Baza danych restauracji

Mateusz Powęska

Wojciech Przybytek

Przemysław Roman

1) Typy użytkowników

1. Administrator
2. Pracownik restauracji
3. Manager restauracji
4. Klient indywidualny
5. Klient grupowy (firma)

2) Opis funkcji systemu z podziałem na typy użytkowników

1. Administrator

- Dodawanie, usuwanie, edycja użytkowników (wszystkich typów użytkowników)
- Dostęp do wszystkich danych
- Tworzenie kopii zapasowych bazy danych

2. Pracownik restauracji

- Dostęp do historii zamówień i historii menu
- Wystawianie faktur:
    - Faktura dla danego zamówienia
    - Faktura zbiorcza z całego miesiąca
- Rezerwacja stolika:
    - Akceptowanie rezerwacji,
    - Przydzielanie stolika,
    - Wysłanie potwierdzenia rezerwacji.
- Rejestrowanie zamówień składanych na miejscu
- Generowanie raportów (miesięcznych, tygodniowych oraz w podanym zakresie (pomiędzy dwoma datami)) dotyczących:
    - Stolików,
    - Rabatów,
    - Menu,
    - Statystyk zamówień dla klientów indywidualnych oraz grupowych, uwzględniających:
        - Kwoty zamówień,
        - Czas składania zamówień.

3. Manager restauracji (dziedziczy z Pracownik restauracji)

- Dostęp do wszystkich danych
- Ustalanie menu:
    - Dodanie produktu
    - Usunięcie produktu
- Ustalanie wartości
    - Z1, K1, R1%, K2, R2%, D1 związanych z rabatami

- WZ, WK związanych z rezerwacjami
- KRD - liczba dni przed odebraniem zamówienia, w których nie można go anulować
- NT - liczba stolików w lokalu dostępnych dla klientów

4. Klient indywidualny

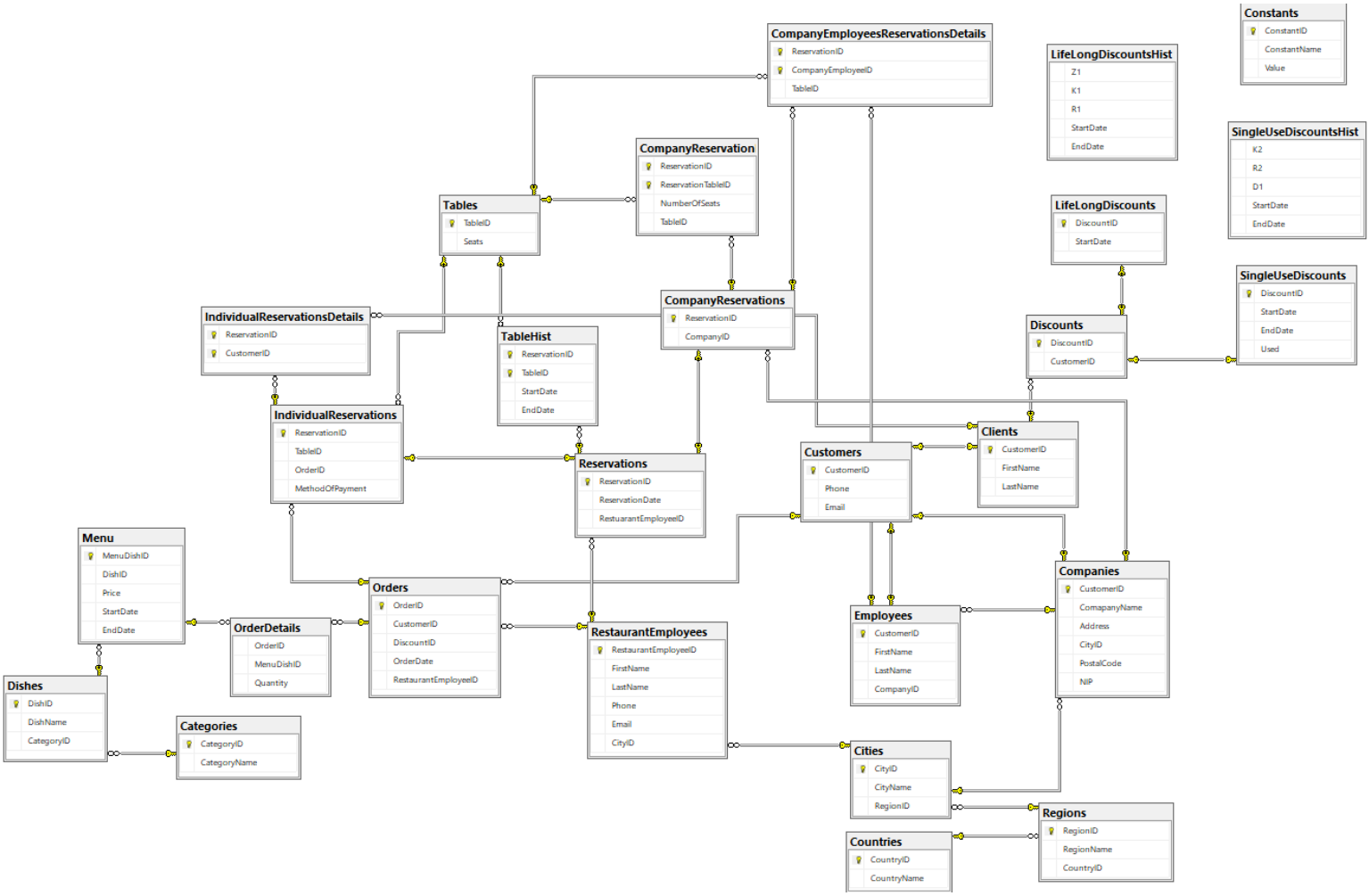
- Złożenie zamówienia przez formularz WWW
  - Forma płatności: przy zamówieniu.
  - Zamówienie na wynos, musi wtedy podać:
    - Preferowaną datę i godzinę odbioru zamówienia,
    - Co jest zamawiane.
  - Zamówienie z rezerwacją stolika, musi wtedy podać:
    - Datę i godzinę rezerwacji,
    - Co jest zamawiane,
  - Zamówienie zawierające owoce morza
    Może być zamawiane na wynos lub z rezerwacją stolika według zasad opisanych dla tych zamówień, dodatkowo:
    - Można je zamówić tylko w czwartek, piątek lub sobotę,
    - Musi być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.
  - Kasowanie rezerwacji (możliwość wykonania tej akcji zależne od KRD)

5. Klient grupowy (firma)

- Rezerwacja stolików:
  - Na firmę
  - Imiennie (dla konkretnych pracowników)

  Dla obu sposobów należy podać:
  - Datę i godzinę rezerwacji,
  - Co jest zamawiane,
  - Rodzaj płatności (przed lub po zamówieniu).

3) Diagram bazy danych



4) Tabele i warunki integralności

<u>Categories</u>

Opis:

Słownik kategorii

Kolumny:

- **(PK)** CategoryID `[int] NOT NULL` - ID kategorii
- CategoryName `[varchar](50) UNIQUE NOT NULL` - nazwa kategorii

```
CREATE TABLE [dbo].[Categories](
	[CategoryID] [int] NOT NULL,
	[CategoryName] [varchar](50) UNIQUE NOT NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
```

```
(
      [CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Cities

Opis:

Słownik miast

Kolumny:

- **(PK)** CityID `[CityID] [varchar](50) NOT NULL` - ID miasta
- CityName `[varchar](50) UNIQUE NOT NULL` - nazwa miasta
- **(FK do Regions.RegionID)** RegionID `[varchar](50) NOT NULL` - ID regionu

```
CREATE TABLE [dbo].[Cities](
      [CityID] [varchar](50) NOT NULL,
      [CityName] [varchar](50) UNIQUE NOT NULL,
      [RegionID] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
      [CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT
[FK_Cities_Regions] FOREIGN KEY([RegionID])
REFERENCES [dbo].[Regions] ([RegionID])
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Regions]
GO
```

Clients

Opis:

Tabela przechowująca informacje o klientach indywidualnych

Kolumny:

- **(PK) (FK do Customers.CustomerID)** CustomerID `[int] NOT NULL` - ID klienta
- FirstName `[varchar](50) NOT NULL` - imie klienta
- LastName `[varchar](50) NOT NULL` - nazwisko klienta

Warunki integralności:

- FirstName składa się z wyrazu rozpoczynającego się wielką literą `CHECK (([FirstName] like '[A-Z][a-z]+'))`
- LastName składa się z jednego wyrazu lub dwóch wyrazów oddzielonych pauzą `CHECK  (([LastName] like '[A-Z][a-z]+([\-][A-z][a-z]*)?'))`

```
CREATE TABLE [dbo].[Clients](
	[CustomerID] [int] NOT NULL,
	[FirstName] [varchar](50) NOT NULL,
	[LastName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Clients] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT
[FK_Clients_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT
[FK_Clients_Customers]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT
[CK_Clients] CHECK  (([FirstName] like '[A-Z][a-z]+'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [CK_Clients]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT
[CK_Clients_1] CHECK  (([LastName] like
'[A-Z][a-z]+([\-][A-z][a-z]*)?'))
```

```
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [CK_Clients_1]
GO
```

CompanyReservations

Opis:

Informacje o rezerwacjach złożonych przez firmy

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID `[int] NOT NULL` - ID rezerwacji
- **(FK)** CompanyID `[int] NOT NULL` - ID firmy

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

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservations_Companies] FOREIGN
KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
[FK_CompanyReservations_Companies]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservations_Reservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
```

```
[FK_CompanyReservations_Reservations]
GO
```

Companies

Opis:

Informacje o firmach, które są klientami restauracji

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** CustomerID `[int] NOT NULL` - ID klienta
- CompanyName `[varchar](50) NOT NULL` - nazwa firmy
- Address `[varchar](50) NOT NULL` - adres firmy
- **(FK do Cities.CityID)** CityID `[varchar](50) NOT NULL` - ID miasta
- PostalCode `[varchar](50) NOT NULL` - kod pocztowy
- NIP `[varchar](50) UNIQUE NOT NULL` - numer identyfikacji podatkowej

Warunki integralności:

- Kod pocztowy spełnia warunki uniwersalnego wyrażenia regularnego na kod pocztowy `CHECK (([PostalCode] like '[a-z0-9][a-z0-9\-]{0,10}[a-z0-9]'))`

```
CREATE TABLE [dbo].[Companies](
	[CustomerID] [int] NOT NULL,
	[CompanyName] [varchar](50) NOT NULL,
	[Address] [varchar](50) NOT NULL,
	[CityID] [varchar](50) NOT NULL,
	[PostalCode] [varchar](50) NOT NULL,
	[NIP] [varchar](50) UNIQUE NOT NULL,
 CONSTRAINT [PK_Companie] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT
[FK_Companies_Companies] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT
[FK_Companies_Companies]
```

```
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT
[FK_Companies_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT
[FK_Companies_Customers]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT
[CK_Companies] CHECK  (([PostalCode] like '[a-z0-9][a-z0-9\-
]{0,10}[a-z0-9]'))
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [CK_Companies]
GO
```

CompanyEmployeesReservationsDetails

Opis:

Informacje o pracownikach firmy na których firmy rezerwowały stoliki

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID `[int] NOT NULL` - ID rezerwacji
- **(PK) (FK do Employees.CustomerID)** CompanyEmployeeID `[int] NOT NULL` - ID pracownika firmy
- **(FK do Tables.TableID)** TableID `[int] NULL` - ID stolika

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

```
ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails]  WITH
CHECK ADD  CONSTRAINT
[FK_CompanyEmployeesReservationsDetails_Employees] FOREIGN
KEY([CompanyEmployeeID])
REFERENCES [dbo].[Employees] ([CustomerID])
GO

ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails] CHECK
CONSTRAINT [FK_CompanyEmployeesReservationsDetails_Employees]
GO

ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails]  WITH
CHECK ADD  CONSTRAINT
[FK_CompanyReservationsDetails_CompanyReservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[CompanyReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails] CHECK
CONSTRAINT [FK_CompanyReservationsDetails_CompanyReservations]
GO

ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails]  WITH
CHECK ADD  CONSTRAINT [FK_CompanyReservationsDetails_Tables]
FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[CompanyEmployeesReservationsDetails] CHECK
CONSTRAINT [FK_CompanyReservationsDetails_Tables]
GO
```

CompanyReservationDetails

Opis:

Informacje o rezerwacjach składanych na firmę

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID `[int] NOT NULL` - ID rezerwacji
- **(PK)** ReservationTableID `[int] NOT NULL` - ID stolika wewnątrz rezerwacji
- NumberOfSeats `[int] NOT NULL` - ilość miejsc siedzących przy stoliku
- **(FK do Tables.TableID)** TableID `[int] NULL` - ID stolika

Warunki integralności:

- Liczba miejsc przy stoliku jest większa od 0 `CHECK (([NumberOfSeats]>(0)))`

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

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationDetails_CompanyReservations]
FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[FK_CompanyReservationDetails_CompanyReservations]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [FK_CompanyReservationDetails_Tables] FOREIGN
KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[FK_CompanyReservationDetails_Tables]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [CK_CompanyReservationDetails] CHECK
(([NumberOfSeats]>(0)))
GO
```

```
ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[CK_CompanyReservationDetails]
GO
```

Constants

Opis:

Tabela przechowuje wartości stałych WZ i WK

Kolumny:

- **(PK)** ConstantID `[int] NOT NULL` - ID stałej
- ConstantName `[varchar](50) NOT NULL` - nazwa stałej
- Value `[int] NOT NULL` - wartość stałej

Warunki integralności:

- ConstantName to WZ lub WK `CHECK  (([ConstantName]='WK' OR [ConstantName]='WZ'))`

```
CREATE TABLE [dbo].[Constants](
      [ConstantID] [int] NOT NULL,
      [ConstantName] [varchar](50) NOT NULL,
      [Value] [int] NOT NULL,
 CONSTRAINT [PK_Constants] PRIMARY KEY CLUSTERED
(
      [ConstantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Constants]  WITH CHECK ADD  CONSTRAINT
[CK_Constants] CHECK  (([Value]>(0)))
GO

ALTER TABLE [dbo].[Constants] CHECK CONSTRAINT [CK_Constants]
GO

ALTER TABLE [dbo].[Constants]  WITH CHECK ADD  CONSTRAINT
[CK_Constants_1] CHECK  (([ConstantName]='WK' OR
[ConstantName]='WZ'))
GO

ALTER TABLE [dbo].[Constants] CHECK CONSTRAINT [CK_Constants_1]
```

```
GO
```

Countries

Opis:

Słownik państw

Kolumny:

- **(PK)** CountryID `[varchar](50) NOT NULL` - ID państwa
- CountryName `[varchar](50) UNIQUE NOT NULL` - nazwa państwa

Warunki integralności:

- Nazwa państwa składa się z od jednego do trzech wyrazów `CHECK (([CountryName] like '[A-Z]\w+\s?[A-Z]?\w*\s?[A-Z]?\w*'))`

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

ALTER TABLE [dbo].[Countries]  WITH CHECK ADD  CONSTRAINT
[CK_Countries] CHECK  (([CountryName] like
'[A-Z]\w+\s?[A-Z]?\w*\s?[A-Z]?\w*'))
GO

ALTER TABLE [dbo].[Countries] CHECK CONSTRAINT [CK_Countries]
GO
```

Customers

Opis:

Informacje o klientach

Kolumny:

- **(PK)** CustomerID `[int] NOT NULL` - ID klienta
- Phone `[varchar](50) NOT NULL` - nr telefonu klienta
- Email `[varchar](50) NOT NULL` - adres email klienta

Warunki integralności:

- Numer telefonu może zawierać znak '+' na początku i składa się z cyfr `CHECK (([Phone] like '[\+]?\d*'))`
- Email zawiera '@' `CHECK  (([Email] like '%@%'))`

```
CREATE TABLE [dbo].[Customers](
	[CustomerID] [int] NOT NULL,
	[Phone] [varchar](50) NOT NULL,
	[Email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT
[CK_Customers] CHECK  (([Email] like '%@%'))
GO

ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_Customers]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT
[CK_Customers_1] CHECK  (([Phone] like '[\+]?\d*'))
GO

ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_Customers_1]
GO
```

Discounts

Opis:

Informacje o tym, kto korzysta ze zniżek

Kolumny:

- **(PK)** DiscountID `[int] NOT NULL` - ID zniżki
- **(FK do Customers.CustomerID)** CustomerID `[int] NOT NULL` - ID klienta, który korzysta z tej zniżki

```
CREATE TABLE [dbo].[Discounts](
	[DiscountID] [int] NOT NULL,
	[CustomerID] [int] NOT NULL,
```

```
 CONSTRAINT [PK_Discount] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discounts]  WITH CHECK ADD  CONSTRAINT
[FK_Discounts_Clients] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Clients] ([CustomerID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT
[FK_Discounts_Clients]
GO
```

Dishes

Opis:

Słownik dań

Kolumny:

- **(PK)** DishID `[int] NOT NULL` - ID dania
- DishName `[varchar](50) NOT NULL` - nazwa dania
- **(FK do Categories.CategoryID)** CategoryID `[int] NOT NULL` - ID kategorii do której należy to danie

```
CREATE TABLE [dbo].[Dishes](
      [DishID] [int] NOT NULL,
      [DishName] [varchar](50) NOT NULL,
      [CategoryID] [int] NOT NULL,
 CONSTRAINT [PK_Dishes] 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]
GO

ALTER TABLE [dbo].[Dishes]  WITH CHECK ADD  CONSTRAINT
[FK_Dishes_Categories] FOREIGN KEY([CategoryID])
```

```
REFERENCES [dbo].[Categories] ([CategoryID])
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Categories]
GO
```

Employees

Opis:

Informacje o pracownikach firm, którzy są klientami restauracji

Kolumny:

- **(PK) (FK do Customers.CustomerID)** CustomerID `[int] NOT NULL` - ID pracownika firmy
- FirstName `[varchar](50) NOT NULL` - imię pracownika
- LastName `[varchar](50) NOT NULL` - nazwisko pracownika
- **(FK do Companies.CustomerID)** CompanyID `[int] NOT NULL` - ID firmy, w której pracuje

Warunki integralności:

- FirstName składa się z wyrazu rozpoczynającego się wielką literą `CHECK (([FirstName] like '[A-Z][a-z]+'))`
- LastName składa się z jednego wyrazu lub dwóch wyrazów oddzielonych pauzą `CHECK (([LastName] like '[A-Z][a-z]+([\-][A-Z][a-z]*)?'))`

```
CREATE TABLE [dbo].[Employees](
	[CustomerID] [int] NOT NULL,
	[FirstName] [varchar](50) NOT NULL,
	[LastName] [varchar](50) NOT NULL,
	[CompanyID] [int] NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Companies] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CustomerID])
GO
```

```
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT
[FK_Employees_Companies]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT
[FK_Employees_Customers]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[CK_Employees] CHECK  (([LastName] like
'[A-Z][a-z]+([\-][A-z][a-z]*)?'))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[CK_Employees_1] CHECK  (([FirstName] like '[A-Z][a-z]+'))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees_1]
GO
```

IndividualReservations

Opis:

Informacje o rezerwacjach klientów indywidualnych

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID `[int] NOT NULL` - ID rezerwacji indywidualnej
- **(FK do Tables.TableID)** TableID `[int] NULL` - ID stolika dla rezerwacji indywidualnej
- **(FK do Orders.OrderID)** OrderID `[int] NULL` - ID zamówienia dla rezerwacji indywidualnej
- MethodOfPayment `[varchar](50) NOT NULL` - metoda płatności przy rezerwacji indywidualnej

Warunki integralności:

- Możliwe płatności to ‘In advance’ oraz ‘At pickup’ `CHECK (([MethodOfPayment]='At pickup' OR [MethodOfPayment]='In advance'))`

```
CREATE TABLE [dbo].[IndividualReservations](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NULL,
	[OrderID] [int] NULL,
	[MethodOfPayment] [varchar](50) 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]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD
CONSTRAINT [FK_IndividualReservations_Orders] FOREIGN
KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[FK_IndividualReservations_Orders]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD
CONSTRAINT [FK_IndividualReservations_Reservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[FK_IndividualReservations_Reservations]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD
CONSTRAINT [FK_IndividualReservations_Tables] FOREIGN
KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO
```

```
ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[FK_IndividualReservations_Tables]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD
CONSTRAINT [CK_IndividualReservations] CHECK
(([MethodOfPayment]='At pickup' OR [MethodOfPayment]='In
advance'))
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[CK_IndividualReservations]
GO
```

LifeLongDiscounts

Opis:

Informacje o zniżkach dożywotnich

Kolumny:

- **(PK) (FK do Discounts.DiscountID)** DiscountID `[int] NOT NULL` - ID dożywotniej zniżki
- StartDate `[datetime] NOT NULL` - data przyznania dożywotniej zniżki

Warunki integralności:

- Data przyznania zniżki nie może być późniejsza niż data dodania rekordu do tabeli `CHECK  (([StartDate]<=getdate()))`

```
CREATE TABLE [dbo].[LifeLongDiscounts](
      [DiscountID] [int] NOT NULL,
      [StartDate] [datetime] NOT NULL,
 CONSTRAINT [PK_LifeLongDiscounts] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[LifeLongDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_LifeLongDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO
```

```
ALTER TABLE [dbo].[LifeLongDiscounts] CHECK CONSTRAINT
[FK_LifeLongDiscounts_Discounts]
GO

ALTER TABLE [dbo].[LifeLongDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_LifeLongDiscounts] CHECK  (([StartDate]<=getdate()))
GO

ALTER TABLE [dbo].[LifeLongDiscounts] CHECK CONSTRAINT
[CK_LifeLongDiscounts]
GO
```

LifeLongDiscountsHist

Opis:

Historia wartości stałych dotyczących zniżek dożywotnich

Kolumny:

- Z1 `[int] NOT NULL` - minimalna liczba zamówień do przyznania zniżki dożywotniej
- K1 `[int] NOT NULL` - minimalna kwota każdego zamówienia do przyznania zniżki dożywotniej
- R1 `[float] NOT NULL` - wartość zniżki dożywotniej
- StartDate `[datetime] NOT NULL` - data od kiedy obowiązywały wartości do uzyskania zniżki dożywotniej
- EndDate `[datetime] NULL` - data do kiedy obowiązywały wartości do uzyskania zniżki dożywotniej

Warunki integralności:

- Wartość Z1 jest większa od 0 `CHECK  (([Z1]>(0)))`
- Wartości K1 jest większa od 0 `CHECK  (([K1]>(0)))`
- Wartości R1 mieści się w zakresie od 0 do 1 `CHECK  (([R1]>=(0) AND [R1]<=(1)))`
- StartDate nie może być późniejsza niż EndDate lub data dodania rekordu do tabeli `CHECK  (([StartDate]<=isnull([EndDate],getdate())))`

```
CREATE TABLE [dbo].[LifeLongDiscountsHist](
      [Z1] [int] NOT NULL,
      [K1] [int] NOT NULL,
      [R1] [float] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[LifeLongDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_LifeLongDiscountsHist] CHECK  (([Z1]>(0)))
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist] CHECK CONSTRAINT
[CK_LifeLongDiscountsHist]
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_LifeLongDiscountsHist_1] CHECK  (([K1]>(0)))
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist] CHECK CONSTRAINT
[CK_LifeLongDiscountsHist_1]
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_LifeLongDiscountsHist_2] CHECK  (([R1]>=(0) AND
[R1]<=(1)))
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist] CHECK CONSTRAINT
[CK_LifeLongDiscountsHist_2]
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_LifeLongDiscountsHist_3] CHECK
(([StartDate]<=isnull([EndDate],getdate())))
GO

ALTER TABLE [dbo].[LifeLongDiscountsHist] CHECK CONSTRAINT
[CK_LifeLongDiscountsHist_3]
GO
```

Menu

Opis:

Tabela zawierająca wszystkie produkty z menu archiwalnych i aktualnego wraz z datą obowiązywania

Kolumny:

- **(PK)** MenuDishID `[int] NOT NULL` - ID dania w menu
- **(FK do Dishes.DishID)** DishID `[int] NOT NULL` - ID dania w słowniku dań

- Price `[money] NOT NULL` - cena dania
- StartDate `[date] NOT NULL` - data od której obowiązuje menu
- EndDate `[date] NULL` - data do której obowiązuje menu

Warunki integralności:

- Cena dania jest większa od 0 `CHECK  (([Price]>(0)))`
- StartDate nie może być późniejsza niż EndDate lub data dodania rekordu do tabeli `CHECK  (([StartDate]<=isnull([EndDate],getdate())))`

```
CREATE TABLE [dbo].[Menu](
	[MenuDishID] [int] NOT NULL,
	[DishID] [int] NOT NULL,
	[Price] [money] NOT NULL,
	[StartDate] [date] NOT NULL,
	[EndDate] [date] NULL,
 CONSTRAINT [PK_Menu] PRIMARY KEY CLUSTERED
(
	[MenuDishID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT
[FK_Menu_Dishes] FOREIGN KEY([DishID])
REFERENCES [dbo].[Dishes] ([DishID])
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Dishes]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [CK_Menu]
CHECK  (([Price]>(0)))
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [CK_Menu]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [CK_Menu_1]
CHECK  (([StartDate]<=isnull([EndDate],getdate())))
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [CK_Menu_1]
```

```
GO
```

OrderDetails

Opis:

Szczegóły związane z zamówieniem

Kolumny:

- **(FK do Orders.OrderID)** OrderID `[int] NOT NULL` - ID zamówienia
- **(FK do Menu.MenuDishID)** MenuDishID `[int] NOT NULL` - ID dania w menu
- Quantity `[int] NOT NULL` - ilość porcji dania

Warunki integralności:

- Wartość Quantity musi być większa od zera `CHECK  (([Quantity]>(0)))`

```
CREATE TABLE [dbo].[OrderDetails](
	[OrderID] [int] NOT NULL,
	[MenuDishID] [int] NOT NULL,
	[Quantity] [int] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Menu] FOREIGN KEY([MenuDishID])
REFERENCES [dbo].[Menu] ([MenuDishID])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Menu]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Orders]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[CK_OrderDetails] CHECK  (([Quantity]>(0)))
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
```

```
[CK_OrderDetails]
GO
```

Orders

Opis:

Lista wszystkich zamówień

Kolumny:

- **(PK)** OrderID `[int] NOT NULL` - ID zamówienia
- **(FK do Customers.CustomerID)** CustomerID `[int] NOT NULL` - ID klienta, który dokonał zamówienia
- **(FK do Discounts.DiscountID)** DiscountID `[int] NULL` - ID zniżki, która została użyta przy zamawianiu
- OrderDate `[date] NOT NULL` - data odbioru zamówienia
- **(FK do RestaurantEmployees.RestaurantEmployeeID)** RestaurantEmployeeID `[int] NOT NULL` - ID pracownika, który przyjął zamówienie

Warunki integralności:

- Data zamówienia musi być ograniczona przez dzisiejszą datę `CHECK (([OrderDate]<=getdate()))`

```
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] NOT NULL,
	[CustomerID] [int] NOT NULL,
	[DiscountID] [int] NULL,
	[OrderDate] [date] NOT NULL,
	[RestaurantEmployeeID] [int] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customers]
GO
```

```
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_RestaurantEmployees] FOREIGN
KEY([RestaurantEmployeeID])
REFERENCES [dbo].[RestaurantEmployees] ([RestaurantEmployeeID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT
[FK_Orders_RestaurantEmployees]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders]
CHECK  (([OrderDate]<=getdate()))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
GO
```

Regions

Opis:

Słownik regionów

Kolumny:

- **(PK)** RegionID `[varchar](50) NOT NULL` - ID rezerwacji
- RegionName `[varchar](50) NOT NULL` - nazwa regionu
- **(FK do Countries.CountryID)** CountryID `[varchar](50) NOT NULL` - ID państwa

```
CREATE TABLE [dbo].[Regions](
      [RegionID] [varchar](50) NOT NULL,
      [RegionName] [varchar](50) NOT NULL,
      [CountryID] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Regions] PRIMARY KEY CLUSTERED
(
      [RegionID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Regions]  WITH CHECK ADD  CONSTRAINT
[FK_Regions_Countries] FOREIGN KEY([CountryID])
```

```
REFERENCES [dbo].[Countries] ([CountryID])
GO

ALTER TABLE [dbo].[Regions] CHECK CONSTRAINT
[FK_Regions_Countries]
GO
```

IndividualReservationsDetails

Opis:

Szczegóły rezerwacji indywidualnych

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID `[int] NOT NULL` - ID rezerwacji
- **(PK) (FK do Customers.CustomerID)** CustomerID `[int] NOT NULL` - ID klienta, który dokonał rezerwacji

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

ALTER TABLE [dbo].[IndividualReservationsDetails]  WITH CHECK ADD
CONSTRAINT [FK_IndivdualReservationsDetails_Clients] FOREIGN
KEY([CustomerID])
REFERENCES [dbo].[Clients] ([CustomerID])
GO

ALTER TABLE [dbo].[IndividualReservationsDetails] CHECK CONSTRAINT
[FK_IndivdualReservationsDetails_Clients]
GO

ALTER TABLE [dbo].[IndividualReservationsDetails]  WITH CHECK ADD
CONSTRAINT
```

```
[FK_IndivdualReservationsDetails_IndividualReservations] FOREIGN
KEY([ReservationID])
REFERENCES [dbo].[IndividualReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[IndividualReservationsDetails] CHECK CONSTRAINT
[FK_IndivdualReservationsDetails_IndividualReservations]
GO
```

Reservations

Opis:

Spis rezerwacji

Kolumny:

- **(PK)** ReservationID `[int] NOT NULL` - ID rezerwacji
- ReservationDate `[datetime] NOT NULL` - data, kiedy dokonano rezerwację
- **(FK do RestaurantEmployees.RestaurantEmployeeID)** RestuarantEmployeeID `[int] NULL` - ID pracownika restauracji, który akceptował rezerwację

Warunki integralności:

- Data kiedy dokonano rezerwacji musi być ograniczona dzisiejszą datą `CHECK (([ReservationDate]<=getdate()))`

```
CREATE TABLE [dbo].[Reservations](
      [ReservationID] [int] NOT NULL,
      [ReservationDate] [datetime] NOT NULL,
      [RestaurantEmployeeID] [int] NULL,
 CONSTRAINT [PK_Resrvations_1] PRIMARY KEY CLUSTERED
(
      [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[FK_Reservations_RestaurantEmployees] FOREIGN
KEY([RestaurantEmployeeID])
REFERENCES [dbo].[RestaurantEmployees] ([RestaurantEmployeeID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
```

```
[FK_Reservations_RestaurantEmployees]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[CK_Reservations] CHECK  (([ReservationDate]<=getdate()))
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[CK_Reservations]
GO
```

RestaurantEmployees

Opis:

Informacje o pracownikach restauracji

Kolumny:

- **(PK)** RestaurantEmployeeID `[int] NOT NULL` - ID pracownika restauracji
- FirstName `[varchar](50) NOT NULL` - imię pracownika restauracji
- LastName `[varchar](50) NOT NULL` - nazwisko pracownika restauracji
- Phone `[varchar](50) NOT NULL` - nr telefonu pracownika restauracji
- Email `[varchar](50) NOT NULL` - adres email pracownika restauracji
- **(FK do Cities.CityID)** CityID `[varchar](50) NOT NULL` - ID miasta, w którym mieszka pracownik restauracji

Warunki integralności:

- FirstName składa się z wyrazu rozpoczynającego się wielką literą `CHECK (([FirstName] like '[A-Z][a-z]+'))`
- LastName składa się z jednego wyrazu lub dwóch wyrazów oddzielonych pauzą `CHECK  (([LastName] like '[A-Z][a-z]+([-][A-z][a-z]*)?'))`
- Email zawiera ‘@’ `CHECK  (([Email] like '%@%'))`
- Numer telefonu może zawierać znak ‘+’ na początku i składa się z cyfr `CHECK (([Phone] like '[\+]?\d*'))`

```
CREATE TABLE [dbo].[RestaurantEmployees](
      [RestaurantEmployeeID] [int] NOT NULL,
      [FirstName] [varchar](50) NOT NULL,
      [LastName] [varchar](50) NOT NULL,
      [Phone] [varchar](50) NOT NULL,
      [Email] [varchar](50) NOT NULL,
      [CityID] [varchar](50) NOT NULL,
 CONSTRAINT [PK_RestaurantEmployees] PRIMARY KEY CLUSTERED
(
      [RestaurantEmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
```

```
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[RestaurantEmployees]  WITH CHECK ADD
CONSTRAINT [FK_RestaurantEmployees_Cities] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[RestaurantEmployees] CHECK CONSTRAINT
[FK_RestaurantEmployees_Cities]
GO

ALTER TABLE [dbo].[RestaurantEmployees]  WITH CHECK ADD
CONSTRAINT [CK_RestaurantEmployees] CHECK  (([FirstName] like
'[A-Z][a-z]+'))
GO

ALTER TABLE [dbo].[RestaurantEmployees] CHECK CONSTRAINT
[CK_RestaurantEmployees]
GO

ALTER TABLE [dbo].[RestaurantEmployees]  WITH CHECK ADD
CONSTRAINT [CK_RestaurantEmployees_1] CHECK  (([LastName] like
'[A-Z][a-z]+([-][A-z][a-z]*)?'))
GO

ALTER TABLE [dbo].[RestaurantEmployees] CHECK CONSTRAINT
[CK_RestaurantEmployees_1]
GO

ALTER TABLE [dbo].[RestaurantEmployees]  WITH CHECK ADD
CONSTRAINT [CK_RestaurantEmployees_2] CHECK  (([Email] like
'%@%'))
GO

ALTER TABLE [dbo].[RestaurantEmployees] CHECK CONSTRAINT
[CK_RestaurantEmployees_2]
GO

ALTER TABLE [dbo].[RestaurantEmployees]  WITH CHECK ADD
```

```
CONSTRAINT [CK_RestaurantEmployees_3] CHECK  (([Phone] like
'[+]?\d*'))
GO

ALTER TABLE [dbo].[RestaurantEmployees] CHECK CONSTRAINT
[CK_RestaurantEmployees_3]
GO
```

SingleUseDiscounts

Opis:

Informacje o jednorazowych zniżkach

Kolumny:

- **(PK) (FK do Discounts.DiscountID)** DiscountID `[int] NOT NULL` - ID zniżki
- StartDate `[datetime] NOT NULL` - data przyznania jednorazowej zniżki
- EndDate `[datetime] NOT NULL` - data, do której obowiązuje jednorazowa zniżka
- Used `[varchar](4) NOT NULL` - wartość logiczna mówiąca o tym, czy wykorzystano jednorazową zniżkę

Warunki integralności:

- StartDate musi być wcześniejsza niż EndDate `CHECK (([StartDate]<[EndDate]))`
- Used może przyjmować tylko wartości FALSE lub TRUE `CHECK (([Used]='FALSE' OR [Used]='TRUE'))`

```
CREATE TABLE [dbo].[SingleUseDiscounts](
      [DiscountID] [int] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NOT NULL,
      [Used] [varchar](4) NOT NULL,
 CONSTRAINT [PK_SingleUseDiscounts] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[SingleUseDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_SingleUseDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO
```

```
ALTER TABLE [dbo].[SingleUseDiscounts] CHECK CONSTRAINT
[FK_SingleUseDiscounts_Discounts]
GO

ALTER TABLE [dbo].[SingleUseDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_SingleUseDiscounts] CHECK  (([StartDate]<[EndDate]))
GO

ALTER TABLE [dbo].[SingleUseDiscounts] CHECK CONSTRAINT
[CK_SingleUseDiscounts]
GO

ALTER TABLE [dbo].[SingleUseDiscounts]  WITH CHECK ADD  CONSTRAINT
[CK_SingleUseDiscounts_1] CHECK  (([Used]='FALSE' OR
[Used]='TRUE'))
GO

ALTER TABLE [dbo].[SingleUseDiscounts] CHECK CONSTRAINT
[CK_SingleUseDiscounts_1]
GO
```

SingleUseDiscountsHist

Opis:

Historia wartości stałych dotyczących zniżek jednorazowych

Kolumny:

- K2 `[int] NOT NULL` - łączna kwota zamówień, po których realizacji przyznawana jest jednorazowa zniżka
- R2 `[float] NOT NULL` - wartość jednorazowej zniżki
- D1 `[int] NOT NULL` - okres ważności jednorazowej zniżki (począwszy od dnia przyznania jej)
- StartDate `[datetime] NOT NULL` - data przyznania jednorazowej zniżki
- EndDate `[datetime] NULL` - data, do której obowiązuje jednorazowa zniżka

Warunki integralności:

- K2 musi być dodatnie `CHECK  (([K2]>(0)))`
- R2 musi należeć do przedziału [0, 1] `CHECK  (([R2]>=(0) AND [R2]<=(1)))`
- D1 musi być dodatnie `(([D1]>(0)))`
- StartDate musi być wcześniejsza lub równa EndDate, trzeba też pokryć sytuację, w której data końcowa nie została podana `CHECK (([StartDate]<=isnull([EndDate],getdate())))`

```
CREATE TABLE [dbo].[SingleUseDiscountsHist](
      [K2] [int] NOT NULL,
      [R2] [float] NOT NULL,
      [D1] [int] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_SingleUseDiscountsHist] CHECK  (([K2]>(0)))
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist] CHECK CONSTRAINT
[CK_SingleUseDiscountsHist]
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_SingleUseDiscountsHist_1] CHECK  (([R2]>=(0) AND
[R2]<=(1)))
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist] CHECK CONSTRAINT
[CK_SingleUseDiscountsHist_1]
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_SingleUseDiscountsHist_2] CHECK  (([D1]>(0)))
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist] CHECK CONSTRAINT
[CK_SingleUseDiscountsHist_2]
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist]  WITH CHECK ADD
CONSTRAINT [CK_SingleUseDiscountsHist_3] CHECK
(([StartDate]<=isnull([EndDate],getdate())))
GO

ALTER TABLE [dbo].[SingleUseDiscountsHist] CHECK CONSTRAINT
[CK_SingleUseDiscountsHist_3]
GO
```

TableHist

Opis:

Historia przydzielania stołów do rezerwacji

Kolumny:

- **(PK) (FK do Reservations.ReservationID)** ReservationID - ID rezerwacji, do której przydzielono stolik
- **(PK) (FK do Tables.TableID)** TableID `[int] NOT NULL` - ID stolika
- StartDate `[int] NOT NULL` - data przydzielenia stolika
- EndDate `[datetime] NOT NULL` - data zwolnienia stolika

Warunki integralności:

- StartDate musi być wcześniejsza lub równa EndDate, trzeba też pokryć sytuację, w której data końcowa nie została podana `CHECK (([StartDate]<=isnull([EndDate],getdate())))`

```
CREATE TABLE [dbo].[TableHist](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
 CONSTRAINT [PK_TableHist] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC,
	[TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TableHist]  WITH CHECK ADD  CONSTRAINT
[FK_TableHist_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[TableHist] CHECK CONSTRAINT
[FK_TableHist_Reservations]
GO

ALTER TABLE [dbo].[TableHist]  WITH CHECK ADD  CONSTRAINT
[FK_TableHist_Tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
```

```
GO

ALTER TABLE [dbo].[TableHist] CHECK CONSTRAINT
[FK_TableHist_Tables]
GO

ALTER TABLE [dbo].[TableHist]  WITH CHECK ADD  CONSTRAINT
[CK_TableHist] CHECK  (([StartDate]<=isnull([EndDate],getdate())))
GO

ALTER TABLE [dbo].[TableHist] CHECK CONSTRAINT [CK_TableHist]
GO
```

Tables

Opis:

Słownik stolików

Kolumny:

- **(PK)** TableID `[int] NOT NULL` - ID stolika
- Seats `[int] NOT NULL` - ilość miejsc siedzących przy stoliku

Warunki integralności:

- Ilość siedzeń musi być dodatnia `CHECK  (([Seats]>(0)))`

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

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [CK_Tables]
CHECK  (([Seats]>(0)))
GO

ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_Tables]
GO
```

5) Widoki

ClientPriceStats

Widok wyświetla tabelę zawierającą statystyki jak często zamawiają klienci indywidualni zamówienia o określonej cenie

```
CREATE VIEW [dbo].[ClientPriceStats]
AS
SELECT FLOOR(OrderPrice / 10) * 10 AS OrderPriceFloor,
CEILING(OrderPrice / 10) * 10 AS OrderPriceCeiling, COUNT(*) AS Quantity
FROM     (SELECT O1.OrderID AS OrderName, SUM(OD1.Quantity * M1.Price) *
(1 -
                                        (SELECT R
                                         FROM
dbo.getClientDiscount(O1.CustomerID, O1.OrderDate) AS
getClientDiscount_1)) AS OrderPrice
                  FROM      dbo.OrderDetails AS OD1 INNER JOIN
                                    dbo.Orders AS O1 ON O1.OrderID =
OD1.OrderID INNER JOIN
                                    dbo.Menu AS M1 ON OD1.MenuDishID =
M1.MenuDishID AND O1.OrderDate BETWEEN M1.StartDate AND M1.EndDate INNER
JOIN
                                    dbo.Clients AS C1 ON C1.CustomerID =
O1.CustomerID
                  GROUP BY O1.OrderID, O1.CustomerID, O1.OrderDate) AS
OrderPrices
GROUP BY FLOOR(OrderPrice / 10) * 10, CEILING(OrderPrice / 10) * 10
GO
```

ClientReservationsRejected

Widok wyświetla tabelę zawierającą wszystkie rezerwacje klientów indywidualnych, które zostały odrzucone

```
create view [dbo].[ClientReservationsRejected] as
select R.ReservationID, R.RestaurantEmployeeID as 'Employee who
rejected'
from Reservations R
inner join IndividualReservations IR on R.ReservationID =
IR.ReservationID
where R.RestaurantEmployeeID is not null and IR.TableID is null
GO
```

ClientsHourStats

Widok wyświetla tabelę zawierającą statystyki jak często zamawiają klienci indywidualni zamówienia w kolejnych godzinach w ciągu dnia

```
create view [dbo].[ClientsHourStats] as
```

```
select DATEPART(HOUR, O.OrderDate) as 'Hour', count(*) as
'NumberOfOrders'
from Orders O
inner join Clients C on O.CustomerID = C.CustomerID
group by DATEPART(HOUR, O.OrderDate)
GO
```

ClientsOrders

Widok wyświetla tabelę zawierającą raport informujący o ilości zamówień przez klientów indywidualnych w poszczególnych miesiącach oraz tygodniach wraz z podsumami

```
SELECT DATEPART(MONTH, O.OrderDate) AS Month, DATEPART(WEEK,
O.OrderDate) AS Week, COUNT(*) AS Count
FROM     dbo.Orders AS O INNER JOIN
                  dbo.Clients ON O.CustomerID = dbo.Clients.CustomerID
GROUP BY DATEPART(MONTH, O.OrderDate), DATEPART(WEEK, O.OrderDate) WITH
ROLLUP
```

CompaniesHourStats

Widok wyświetla tabelę zawierającą statystyki jak często firmy zamawiają w kolejnych godzinach w ciągu dnia

```
create view [dbo].[CompaniesHourStats] as
select DATEPART(HOUR, O.OrderDate) as 'Hour', count(*) as
'NumberOfOrders'
from Orders O
inner join Companies C on O.CustomerID = C.CustomerID
group by DATEPART(HOUR, O.OrderDate)
GO
```

CompaniesPriceStats

Widok wyświetla tabelę zawierającą statystyki jak często firmy zamawiają zamówienia o konkretnych cenach

```
CREATE VIEW [dbo].[CompaniesPriceStats]
AS
SELECT FLOOR(OrderPrice / 10) * 10 AS OrderPriceFloor,
CEILING(OrderPrice / 10) * 10 AS OrderPriceCeiling, COUNT(*) AS Quantity
FROM     (SELECT O1.OrderID AS OrderName, SUM(OD1.Quantity * M1.Price)
AS OrderPrice
                  FROM      dbo.OrderDetails AS OD1 INNER JOIN
                                    dbo.Menu AS M1 ON OD1.MenuDishID =
M1.MenuDishID INNER JOIN
                                    dbo.Orders AS O1 ON O1.OrderID =
OD1.OrderID INNER JOIN
                                    dbo.Companies AS C1 ON C1.CustomerID
```

```
= O1.CustomerID
                  GROUP BY O1.OrderID) AS OrderPrices
WHERE  (OrderPrice > 0)
GROUP BY FLOOR(OrderPrice / 10) * 10, CEILING(OrderPrice / 10) * 10
GO
```

CompanyEmployeesReservationsRejected

Widok wyświetla tabelę zawierającą wszystkie rezerwacje pracowników firm, które zostały odrzucone

```
create view [dbo].[CompanyEmployeesReservationsRejected] as
select distinct R.ReservationID, R.RestaurantEmployeeID as 'Employee who
rejected'
from Reservations R
inner join CompanyEmployeesReservationsDetails C on R.ReservationID =
C.ReservationID
where R.RestaurantEmployeeID is not null and C.TableID is null
GO
```

CompanyOrders

Widok wyświetla tabelę zawierającą raport informujący o ilości zamówień składanych przez firmy w poszczególnych miesiącach oraz tygodniach wraz z podsumami

```
SELECT DATEPART(MONTH, O.OrderDate) AS Month, DATEPART(WEEK,
O.OrderDate) AS Week, COUNT(*) AS Count
FROM     dbo.Orders AS O INNER JOIN
                  dbo.Companies ON O.CustomerID =
dbo.Companies.CustomerID
GROUP BY DATEPART(MONTH, O.OrderDate), DATEPART(WEEK, O.OrderDate) WITH
ROLLUP
```

CompanyReservationsRejected

Widok wyświetla tabelę zawierającą wszystkie rezerwacje firm, które zostały odrzucone

```
create view [dbo].[CompanyReservationsRejected] as
select R.ReservationID, R.RestaurantEmployeeID as 'Employee who
rejected'
from Reservations R
inner join CompanyReservationDetails CRD on R.ReservationID =
CRD.ReservationID
where R.RestaurantEmployeeID is not null and CRD.TableID is null
GO
```

LifeLongDiscountsMonthlyReport

Widok wyświetla tabelę zawierającą raport dotyczący zniżek dożywotnich przyznanych w tym miesiącu

```
CREATE VIEW [dbo].[LifeLongDiscountsMonthlyReport]
AS
SELECT dbo.Discounts.DiscountID, LLD.StartDate,
                      (SELECT Z1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS Z1,
                      (SELECT K1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS K1,
                      (SELECT R1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS R1
FROM     dbo.Discounts INNER JOIN
                  dbo.LifeLongDiscounts AS LLD ON
dbo.Discounts.DiscountID = LLD.DiscountID
WHERE  (DATEDIFF(month, LLD.StartDate, GETDATE()) < 1)
GO`
```

LifeLongDiscountsWeeklyReport

Widok wyświetla tabelę zawierającą raport dotyczący zniżek dożywotnich przyznanych w tym tygodniu

```
CREATE VIEW [dbo].[LifeLongDiscountsWeeklyReport]
AS
SELECT dbo.Discounts.DiscountID, LLD.StartDate,
                      (SELECT Z1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS Z1,
                      (SELECT K1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS K1,
                      (SELECT R1
                       FROM      dbo.LifeLongDiscountsHist AS _LLD
                       WHERE   (StartDate <= LLD.StartDate) AND
(LLD.StartDate <= EndDate)) AS R1
FROM     dbo.Discounts INNER JOIN
                  dbo.LifeLongDiscounts AS LLD ON
dbo.Discounts.DiscountID = LLD.DiscountID
```

```
WHERE  (DATEDIFF(week, LLD.StartDate, GETDATE()) < 1)
GO
```

MenuCurrentReport

Widok wyświetla tabelę zawierającą aktualne menu

```
create view [dbo].[MenuCurrentReport] as
select MenuDishID, D.DishID, DishName, C.CategoryID, CategoryName,
Price, StartDate
from Menu
inner join Dishes D on D.DishID = Menu.DishID
inner join Categories C on D.CategoryID = C.CategoryID
where (Menu.EndDate is null)
GOS
```

MenuMonthlyReport

Widok wyświetla tabelę zawierającą miesięczny raport dotyczący menu

```
create view [dbo].[MenuMonthlyReport] as
select MenuDishID, D.DishID, DishName, C.CategoryID, CategoryName,
Price, StartDate, EndDate
from Menu
inner join Dishes D on D.DishID = Menu.DishID
inner join Categories C on D.CategoryID = C.CategoryID
where (datediff(month, StartDate, getdate()) < 1)
GO
```

MenuWeeklyReport

Widok wyświetla tabelę zawierającą tygodniowy raport dotyczący menu

```
create view [dbo].[MenuWeeklyReport] as
select MenuDishID, D.DishID, DishName, C.CategoryID, CategoryName,
Price, StartDate, EndDate
from Menu
inner join Dishes D on D.DishID = Menu.DishID
inner join Categories C on D.CategoryID = C.CategoryID
where (datediff(week, StartDate, getdate()) < 1)
GO
```

ReservationsPending

Widok wyświetla tablicę zawierającą rezerwacje, które nie zostały jeszcze zatwierdzone lub odrzucone

```
create view [dbo].[ReservationsPending] as
select R.ReservationID
from Reservations R
```

```
where R.RestaurantEmployeeID is null
GO
```

SingleUseDiscountsMonthlyReport

Widok wyświetla tabelę zawierającą miesięczny raport dotyczący zniżek jednorazowych

```
create view [dbo].[SingleUseDiscountsMonthlyReport] as
select Discounts.DiscountID, SUD.StartDate, SUD.EndDate,
       (select K2 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
K2,
       (select R2 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
R2,
       (select D1 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
D1,
       Used
from Discounts
inner join SingleUseDiscounts SUD on Discounts.DiscountID =
SUD.DiscountID
where (datediff(month, StartDate, getdate()) < 1)
GO
```

SingleUseDiscountsWeeklyReport

Widok wyświetla tabelę zawierającą tygodniowy raport dotyczący zniżek jednorazowych

```
create view [dbo].[SingleUseDiscountsWeeklyReport] as
select Discounts.DiscountID, SUD.StartDate, SUD.EndDate,
       (select K2 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
K2,
       (select R2 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
R2,
       (select D1 from SingleUseDiscountsHist as _SUD where
_SUD.StartDate <= SUD.StartDate and SUD.StartDate <= _SUD.EndDate) as
D1,
       Used
from Discounts
inner join SingleUseDiscounts SUD on Discounts.DiscountID =
SUD.DiscountID
where (datediff(week, StartDate, getdate()) < 1)
GO
```

TablesReservationsCounts

Widok wyświetla tabelę zawierającą raport informujący jak często był rezerwowany dany stolik w poszczególnych miesiącach oraz tygodniach wraz z podsumami

```
SELECT TOP (100) PERCENT TableID, DATEPART(MONTH, EndDate) AS Month,
DATEPART(WEEK, EndDate) AS Week, COUNT(*) AS ReservationsCount
FROM     dbo.TableHist
GROUP BY TableID, DATEPART(MONTH, EndDate), DATEPART(WEEK, EndDate) WITH
ROLLUP
ORDER BY TableID
```

TablesReservationsWeeklyReport

Widok wyświetla tabelę zawierającą tygodniowy raport informujący o wszystkich rezerwacjach

```
CREATE VIEW [dbo].[TablesReservationsWeeklyReport]
AS
SELECT TableID, StartDate, EndDate
FROM     dbo.TableHist
WHERE  (DATEDIFF(week, EndDate, GETDATE()) < 1)
GO
```

TablesReservationsMonthlyReport

Widok wyświetla tabelę zawierającą miesięczny raport informujący o wszystkich rezerwacjach

```
CREATE VIEW [dbo].[TablesReservationsMonthlyReport]
AS
SELECT TableID, StartDate, EndDate
FROM     dbo.TableHist
WHERE  (DATEDIFF(month, EndDate, GETDATE()) < 1)
GO
```

LifeLongDiscountsReport

Raport miesięczny i tygodniowy dotyczący zniżek dożywotnich

```
create view LifeLongDiscountReport as
select DATEPART(MONTH, D.StartDate) AS Month, DATEPART(WEEK,
D.StartDate) AS Week, COUNT(*) AS Count
from LifeLongDiscounts D
GROUP BY DATEPART(MONTH, D.StartDate), DATEPART(WEEK, D.StartDate)
WITH ROLLUP
GO
```

SingleUseDiscountsReport

Raport miesięczny i tygodniowy dotyczący zniżek jednorazowych

```
create view SingleUseDiscountReport as
select DATEPART(MONTH, D.StartDate) AS Month, DATEPART(WEEK,
D.StartDate) AS Week, COUNT(*) AS Count
from SingleUseDiscounts D
GROUP BY DATEPART(MONTH, D.StartDate), DATEPART(WEEK, D.StartDate)
WITH ROLLUP
GO
```

TableReservationsReport

Raport tygodniowy i miesięczny dotyczący rezerwacji stolików

```
create view TableReservationsReport as
select DATEPART(MONTH, D.StartDate) AS Month, DATEPART(WEEK,
D.StartDate) AS Week, COUNT(*) AS Count
from TableHist D
GROUP BY DATEPART(MONTH, D.StartDate), DATEPART(WEEK, D.StartDate)
WITH ROLLUP
GO
```

6) Funkcje

getCategory

Funkcja zwraca nazwę kategorii dania

```
create function [dbo].[getCategory] (@DishID INT)
returns table as
return (
      select CategoryName
      from Categories
      inner join Dishes on Categories.CategoryID = Dishes.CategoryID
      where Dishes.DishID = @DishID
)
GO
```

getClientLifelongDiscount

Funkcja zwraca informacje na temat zniżki dożywotniej danego klienta

```
create function [dbo].[getClientLifelongDiscount] (@ClientID INT)
returns table as
return (
      select D.DiscountID,
      (select Z1 from dbo.getLifelongDiscountValuesAt(LLD.StartDate)) as
Z1,
      (select K1 from dbo.getLifelongDiscountValuesAt(LLD.StartDate)) as
K1,
```

```
	(select R1 from dbo.getLifelongDiscountValuesAt(LLD.StartDate)) as
R1,
LLD.StartDate
	from Discounts D
	inner join LifeLongDiscounts LLD on D.DiscountID = LLD.DiscountID
	where D.CustomerID = @ClientID
)
GO
```

getClientOrders

Funkcja zwraca informacje na temat wszystkich zamówień danego klienta

```
create function [dbo].[getClientOrders] (@ClientID INT)
returns table as
return (
	select O.OrderID, sum(OD.Quantity * M.Price) as 'Order Price'
	from Orders O
	inner join OrderDetails OD on O.OrderID = OD.OrderID
	inner join Menu M on OD.MenuDishID = M.MenuDishID
		and O.OrderDate between M.StartDate and M.EndDate
	where O.CustomerID = @ClientID
	group by O.OrderID
)
GO
```

getClientReservations

Funkcja zwraca wszystkie rezerwacje klienta indywidualnego

```
create function [dbo].[getClientReservations] (@ClientID INT)
returns table as
return (
	select IR.ReservationID, IR.TableID, IR.MethodOfPayment,
R.ReservationDate
	from IndividualReservations IR
	inner join IndividualReservationsDetails IRD on IR.ReservationID =
IRD.ReservationID
	inner join Reservations R on IR.ReservationID = R.ReservationID
	where IRD.CustomerID = @ClientID
)
GO
```

getClientSingleUseDiscount

Funkcja zwraca informacje na temat wszystkich zniżek jednorazowych danego klienta

```
create function [dbo].[getClientSingleUseDiscount] (@ClientID INT)
returns table as
```

```
return (
      select D.DiscountID,
      (select K2 from dbo.getSingleUseDiscountValuesAt(SUD.StartDate))
as K2,
      (select R2 from dbo.getSingleUseDiscountValuesAt(SUD.StartDate))
as R2,
      (select D1 from dbo.getSingleUseDiscountValuesAt(SUD.StartDate))
as D1,
SUD.StartDate,
SUD.EndDate
      from Discounts D
      inner join SingleuseDiscounts SUD on D.DiscountID = SUD.DiscountID
      where D.CustomerID = @ClientID
)
GO
```

getCompanyEmployeeReservations

Funkcja zwraca wszystkie rezerwacje pracownika firmy

```
create function [dbo].[getCompanyEmployeeReservations] (@EmployeeID INT)
returns table as
return (
      select CERD.ReservationID, R.ReservationDate,
R.RestaurantEmployeeID
      from CompanyEmployeesReservationsDetails CERD
      inner join Reservations R on CERD.ReservationID = R.ReservationID
      where CERD.CompanyEmployeeID = @EmployeeID
)
GO
```

getCompanyOrders

Funkcja zwraca informacje na temat wszystkich zamówień danej firmy

```
create function [dbo].[getCompanyOrders] (@CompanyID INT)
returns table as
return (
      select O.OrderID, sum(OD.Quantity * M.Price) as 'Order Price'
      from Orders O
      inner join OrderDetails OD on O.OrderID = OD.OrderID
      inner join Menu M on OD.MenuDishID = M.MenuDishID
            and O.OrderDate between M.StartDate and M.EndDate
      where O.CustomerID = @CompanyID
      group by O.OrderID
)
```

```
GO
```

getCompanyReservations

Funkcja zwraca wszystkie rezerwacje firmy

```
create function [dbo].[getCompanyReservations] (@CompanyID INT)
returns table as
return (
      select CR.ReservationID, count(CRD.ReservationTableID) as 'Tables
Count', R.ReservationDate, R.RestaurantEmployeeID
      from Reservations R
      inner join CompanyReservations CR on R.ReservationID =
CR.ReservationID
      inner join CompanyReservationDetails CRD on CRD.ReservationID =
CR.ReservationID
      where CR.CompanyID = @CompanyID
      group by CR.ReservationID, R.ReservationDate,
R.RestaurantEmployeeID
)
GO
```

getLifelongDiscountValuesAt

Funkcja zwraca tablicę stałych dotyczących zniżek dożywotnich obowiązujących w danym czasie

```
create function [dbo].[getLifelongDiscountValuesAt] (@Date datetime)
returns table as
return (
      select Z1, K1, R1 from LifeLongDiscountsHist
      where @Date between StartDate and EndDate
)
GO
```

getSingleUseDiscountValuesAt

Funkcja zwraca tablicę stałych dotyczących zniżek jednorazowych obowiązujących w danym czasie

```
create function [dbo].[getSingleUseDiscountValuesAt] (@Date datetime)
returns table as
return (
      select K2, R2, D1 from SingleUseDiscountsHist
      where @Date between StartDate and EndDate
)
GO
```

getClientDiscount

Zwraca zniżkę, która została użyta do zamówienia, jeżeli klient nie miał zniżki zwraca 0.

```
Create function [dbo].[getClientDiscount](@ClientID INT, @OrderDate
datetime)
returns table as
return (
      select top 1 R from (
            select R2 as R from getClientSingleUseDiscount(@ClientID)
            where @OrderDate between StartDate and EndDate
            UNION
            select R1 as R from getClientLifelongDiscount(@ClientID)
            where @OrderDate > StartDate
            Union
            select 0 as R
      ) as R
)
GO
```

containsSeafood

Zwraca wszystkie dania z zamówienia, które są owocami morza

```
create function containsSeafood (@OrderDishes OrderDishesType READONLY)
returns table as
return (
      select O.MenuDishID from @OrderDishes O
      inner join Menu M on M.MenuDishID = O.MenuDishID
      inner join Dishes D on M.DishID = D.DishID
      inner join Categories C on D.CategoryID = C.CategoryID
      where C.CategoryName = 'Seafood'
)
```

getOrderValue

Zwraca wartość nowego zamówienia

```
CREATE function [dbo].[getOrderValue] (@OrderDishes NewOrderDishesType
READONLY)
returns table as
return (
      select sum(O.Quantity * M.Price) as 'Order Price'
      from @OrderDishes O
      inner join Menu M on O.MenuDishID = M.MenuDishID
)
```

7) Procedury

SwapMenu

Procedura wymienia pozycję z menu na nową

```
CREATE procedure [dbo].[SwapMenu] @OldMenuDishID int, @MenuDishID int,
@DishID int,
                            @Price money, @StartDate datetime,
                            @EndDate datetime
as
update Menu set EndDate=@StartDate where MenuDishID=@OldMenuDishID;
execute AddMenuItem @MenuDishID, @DishID, @Price, @StartDate,
    @EndDate
GO
```

AddCity

Procedura dodaje miasto do tabeli Cities

```
CREATE procedure [dbo].[AddCity] @CityID varchar(50), @CityName
varchar(50), @RegionID varchar(50)
as
insert into Cities([CityID], [CityName], [RegionID])
Values (@CityID, @CityName, @RegionID)
go
```

AddClient

Procedura dodaje klientów indywidualnych do tabeli Clients

```
CREATE procedure [dbo].[AddClient] @CustomerID int = null, @FirstName
varchar(50) = null,
               @LastName varchar(50) = null
as
insert into Clients([CustomerID], [FirstName], [LastName])
Values(@CustomerID, @FirstName, @LastName)
go
```

AddCompany

Procedura dodaje firmę do tabeli Companies

```
create procedure AddCompany @CustomerID int, @CompanyName varchar(50),
@Address varchar(50), @CityID varchar(50),
                            @PostalCode varchar(50), @NIP varchar(50)
as
insert into Companies([CustomerID], [CompanyName], [Address], [CityID],
[PostalCode], [NIP])
Values (@CustomerID, @CompanyName, @Address, @CityID, @PostalCode, @NIP)
go
```

AddCompanyEmployeeReservationsDetails

Procedura dodaje szczegóły rezerwacji firmy (imiennie na pracownika firmy) do tabeli CompanyEmployeesReservationsDetails

```
CREATE procedure AddCompanyEmployeeReservationDetails @ReservationID
int, @CompanyEmployeeID int, @TableID int
as
insert into CompanyEmployeesReservationsDetails([ReservationID],
[CompanyEmployeeID], [TableID])
Values(@ReservationID, @CompanyEmployeeID, @TableID)
go
```

AddCompanyReservation

Procedura dodaje rezerwację dla firm do tabeli CompanyReservations

```
create procedure AddCompanyReservation @ReservationID int, @CompanyID
int
as
insert into CompanyReservations([ReservationID], [CompanyID])
Values(@ReservationID, @CompanyID)
go
```

AddCompanyReservationDetails

Procedura dodaje szczegóły rezerwacji dla firmy do tabeli AddCompanyReservationDetails

```
CREATE procedure AddCompanyReservationDetails @ReservationID int,
@ReservationTableID int, @NumberOfSeats int, @TableID int
as
insert into CompanyReservationDetails([ReservationID],
[ReservationTableID], [NumberOfSeats], [TableID])
Values(@ReservationID, @ReservationTableID, @NumberOfSeats, @TableID)
go
```

AddCountry

Procedura dodaje kraj do tabeli Countries

```
CREATE procedure AddCountry @CountryID varchar(50), @CountryName
varchar(50)
as
begin
insert into Countries([CountryID], [CountryName])
values(@CountryID, @CountryName)
end
go
```

AddCustomer

Procedura dodaje klientów do tabeli Customers

```
CREATE procedure [dbo].[AddCustomer] @CustomerID int, @Phone
varchar(50),
                            @Email varchar(50)
as
insert into Customers([CustomerID], [Phone], [Email])
Values (@CustomerID, @Phone, @Email)
go
```

AddDiscount

Procedura dodaje zniżki do tabeli Discounts

```
create procedure AddDiscount @DiscountID int, @CustomerID int
as
insert into Discounts([DiscountID], [CustomerID])
Values(@DiscountID, @CustomerID)
go
```

AddDish

Procedura dodaje danie do tabeli Dishes

```
CREATE procedure [dbo].[AddDish] @DishID int, @DishName varchar(50),
@CategoryID int
as
insert into Dishes([DishID], [DishName], [CategoryID])
Values(@DishID, @DishName, @CategoryID)
go
```

AddDishCategory

Procedura dodaje kategorię dania do tabeli Categories

```
CREATE procedure AddDishCategory @CategoryID int, @CategoryName varchar
as
insert into Categories([CategoryID], [CategoryName]) Values(@CategoryID,
@CategoryName)
Go
```

AddEmployee

Procedura dodaje pracownika do tabeli Employees

```
create procedure AddEmployee @CustomerID int, @FirstName varchar(50),
               @LastName varchar, @CompanyID int
as
insert into Employees([CustomerID], [FirstName], [LastName],
[CompanyID])
```

```
Values(@CustomerID, @FirstName, @LastName, @CompanyID)
go
```

AddIndividualReservation

Procedura dodaje indywidualną rezerwację do tabeli IndividualReservations

```
create procedure AddIndividualReservation @ReservationID int, @TableID
int = null, @OrderID int = null, @MethodOfPayment varchar
as
insert into IndividualReservations([ReservationID], [TableID],
[OrderID], [MethodOfPayment])
Values(@ReservationID, @TableID, @OrderID, @MethodOfPayment)
go
```

AddIndividualReservationDetails

Procedura dodaje szczegóły rezerwacji indywidualnej do tabeli IndividualReservationsDetails

```
create procedure AddIndividualReservationDetails @ReservationID int,
@CustomerID int
as
insert into IndividualReservationsDetails([ReservationID], [CustomerID])
Values(@ReservationID, @CustomerID)
go
```

AddLifeLongDiscount

Procedura dodaje zniżki dożywotnie do tabeli LifeLongDiscounts

```
create procedure AddLifeLongDiscount @DiscountID int = null, @StartDate
datetime = null
as
insert into LifeLongDiscounts([DiscountID], [StartDate])
Values(@DiscountID, @StartDate)
go
```

AddLifeLongDiscountHist

Procedura dodaje parametry zniżek dożywotnich do tabeli LifeLongDiscountsHist

```
CREATE procedure AddLifeLongDiscountHist @Z1 int, @K1 int, @R1 float,
                                         @StartDate datetime, @EndDate
datetime
as
insert into LifeLongDiscountsHist([Z1], [K1], [R1], [StartDate],
[EndDate])
Values (@Z1, @K1, @R1, @StartDate, @EndDate)
go
```

AddMenuItem

Procedura dodaje danie do tabeli Menu

```
CREATE procedure AddMenuItem @MenuDishID int, @DishID int,
                             @Price money, @StartDate datetime,
                             @EndDate datetime = null
as
insert into Menu([MenuDishID], [DishID], [Price], [StartDate],
[EndDate])
Values (@MenuDishID, @DishID, @Price, @StartDate, @EndDate)
Go
```

AddOrder

Procedura dodaje zamówienie do tabeli Orders

```
create procedure AddOrder @OrderID int, @CustomerID int,
                          @DiscountID int, @OrderDate date,
@RestaurantEmployeeID int
as
insert into Orders([OrderID], [CustomerID], [DiscountID], [OrderDate],
[RestaurantEmployeeID])
Values (@OrderID, @CustomerID, @DiscountID, @OrderDate,
@RestaurantEmployeeID)
Go
```

AddOrderDetails

Procedura dodaje szczegóły zamówienia do tabeli OrderDetails

```
create procedure AddOrderDetails @OrderID int, @MenuDishID int,
@Quantity int
as
insert into OrderDetails([OrderID], [MenuDishID], [Quantity])
Values (@OrderID, @MenuDishID, @Quantity)
go
```

AddRegion

Procedura dodaje region do tabeli Regions

```
CREATE procedure [dbo].[AddRegion] @RegionID varchar(50), @RegionName
varchar(50),
                           @CountryID varchar(50)
as
insert into Regions([RegionID], [RegionName], [CountryID])
Values (@RegionID, @RegionName, @CountryID)
Go
```

AddReservation

Procedura dodaje rezerwację do tabeli Reservations

```
CREATE procedure [dbo].[AddReservation] @ReservationID int,
@ReservationDate datetime, @RestaurantEmployeeID int
as
insert into Reservations([ReservationID], [ReservationDate],
[RestaurantEmployeeID])
Values (@ReservationID, @ReservationDate, @RestaurantEmployeeID)
go
```

AddRestaurantEmployee

Procedura dodaje pracownika restauracji do tabeli RestaurantEmployees

```
CREATE procedure [dbo].[AddRestaurantEmployee] @RestaurantEmployeeID
int, @FirstName varchar(50),
                                    @LastName varchar(50), @Phone
varchar(50), @Email varchar(50), @CityID varchar(50)
as
insert into RestaurantEmployees([RestaurantEmployeeID], [FirstName],
[LastName], [Phone], [Email], [CityID])
Values (@RestaurantEmployeeID, @FirstName, @LastName, @Phone, @Email,
@CityID)
go
```

AddSingleUseDiscount

Procedura dodaje zniżki jednorazowe do tabeli SingleUseDiscounts

```
create procedure AddSingleUseDiscount @DiscountID int = null, @StartDate
datetime = null,
                            @EndDate datetime = null, @Used varchar(50) =
null
as
insert into SingleUseDiscounts([DiscountID], [StartDate], [EndDate],
[Used])
Values (@DiscountID, @StartDate,@EndDate, @Used)
go
```

AddSingleUseDiscountHist

Procedura dodaje parametry zniżek jednorazowych do tabeli SingleUseDiscountsHist

```
create procedure AddSingleUseDiscountHist @K2 int, @R2 float,
                            @D1 int, @StartDate datetime, @EndDate
datetime
as
insert into SingleUseDiscountsHist([K2], [R2], [D1], [StartDate],
```

```
[EndDate])
Values (@K2, @R2,@D1, @StartDate,@EndDate )
go
```

AddTable

Procedura dodaje stolik to tabeli Tables

```
CREATE procedure AddTable @TableID int = null, @Seats int = null
as
insert into Tables([TableID], [Seats])
Values (@TableID, @Seats)
go
```

AddTableHist

Procedura dodaje informacje dotyczące stolika do tabeli TableHist

```
CREATE procedure [dbo].[AddTableHist] @ReservationID int, @TableID int,
                            @StartDate datetime, @EndDate datetime
as
insert into TableHist([ReservationID], [TableID], [StartDate],
                      [EndDate])
Values (@ReservationID, @TableID, @StartDate, @EndDate)
go
```

AddConstant

Procedura dodaje stałe do tabeli Constants

```
CREATE procedure [dbo].[AddConstant] @ConstantID int, @ConstantName
varchar(50), @Value int
as
insert into Constants([ConstantID], [ConstantName], [Value])
Values (@ConstantID, @ConstantName, @Value)
go
```

NewClientReservation

Procedura sprawdza czy rezerwacja spełnia warunki i dodaje ją do bazy

```
CREATE procedure [dbo].[NewClientReservation] @CustomerID int, @OrderID
int, @ReservationID int, @ReservationDate datetime, @OrderDishes
NewOrderDishesType READONLY
as
begin
declare @canOrder varchar(50)
      set @canOrder = iif(((select count(*) from
getClientOrders(@CustomerID)) >=
                        (select Value from Constants where ConstantName =
```

```
'WK')) and
                        ((select [Order Price] from
getOrderValue(@OrderDishes)) >=
                        (select Value from Constants where ConstantName =
'WZ')) and
                        ((select count(*) from
containsSeafood(@OrderDishes)) = 0 or ( datepart(dw, @ReservationDate)
in (5,6,7) and datediff(day, @ReservationDate, getdate()) >= 3)),
                  ('true'),
                  'false')
      if (@canOrder = 'true')
      begin
            exec AddReservation @ReservationID, @ReservationDate, null
            exec AddOrder @OrderID, @CustomerID, @ReservationDate, null
            insert into OrderDetails select * from @OrderDishes
      end
end
go
```

RemoveRecordsFromTables

Procedura usuwa wszystkie dane z bazy

```
create procedure RemoveRecordsFromTables
as
delete from Companies
delete from CompanyEmployeesReservationsDetails
delete from CompanyReservationDetails
delete from CompanyReservations
delete from Constants
delete from Customers
delete from Discounts
delete from Dishes
delete from Employees
delete from IndividualReservations
delete from IndividualReservationsDetails
delete from LifeLongDiscounts
delete from LifeLongDiscountsHist
delete from Menu
delete from OrderDetails
delete from Orders
delete from Reservations
delete from RestaurantEmployees
delete from SingleUseDiscountsHist
delete from SingleUseDiscounts
delete from TableHist
delete from Tables
```

```
delete from Categories
delete from Cities
delete from Regions
delete from Countries
delete from Clients
go
```

8) Triggery

OnReservationDelete

Trigger ma na celu usunięcie rekordów związanych z usuwaną rezerwacją, znajdujących się w tabelach korzystających z id rezerwacji.

```
create trigger OnReservationDelete on Reservations
after delete
as
begin
   delete from IndividualReservations where ReservationID in (select
deleted.ReservationID from deleted)
   delete from IndividualReservationsDetails where ReservationID in
(select deleted.ReservationID from deleted)
   delete from CompanyReservations where ReservationID in (select
deleted.ReservationID from deleted)
   delete from CompanyEmployeesReservationsDetails where ReservationID
in (select deleted.ReservationID from deleted)
end
```

OnOrderDelete

Trigger usuwa zamówienie z tabeli OrderDetails po usunięciu zamówienia w tabeli Orders.

```
create trigger OnOrderDelete on Orders
after delete
as
begin
  delete from OrderDetails where OrderID in (select deleted.OrderID from
deleted)
end
```

9) Indeksy

Orders_CustomerID

ID klientów, którzy złożyli kiedyś zamówienie
Tabela: Orders
Kolumna: CustomerID

```
create index Orders_CustomerID
on Orders (CustomerID)
```

IndividualReservationsDetails_CustomerID

ID klientów indywidualnych, którzy złożyli kiedyś rezerwację
Tabela: IndividualReservationsDetails
Kolumna: CustomerID

```
create index IndividualReservationsDetails_CustomerID
on IndividualReservationsDetails (CustomerID)
```

CompanyReservations_CompanyID

ID firm, które złożyły kiedyś rezerwację
Tabela: CompanyReservations
Kolumna: CustomerID

```
create index CompanyReservations_CompanyID
on CompanyReservations (CompanyID)
```

CompanyEmployeesReservationsDetails_CompanyEmployeeID

ID pracowników firm, którzy złożyli kiedyś rezerwację
Tabela: CompanyEmployeesReservationsDetails
Kolumna: CompanyEmployeeID

```
create index CompanyEmployeesReservationsDetails_CompanyEmployeeID
on CompanyEmployeesReservationsDetails (CompanyEmployeeID)
```

Discounts_CustomerID

ID klientów indywidualnych, którzy mają jakąś zniżkę
Tabela: Discounts
Kolumna: CustomerID

```
create index Discounts_CustomerID
on Discounts (CustomerID)
```

Constants_Values

Wartości stałych WZ, WK
Tabela: Constants
Kolumna: Values

```
create index Constants_Values
on Constants (ConstantName, Value)
```

Companies_CustomerID

ID klientów, którzy są firmami

Tabela: Companies
Kolumna: CustomerID

```
Create unique index Companies_CustomerID
on Companies (CustomerID)
```

Clients_CustomerID

ID klientów, którzy są klientami indywidualnymi
Tabela: Clients
Kolumna: CustomerID

```
Create unique index Clients_CustomerID
on Clients (CustomerID)
```

Employees_CustomerID

ID klientów, którzy są pracownikami firm
Tabela: Employees
Kolumna: CustomerID

```
Create unique index CompanyEmployees_CustomerID
on Employees (CustomerID)
```

Categories_CategoryName

Nazwy wszystkich kategorii
Tabela: Categories
Kolumna: CategoryName

```
create unique index Categories_CategoryName
on Categories (CategoryName)
```

10) Role

Admin

Administrator systemu, ma dostęp do wszystkich uprawnień

```
create role Admin
grant all privileges to Admin
```

Manager

Manager restauracji, ma uprawnienia select, insert, update w całej bazie

```
create role Manager
grant select, insert, update to Manager
```

Employee

Pracownik restauracji ma uprawnienia select oraz do insert i update w tabelach: Orders, OrderDetails, Reservations, IndividualReservations, IndividualReservationsDetails,

CompanyReservations, CompanyReservationDetails, CompanyEmployeesReservationsDetails

```
create role Employee
grant select to Employee
grant insert on Orders to Employee
grant insert on OrderDetails to Employee
grant insert on Reservations to Employee
grant insert on IndividualReservations to Employee
grant insert on IndividualReservationsDetails to Employee
grant insert on CompanyReservations to Employee
grant insert on CompanyReservationDetails to Employee
grant insert on CompanyEmployeesReservationsDetails to Employee
grant update on Orders to Employee
grant update on OrderDetails to Employee
grant update on Reservations to Employee
grant update on IndividualReservations to Employee
grant update on IndividualReservationsDetails to Employee
grant update on CompanyReservations to Employee
grant update on CompanyReservationDetails to Employee
grant update on CompanyEmployeesReservationsDetails to Employee
```

11) Generowanie danych

Do wygenerowania danych napisaliśmy skrypt w języku Python, którego kod znajduje się tutaj.