrom,
DiscountID)
        values (@CustomerID, getdate(), @discountID);
    END TRY
    BEGIN CATCH
        DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać zniżki!: '
+ char(13) + char(10) + ERROR_MESSAGE();
        THROW 52000, @message, 1;
    end catch
END
```

10. AddProduct - Dodaje produkt do listy produktów - dodawać od razu do menu??

```
create procedure AddProduct @description nvarchar(1000) = null,
                            @price int,
                            @categoryID int,
                            @productName nvarchar(100)
as
begin
    begin try
        insert into Products values (@description, @price, @categoryID,
@productName)
    end try
    begin catch
        DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać produktu: ' +
ERROR_MESSAGE();
        THROW 52000, @message, 1;
    end catch
end
```

11. AddToMenu - Dodaje podany produkt do menu i ustala jego dostępność od aktualnej daty aż do podanej

```
create procedure AddToMenu @productID int, @price money, @timeFrom date = getdate(), @timeTo date =
null
as
begin
    begin try
        insert into PriceHistory values (@productID, @price, @timeFrom, @timeTo)
    end try
    begin catch
        DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać produktu do menu: ' +
ERROR_MESSAGE();
        THROW 52000, @message, 1;
    end catch
end
go
```

12. RemoveFromCurrentMenu - wyłącza dostępność danego entryID (TimeTo na aktualną datę)

```
create procedure RemoveFromMenu @entryID int
as
begin try
   update PriceHistory set TimeTo = getdate() where EntryID = @entryID
end try
begin catch
   DECLARE @message NVARCHAR(1000) = N'Nie udało się ustawić terminu dotępości
produktu: ' + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch
```

13. SetToTimeToDateMenu - Ustala datę w której dostępność danego dania powinna zostać wyłączona

```
create procedure RemoveFromMenu @entryID int, @dateOfRemoval date
as
begin try
   update PriceHistory set TimeTo = @dateOfRemoval where EntryID = @entryID
end try
begin catch
   DECLARE @message NVARCHAR(1000) = N'Nie udało się ustawić terminu dostępności
produktu: ' + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch
```

14. AddOrder - Dodaje zamówienie do historii zamówień

```
CREATE PROCEDURE AddOrder
@CustomerID int,
@EmployeeID int,
@OrderDate datetime,
@TargetDate datetime,
@PayLater bit
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY Insert Into Orders (CustomerID, OrderDate, TargetDate,
```

```
EmployeeID, PayLater)
   VALUES (@CustomerID, @OrderDate, @TargetDate, @EmployeeID,
@PayLater);
   end TRY
   BEGIN CATCH
        DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać
zamówienia: ' + char(13) + char(10) + ERROR_MESSAGE();
   THROW 52000, @message, 1;
   end catch
end
go
```

15. AddOrderProduct - Dodaje Produkt Do Istniejącego zamówienia

```
CREATE PROCEDURE AddOrderProduct
@orderID int,
@entryID int,
@quantity int
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @orderDate datetime = (Select OrderDate from Orders where OrderID = @orderID)
   DECLARE @productID int = (Select ProductID from PriceHistory where EntryID =
@entryID);
   DECLARE @productCategory int = (Select CategoryID from Products where
ProductID=@productID);
   DECLARE @orderDay int = (Select DATEPART(weekday, @orderDate));
   DECLARE @collectDay int = (Select DATEPART(weekday, (SELECT TargetDate from ORders
Where OrderID=@orderID)));
   DECLARE @pickupWeek int = (SELECT DATEPART(Week, (SELECT TargetDate from Orders Where
OrderID=@orderID)));
   DECLARE @orderWeek int = (SELECT DATEPART(week, @orderDate));
   Declare @timeFrom date = (Select timeFrom from PriceHistory where EntryID=@entryID);
   DECLARE @timeTo date = (Select TimeTo from PriceHistory where EntryID=@entryID);
   BEGIN
       IF @productCategory = 5 and @collectDay NOT IN (5,6,7)
           BEGIN
               DECLARE @mess NVARCHAR(1000) = N'Owoce morza można zamówić tylko w
czwartki, piątki i soboty';
               THROW 52000,@mess,1;
           end
       IF @pickupWeek = @orderWeek and  @productCategory = 5
           BEGIN
               DECLARE @mess1 NVARCHAR(1000) = N'Owoce należy zamówić przed
poniedziałkiem poprzedzającym odbiór';
               THROW 52000,@mess1,1;
           end
       IF  @orderDate < @timeFrom or (@orderDate > @timeTo and @timeTo is not NULL)
           BeGIN
              DECLARE @mess2 NVARCHAR(1000) = N'Wybrany produkt nie jest dostępny!';
               THROW 52000,@mess2,1;
           end
```

```
    END
    BEGIN TRY
        DECLARE @Price money = (SELECT Price from PriceHistory where EntryID=@entryID);
        INSERT INTO OrderDetails(OrderID, EntryID, Quantity, Price)
        VALUES(@orderID,@entryID,@quantity,@Price)
    END TRY
    BEGIN CATCH
        DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać produktu do Zamówienia: '
+ char(13) + char(10) + ERROR_MESSAGE();
        THROW 52000, @message, 1;
    end catch
end
go
```

16. AddPrivateReservation - Tworzy rezerwację prywatną

```
Create PROCEDURE AddPrivateReservation
@customerID int,
@timeFrom datetime,
@timeTo datetime,
@orderID int
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @orderValue money = (Select Sum(Price) from OrderDetails where
OrderID=@orderID);
    Declare @minValue money = (SELECT ConstValue from ReservationRequirements
where ConstName like 'WZ')
    if not @customerID in (Select PrivateCustomerID from
CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS)
    BEGIN
        THROW 52000,'This customer ID: cant make private reservations', 1;
    end
    if @orderValue<@minValue BEGIN
        THROW 52000,'Order Value is too small!', 1;
    end
    BEGIN TRY
        Insert Into PrivateReservations (CustomerID, OrderID, TimeFrom, TimeTo)
        values (@customerID, @orderID, @timeFrom, @timeTo);
        DECLARE @reservationID int = @@IDENTITY;
        Insert Into PrivateStatusHistories(ReservationID, StatusID, TimeFrom,
TimeTo)
        values (@reservationID, 1, getdate(), null);
    END TRY
    BEGIN CATCH
        DECLARE @message NVARCHAR(1000) = N'Nie udało się zarejestrować
reserwacji!: ' + char(13) + char(10) + ERROR_MESSAGE();
        THROW 52000, @message, 1;
     END CATCH
end
go
```

17. AddPrivateReservationTable - Dodaje stolik do rezerwacji prywatnej

```
Create PROCEDURE AddPrivateReservationTable
@reservationID int,
@tableID int,
@guestsNumber int
as
BEGIN
   SET NOCOUNT ON;
   DECLARE @timeFrom datetime = (SELECT TimeFrom from
PrivateReservations where ReservationID=@reservationID);
   DECLARE @timeTo datetime = (SELECT TimeTo from PrivateReservations
where ReservationID=@reservationID);
   Declare @T Table (TableID int);
   Insert @T Exec FindAvailableTables @timeFrom, @timeTo;
   if @tableID not in (Select TableID from @T) BEGIN
       DECLARE @message varchar (1000) = 'Stolik o ID: ' +
Convert(Varchar, @tableID) + N' jest zajęty w podanym terminie';
       THROW 52000, @message, 1;
   end
   if (Select SeatsNo from Tables where TableID=@tableID) <
@guestsNumber BEGIN
   DECLARE @message1 varchar (1000) = 'Stolik o ID: ' + Convert(Varchar,
@tableID) + N' nie pomieści ' + CONVERT(varchar, @guestsNumber) + N'
gości!';
       THROW 52000, @message1, 1;
       END
   BEGIN TRY
       Insert Into PrivateReservationDetails(ReservationID, TableID,
GuestsNumber)
       values (@reservationID, @tableID, @guestsNumber)
   END TRY
   BEGIN CATCH
       DECLARE @message2 NVARCHAR(1000) = N'Nie udało się dodać stolika
do rezerwacji!: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message2, 1;
   end catch
end
go
```

18. AddCompanyReservation - Tworzy rezerwację firmową

```
Create PROCEDURE AddCompanyReservation
@customerID int,
@timeFrom datetime,
@timeTo datetime
AS
BEGIN
  SET NOCOUNT ON;
  if not @customerID in (Select  CompanyCustomerID from
CompanyCustomers)
  BEGIN
      THROW 52000,'This customer ID cant make company reservations', 1;
  end
  BEGIN TRY
      Insert Into CompanyReservations (CompanyID, TimeFrom, TimeTo)
      values (@customerID, @timeFrom, @timeTo);
      DECLARE @reservationID int = @@IDENTITY;
      Insert Into CompanyStatusHistories(ReservationID, StatusID,
TimeFrom, TimeTo)
      values (@reservationID, 1, getdate(), null);
  END TRY
  BEGIN CATCH
      DECLARE @message NVARCHAR(1000) = N'Nie udało się zarejestrować
firmowej reserwacji: ' + char(13) + char(10) + ERROR_MESSAGE();
      THROW 52000, @message, 1;
   END CATCH
end
go
```

19. AddCompanyReservationTable - Dodaje stolik do rezerwacji firmowej

```
Create PROCEDURE AddCompanyReservationTable
@reservationID int,
@tableID int,
@representantID int,
@guestsNumber int
as
BEGIN
   SET NOCOUNT ON;
   DECLARE @companyID int = (Select CompanyID from CompanyReservations
where ReservationID=@reservationID);
   DECLARE @timeFrom datetime = (SELECT TimeFrom from
CompanyReservations where ReservationID=@reservationID);
   DECLARE @timeTo datetime = (SELECT TimeTo from CompanyReservations
where ReservationID=@reservationID);
```

```
   Declare @T Table (TableID int);
   Insert @T Exec FindAvailableTables @timeFrom, @timeTo;
   if @tableID not in (Select TableID from @T) BEGIN
       DECLARE @message varchar (1000) = 'Stolik o ID: ' +
Convert(Varchar, @tableID) + N' jest zajęty w podanym terminie';
       THROW 52000, @message, 1;
   end
   if (Select SeatsNo from Tables where TableID=@tableID) <
@guestsNumber BEGIN
   DECLARE @message1 varchar (1000) = 'Stolik o ID: ' + Convert(Varchar,
@tableID) + N' nie pomieści ' + CONVERT(varchar, @guestsNumber) + N'
gości!';
       THROW 52000, @message1, 1;
   END
   IF (Select CompanyID from PrivateCustomers where
PrivateCustomerID=@representantID) != @companyID and @representantID !=
@companyID
   BEGIN
       DECLARE @message3 varchar (1000) = N'Użytkownik o ID: ' +
Convert(Varchar, @tableID) + N' nie jest reprezentatntem firmy ani samą
firmą';
       THROW 52000, @message3, 1;
   END
   BEGIN TRY
       Insert Into CompanyReservationDetails(ReservationID, TableID,
CustomerID, GuestsNumber)
       values (@reservationID, @tableID, @representantID, @guestsNumber)
   END TRY
   BEGIN CATCH
       DECLARE @message2 NVARCHAR(1000) = N'Nie udało się dodać stolika
do rezerwacji firmowej: ' + char(13) + char(10) + ERROR_MESSAGE();
       THROW 52000, @message2, 1;
   end catch
end
go
```

20. DisableParameter - wyłącza dany parametr (ustala ValidTo na aktualną datę)

```
CREATE procedure DisableParameter @entryID int
as
begin try
   update ParametersHistory set ValidTo = getdate() where EntryID = @entryID
   if (@@ROWCOUNT = 0)
       throw 52000, N'Niepoprawne EntryID', 1;
end try
begin catch
```

```
   DECLARE @message NVARCHAR(1000) = N'Nie udało się ustawić atrybuty parametru:
' + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch
go
```

21. EnableParameter - dodaje nową pozycję do ParametersHistory, gdzie podany parametr ma ValidFrom i ValidTo ustawione na podane w procedurze

```
CREATE procedure EnableParameter @parameterID int, @validFrom date, @validTo
date = null, @value int
as
begin try
   insert into ParametersHistory values (@validFrom, @validTo, @value,
@parameterID)
end try
begin catch
   DECLARE @message NVARCHAR(1000) = N'Nie udało się ustawić zakresu dotępości
parametru: ' + ERROR_MESSAGE();
       THROW 52000, @message, 1;
end catch
go
```

22. AddParameter - Dodaje nową pozycję do parameters

```
create procedure AddParameter @parametername nvarchar(50),
                              @unit nchar(10)
as
begin
 begin try
      insert into DiscountsParameters(ParameterName, Unit) values
(@parametername, @unit)
  end try
 begin catch
     DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać parametru: '
+ ERROR_MESSAGE();
     THROW 52000, @message, 1;
 end catch
end
go
```

23. AddCategory - Dodaje kategorię produktu

```
create procedure AddCategory @categoryname nvarchar(50)
as
begin
  begin try
       insert into Categories(CategoryName) values (@categoryname)
   end try
  begin catch
      DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać kategorii:
' + ERROR_MESSAGE();
      THROW 52000, @message, 1;
  end catch
end
go
```

24. SetCompany - Ustala ID firmy dla klienta indywidualnego

```
create procedure SetCompany @companyid int, @privatecustomerid int
as
begin
 begin try
     update PrivateCustomers set CompanyID = @companyid where
PrivateCustomerID=@privatecustomerid
  end try
 begin catch
     DECLARE @message NVARCHAR(1000) = N'Nie udało się przypisać firmy do
klienta: ' + ERROR_MESSAGE();
     THROW 52000, @message, 1;
 end catch
end
go
```

25. AddTable - Dodaje nowy stolik

```
create procedure AddTable @seatsno int
as
begin
  begin try
       insert into Tables(SeatsNo) values (@seatsno)
   end try
  begin catch
      DECLARE @message NVARCHAR(1000) = N'Nie udało się dodać nowego
```

```
stolika: ' + ERROR_MESSAGE();
      THROW 52000, @message, 1;
  end catch
end
go
```

26. FindAvailableTables - Znajduje Stoliki dostępne do rezerwacji w podanym przedziale czasowym

```
CREATE PROCEDURE FindAvailableTables
@timeFrom datetime,
@timeTo datetime
AS
SELECT TableID from Tables
where TableID not in (
   Select TableID from PrivateReservations PR
   inner join PrivateReservationDetails PRD on PR.ReservationID =
PRD.ReservationID
   where PR.TimeTo > @timeFrom and PR.TimeTo < @timeTo  or PR.TimeFrom <
@timeTo and PR.TimeTo > @timeTo
   UNION
   Select TableID from CompanyReservations PR
   inner join CompanyReservationDetails PRD on PR.ReservationID =
PRD.ReservationID
   where PR.TimeTo > @timeFrom and PR.TimeTo < @timeTo  or PR.TimeFrom <
@timeTo and PR.TimeTo > @timeTo
   )
go
```

27. CheckMenuUpdate - Sprawdza czy menu wymaga aktualizacji, jeśli tak to ile pozycji wymaga zmiany

```
CREATE PROCEDURE CheckMenuUpdate
as begin
   declare @entriesNo float = (Select Count(*) from CURRENT_MENU);
   declare @updatedEntriesNo float = (Select Count(*) from CURRENT_MENU
where DATEDIFF(day, TimeFrom, getdate()) < 14);
   if (@updatedEntriesNo/@entriesNo > 0.5)
       BEGIN
           Select  'Menu Nie Wymaga Aktualizacji';
       END
   else
       BEGIN
           Select 'Przynajmniej ' +CONVERT(varchar,
ROUND(@entriesNo*(0.5) - @updatedEntriesNo, 0)) + ' pozycji/e wymaga
aktualizacji';
```

```
        END
end
go
```

28. AddPrivateReservationWithOrder - Dodaje rezerwację prywatną i od razu kładzie zamówienie - Zawiera w sobie transakcję

```
CREATE PROCEDURE AddPrivateReservationWithOrder
@CustomerID int,
@timeFrom datetime,
@timeTo datetime,
@TableID int,
@guestsNumber int,
@employeeID int
as
BEGIN
   BEGIN TRY
       BEGIN TRAN
           declare @currentTime datetime = (Select getdate());
           EXECUTE AddOrder @CustomerID, @employeeID, @currentTime,
@timeFrom, True;
           declare @orderID int = @@IDENTITY;
           EXECUTE AddPrivateReservation @CustomerID, @timeFrom,
@timeTo, @orderID;
           declare @reservationID int = (Select top 1 ReservationID from
PrivateReservations order by ReservationID desc);
           EXECUTE AddPrivateReservationTable @reservationID, @tableID,
@guestsNumber;
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       declare @message Varchar(1000) = N'Nie udało się dodać rezerwacji
z zamówieniem: ' + ERROR_MESSAGE();
       IF(@@TRANCOUNT > 0)
           ROLLBACK TRAN;
       THROW 52000, @message, 1;
   END CATCH
end
```

29. GrantDiscounts - Dla podanego klienta i zamówienia sprawdza czy dodanie tego zamówienia pozwoliło mu osiągnąc którąś ze zniżek, jeżeli tak, to ją otrzymuje.

```
Create PROCEDURE GrantDiscounts
   @CustomerID int,
   @orderID int
as
Begin
   declare @constZ int = (Select top 1 ParameterValue from ParametersHistory
```

```
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName LIKE 'ConstZ%' and ValidFrom <= getdate() and
(ValidTo >= getdate() or ValidTo is NULL));
   declare @constK int = (Select top 1 ParameterValue from ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName LIKE 'ConstK%' and ValidFrom <= getdate() and
(ValidTo >= getdate() or ValidTo is NULL));
   declare @constID int = (Select top 1 ParametersHistory.ParameterID from
ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName LIKE 'ConstR%' and ValidFrom <= getdate() and
(ValidTo >= getdate() or ValidTo is NULL));

   declare @timedK int = (Select top 1 ParameterValue from ParametersHistory
       inner join DiscountsParameters DP on DP.ParameterID =
ParametersHistory.ParameterID
       where DP.ParameterName LIKE 'timedK%' and ValidFrom <= getdate() and
(ValidTo >= getdate() or ValidTo is NULL));
   declare @hasGrantedTimed bit = 0;
   declare @hasGrantedConst bit = 0;
   if (@CustomerID in (Select CustomerID from GrantedTimedDiscounts where
CustomerID=@CustomerID and DiscountID=@constID and ActiveTo < getdate()))
       begin
           set @hasGrantedTimed = 1;
       end
   if (@CustomerID in (Select CustomerID from GrantedOneTimeDiscounts where
CustomerID=@CustomerID))
       begin
           set @hasGrantedConst = 1;
       end

   declare @orderCount int = (Select Count(OrderID) from ORDERS_WORKHORSE where
CustomerID=@CustomerID HAVING SUM(Quantity*Price)>@constK)
   if (@orderCount >= @constZ and @hasGrantedConst=0) BEGIN
       Execute GrantLifetimeDiscount @CustomerID;
   end
   declare @orderValue money = (Select SUM(Price*Quantity) from ORDERS_WORKHORSE
where CustomerID=@CustomerID and OrderID=@orderID);
   if (@orderValue > @timedK and @hasGrantedConst=0)BEGIN
       Execute GrantTimedDiscount @CustomerID;
   end
end
```

Funkcje

1. GetParameterValue - Zwraca aktualną wartość parametru po nazwie

```
create or alter function GetParameterValue(@par_name nvarchar(50))
   returns TABLE as
   return (select top 1 ParameterValue
           from ParametersHistory PH
                    inner join DiscountsParameters DP on DP.ParameterID =
PH.ParameterID
       where ParameterName = @par_name order by ValidFrom desc)
```

2. IsTableAvailableOn - Podaje czy podany stolik jest dosępny w danym przedziale czasowym (True/False)

```
CREATE Function IsTableAvailableOn(
@tableID int,
@timeFrom datetime,
@timeTo datetime
)
RETURNS bit AS
   BEGIN
       declare @occupiedTable int = (SELECT TOP 1 TableID from Tables
           where Tables.TableID not in (
               Select TableID from PrivateReservations PR
               inner join PrivateReservationDetails PRD on
PR.ReservationID = PRD.ReservationID
               where PR.TimeTo >= @timeFrom and PR.TimeTo <= @timeTo  or
PR.TimeFrom <= @timeTo and PR.TimeTo >= @timeTo
               UNION
               Select TableID from CompanyReservations PR
               inner join CompanyReservationDetails PRD on
PR.ReservationID = PRD.ReservationID
               where PR.TimeTo >= @timeFrom and PR.TimeTo <= @timeTo  or
PR.TimeFrom <= @timeTo and PR.TimeTo >= @timeTo
               ) and Tables.TableID = @tableID)
       declare @result bit = 0;
       if (@occupiedTable is not null)BEGIN
           set @result = 1;
       END
       RETURN @result
       END
BEGIN
```

3. GetMaxOrderValue - Zwraca Najbardziej Wartościowe Zamówienie Klienta bez uwzględnia zniżek

```
CREATE FUNCTION GetMaxOrderValue(
@customerID int
) RETURNS money as
   Begin
       declare @res money = (Select top 1 SUM(Price*Quantity) from
ORDERS_WORKHORSE where CustomerID=@customerID group by OrderID order by
1 desc );
       return @res;
   end
go
```

4. GetOrderCount - Zwraca Ilośc zamówien dla klienta

```
CREATE FUNCTION GetOrderCount(
@customerID int
) RETURNS int as
   Begin
       declare @res int = (Select count( Distinct OrderID) from
ORDERS_WORKHORSE where CustomerID=@customerID);
       return @res;
   end
```

5. ReservationStatusCompany - Zwraca aktualny status podanej rezerwacji firmowej

```
create or alter function ReservationStatusCompany(@reservationID int)
   returns table as
       return
           (
               select top 1 StatusName
               from CompanyStatusHistories CSH
                        inner join ReservationStatuses RS on CSH.StatusID =
RS.StatusID
               where ReservationID = @reservationID
               order by TimeFrom
           )
```

6. ReservationStatusPrivate - zwraca aktualny stan podanej rezerwacji klienta indywidualnego

```
create or alter function ReservationStatusCompany(@reservationID int)
   returns table as
       return
           (
               select top 1 StatusName
               from CompanyStatusHistories CSH
                        inner join ReservationStatuses RS on CSH.StatusID =
RS.StatusID
```

```
            where ReservationID = @reservationID
            order by TimeFrom
        )
```

7. CheckMenuUpdate - funkcja sprawdzająca czy menu wymaga zmian

```
create function CheckMenuUpdate()
returns varchar(max)
as begin
   declare @entriesNo float = (Select Count(*) from CURRENT_MENU);
   declare @updatedEntriesNo float = (Select Count(*) from CURRENT_MENU where
DATEDIFF(day, TimeFrom, getdate()) < 14);
   declare @result varchar(max) = '';
   if (@updatedEntriesNo/@entriesNo > 0.5)
       BEGIN
           set @result =  'Menu Nie Wymaga Aktualizacji';
       END
   else
       BEGIN
           set @result = 'Przynajmniej ' +CONVERT(varchar,
ROUND(@entriesNo*(0.5) - @updatedEntriesNo, 0)) + ' pozycji/e wymaga
aktualizacji';
       END
   return @result;
end
```

Triggery

1. GrantDiscount - Sprawdza czy klient może dostać zniżke i automatycznie przyznaje

```
Create TRIGGER GrantDiscountsTrigger On OrderDetails
AFTER INSERT
AS
Begin
   declare @orderID int = (Select top 1 OrderID from inserted );
   declare @customerID int = (Select top 1 CustomerID from Orders where
Orders.OrderID=@orderID);

   Execute GrantDiscounts @customerID, @orderID;
end
```

2. Usunięcie zamowienia - Jeśli zamówienie pomimo constraintów zostnie usunięte z Orders to trigger usuwa je z OrderDetails w celu zachowania spójności bazy

```
Create TRIGGER OnRemoveOrderTrigger on Orders
   FOR DELETE
AS
Begin
   declare @orderID int  = (Select OrderID from deleted);
   delete from OrderDetails where OrderID=@orderID;
END
```

Indeksy

Jeśli nie wskazano inaczej indeksy są nieklastrowane

1. Indeks na identyfikator klienta znajdujący się w tabeli Orders

```
CREATE NONCLUSTERED INDEX CustomerIDIndex on Orders(CustomerID)
```

2. Indeks na identyfikator klienta prywatnego znajdujący się w tabeli PrivateCustomers

```
CREATE NONCLUSTERED INDEX CustomerIDIndex on
   PrivateCustomers(PrivateCustomerID)
```

3. Indeks na identyfikator klienta znajdujący się w tabeli CompanyReservationDetails

```
CREATE NONCLUSTERED INDEX CustomerIDIndex on
   CompanyReservationDetails(CustomerID)
```

4. Indeks na identyfikator klienta znajdujący się w tabeli GrantedOneTimeDiscounts

```
CREATE NONCLUSTERED INDEX CustomerIDIndex on
   GrantedOneTimeDiscounts(CustomerID)
```

5. Indeks na identyfikator klienta znajdujący się w tabeli CustomersWithNoDiscountAccess

```
CREATE NONCLUSTERED INDEX CustomerIDIndex on
   CustomersWithNoDiscountAccess(CustomerID)
```

6. Indeks na identyfikator zamówienia znajdujący się w tabeli OrderDetails

```
   CREATE NONCLUSTERED INDEX OrderIDIndex on OrderDetails(OrderID)
```

7. Indeks na identyfikator zamówienia znajdujący się w tabeli PrivateReservations

```
CREATE NONCLUSTERED INDEX OrderIDIndex on
   PrivateReservations(OrderID)
```

8. Indeks na identyfikator pozycji z menu znajdujący się w tabeli OrderDetails

```
CREATE NONCLUSTERED INDEX EntryIDIndex on OrderDetails(EntryID)
```

9. Indeks na identyfikator klienta firmowego znajdujący się w tabeli

CompanyReservations

```
CREATE NONCLUSTERED INDEX CompanyIDIndex on
   CompanyReservations(CompanyID)
```

10. Indeks na identyfikator pracownika znajdujący się w tabeli Orders

```
CREATE NONCLUSTERED INDEX EmployeeIDIndex on Orders(EmployeeID)
```

11. Indeks na identyfikator pracownika znajdujący się w tabeli Administrators

```
   CREATE NONCLUSTERED INDEX EmployeeIDIndex on
   Administrators(EmployeeID)
```

12. Indeks na identyfikator produktu znajdujący się w tabeli PriceHistory

```
CREATE NONCLUSTERED INDEX ProductIDIndex on PriceHistory(ProductID)
```

13. Indeks na identyfikator rezerwacji znajdujący się w tabeli PrivateReservations

```
CREATE NONCLUSTERED INDEX ReservationIDIndex on
   PrivateReservations(ReservationID)
```

14. Indeks na identyfikator rezerwacji znajdujący się w tabeli PrivateStatusHistories

```
CREATE NONCLUSTERED INDEX ReservationIDIndex on
   PrivateStatusHistories(ReservationID)
```

15. Indeks na identyfikator rezerwacji znajdujący się w tabeli CompanyStatusHistories

```
CREATE NONCLUSTERED INDEX ReservationIDIndex on
   CompanyStatusHistories(ReservationID)
```

16. Indeks na identyfikator rezerwacji znajdujący się w tabeli CompanyReservationDetails

```
CREATE NONCLUSTERED INDEX ReservationIDIndex on
   CompanyReservationDetails(ReservationID)
```

17. Indeks na identyfikator statusu znajdujący się w tabeli PrivateStatusHistories

```
CREATE NONCLUSTERED INDEX StatusIDIndex on
   PrivateStatusHistories(StatusID)
```

18. Indeks na identyfikator statusu znajdujący się w tabeli CompanyStatusHistories

```
CREATE NONCLUSTERED INDEX StatusIDIndex on
   CompanyStatusHistories(StatusID)
```

19. Indeks na imię i nazwisko znajdujące się w tabeli PrivateCustomers

```
CREATE NONCLUSTERED INDEX fullNameIndex on
   PrivateCustomers(Fristname, lastname)
```

20. Indeks na imię i nazwisko znajdujące się w tabeli Employees

```
CREATE NONCLUSTERED INDEX fullNameIndex on Employees(Fristname,
   lastname)
```

21. Indeks na nazwę produktu znajdującą się w tabeli Products

```
CREATE NONCLUSTERED INDEX productNameIndex on Products(ProductName)
```

22. Indeks na identyfikator stolika znajdujący się w tabeli PrivateReservationDetails

```
CREATE NONCLUSTERED INDEX tableIDIndex on
   PrivateReservationDetails(TableID)
```

23. Indeks na identyfikator stolika znajdujący się w tabeli

```
CREATE NONCLUSTERED INDEX tableIDIndex on
   CompanyReservationDetails(TableID)
```

24. Indeks na datę złożenia zamówienia znajdującą się w tabeli Orders

```
CREATE NONCLUSTERED INDEX orderDateIDIndex on Orders(OrderDate)
```

Uprawnienia

1. **Administrator**
    a. Wszystkie Uprawnienia

```
create role administrator authorization dbo
grant ALL to administrator
```

2. **PrivateCustomer**
    a. Uprawnienia dostępu do widoków
        i. AWAINTING_APPROVAL
        ii. CURRENT_PRIVATE_RESERVATIONS
        iii. CURRENT_DISCOUNTS
        iv. CURRENT_MENU
        v. CUSTOMERS THAT CAN MAKE RESERVATIONS
        vi. ORDER_VALUES_REPORT_PRIVATE
        vii. ORDER_VALUES_PRIVATE
        viii. PLANNED_MENU_CHANGES
        ix. PRIVATE_CUSTOMER_VALUES_REPORT_MONTH
        x. PRIVATE_CUSTOMER_VALUES_REPORT_WEEK
        xi.
    b. Uprawnienia dostępu do Procedur i Funkcji
        i. AddOrder
        ii. AddOrderProduct
        iii. AddPrivateReservationWithOrder
        iv. IsTableAvailableOn

```
create role private_customer authorization dbo
grant select on AWAITING_APPROVAL to private_customer
grant select on CURRENT_PRIVATE_RESERVATIONS to private_customer
grant select on CURRENT_DISCOUNTS to private_customer
grant select on CURRENT_MENU to private_customer
grant select on CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS to private_customer
grant select on ORDER_VALUES_REPORT_PRIVATE to private_customer
grant select on ORDER_VALUES_PRIVATE to private_customer
grant select on PLANNED_MENU_CHANGES to private_customer
grant select on PRIVATE_CUSTOMER_VALUES_REPORT_MONTH to private_customer
grant select on PRIVATE_CUSTOMER_VALUES_REPORT_WEEK to private_customer
grant execute on AddOrder to private_customer
grant execute on AddOrderProduct to private_customer
grant execute on AddPrivateReservationWithOrder to private_customer
grant execute on IsTableAvailableOn to private_customer
```

3. **CompanyCusomer**
    a. Uprawnienia dostępu do widoków
        i. AWAINTING_APPROVAL
        ii. CURRENT_COMPANY_RESERVATIONS
        iii. CURRENT_DISCOUNTS
        iv. CURRENT_MENU
        v. CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS
        vi. ORDER_VALUES_REPORT_COMPANY
        vii. ORDER_VALUES_COMPANY
        viii. PLANNED_MENU_CHANGES
        ix. COMPANY_CUSTOMER_VALUES_REPORT_MONTH
        x. COMPANY_CUSTOMER_VALUES_REPORT_WEEK
    b. Uprawnienia dostępu do Procedur i Funkcji
        i. AddOrder
        ii. AddOrderProduct
        iii. AddCompanyReservation
        iv. IsTableAvailableOn
        v. SetCompany

```
create role company_customer authorization dbo
grant select on AWAITING_APPROVAL to company_customer
grant select on CURRENT_COMPANY_RESERVATIONS to company_customer
grant select on CURRENT_DISCOUNTS to company_customer
grant select on CURRENT_MENU to company_customer
grant select on CUSTOMERS_THAT_CAN_MAKE_RESERVATIONS to company_customer
grant select on ORDER_VALUES_REPORT_COMPANY to company_customer
grant select on ORDER_VALUES_COMPANY to company_customer
grant select on PLANNED_MENU_CHANGES to company_customer
grant select on COMPANY_CUSTOMER_VALUES_REPORT_MONTH to company_customer
grant select on COMPANY_CUSTOMER_VALUES_REPORT_WEEK to company_customer
grant execute on AddOrder to company_customer
grant execute on AddOrderProduct to company_customer
grant execute on AddCompanyReservation to company_customer
grant execute on IsTableAvailableOn to company_customer
```

4. **Employee**
    a. Uprawnienia dziedziczone
        i. Wszystkie uprawnienia Private i Company customer
    b. Uprawnienia dostępu do widoków
        i. PRIVATE RESERVATIONS
        ii. COMPANY RESERVATIONS
        iii. MENU_HISTORY
        iv. ORDER_VALUES
        v. ORDER_VALUES_PRIVATE
        vi. ORDER_VALUES_COMPANY
        vii. PARAMETERS_VIEW
    c. Uprawnienia dostępu do Procedur i Funkcji
        i. AddPrivateCustomer
        ii. AddCompanyCustomer

iii. AddPrivateReservationTable
iv. AddCompanyReservationTable
v. AddTable
vi. ChangeCompanyReservationStatus
vii. ChangePrivateReservationStatus
viii. CheckMenuUpdate
ix. FindAvailableTables
x. GetMaxOrderValue
xi. GetParameterValue
xii. AddCountry
xiii. AddCity
xiv. GrantLifetimeDiscount
xv. GrantTimedDiscount
xvi. GrantDiscounts

```
create role employee authorization dbo
grant private_customer to employee
grant company_customer to employee
grant select on PRIVATE_RESERVATIONS to employee
grant select on COMPANY_RESERVATIONS to employee
grant select on MENU_HISTORY to employee
grant select on ORDER_VALUES to employee
grant select on ORDER_VALUES_PRIVATE to employee
grant select on ORDER_VALUES_COMPANY to employee
grant select on PARAMETERS_VIEW to employee

grant execute on AddPrivateCustomer to employee
grant execute on AddCompanyCustomer to employee
grant execute on AddPrivateReservationTable to employee
grant execute on AddCompanyReservationTable to employee
grant execute on AddTable to employee
grant execute on ChangeCompanyReservationStatus to employee
grant execute on ChangePrivateReservationStatus to employee
grant execute on CheckMenuUpdate to employee
grant execute on FindAvailableTables to employee
grant execute on GetMaxOrderValue to employee
grant select on GetParameterValue to employee
grant execute on AddCountry to employee
grant execute on AddCity to employee
grant execute on GrantLifetimeDiscount to employee
grant execute on GrantTimedDiscount to employee
grant execute on GrantDiscounts to employee
go
```

5. Manager
    a. Uprawnienia dziedziczone
        i. Wszystkie uprawnienia Employee
    b. Uprawnienia dostępu do widoków
        i. COMPANY_RESERVATIONS_REPORT_MONTH
        ii. COMPANY_RESERVATIONS_REPORT_MONTH
        iii. ONE_TIME_DISCOUNTS_REPORT_MONTH
        iv. ONE_TIME_DISCOUNTS_REPORT_WEEK
        v. ORDER_VALUES_MONTH_REPORT
        vi. OREDER_VALUES_MONTH_REPORT_COMPANY
        vii. OREDER_VALUES_MONTH_REPORT_PRIVATE
        viii. PRODUCTS_ORDERED_REPORT_MONTH
        ix. PRODUCTS_ORDERS_REPORT_WEEK
        x. PRODUCT_VALUES_MONTH
        xi. PRODUCT_BALUES_WEEK
        xii. TIMED_DISCOUNTS_REPORT_WEEK
        xiii. TIMED_DISCOUNTS_REPORT_MONTH
    c. Uprawnienia dostępu do Procedur i Funkcji
        i. AddEmployee
        ii. AddProduct
        iii. RemoveFromMenu
        iv. AddToMenu

```
create role manager authorization dbo
grant employee to manager
grant select on COMPANY_RESERVATIONS_REPORT_MONTH to manager
grant select on COMPANY_RESERVATIONS_REPORT_MONTH to manager
grant select on ONE_TIME_DISCOUNTS_REPORT_MONTH to manager
grant select on ONE_TIME_DISCOUNTS_REPORT_WEEK to manager
grant select on ORDER_VALUES_MONTH_REPORT to manager
grant select on ORDER_VALUES_MONTH_REPORT_COMPANY to manager
grant select on ORDER_VALUES_MONTH_REPORT_PRIVATE to manager
grant select on PRODUCTS_ORDERED_REPORT_MONTH to manager
grant select on PRODUCTS_ORDERED_REPORT_WEEK to manager
grant select on PRODUCTS_VALUES_MONTH to manager
grant select on PRODUCTS_VALUES_MONTH to manager
grant select on TIMED_DISCOUNTS_REPORT_WEEK to manager
grant select on TIMED_DISCOUNTS_REPORT_MONTH to manager
grant execute on AddEmployee to manager
grant execute on AddProduct to manager
grant execute on RemoveFromMenu to manager
grant execute on AddToMenu to manager
```

6. business_owner
    a. Wszystkie uprawnienia

```
create role business_owner authorization dbo
grant ALL to business_owner
```