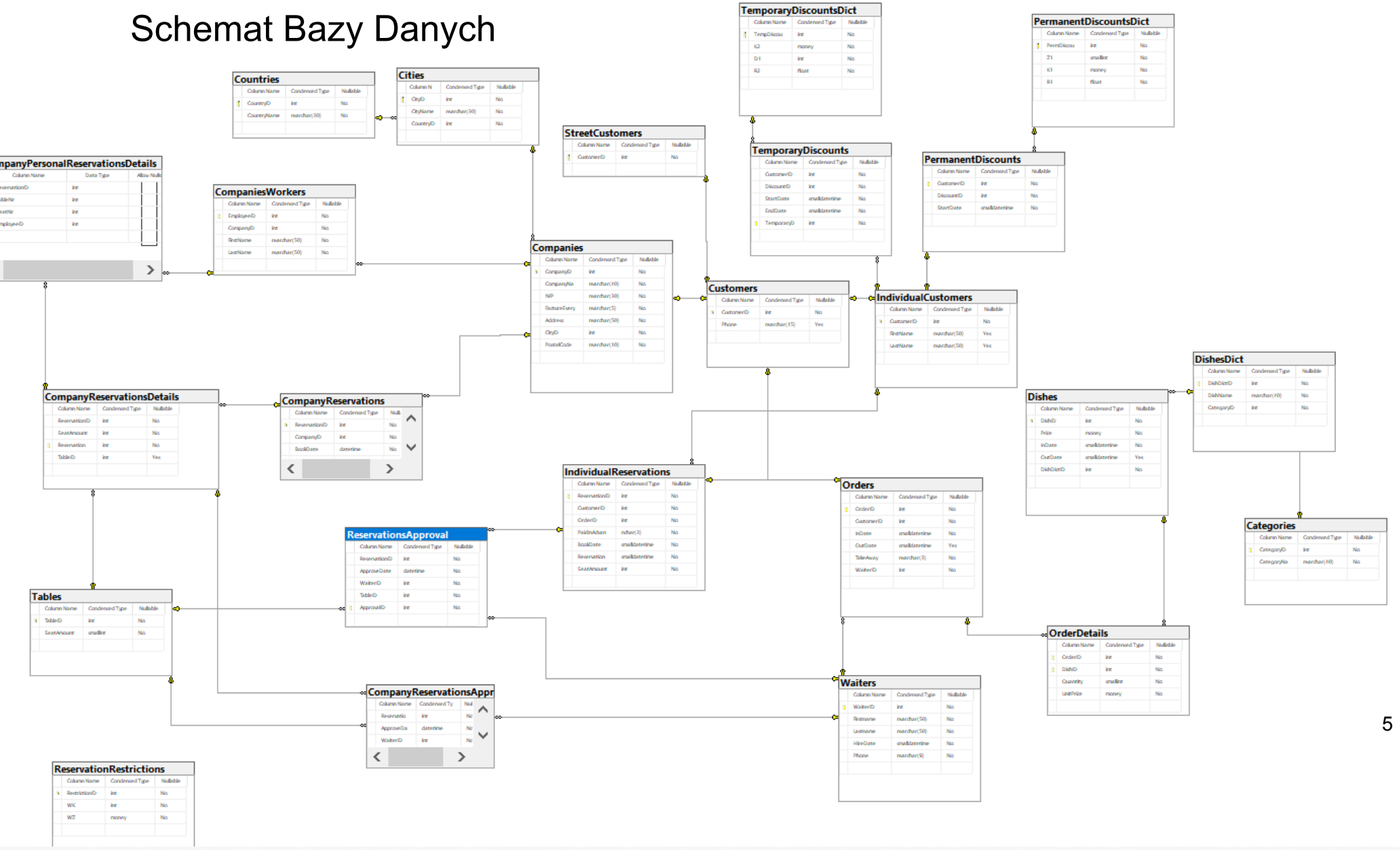
Marcel Spryszyński
Kamil Błażewicz

Spis Treści

Marcel Spryszyński
Kamil Błażewicz

Schemat Bazy Danych



Funkcje użytkowników systemu

1. Klient indywidualny
    a. składanie zamówienia na wynos (złożone na miejscu)
    b. składanie zamówienia na wynos (złożone przez formularz)
    c. składanie zamówienia na owoce morza
    d. składanie rezerwacji stolika z zamówieniem
    e. składanie zamówienia na miejscu (gdy pozostały wolne miejsca)
    f. anulowanie/edycja zamówienia (nie można anulować zamówień na owoce morza, anulowanie/edycja możliwa jest 24h przed umówioną godziną)

2. Firma
    a. wszystkie funkcje dostępne dla klienta indywidualnego
    b. możliwość zamówienia na fakturę zbiorczą (raz w miesiącu)
    c. możliwość zamówienia na fakturę jednorazową
    d. możliwość rezerwacji kilku stolików

3. Kelner
    a. wystawianie faktury przy zamówieniu na miejscu
    b. tworzenie zamówienia składanego na miejscu
    c. anulowanie rezerwacji stolika
    d. oznaczenie zamówienia jako wykonane
    e. usuwanie niesfinalizowanych zamówień
    f. zatwierdzanie stolików

4. Manager restauracji
    a. Wszystkie funkcje dostępne dla kelnera
    b. edytowanie menu
    c. aktualizacja stanu produktów
    d. wystawianie faktur dla zamówień składanych poprzez formularz
    e. generowanie raportów miesięcznych i tygodniowych
    f. generowanie raportów dotyczących zamówień oraz rabatów dla klientów indywidualnych oraz firm

5. Administrator systemu
    a. aktualizacja bazy klientów oraz pracowników
    b. zmiana wartości/progów rabatów
    c. tworzenie backupu bazy danych
    d. zmiana obostrzeń

6. System
    a. aktualizacja aktywnych rabatów
    b. przypomnienia o wymaganej zmianie menu
    c. rozplanowuje stoliki możliwe do użycia
    d. przechowywanie stanu płatności

Marcel Spryszyński
Kamil Błażewicz

Opisy tabel w bazie danych

1.Categories

Słownik kategorii dań

- a.**PK** CategoryID (int) - ID kategorii NOT NULL
- b.CategoryName(nvarchar(40))- nazwa kategorii NOT NULL

Warunki integralności:

- CategoryName nie zawiera cyfr CHECK ((NOT [CategoryName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[Categories]
(
    [CategoryID] [int] NOT NULL,
    [CategoryName] [nvarchar](40) NOT NULL,
    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],
    UNIQUE NONCLUSTERED
(
    [CategoryName] 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]
ALTER TABLE [dbo].[Categories]  WITH CHECK ADD  CONSTRAINT [CK_Categories_1] CHECK  ((NOT [CategoryName] like '%[0-9]%'))
```

2. Cities

Słownik kategorii miast

- a.**PK** CityID (int) - ID kategorii NOT NULL
- b.CityName(nvarchar(30))- nazwa miasta NOT NULL
- c.**FK** CountryID - ID państwa NOT NULL

Warunki integralności:

- CityName nie zawiera cyfr CHECK ((NOT [CityName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[Cities]
(
    [CityID] [int] NOT NULL,
    [CityName] [nvarchar](30) NOT NULL,
    [CountryID] [int] NOT NULL,
    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]
ALTER TABLE [dbo].[Cities]  WITH CHECK ADD FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT [CK_City_1] CHECK  ((NOT [CityName] like '%[0-9]%'))
```

3.Companies

Tabela zawierająca podstawowe dane o zewnętrznej firmie

- **PK FK** CompanyID (int) - ID firmy NOT NULL
- CompanyName (nvarchar(40) - nazwa firmy NOT NULL
- NIP (int) - numer Nip firmy NOT NULL
- FactureEvery - (bit)) - czy rachunki są na fakturę raz na miesiąc NOT NULL
- Address (nvarchar(50)) - adres firmy NOT NULL
- **FK** CityID - ID miasta NOT NULL
- PostalCode - kod pocztowy miasta NOT NULL

Warunki integralności:

- numer NIP jest unikalny oraz składa się z cyfr UNIQUE,
  CHECK(NIP like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
- CompanyName nie zawiera cyfr CHECK ((NOT [CompanyName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[Companies]
(
    [CompanyID] [int] NOT NULL,
    [CompanyName] [nvarchar](40) NOT NULL,
    [NIP] [nvarchar](30) NOT NULL,
    [FactureEvery] [nvarchar](5) NOT NULL,
    [Address] [nvarchar](50) NOT NULL,
    [CityID] [int] NOT NULL,
    [PostalCode] [nvarchar](10) NOT NULL,
    CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
    [CompanyID] 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],
    CONSTRAINT [U_NIP] UNIQUE NONCLUSTERED
(
    [NIP] 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]
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK__Companies__CityI__0D44F85C] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK__Companies__CityI__0D44F85C]
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK__Companies__Custo__0C50D423] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Customers] ([CustomerID])
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK__Companies__Custo__0C50D423]
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [check_nip] CHECK  (([NIP]>(0)))
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [check_nip]
ALTER TABLE dbo.Companies ADD CONSTRAINT U_NIP UNIQUE NONCLUSTERED (NIP)
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [CK_CompanyName_1] CHECK  ((NOT [CompanyName] like '%[0-9]%'))
```

4.CompaniesWorkers

Tabela zawierająca dane o pracownikach zewnętrznej firmy

- **PK** EmployeeID(int) - ID pracownika zewnętrznej firmy NOT NULL
- **FK** CompanyID (int) - ID firmy NOT NULL
- FirstName(nvarchar(50) - imię pracownika firmy NOT NULL
- LastName (nvarchar(50) - nazwisko pracownika firmy NOT NULL

Warunki integralności:

- FirstName, LastName nie zawierają cyfr

  CHECK ((NOT [FirstName] like '%[0-9]%')),
  CHECK ((NOT [LastName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[CompaniesWorkers]
(
    [EmployeeID] [int] NOT NULL,
    [CompanyID] [int] NOT NULL,
    [FirstName] [nvarchar](50) NOT NULL,
    [LastName] [nvarchar](50) NOT NULL,
    CONSTRAINT [PK_CompaniesWorkers_1] 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]
ALTER TABLE [dbo].[CompaniesWorkers]  WITH CHECK ADD  CONSTRAINT [FK_CompaniesWorkers_Companies1] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CompanyID])
ALTER TABLE [dbo].[CompaniesWorkers] CHECK CONSTRAINT [FK_CompaniesWorkers_Companies1]
ALTER TABLE [dbo].[CompaniesWorkers]  WITH CHECK ADD
CONSTRAINT [CK_CompaniesWorkersFirstName_1] CHECK  ((NOT [FirstName] like '%[0-9]%'))
ALTER TABLE [dbo].[CompaniesWorkers]  WITH CHECK ADD
CONSTRAINT [CK_CompaniesWorkersLastName_1] CHECK  ((NOT [LastName] like '%[0-9]%'))
```

5. CompanyPersonalReservationDetail

Tabela zawierająca dane o rezerwacji firmowej na pracownika firmy

- **PK FK** ReservationID (int) - ID rezerwacji NOT NULL
- **PK FK** EmployeeID (int) - ID pracownika firmy NOT NULL
- **PK FK** TableID (int) - ID stolika przy którym będzie siedział konkretny pracownik

```
CREATE TABLE [dbo].[CompanyPersonalReservationsDetails]
(
    [ReservationID] [int] NOT NULL,
    [TableNr] [int] NOT NULL,
    [SeatNr] [int] NOT NULL,
    [EmployeeID] [int] NOT NULL,
    CONSTRAINT [PK_CompanyPersonalReservationsDetails] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [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]

ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]  WITH CHECK
ADD  CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompaniesWorkers] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[CompaniesWorkers] ([EmployeeID])
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails] CHECK CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompaniesWorkers]
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]  WITH CHECK
ADD  CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompanyReservationsDetails] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservationsDetails] ([ReservationDeatilsID])
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]
CHECK CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompanyReservationsDetails]
```

6. CompanyReservationDetails

Tabela pozwalająca na przypisanie konkretnych stolików dla naszej firmowej rezerwacji bez podawania pracowników

- **PK FK** ReservationID (int) - ID rezerwacji NOT NULL
- **PK FK** TableID (int) - ID zarezerwowanego przez firmę stolika NOT NULL

```
CREATE TABLE [dbo].[CompanyPersonalReservationsDetails]
(
    [ReservationID] [int] NOT NULL,
    [TableNr] [int] NOT NULL,
    [SeatNr] [int] NOT NULL,
    [EmployeeID] [int] NOT NULL,
    CONSTRAINT [PK_CompanyPersonalReservationsDetails] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [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]

ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]  WITH CHECK
ADD  CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompaniesWorkers] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[CompaniesWorkers] ([EmployeeID])
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails] CHECK CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompaniesWorkers]
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]  WITH CHECK
ADD  CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompanyReservationsDetails] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservationsDetails] ([ReservationDeatilsID])
ALTER TABLE [dbo].[CompanyPersonalReservationsDetails]
CHECK CONSTRAINT [FK_CompanyPersonalReservationsDetails_CompanyReservationsDetails]
```

7.CompanyReservations

Tabela przypisująca firmie konkretną rezerwację

- **PK FK** ReservationID (int) - ID rezerwacji NOT NULL
- **FK** CompanyID (int) - ID zewnętrznej firmy NOT NULL

```
CREATE TABLE [dbo].[CompanyReservations]
(
    [ReservationID] [int] NOT NULL,
    [CompanyID] [int] NOT NULL,
    [BookDate] [datetime] NOT NULL,
    [ReservationDate] [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]
ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT [FK_CompanyReservations_Companies] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CompanyID])
ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [FK_CompanyReservations_Companies]
```

8.Countries

Słownik państw

- a.**PK** CountryID (int) - ID państwa NOT NULL
- b.CountryName(nvarchar(30))- nazwa państwa NOT NULL

Warunki Integralności:

- CountryName nie zawiera cyfr CHECK ((NOT [CountryName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[Countries]
(
    [CountryID] [int] NOT NULL,
    [CountryName] [nvarchar](30) NOT NULL,
    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],
    UNIQUE NONCLUSTERED
(
    [CountryName] 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]
ALTER TABLE [dbo].[Countries]  WITH CHECK ADD  CONSTRAINT [CK_CountryName_1] CHECK  ((NOT [CountryName] like '%[0-9]%'))
```

9.Customers

Tabela zawierająca zewnętrzne firmy oraz klientów indywidualnych

- **PK** CustomerID (Int) - ID klienta NOT NULL
- Phone (nvarchar(9)) - numer telefonu klienta

Warunki Integralności:

- Phone jest 9 znakowym ciągiem znaków, każdy znak jest cyfrą od 0 do 9
  CHECK (([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))

```
CREATE TABLE [dbo].[Customers]
(
    [CustomerID] [int] NOT NULL,
    [Phone] [nvarchar](15) NULL,
    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],
    CONSTRAINT [U_Phone] UNIQUE NONCLUSTERED
(
    [Phone] 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]
ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT [CK_customer_phone]
 CHECK  (([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_customer_phone]
```

10.Dishes

Tabela zawierająca historie serwowanych dań w konkretnym czasie i cenę w danym menu

- **PK** DishID (int) - ID dania NOT NULL
- Price (money) - cena dania w konkretnym przedziale czasowym NOT NULL
- InDate (smallDateTime) - czas wejścia dania do menu NOT NULL
- OutDate (smallDateTime) - czas usunięcia dania z menu - jesli jest null'em to znaczy że jest obecnie w menu NOT NULL
- **FK** DishDictID - ID dania w slowniku dostępnych opcji NOT NULL

Warunki integralności:

- InDate oraz OutDate są większe niż 1. stycznia 1900 roku i mniejsze od daty obecnej
  CHECK (([InDate]>'01-01-1900' AND [InDate]<getdate()))
  CHECK (([OutDate]>'01-01-1900' AND [InDate]<getdate()))

```
CREATE TABLE [dbo].[Dishes]
(
    [DishID] [int] NOT NULL,
    [Price] [money] NOT NULL,
    [InDate] [smalldatetime] NOT NULL,
    [OutDate] [smalldatetime] NULL,
    [DishDictID] [int] NOT NULL,
    CONSTRAINT [PK__Dishes__18834F70B911114F] PRIMARY KEY CLUSTERED
(
    [DishID] 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]
ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT [FK_Dishes_DishesDict1] FOREIGN KEY([DishDictID])
REFERENCES [dbo].[DishesDict] ([DishDictID])
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_DishesDict1]
ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT [CK__Dishes__InDate__345EC57D]
 CHECK  (([InDate]>'01-01-1900' AND [InDate]<getdate()))
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [CK__Dishes__InDate__345EC57D]
ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT [CK__Dishes__OutDate__3552E9B6]
 CHECK  (([OutDate]>'01-01-1900'))
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [CK__Dishes__OutDate__3552E9B6]
```

11.DishesDict

Tabela zawierająca wszystkie dostępne opcje w restauracji włącznie z opcjami niedostępnymi obecnie w menu

- **PK** DishDictID (int) - ID dania w restauracji NOT NULL
- DishName (nvarchar(40) - nazwa dania NOT NULL
- **FK** CategoryID (int) - ID kategorii do której należy danie NOT NULL

Warunki integralności:

- DishName nie zawiera cyfr CHECK ((NOT [DishName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[DishesDict]
(
    [DishDictID] [int] NOT NULL,
    [DishName] [nvarchar](40) NOT NULL,
    [CategoryID] [int] NOT NULL,
    CONSTRAINT [PK_DishesDict_1] PRIMARY KEY CLUSTERED
(
    [DishDictID] 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]
ALTER TABLE [dbo].[DishesDict]  WITH CHECK ADD FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
ALTER TABLE [dbo].[DishesDict]  WITH CHECK ADD  CONSTRAINT [CK_DishesDictName_1] CHECK  ((NOT [DishName] like '%[0-9]%'))
```

12.Tables

Słownik dostępnych stolików wraz z ilością miejsc

- **PK** TableID (int) - ID dostępnych stolików NOT NULL
- SeatAmount (int) - liczba dostępnych przy stoliku miejsc NOT NULL

Warunki Integralności:

- Liczba miejsc jest większa od zera CHECK (([SeatAmount]>(0)))

```
CREATE TABLE [dbo].[Tables]
(
    [TableID] [int] NOT NULL,
    [SeatAmount] [smallint] NOT NULL,
    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]
ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [check_seats] CHECK  (([SeatAmount]>(0)))
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [check_seats]
```

13. IndividualCustomers

Tabela reprezentująca klientów indywidualnych

- **PK FK** CustomerID - id klienta restauracji NOT NULL
- FirstName(nvarchar(50) - imię pracownika firmy
- c. LastName (nvarchar(50) - nazwisko pracownika firmy

Warunki integralności:

- FirstName,LastName nie zawierają cyfr
  CHECK ((NOT [FirstName] like '%[0-9]%')),
  CHECK ((NOT [LastName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[IndividualCustomers]
(
    [CustomerID] [int] NOT NULL,
    [FirstName] [nvarchar](50) NULL,
    [LastName] [nvarchar](50) NULL,
    CONSTRAINT [PK_IndividualCustomers] 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]
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [FK_IndividualCustomers_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
ALTER TABLE [dbo].[IndividualCustomers] CHECK CONSTRAINT [FK_IndividualCustomers_Customers]
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [CK_IndividualCustomersFirstName_1]
 CHECK  ((NOT [FirstName] like '%[0-9]%'))
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [CK_IndividualCustomersLastName_1]
 CHECK  ((NOT [LastName] like '%[0-9]%'))
```

13. IndividualCustomers

Tabela zawiera informacje dotyczące klientów indywidualnych

- **PK FK** CustomerID (int) NOT NULL - ID klienta z tabeli Customers
- FirstName(nvarchar(50) - imię pracownika firmy NOT NULL
- LastName (nvarchar(50) - nazwisko pracownika firmy NOT NULL

Warunki integralności:

- FirstName, LastName nie zawierają cyfr

  CHECK ((NOT [FirstName] like '%[0-9]%')),
  CHECK ((NOT [LastName] like '%[0-9]%'))

```
CREATE TABLE [dbo].[IndividualCustomers]
(
    [CustomerID] [int] NOT NULL,
    [FirstName] [nvarchar](50) NULL,
    [LastName] [nvarchar](50) NULL,
    CONSTRAINT [PK_IndividualCustomers] 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]
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [FK_IndividualCustomers_Customers]
 FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
ALTER TABLE [dbo].[IndividualCustomers] CHECK CONSTRAINT [FK_IndividualCustomers_Customers]
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [CK_IndividualCustomersFirstName_1]
 CHECK  ((NOT [FirstName] like '%[0-9]%'))
ALTER TABLE [dbo].[IndividualCustomers]  WITH CHECK ADD  CONSTRAINT [CK_IndividualCustomersLastName_1]
 CHECK  ((NOT [LastName] like '%[0-9]%'))
```

14. IndividualReservations

Tabela zawiera informacje dotyczące rezerwacji indywidualnych

- **PK FK** ReservationID (int) NOT NULL - ID rezerwacji z tabeli Reservations
- **FK** CustomerID (int) NOT NULL - ID klienta z tabeli IndividualCustomers
- **FK** OrderID (int) NOT NULL - ID zamówenia z tabeli Orders
- PaidInAdvance (int) NOT NULL - Czy zapłacono przy wypełnianiu formularza
- BookDate (smalldatetime) NOT NULL - Data złożenia rezerwacji
- ReservationDate (smalldatetime) NOT NULL - Data rezerwacji
- **FK** TableID (int) NOT NULL - ID stolika z tabeli EatingTables

Warunki integralnościowe:

- PaidInAdvance zawiera wartości 'Yes' lub 'No', wartość domyślna 'No'
- OrderID ustawione jako unique
- BookDate z wartością domyślną getdate()
- BookDate oraz ReservationDate ustawione ramy czasowe

([PaidInAdvance]='No' OR [PaidInAdvance]='Yes') - check
([BookDate]>'01-01-1900' AND [BookDate]<=getdate()) -check
([ReservationDate]>'01-01-1900') - check
OrderID ustawiony jako unique key
Default w properties bookdate ustawiony na getdate()

```
CREATE TABLE [dbo].[IndividualReservations]
(
    [ReservationID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [PaidInAdvance] [nchar](3) NOT NULL,
    [BookDate] [smalldatetime] NOT NULL,
    [ReservationDate] [smalldatetime] NOT NULL,
    [SeatAmount] [int] NOT NULL,
    CONSTRAINT [PK_IndividualReservations] 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],
    CONSTRAINT [orders_unique] UNIQUE NONCLUSTERED
(
    [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]
ALTER TABLE [dbo].[IndividualReservations] ADD
  CONSTRAINT [DF_IndividualReservations_PaidInAdvance]  DEFAULT ('No') FOR [PaidInAdvance]
ALTER TABLE [dbo].[IndividualReservations] ADD
  CONSTRAINT [DF_IndividualReservations_BookDate]  DEFAULT (getdate()) FOR [BookDate]
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK
 ADD  CONSTRAINT [FK_IndividualReservations_IndividualCustomers1] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndividualCustomers] ([CustomerID])
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [FK_IndividualReservations_IndividualCustomers1]
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT [FK_IndividualReservations_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [FK_IndividualReservations_Orders]
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK
 ADD  CONSTRAINT [CK_IndividualReservations] CHECK  (([PaidInAdvance]='No' OR [PaidInAdvance]='Yes'))
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [CK_IndividualReservations]
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK
 ADD  CONSTRAINT [CK_IndividualReservations_1] CHECK  (([BookDate]>'01-01-1900' AND [BookDate]<=getdate()))
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [CK_IndividualReservations_1]
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK
 ADD  CONSTRAINT [CK_IndividualReservations_2] CHECK  (([ReservationDate]>'01-01-1900'))
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [CK_IndividualReservations_2]
```

15. Order Details

Tabela zawierająca szczegółowe informacje o zamówieniu

- **PK FK** OrderID (int) NOT NULL- ID zamówienia
- **PK FK** DishID (int) NOT NULL- ID dania
- Quantity (int) NOT NULL- ilość zamówionych dań
- UnitPrice (money) NOT NULL- cena za sztukę dania

Warunki integralnościowe:

- Quantity jest większe od 0, warość domyślna 1
- UnitPrice jest większe od 0

([UnitPrice]>(0)) - check
([Quantity]>(0)) - check
Default w properties Quantity ustawione na 1

```
CREATE TABLE [dbo].[OrderDetails]
(
    [OrderID] [int] NOT NULL,
    [DishID] [int] NOT NULL,
    [Quantity] [smallint] NOT NULL,
    [UnitPrice] [money] NOT NULL,
    CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC,
    [DishID] 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]
ALTER TABLE [dbo].[OrderDetails] ADD  CONSTRAINT [DF_OrderDetails_Quantity]  DEFAULT ((0)) FOR [Quantity]
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [FK__OrderDeta__DishI__3DE82FB7] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK__OrderDeta__DishI__3DE82FB7]
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [FK__OrderDeta__Order__3CF40B7E] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK__OrderDeta__Order__3CF40B7E]
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [CK_OrderDetails] CHECK  (([UnitPrice]>(0)))
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [CK_OrderDetails]
ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [CK_OrderDetails_1] CHECK  (([Quantity]>(0)))
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [CK_OrderDetails_1]
```

16.Orders

Tabela zawierająca informacje o zamówieniu

- **PK** OrderID (int) NOT NULL - ID zamówienia
- **FK** CustomerID (int) NOT NULL - ID klienta z tabeli Customers
- InDate (smalldatetime) NOT NULL - data złożenia zamówienia
- OutDate (smalldatetime) NOT NULL - data wykonania zamówienia
- TakeAway (nvarchar(3)) NOT NULL - czy jest to zamówienie na wynos
- **FK** WaiterID (int) NOT NULL - ID kelnera obsługującego zamówienie z tabeli Waiters

Warunki integralnościowe:

- Dla InDate oraz OutDate sprawdzenie czy mieszczą się w rozsądnych ramach czasowych
- Wartość domyślna InDate ustawiona na getdate()
- TakeAway przyjmuje wartości 'Yes' lub 'No'

([InDate]>'01-01-1900' AND [InDate]<=getdate()) - check
([OutDate]>'01-01-1900' AND [OutDate]<=getdate() OR [OutDate] IS NULL) -check
([TakeAway]='No' OR [TakeAway]='Yes') - check
Default w properties InDate ustawiony na getdate()

```
CREATE TABLE [dbo].[Orders]
(
    [OrderID] [int] NOT NULL,
    [CustomerID] [int] NOT NULL,
    [InDate] [smalldatetime] NOT NULL,
    [OutDate] [smalldatetime] NULL,
    [TakeAway] [nvarchar](3) NOT NULL,
    [WaiterID] [int] NOT NULL,
    CONSTRAINT [PK__Orders__C3905BAFB48C9799] 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]
ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_InDate]  DEFAULT (getdate()) FOR [InDate]
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK__Orders__Customer__70A8B9AE] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK__Orders__Customer__70A8B9AE]
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK__Orders__WaiterID__719CDDE7] FOREIGN KEY([WaiterID])
REFERENCES [dbo].[Waiters] ([WaiterID])
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK__Orders__WaiterID__719CDDE7]
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD
  CONSTRAINT [CK__Orders__InDate__72910220] CHECK  (([InDate]>'01-01-1900' AND [InDate]<=getdate()))
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK__Orders__InDate__72910220]
ALTER TABLE [dbo].[Orders]  WITH CHECK
 ADD  CONSTRAINT [CK__Orders__OutDate__73852659] CHECK  (([OutDate]>'01-01-1900' AND [OutDate]<=getdate() OR [OutDate] IS NULL))
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK__Orders__OutDate__73852659]
ALTER TABLE [dbo].[Orders]  WITH CHECK
 ADD  CONSTRAINT [CK_Orders] CHECK  (([TakeAway]='No' OR [TakeAway]='Yes'))
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
```

17.PermanentDiscounts

Tabela zawierająca wszystkie rabaty stałe

- **PK FK** CustomerID (int) NOT NULL - ID klienta z tabeli IndividualCustomers
- **FK** DiscountID (int) NOT NULL - ID rabatu z tabeli PermanentDiscountsDict
- StartDate (smalldatetime) NOT NULL - Data rozpoczęcia działania rabatu

Warunki integralnościowe:

- Dla StartDate sprawdzenie czy mieszczą się w rozsądnych ramach czasowych
- Wartość domyślna stardate ustawiona na getdate()

([StartDate]>'01-01-1900' AND [StartDate]<=getdate()) - check
getdate() ustawione jako default w properties StartDate

```
CREATE TABLE [dbo].[PermanentDiscounts]
(
    [CustomerID] [int] NOT NULL,
    [DiscountID] [int] NOT NULL,
    [StartDate] [smalldatetime] NOT NULL,
    CONSTRAINT [PK_PermanentDiscounts] 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]

ALTER TABLE [dbo].[PermanentDiscounts] ADD  CONSTRAINT [DF_PermanentDiscounts_StartDate]  DEFAULT (getdate()) FOR [StartDate]
ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD
  CONSTRAINT [FK_PermanentDiscounts_IndividualCustomers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndividualCustomers] ([CustomerID])
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT [FK_PermanentDiscounts_IndividualCustomers]
ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK
 ADD  CONSTRAINT [FK_PermanentDiscounts_PermanentDiscountsDict] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[PermanentDiscountsDict] ([PermDiscountID])
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT [FK_PermanentDiscounts_PermanentDiscountsDict]
ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK
 ADD  CONSTRAINT [CK_PermanentDiscounts] CHECK  (([StartDate]>'01-01-1900' AND [StartDate]<=getdate()))
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT [CK_PermanentDiscounts]
```

18.PermanentDiscountsDict

Tabela zawierająca szczegółowe informacje o rabatach stałych

- **PK** PermDiscountID (int) NOT NULL - ID rabatu
- Z1 (int) NOT NULL - liczba zamówień potrzebna dla uzyskania rabatu
- K1 (money) NOT NULL - minimalna cena zamówienia
- R1 (float) NOT NULL - wysokość rabatu

Warunki integralnościowe:

- Z1, K1 oraz R1 są dodatnie
- R1 nie może być większe niż 1

([Z1]>(0)) - check
([K1]>(0)) - check
([R1]>(0) AND [R1]<(1))

```
CREATE TABLE [dbo].[PermanentDiscountsDict]
(
    [PermDiscountID] [int] NOT NULL,
    [Z1] [smallint] NOT NULL,
    [K1] [money] NOT NULL,
    [R1] [float] NOT NULL,
    CONSTRAINT [PK_PermanentDiscountsDict] PRIMARY KEY CLUSTERED
(
    [PermDiscountID] 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]
ALTER TABLE [dbo].[PermanentDiscountsDict]  WITH CHECK
 ADD  CONSTRAINT [CK_PermanentDiscountsDict] CHECK  (([Z1]>(0)))
ALTER TABLE [dbo].[PermanentDiscountsDict] CHECK CONSTRAINT [CK_PermanentDiscountsDict]
ALTER TABLE [dbo].[PermanentDiscountsDict]  WITH CHECK
 ADD  CONSTRAINT [CK_PermanentDiscountsDict_1] CHECK  (([K1]>(0)))
ALTER TABLE [dbo].[PermanentDiscountsDict] CHECK CONSTRAINT [CK_PermanentDiscountsDict_1]
ALTER TABLE [dbo].[PermanentDiscountsDict]  WITH CHECK
 ADD  CONSTRAINT [CK_PermanentDiscountsDict_2] CHECK  (([R1]>(0) AND [R1]<(1)))
ALTER TABLE [dbo].[PermanentDiscountsDict] CHECK CONSTRAINT [CK_PermanentDiscountsDict_2]
```

19.ReservationsRestrictions

Tabela zawierająca warunki złożenia rezerwacji

- **PK** RestrictionID (int) NOT NULL - ID restrykcji
- WK (int) NOT NULL - ilość wymaganych wykonanych zamówień
- WZ (money) NOT NULL - minimalna wartość zamówienia

Warunki integralnościowe:

- WK oraz WZ są wartościami dodatnimi

([WK]>(0)) - check
([WZ]>(0)) - check

```
CREATE TABLE [dbo].[ReservationRestrictions]
(
    [RestrictionID] [int] NOT NULL,
    [WK] [int] NOT NULL,
    [WZ] [money] NOT NULL,
    PRIMARY KEY CLUSTERED
(
    [RestrictionID] 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]
ALTER TABLE [dbo].[ReservationRestrictions]  WITH CHECK ADD  CONSTRAINT [CK_ReservationRestrictions] CHECK  (([WK]>(0)))
ALTER TABLE [dbo].[ReservationRestrictions] CHECK CONSTRAINT [CK_ReservationRestrictions]
ALTER TABLE [dbo].[ReservationRestrictions]  WITH CHECK ADD  CONSTRAINT [CK_ReservationRestrictions_1] CHECK  (([WZ]>(0)))
ALTER TABLE [dbo].[ReservationRestrictions] CHECK CONSTRAINT [CK_ReservationRestrictions_1]
```

20.CompanyReservationsApproval

Tabela zawierająca informacje o zatwierdzeniu rezerwacji oraz przydzielonych stolikach dla rezerwacji firmowej

- **FK** ReservationID (int) NOT NULL- ID rezerwacji z tabeli Reservations
- ApproveDate (datetime) NOT NULL - Czas zaakcpetowania rezerwacji
- **FK** WaiterID (int) NOT NULL- ID kelnera, który zatwierdził rezerwację z tabeli Waiters
- **FK** TableID (int) NOT NULL - ID stoliku, który został przypisany do tej rezerwacji
- **PK** ApprovalID (int) NOT NULL - ID sztuczne zatwierdzonej rezerwacji firmowej

Warunki integralnościowe:

- Dla ApproveDate sprawdzenie czy mieci się w rozsądnych ramach czasowych
- Wartość domyślna ApproveDate ustawiona na getdate()

([ApproveDate]>'01-01-1900' AND [ApproveDate]<=getdate()) - check
getdate() - wartość default w properties ReservationsApproval

```
CREATE TABLE [dbo].[CompanyReservationsApproval]
(
    [ReservationID] [int] NOT NULL,
    [ApproveDate] [datetime] NOT NULL,
    [WaiterID] [int] NOT NULL,
    [TableID] [int] NOT NULL,
    [ApprovalID] [int] NOT NULL,
    CONSTRAINT [PK_CompanyReservationsApproval] PRIMARY KEY CLUSTERED
(
    [ApprovalID] 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]
ALTER TABLE [dbo].[CompanyReservationsApproval]  WITH CHECK
 ADD  CONSTRAINT [FK_CompanyReservationsApproval_CompanyReservationsDetails] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservationsDetails] ([ReservationDeatilsID])
ALTER TABLE [dbo].[CompanyReservationsApproval] CHECK CONSTRAINT [FK_CompanyReservationsApproval_CompanyReservationsDetails]
ALTER TABLE [dbo].[CompanyReservationsApproval]  WITH CHECK
 ADD  CONSTRAINT [FK_CompanyReservationsApproval_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
ALTER TABLE [dbo].[CompanyReservationsApproval] CHECK CONSTRAINT [FK_CompanyReservationsApproval_Tables]
ALTER TABLE [dbo].[CompanyReservationsApproval]  WITH CHECK
 ADD  CONSTRAINT [FK_CompanyReservationsApproval_Waiters] FOREIGN KEY([WaiterID])
REFERENCES [dbo].[Waiters] ([WaiterID])
ALTER TABLE [dbo].[CompanyReservationsApproval] CHECK CONSTRAINT [FK_CompanyReservationsApproval_Waiters]
```

21.ReservationsApproval

Tabela zawierająca informacje o zatwierdzeniu rezerwacji oraz przydzielonych stolikach dla klientów indywidualnych

- **FK** ReservationID (int) NOT NULL- ID rezerwacji z tabeli Reservations
- ApproveDate (datetime) NOT NULL - Czas zaakcpetowania rezerwacji
- **FK** WaiterID (int) NOT NULL- ID kelnera, który zatwierdził rezerwację z tabeli Waiters
- **FK** TableID (int) NOT NULL - ID stoliku, który został przypisany do tej rezerwacji
- **PK** ApprovalID (int) NOT NULL - ID sztuczne zatwierdzonej rezerwacji indywidualnej

Warunki integralnościowe:

- Dla ApproveDate sprawdzenie czy mieci się w rozsądnych ramach czasowych
- Wartość domyślna ApproveDate ustawiona na getdate()

([ApproveDate]>'01-01-1900' AND [ApproveDate]<=getdate()) - check
getdate() - wartość default w properties ReservationsApproval

```
CREATE TABLE [dbo].[ReservationsApproval]
(
    [ReservationID] [int] NOT NULL,
    [ApproveDate] [datetime] NOT NULL,
    [WaiterID] [int] NOT NULL,
    [TableID] [int] NOT NULL,
    [ApprovalID] [int] NOT NULL,
    CONSTRAINT [PK_ReservationsApproval_1] PRIMARY KEY CLUSTERED
(
    [ApprovalID] 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]
ALTER TABLE [dbo].[ReservationsApproval] ADD  CONSTRAINT [DF_ReservationsApproval_ApproveDate]  DEFAULT (getdate()) FOR [ApproveDate]
ALTER TABLE [dbo].[ReservationsApproval]  WITH CHECK ADD  CONSTRAINT [FK_ReservationsApproval_IndividualReservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[IndividualReservations] ([ReservationID])
ALTER TABLE [dbo].[ReservationsApproval] CHECK CONSTRAINT [FK_ReservationsApproval_IndividualReservations]
ALTER TABLE [dbo].[ReservationsApproval]  WITH CHECK ADD  CONSTRAINT [FK_ReservationsApproval_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
ALTER TABLE [dbo].[ReservationsApproval] CHECK CONSTRAINT [FK_ReservationsApproval_Tables]
ALTER TABLE [dbo].[ReservationsApproval]  WITH CHECK ADD  CONSTRAINT [FK_ReservationsApproval_Waiters] FOREIGN KEY([WaiterID])
REFERENCES [dbo].[Waiters] ([WaiterID])
ALTER TABLE [dbo].[ReservationsApproval] CHECK CONSTRAINT [FK_ReservationsApproval_Waiters]
ALTER TABLE [dbo].[ReservationsApproval]  WITH CHECK ADD
  CONSTRAINT [CK__Reservati__Appro__19AACF41] CHECK  (([ApproveDate]>'01-01-1900' AND [ApproveDate]<=getdate()))
ALTER TABLE [dbo].[ReservationsApproval] CHECK CONSTRAINT [CK__Reservati__Appro__19AACF41]
```

22. StreetCustomers

Tabela reprezentująca przypadkowych niezarejestrowanych klientów

- **PK FK** CustomerID (int) NOT NULL - ID klienta z tabeli IndividualCustomers

```
CREATE TABLE [dbo].[StreetCustomers]
(
    [CustomerID] [int] NOT NULL,
    CONSTRAINT [PK_StreetCustomers] 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]
ALTER TABLE [dbo].[StreetCustomers]  WITH CHECK ADD FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
```

23. TemporaryDiscounts

Tabela reprezentująca rabaty tymczasowe dla klientów indywidualnych

- **FK** CustomerID (int) NOT NULL - ID klienta z tabeli IndividualCustomers
- **FK** DiscountID (int) NOT NULL - ID rabatu z tabeli TemporaryDiscountsDict
- StartDate (smalldatetime) NOT NULL - Czas rozpoczęcia działania rabatu
- EndDate (smalldatetime) NOT NULL - Czas zakończenia działania rabatu
- **PK** TemporaryDiscountID (int) NOT NULL - ID rabatu tymczasowego

Warunki integralnościowe:

- Dla StartDate oraz EndDate sprawdzenie czy mieszczą się w rozsądnych ramach czasowych
- Wartość domyślna StartDate ustawiona na getdate()

([StartDate]>'01-01-1900' AND [StartDate]<=getdate())
getdate() ustawione w properties dla StartDate

```
CREATE TABLE [dbo].[TemporaryDiscounts]
(
    [CustomerID] [int] NOT NULL,
    [DiscountID] [int] NOT NULL,
    [StartDate] [smalldatetime] NOT NULL,
    [EndDate] [smalldatetime] NOT NULL,
    [TemporaryDiscountID] [int] NOT NULL,
    CONSTRAINT [PK_TemporaryDiscounts] PRIMARY KEY CLUSTERED
(
    [TemporaryDiscountID] 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]
ALTER TABLE [dbo].[TemporaryDiscounts] ADD  CONSTRAINT [DF_TemporaryDiscounts_StartDate]  DEFAULT (getdate()) FOR [StartDate]
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK
 ADD  CONSTRAINT [FK_TemporaryDiscounts_IndividualCustomers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[IndividualCustomers] ([CustomerID])
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [FK_TemporaryDiscounts_IndividualCustomers]
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK
 ADD  CONSTRAINT [FK_TemporaryDiscounts_TemporaryDiscountsDict] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[TemporaryDiscountsDict] ([TempDiscountID])
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [FK_TemporaryDiscounts_TemporaryDiscountsDict]
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD
  CONSTRAINT [CK_TemporaryDiscounts] CHECK  (([StartDate]>'01-01-1900' AND [StartDate]<=getdate()))
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [CK_TemporaryDiscounts]
ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD
  CONSTRAINT [CK_TemporaryDiscounts_1] CHECK  (([EndDate]>'01-01-1900'))
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [CK_TemporaryDiscounts_1]
```

24.TemporaryDiscountsDict

Słownik zawierający szczegółowe parametry rabatów tymczasowych

- **PK** TempDiscountID (int) NOT NULL - ID rabatu
- K2 (money) NOT NULL - wymagana kwota
- D1 (int) NOT NULL - długość trwania rabatu
- R2 (float) NOT NULL - wysokość zniżki

Warunki integralnościowe:

- K2, D1, R2 większe od 0
- R2 mniejsze od 1

([K2]>(0)) - check
([D1]>(0)) - check
([R2]>(0) AND [R2]<(1)) - check

```
CREATE TABLE [dbo].[TemporaryDiscountsDict]
(
    [TempDiscountID] [int] NOT NULL,
    [K2] [money] NOT NULL,
    [D1] [int] NOT NULL,
    [R2] [float] NOT NULL,
    CONSTRAINT [PK_TemporaryDiscountsDict] PRIMARY KEY CLUSTERED
(
    [TempDiscountID] 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]
ALTER TABLE [dbo].[TemporaryDiscountsDict]  WITH CHECK ADD  CONSTRAINT [CK_TemporaryDiscountsDict] CHECK  (([K2]>(0)))
ALTER TABLE [dbo].[TemporaryDiscountsDict] CHECK CONSTRAINT [CK_TemporaryDiscountsDict]
ALTER TABLE [dbo].[TemporaryDiscountsDict]  WITH CHECK ADD  CONSTRAINT [CK_TemporaryDiscountsDict_1] CHECK  (([D1]>(0)))
ALTER TABLE [dbo].[TemporaryDiscountsDict] CHECK CONSTRAINT [CK_TemporaryDiscountsDict_1]
ALTER TABLE [dbo].[TemporaryDiscountsDict]  WITH CHECK ADD  CONSTRAINT [CK_TemporaryDiscountsDict_2] CHECK  (([R2]>(0) AND [R2]<(1)))
ALTER TABLE [dbo].[TemporaryDiscountsDict] CHECK CONSTRAINT [CK_TemporaryDiscountsDict_2]
```

25.Waiters

Tabela zawierająca dane o kelnerach naszej restauracji

- **PK** WaiterID (int)- ID kelnera
- Firstname (nvarchar(50)) NOT NULL - imię kelnera
- Lastname (nvarchar(50)) NOT NULL- nazwisko kelnera
- HireDate (smalldatetime) NOT NULL- data zatrudnienia kelnera
- Phone (nvarchar(9)) NOT NULL - telefon

Warunki integralnościowe:

- Dla HireDate sprawdzenie czy mieści się w rozsądnych ramach czasowych
- 9-cyfrowy numer telefonu
- Brak cyfr w imieniu i nazwisku

([HireDate]>'01-01-1900' AND [HireDate]<=getdate()) - check
([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') - check
(NOT [Lastname] like '%[0-9]%') - check
(NOT [Firstname] like '%[0-9]%') - check

```
CREATE TABLE [dbo].[Waiters]
(
    [WaiterID] [int] NOT NULL,
    [Firstname] [nvarchar](50) NOT NULL,
    [Lastname] [nvarchar](50) NOT NULL,
    [HireDate] [smalldatetime] NOT NULL,
    [Phone] [nvarchar](9) NOT NULL,
    CONSTRAINT [PK__Waiters__88C0F0724DBE2CA3] PRIMARY KEY CLUSTERED
(
    [WaiterID] 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],
    CONSTRAINT [unique_phone] UNIQUE NONCLUSTERED
(
    [Phone] 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]
ALTER TABLE [dbo].[Waiters] ADD  CONSTRAINT [DF_Waiters_HireDate]  DEFAULT (getdate()) FOR [HireDate]
ALTER TABLE [dbo].[Waiters]  WITH CHECK ADD  CONSTRAINT [CK__Waiters__HireDat__6DCC4D03]
 CHECK  (([HireDate]>'01-01-1900' AND [HireDate]<=getdate()))
ALTER TABLE [dbo].[Waiters] CHECK CONSTRAINT [CK__Waiters__HireDat__6DCC4D03]
ALTER TABLE [dbo].[Waiters]  WITH CHECK
 ADD  CONSTRAINT [CK_Waiters] CHECK  (([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
ALTER TABLE [dbo].[Waiters] CHECK CONSTRAINT [CK_Waiters]
ALTER TABLE [dbo].[Waiters]  WITH CHECK
 ADD  CONSTRAINT [CK_Waiters_1] CHECK  ((NOT [Lastname] like '%[0-9]%'))
ALTER TABLE [dbo].[Waiters] CHECK CONSTRAINT [CK_Waiters_1]
ALTER TABLE [dbo].[Waiters]  WITH CHECK
 ADD  CONSTRAINT [CK_Waiters_2] CHECK  ((NOT [Lastname] like '%[0-9]%'))
ALTER TABLE [dbo].[Waiters] CHECK CONSTRAINT [CK_Waiters_2]
```

Widoki

1. ActiveTempDiscounts

Pokazuje użytkowników o aktywnych rabatach jednorazowych

```
CREATE VIEW ACTIVETEMPDISCOUNTS
AS SELECT CustomerID, StartDate, EndDate FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON DiscountID = TempDiscountID
WHERE EndDate > GETDATE()
```

2. CustomersWithPermDiscount

Pokazuje użytkowników posiadających rabaty stałe

```
CREATE VIEW CustomersWithPermDiscount
AS SELECT IndividualCustomers.CustomerID, R1 FROM IndividualCustomers
INNER JOIN PermanentDiscounts
ON IndividualCustomers.CustomerID = PermanentDiscounts.CustomerID
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
```

3. GetSeaFood

pokazuje listę dostępnych w menu owoców morza

```
CREATE VIEW GetSeafood
AS SELECT DishID FROM Dishes
INNER JOIN DishesDict
ON Dishes.DishDictID = DishesDict.DishDictID
INNER JOIN Categories
ON Categories.CategoryID = DishesDict.CategoryID
WHERE CategoryName='Seafood' AND OutDate < GETDATE()
```

4. ApprovedReservations

pokazuje potwierdzone rezerwacje na dany dzień

```
CREATE VIEW ApprovedReservations
AS SELECT ReservationsApproval.ReservationID, TableID
FROM ReservationsApproval
INNER JOIN IndividualReservations
ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
WHERE ReservationDate = GETDATE()
UNION
SELECT ReservationsApproval.ReservationID, ReservationsApproval.TableID
FROM ReservationsApproval
INNER JOIN CompanyReservationsDetails
ON CompanyReservationsDetails.ReservationID = ReservationsApproval.ReservationID
INNER JOIN CompanyReservations
ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
WHERE ReservationDate = GETDATE()
```

5. WaiterOfTheMonth

pokazuje kelnera który obsłużył najwięcej zamówień w tym miesiącu

```
CREATE VIEW WaiterOfTheMonth
AS SELECT TOP 1 Waiters.WaiterID, Firstname, Lastname, COUNT(*) as orders_completed FROM Waiters
INNER JOIN Orders
ON Orders.WaiterID = Waiters.WaiterID
WHERE MONTH(OutDate) = MONTH(GETDATE()) AND YEAR(OutDate) = YEAR(GETDATE())
GROUP BY Waiters.WaiterID, Firstname, Lastname
ORDER BY orders_completed DESC
```

6. CustomerOfTheMonth

pokazuje klienta który złożył najwięcej zamówień w tym miesiącu

```
CREATE VIEW CustomerOfTheMonth
AS SELECT TOP 1 IndividualCustomers.CustomerID, Lastname, Firstname, COUNT(*) as orders_made  FROM IndividualCustomers
INNER JOIN Customers
ON IndividualCustomers.CustomerID = Customers.CustomerID
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
WHERE MONTH(OutDate) = MONTH(GETDATE()) AND YEAR(OutDate) = YEAR(GETDATE())
GROUP BY IndividualCustomers.CustomerID, Lastname, Firstname
ORDER BY orders_made DESC
```

7. **ShowExpiredDishes**

   pokazuje dania w menu, które należy wymienić

```
CREATE VIEW ShowExpiredDishes
AS SELECT * FROM Dishes
WHERE DATEDIFF(DAY, InDate, GETDATE()) > 14 AND OutDate IS NULL
```

8. BestDishes

pokazuje dania które były najczęściej zamawiane w bieżącym roku

```
CREATE VIEW BestDishes
AS SELECT TOP 12 Dishes.DishID, COUNT(*) as ordered FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
WHERE DATEDIFF(YEAR, InDate, GETDATE()) < 1
GROUP BY Dishes.DishID
```

9. DishWithBestIncome

pokazuje danie, które dostarczyło najwięcej przychodu w bieżącym miesiącu

```
CREATE VIEW DishWithBestIncome
AS SELECT TOP 1 Dishes.DishID, SUM(Quantity*UnitPrice) as income FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
WHERE DATEDIFF(MONTH, InDate, GETDATE()) < 1
GROUP BY Dishes.DishID
ORDER BY income DESC
```

10. CurrentPermDiscount

pokazuje aktualne parametry dla zniżki stałej

```
CREATE VIEW DishWithBestIncome
AS SELECT TOP 1 Dishes.DishID, SUM(Quantity*UnitPrice) as income FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
WHERE DATEDIFF(MONTH, InDate, GETDATE()) < 1
GROUP BY Dishes.DishID
ORDER BY income DESC
```

11. CurrentTempDiscount

pokazuje aktualne parametry dla zniżki tymczasowej

```
CREATE VIEW CurrentTempDiscount
AS SELECT TOP 1 * FROM TemporaryDiscountsDict
ORDER BY TempDiscountID DESC
```

12. NotConfirmedReservations

pokazuje jeszcze nie zatwierdzone rezerwacje

```
CREATE VIEW NotConfirmedReservations
AS SELECT ReservationID , 'individual' as reservation_type FROM IndividualReservations
WHERE ReservationID
NOT IN (SELECT ReservationID FROM ReservationsApproval)
UNION
SELECT ReservationID, 'company' as reservation_type FROM CompanyReservationsDetails
WHERE ReservationID
NOT IN ( SELECT CompanyReservationsDetails.ReservationID FROM CompanyReservationsDetails
INNER JOIN CompanyReservationsApproval
ON CompanyReservationsApproval.ReservationID = CompanyReservationsDetails.ReservationDeatilsID)
```

13. MonthTableRaport

raporty wykorzystania stolików w ostatnim miesiącu z podziałem na miesiące

```
ALTER VIEW [dbo].[MonthTableRaport]
AS SELECT Tables.TableID, COUNT(*) as times_used, YEAR(ReservationDate) as year,
MONTH(ReservationDate) as month FROM Tables
INNER JOIN ReservationsApproval
ON Tables.TableID = ReservationsApproval.TableID
INNER JOIN IndividualReservations
ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate), MONTH(ReservationDate)
GO
```

14. WeekTableRaport

raport wykorzystania konkretnych stolików w ostatnim tygodniu

```
ALTER VIEW [dbo].[WeekTableRaport]
AS SELECT Tables.TableID, COUNT(*) as times_used, YEAR(ReservationDate) as year,
MONTH(ReservationDate) as month, DATEPART(WEEK, ReservationDate) as week FROM Tables
INNER JOIN ReservationsApproval
ON Tables.TableID = ReservationsApproval.TableID
INNER JOIN IndividualReservations
ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate), MONTH(ReservationDate), DATEPART(WEEK, ReservationDate)
GO
```

15. YearTableRaport

raport wykorzystania stolików z podziałem na lata

```
ALTER VIEW [dbo].[YearTableRaport]
AS SELECT Tables.TableID, COUNT(*) as times_used, YEAR(ReservationDate) as year FROM Tables
INNER JOIN ReservationsApproval
ON Tables.TableID = ReservationsApproval.TableID
INNER JOIN IndividualReservations
ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate)
```

16. WeekCompanyTableRaport

Raport wykorzystania stolików przez firmy z podziałem na tygodnie

```
CREATE VIEW WeekCompanyTableRaport
AS SELECT Tables.TableID, COUNT(*) as times_used, YEAR(ReservationDate) as 'year',
MONTH(ReservationDate) as 'month', DATEPART(WEEK, ReservationDate) as 'week' FROM Tables
INNER JOIN CompanyReservationsApproval
ON Tables.TableID = CompanyReservationsApproval.TableID
INNER JOIN CompanyReservationsDetails
ON CompanyReservationsDetails.ReservationDeatilsID = CompanyReservationsApproval.ReservationID
INNER JOIN CompanyReservations
ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate), MONTH(ReservationDate), DATEPART(WEEK, ReservationDate)
```

17. MonthCompanyTableRaport

Raport wykorzystania stolików przez firmy z podziałem na miesiące

```
CREATE VIEW MonthCompanyTableRaport
AS SELECT Tables.TableID, COUNT(*) as times_used , YEAR(ReservationDate) as 'year',
MONTH(ReservationDate) as'month' FROM Tables
INNER JOIN CompanyReservationsApproval
ON Tables.TableID = CompanyReservationsApproval.TableID
INNER JOIN CompanyReservationsDetails
ON CompanyReservationsDetails.ReservationDeatilsID = CompanyReservationsApproval.ReservationID
INNER JOIN CompanyReservations
ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate), MONTH(ReservationDate)
```

18. YearTableRaport

Raport wykorzystania stolików przez klientów indywidualnych

```
ALTER VIEW [dbo].[YearTableRaport]
AS SELECT Tables.TableID, COUNT(*) as times_used, YEAR(ReservationDate) as year FROM Tables
INNER JOIN ReservationsApproval
ON Tables.TableID = ReservationsApproval.TableID
INNER JOIN IndividualReservations
ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
GROUP BY Tables.TableID, YEAR(ReservationDate)
```

19. MonthPermDiscountReport

raport rabatów stałych przyznanych przez ostatni miesiąc

```
CREATE VIEW MonthPermDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, Z1, K1, R1 FROM PermanentDiscount
INNER JOIN PermanentDiscountsDict
ON PermanentDiscounts.DiscountID = PermanentDiscountsDict.PermDiscountID
WHERE DATEDIFF(MONTH, StartDate, GETDATE()) < 1
```

20. YearPermDiscountReport

raport rabatów stałych przyznanych przez ostatni rok

```
CREATE VIEW YearPermDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, Z1, K1, R1 FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermanentDiscounts.DiscountID = PermanentDiscountsDict.PermDiscountID
WHERE DATEDIFF(YEAR, StartDate, GETDATE()) < 1
```

21. WeekPermDiscountReport

raport rabatów stałych przyznanych przez ostatni tydzień

```
CREATE VIEW WeekPermDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, Z1, K1, R1 FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermanentDiscounts.DiscountID = PermanentDiscountsDict.PermDiscountID
WHERE DATEDIFF(DAY, StartDate, GETDATE()) < 7
```

22. MonthTempDiscountReport

raport rabatów tymczasowych przyznanych przez ostatni miesiąc

```
CREATE VIEW MonthTempDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, K2, D1, R2 FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TemporaryDiscounts.DiscountID = TemporaryDiscountsDict.TempDiscountID
WHERE DATEDIFF(MONTH, StartDate, GETDATE()) < 1
```

23. YearTempDiscountReport

raport rabatów tymczasowych przyznanych przez ostatni rok

```
CREATE VIEW YearTempDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, K2, D1, R2 FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TemporaryDiscounts.DiscountID = TemporaryDiscountsDict.TempDiscountID
WHERE DATEDIFF(YEAR, StartDate, GETDATE()) < 1
```

24. WeekTempDiscountReport

raport rabatów tymczasowych przyznanych przez ostatni tydzień

```
CREATE VIEW WeekTempDiscountReport
AS SELECT DiscountID, CustomerID, StartDate, K2, D1, R2 FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TemporaryDiscounts.DiscountID = TemporaryDiscountsDict.TempDiscountID
WHERE DATEDIFF(DAY, StartDate, GETDATE()) < 7
```

25. WeekMenuReport

raport zarobków dań z podziałem na tygodnie

```
CREATE VIEW [dbo].[WeekMenuRaport]
AS SELECT DishID, SUM(suma) as suma, YEAR(InDate) as year, MONTH(InDate) as month, DATEPART(WEEK, InDate) as week
FROM(SELECT Dishes.DishID, Orders.InDate, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
INNER JOIN Orders
ON Orders.OrderID = OrderDetails.OrderID
) as total
GROUP BY DishID, YEAR(InDate), MONTH(InDate), DATEPART(WEEK, InDate)
GO
```

26. MonthMenuReport

raport zarobków dań z podziałem na miesiące

```
CREATE VIEW [dbo].[MonthMenuRaport]
AS SELECT DishID, SUM(suma) as suma, YEAR(InDate) as year, MONTH(InDate) as month
FROM(SELECT Dishes.DishID, Orders.InDate, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
INNER JOIN Orders
ON Orders.OrderID = OrderDetails.OrderID
) as total
GROUP BY DishID, YEAR(InDate), MONTH(InDate)
GO
```

27. YearMenuReport

raport zarobków dań z podziałem na lata

```
CREATE VIEW YearMenuRaport
AS SELECT DishID, SUM(suma) as suma, YEAR(InDate) as year FROM(SELECT Dishes.DishID, Orders.InDate, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Dishes
INNER JOIN OrderDetails
ON OrderDetails.DishID = Dishes.DishID
INNER JOIN Orders
ON Orders.OrderID = OrderDetails.OrderID
) as total
GROUP BY DishID, YEAR(InDate)
```

28. WeekCustomerReport

raport wydanych pieniędzy przez klientów z podziałem na tygodnie

```
ALTER VIEW [dbo].[WeekCustomerRaport]
AS SELECT CustomerID, SUM(suma) as suma , YEAR(InDate) as year, MONTH(InDate) as month, DATEPART(WEEK, InDate) as week
FROM(SELECT Orders.InDate, Customers.CustomerID, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Customers
INNER JOIN Orders
ON Orders.CustomerID = Customers.CustomerID
INNER JOIN OrderDetails
ON OrderDetails.OrderID = Orders.OrderID) as total
GROUP BY CustomerID, YEAR(InDate), MONTH(InDate), DATEPART(WEEK, InDate)
GO
```

29. MonthCustomerReport

raport wydanych pieniędzy przez klientów z podziałem na miesiące

```
ALTER VIEW [dbo].[MonthCustomerRaport]
AS SELECT CustomerID, SUM(suma) as suma , YEAR(InDate) as year, MONTH(InDate) as month
FROM(SELECT Orders.InDate, Customers.CustomerID, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Customers
INNER JOIN Orders
ON Orders.CustomerID = Customers.CustomerID
INNER JOIN OrderDetails
ON OrderDetails.OrderID = Orders.OrderID) as total
GROUP BY CustomerID, YEAR(InDate), MONTH(InDate)
GO
```

30. YearCustomerReport

raport wydanych pieniędzy przez klientów z podziałem na lata

```
ALTER VIEW [dbo].[YearCustomerRaport]
AS SELECT CustomerID, SUM(suma) as suma , YEAR(InDate) as year
FROM(SELECT Orders.InDate, Customers.CustomerID, Quantity*UnitPrice*(1-ISNULL(
(SELECT TOP 1 R FROM(
SELECT R2 as R, CustomerID FROM TemporaryDiscounts
INNER JOIN TemporaryDiscountsDict
ON TempDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.InDate < EndDate AND Orders.CustomerID = TemporaryDiscounts.CustomerID
UNION
SELECT R1 as R, CustomerID FROM PermanentDiscounts
INNER JOIN PermanentDiscountsDict
ON PermDiscountID = DiscountID
WHERE StartDate< Orders.InDate AND Orders.CustomerID = PermanentDiscounts.CustomerID
)
as R
Order By R DESC
),0)) as suma FROM Customers
INNER JOIN Orders
ON Orders.CustomerID = Customers.CustomerID
INNER JOIN OrderDetails
ON OrderDetails.OrderID = Orders.OrderID) as total
GROUP BY CustomerID, YEAR(InDate)
GO
```

31. YearCompanyReport

Raport roczny dotyczący wydatków firm

```
ALTER VIEW [dbo].[YearCompanyRaport]
AS
SELECT CompanyID, SUM(suma) as suma,YEAR(InDate) as year
FROM (SELECT InDate, Companies.CompanyID,
             (Quantity * UnitPrice) as suma
      FROM Customers
               INNER JOIN Companies ON Customers.CustomerID = Companies.CompanyID
               INNER JOIN Orders
                          ON Orders.CustomerID = Customers.CustomerID
               INNER JOIN OrderDetails
                          ON OrderDetails.OrderID = Orders.OrderID) as total
inner join Customers on CompanyID = Customers.CustomerID
GROUP BY CompanyID,YEAR(InDate)
```

32. MonthlyCompanyReport

Raport miesięczny dotyczący wydatków firm

```
ALTER VIEW [dbo].[MonthlyCompanyRaport]
AS
SELECT CompanyID, SUM(suma) as suma,YEAR(InDate) as year, MONTH(InDate) as month
FROM (SELECT InDate, Companies.CompanyID,
             (Quantity * UnitPrice) as suma
                FROM Customers
               INNER JOIN Companies ON Customers.CustomerID = Companies.CompanyID
               INNER JOIN Orders
                          ON Orders.CustomerID = Customers.CustomerID
               INNER JOIN OrderDetails
                          ON OrderDetails.OrderID = Orders.OrderID) as total
inner join Customers on CompanyID = Customers.CustomerID
GROUP BY CompanyID, YEAR(InDate),Month(InDate)
```

33. WeeklyCompanyReport

Raport tygodniowy dotyczący wydatków firm

```
ALTER VIEW [dbo].[WeekCompanyRaport]
AS
SELECT CompanyID, SUM(suma) as suma,YEAR(InDate) as year, MONTH(InDate) as month, DATEPART(WEEK, InDate) as week
FROM (SELECT InDate, Companies.CompanyID,
             (Quantity * UnitPrice) as suma
      FROM Customers
               INNER JOIN Companies ON Customers.CustomerID = Companies.CompanyID
               INNER JOIN Orders
                          ON Orders.CustomerID = Customers.CustomerID
               INNER JOIN OrderDetails
                          ON OrderDetails.OrderID = Orders.OrderID) as total
inner join Customers on CompanyID = Customers.CustomerID
GROUP BY CompanyID,YEAR(InDate),MONTH(InDate),DATEPART(WEEK, InDate)
```

34. WeekIndividualOrderTimeReport

Raport dotyczący godziny składania zamówień przez klientów indywidualnych z podziałem na tygodnie

```
CREATE VIEW WeekIndividualOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year', MONTH(InDate) as 'month', DATEPART(WEEK, InDate) as 'week' FROM Orders
INNER JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate), MONTH(InDate), DATEPART(WEEK, InDate)
```

35. MonthIndividualOrderTimeReport

Raport dotyczący godziny składania zamówień przez klientów indywidualnych z podziałem na miesiące

```
CREATE VIEW MonthIndividualOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year', MONTH(InDate) as 'month' FROM Orders
INNER JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate), MONTH(InDate)
```

36. YearIndividualOrderTimeReport

Raport dotyczący godziny składania zamówień przez klientów indywidualnych z podziałem na lata

```
CREATE VIEW YearIndividualOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year' FROM Orders
INNER JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate)
```

37. WeekCompanyOrderTimeReport

Raport dotyczący godziny składania zamówień przez firmy z podziałem na tygodnie

```
CREATE VIEW WeekCompanyOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year', MONTH(InDate) as 'month', DATEPART(WEEK, InDate) as 'week' FROM Orders
INNER JOIN Companies ON
Companies.CompanyID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate), MONTH(InDate), DATEPART(WEEK, InDate)
```

38. MonthCompanyOrderTimeReport

Raport dotyczący godziny składania zamówień przez firmy z podziałem na miesiące

```
CREATE VIEW MonthCompanyOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year', MONTH(InDate) as 'month' FROM Orders
INNER JOIN Companies ON
Companies.CompanyID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate), MONTH(InDate)
```

39. YearCompanyOrderTimeReport

Raport dotyczący godziny składania zamówień przez firmy z podziałem na lata

```
CREATE VIEW YearCompanyOrderTimeRaport
AS SELECT Count(*) as liczba_zamowien, DATEPART(HOUR, InDate) as 'godzina',
YEAR(InDate) as 'year' FROM Orders
INNER JOIN Companies ON
Companies.CompanyID = Orders.CustomerID
GROUP BY DATEPART(HOUR, InDate), YEAR(InDate)
```

40. ShowCurrentMenu

Pokazuje aktualnie dostępne menu

```
CREATE VIEW ShowCurrentMenu
AS
SELECT DishID, Price, DishName
FROM Dishes
        INNER JOIN DishesDict DD on DD.DishDictID = Dishes.DishDictID
WHERE OutDate IS NULL
```

Procedury

AddCountry

Dodawania państwa

```
ALTER PROCEDURE [dbo].[AddCountry]
@country nvarchar(30)
AS
BEGIN
    IF @country NOT IN (SELECT CountryName FROM Countries)
        BEGIN
            DECLARE @lastID int;
            SET @lastID =
                (SELECT TOP 1 CountryID FROM Countries
                ORDER BY CountryID DESC)
            INSERT INTO Countries(CountryID, CountryName) VALUES (@lastID+1, @country)
        END
    ELSE
        BEGIN
            THROW 52000, 'This Country is already in Countries',1
        END
END
```

AddCity

Dodawanie miasta

```
ALTER PROCEDURE [dbo].[AddCity]
@city nvarchar(30),
@country nvarchar(30)
AS
BEGIN
    IF @country NOT IN (SELECT CountryName FROM Countries)
        BEGIN
            THROW 52000,'No country with that name', 1
        END
    DECLARE @countryID int;
    SET @countryID =
        (SELECT CountryID FROM Countries
        WHERE CountryName LIKE @country
        )
    IF @city NOT IN (SELECT CityName FROM Cities)
        BEGIN
            DECLARE @lastID int;
            SET @lastID =
                (SELECT TOP 1 CityID FROM Cities
                ORDER BY CityID DESC)
            INSERT INTO Cities(CityID, CityName, CountryID) VALUES (@lastID+1, @city, @countryID)
        END
    ELSE
        BEGIN
            THROW 52000, 'This City is already in Cities', 1
        END
END
```

AddCompany

Dodawanie firmy

```
ALTER PROCEDURE [dbo].[AddCompany](
    @CompanyName NVARCHAR(30),
    @city NVARCHAR(40),
    @NIP NVARCHAR(10),
    @Facture_every NVARCHAR(5),
    @Phone NVARCHAR(9),
    @Address NVARCHAR(50),
    @Postal_code NVARCHAR(10)
)
AS
BEGIN
    IF @city NOT IN (SELECT CityName FROM Cities)
        BEGIN
            THROW 52000,'You have to add new city first',1
        END
    DECLARE @CityID int =
        (SELECT CityID FROM Cities
        WHERE CityName=@city)
    DECLARE @index int = ISNULL((SELECT MAX(CustomerID) FROM Customers),0)+1
    INSERT INTO Customers(CustomerID, Phone) VALUES (@index, @phone)
    INSERT INTO Companies(CompanyID, CompanyName, NIP, FactureEvery, Address, CityID, PostalCode)
    VALUES (@index,@CompanyName,@NIP,@Facture_every,@Address,@CityID,@Postal_code)
END
```

AddCompanyEmployee

Dodawanie pracownika firmy

```
ALTER PROCEDURE [dbo].[AddCompanyEmployee] @company_name VARCHAR(40),
                                           @firstname VARCHAR(40),
                                           @lastname VARCHAR(40)

AS
BEGIN
    IF @company_name NOT IN (SELECT CompanyName FROM Companies)
        BEGIN
            THROW 52000, 'You have to add company first', 1
        END
    DECLARE @lastID int = (SELECT MAX(EmployeeID) FROM CompaniesWorkers)
    DECLARE @companyID int;
    SET @companyID = (
        SELECT companyID
        FROM Companies
        WHERE CompanyName LIKE @company_name)
    INSERT INTO CompaniesWorkers(EmployeeID, CompanyID, FirstName, LastName)
    VALUES (@lastID + 1, @companyID, @firstname, @lastname)
END
```

AddTable

Dodawanie stolika

```
ALTER Procedure [dbo].[AddTable]
@seat_amount int
AS
BEGIN
DECLARE @lastID int = (SELECt MAX(TableID) FROM Tables)
INSERT INTO Tables (TableID, SeatAmount)
VALUES (@lastID+1, @seat_amount)
END
```

ApproveReservation

Zatwierdzanie rezerwacji indywidualnej

```
ALTER Procedure [dbo].[ApproveReservation]
@reservationID int,
@waiterID int
AS
BEGIN
    IF @reservationID NOT IN (SELECT ReservationID FROM IndividualReservations)
        BEGIN
            THROW 52000, 'No individual reservation with this ID', 1
        END
    IF @waiterID NOT IN (SELECT WaiterID FROM Waiters)
        BEGIN
            THROW 52000, 'Wrong WaiterID', 1
        END
    DECLARE @lastID int = ISNULL((SELECT MAX(ApprovalID) FROM ReservationsApproval),0)
    DECLARE @seat_amount int = (SELECT SeatAmount FROM IndividualReservations WHERE ReservationID = @reservationID)
    DECLARE @reservation_date date = (SELECT ReservationDate FROM IndividualReservations WHERE ReservationID = @reservationID)
    DECLARE @tableID int =  (SELECT TableID FROM FindBestTable(@reservation_date, @seat_amount))
    IF @tableID IS NULL
        BEGIN
            THROW 52000, 'No tables available', 1
        END
    INSERT INTO ReservationsApproval(ReservationID,ApproveDate,WaiterID,TableID, ApprovalID)
    VALUES (@reservationID, GETDATE(), @waiterID,@tableID, @lastID+1)
END
```

ApproveCompanyReservation

Zatwierdzanie rezerwacji firmowej

```
ALTER Procedure [dbo].[ApproveCompanyReservation]
@reservationDetailID int,
@waiterID int
AS
BEGIN
    IF @reservationDetailID NOT IN (SELECT ReservationDeatilsID FROM CompanyReservationsDetails)
        BEGIN
            THROW 52000, 'No individual reservation with this ID', 1
        END
    IF @waiterID NOT IN (SELECT WaiterID FROM Waiters)
        BEGIN
            THROW 52000, 'Wrong WaiterID', 1
        END
    DECLARE @lastID int = (SELECT MAX(ApprovalID) FROM CompanyReservationsApproval)
    DECLARE @seat_amount int = (SELECT SeatAmount FROM CompanyReservationsDetails WHERE ReservationDeatilsID = @reservationDetailID)
    DECLARE @reservation_date date =
                        (SELECT ReservationDate FROM CompanyReservationsDetails
                        INNER JOIN CompanyReservations
                        ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
                        WHERE ReservationDeatilsID = @reservationDetailID)
    DECLARE @tableID int =  (SELECT TableID FROM FindBestTable(@reservation_date, @seat_amount))
    IF @tableID IS NULL
        BEGIN
            THROW 52000, 'No tables available', 1
        END
    INSERT INTO CompanyReservationsApproval(ReservationID,ApproveDate,TableID,WaiterID, ApprovalID)
    VALUES (@reservationDetailID, GETDATE(), @tableID, @waiterID, @lastID+1)
END
```

AddIndividualReservation

Dodawanie rezerwacji indywidualnej

```
ALTER Procedure [dbo].[AddIndividualReservation]
@customerID int,
@orderID int,
@reservationDate smalldatetime,
@seat_amount int
AS
BEGIN
    IF @customerID NOT IN (SELECT CustomerID FROM IndividualCustomers)
        BEGIN
            THROW 52000, 'No Individual Customer with this ID',1
        END
    IF @orderID NOT IN (SELECT OrderID FROM Orders)
        BEGIN
            THROW 52000, 'No Order with this ID',1
        END
    IF @customerID != (SELECT CustomerID FROM Orders WHERE OrderID=@orderID)
        BEGIN
            THROW 52000, 'Order złego customera', 1
        END
    IF (SELECT COUNT(*) FROM ListCustomerOrders(@customerID)) < (SELECT TOP 1 WK FROM ReservationRestrictions)+1
        BEGIN
            THROW 52000, 'Zamawiający nie spełnia wymagań', 1
        END
    IF (dbo.GetScalarTotalOrder(@orderID) < (SELECT TOP 1 WZ FROM ReservationRestrictions))
        BEGIN
            THROW 52000, 'Zamówienie nie spełnia wymagań', 1
        END
    DECLARE @paid_in_advance varchar(3) = (SELECT IsPaid FROM Orders WHERE @orderID= OrderID)
    DECLARE @lastID int = (SELECT MAX(ReservationID) FROM IndividualReservations)
    INSERT INTO IndividualReservations(ReservationID, CustomerID, OrderID, PaidInAdvance, BookDate, ReservationDate, SeatAmount)
```

AddCompanyReservation

Dodawanie rezerwacji firmowej

```
ALTER PROCEDURE [dbo].[AddCompanyReservation]
@CompanyID int,
@ReservationDate smalldatetime
AS
BEGIN
    IF @CompanyID NOT IN (SELECT CompanyID FROM Companies)
        BEGIN
            THROW 52000, 'No company like this', 1
        END
    DECLARE @lastID int = (SELECT MAX(ReservationID) FROM CompanyReservations)
    INSERT INTO CompanyReservations(ReservationID, CompanyID, BookDate, ReservationDate)
    VALUES (@lastID+1, @CompanyID, GETDATE(), @ReservationDate)
END
```

AddCompanyReservationDetail

Dodawanie szczegółów rezerwacji firmowej

```
ALTER PROCEDURE [dbo].[AddCompanyReservationDeatil]
@ReservationID int,
@SeatAmount int
AS
BEGIN
    IF @ReservationID NOT IN (SELECT ReservationID FROM CompanyReservations)
        BEGIN
            THROW 52000, 'No company reservation like this', 1
        END
    IF @ReservationID IN (SELECT ReservationID FROM CompanyReservations
    WHERE ReservationDate < GETDATE())
        BEGIN
            THROW 52000, 'Old reservation used', 1
        END
    DECLARE @lastID int = (SELECT MAX(ReservationDeatilsID) FROM CompanyReservationsDetails)
    INSERT INTO CompanyReservationsDetails(ReservationID, SeatAmount, ReservationDeatilsID)
    VALUES (@ReservationID, @SeatAmount ,@lastID+1)
END
```

AddCompanyPersonalReservation

Przypisywanie pracowników do rezerwacji firmowej

```
ALTER PROCEDURE [dbo].[AddCompanyPersonalReservation]
    @ReservationDetailID int,
    @EmployeeID int
AS
BEGIN
    IF @ReservationDetailID NOT IN (SELECT ReservationDeatilsID FROM CompanyReservationsDetails)
    BEGIN
        THROW 52000, 'Wrong Reservation detail ID', 1
    END
    IF (SELECT COUNT(*) FROM CompanyPersonalReservationsDetails
        WHERE ReservationDetailID = @ReservationDetailID)
        >=
        (SELECT SeatAmount FROM CompanyReservationsDetails
        WHERE ReservationDeatilsID = @ReservationDetailID)
    BEGIN
        THROW 52000, 'Table is already full', 1
    END
    IF @EmployeeID NOT IN (SELECT EmployeeID FROM CompaniesWorkers)
    BEGIN
        THROW 52000, 'Wrong EmployeeID', 1
    END
    DECLARE @companyID int = (SELECT CompanyID FROM CompaniesWorkers
                              WHERE EmployeeID = @EmployeeID)
    IF (SELECT CompanyID FROM CompanyReservationsDetails
        INNER JOIN CompanyReservations
        ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
        WHERE ReservationDeatilsID = @ReservationDetailID) != @companyID
            BEGIN
                THROW 52000, 'Employee from wrong company', 1
            END
    IF @EmployeeID IN (SELECT EmployeeID FROM CompanyPersonalReservationsDetails
        WHERE @ReservationDetailID = ReservationDetailID)
            BEGIN
                THROW 52000, 'Employee already added to this table', 1
            END
    DECLARE @lastID int = ISNULL((SELECT MAX(CPReservationID) FROM CompanyPersonalReservationsDetails),0)
    INSERT INTO CompanyPersonalReservationsDetails(CPReservationID, EmployeeID, ReservationDetailID)
    VALUES(@lastID+1, @EmployeeID, @ReservationDetailID)
END
```

RemoveIndicidualReservation

Usuwanie rezerwacji indywidualnej

```
ALTER PROCEDURE [dbo].[RemoveIndividualReservation](@reservationID int)
AS
BEGIN
    IF @reservationID NOT IN (SELECT ReservationID FROM IndividualReservations)
        BEGIN
            THROW 52000, 'No reservation with this ID', 1
        END
    IF (SELECT ReservationDate FROM IndividualReservations
        WHERE @reservationID = ReservationID) < GETDATE()
        BEGIN
            THROW 52000, 'It is old reservation', 1
        END
    DELETE ReservationsApproval WHERE @reservationID = ReservationID
    DELETE IndividualReservations WHERE @reservationID = ReservationID
END
```

RemoveCompanyReservation

Usuwanie rezerwacji firmowej

```
CREATE PROCEDURE RemoveCompanyReservation(@reservationID int)
AS
BEGIN
    IF @reservationID NOT IN (SELECT ReservationID FROM CompanyReservations)
        BEGIN
            THROW 52000, 'No reservation with this ID', 1
        END
    IF (SELECT ReservationDate FROM CompanyReservations
        WHERE @reservationID = ReservationID) < GETDATE()
        BEGIN
            ;THROW 52000, 'It is old reservation', 1
        END
    DELETE CompanyReservationsDetails WHERE @reservationID = ReservationID
    DELETE CompanyReservations WHERE @reservationID = ReservationID
END
```

AddCategory

Dodawanie kategorii

```
CREATE PROCEDURE AddCategory @CategoryName nvarchar(40)
AS
BEGIN
    IF @CategoryName NOT IN (SELECT CategoryName FROM Categories)
        BEGIN
            DECLARE @index INT
            SET @index = (SELECT TOP 1 CategoryID FROM Categories ORDER BY CategoryID desc)+1
            INSERT INTO Categories VALUES (@index, @CategoryName)
        END
    ELSE
        BEGIN
            THROW 52000, 'This Category is already in CategoryName',1
        END
END
```

AddDish

Dodawanie nowego dania do listy możliwości

```
CREATE PROCEDURE AddDish(
    @DishName nvarchar(40),
    @CategoryName nvarchar(40))
AS
BEGIN
    IF @DishName IN (SELECT DishName from DishesDict)
        BEGIN
            THROW 52000, 'Dish already exists in DishesDict',1
        END

    DECLARE @CategoryID INT;
    SET @CategoryID = (SELECT CategoryID FROM Categories WHERE CategoryName = @CategoryName)
    IF @CategoryName IS NULL
        BEGIN
            THROW 52000, 'Category does not exists',1
        END
    DECLARE @index INT;
    SET @index = (SELECT TOP 1 DishDictID FROM DishesDict ORDER BY DishDictID DESC ) + 1
   INSERT INTO DishesDict VALUES (@index, @DishName, @CategoryID)
END
```

AddDishToOrder

Dodaje danie do istniejącego zamówienia

```
CREATE PROCEDURE AddDishToOrder(
    @OrderID int,
    @DishID int,
    @Quantity int
)
AS
BEGIN
    IF @OrderID NOT IN (SELECT OrderID FROM Orders)
        BEGIN
            THROW 52000,'You have to make order fist.',1
        END
    IF @DishID NOT IN (SELECT DishID FROM ShowCurrentMenu)
        BEGIN
            THROW 52000,'Dish is not in menu',1
        END
    DECLARE @price money

    IF @DishID IN (SELECT DishID
                   FROM Dishes
                            INNER JOIN DishesDict DD on DD.DishDictID = Dishes.DishDictID
                            INNER JOIN Categories C on C.CategoryID = DD.CategoryID
                   WHERE CategoryName LIKE 'seafood')
        BEGIN
            DECLARE @OutDate DATE = (SELECT OutDate FROM Orders WHERE OrderID = @OrderID);
            DECLARE @InDate DATE = (SELECT InDate FROM Orders WHERE OrderID = @OrderID)
            DECLARE @WeekDayOut INT= DATEPART(WEEKDAY, @OutDate);
```

```
            IF (@WeekDayOut < 4 OR @WeekDayOut > 6)
                BEGIN
                    THROW 52000,'You can eat seafood only between thursday and saturday',1;
                END
            IF @InDate > DATEADD(DAY, -DATEPART(WEEKDAY, @OutDate) + 2, @OutDate)
                BEGIN
                    THROW 52000,'You have to make an order for seafood until Monday',1;
                END
        END

    SET @price = (SELECT Price FROM ShowCurrentMenu WHERE DishID = @DishID)

    IF @DishID NOT IN (SELECT DishID FROM OrderDetails WHERE OrderID = @OrderID)
        BEGIN
            INSERT INTO OrderDetails VALUES (@OrderID, @DishID, @Quantity, @price)
        END
    ELSE
        BEGIN
            DECLARE @sumQuantity int
            SET @sumQuantity = (SELECT Quantity
                                FROM OrderDetails
                                WHERE OrderID = @OrderID
                                  AND DishID = @DishID) + @Quantity

            DELETE FROM OrderDetails WHERE OrderID = @OrderID AND DishID = @DishID
            INSERT INTO OrderDetails VALUES (@OrderID, @DishID, @sumQuantity, @price)
        END
END
```

AddIndividualCustomer

Tworzy klienta indywidualnego

```
CREATE PROCEDURE AddIndividualCustomer(
    @Phone nvarchar(9),
    @FirstName nvarchar(40),
    @LastName nvarchar(40)
)
AS
BEGIN
    DECLARE @CustomerID INT
    SET @CustomerID = ISNULL((SELECT MAX(CustomerID)FROM Customers),1)+1

    INSERT INTO Customers VALUES (@CustomerID,@Phone)
    INSERT INTO IndividualCustomers VALUES (@CustomerID,@FirstName,@LastName)
END
```

AddToMenu

Z listy dostępnych dań dodaje danie do aktualnego menu

```
CREATE PROCEDURE AddToMenu(
    @DishDictID int,
    @Price money)
AS
BEGIN
    IF @DishDictID NOT IN (SELECT DishDictID FROM DishesDict)
        BEGIN
            THROW 52000,'Dish does not exist in DishesDict',1
        END
    DECLARE @index int
    SET @index = (SELECT TOP 1 DishID from Dishes ORDER BY DishID DESC)+1
    INSERT INTO Dishes VALUES (@index, @Price, DATEADD(SECOND,1,GETDATE()), null, @DishDictID)
END
```

CancelOrder

Anuluje zamówienie, jeśli wykonał je klient indywidualny i z jego powodu została przyznana mu zniżka stała, anuluje przyznanie zniżki

```
CREATE PROCEDURE CancelOrder(
    @OrderID INT
)
AS
BEGIN
    IF @OrderID NOT IN (SELECT OrderID FROM Orders WHERE OutDate >= GETDATE())
        BEGIN
            THROW 52000,'You cannot cancel this order',1
        END
    DELETE FROM OrderDetails WHERE OrderID = @OrderID
    DELETE FROM Orders WHERE OrderID = @OrderID

    ---DELETING PERMANENT DISCOUNT IF NEEDED
    DECLARE @CustomerID INT = (Select CustomerID from Orders where OrderID = @OrderID)
    DECLARE @K1 MONEY= (SELECT TOP 1 K1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    DECLARE @Z1 INT= (SELECT TOP 1 Z1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    IF @CustomerID
           IN (SELECT CustomerID
               FROM IndividualCustomers
               WHERE CustomerID NOT IN (SELECT CustomerID from PermanentDiscounts))
        AND @Z1 > (SELECT COUNT(*)
                   FROM (SELECT Orders.OrderID
                         FROM Orders
                                  inner join OrderDetails OD on Orders.OrderID = OD.OrderID
                         where CustomerID = @CustomerID
                         GROUP BY Orders.OrderID
                         HAVING dbo.GetScalarTotalOrder( @OrderID: Orders.OrderID) > @K1) AS SUM)
        BEGIN
            DELETE FROM PermanentDiscounts WHERE CustomerID=@CustomerID
        END
END
```

AddPermamentDiscount

Jeśli klient spełnia warunki, dodaje mu aktualnie przyznawaną zniżkę stałą

```
CREATE PROCEDURE AddPermanentDiscount(
    @CustomerID int
)
AS
BEGIN
    IF @CustomerID NOT IN (SELECT CustomerID FROM IndividualCustomers)
        BEGIN
            THROW 52000,'Customer does not exist',1
        END
    DECLARE @DiscountID int
    SET @DiscountID = (SELECT TOP 1 PermDiscountID FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    IF @DiscountID IS NULL
        BEGIN
            THROW 52000,'There are no discounts available',1
        END
    IF @DiscountID IN (SELECT DiscountID from PermanentDiscounts where CustomerID = @CustomerID)
        BEGIN
            THROW 52000,'Customer has already used this discount',1
        END
    DECLARE @K1 MONEY
    DECLARE @Z1 INT
    DECLARE @R1 FLOAT
    SET @K1 = (SELECT TOP 1 K1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    SET @Z1 = (SELECT TOP 1 Z1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    SET @R1 = (SELECT TOP 1 R1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    DECLARE @check int


    SET @check = (SELECT COUNT(*)
                  FROM (SELECT Orders.OrderID
                        FROM Orders
                                 inner join OrderDetails OD on Orders.OrderID = OD.OrderID
                        where CustomerID  = @CustomerID
                        GROUP BY Orders.OrderID
                        HAVING dbo.GetScalarTotalOrder ( @OrderID: Orders.OrderID) > @K1) AS SUM)

    IF @check < @Z1
        BEGIN
            THROW 52000,'Customer has to make more valuable orders',1;
        END
    DELETE FROM PermanentDiscounts WHERE @CustomerID = CustomerID
    INSERT INTO PermanentDiscounts VALUES(@CustomerID,@DiscountID,DATEADD(SECOND ,-100,GETDATE()))
END
```

AddWaiter

Dodaje nowego Kelnera

```
CREATE PROCEDURE AddWaiter(
    @FirstName nvarchar(50),
    @LastName nvarchar(50),
    @Phone nvarchar(9)
)
AS
BEGIN
    DECLARE @index int
    SET @index = ISNULL((SELECT MAX(WaiterID) FROM Waiters), 0) + 1
    INSERT INTO Waiters VALUES (@index, @FirstName, @LastName,  DATEADD(SECOND,-100,GETDATE()), @Phone)
END
```

CreatePermanentDiscount

Dodaje aktualnie przyznawaną zniżkę stałą do listy zniżek stałych

```
CREATE PROCEDURE createPermanentDiscount(
    @K1 MONEY,
    @Z1 INT,
    @R1 FLOAT
)
AS
BEGIN
    DECLARE @index int = (SELECT MAX(PermDiscountID) FROM PermanentDiscountsDict)
    INSERT INTO PermanentDiscountsDict VALUES (@index, @K1, @Z1, @R1)
END
```

CreateTemporaryDiscount

Dodaje zniżkę tymczasową do listy zniżek tymczasowych

```
CREATE PROCEDURE createTemporaryDiscount(
    @K2 MONEY,
    @D1 INT,
    @R2 FLOAT
)
AS
BEGIN
    DECLARE @index int = (SELECT MAX(TempDiscountID) FROM TemporaryDiscountsDict)
    INSERT INTO TemporaryDiscountsDict VALUES (@index, @K2, @D1, @R2)
END
```

MakeOrder

Tworzy nowe zamówienie

```
CREATE PROCEDURE MakeOrder(
    @CustomerID INT,
    @OutDate DATE,
    @TakeAway nvarchar(3),
    @WaiterID int = 1,
    @DishID int,
    @Quantity int
)
AS
BEGIN
    IF @CustomerID NOT IN (SELECT CustomerID FROM Customers)
        BEGIN
            THROW 52000,'Customer does not exist',1
        END
    IF @OutDate < GETDATE()
        BEGIN
            THROW 52000,'Order date must be at least equal to current date or later',1
        END
    IF @WaiterID NOT IN (SELECT WaiterID FROM Waiters)
        THROW 52000,'Waiter does not exists',1
    IF @DishID NOT IN (SELECT DishID FROM ShowCurrentMenu)
        BEGIN
            THROW 52000,'Dish is not in menu',1
        END
    IF @TakeAway != 'Yes' AND @TakeAway != 'No'
        BEGIN
            THROW 52000, 'Takeaway has to be declared as Yes or No',1
        END
    DECLARE @index int
    SET @index = (SELECT TOP 1 OrderID
                  FROM Orders
                  order by OrderID desc) + 1

    DECLARE @price money
    SET @price = (SELECT Price FROM ShowCurrentMenu WHERE DishID = @DishID)


    INSERT INTO Orders
    VALUES (@index, @CustomerID, DATEADD(SECOND, -100, GETDATE()), @OutDate, @TakeAway, @WaiterID, 'No')
    EXECUTE AddDishToOrder @OrderID: @index, @DishID: @DishID, @Quantity: @Quantity
END
```

RemoveDishFromOrder

Usuwa danie z jeszcze nie wykonanego zamówienia

```
CREATE PROCEDURE RemoveDishFromOrder(
    @OrderID int,
    @DishID int
)
AS
BEGIN
    IF @OrderID NOT IN (SELECT OrderID FROM Orders)
        BEGIN
            THROW 52000,'You have to make order fist.',1
        END
    IF @DishID NOT IN (SELECT DishID
                       FROM Orders
                                JOIN OrderDetails OD on Orders.OrderID = OD.OrderID
                       WHERE Orders.OrderID = @OrderID
                         and OutDate < GETDATE())
        BEGIN
            THROW 52000,'You are not able to remove dish from this order',1
        END
    DELETE FROM OrderDetails WHERE OrderID = @OrderID AND DishID = @DishID
END
```

AddTemporaryDiscount

Przypisuje użytkownikowi podaną zniżkę tymczasową jeśli spełnia jej warunki

```
CREATE PROCEDURE AddTemporaryDiscount(
    @CustomerID int,
    @TemporaryID int
)
AS
BEGIN
    IF @CustomerID NOT IN (SELECT CustomerID FROM IndividualCustomers)
        BEGIN
            THROW 52000,'Customer does not exist',1
        END
    IF @TemporaryID NOT IN (SELECT TempDiscountID FROM TemporaryDiscountsDict)
        BEGIN
            THROW 52000,'Discount does not exist',1
        END


    DECLARE @K2 MONEY = (SELECT K2 FROM TemporaryDiscountsDict WHERE TempDiscountID = @TemporaryID)
    DECLARE @D1 INT = (SELECT D1 FROM TemporaryDiscountsDict WHERE TempDiscountID = @TemporaryID)
    DECLARE @R2 FLOAT= (SELECT R2 FROM TemporaryDiscountsDict WHERE TempDiscountID = @TemporaryID)

    IF dbo.GetScalarTotalCustomerValue( @CustomerID: @CustomerID) < @K2
        BEGIN
            THROW 52000,'Customer has to make more valuable orders',1;
        END

    DECLARE @ID INT = (SELECT MAX(TemporaryDiscountID) FROM TemporaryDiscounts) + 1

    INSERT INTO TemporaryDiscounts
    VALUES (@CustomerID, @TemporaryID, DATEADD(SECOND , -100, GETDATE()), DATEADD(DAY, @D1, GETDATE()), @ID)
END
```

DeleteMenuPosition

Dezaktywuje pozycje z aktualnego menu

```
CREATE PROCEDURE deleteMenuPosition(
    @DishID INT
)
AS
BEGIN
    IF @DishID NOT IN (SELECT DishID FROM ShowCurrentMenu)
        BEGIN
            THROW 52000,'This position is not in current menu',1
        END
    UPDATE Dishes SET OutDate = GETDATE() WHERE @DishID = DishID
END
go
```

PayOrder

Zmienia stan płatności zamówienia na opłacone

```
CREATE PROCEDURE PayOrder(@OrderID INT)
AS
    BEGIN
        IF @OrderID NOT IN(SELECT OrderID FROM Orders WHERE IsPaid='No')
        BEGIN
            THROW 52000,'You cannot pay this order',1
        END
        UPDATE Orders SET IsPaid='Yes' WHERE OrderID = @OrderID
    END
```

Funkcje

Free Tables

Znajduje wolne stoliki dla danego dnia

```
CREATE FUNCTION FreeTables(@day date)
returns table as
    return SELECT * FROM Tables
    WHERE TableID NOT IN
        (SELECT TableID from ReservationsApproval
        INNER JOIN IndividualReservations
        ON IndividualReservations.ReservationID = ReservationsApproval.ReservationID
        WHERE ReservationDate = @day
        UNION
        SELECT CompanyReservationsApproval.TableID from CompanyReservationsApproval
        INNER JOIN CompanyReservationsDetails
        ON CompanyReservationsDetails.ReservationDeatilsID = CompanyReservationsApproval.ReservationID
        INNER JOIN CompanyReservations
        ON CompanyReservations.ReservationID = CompanyReservationsDetails.ReservationID
        WHERE ReservationDate = @day
        )
```

FindBestTable

Znajduje stolik odpowiedni dla danej daty oraz ilości miejsc

```
CREATE FUNCTION FindBestTable (@day date, @seat_amount int)
returns table as
    return SELECT TOP 1 * From FreeTables(@day)
    WHERE SeatAmount>=@seat_amount
    ORDER BY SeatAmount
```

ListCustomerOrders

Listuje zamówienia danego klienta

```
CREATE FUNCTION ListCustomerOrders(@customerID int)
returns table as
    return SELECT * FROM Orders
    WHERE CustomerID = @customerID
```

GenerateFacture

Generuje fakturę dla pojedynczego zamówienia

```
CREATE FUNCTION GenerateFacture(@companyID int, @orderID int)
RETURNS TABLE AS
RETURN SELECT CompanyName, NIP, dbo.GetScalarTotalOrder(@orderID) as total FROM Companies
        WHERE @companyID = CompanyID
```

GenerateMonthFacture

Generuje fakturę miesięczną

```
CREATE FUNCTION GenerateMonthFacture(@companyID int)
RETURNS TABLE
AS
RETURN SELECT CompanyName, NIP, OrderID, dbo.GetScalarTotalOrder(OrderID) as total, InDate FROM
        (SELECT OrderID, CompanyID, CompanyName, NIP, IsPaid, InDate
        from orders
         INNER JOIN Customers C on C.CustomerID = Orders.CustomerID
         inner join  Companies C2 on C.CustomerID = C2.CompanyID) as company_orders
        WHERE @companyID = CompanyID AND IsPaid = 'No' AND DATEDIFF(MONTH, InDate, GETDATE()) < 1
```

CustomerRaport

Zwraca informacje o zamówieniach dokonanych przez pojedyńczego klienta indywidualnego

```
CREATE FUNCTION CustomerRaport(@CustomerID INT)
    RETURNS TABLE as RETURN
            (
                SELECT I.FirstName + ' ' + I.LastName   as Customer,
                       OrderID,
                       dbo.GetScalarTotalOrder( @OrderID: OrderID) as Value,
                       InDate
                from IndividualCustomers I
                         join Customers C on C.CustomerID = I.CustomerID
                         join Orders O on C.CustomerID = O.CustomerID
                where C.CustomerID = @CustomerID
            )
```

GetCustomerDiscount

Zwraca informacje o aktualnej (najwyższej z dwóch) zniżce dla klienta indywidualnego w postaci tabeli

```
CREATE FUNCTION GetCustomerDiscount(
    @CustomerID INT,
    @Date DATE
)
    RETURNS @DiscountTable TABLE
                           (
                               CustomerID int,
                               R          float
                           )
AS
BEGIN
    DECLARE @PD FLOAT
    SET @PD = (SELECT TOP 1 PDD.R1 AS R
               FROM PermanentDiscounts
                        JOIN PermanentDiscountsDict PDD
                             on PermanentDiscounts.DiscountID = PDD.PermDiscountID
               WHERE CustomerID = @CustomerID
                AND @Date >= StartDate
               ORDER BY R)

    DECLARE @TD FLOAT
    SET @TD = (SELECT TOP 1 R2 AS R
               FROM TemporaryDiscounts
                        JOIN TemporaryDiscountsDict TDD
                             on TemporaryDiscounts.DiscountID = TDD.TempDiscountID
               WHERE CustomerID = @CustomerID
                 AND TemporaryDiscounts.StartDate <= @DATE
                 AND (EndDate IS NULL OR EndDate <= @DATE)
               ORDER BY R)
```

GetDiscountDuringOrder

Zwraca w postaci tabeli discount jaki był naliczany dla klienta indywidualnego przy danym zamówieniu

```
CREATE FUNCTION discountDuringOrder(
    @OrderID INT
)
    RETURNS @DiscountTable TABLE
                           (
                               UserID INT,
                               R      FLOAT
                           )
AS
BEGIN
    IF @OrderID NOT IN (SELECT OrderID FROM Orders)
        BEGIN
            THROW 52000,'There is no order with this ID',1
        END
    DECLARE @Date DATE = (SELECT InDate FROM Orders WHERE OrderID = @OrderID)
    DECLARE @CustomerID INT = (SELECT CustomerID FROM Orders WHERE OrderID = @OrderID)
    INSERT INTO @DiscountTable
    Values (@CustomerID, (SELECT r FROM GetCustomerDiscount( @CustomerID: @CustomerID,  @Date: @Date)))
    RETURN
END
```

GetTotalCustomerValue

Zwraca w postaci tabeli ilość pieniędzy jakie wydał klient w naszej restauracji

```
CREATE FUNCTION GetTotalCustomerValue(@CustomerID INT)
    RETURNS @Total TABLE
                   (
                       CustomerID INT,
                       Total      MONEY
                   )
AS
BEGIN
    DECLARE @Value MONEY = (SELECT ISNULL(SUM(dbo.GetScalarTotalOrder( @OrderID: OrderID)), 0) AS TotalAmount
                            FROM Orders
                            WHERE CustomerID = @CustomerID)

    INSERT INTO @Total VALUES (@CustomerID, @Value)
    RETURN
END
```

GetScalarTotalCustomerValue

Zwraca jako MONEY ilość pieniędzy jakie wydał klient w naszej restauracji

```
CREATE FUNCTION GetScalarTotalCustomerValue(@CustomerID INT)
    RETURNS MONEY
AS
BEGIN

    DECLARE @Value MONEY = (SELECT ISNULL(SUM(dbo.GetScalarTotalOrder( @OrderID: OrderID)), 0) AS TotalAmount
                            FROM Orders
                            WHERE CustomerID = @CustomerID)
    RETURN @Value
END
```

GetTotalOrderValue

Zwraca w postaci tabeli informację o danym zamówieniu

```
CREATE FUNCTION GetTotalOrderValue(@OrderID INT)
    RETURNS @TotalValue TABLE
                        (
                            OrderID    INT,
                            CustomerID INT,
                            Total      MONEY,
                            OrderDate  DATE
                        )
AS
BEGIN
    IF @OrderID not in (select OrderID from Orders)
        BEGIN
            THROW 52000,'There is no order with this ID',1
        END
    DECLARE @Date DATE = (SELECT InDate FROM Orders WHERE OrderID = @OrderID)
    DECLARE @CustomerID INT = (SELECT CustomerID FROM Orders WHERE OrderID = @OrderID)
    DECLARE @Discount FLOAT = (SELECT R FROM GetDiscountDuringOrder( @OrderID: @OrderID))
    DECLARE @Total MONEY = (SELECT SUM((1 - @Discount) * UnitPrice * OrderDetails.Quantity)
                            FROM OrderDetails
                            WHERE OrderID = @OrderID)
    INSERT INTO @TotalValue VALUES (@OrderID, @CustomerID, @Total, @Date)
    RETURN
END
```

GetScalarTotalOrder

Zwraca jako MONEY wartość konkretnego zamówienia

```
CREATE FUNCTION GetScalarTotalOrder(@OrderID INT)
    RETURNS Money
AS
BEGIN
    DECLARE @Discount FLOAT = (SELECT R FROM GetDiscountDuringOrder( @OrderID: @OrderID))
    DECLARE @Total MONEY = (SELECT SUM((1 - @Discount) * UnitPrice * OrderDetails.Quantity)
                            FROM OrderDetails
                            WHERE OrderID = @OrderID)
    RETURN @Total
END
```

Triggery

CheckValidIndReservationDates

Sprawdza czy data rezerwacji nie jest wcześniejsza niż data jej złożenia

```
CREATE TRIGGER CheckValidIndReservationDates
ON IndividualReservations
AFTER INSERT
AS
BEGIN
    DECLARE @book_date date =(SELECT BookDate FROM inserted)
    DECLARE @reservation_date date =(SELECT ReservationDate FROM inserted)
    IF DATEDIFF(day,@book_date,@reservation_date)<0
    BEGIN
        THROW 50002, 'Data odbioru nie moze byc wczesniejsza niz data zlozenia zamowienia',1
    END
END
```

CheckValidCompanyReservationDates

Sprawdza czy data rezerwacji nie jest wcześniejsza niż data jej złożenia

```
CREATE TRIGGER CheckValidCompanyReservationDates
ON CompanyReservations
AFTER INSERT
AS
BEGIN
    DECLARE @book_date date =(SELECT BookDate FROM inserted)
    DECLARE @reservation_date date =(SELECT ReservationDate FROM inserted)
    IF DATEDIFF(day,@book_date,@reservation_date)<0
    BEGIN
        THROW 50002, 'Data odbioru nie moze byc wczesniejsza niz data zlozenia zamowienia',1
    END
END
```

CheckDuplicatedCompanyReservation

Sprawdza czy nie istnieje już taka sama rezerwacja firmowa

```
ALTER TRIGGER [dbo].[CheckDuplicatedCompanyReservation]
ON [dbo].[CompanyReservations]
AFTER INSERT
AS
BEGIN
    DECLARE @CompanyID int = (SELECT CompanyID FROM inserted)
    DECLARE @ReservationDate smalldatetime = (SELECT ReservationDate FROM inserted)
    DECLARE @ReservationID int = (SELECT ReservationID FROM inserted)
    IF EXISTS(SELECT * FROM CompanyReservations
    WHERE CompanyID=@CompanyID AND DATEDIFF(DAY, ReservationDate, @ReservationDate)<1 AND @ReservationID != ReservationID)
    BEGIN
        THROW 50002, 'Zduplikowana rezerwacja',1
    END
END
```

CheckDuplicatedIndReservation

Sprawdza czy nie istnieje taka sama rezerwacja indywidualna

```
CREATE TRIGGER CheckDuplicatedIndReservation
ON IndividualReservations
AFTER INSERT
AS
BEGIN
    SELECT * FROM inserted
    DECLARE @CustomerID int = (SELECT CustomerID FROM inserted)
    DECLARE @ReservationDate smalldatetime = (SELECT ReservationDate FROM inserted)
    DECLARE @ReservationID int = (SELECT ReservationID FROM inserted)
    SELECT @CustomerID
    SELECT @ReservationDate
    IF EXISTS(SELECT ReservationID FROM IndividualReservations
    WHERE CustomerID=@customerID AND YEAR(ReservationDate)=YEAR(@ReservationDate)
    AND MONTH(ReservationDate) = MONTH(@ReservationDate) AND DAY(ReservationDate) = DAY(@ReservationDate)
    AND @ReservationID != ReservationID)
        BEGIN
            ;THROW 50002, 'Zduplikowana rezerwacja',1
        END
END
```

UpdatePermanentDiscount

Po dodaniu zamówienia sprawdza czy klient dostanie zniżkę stałą

```
Create TRIGGER UpdatePermamentDiscount
    on Orders
    after
        insert
    AS
BEGIN
    Declare @CustomerID INT= (SELECT CustomerID FROM inserted)
    DECLARE @K1 MONEY= (SELECT TOP 1 K1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)
    DECLARE @Z1 INT= (SELECT TOP 1 Z1 FROM PermanentDiscountsDict ORDER BY PermDiscountID DESC)

    IF @CustomerID IN (SELECT CustomerID
                       FROM IndividualCustomers
                       WHERE CustomerID NOT IN (SELECT CustomerID from PermanentDiscounts))
        AND @Z1 <= (SELECT COUNT(*)
                    FROM (SELECT Orders.OrderID
                          FROM Orders
                                   inner join OrderDetails OD on Orders.OrderID = OD.OrderID
                          where CustomerID = @CustomerID
                          GROUP BY Orders.OrderID
                          HAVING dbo.GetScalarTotalOrder( @OrderID: Orders.OrderID) > @K1) AS SUM)
        BEGIN
            EXECUTE AddPermanentDiscount  @CustomerID: @CustomerID
        END
END
```

CheckMenuDuplicates

Sprawdza czy przy dodawaniu dania do menu nie występują duplikat

```
Create TRIGGER CheckMenuDuplicates
    on Dishes
    after
        insert
    AS
BEGIN
    Declare @DishDictID INT = (SELECT DishDictID FROM inserted)
    IF 2 = (SELECT count(DishDictID)
            FROM Dishes
            WHERE DishID IN (SELECT DishID FROM ShowCurrentMenu)
              And DishDictID = @DishDictID)
        BEGIN
            THROW 52002,'This dish is already in menu',1
        END
END
```

Indexy

IX_countryname_countries

```
CREATE INDEX ix_countryname_countries
ON Countries(CountryName);
```

IX_cityname_cities

```
CREATE INDEX ix_cityname_cities
ON Cities(CityName)
```

IX_companyname_companies

```
CREATE INDEX ix_companyname_companies
ON Companies(CompanyName)
```

IX_companycity_companies

```
CREATE INDEX ix_companycity_companies
ON Companies(CityID)
```

IX_companyID_employees

```
CREATE INDEX ix_companyID_employees
ON CompaniesWorkers(CompanyID)
```

IX_reservationdate_individualreservations

```
CREATE INDEX ix_reservationdate_indreservations
ON IndividualReservations(ReservationDate)
```

IX_reservationdate_companyreservations

```
CREATE INDEX ix_reservationdate_companyreservations
ON CompanyReservations(ReservationDate)
```

IX_seatamount_tables

```
CREATE INDEX ix_seatamount_tables
ON Tables(SeatAmount)
```

IX_reservationID_companydetails

```
CREATE INDEX ix_reservationID_companydetails
ON CompanyReservationsDetails(ReservationID)
```

IX_reservationID_companypersonaldetails

```
CREATE INDEX ix_reservationID_companypersonaldetails
ON CompanyPersonalReservationsDetails(ReservationDetailID)
```

IX_reservationID_individualreservationapproval

```
CREATE INDEX ix_reservationID_indreservationApproval
ON ReservationsApproval(ReservationID)
```

IX_reservationID_companyreservationapproval

```
CREATE INDEX ix_reservationID_companyreservationapproval
ON CompanyReservationsApproval(ReservationID)
```

IX_categoryName_categories

```
CREATE index ix_categoryName_categories on Categories(CategoryName)
```

IX_caterogyID_dishesDictID

```
CREATE index ix_caterogyID_dishesDictID on DishesDict(CategoryID)
```

IX_dishDictID_dishes

```
CREATE index ix_dishDictID_dishes on Dishes(DishDictID)
```

IX_orderID_orderDetails

```
CREATE index ix_orderID_orderDetails on OrderDetails(OrderID)
```

IX_dishID_orderDetails

```
CREATE index ix_dishID_orderDetails on OrderDetails(DishID)
```

IX_customerID_orders

```
CREATE index ix_customerID_orders on Orders(CustomerID)
```

IX_custostomerID_temporaryDiscounts

```
CREATE index ix_custostomerID_temporaryDiscounts on TemporaryDiscounts(CustomerID)
```

IX_custostomerID_permanentDiscounts

```
CREATE index ix_custostomerID_permanentDiscounts on PermanentDiscounts(CustomerID)
```

IX_r2_temporaryDiscountsDict

```
CREATE index ix_r2_temporaryDiscountsDict on TemporaryDiscountsDict(R2)
```

IX_r1_permanentDiscountsDict

```
CREATE index ix_r1_permanentDiscountsDict on PermanentDiscountsDict(R1)
```

```
CREATE index ix_r1_permanentDiscountsDict on PermanentDiscountsDict(R1)
```

Uprawnienia

Klient Indywidualny

- AddDishToOrder
- AddIndividualReservation
- CancelOrder
- MakeOrder
- PayOrder
- RemoveDishFromOrder
- RemoveIndividualReservation

Firma

- AddCompanyPersonalReservation
- AddCompanyReservation
- AddCompanyReservationDeatil
- AddDishToOrder
- CancelOrder
- MakeOrder
- PayOrder
- RemoveDishFromOrder

Kelner

- AddDishToOrder
- ApproveCompanyReservation
- ApproveReservation
- CancelOrder
- GenerateFacture
- GenerateMonthFacture
- GetCustomerDiscount
- GetTotalOrderValue
- ListCustomerOrders
- MakeOrder
- PayOrder
- RemoveDishFromOrder
- RemoveIndividualReservation

Manager Restauracji

- AddCategory
- AddCity
- AddCompany
- AddCompanyEmployee
- AddCountry
- AddDish
- AddIndividualCustomer
- AddPermanentDiscount
- AddTable
- AddToMenu
- AddWaiter
- ApproveCompanyReservation
- ApproveReservation
- CancelOrder
- CreatePermanentDiscount
- CreateTemporaryDiscount
- GenerateFacture
- GenerateMonthFacture
- DeleteMenuPosition
- FindBestTable
- GetCustomerDiscount
- GetTotalCustomerValue
- GetTotalOrderValue
- ListCustomerOrders
- MakeOrder
- AddDishToOrder
- PayOrder
- RemoveDishFromOrder
- RemoveIndividualReservation

Administrator Systemu

- AddCity
- AddCategory
- AddCompany
- AddCompanyEmployee
- AddCountry
- AddDish
- AddIndividualCustomer
- AddPermanentDiscount
- AddTable
- AddToMenu
- AddWaiter
- CreatePermanentDiscount
- CreateTemporaryDiscount
- GetTotalCustomerValue
- GetTotalOrderValue
- ListCustomerOrders

System

- AddPermanentDiscount
- GetCustomerDiscount
- GetDiscountDuringOrder
- GetScalarTotalCustomerValue
- GetScalarTotalOrder
- ListCustomerOrders