-

Podstawy baz danych

Projekt: Restauracje

Mateusz Skowron
Jolanta Śliwa
Arkadiusz Tyrański

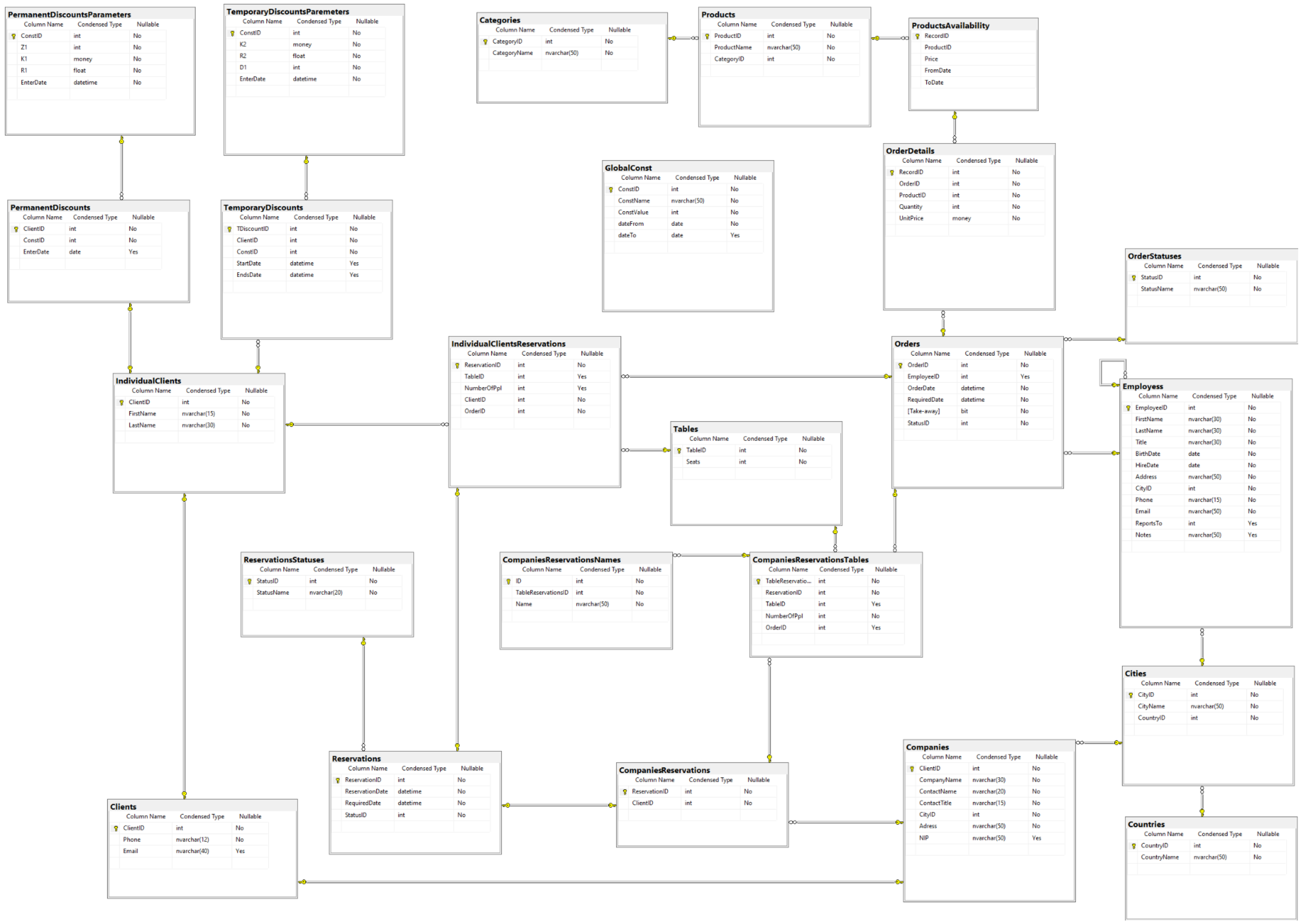
Spis treści

1. Role i funkcje użytkowników systemu

1. **system/automat**
    a. sprawdzić, czy po upływie danego czasu pozycje w menu zostały podmienione
    b. przydziela stolik przy rezerwacji formularzem (i informuje, gdy nie ma żadnego dostępnego)
    c. generowanie regularnych raportów
    d. naliczanie rabatów
2. **administrator**
    a. dostęp do wszystkiego
    b. możliwość edycji rekordów
    c. możliwość edycji tabel
3. **właściciel**
    a. dodawanie pracownika
    b. wprowadzanie rabatów
    c. modyfikowanie rabatów
    d. dodaj danie do menu
    e. modyfikuj danie w menu (zmień cenę, oznacz jako niedostępne)
    f. usuń danie z menu
    g. wprowadź dane klienta biznesowego
    h. generowanie raportu z zamówieniami z zadanego okresu
4. **pracownik**
    a. przyjęcie zamówienia do realizacji (zaakceptowanie zamówienia)
    b. modyfikowanie zamówienia
    c. anulowanie zamówienia
    d. akceptowanie rezerwacji
    e. modyfikowanie rezerwacji (zmiana stolika, zmiana ilości miejsc)
    f. anulowanie rezerwacji
    g. wystawianie rachunku/faktury (sporządzenie raportu z zamówienia)
    h. modyfikowanie rachunku (naliczanie rabatów)
    i. przyjęcie płatności za zamówienie
    j. dostęp do historii zamówień z danego okresu
5. **kierownik zmiany** (rozszerza funkcje pracownika)
    a. udzielanie rabatu
    b. generowanie raportu z zamówieniami z zadanego okresu
    c. modyfikuj danie w menu (zmień cenę, oznacz jako niedostępne)
6. **klient indywidualny**
    a. rezerwacja (dla minimum 2 osób), przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu i anulowanie rezerwacji
    b. możliwość generowania raportów dotyczących zamówień oraz rabatów
    c. zlecenie zamówienia na wynos za pomocą formularza WWW - wybór preferowanej daty i godziny odbioru zamówienia
    d. dostęp do pozycji w menu
7. **Klient biznesowy** (firma)

a. Zbiorowa rezerwacja miejsc przy jednoczesnym złożeniu zamówienia oraz ich anulowanie ,a także jednorazowe opłacenie. Rezerwacja stolików możliwa jest w dwóch opcjach: rezerwacja stolików na firmę i/lub dla konkretnych pracowników (imiennie).
b. Generowania raportów okresowych.
c. Złożenie zamówienia poprzez formularz WWW.

2. Schemat



PermanentDiscountsParameters

Column Name	Condensed Type	Nullable
ConstID	int	No
Z1	int	No
K1	money	No
R1	float	No
EnterDate	datetime	No

TemporaryDiscountsParemeters

Column Name	Condensed Type	Nullable
ConstID	int	No
K2	money	No
R2	float	No
D1	int	No
EnterDate	datetime	No

Categories

Column Name	Condensed Type	Nullable
CategoryID	int	No
CategoryName	nvarchar(50)	No

Products

Column Name	Condensed Type	Nullable
ProductID	int	No
ProductName	nvarchar(50)	No
CategoryID	int	No

ProductsAvailability

Column Name
RecordID
ProductID
Price
FromDate
ToDate

GlobalConst

Column Name	Condensed Type	Nullable
ConstID	int	No
ConstName	nvarchar(50)	No
ConstValue	int	No
dateFrom	date	No
dateTo	date	Yes

OrderDetails

Column Name	Condensed Type	Nullable
RecordID	int	No
OrderID	int	No
ProductID	int	No
Quantity	int	No
UnitPrice	money	No

PermanentDiscounts

Column Name	Condensed Type	Nullable
ClientID	int	No
ConstID	int	No
EnterDate	date	Yes

TemporaryDiscounts

Column Name	Condensed Type	Nullable
TDiscountID	int	No
ClientID	int	No
ConstID	int	No
StartDate	datetime	Yes
EndsDate	datetime	Yes

OrderStatuses

Column Name	Condensed Type	Nullable
StatusID	int	No
StatusName	nvarchar(50)	No

IndividualClientsReservations

Column Name	Condensed Type	Nullable
ReservationID	int	No
TableID	int	Yes
NumberOfPpl	int	Yes
ClientID	int	No
OrderID	int	No

Orders

Column Name	Condensed Type	Nullable
OrderID	int	No
EmployeeID	int	Yes
OrderDate	datetime	No
RequiredDate	datetime	No
[Take-away]	bit	No
StatusID	int	No

IndividualClients

Column Name	Condensed Type	Nullable
ClientID	int	No
FirstName	nvarchar(15)	No
LastName	nvarchar(30)	No

Employess

Column Name	Condensed Type	Nullable
EmployeeID	int	No
FirstName	nvarchar(30)	No
LastName	nvarchar(30)	No
Title	nvarchar(30)	No
BirthDate	date	No
HireDate	date	No
Address	nvarchar(50)	No
CityID	int	No
Phone	nvarchar(15)	No
Email	nvarchar(50)	No
ReportsTo	int	Yes
Notes	nvarchar(50)	Yes

Tables

Column Name	Condensed Type	Nullable
TableID	int	No
Seats	int	No

ReservationsStatuses

Column Name	Condensed Type	Nullable
StatusID	int	No
StatusName	nvarchar(20)	No

CompaniesReservationsNames

Column Name	Condensed Type	Nullable
ID	int	No
TableReservationsID	int	No
Name	nvarchar(50)	No

CompaniesReservationsTables

Column Name	Condensed Type	Nullable
TableReservatio...	int	No
ReservationID	int	No
TableID	int	Yes
NumberOfPpl	int	No
OrderID	int	Yes

Cities

Column Name	Condensed Type	Nullable
CityID	int	No
CityName	nvarchar(50)	No
CountryID	int	No

Reservations

Column Name	Condensed Type	Nullable
ReservationID	int	No
ReservationDate	datetime	No
RequiredDate	datetime	No
StatusID	int	No

CompaniesReservations

Column Name	Condensed Type	Nullable
ReservationID	int	No
ClientID	int	No

Companies

Column Name	Condensed Type	Nullable
ClientID	int	No
CompanyName	nvarchar(30)	No
ContactName	nvarchar(20)	No
ContactTitle	nvarchar(15)	No
CityID	int	No
Adress	nvarchar(50)	No
NIP	nvarchar(50)	Yes

Clients

Column Name	Condensed Type	Nullable
ClientID	int	No
Phone	nvarchar(12)	No
Email	nvarchar(40)	Yes

Countries

Column Name	Condensed Type	Nullable
CountryID	int	No
CountryName	nvarchar(50)	No

3.Tabele

1. **Clients**

   Przechowuje informacje o klientach.
   - **ClientID** [int] [NOT NULL] - Identyfikator klienta, wartość auto inkrementowana (klucz główny).
   - **Phone** [nvarchar(12)] [NOT NULL] - numer telefonu.
   - **Email** [nvarchar(20)] [NULL] - adres email.

   Warunki integralności:
   - Email jest unikalny i zawiera “@”.
     - **CHECK (([Email] like '%@%'))**
     - **CONSTRAINT [CK2_Email] UNIQUE NONCLUSTERED**
   - Phone jest unikalny i składa się wyłącznie z cyfr.
     - **CHECK ((isnumeric([Phone])=(1)))**
     - **CONSTRAINT [CK2_Phone] UNIQUE NONCLUSTERED**

```
CREATE TABLE [dbo].[Clients](
	[ClientID] [int] IDENTITY(1,1) NOT NULL,
	[Phone] [nvarchar](12) NOT NULL,
	[Email] [nvarchar](40) NULL,
 CONSTRAINT [PK_Clients] PRIMARY KEY CLUSTERED
(
	[ClientID] 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 [CK2_Email] UNIQUE NONCLUSTERED
(
	[Email] 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 [CK2_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],
 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]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [CK_Clients] CHECK
((isnumeric([Phone])=(1)))
GO

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

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [CK_Clients_1] CHECK  (([Email] like
'%@%'))
GO
```

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

2. **IndividualClients**
   Przechowuje informacje dotyczące klientów indywidualnych:
   - **ClientID** [int] [NOT NULL] - Identyfikator klienta (klucz obcy z Clients)
   - **FirstName** [nvarchar(15)] [NOT NULL] - imię klienta
   - **LastName** [nvarchar(30)] [NOT NULL] - nazwisko klienta
   - **DiscountBalance** [money] - liczba wydanych pieniędzy do kolejnej zniżki jednorazowej

   Warunki integralności:
   - FirstName zawiera tylko litery
     - CHECK (([FirstName] like '[A-Za-z]%'))
   - LastName zawiera tylko litery
     - CHECK (([LastName] like '[A-Za-z]%'))

```
CREATE TABLE [dbo].[IndividualClients](
        [ClientID] [int] NOT NULL,
        [FirstName] [nvarchar](15) NOT NULL,
        [LastName] [nvarchar](30) NOT NULL,
       [DiscountBalance] [money] NOT NULL,
 CONSTRAINT [PK_IndividualClients] PRIMARY KEY CLUSTERED
(
        [ClientID] 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].[IndividualClients]  WITH CHECK ADD  CONSTRAINT [FK_IndividualClients_Clients]
FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[IndividualClients] ADD  CONSTRAINT [DF_IndividualClients_DiscountBalance]
DEFAULT (0) FOR [DiscountBalance]
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT [FK_IndividualClients_Clients]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT [CK_IndividualClients] CHECK
(([FirstName] like  '[A-Za-z]%'))
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT [CK_IndividualClients]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT [CK_IndividualClients_1]
CHECK  (([LastName] like  '[A-Za-z]%'))
GO
```

```
ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT [CK_IndividualClients_1]
GO
```

3. **Companies**
   Przechowuje szczegółowe informacje dotyczące klientów biznesowych.
   - **ClientID** [int] [NOT NULL] - klucz obcy (z tabeli Clients) określający numer ID firmy jako klienta i klucz główny
   - **CompanyName** [nvarchar(30)] [NOT NULL] - nazwa firmy
   - **NIP** [nvarchar(10)] [NULL] - numer NIP firmy
   - **ContactName** [nvarchar(20)] [NOT NULL] - imię osoby kontaktowej.
   - **ContactTitle** [nvarchar(15)] [NOT NULL] - tytuł osoby kontaktowej.
   - **CityID** [int] [NOT NULL]- klucz obcy (z tabeli City) do tabeli z nazwami miast
   - **Address** [nvarchar(50)] [NOT NULL] - adres firmy

   Warunki integralnościowe:
   - NIP - unikalny, składa się wyłącznie z cyfr
     - **CHECK ((isnumeric([NIP])=(1)))**
     - **CONSTRAINT [CK2_NIP] UNIQUE NONCLUSTERED**

```
CREATE TABLE [dbo].[Companies](
	[ClientID] [int] NOT NULL,
	[CompanyName] [nvarchar](30) NOT NULL,
	[ContactName] [nvarchar](20) NOT NULL,
	[ContactTitle] [nvarchar](15) NOT NULL,
	[CityID] [int] NOT NULL,
	[Adress] [nvarchar](50) NOT NULL,
	[NIP] [nvarchar](50) NULL,
 CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
	[ClientID] 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 [CK2_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]
GO

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

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

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK_Companies_Clients] FOREIGN
KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
```

```
GO

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

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [CK_Companies] CHECK
((isnumeric([NIP])=(1)))
GO

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

4. Reservations

Przechowuje podstawowe informacje o rezerwacjach.

- **ReservationID** [int] [NOT NULL] - klucz główny
- **ClientID** [int] [NOT NULL] - ID klienta (z tabeli Clients), do którego jest przypisana
- **ReservationDate** [date] [NOT NULL]- data złożenia rezerwacji
- **RequiredDate** [date] [NOT NULL] - data do kiedy rezerwacja jest ważna
- **StatusID** [int] [NOT NULL] - klucz obcy (do tabeli ReservationStatuses), ze statusami

Warunki integralnościowe:

- ReservationDate - domyślnie obecna data dodania rezerwacji.
  - **DEFAULT (getdate()) FOR [ReservationDate**
- RequiredDate jest datą późniejszą/tą samą ReservationDate.
  - **CHECK (([ReservationDate]>=[RequiredDate]))**

```
CREATE TABLE [dbo].[Reservations](
	[ReservationID] [int] IDENTITY(1,1) NOT NULL,
	[ReservationDate] [date] NOT NULL,
	[RequiredDate] [date] NOT NULL,
	[StatusID] [int] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO
 ALTER TABLE [dbo].[Reservations] ADD  CONSTRAINT [DF_Reservations_ReservationDate]  DEFAULT (getdate()) FOR
[ReservationDate]
GO

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

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

```
GO

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

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_IndividualClientsReservations]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_ReservationsStatuses]
FOREIGN KEY([StatusID])
REFERENCES [dbo].[ReservationsStatuses] ([StatusID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_ReservationsStatuses]
GO

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

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

5. **IndividualClientsReservations**
   Tabela przechowuje szczegóły rezerwacji dla klienta indywidualnego.
   - **ReservationID** [int] [NOT NULL] - klucz obcy(z tabeli Reservations) i jednocześnie klucz główny
   - **ClientID** [int] [NOT NULL] - klucz obcy (z tabeli Clients) z ID klienta, do którego jest przypisana
   - **OrderID** [int] [NOT NULL] - klucz obcy (z tabeli Orders) z ID zamówienia, które zostało złożone przy rezerwacji
   - **NumberOfPeople** [int] [NULL] - liczba osób, na jaką dokonano rezerwacji
   - **TableID** [int] [NULL]- klucz obcy (do tabeli Tables)

   Warunki integralnościowe:
   - NumberOfPelople jest większe równe 2.
     - `CHECK  (([NumberOfPpl]>(1)))`

```
CREATE TABLE [dbo].[IndividualClientsReservations](
        [ReservationID] [int] NOT NULL,
        [TableID] [int] NULL,
        [NumberOfPpl] [int] NOT NULL,
        [ClientID] [int] NOT NULL,
        [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_IndividualClientsReservations] 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].[IndividualClientsReservations]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualClientsReservations_IndividualClients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
GO

ALTER TABLE [dbo].[IndividualClientsReservations] CHECK CONSTRAINT
[FK_IndividualClientsReservations_IndividualClients]
GO

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

ALTER TABLE [dbo].[IndividualClientsReservations] CHECK CONSTRAINT
[FK_IndividualClientsReservations_Orders]
GO

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

ALTER TABLE [dbo].[IndividualClientsReservations] CHECK CONSTRAINT
[FK_IndividualClientsReservations_Tables]
GO

ALTER TABLE [dbo].[IndividualClientsReservations]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualClientsReservations] CHECK  (([NumberOfPpl]>(1)))
GO

ALTER TABLE [dbo].[IndividualClientsReservations] CHECK CONSTRAINT
[CK_IndividualClientsReservations]
GO
```

6. **CompaniesReservationsNames**
   Tabela przechowująca dane osoby, na którą dokonano rezerwacji dla firmy (jeżeli zarezerwowano na nazwisko).
   - **ID** [int] [NOT NULL] - klucz główny tabeli
   - **TableReservationsID** [int] [NOT NULL] - klucz obcy (z tabeli Reservations) z ID rezerwacji
   - **Name** [nvarchar(50)] [NOT NULL] - imię i nazwisko osoby wyszczególnionej w rezerwacji

```
CREATE TABLE [dbo].[CompaniesReservationsNames](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[TableReservationsID] [int] NOT NULL,
	[Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_CompaniesReservationsNames] PRIMARY KEY CLUSTERED
(
	[ID] 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].[CompaniesReservationsNames]  WITH CHECK ADD  CONSTRAINT
[FK_CompaniesReservationsNames_CompaniesReservationsTables] FOREIGN KEY([TableReservationsID])
REFERENCES [dbo].[CompaniesReservationsTables] ([TableReservationsID])
GO

ALTER TABLE [dbo].[CompaniesReservationsNames] CHECK CONSTRAINT
[FK_CompaniesReservationsNames_CompaniesReservationsTables]
GO
```

7. **CompaniesReservations**
   Tabela przejściowa pozwalająca połączyć klientów z rezerwacjami.
   - **ReservationID** [int] [NOT NULL] - klucz obcy (z tabeli Reservations) i główny tabeli
   - **ClientID** [int] [NOT NULL] - klucz obcy (z tabeli Clients) z ID klienta, którego dotyczy rezerwacja

```
CREATE TABLE [dbo].[CompaniesReservations](
	[ReservationID] [int] NOT NULL,
	[ClientID] [int] NOT NULL,
 CONSTRAINT [PK_CompaniesReservations] 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].[CompaniesReservations]  WITH CHECK ADD  CONSTRAINT
[FK_CompaniesReservations_Companies] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Companies] ([ClientID])
GO

ALTER TABLE [dbo].[CompaniesReservations] CHECK CONSTRAINT
[FK_CompaniesReservations_Companies]
GO
```

8. **CompaniesReservationsTables**
   Tabela zawierająca informacje jakie stoliki zostały przydzielone dla rezerwacji dla firmy.RecordID
   - **ReservationID** [int] [NOT NULL] - klucz obcy (z tabeli Reservations)

- **TableReservationsID** [int] {not null} - klucz główny
- **TableID** [int] [NULL] - klucz obcy (z tabeli Tables), z id stolika, który jest przypisany dla danej rezerwacji
- **NumberOfPpl** [int] [NOT NULL] - liczba osób, na którą dokonano rezerwacji.
- **OrderID** [int] [NULL] - klucz obcy (z tabeli Orders) określająca zamówienie

Warunki integralnościowe:

- NumberOfPpl jest większe od 1
    - CHECK (([NumbeerOfPp]>(1)))

```
CREATE TABLE [dbo].[CompaniesReservationsTables](
	[TableReservationsID] [int] IDENTITY(1,1) NOT NULL,
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NULL,
	[NumberOfPpl] [int] NOT NULL,
	[OrderID] [int] NULL,
 CONSTRAINT [PK_CompaniesReservationsTables] PRIMARY KEY CLUSTERED
(
	[TableReservationsID] 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].[CompaniesReservationsTables]  WITH CHECK ADD  CONSTRAINT
[FK_CompaniesReservationsTables_CompaniesReservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompaniesReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompaniesReservationsTables] CHECK CONSTRAINT
[FK_CompaniesReservationsTables_CompaniesReservations]
GO

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

ALTER TABLE [dbo].[CompaniesReservationsTables] CHECK CONSTRAINT
[FK_CompaniesReservationsTables_Orders]
GO

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

ALTER TABLE [dbo].[CompaniesReservationsTables] CHECK CONSTRAINT
[FK_CompaniesReservationsTables_Tables]
GO
```

9. Countries

Tabela zawierająca informacje na temat państw.

- **CountryID** [int] [NOT NULL] - klucz główny tabeli
- **CountryName** [nvarchar(50)] [NOT NULL] - nazwa kraju

Warunki integralnościowe:

- Nazwa państwa jest unikalna.
  - **CONSTRAINT [U_CountryName] UNIQUE NONCLUSTERED**
- CountryName tylko litery.
  - **CHECK (([CountryName] like '[a-zA-Z]%'))**

```
CREATE TABLE [dbo].[Countries](
	[CountryID] [int] IDENTITY(1,1) NOT NULL,
	[CountryName] [nvarchar](50) 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]
 CONSTRAINT [U_CountryName] 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]
GO

ALTER TABLE [dbo].[Countries]  WITH CHECK ADD  CONSTRAINT [CK_Countries] CHECK
(([CountryName] like '[a-zA-Z]%'))
GO

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

10. Cities

Tabela zawierająca informacje na temat miast.

- **CityID** [int][NOT NULL] - klucz główny tabeli
- **CityName** [nvarchar(50)] [NOT NULL]- nazwa własna miasta
- **CountryID** [int] [NOT NULL] - identyfikator państwa w którym znajduje się dane miasto, klucz obcy z tabeli Country

Warunki integralności:

- Nazwa miasta składa się tylko z liter.
  - **CHECK (([CityName] like '[A-Za-z]%'))**

```
CREATE TABLE [dbo].[Cities](
	[CityID] [int] IDENTITY(1,1) NOT NULL,
	[CityName] [nvarchar](50) NOT NULL,
	[CountryID] [int] 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_Countries] FOREIGN
KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Countries]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT [CK_CityName] CHECK  (( [CityName] like
'[a-zA-Z]%'))
GO

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

11. Products

Tabela zawierająca informacje na temat oferowanych produktów.

- **ProductID** [int] [NOT NULL]- klucz główny tabeli
- **ProductName** [nvarchar(50)] [NOT NULL]- nazwa produktu
- **CategoryID** [int] [NOT NULL]- identyfikator kategori do której należy dany produkt, klucz obcy z tabeli Categories

Warunki integralności:

- Nazwa produktu jest unikalna.
  - `CONSTRAINT [U_ProductName] UNIQUE NONCLUSTERED`

```
CREATE TABLE [dbo].[Products](
	[ProductID] [int] IDENTITY(1,1) NOT NULL,
	[ProductName] [nvarchar](50) NOT NULL,
	[CategoryID] [int] NOT NULL,
 CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
	[ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [U_ProductName] UNIQUE NONCLUSTERED
(
	[ProductName] 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].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_Categories] FOREIGN
KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_Categories]
GO
```

12. Categories

Tabela słownikowa zawierająca nazwy kategori produktówi.

- **CategoryID** (int)[NOT NULL] - klucz główny tabeli
- **CategoryName** (nvarchar) [NOT NULL]- nazwa kategori

Warunki integralności:

- Nazwa kategorii jest unikalna.
  - `CONSTRAINT [U_CategoryName] UNIQUE NONCLUSTERED`
- Nazwa kategorii składa się tylko z liter
  - `CHECK  ((NOT [CategoryName] like '[a-zA-Z]%'))`

```
CREATE TABLE [dbo].[Categories](
	[CategoryID] [int] IDENTITY(1,1) NOT NULL,
	[CategoryName] [nvarchar](50) 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],
 CONSTRAINT [U_CategoryName] 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]
GO

ALTER TABLE [dbo].[Categories]  WITH CHECK ADD  CONSTRAINT [CK_Categories] CHECK  ((NOT
[CategoryName] like '[a-zA-Z]%'))
GO

ALTER TABLE [dbo].[Categories] CHECK CONSTRAINT [CK_Categories]
GO
```

13. ProductAvailability

Tabela rejestrująca kiedy dane danie było dostępne i po jakiej cenie. (ToDate) - data do której dany produkt był dostępny po danej cenie

- **RecordID** [int] [NOT NULL]- klucz główny
- **ProductID** [int] [NOT NULL]- produkt, klucz obcy z tabeli Products
- **Price** [int] [NOT NULL]- cena produktu
- **FromDate** [date][NOT NULL] - data od której dany produkt był/jest/będzie dostępny
- **ToDate** [date] [NULL]- data do której dany produkt był/jest/będzie dostępny

Warunki integralności:

- Cena produktu musi być większa od zera.
  - `CHECK  (([Price]>(0)))`
- Data końca dostępności następuje po dacie wprowadzenia produktu lub data końca dostępności jest nullem - nie ustalono jeszcze daty wycofania produktu.
  - `CHECK  (([ToDate]>[FromDate] OR [ToDate] IS NULL))`

```
CREATE TABLE [dbo].[ProductsAvailability](
	[RecordID] [int] IDENTITY(1,1) NOT NULL,
	[ProductID] [int] NOT NULL,
	[Price] [money] NOT NULL,
	[FromDate] [date] NOT NULL,
	[ToDate] [date] NULL,
 CONSTRAINT [PK_ProductsAvailability] PRIMARY KEY CLUSTERED
(
	[RecordID] 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].[ProductsAvailability]  WITH CHECK ADD  CONSTRAINT
[FK_ProductsAvailability_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[ProductsAvailability] CHECK CONSTRAINT [FK_ProductsAvailability_Products]
GO

ALTER TABLE [dbo].[ProductsAvailability]  WITH CHECK ADD  CONSTRAINT [CK_FromToDate] CHECK
(([ToDate]>[FromDate] OR [ToDate] IS NULL))
GO

ALTER TABLE [dbo].[ProductsAvailability] CHECK CONSTRAINT [CK_FromToDate]
GO

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

ALTER TABLE [dbo].[ProductsAvailability] CHECK CONSTRAINT [CK_Price]
GO
```

14. Orders

Tabela rejestrująca wszystkie składane zamówienia.

- **OrderID** [int] [NOT NULL]- klucz główny, numer zamówienia
- **EmployeeID** [int] [NULL]- klucz obcy z tabeli Employee indentyfikujący pracownika, który obsługiwał dane zamówienie
- **OrderDate** [date] [NOT NULL]- data złożenia zamówienia
- **RequiredDate** [date] [NOT NULL]- data realizacji zamówienia
- **Take-away** [bit] [NOT NULL]- informacja czy zamówienie jest na wynos czy na miejscu
- **StatusID** [int] [NOT NULL]- informacja na temat statusu zamówienia, klucz obcy z tabeli Orderstatuses

Warunki integralności:

- Data realizacji następuje po dacie złożenia zamówienia lub w tym samym dniu.

- CHECK (([RequiredDate]>=[OrderDate]))

```
CREATE TABLE [dbo].[Orders](
	[OrderID] [int] IDENTITY(1,1) NOT NULL,
	[EmployeeID] [int] NULL,
	[OrderDate] [datetime] NOT NULL,
	[RequiredDate] [datetime] NOT NULL,
	[Take-away] [bit] NOT NULL,
	[StatusID] [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_Employess] FOREIGN
KEY([EmployeeID])
REFERENCES [dbo].[Employess] ([EmployeeID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_Orders_Statuses] FOREIGN
KEY([StatusID])
REFERENCES [dbo].[OrderStatuses] ([StatusID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Required_Order_Date] CHECK
(([RequiredDate]>=[OrderDate]))
GO

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

15. OrderDetails

Tabela zawierająca szczegóły dotyczące danego zamówienia.

- **RecordID** [int] [NOT NULL] - klucz główny tabeli
- **OrderID** [int] [NOT NULL]- zamówienie do którego odnoszą się informacje, klucz obcy z tabeli Orders
- **ProductID** [int] [NOT NULL] - zamówiony produkt, klucz obcy z tabeli ProductAvilability (RecordID)
- **Quantity** [int] [NOT NULL] - ilość zamówionego produkty
- **UnitPrice** [money] [NOT NULL] - cena jednostkowa produktu

Warunki integralności:

- Zamówiona ilość produktów musi być większa od zera.

        ○ CHECK ((\[Quantity\]>(0)))
- Cena produktu musi być większa od zera.
        ○ CHECK ((\[UnitPrice\]>(0)))

```sql
CREATE TABLE [dbo].[OrderDetails](
	[RecordID] [int] IDENTITY(1,1) NOT NULL,
	[OrderID] [int] NOT NULL,
	[ProductID] [int] NOT NULL,
	[Quantity] [int] NOT NULL,
	[UnitPrice] [money] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
	[RecordID] 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].[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
[FK_OrderDetails_ProductsAvailability] FOREIGN KEY([ProductID])
REFERENCES [dbo].[ProductsAvailability] ([RecordID])
GO

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


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

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

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

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

16. Employees

Tabela zawierająca informacje o pracownikach restauracji.

- ○ **EmployeeID**[int] [NOT NULL]- klucz główny
- ○ **FirstName**[nvarchar(30)] [NOT NULL]- imię pracownika

- **LastName**[nvarchar(30)] [NOT NULL]- nazwisko pracownika
- **Title**[nvarchar(30)] [NOT NULL]- stanowisko pracownika
- **BirthDate**[date] [NOT NULL]- data urodzenia pracownika
- **HireDate**[date] [NOT NULL]- data zatrudnienia pracownika
- **Address**[nvarchar(50)][NOT NULL] - adres zamieszkania
- **CityID**[int][NOT NULL] - miasto w którym mieszka pracownik przyporządkowane do odpowiedniego państwa, klucz obcy
- **Phone**[nvarchar(15)][NOT NULL] - numer telefonu pracownika
- **Email**[nvarchar(50)][NOT NULL] - adres email pracownika
- **ReportsTo**[int][NULL] - Inny pracownik który jest przełożonym, klucz z tej tabeli (EmployeeID)
- **Notes**[nvarchar(50)][NULL] - wszelkie notatki na temat pracownika

Warunki integralności:

- Imię składa się jedynie z liter.
  - **CHECK (([FirstName] like '[a-zA-Z]%'))**
- Nazwisko składa się jedynie z liter.
  - **CHECK (([LastName] like '[a-zA-Z]%'))**
- Wiek pracownika między 100 i 18 - data urodzenia nie wcześniejsza niż 100 lat temu oraz nie późniejsza niż 18 lat temu.
  - **CHECK ((datepart(year,[BirthDate])>(datepart(year,getdate())-(100)) AND [BirthDate]<=(getdate()-(18))))**
- Data zatrudnienia następuje po dacie urodzenia oraz najpóźniej w dniu wprowadzania danych, zatrudnienie następuje gdy osoba ma ukończone 18 lat. Domyślnie data zatrudnienia jest datą dodania pracownika do bazy.
  - **CHECK ((datepart(year,[HireDate])>(datepart(year,[BirthDate])+(18)) AND [HireDate]>=getdate()))**
  - **DEFAULT (getdate()) FOR [HireDate]**
- Email jest unikalny i zawiera '@'.
  - **CHECK (([Email] like '%@%'))**
- Numer telefonu składa się wyłącznie ze znaków numerycznych.
  - **CHECK ((isnumeric([Phone])=(1)))**
- Pracownik nie może sam być swoim przełożonym.
  - **CHECK (([ReportsTo]<>[EmployeeID]))**

```
CREATE TABLE [dbo].[Employess](
	[EmployeeID] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](30) NOT NULL,
	[LastName] [nvarchar](30) NOT NULL,
	[Title] [nvarchar](30) NOT NULL,
	[BirthDate] [date] NOT NULL,
	[HireDate] [date] NOT NULL,
	[Address] [nvarchar](50) NOT NULL,
	[CityID] [int] NOT NULL,
	[Phone] [nvarchar](15) NOT NULL,
	[Email] [nvarchar](50) NOT NULL,
	[ReportsTo] [int] NULL,
	[Notes] [nvarchar](50) NULL,
 CONSTRAINT [PK_Employess] 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],
 CONSTRAINT [U_Email] UNIQUE NONCLUSTERED
(
	[Email] 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].[Employess] ADD  CONSTRAINT [DF_Employess_HireDate]  DEFAULT (getdate()) FOR
[HireDate]
GO

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

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [FK_Employess_Cities]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [FK_Employess_Employess] FOREIGN
KEY([ReportsTo])
REFERENCES [dbo].[Employess] ([EmployeeID])
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [FK_Employess_Employess]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_BirthDate] CHECK
((datepart(year,[BirthDate])>(datepart(year,getdate())-(100)) AND [BirthDate]<=(getdate()-(18))))
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_BirthDate]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_Email] CHECK  (([Email] like
'%@%'))
```

```
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_Email]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_FirstName] CHECK  ([FirstName]
like '[a-zA-Z]%')
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_FirstName]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_HireDate] CHECK
((datepart(year,[HireDate])>(datepart(year,[BirthDate])+(18)) AND [HireDate]>=getdate()))
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_HireDate]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_LastName] CHECK  ([LastName]
like '[a-zA-Z]%')
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_LastName]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_Phone] CHECK
((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_Phone]
GO

ALTER TABLE [dbo].[Employess]  WITH CHECK ADD  CONSTRAINT [CK_ReportsTo] CHECK
(([ReportsTo]<>[EmployeeID]))

GO



ALTER TABLE [dbo].[Employess] CHECK CONSTRAINT [CK_ReportsTo]
GO
```

17. Tables

Przechowuje listę wszystkich stolików znajdujących się w restauracji.

- **TableID**[int][NOT NULL] - Identyfikator stolika, wartość auto inkrementowana (klucz główny).
- **Seats**[int][NOT NULL] - Ilość miejsc przy stoliku.

Warunki integralności:

- Seats jest wartością dodatnią
  - CHECK (([Seats]>(0))).

```
CREATE TABLE [dbo].[Tables](
	[TableID] [int] IDENTITY(1,1) 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
```

18. GlobalConst

Przechowuje informację warunkach jakie musi spełnić klient aby móc skorzystać z formularza online.

- **ConstID**[int][NOT NULL] - klucz główny
- **ConstName**[nvarchar(50)][NOT NULL], - nazwa warunku
- **ConstValue**[int][NOT NULL] - wartość danego warunku
- **dateFrom**[date][NOT NULL] - data od kiedy obowiązuje warunek
- **dateTo**[date][NULL] - data do kiedy obowiązuje warunek

Warunki integralności:

- ConstValue wartość dodatnia.
  - `CHECK  (([ConstValue]>(0)))`
- dateTo jest datą późniejszą niż dateFrom.
  - `CHECK  (( [dateTo] IS NULL OR [dateTo]>[dateFrom]))`
- dateFrom - domyślnie data dodania do tabeli.
  - `DEFAULT (getdate()) FOR [dateFrom]`

```
CREATE TABLE [dbo].[GlobalConst](
	[ConstID] [int] IDENTITY(1,1) NOT NULL,
	[ConstName] [nvarchar](50) NOT NULL,
	[ConstValue] [int] NOT NULL,
	[dateFrom] [date] NOT NULL,
	[dateTo] [date] NULL,
 CONSTRAINT [PK_GlobalConst] PRIMARY KEY CLUSTERED
(
	[ConstID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[GlobalConst] ADD  CONSTRAINT [DF_GlobalConst_dateFrom]  DEFAULT (getdate())
FOR [dateFrom]
GO
ALTER TABLE [dbo].[GlobalConst]  WITH CHECK ADD  CONSTRAINT [CK_GlobalConstDates] CHECK  ((
[dateTo] IS NULL OR [dateTo]>[dateFrom]))
GO
ALTER TABLE [dbo].[GlobalConst] CHECK CONSTRAINT [CK_GlobalConstDates]
GO
ALTER TABLE [dbo].[GlobalConst]  WITH CHECK ADD  CONSTRAINT [CK_GlobalConstPositivaConstValue]
CHECK  (([ConstValue]>(0)))
GO
ALTER TABLE [dbo].[GlobalConst] CHECK CONSTRAINT [CK_GlobalConstPositivaConstValue]
GO
```

19. PermanentDiscountsParameters

Przechowuje informację o parametrach zniżek trwałych.

- **ConstID**[int][NOT NULL] - Identyfikator parametru (klucz główny).
- **Z1**[int][NOT NULL] - Wymagana liczba zamówień do otrzymania zniżki.
- **K1**[money][NOT NULL]- Minimalna wymagana kwota za każde zamówienie.
- **R1**[float][NOT NULL] - Wartość zniżki.
- **EnterDate**[datetime][NOT NULL] - Data dodania parametru.

Warunki integralności:

- Z1 jest wartością dodatnią.
- K1 jest wartością dodatnią.
- R1 jest wartością z zakresu [0,1].
  - `CHECK  (([Z1]>(0) AND [K1]>(0) AND ([R1]>=(0) AND [R1]<=(1))))`
- EnterDate - domyślnie data dodania do tabeli.
  - `DEFAULT (getdate()) FOR [EnterDate]`

```
CREATE TABLE [dbo].[PermanentDiscountsParameters](
	[ConstID] [int] IDENTITY(1,1) NOT NULL,
	[Z1] [int] NOT NULL,
	[K1] [money] NOT NULL,
	[R1] [float] NOT NULL,
	[EnterDate] [datetime] NOT NULL,
 CONSTRAINT [PK_PermanentDiscountsParameters] PRIMARY KEY CLUSTERED
(
	[ConstID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PermanentDiscountsParameters] ADD  CONSTRAINT
[DF_PermanentDiscountsParameters_EnterDate]  DEFAULT (getdate()) FOR [EnterDate]
GO

ALTER TABLE [dbo].[PermanentDiscountsParameters]  WITH CHECK ADD  CONSTRAINT
[CK_PermanentDiscountsParametersNumbers] CHECK  (([Z1]>(0) AND [K1]>(0) AND ([R1]>=(0) AND
[R1]<=(1))))
GO

ALTER TABLE [dbo].[PermanentDiscountsParameters] CHECK CONSTRAINT
[CK_PermanentDiscountsParametersNumbers]
GO
```

20. TemporaryDiscountsParameters

Przechowuje informację o parametrach zniżek tymczasowych

- **ConstID**[int][NOT NULL]- Identyfikator parametru (klucz główny).
- **K2**[money][NOT NULL] - Wymagana łączna kwota zrealizowanych zamówień do otrzymania zniżki.
- **R2**[float][NOT NULL] - Wartość zniżki.
- **D1**[int][NOT NULL] - Długość trwania zniżki w ilościach dni.
- **EnterDate**[datetime][NOT NULL] - Data dodania parametru.

Warunki integralności:

- K2 jest wartością nieujemną.
- R2 jest wartością z zakresu [0,1].
- D1 jest wartością dodatnią.
  - `CHECK  (([K2]>(0) AND ([R2]>=(0) AND [R2]<=(1)) AND [D1]>(0)))`
- EnterDate - domyślnie data dodania do tabeli.
  - `DEFAULT (getdate()) FOR [EnterDate]`

```
CREATE TABLE [dbo].[TemporaryDiscountsParemeters](
	[ConstID] [int] IDENTITY(1,1) NOT NULL,
	[K2] [money] NOT NULL,
	[R2] [float] NOT NULL,
	[D1] [int] NOT NULL,
	[EnterDate] [datetime] NOT NULL,
 CONSTRAINT [PK_TemporaryDiscountsParemeters] PRIMARY KEY CLUSTERED
(
	[ConstID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TemporaryDiscountsParemeters] ADD  CONSTRAINT
[DF_TemporaryDiscountsParemeters_EnterDate]  DEFAULT (getdate()) FOR [EnterDate]
GO

ALTER TABLE [dbo].[TemporaryDiscountsParemeters]  WITH CHECK ADD  CONSTRAINT
[CK_TemporaryDiscountsParemeters] CHECK  (([K2]>(0) AND ([R2]>=(0) AND [R2]<=(1)) AND [D1]>(0)))
GO

ALTER TABLE [dbo].[TemporaryDiscountsParemeters] CHECK CONSTRAINT
[CK_TemporaryDiscountsParemeters]
GO
```

21. PermanentDiscounts

Przechowuje informację na temat rabatów trwałych.

- ○ **ClientID**[int][NOT NULL] - Identyfikator klienta. (klucz główny będący także kluczem obcym do tabeli IndividualClients).
- ○ **ConstID**[int][NOT NULL] - Identyfikator parametru zniżki (klucz obcy do tabeli PermanentDiscountsParameters).
- ○ **EnterDate**[date][NULL] - Data wprowadzenia

Warunki integralności:

- EnterDate - domyślnie data dodania do tabeli.
  - ○ DEFAULT (getdate()) FOR [EnterDate]

```
CREATE TABLE [dbo].[PermanentDiscounts](
	[ClientID] [int] NOT NULL,
	[ConstID] [int] NOT NULL,
	[EnterDate] [date] NULL,
 CONSTRAINT [PK_PermanentDiscounts] PRIMARY KEY CLUSTERED
(
	[ClientID] 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].[PermanentDiscounts] ADD  CONSTRAINT [DF_PermanentDiscounts_EnterDate]
DEFAULT (getdate()) FOR [EnterDate]
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscounts_IndividualClients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
```

```
GO

ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT [FK_PermanentDiscounts_IndividualClients]
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscounts_PermanentDiscountsParameters] FOREIGN KEY([ConstID])
REFERENCES [dbo].[PermanentDiscountsParameters] ([ConstID])
GO

ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT
[FK_PermanentDiscounts_PermanentDiscountsParameters]
GO
```

22. TemporaryDiscounts

Przechowuje informację na temat rabatów tymczasowych trwających określoną liczbę dni.

- **TDiscountID**[int][NOT NULL] - Identyfikator rabatu (klucz główny).
- **ClientID**[int][NOT NULL] - Identyfikator klienta (klucz obcy do tabeli IndividualClients).
- **ConstID**[int][NOT NULL]- Identyfikator parametru zniżki (klucz obcy do tabeli TemporaryDiscountsParameters).
- **StartDate**[datetime][NULL] - Data przyznania zniżki.
- **EndDate**[datetime][NULL] - Data zakończenia zniżki.

Warunki integralności:

- EndsDate jest datą późniejszą niż StartDate
  - `CHECK  (([EndsDate]>[StartDate]) OR [EndsDate] IS NULL )`
- EndsDate - domyślnie data dodania do tabeli.
  - `DEFAULT (getdate()) FOR [StartDate]`

```
CREATE TABLE [dbo].[TemporaryDiscounts](
	[TDiscountID] [int] IDENTITY(1,1) NOT NULL,
	[ClientID] [int] NOT NULL,
	[ConstID] [int] NOT NULL,
	[StartDate] [datetime] NULL,
	[EndsDate] [datetime] NULL,
 CONSTRAINT [PK_TemporaryDiscounts] PRIMARY KEY CLUSTERED
(
	[TDiscountID] 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].[TemporaryDiscounts] ADD  CONSTRAINT [DF_TemporaryDiscounts_StartDate]
DEFAULT (getdate()) FOR [StartDate]
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_TemporaryDiscounts_IndividualClients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
GO
```

```
ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[FK_TemporaryDiscounts_IndividualClients]
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_TemporaryDiscounts_TemporaryDiscountsParemeters] FOREIGN KEY([ConstID])
REFERENCES [dbo].[TemporaryDiscountsParemeters] ([ConstID])
GO

ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT
[FK_TemporaryDiscounts_TemporaryDiscountsParemeters]
GO

ALTER TABLE [dbo].[TemporaryDiscounts]  WITH CHECK ADD  CONSTRAINT [CK_TemporaryDiscounts]
CHECK  (([EndsDate]>[StartDate]))
GO

ALTER TABLE [dbo].[TemporaryDiscounts] CHECK CONSTRAINT [CK_TemporaryDiscounts]
GO
```

23. OrderStatuses

Słownik przechowujący statusy zamówień.

- **StatusID**[int][NOT NULL] - Identyfikator statusu (klucz główny).
- **StatusName**[nvarchar(50)][NOT NULL]- Nazwa statusu.

Warunki integralności:

- StatusName tylko litery
  - CHECK (([StatusName] like '[A-Za-z]%'))
- StatusName in (‘R’,’NR’) zrealizowane/niezrealizowane.
  - CHECK (([StatusName]='NR' OR [StatusName]='R'))

```
CREATE TABLE [dbo].[OrderStatuses](
	[StatusID] [int] IDENTITY(1,1) NOT NULL,
	[StatusName] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Statuses] PRIMARY KEY CLUSTERED
(
	[StatusID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderStatuses]  WITH CHECK ADD  CONSTRAINT [CK_OrderStatuses] CHECK
(([StatusName] like '[A-Za-z]%'))
GO

ALTER TABLE [dbo].[OrderStatuses] CHECK CONSTRAINT [CK_OrderStatuses]
GO

ALTER TABLE [dbo].[OrderStatuses]  WITH CHECK ADD  CONSTRAINT [CK_OrderStatuses_Names]
CHECK  (([StatusName]='NR' OR [StatusName]='R'))
GO

ALTER TABLE [dbo].[OrderStatuses] CHECK CONSTRAINT [CK_OrderStatuses_Names]
GO
```

24. **ReservationStatuses**

Słownik przechowujący statusy rezerwacji.

- **StatusID**[int][NOT NULL] - Identyfikator statusu (klucz główny).
- **StatusName**[nvarchar(20)][NOT NULL] - Nazwa statusu.

Warunki integralności:

- StatusName tylko litery.
  - **CHECK (([StatusName] like '[A-Za-z]%'))**
- StatusName przyjmuje tylko wartości (‘W’,’C’,’P’,’A) ,gdzie:
  - W - Zamówienie oczekujące.
  - C - Zamówienie potwierdzone.
  - P - Zamówienie potwierdzone i opłacone.
  - A - zamówienie anulowane.
  - **CHECK (([StatusName]='A' OR [StatusName]='P' OR [StatusName]='C' OR [StatusName]='W'))**

```
CREATE TABLE [dbo].[ReservationsStatuses](
	[StatusID] [int] IDENTITY(1,1) NOT NULL,
	[StatusName] [nvarchar](20) NOT NULL,
 CONSTRAINT [PK_ReservationsStatuses] PRIMARY KEY CLUSTERED
(
	[StatusID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservationsStatuses]  WITH CHECK ADD  CONSTRAINT [CK_ReservationsStatuses]
CHECK  (([StatusName] like '[A-Za-z]%'))
GO

ALTER TABLE [dbo].[ReservationsStatuses] CHECK CONSTRAINT [CK_ReservationsStatuses]
GO

ALTER TABLE [dbo].[ReservationsStatuses]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationsStatuses_Names] CHECK  (([StatusName]='A' OR [StatusName]='P' OR [StatusName]='C' OR
[StatusName]='W'))
GO

ALTER TABLE [dbo].[ReservationsStatuses] CHECK CONSTRAINT [CK_ReservationsStatuses_Names]
GO
```

4. Widoki

1. **OrdersPerIndClient**

Ilość wszystkich zamówień danego klienta.

```
CREATE VIEW OrdersPerIndClient
AS
SELECT ClientID,Count(*) AS Orders FROM IndividualClientsReservations
GROUP BY ClientID
```

```
          UNION
          SELECT CR.ClientID,Count(CRT.OrderID) AS Orders FROM CompaniesReservations AS CR
          INNER JOIN CompaniesReservationsTables AS CRT ON  CR.ReservationID=CRT.ReservationID
          WHERE CRT.OrderID IS NOT NULL
GROUP BY CR.ClientID
```

2. **CompaniesCountries**
   Ilość firm z danego kraju.

```
          CREATE VIEW CompaniesCountries
          AS
          SELECT Countries.CountryID,Countries.CountryName,COUNT(ClientID) AS Companies FROM
          Companies INNER JOIN Cities ON Companies.CityID = Cities.CityID
          INNER JOIN Countries ON Cities.CountryID=Countries.CountryID
GROUP BY Countries.CountryID,Countries.CountryName
```

3. **ROrders**
   Zrealizowane zamówienia.

```
          CREATE VIEW ROrders
          AS
          SELECT OrderID FROM Orders INNER JOIN OrderStatuses ON
          Orders.StatusID=OrderStatuses.StatusID
WHERE OrderStatuses.StatusName='R'
```

4. **NROrders**
   Niezrealizowane zamówienia.

```
          CREATE VIEW NROrders
          AS
          SELECT OrderID FROM Orders INNER JOIN OrderStatuses ON
          Orders.StatusID=OrderStatuses.StatusID
WHERE OrderStatuses.StatusName='NR'
```

5. **InPlaceOrders**
   Zamówienia na miejscu.

```
          CREATE VIEW InPlaceOrders
          AS
          SELECT Orders.OrderID FROM Orders
WHERE [Take-away]='False'
```

6. TakeAwayOrders

Zamówienia na wynos.

```
CREATE VIEW TakeAwayOrders
AS
SELECT Orders.OrderID FROM Orders
WHERE [Take-away]='True''
```

7. WReservations

Oczekujące rezerwacje.

```
        CREATE VIEW WReservations
        AS
        SELECT R.ReservationID,ICR.ClientID,R.ReservationDate,R.RequiredDate,'individual' as clientType
        FROM Reservations AS R
        INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
        INNER JOIN IndividualClientsReservations AS ICR ON R.ReservationID=ICR.ReservationID
        WHERE RS.StatusName = 'W'
        UNION
        SELECT R.ReservationID,CR.ClientID,R.ReservationDate,R.RequiredDate,'company' as clientType
        FROM Reservations AS R
        INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
        INNER JOIN CompaniesReservations AS CR ON R.ReservationID=CR.ReservationID
WHERE RS.StatusName = 'W'
```

8. AReservations

Anulowane rezerwacje.

```
CREATE VIEW AReservations
AS
SELECT R.ReservationID,ICR.ClientID,R.ReservationDate,R.RequiredDate,'individual' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN IndividualClientsReservations AS ICR ON R.ReservationID=ICR.ReservationID
WHERE RS.StatusName = 'A'
UNION
SELECT R.ReservationID,CR.ClientID,R.ReservationDate,R.RequiredDate,'company' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN CompaniesReservations AS CR ON R.ReservationID=CR.ReservationID
WHERE RS.StatusName = 'A'
```

9. CReservations

Potwierdzone rezerwacje.

```
CREATE VIEW CReservations
AS
```

```sql
SELECT R.ReservationID,ICR.ClientID,R.ReservationDate,R.RequiredDate,'individual' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN IndividualClientsReservations AS ICR ON R.ReservationID=ICR.ReservationID
WHERE RS.StatusName = 'C'
UNION
SELECT R.ReservationID,CR.ClientID,R.ReservationDate,R.RequiredDate,'company' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN CompaniesReservations AS CR ON R.ReservationID=CR.ReservationID
WHERE RS.StatusName = 'C'
```

10. PReservations
Potwierdzone i opłacone rezerwacje.

```sql
CREATE VIEW PReservations
AS
SELECT R.ReservationID,ICR.ClientID,R.ReservationDate,R.RequiredDate,'individual' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN IndividualClientsReservations AS ICR ON R.ReservationID=ICR.ReservationID
WHERE RS.StatusName = 'P'
UNION
SELECT R.ReservationID,CR.ClientID,R.ReservationDate,R.RequiredDate,'company' as clientType
FROM Reservations AS R
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
INNER JOIN CompaniesReservations AS CR ON R.ReservationID=CR.ReservationID
WHERE RS.StatusName = 'P'
```

11. ReservationsWithNoAssignedTables
Rezerwacje, które nie mają jeszcze przydzielonych stolików.

```sql
CREATE VIEW ReservationsWithNoAssignedTables
AS
SELECT IndividualClientsReservations.ReservationID,RequiredDate,NumberOfPpl FROM
IndividualClientsReservations
INNER JOIN Reservations  ON IndividualClientsReservations.ReservationID =
Reservations.ReservationID
INNER JOIN ReservationsStatuses ON Reservations.StatusID = ReservationsStatuses.StatusID
WHERE TableID IS NULL AND NumberOfPpl IS NOT NULL AND RequiredDate >= GETDATE()
AND StatusName != 'A'
UNION
SELECT CompaniesReservationsTables.ReservationID,RequiredDate,NumberOfPpl FROM
CompaniesReservationsTables
INNER JOIN Reservations  ON CompaniesReservationsTables.ReservationID =
Reservations.ReservationID
INNER JOIN ReservationsStatuses ON Reservations.StatusID = ReservationsStatuses.StatusID
WHERE TableID IS NULL AND RequiredDate >= GETDATE() AND StatusName != 'A'
```

12. TodaysReservations
Dzisiejsze rezerwacje wraz z godziną,stolikiem i liczbą osób (Potwierdzone/Potwierdzone i opłacone).

```
          CREATE VIEW TodaysReservations
          AS
          SELECT Reservations.ReservationID,cast(RequiredDate as time) AS Time,TableID,NumberOfPpl
          FROM Reservations
          INNER JOIN IndividualClientsReservations ON
          Reservations.ReservationID=IndividualClientsReservations.ReservationID
          INNER JOIN ReservationsStatuses ON Reservations.StatusID=ReservationsStatuses.StatusID
          WHERE DAY(RequiredDate)=DAY(GETDATE()) AND
          MONTH(RequiredDate)=MONTH(GETDATE()) AND YEAR(RequiredDate)=YEAR(GETDATE())
          AND StatusName IN ('C','P')
          UNION
          SELECT Reservations.ReservationID,cast(RequiredDate as time),TableID,NumberOfPpl AS Time
          FROM Reservations
          INNER JOIN CompaniesReservationsTables ON
          Reservations.ReservationID=CompaniesReservationsTables.ReservationID
          INNER JOIN ReservationsStatuses ON Reservations.StatusID=ReservationsStatuses.StatusID
          WHERE DAY(RequiredDate)=DAY(GETDATE()) AND
          MONTH(RequiredDate)=MONTH(GETDATE()) AND YEAR(RequiredDate)=YEAR(GETDATE())
AND StatusName IN ('C','P')
```

13. TodaysNotConfirmedReservations

Dzisiejsze rezerwacje wraz z danymi kontaktowymi klienta (Oczekujące).

```
CREATE VIEW TodaysNotConfirmedReservations
AS
SELECT Reservations.ReservationID,cast(RequiredDate as time) AS
Time,Clients.ClientID,Clients.Phone,Clients.Email FROM Reservations
INNER JOIN IndividualClientsReservations ON
Reservations.ReservationID=IndividualClientsReservations.ReservationID
INNER JOIN Clients ON IndividualClientsReservations.ClientID=Clients.ClientID
INNER JOIN ReservationsStatuses ON Reservations.StatusID=ReservationsStatuses.StatusID
WHERE DAY(RequiredDate)=DAY(GETDATE()) AND
MONTH(RequiredDate)=MONTH(GETDATE()) AND YEAR(RequiredDate)=YEAR(GETDATE())
AND StatusName IN ('W')
UNION
SELECT Reservations.ReservationID,cast(RequiredDate as
time),Clients.ClientID,Clients.Phone,Clients.Email AS Time FROM Reservations
INNER JOIN CompaniesReservations ON
Reservations.ReservationID=CompaniesReservations.ReservationID
INNER JOIN Clients ON CompaniesReservations.ClientID = Clients.ClientID
INNER JOIN ReservationsStatuses ON Reservations.StatusID=ReservationsStatuses.StatusID
WHERE DAY(RequiredDate)=DAY(GETDATE()) AND
MONTH(RequiredDate)=MONTH(GETDATE()) AND YEAR(RequiredDate)=YEAR(GETDATE())
AND StatusName IN ('W')
```

14. TodaysOrders

Dzisiejsze zamówienia zawierające informacje o produkcie wraz z jego ilością.

```
CREATE VIEW TodaysOrders
AS
SELECT OrderDetails.OrderID,cast(RequiredDate as time) AS Time,ProductName,Quantity FROM
Orders
INNER JOIN OrderDetails ON Orders.OrderID =OrderDetails.OrderID
INNER JOIN Products ON OrderDetails.ProductID=Products.ProductID
```

```sql
WHERE YEAR(RequiredDate)=YEAR(GETDATE()) AND
MONTH(RequiredDate)=MONTH(GETDATE()) AND DAY(RequiredDate)=DAY(GETDATE())
```

15. TodaysReservedTables

Dzisiejsze zajęte stoliki z rezerwacji, które są potwierdzone/potwierdzone i opłacone.

```sql
CREATE VIEW TodaysReservedTables
AS
SELECT TableID,cast(RequiredDate as time) AS Time FROM Reservations
INNER JOIN IndividualClientsReservations ON
Reservations.ReservationID=IndividualClientsReservations.ReservationID
INNER JOIN ReservationsStatuses ON Reservations.StatusID = ReservationsStatuses.StatusID
WHERE StatusName IN ('C','P')
AND
DAY(RequiredDate)=DAY(GETDATE()) AND MONTH(RequiredDate)=MONTH(GETDATE()) AND
YEAR(RequiredDate)=YEAR(GETDATE())
UNION
SELECT TableID,cast(RequiredDate as time) AS Time FROM Reservations
INNER JOIN CompaniesReservationsTables ON Reservations.ReservationID =
CompaniesReservationsTables.ReservationID
INNER JOIN ReservationsStatuses ON Reservations.StatusID = ReservationsStatuses.StatusID
WHERE StatusName IN ('C','P')
AND
DAY(RequiredDate)=DAY(GETDATE()) AND MONTH(RequiredDate)=MONTH(GETDATE()) AND
YEAR(RequiredDate)=YEAR(GETDATE())
```

16. CurrentlyAvailableTables

Stoliki, które nie są zarezerwowane(potwierdzone i opłacone/potwierdzone) do godziny od obecnej chwili.

```sql
CREATE VIEW CurrentlyAvailableTables
AS
SELECT T1.TableID,T1.Seats FROM Tables AS T1
WHERE T1.TableID NOT IN
(
SELECT T2.TableID  FROM Tables AS T2
INNER JOIN IndividualClientsReservations AS ICR ON T2.TableID = ICR.TableID
INNER JOIN Reservations AS R ON R.ReservationID = ICR.ReservationID
INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
WHERE
(RequiredDate >= GETDATE()) AND (DATEDIFF(minute,GETDATE(),RequiredDate)<=60)
AND
StatusName IN ('C','P')
)
UNION
SELECT T1.TableID,T1.Seats FROM Tables AS T1
WHERE T1.TableID NOT IN
(
SELECT T2.TableID  FROM Tables AS T2
INNER JOIN CompaniesReservationsTables AS CRT ON T2.TableID = CRT.TableID
INNER JOIN Reservations AS R ON R.ReservationID = CRT.ReservationID
```

```
        INNER JOIN ReservationsStatuses AS RS ON R.StatusID = RS.StatusID
        WHERE
        (RequiredDate >= GETDATE()) AND (DATEDIFF(minute,GETDATE(),RequiredDate)<=60)
        AND
        StatusName IN ('C','P')
        )
```

17. RealizedTodaysOrders

Zrealizowane zamówienia z obecnego dnia.

```
        CREATE VIEW RealizedTodaysOrders
        AS
        SELECT ICR.ClientID,ICR.OrderID,SUM(Quantity*UnitPrice) AS Value,cast(O.RequiredDate as time)
        AS Time
        FROM IndividualClientsReservations AS ICR
        INNER JOIN Orders AS O ON ICR.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON ICR.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE DAY(RequiredDate) = DAY(GETDATE())
        AND MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
        GROUP BY OD.OrderID,ICR.ClientID,ICR.OrderID,O.RequiredDate
        UNION
        SELECT CR.ClientID,CRT.OrderID,SUM(Quantity*UnitPrice) AS Value,cast(O.RequiredDate as time)
        AS Time
        FROM CompaniesReservations AS CR
        INNER JOIN CompaniesReservationsTables AS CRT ON CR.ReservationID = CRT.ReservationID
        INNER JOIN Orders AS O ON CRT.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON CRT.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID = OS.StatusID
        WHERE CRT.OrderID IS NOT NULL
        AND DAY(RequiredDate) = DAY(GETDATE())
        AND MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
GROUP BY OD.OrderID,CR.ClientID,CRT.OrderID,O.RequiredDate
```

18. RealizedThisWeekOrders

Zrealizowane zamówienia z obecnego tygodnia.

```
        CREATE VIEW RealizedThisWeekOrders
        AS
        SELECT ICR.ClientID,ICR.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM IndividualClientsReservations AS ICR
        INNER JOIN Orders AS O ON ICR.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON ICR.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE DATEDIFF(DAY,RequiredDate,GETDATE()) <= 7
        AND MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
        GROUP BY OD.OrderID,ICR.ClientID,ICR.OrderID,O.RequiredDate
        UNION
        SELECT CR.ClientID,CRT.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM CompaniesReservations AS CR
        INNER JOIN CompaniesReservationsTables AS CRT ON CR.ReservationID = CRT.ReservationID
        INNER JOIN Orders AS O ON CRT.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON CRT.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID = OS.StatusID
        WHERE CRT.OrderID IS NOT NULL
        AND DATEDIFF(DAY,RequiredDate,GETDATE()) <= 7
        AND MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
GROUP BY OD.OrderID,CR.ClientID,CRT.OrderID,O.RequiredDate
```

19. RealizedThisMonthOrders

Zrealizowane zamówienia z obecnego miesiąca.

```
        CREATE VIEW RealizedThisMonthOrders
        AS
        SELECT ICR.ClientID,ICR.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM IndividualClientsReservations AS ICR
        INNER JOIN Orders AS O ON ICR.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON ICR.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
        GROUP BY OD.OrderID,ICR.ClientID,ICR.OrderID,O.RequiredDate
        UNION
        SELECT CR.ClientID,CRT.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM CompaniesReservations AS CR
        INNER JOIN CompaniesReservationsTables AS CRT ON CR.ReservationID = CRT.ReservationID
        INNER JOIN Orders AS O ON CRT.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON CRT.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID = OS.StatusID
        WHERE CRT.OrderID IS NOT NULL
        AND MONTH(RequiredDate) = MONTH(GETDATE())
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
GROUP BY OD.OrderID,CR.ClientID,CRT.OrderID,O.RequiredDate
```

20. RealizedThisYearOrders

Zrealizowane zamówienia z obecnego roku.

```
        CREATE VIEW RealizedThisYearOrders
        AS
        SELECT ICR.ClientID,ICR.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM IndividualClientsReservations AS ICR
        INNER JOIN Orders AS O ON ICR.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON ICR.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
        GROUP BY OD.OrderID,ICR.ClientID,ICR.OrderID,O.RequiredDate
        UNION
        SELECT CR.ClientID,CRT.OrderID,SUM(Quantity*UnitPrice) AS Value,O.RequiredDate AS Date
        FROM CompaniesReservations AS CR
        INNER JOIN CompaniesReservationsTables AS CRT ON CR.ReservationID = CRT.ReservationID
        INNER JOIN Orders AS O ON CRT.OrderID = O.OrderID
        INNER JOIN OrderDetails AS OD ON CRT.OrderID = OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID = OS.StatusID
        WHERE CRT.OrderID IS NOT NULL
        AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND StatusName = 'R'
        GROUP BY OD.OrderID,CR.ClientID,CRT.OrderID,O.RequiredDate
```

21. TodaysReservationsValues

Rezerwacje z zrealizowanymi zamówieniami wraz z wartością z obecnego dnia

```
        CREATE VIEW TodaysReservationsValues
        AS
        SELECT C.ClientID, CR.ReservationID, SUM(Quantity*UnitPrice) AS Value FROM
        CompaniesReservationsTables CR
        INNER JOIN Orders O ON O.OrderID = CR.OrderID
        INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        INNER JOIN CompaniesReservations AS C ON C.ReservationID = CR.ReservationID
        WHERE DAY(RequiredDate) = DAY(GETDATE())  AND MONTH(RequiredDate) =
        MONTH(GETDATE()) AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND
        StatusName = 'R'
        GROUP BY C.ClientID, CR.ReservationID
        UNION
        SELECT IR.ClientID, IR.ReservationID, SUM(Quantity*UnitPrice) FROM
        IndividualClientsReservations IR
        INNER JOIN Orders O ON O.OrderID = IR.OrderID
        INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE DAY(RequiredDate) = DAY(GETDATE())  AND MONTH(RequiredDate) =
        MONTH(GETDATE()) AND YEAR(RequiredDate) = YEAR(GETDATE())
        AND
        StatusName = 'R'
GROUP BY IR.ClientID, IR.ReservationID
```

22. ThisMonthReservationsValues

Rezerwacje z zrealizowanymi zamówieniami wraz z wartością z obecnego miesiąca

```
        CREATE VIEW ThisMonthReservationsValues
        AS
        SELECT C.ClientID, CR.ReservationID, SUM(Quantity*UnitPrice) AS Value FROM
        CompaniesReservationsTables CR
        INNER JOIN Orders O ON O.OrderID = CR.OrderID
        INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        INNER JOIN CompaniesReservations AS C ON C.ReservationID = CR.ReservationID
        WHERE MONTH(RequiredDate) = MONTH(GETDATE()) AND YEAR(RequiredDate) =
        YEAR(GETDATE())
```

```
AND
StatusName = 'R'
GROUP BY C.ClientID, CR.ReservationID
UNION
SELECT IR.ClientID, IR.ReservationID, SUM(Quantity*UnitPrice) FROM
IndividualClientsReservations IR
INNER JOIN Orders O ON O.OrderID = IR.OrderID
INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
WHERE MONTH(RequiredDate) = MONTH(GETDATE()) AND YEAR(RequiredDate) =
YEAR(GETDATE())
AND
StatusName = 'R'
GROUP BY IR.ClientID, IR.ReservationID
```

23. ThisYearReservationsValues

Rezerwacje z zrealizowanymi zamówieniami wraz z wartością z obecnego roku

```
        CREATE VIEW ThisYearReservationsValues
        AS
        SELECT C.ClientID, CR.ReservationID, SUM(Quantity*UnitPrice) AS Value FROM
        CompaniesReservationsTables CR
        INNER JOIN Orders O ON O.OrderID = CR.OrderID
        INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        INNER JOIN CompaniesReservations AS C ON C.ReservationID = CR.ReservationID
        WHERE YEAR(RequiredDate) = YEAR(GETDATE())
        AND
        StatusName = 'R'
        GROUP BY C.ClientID, CR.ReservationID
        UNION
        SELECT IR.ClientID, IR.ReservationID, SUM(Quantity*UnitPrice) FROM
        IndividualClientsReservations IR
        INNER JOIN Orders O ON O.OrderID = IR.OrderID
        INNER JOIN OrderDetails AS OD ON O.OrderID=OD.OrderID
        INNER JOIN OrderStatuses AS OS ON O.StatusID=OS.StatusID
        WHERE YEAR(RequiredDate) = YEAR(GETDATE())
        AND
        StatusName = 'R'
GROUP BY IR.ClientID, IR.ReservationID
```

24. ActualConstantsValues

Aktualne stałe oraz ich wartości

```
CREATE VIEW ActualConstantsValues
AS
SELECT GC.ConstName, GC.ConstValue FROM GlobalConst AS GC
WHERE (GC.dateTo >= GETDATE() OR GC.dateTo IS NULL) AND GC.dateFrom <= GETDATE()
UNION
SELECT 'K2' AS 'ConstName', TD.K2 FROM TemporaryDiscountsParemeters AS TD
WHERE TD.ConstID IN (SELECT TOP 1 ConstID FROM TemporaryDiscountsParemeters ORDER
BY EnterDate)
```

```
UNION
SELECT 'D1' AS 'ConstName', TD.D1 FROM TemporaryDiscountsParemeters AS TD
WHERE TD.ConstID IN (SELECT TOP 1 ConstID FROM TemporaryDiscountsParemeters ORDER
BY EnterDate)
UNION
SELECT 'R2' AS 'ConstName', TD.R2 FROM TemporaryDiscountsParemeters AS TD
WHERE TD.ConstID IN (SELECT TOP 1 ConstID FROM TemporaryDiscountsParemeters ORDER
BY EnterDate)
UNION
SELECT 'K1' AS 'ConstName', PD.K1 FROM PermanentDiscountsParameters AS PD
WHERE PD.ConstID IN (SELECT TOP 1 ConstID FROM PermanentDiscountsParameters ORDER
BY EnterDate)
UNION
SELECT 'R1' AS 'ConstName', PD.R1 FROM PermanentDiscountsParameters AS PD
WHERE PD.ConstID IN (SELECT TOP 1 ConstID FROM PermanentDiscountsParameters ORDER
BY EnterDate)
UNION
SELECT 'Z1' AS 'ConstName', PD.Z1 FROM PermanentDiscountsParameters AS PD
WHERE PD.ConstID IN (SELECT TOP 1 ConstID FROM PermanentDiscountsParameters ORDER
BY EnterDate)
```

25. IndividualClientsList
Lista klientów indywidualnych wraz z danymi kontaktowymi

```
CREATE VIEW IndividualClientsList
AS
SELECT FirstName, LastName, Phone, Email FROM IndividualClients AS IC
INNER JOIN Clients AS C ON IC.ClientID = C.ClientID
```

26. CompaniesList
Lista klientów biznesowych (firm) wraz z danymi kontaktowymi

```
CREATE VIEW CompaniesList
AS
SELECT C.CompanyName, C.ContactTitle + ' ' + C.ContactName AS 'ContactPerson', Phone, Email
FROM Companies AS C
INNER JOIN Clients AS Cl ON Cl.ClientID = C.ClientID
```

27. OrdersPerReservation
Ilość zamówień na rezerwację (wykonanych)

```
CREATE VIEW OrdersPerReservation
AS
SELECT R.ReservationID, COUNT(*) AS 'NumberOfOrders' FROM CompaniesReservationsTables R
INNER JOIN Orders O ON O.OrderID = R.OrderID
INNER JOIN OrderStatuses OS ON OS.StatusID = O.StatusID
WHERE StatusName = 'R'
GROUP BY R.ReservationID
```

```
UNION
SELECT R.ReservationID, COUNT(*) FROM IndividualClientsReservations R
INNER JOIN Orders O ON O.OrderID = R.OrderID
INNER JOIN OrderStatuses OS ON OS.StatusID = O.StatusID
WHERE StatusName = 'R'
GROUP BY R.ReservationID
```

28. ReservationsPerClient
Ilość rezerwacji na klienta (z podziałem na status)

```
CREATE VIEW ReservationsPerClient
AS
SELECT C.ClientID, RS.StatusName, COUNT(*) AS 'NumberOfReservations' FROM Clients C
INNER JOIN IndividualClientsReservations IR ON IR.ClientID = C.ClientID
INNER JOIN Reservations R ON R.ReservationID = IR.ReservationID
INNER JOIN ReservationsStatuses RS ON RS.StatusID = R.StatusID
GROUP BY C.ClientID, RS.StatusName
UNION
SELECT C.ClientID, RS.StatusName, COUNT(*) FROM Clients C
INNER JOIN CompaniesReservations CR ON CR.ClientID = C.ClientID
INNER JOIN Reservations R ON R.ReservationID = CR.ReservationID
INNER JOIN ReservationsStatuses RS ON RS.StatusID = R.StatusID
GROUP BY C.ClientID, RS.StatusName
```

29. RealisedOrdersPerEmployee
Ilość obsłużonych zamówień na pracownika

```
CREATE VIEW RealisedOrdersPerEmployee
AS
SELECT E.FirstName, E.LastName, COUNT(*) AS 'RealisedOrders' FROM Employess E
INNER JOIN Orders O ON O.EmployeeID = E.EmployeeID
INNER JOIN OrderStatuses OS ON OS.StatusID = O.StatusID
WHERE StatusName = 'R'
GROUP BY E.FirstName, E.LastName
```

30. OrdersWithClientID
Lista zamówień wraz z identyfikatorem klienta indywidualnego

```
CREATE VIEW OrdersWithClientID
AS
SELECT O.OrderID, ICR.ClientID, O.EmployeeID, O.OrderDate, O.RequiredDate, O.[Take-away],
O.StatusID
FROM Orders O
INNER JOIN IndividualClientsReservations ICR ON O.OrderID = ICR.OrderID;
```

31. Top5MostFrequentlyPurchasedProducts
Pięć najczęściej zamawianych produktów

```
CREATE VIEW Top5MostFrequentlyPurchasedProducts
AS
SELECT TOP 5 P.ProductID, P.ProductName, SUM(OD.Quantity) AS Amount
FROM OrderDetails OD
INNER JOIN ProductsAvailability PA ON OD.ProductID = PA.RecordID
INNER JOIN Products P ON PA.ProductID = P.ProductID
GROUP BY P.ProductID, P.ProductName
ORDER BY 3 DESC
```

32. Top5MostFrequentlyOrderingIndividualClients

Pięciu klientów indywidualnych, którzy złożyli najwięcej zamówień

```
CREATE VIEW Top5MostFrequentlyOrderingIndividualClients
AS
SELECT TOP 5 ICR.ClientID, IC.FirstName, IC.LastName, COUNT(O.OrderID) AS Amount
FROM Orders O
INNER JOIN IndividualClientsReservations ICR ON O.OrderID = ICR.OrderID
INNER JOIN IndividualClients IC ON ICR.ClientID = IC.ClientID
GROUP BY ICR.ClientID, IC.FirstName, IC.LastName
ORDER BY 4 DESC
```

33. Top5MostFrequentlyOrderingCompanies

Pięć firm, które złożyły najwięcej zamówień

```
CREATE VIEW Top5MostFrequentlyOrderingCompanies
AS
SELECT TOP 5 C.ClientID, C.CompanyName, COUNT(O.OrderID) AS Amount
FROM Orders O
INNER JOIN CompaniesReservationsTables CRT ON O.OrderID = CRT.OrderID
INNER JOIN CompaniesReservations CR ON CRT.ReservationID = CR.ReservationID
INNER JOIN Companies C ON CR.ClientID = C.ClientID
GROUP BY C.ClientID, C.CompanyName
ORDER BY 3 DESC
```

34. Top5MostFrequentlyOrderingIndividualClientWithPayment

Pięciu klientów indywidualnych, którzy najczęściej realizują zamówienie

```
CREATE VIEW Top5MostFrequentlyOrderingIndividualClientWithPayment
AS
SELECT TOP 5 ICR.ClientID, IC.FirstName, IC.LastName, COUNT(O.OrderID) AS Amount
FROM Orders O
INNER JOIN IndividualClientsReservations ICR ON O.OrderID = ICR.OrderID
INNER JOIN IndividualClients IC ON ICR.ClientID = IC.ClientID
INNER JOIN OrderStatuses OS ON O.StatusID = OS.StatusID
WHERE OS.StatusName = 'R'
GROUP BY ICR.ClientID, IC.FirstName, IC.LastName
ORDER BY 4 DESC
```

35. Top5MostFrequentlyOrderingCompaniesWithPayment
Pięć firm, które najczęściej realizują zamówienie

```
CREATE VIEW Top5MostFrequentlyOrderingCompaniesWithPayment
AS
SELECT TOP 5 C.ClientID, C.CompanyName, COUNT(O.OrderID) AS Amount
FROM Orders O
INNER JOIN CompaniesReservationsTables CRT ON O.OrderID = CRT.OrderID
INNER JOIN CompaniesReservations CR ON CRT.ReservationID = CR.ReservationID
INNER JOIN Companies C ON CR.ClientID = C.ClientID
INNER JOIN OrderStatuses OS ON O.StatusID = OS.StatusID
WHERE OS.StatusName = 'R'
GROUP BY C.ClientID, C.CompanyName
ORDER BY 3 DESC
```

36. Top5MostExpensiveOrdersFromIndividualClients
Pięć zamówień klientów indywidualnych o największej wartości

```
CREATE VIEW Top5MostExpensiveOrdersFromIndividualClients
AS
SELECT TOP 5 OD.OrderID, IC.ClientID, IC.FirstName, IC.LastName, SUM(OD.Quantity *
OD.UnitPrice) AS Amount
FROM OrderDetails OD
INNER JOIN IndividualClientsReservations ICR ON OD.OrderID = ICR.OrderID
INNER JOIN IndividualClients IC ON ICR.ClientID = IC.ClientID
GROUP BY OD.OrderID, IC.ClientID, IC.FirstName, IC.LastName
ORDER BY 5 DESC
```

37. Top5MostExpensiveOrdersFromCompanies
Pięć zamówień firm o największej wartości

```
CREATE VIEW Top5MostExpensiveOrdersFromCompanies
AS
SELECT TOP 5 OD.OrderID, C.ClientID, C.CompanyName, SUM(OD.Quantity * OD.UnitPrice) AS
Amount
FROM OrderDetails OD
INNER JOIN CompaniesReservationsTables CRT ON OD.OrderID = CRT.OrderID
INNER JOIN CompaniesReservations CR ON CRT.ReservationID = CR.ReservationID
INNER JOIN Companies C ON CR.ClientID = C.ClientID
GROUP BY OD.OrderID, C.ClientID, C.CompanyName
ORDER BY 4 DESC
```

38. IndividualClientsSummaryOrderValue
Sumaryczna wartość zamówień dla każdego klienta indywidualnego

```
CREATE VIEW IndividualClientsSummaryOrderValue
AS
SELECT ICR.ClientID, IC.FirstName, IC.LastName, SUM(T.Amount) AS Amount
```

```
FROM (
        SELECT OD.OrderID, SUM(OD.Quantity * OD.UnitPrice) AS Amount
        FROM OrderDetails OD
        GROUP BY OD.OrderID
) AS T
INNER JOIN IndividualClientsReservations ICR ON T.OrderID = ICR.OrderID
INNER JOIN IndividualClients IC ON ICR.ClientID = IC.ClientID
GROUP BY ICR.ClientID, IC.FirstName, IC.LastName
```

39. IndividualClientsActiveTemporaryDiscounts

Lista klientów indywidualnych mających ważne zniżki jednorazowe

```
CREATE VIEW IndividualClientsActiveTemporaryDiscounts
AS
SELECT IC.FirstName, IC.LastName, TDP.D1, TDP.K2, TDP.R2, TD.StartDate, TD.EndsDate
FROM TemporaryDiscounts TD
INNER JOIN IndividualClients IC ON TD.ClientID = IC.ClientID
INNER JOIN TemporaryDiscountsParemeters TDP ON TD.ConstID = TDP.ConstID
WHERE TD.EndsDate IS NULL
```

40. IndividualClientsActivePermanentDiscounts

Lista klientów indywidualnych mających ważne zniżki stałe

```
CREATE VIEW IndividualClientsActivePermanentDiscounts
AS
SELECT IC.FirstName, IC.LastName, PDP.K1, PDP.R1, PDP.Z1, PD.EnterDate
FROM PermanentDiscounts PD
INNER JOIN IndividualClients IC ON PD.ClientID = IC.ClientID
INNER JOIN PermanentDiscountsParameters PDP ON PD.ConstID = PDP.ConstID
```

41. ReservationsRaport

Raport rezerwacji (z podziałem na lata, miesiące i dni)

```
CREATE VIEW ReservationsRaport
AS
SELECT TOP 100 percent *
FROM (
        SELECT 'Ind' AS ClientType, R.ReservationID, ICR.ClientID, ICR.OrderID,
ICR.NumberOfPpl,
        YEAR(R.ReservationDate) AS Year, MONTH(R.ReservationDate) AS Month,
DAY(R.ReservationDate) AS Day,
        RS.StatusName AS Status
        FROM IndividualClientsReservations ICR
        INNER JOIN Reservations R ON ICR.ReservationID = R.ReservationID
        INNER JOIN ReservationsStatuses RS ON R.StatusID = RS.StatusID
        UNION
        SELECT 'Com' AS ClientType, R.ReservationID, CR.ClientID, CRT.OrderID,
CRT.NumberOfPpl,
        YEAR(R.ReservationDate) AS Year, MONTH(R.ReservationDate) AS Month,
DAY(R.ReservationDate) AS Day,
        RS.StatusName AS Status
```

```
            FROM CompaniesReservations CR
            INNER JOIN Reservations R ON CR.ReservationID = R.ReservationID
            INNER JOIN ReservationsStatuses RS ON R.StatusID = RS.StatusID
            INNER JOIN CompaniesReservationsTables CRT ON CR.ReservationID =
CRT.ReservationID
            ) AS T
ORDER BY 6,7,8
```

42. MenuRaport

Raport menu (z podziałem na lata, miesiące i dni)

```
CREATE VIEW MenuRaport
AS
SELECT TOP 100 percent *
FROM (
            SELECT PA.ProductID, P.ProductName, PA.Price,
            '' AS FromDate, YEAR(PA.FromDate) AS YearFrom, MONTH(PA.FromDate) AS
MonthFrom, DAY(PA.FromDate) AS DayFrom,
            '' AS ToDate, YEAR(PA.ToDate) AS YearTo, MONTH(PA.ToDate) AS MonthTo,
DAY(PA.ToDate) AS DayTo,
            C.CategoryName
            FROM ProductsAvailability PA
            INNER JOIN Products P ON PA.ProductID = P.ProductID
            INNER JOIN Categories C ON P.CategoryID = C.CategoryID
            ) AS T
ORDER BY 8,9,10
```

43. Menu

Widok obecnego menu

```
CREATE VIEW [dbo].[Menu]
AS
SELECT PA.RecordID, P.ProductName, PA.Price FROM ProductsAvailability AS PA
 INNER JOIN Products AS P ON P.ProductID = PA.ProductID
 WHERE FromDate <= GETDATE()
 AND (ToDate >= GETDATE() OR ToDate IS NULL)
```

44. PermanentDiscountsRaport

Raport zniżek stałych (z podziałem na lata, miesiące i dni)

```
CREATE VIEW PermanentDiscountsRaport
AS
SELECT TOP 100 percent *
FROM (
            SELECT PD.ClientID, IC.FirstName, IC.LastName, PDP.Z1, PDP.K1, PDP.R1,
            '' AS EnterDate, YEAR(PD.EnterDate) AS Year, MONTH(PD.EnterDate) AS Month,
DAY(PD.EnterDate) AS Day
            FROM PermanentDiscounts PD
            INNER JOIN IndividualClients IC ON PD.ClientID = IC.ClientID
```

```
          INNER JOIN PermanentDiscountsParameters PDP ON PD.ConstID = PDP.ConstID
          ) AS T
ORDER BY 8,9,10
```

45. TemporaryDiscountsRaport

Raport zniżek jednorazowych (z podziałem na lata, miesiące i dni)

```
CREATE VIEW TemporaryDiscountsRaport
AS
SELECT TOP 100 percent *
FROM (
          SELECT TD.ClientID, IC.FirstName, IC.LastName, IC.DiscountBalance, TDP.D1, TDP.K2,
TDP.R2,
          '' AS StartDate, YEAR(TD.StartDate) AS YearStart, MONTH(TD.StartDate) AS MonthStart,
DAY(TD.StartDate) AS DayStart,
          '' AS EndDate, YEAR(TD.EndsDate) AS YearEnd, MONTH(TD.EndsDate) AS MonthEnd,
DAY(TD.EndsDate) AS DayEnd
          FROM TemporaryDiscounts TD
          INNER JOIN TemporaryDiscountsParemeters TDP ON TD.ConstID = TDP.ConstID
          INNER JOIN IndividualClients IC ON TD.ClientID = IC.ClientID
          ) AS T
ORDER BY 13,14,15
```

46. IndividualClientsOrdersRaport

Raport zamówień klientów indywidualnych (z podziałem na lata, miesiące i dni)

```
CREATE VIEW IndividualClientsOrdersRaport
AS
SELECT TOP 100 percent *, T.OrderValue * (1 - T.DiscountValue) AS ValueDiscounted
FROM (
          SELECT O.OrderID, ICR.ClientID, O.EmployeeID,
          '' AS OrderDate, YEAR(O.OrderDate) AS Year, MONTH(O.OrderDate) as Month,
DAY(O.OrderDate) AS Day,
          O.[Take-away], O.StatusID, OrderValue =
          (
                    SELECT SUM(OD.Quantity * OD.UnitPrice) AS Summary
                    FROM OrderDetails OD
                    WHERE OD.OrderID = O.OrderID
                    GROUP BY OD.OrderID
          ),
          DiscountValue =
          (
                    SELECT TOP 1 *
                    FROM (
                    SELECT TDP.R2 AS DiscountValue
                    FROM TemporaryDiscounts TD
                    INNER JOIN TemporaryDiscountsParemeters TDP ON TD.ConstID = TDP.ConstID
                    WHERE TD.ClientID = ICR.ClientID AND TD.EndsDate >= GETDATE() AND
TD.StartDate >= O.OrderDate
                    UNION
                    SELECT PDP.R1 AS DiscountValue
                    FROM PermanentDiscounts PD
                    INNER JOIN PermanentDiscountsParameters PDP ON PD.ConstID = PDP.ConstID
                    WHERE PD.ClientID = ICR.ClientID
```

```
                        UNION
                        SELECT 0 AS DiscountValue
                        ) AS T
                        ORDER BY 1 DESC
                )
                FROM IndividualClientsReservations ICR
                INNER JOIN Orders O ON ICR.OrderID = O.OrderID
                ) AS T
        ORDER BY 5,6,7
```

47. CompaniesOrdersRaport
Raport zamówień firm (z podziałem na lata, miesiące i dni)

```
        CREATE VIEW CompaniesOrdersRaport
        AS
        SELECT TOP 100 percent *
        FROM (
                SELECT O.OrderID, CR.ClientID, C.CompanyName, C.ContactTitle, C.ContactName,
        O.EmployeeID,
                '' AS OrderDate, YEAR(O.OrderDate) AS Year, MONTH(O.OrderDate) as Month,
        DAY(O.OrderDate) AS Day,
                O.[Take-away], O.StatusID, OrderValue =
                (
                        SELECT SUM(OD.Quantity * OD.UnitPrice) AS Summary
                        FROM OrderDetails OD
                        WHERE OD.OrderID = O.OrderID
                        GROUP BY OD.OrderID
                ),
                '' AS DiscountValue
                FROM CompaniesReservations CR
                INNER JOIN CompaniesReservationsTables CRT ON CR.ReservationID =
        CRT.ReservationID
                INNER JOIN Companies C ON CR.ClientID = C.ClientID
                INNER JOIN Orders O ON CRT.OrderID = O.OrderID
                ) AS T
        ORDER BY 7,8,9
```

5. Procedury

1. FindCountry
wyszukuje id kraju, jeśli nie istnieje to wstawia do bazy.

```
        CREATE PROCEDURE  [dbo].[sp_FindCountry]
                @countryName nvarchar(50),
                @countryID int OUTPUT
        AS
        BEGIN
                SET NOCOUNT ON;
                SET @countryID = (SELECT CountryID FROM Countries WHERE CountryName = @countryName)
                IF(@countryID IS NULL)
                BEGIN
```

```
            INSERT INTO Countries(CountryName)
            VALUES (@countryName)
            SET @countryID = @@IDENTITY
            END
END
```

2. **FindCity**
   wyszukuje id miasta, jeśli nie istnieje to wstawia do bazy.

```
CREATE PROCEDURE  [dbo].[sp_FindCity]
            @cityName nvarchar(50),
            @countryName nvarchar(50),
            @cityID int OUTPUT
AS
BEGIN
            SET NOCOUNT ON;
            SET @cityID = (SELECT CityID FROM Cities WHERE CityName = @cityName)
            IF(@cityID IS NULL)
            BEGIN
            DECLARE @countryID INT;
            EXEC sp_FindCountry @countryName,@countryID OUTPUT;
            INSERT INTO Cities(CityName,CountryID)
            VALUES (@cityName,@countryID);
            SET @cityID = @@IDENTITY
            END
END
```

3. **InsertClient**
   wstaw klienta do bazy

```
CREATE PROCEDURE [dbo].[sp_InsertClient]
            @phone nvarchar(12),
            @email nvarchar(40) = NULL,
            @clientID INT OUTPUT
AS
BEGIN
            SET NOCOUNT ON;
            BEGIN
            INSERT INTO Clients(Phone,Email)
            VALUES(@phone,@email);
            SET @clientID = @@IDENTITY
            END
END
```

4. **InsertIndividualClient**
   wstaw klienta indywidualnego do bazy.

```
CREATE PROCEDURE [dbo].[sp_InsertIndividualClient]
            @phone nvarchar(12),
            @email nvarchar(40) = NULL,
            @firstName nvarchar(15),
            @lastName nvarchar(30),
            @individualClientID INT OUTPUT
AS
```

```
BEGIN
        SET NOCOUNT ON;
        BEGIN
        DECLARE @newID INT;
        EXEC sp_InsertClient @phone,@email,@newID OUTPUT;
        SET @individualClientID = @newID;
        INSERT INTO IndividualClients(ClientID,FirstName,LastName,DiscountBalance)
        VALUES(@newID,@firstName,@lastName,0);
        END
END
```

5. InsertCompany

wstaw klienta biznesowego do bazy.

```
CREATE PROCEDURE [dbo].[sp_InsertCompany]
        @phone nvarchar(12),
        @email nvarchar(40) = NULL,
        @companyName nvarchar(30),
        @contactName nvarchar(20),
        @contactTitle nvarchar(15),
        @cityName nvarchar(50),
        @countryName nvarchar(50),
        @address nvarchar(50),
        @NIP nvarchar(50) = NULL,
        @CompanyID INT OUTPUT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN
        DECLARE @newID INT;
        EXEC sp_InsertClient @phone,@email,@newID OUTPUT;
        SET @CompanyID = @newID;

        DECLARE @cityID INT;
        EXEC sp_FindCity @cityName,@countryName,@cityID OUTPUT;

        INSERT INTO Companies(ClientID,CompanyName,ContactName,ContactTitle,CityID,Adress,NIP)
         VALUES(@newID,@companyName,@contactName,@contactTitle,@cityID,@address,@NIP);
        END
END
```

6. AddEmployeeToOrder

Przypisanie pracownika do zamówienia.

```
CREATE PROCEDURE [dbo].[sp_AddEmployeeToOrder]
        @orderID INT,
        @employeeID INT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
        BEGIN
        ;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
        END
        ELSE IF NOT EXISTS (SELECT 'X' FROM Employess WHERE EmployeeID = @employeeID)
        BEGIN
        ;THROW 52000,'Pracownik o podanym ID nie istnieje!',1;
```

```
		END
		DECLARE @currentEmployee INT;
		SET @currentEmployee = (SELECT EmployeeID FROM Orders WHERE OrderID = @orderID);
		IF @currentEmployee IS NOT NULL
		BEGIN
		;THROW 52000,'Pracownik jest już przypisany do tego zamówienia',1;
		END
		UPDATE Orders SET EmployeeID=@employeeID WHERE OrderID = @orderID;
		END TRY
		BEGIN CATCH
		DECLARE @msg nvarchar(2048) = 'Błąd dodania pracownika do zamówienia:'
		+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
		THROW 52000,@msg,1;
		END CATCH
	END
```

7. FindCategory

Wyszukuje id kategorii, jeśli nie istnieje to wstawia do bazy.

```
	CREATE PROCEDURE [dbo].[sp_FindCategory]
		@categoryName nvarchar(50),
		@categoryID INT OUTPUT
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN
		SET @categoryID = (SELECT CategoryID FROM Categories WHERE CategoryName = @categoryName);
		IF(@categoryID IS NULL)
		BEGIN
		INSERT INTO Categories(CategoryName)
		VALUES (@categoryName);
		SET @categoryID = @@IDENTITY;
		END
		END
	END
```

8. InsertGlobalConst

wprowadza do bazy nową wartość zmiennej globalnej

```
	CREATE PROCEDURE [dbo].[sp_InsertGlobalConst]
		@ClientID INT OUTPUT,
		@ConstName nvarchar(50),
		@ConstValue int,
		@DateFrom date,
		@DateTo date,

		@existingID int
	AS
	BEGIN
		SET NOCOUNT ON;
		SET @existingID = (
		SELECT GC.ConstID
		FROM GlobalConst GC
		WHERE GC.ConstName = @ConstName AND dateTo IS NULL)

		IF(@existingID IS NOT NULL) BEGIN
		UPDATE GlobalConst
		SET dateTo = DATEADD(day, -1, @DateFrom)
		WHERE ConstID = @existingID
```

```
        END

        BEGIN
        INSERT INTO GlobalConst(ConstName, ConstValue, dateFrom, dateTo)
        VALUES (@ConstName, @ConstValue, @DateFrom, @DateTo)
        SET @ClientID = @@IDENTITY
        END
END
```

9. InsertPermanentDiscountParemeters
wprowadza do bazy nowe wartości parametrów zniżek stałych

```
CREATE PROCEDURE [dbo].[sp_InsertPermanentDiscountParemeters]
        @ConstID INT OUTPUT,
        @Z1 int,
        @K1 money,
        @R1 float
AS
BEGIN
        SET NOCOUNT ON;

        BEGIN
        INSERT INTO PermanentDiscountsParameters(Z1, K1, R1)
        VALUES (@Z1, @K1, @R1)
        SET @ConstID = @@IDENTITY
        END
END
```

10. InsertTemporaryDiscountParemeters
wprowadza do bazy nowe wartości parametrów zniżek jednorazowych

```
CREATE PROCEDURE [dbo].[sp_InsertTemporaryDiscountParemeters]
        @ConstID INT OUTPUT,
        @K2 money,
        @R2 float,
        @D1 int,
        @EnterDate date
AS
BEGIN
        SET NOCOUNT ON;

        BEGIN
        INSERT INTO TemporaryDiscountsParemeters(K2, R2, D1, EnterDate)
        VALUES (@K2, @R2, @D1, @EnterDate)
        SET @ConstID = @@IDENTITY
        END
END
```

11. ConfirmReservation
zatwierdza wskazaną rezerwację

```
CREATE PROCEDURE [dbo].[sp_ConfirmReservation]
        @ReservationID int
AS
BEGIN
```

```
			SET NOCOUNT ON;

			BEGIN
			UPDATE Reservations
			SET StatusID = 4
			WHERE ReservationID = @ReservationID
			END
		END
```

12. PayForReservation

odznacza wskazaną rezerwację jako zatwierdzoną i opłaconą

```
		CREATE PROCEDURE [dbo].[sp_PayForReservation]
			@ReservationID int
		AS
		BEGIN
			SET NOCOUNT ON;

			BEGIN
			UPDATE Reservations
			SET StatusID = 2
			WHERE ReservationID = @ReservationID
			END
		END
```

13. CancelReservation

oznacza wskazaną rezerwację jako anulowaną

```
CREATE PROCEDURE [dbo].[sp_CancelReservation]
	@ReservationID int
AS
BEGIN
	SET NOCOUNT ON;

	BEGIN
	UPDATE Reservations
	SET StatusID = 1
	WHERE ReservationID = @ReservationID
	END
END
```

14. AddProductToOrder

Dodaj produkt do danego zamówienia.

```
		CREATE PROCEDURE [dbo].[sp_AddProductToOrder]
			@orderID INT,
			@productID INT,
			@quantity INT
		AS
		BEGIN
			SET NOCOUNT ON;

			BEGIN TRY
			IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
			BEGIN
			;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
			END

			IF NOT EXISTS (SELECT 'X' FROM Products WHERE ProductID = @productID)
		BEGIN
		;THROW 52000,'Produkt o podanym ID nie istnieje!',1;
		END

		IF EXISTS (SELECT 'X' FROM OrderDetails WHERE OrderID = @orderID AND ProductID = @productID)
		BEGIN
		UPDATE OrderDetails SET Quantity = @quantity + (SELECT Quantity FROM OrderDetails WHERE OrderID=@orderID
	AND ProductID=@productID) WHERE OrderID = @orderID AND ProductID=@productID
		END
		ELSE
		BEGIN
		IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@productID AND (GETDATE()>=FromDate
	AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
		BEGIN
		;THROW 52000,'Brak produktu w aktualnym menu',1;
		END

		DECLARE @value INT;
		SET @value = (SELECT Price FROM ProductsAvailability WHERE ProductID=@productID AND ((GETDATE()>=FromDate
	AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));

		INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
		VALUES(@orderID,@productID,@quantity,@value)
		END
		END TRY
		BEGIN CATCH
		DECLARE @msg nvarchar(2048) = 'Błąd przy dodawaniu produktu do zamówienia:'
		+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
		THROW 52000,@msg,1;
		END CATCH
	END
```

15. RemoveProductToOrder

usuń dany produkt z danego zamówienia.

```
	CREATE PROCEDURE [dbo].[sp_RemoveProductToOrder]
		@orderID INT,
		@productID INT,
		@quantity INT
	AS
	BEGIN
		SET NOCOUNT ON;

		BEGIN TRY
		IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
		BEGIN
		;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
		END

		IF NOT EXISTS (SELECT 'X' FROM Products WHERE ProductID = @productID)
		BEGIN
		;THROW 52000,'Produkt o podanym ID nie istnieje!',1;
		END

		IF EXISTS (SELECT 'X' FROM OrderDetails WHERE OrderID = @orderID AND ProductID = @productID)
		BEGIN
		DECLARE @currentQuantity INT;
		SET @currentQuantity = (SELECT Quantity FROM OrderDetails WHERE OrderID = @orderID AND ProductID =
	@productID);
		IF(@currentQuantity - @quantity <= 0)
		BEGIN
			DELETE FROM OrderDetails WHERE OrderID = @orderID AND ProductID = @productID;
		END
		ELSE
		BEGIN
			UPDATE OrderDetails SET Quantity = @currentQuantity - @quantity WHERE OrderID = @orderID AND
	ProductID = @productID;
		END
		END
		END TRY
		BEGIN CATCH
		DECLARE @msg nvarchar(2048) = 'Błąd przy dodawaniu produktu do zamówienia:'
		+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
		THROW 52000,@msg,1;
		END CATCH
END
```

16. ChangeProductQuantityInOrder

zmień ilość zamówień danego produktu w ramach danego zamówienia.

```
CREATE PROCEDURE [dbo].[sp_ChangeProductQuantityInOrder]
		@orderID INT,
		@productID INT,
		@quantity INT
AS
BEGIN
		SET NOCOUNT ON;

		BEGIN TRY
		IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
		BEGIN
		;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
		END

		IF NOT EXISTS (SELECT 'X' FROM Products WHERE ProductID = @productID)
		  BEGIN
		;THROW 52000,'Produkt o podanym ID nie istnieje!',1;
		END

		IF EXISTS (SELECT 'X' FROM OrderDetails WHERE OrderID = @orderID AND ProductID = @productID)
		BEGIN
		IF(@quantity <= 0)
		BEGIN
				;THROW 52000,'Ilość zamówień produktu musi być dodatnia!',1;
		END
		ELSE
		BEGIN
				UPDATE OrderDetails SET Quantity = @quantity WHERE OrderID = @orderID AND ProductID = @productID;
		END
		END
		ELSE
		BEGIN
		;THROW 52000,'Zamówienie o podanym ID, nie zawiera danego produktu!',1;
		END
		END TRY
		BEGIN CATCH
		DECLARE @msg nvarchar(2048) = 'Błąd przy dodawaniu produktu do zamówienia:'
		+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
		THROW 52000,@msg,1;
		END CATCH
END
```

17. MarkOrderAsRealized

zmień status zamówienia na zrealizowane.

```
CREATE PROCEDURE [dbo].[sp_MarkOrderAsRealized]
		@OrderID int
AS
BEGIN
		SET NOCOUNT ON;

		BEGIN TRY
		IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
```

```
          BEGIN
          ;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
          END
          DECLARE @currentStatus INT;
          SET @currentStatus = (SELECT StatusID FROM Orders WHERE OrderID = @OrderID)
          IF(@currentStatus = 2)
          BEGIN
          ;THROW 52000,'Zamówienie o podanym ID ma już status zrealizowanego!',1;
          END
          UPDATE Orders
          SET StatusID = 2
          WHERE OrderID = @OrderID
          END TRY
          BEGIN CATCH
          DECLARE @msg nvarchar(2048) = 'Błąd przy ustawieniu zamówienia jako zrealizowane:'
          + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
          THROW 52000,@msg,1;
          END CATCH
END
```

18. MarkOrderAsNotRealized

zmień status zamówienia na niezrealizowane

```
CREATE PROCEDURE [dbo].[sp_MarkOrderAsNotRealized]
          @OrderID int
AS
BEGIN
          SET NOCOUNT ON;

          BEGIN TRY
          IF NOT EXISTS (SELECT 'X' FROM Orders WHERE OrderID = @orderID)
          BEGIN
          ;THROW 52000,'Zamówienie o podanym ID nie istnieje!',1;
          END
          DECLARE @currentStatus INT;
          SET @currentStatus = (SELECT StatusID FROM Orders WHERE OrderID = @OrderID)
          IF(@currentStatus = 1)
          BEGIN
          ;THROW 52000,'Zamówienie o podanym ID ma już status niezrealizowanego!',1;
          END
          UPDATE Orders
          SET StatusID = 1
          WHERE OrderID = @OrderID
          END TRY
          BEGIN CATCH
          DECLARE @msg nvarchar(2048) = 'Błąd przy ustawieniu zamówienia jako zrealizowane:'
          + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
          THROW 52000,@msg,1;
          END CATCH
END
```

19. InsertProductAvailability

dodaje nową pozycję do aktualnego menu (oraz jeśli dana pozycja już była, to kończy poprzednie jej wystąpienie)

```
CREATE PROCEDURE [dbo].[sp_InsertProductAvailability]
          @RecordID INT OUTPUT,
          @ProductID int,
          @Price money,
          @FromDate date,
          @ToDate date,
          @existingID int
AS
BEGIN
          SET NOCOUNT ON;

          SET @existingID = (
          SELECT PA.RecordID
          FROM ProductsAvailability PA
          WHERE PA.ProductID = @ProductID AND PA.ToDate IS NULL)

          IF(@existingID IS NOT NULL) BEGIN
          UPDATE ProductsAvailability
          SET ToDate = DATEADD(day, -1, @FromDate)
          WHERE RecordID = @existingID
          END

          BEGIN
          INSERT INTO ProductsAvailability(ProductID, Price, FromDate, ToDate)
          VALUES (@ProductID, @Price, @FromDate, @ToDate)
          SET @RecordID = @@IDENTITY
          END
END
```

20. InsertOrder

dodaje nowe zamówienie do bazy

```
CREATE PROCEDURE [dbo].[sp_InsertOrder]
          @OrderID INT OUTPUT,
          @OrderDate datetime,
          @RequiredDate datetime,
          @Take_away bit,
          @EmployeeID int = NULL
AS
BEGIN
          SET NOCOUNT ON;

          BEGIN
                    INSERT INTO Orders(EmployeeID, OrderDate, RequiredDate, [Take-away], StatusID)
                    VALUES (@EmployeeID, @OrderDate, @RequiredDate, @Take_away, 1)
    SET @OrderID = @@IDENTITY
          END
END
```

21. FindTable

szuka wolnego stolika w danym terminie i o odpowiedniej ilości miejsc

```
          CREATE PROCEDURE [dbo].[sp_FindTable]
                    @numberOfPeople int,
                    @date datetime,
                    @tableID int OUTPUT
          AS
          BEGIN
                    SET NOCOUNT ON;
                    BEGIN
                    SET @tableID = (SELECT TOP 1 TableID FROM Tables
                    WHERE Seats >= @numberOfPeople
                    AND (TableID NOT IN
                    (SELECT TableID FROM CompaniesReservationsTables CR
                              INNER JOIN Reservations R ON R.ReservationID = CR.ReservationID
                              WHERE (YEAR(RequiredDate) = YEAR(@date) AND MONTH(RequiredDate) = MONTH(@date) AND
          DAY(RequiredDate) = DAY(@date))
```

```
			AND ABS(DATEDIFF(HOUR, @date, RequiredDate)) <= 2)
		OR TableID NOT IN
		(SELECT TableID FROM IndividualClientsReservations IR
			INNER JOIN Reservations R ON R.ReservationID = IR.ReservationID
			WHERE (YEAR(RequiredDate) = YEAR(@date) AND MONTH(RequiredDate) = MONTH(@date) AND
DAY(RequiredDate) = DAY(@date))
			AND ABS(DATEDIFF(HOUR, @date, RequiredDate)) <= 2))
		ORDER BY Seats)
		END
END
```

22. AddTableToOneOfCompaniesTables

dodanie stolika do pojedynczej pod rezerwacji danej rezerwacji

```
CREATE PROCEDURE [dbo].[sp_AddTableToOneOfCompaniesTables]
		@reservationID int,
		@ID int,
		@tableID int OUTPUT
AS
BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		IF NOT EXISTS (SELECT 'X' FROM CompaniesReservationsTables WHERE ReservationID = @reservationID)
		BEGIN
		;THROW 52000,'Rezerwacja firmowa o podanym ID nie istnieje!',1;
		END
		ELSE IF NOT EXISTS (SELECT 'X' FROM CompaniesReservationsTables WHERE ID = @ID)
		BEGIN
		;THROW 52000,'Podrezerwacja firmowa o podanym ID nie istnieje!',1;
		END
		SET @tableID = (SELECT TableID FROM CompaniesReservationsTables WHERE ID = @ID)
		IF (@tableID IS NULL)
		BEGIN
		DECLARE @numberOfPpl int = (SELECT NumberOfPpl FROM CompaniesReservationsTables WHERE ID = @ID)
		DECLARE @date datetime = (SELECT RequiredDate FROM Reservations WHERE ReservationID = @reservationID)
		EXEC sp_FindTable @numberOfPpl, @date, @tableID OUTPUT
		UPDATE CompaniesReservationsTables
		SET TableID = @tableID WHERE ID = @ID
		END
		END TRY
		BEGIN CATCH
		DECLARE @msg nvarchar(2048) = 'Błąd dodania stolika:'
		  + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
		THROW 52000,@msg,1;
		END CATCH
END
```

23. AddEmployee

dodaje pracownika do bazy

```
CREATE PROCEDURE [dbo].[sp_AddEmployee]
		@firstName nvarchar(30),
		@lastName nvarchar(30),
		@title nvarchar(30),
		@birthDate date,
		@hireDate date = GETDATE,
		@address nvarchar(50),
		@cityName nvarchar(50),
		@countryName nvarchar(50),
		@phone nvarchar(15),
		@email nvarchar(50),
		@reportsTo int = NULL,
		@notes nvarchar(50) = NULL,
		@employeeID INT OUTPUT
```

```
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN
        DECLARE @cityID INT;
        EXEC sp_FindCity @cityName, @countryName, @cityID OUTPUT;
        INSERT INTO Employess(FirstName, LastName, Title, BirthDate, HireDate, Address, CityID, Phone, Email, ReportsTo, Notes)
        VALUES(@firstName, @lastname, @title, @birthDate, @hireDate, @address, @cityID, @phone, @email, @reportsTo, @notes);
        SET @employeeID = @@IDENTITY
        END
END
```

24. FindProduct

Wyszukuje id produktu, jeśli nie istnieje to wstawia do bazy

```
CREATE PROCEDURE [dbo].[sp_FindProduct]
        @productName nvarchar(50),
        @productCategory nvarchar(50),
        @productID INT OUTPUT
AS
BEGIN
        SET @productID = (SELECT ProductID FROM Products WHERE ProductName = @productName)
        IF(@productID IS NULL)
        BEGIN
        DECLARE @categoryID INT;
        EXEC sp_FindCategory @productCategory, @categoryID OUTPUT
        INSERT INTO Products(ProductName, CategoryID)
        VALUES(@productName, @categoryID);
        SET @productID = @@IDENTITY
        END
END
```

25. AddTableToReservation

dodaje stoliki do danej rezerwacji

```
CREATE PROCEDURE [dbo].[sp_AddTableToReservation]
        @reservationID int,
        @tableID INT OUTPUT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        IF NOT EXISTS(SELECT 'X' FROM Reservations WHERE ReservationID = @reservationID)
        BEGIN
                ;THROW 52000,'Rezerwacja o podanym ID nie istnieje!',1;
        END
        SET @tableID = (SELECT TableID FROM IndividualClientsReservations WHERE ReservationID = @reservationID
        UNION SELECT TableID FROM CompaniesReservationsTables WHERE ReservationID = @reservationID)
        IF(@tableID IS NULL)
        BEGIN
        IF ((SELECT 'X' FROM IndividualClientsReservations WHERE ReservationID = @reservationID) IS NOT NULL)
        BEGIN
                DECLARE @numberOfPpl int = (SELECT NumberOfPpl FROM IndividualClientsReservations WHERE ReservationID =
@reservationID)
                DECLARE @date datetime = (SELECT RequiredDate FROM Reservations WHERE ReservationID = @reservationID)
        EXEC sp_FindTable @numberOfPpl, @date, @tableID OUTPUT
                 IF(@tableID IS NULL)
                 BEGIN
                ;THROW 52000,'Brak wolnych stolików!',1;
```

```
                    END
                  UPDATE IndividualClientsReservations
                  SET TableID = @tableID WHERE ReservationID = @reservationID
        END
        IF ((SELECT 'X' FROM CompaniesReservationsTables WHERE ReservationID = @reservationID) IS NOT NULL)
        BEGIN
                  DECLARE @id int
                  DECLARE reservationsList CURSOR FOR
                  SELECT ID FROM CompaniesReservationsTables WHERE ReservationID = @reservationID
                  OPEN reservationsList
                  FETCH NEXT FROM reservationsList INTO @id
                  WHILE @@FETCH_STATUS = 0
                  BEGIN
                  EXEC sp_AddTableToOneOfCompaniesTables @reservationID, @id, @tableID OUTPUT
                  END
                  CLOSE reservationsList
                  DEALLOCATE reservationsList
        END
        END
        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048) = 'Błąd dodania stolika:'
        + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg,1;
        END CATCH
END
```

26. AddNewTable

dodaje nowy stolik

```
        CREATE PROCEDURE [dbo].[sp_AddNewTable]
                  @seats int,
                  @tableID int OUTPUT
        AS
        BEGIN
                  SET NOCOUNT ON;
                  BEGIN
                  INSERT INTO Tables(Seats)
                  VALUES (@seats);
                  SET @tableID = @@IDENTITY
                  END
        END
```

27. CreateOrderInPlace

Złożenie zamówienia na miejscu

```
        CREATE PROCEDURE [dbo].[sp_CreateOrderInplace]
                  @ClientID int,
                  @NumberOfPpl int = NULL,
                  @DetailsList DetailsInsert READONLY,
          @TakeAway bit,
          @employeeID int = NULL
        AS
        BEGIN
                  SET NOCOUNT ON;
                  BEGIN TRY

                  DECLARE @orderID INT;
```

```
                DECLARE @date AS nvarchar(50)
                SET @date = CONVERT(datetime, GETDATE())
                DECLARE @reservationID INT;
                DECLARE @price money;
                DECLARE @categoryID int
                DECLARE
                @ProductID INT,
                @Quantity INT;
                DECLARE @tableID int

        BEGIN TRANSACTION CREATE_ORDER_IN_PLACE

        IF EXISTS (SELECT 'X' FROM Clients WHERE ClientID = @ClientID)
        BEGIN
        IF(@TakeAway = 1)
        BEGIN

        EXEC sp_InsertOrder
                @EmployeeID = @employeeID,
                @OrderID = @orderID OUTPUT,
                  @OrderDate = @date,
                @RequiredDate = @date,
                @Take_away = 1

        UPDATE Orders
        SET StatusID = 2
        WHERE OrderID = @orderID

        INSERT INTO Reservations(ReservationDate,RequiredDate,StatusID)
        VALUES (GETDATE(),GETDATE(),2)
        SET @reservationID = @@IDENTITY;

        INSERT INTO IndividualClientsReservations(ReservationID,TableID,NumberOfPpl,ClientID,OrderID)
        VALUES(@reservationID, NULL,NULL,@ClientID,@orderID)

        DECLARE DetailsList_Cursor cursor for select * from @DetailsList
             open DetailsList_Cursor
        fetch next from DetailsList_Cursor INTO @ProductID, @Quantity;
        WHILE @@FETCH_STATUS = 0
        BEGIN
                IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@ProductID AND
(GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
                BEGIN
                ROLLBACK TRANSACTION CREATE_ORDER_IN_PLACE
             ;THROW 52000,'Brak produktu w aktualnym menu',1;
                END
                ELSE
                BEGIN

                IF NOT EXISTS(SELECT 'X' FROM Products AS P WHERE P.ProductID = @ProductID AND P.CategoryID = 6)
                BEGIN
                SET @price =  (SELECT Price FROM ProductsAvailability WHERE ProductID=@ProductID AND
((GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));
                INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
                VALUES(@orderID,@ProductID,@Quantity,@price)

                fetch next from DetailsList_Cursor INTO @ProductID,@Quantity
                END
                ELSE
                BEGIN
                ROLLBACK TRANSACTION CREATE_ORDER_IN_PLACE
                ;THROW 52000,'Nie można zamówić owoców morza',1;
                END


                END
        END
        CLOSE DetailsList_Cursor
        DEALLOCATE DetailsList_Cursor

           EXEC sp_AddTableToReservation @reservationID, @tableID OUTPUT

        END
        ELSE
```

```
            BEGIN
            EXEC sp_InsertOrder
                        @EmployeeID = @employeeID,
                        @OrderID = @orderID OUTPUT,
                        @OrderDate = @date,
                        @RequiredDate = @date,
                        @Take_away = 1


            INSERT INTO Reservations(ReservationDate,RequiredDate,StatusID)
            VALUES (GETDATE(),GETDATE(),2)
            SET @reservationID = @@IDENTITY;

            INSERT INTO IndividualClientsReservations(ReservationID,TableID,NumberOfPpl,ClientID,OrderID)
            VALUES(@reservationID, NULL,@NumberOfPpl,@ClientID,@orderID)

            DECLARE DetailsList_Cursor cursor for select * from @DetailsList
            open DetailsList_Cursor
            fetch next from DetailsList_Cursor INTO @ProductID, @Quantity
            WHILE @@FETCH_STATUS = 0
            BEGIN
                        IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@ProductID AND
(GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
                        BEGIN
                        ROLLBACK TRANSACTION CREATE_ORDER_IN_PLACE
                        ;THROW 52000,'Brak produktu w aktualnym menu',1;
                        END
                        ELSE
                        BEGIN

                        IF NOT EXISTS(SELECT 'X' FROM Products AS P WHERE P.ProductID = @ProductID AND P.CategoryID = 6)
                        BEGIN
                        SET @price =  (SELECT Price FROM ProductsAvailability WHERE ProductID=@ProductID AND
((GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));
                        INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
                        VALUES(@orderID,@ProductID,@Quantity,@price)

                        fetch next from DetailsList_Cursor INTO @ProductID,@Quantity
                        END
                        ELSE
                        BEGIN
                        ROLLBACK TRANSACTION CREATE_ORDER_IN_PLACE
                        ;THROW 52000,'Nie można zamówić owoców morza',1;
                        END

                        END
            END
            CLOSE DetailsList_Cursor
             DEALLOCATE DetailsList_Cursor

                        EXEC sp_AddTableToReservation @reservationID, @tableID OUTPUT

            END
            END
            ELSE
            BEGIN
            ROLLBACK TRANSACTION CREATE_ORDER_IN_PLACE
            ;THROW 52000,'Brak klienta o podanym ID',1;
            END

            COMMIT TRANSACTION CREATE_ORDER_IN_PLACE

    END TRY
    BEGIN CATCH
      DECLARE @msg nvarchar(2048) = 'Błąd przy składaniu zamówienia na miejscu:'
        + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg,1;
    END CATCH
END
```

28. PlaceIndividualClientReservation

Złożenie rezerwacji przez klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[sp_PlaceIndividualClientReservation]
        @ClientID int,
        @NumberOfPpl int = NULL,
        @DetailsList DetailsInsert READONLY,
        @TakeAway bit,
        @requiredDate datetime,
        @employeeID int = NULL
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY

        DECLARE @orderID INT,
                @date AS nvarchar(50),
                @reservationID INT,
                @price money,
                @categoryID int,
                @ProductID INT,
                @Quantity INT,
                @tableID int,
                @totalOrderValue money,
                @CurrentWK INT,
                @CurrentWZ INT,
                @ClientWK INT,
                @ClientDiscount float
          SET @date = CONVERT(datetime, GETDATE())
          SET @totalOrderValue = 0

          BEGIN TRANSACTION ICR
        IF(@NumberOfPpl < 2)
        BEGIN
                ;THROW 52000,'Wymagana liczba osób do złożenia rezerwacji wynosi 2',1;
        END

                IF EXISTS (SELECT 'X' FROM Clients WHERE ClientID = @ClientID)
                BEGIN

         SET @CurrentWK = (SELECT GC.ConstValue FROM GlobalConst GC WHERE GC.ConstName = 'WK' AND GC.dateTo IS
NULL)
         SET @ClientWK = (SELECT COUNT(*) FROM Orders O INNER JOIN IndividualClientsReservations ICR ON O.OrderID =
ICR.OrderID WHERE ICR.ClientID = @ClientID GROUP BY ICR.ClientID)
         EXEC sp_FindClientDiscount @ClientID, @date, @ClientDiscount

                IF(@ClientWK IS NULL)
                BEGIN
                ;THROW 52000,'Liczba zamówień klienta jest zbyt mała.',1;
                END
         IF(@ClientWK < @CurrentWK)
         BEGIN
                ;THROW 52000,'Liczba zamówień klienta jest zbyt mała.',1;
         END
         ELSE
        IF(@TakeAway = 1)
         BEGIN

                EXEC sp_InsertOrder
                 @EmployeeID = @employeeID,
                 @OrderID = @orderID OUTPUT,
                 @OrderDate = @date,
                @RequiredDate = @requiredDate,
                 @Take_away = 1

                UPDATE Orders
                SET StatusID = 2
                WHERE OrderID = @orderID

                INSERT INTO Reservations(ReservationDate,RequiredDate,StatusID)
                VALUES (GETDATE(),@requiredDate,2)
                SET @reservationID = @@IDENTITY;

              INSERT INTO IndividualClientsReservations(ReservationID,TableID,NumberOfPpl,ClientID,OrderID)
```

```
                    VALUES(@reservationID, NULL,NULL,@ClientID,@orderID)

                    DECLARE DetailsList_Cursor cursor for select * from @DetailsList
                    open DetailsList_Cursor
                    fetch next from DetailsList_Cursor INTO @ProductID, @Quantity;
                    WHILE @@FETCH_STATUS = 0
                    BEGIN
            IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@ProductID AND (GETDATE()>=FromDate
    AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
            BEGIN
                         ROLLBACK TRANSACTION ICR
                         ;THROW 52000,'Brak produktu w aktualnym menu',1;
            END
            ELSE
            BEGIN
                         IF EXISTS(SELECT 'X' FROM Products AS P WHERE P.ProductID = @ProductID AND P.CategoryID = 6)
                         BEGIN
                                    IF (((DATEPART(WEEKDAY, @requiredDate) - 1)  NOT IN (4,5,6) ) OR  ((DATEPART(WEEKDAY,
    @requiredDate) - 1) IN (4,5,6) AND DATEDIFF(DAY,GETDATE(),DATEADD(DAY,(-1)*(DATEPART(WEEKDAY,@requiredDate) -
    2),@requiredDate)) < 0))
                                    BEGIN
                                    CLOSE DetailsList_Cursor
                                    DEALLOCATE DetailsList_Cursor
                                    ROLLBACK TRANSACTION ICR
                                                ;THROW 52000,'Data złożenia zamówienia na owoc morza jest niewłaściwa.',1;
                                    END
                    END

                                    SET @price =  (SELECT Price FROM ProductsAvailability WHERE ProductID=@ProductID AND
    ((GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));

                                    SET @totalOrderValue = @totalOrderValue + @price * @Quantity
                         INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
                                    VALUES(@orderID,@ProductID,@Quantity,@price)
                         fetch next from DetailsList_Cursor INTO @ProductID,@Quantity
            END
                    END
                         CLOSE DetailsList_Cursor
                    DEALLOCATE DetailsList_Cursor
                    EXEC sp_AddTableToReservation @reservationID, @tableID OUTPUT

             END
             ELSE
             BEGIN
                    EXEC sp_InsertOrder
                     @EmployeeID = @employeeID,
                     @OrderID = @orderID OUTPUT,
                     @OrderDate = @date,
                                 @RequiredDate = @requiredDate,
                     @Take_away = 1


                    INSERT INTO Reservations(ReservationDate,RequiredDate,StatusID)
                    VALUES (GETDATE(),@requiredDate,2)
                    SET @reservationID = @@IDENTITY;

                    INSERT INTO IndividualClientsReservations(ReservationID,TableID,NumberOfPpl,ClientID,OrderID)
                    VALUES(@reservationID, NULL,@NumberOfPpl,@ClientID,@orderID)

                    DECLARE DetailsList_Cursor cursor for select * from @DetailsList
                    open DetailsList_Cursor
                    fetch next from DetailsList_Cursor INTO @ProductID, @Quantity
                    WHILE @@FETCH_STATUS = 0
                    BEGIN
            IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@ProductID AND (GETDATE()>=FromDate
    AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
                          BEGIN
                         ROLLBACK TRANSACTION ICR
                         ;THROW 52000,'Brak produktu w aktualnym menu.',1;
            END
            ELSE
            BEGIN
                         IF EXISTS(SELECT 'X' FROM Products AS P WHERE P.ProductID = @ProductID AND P.CategoryID = 6)
                         BEGIN
```

```
                        IF (((DATEPART(WEEKDAY, @requiredDate) - 1)  NOT IN (4,5,6) ) OR  ((DATEPART(WEEKDAY,
@requiredDate) - 1) IN (4,5,6) AND DATEDIFF(DAY,GETDATE(),DATEADD(DAY,(-1)*(DATEPART(WEEKDAY,@requiredDate) -
2),@requiredDate)) < 0))
                        BEGIN
                                CLOSE DetailsList_Cursor
                                DEALLOCATE DetailsList_Cursor
                                ROLLBACK TRANSACTION ICR
                        ;THROW 52000,'Data złożenia zamówienia na owoc morza jest niewłaściwa.',1;
                        END
                        END
                          SET @price =  (SELECT Price FROM ProductsAvailability WHERE ProductID=@ProductID AND
((GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));
                         SET @totalOrderValue = @totalOrderValue + @price * @Quantity
                         INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
                                VALUES(@orderID,@ProductID,@Quantity,@price)
                         fetch next from DetailsList_Cursor INTO @ProductID,@Quantity
                END
                        END

                        CLOSE DetailsList_Cursor
                        DEALLOCATE DetailsList_Cursor
                        EXEC sp_AddTableToReservation @reservationID, @tableID OUTPUT

                END
                SET @CurrentWZ = (SELECT GC.ConstValue FROM GlobalConst GC WHERE GC.ConstName = 'WZ' AND GC.dateTo IS
NULL)
                IF(convert(int, floor(@totalOrderValue)) < @CurrentWZ)
                BEGIN
                        ROLLBACK TRANSACTION ICR
                          PRINT @totalOrderValue
                        ;THROW 52000,'Wartość zamówienia jest za mała.',1;
                END

                        END
                        ELSE
                        BEGIN
                ROLLBACK TRANSACTION ICR
                        ;THROW 52000,'Brak klienta o podanym ID',1;
                        END

     COMMIT TRANSACTION CREATE_ORDER_IN_PLACE

   END TRY
   BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd przy składaniu zamówienia na miejscu:'
            + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
            THROW 52000,@msg,1;
   END CATCH
END
```

29. AddOrderToCompaniesReservationTable

Złożenie zamówienia na firmę.

```
CREATE PROCEDURE [dbo].[sp_AddOrderToCompaniesReservationTable]
            @ID int,
            @DetailsList DetailsInsert READONLY,
            @EmployeeID int,
              @RequiredDate datetime = NULL
AS
BEGIN
            SET NOCOUNT ON;
            BEGIN TRY

            BEGIN TRANSACTION ICR

            IF EXISTS (SELECT 'X' FROM CompaniesReservationsTables WHERE TableReservationsID = @ID)
            BEGIN

            DECLARE @orderID INT,
                        @ProductID INT,
                        @Quantity INT,
                                @date varchar(50),
                                @price money

                        SET @date = CONVERT(datetime, GETDATE())
                SET @orderID = (SELECT OrderID FROM CompaniesReservationsTables WHERE TableReservationsID = @ID)
                IF(@RequiredDate IS NULL)
                        BEGIN
                        SET @RequiredDate = GETDATE()
                        PRINT @RequiredDate
                        END

                        IF(@orderID IS NULL)
                BEGIN
                        EXEC sp_InsertOrder @orderID OUTPUT,@date,@RequiredDate,0,@EmployeeID

                        UPDATE CompaniesReservationsTables
                        SET OrderID = @orderID
                        WHERE TableReservationsID = @ID
                END


                DECLARE DetailsList_Cursor cursor for select * from @DetailsList
                        open DetailsList_Cursor
                        fetch next from DetailsList_Cursor INTO @ProductID, @Quantity;
                        WHILE @@FETCH_STATUS = 0
                        BEGIN
                 IF NOT EXISTS (SELECT 'X' FROM ProductsAvailability WHERE ProductID=@ProductID AND (GETDATE()>=FromDate
        AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate))
                 BEGIN
                         ROLLBACK TRANSACTION ICR
                         ;THROW 52000,'Brak produktu w aktualnym menu',1;
                 END
                 ELSE
                 BEGIN
                         IF EXISTS(SELECT 'X' FROM Products AS P WHERE P.ProductID = @ProductID AND P.CategoryID = 6)
                         BEGIN

                                IF (((DATEPART(WEEKDAY, @RequiredDate) - 1)  NOT IN (4,5,6) ) OR  ((DATEPART(WEEKDAY,
        @RequiredDate) - 1) IN (4,5,6) AND DATEDIFF(DAY,GETDATE(),DATEADD(DAY,(-1)*(DATEPART(WEEKDAY,@RequiredDate) -
        2),@RequiredDate)) < 0))
                                BEGIN
                                CLOSE DetailsList_Cursor
                                DEALLOCATE DetailsList_Cursor
                                ROLLBACK TRANSACTION ICR
                                ;THROW 52000,'Data złożenia zamówienia na owoc morza jest niewłaściwa.',1;
                        END
                        END

                                SET @price =  (SELECT Price FROM ProductsAvailability WHERE ProductID=@ProductID AND
        ((GETDATE()>=FromDate AND ToDate IS NULL) OR (GETDATE() BETWEEN FromDate AND ToDate)));

                        INSERT INTO OrderDetails(OrderID,ProductID,Quantity,UnitPrice)
                                VALUES(@orderID,@ProductID,@Quantity,@price)
                          fetch next from DetailsList_Cursor INTO @ProductID,@Quantity
                  END
                        END
                              CLOSE DetailsList_Cursor
                        DEALLOCATE DetailsList_Cursor

                END
                ELSE
                BEGIN
                ROLLBACK TRANSACTION ICR
                ;THROW 52000,'Brak klienta o podanym ID',1;
                END

                COMMIT TRANSACTION ICR

          END TRY
          BEGIN CATCH
            DECLARE @msg nvarchar(2048) = 'Błąd przy dodawaniu zamówienia'
              + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
              THROW 52000,@msg,1;
          END CATCH
        END
```

30. CompanyReservation

złożenie rezerwacji przez firmę.

```
CREATE PROCEDURE [dbo].[sp_CompanyReservation]
            @ClientID int,
            @RequiredDate datetime = NULL,
            @EmployeeID int
AS
BEGIN
            SET NOCOUNT ON;
            BEGIN TRANSACTION ICR
              BEGIN TRY
            DECLARE @ReservationID INT
            INSERT INTO Reservations(ReservationDate,RequiredDate,StatusID)
            VALUES(GETDATE(),@RequiredDate,3)
            SET @ReservationID = @@IDENTITY

            INSERT INTO CompaniesReservations(ReservationID,ClientID)
            VALUES(@ReservationID,@ClientID)
            COMMIT TRANSACTION ICR
            END TRY
            BEGIN CATCH
                        ROLLBACK TRANSACTION ICR
                        DECLARE @msg nvarchar(2048) = 'Błąd przy składaniu rezerwacji:'
             + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
             THROW 52000,@msg,1;
            END CATCH
END
```

31. AddCompanyReservationTable

dodanie stolika do rezerwacji firmy wraz z opcjonalnym imieniem.

```
CREATE PROCEDURE [dbo].[sp_AddCompanyReservationTable]
            @NumberOfPpl int,
            @ReservationID int,
            @Name nvarchar(50) = NULL
AS
BEGIN
            SET NOCOUNT ON;
            BEGIN TRANSACTION ICR
              BEGIN TRY

            DECLARE @ID INT,
                        @TableID int
            EXEC sp_AddTableToReservation @ReservationID,@TableID OUTPUT

            INSERT INTO CompaniesReservationsTables(ReservationID,TableID,NumberOfPpl,OrderID)
            VALUES (@ReservationID,@TableID,@NumberOfPpl,NULL)
              SET @ID = @@IDENTITY

            IF(@Name IS NOT NULL)
            BEGIN
                        INSERT INTO CompaniesReservationsNames(TableReservationsID,Name)
            VALUES(@ID,@Name)
            END
            COMMIT TRANSACTION ICR
            END TRY
            BEGIN CATCH
             ROLLBACK TRANSACTION ICR
             DECLARE @msg nvarchar(2048) = 'Błąd przy składaniu rezerwacji:'
            + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
            THROW 52000,@msg,1;
            END CATCH
END
```

32. BindPermanentDiscountToClient

Przydzielenie klientowi indywidualnemu typu zniżki dożywotniej na którą zaczyna zbierać.

```
CREATE PROCEDURE [dbo].[sp_BindPermanentDiscountToClient]
            @ClientID int,
            @ConstID int OUTPUT
AS
BEGIN
            SET NOCOUNT ON;

            DECLARE @CurrentConstID int

            SET @CurrentConstID = (SELECT TOP 1 PDP.ConstID FROM PermanentDiscountsParameters PDP ORDER BY
PDP.EnterDate DESC)

            INSERT INTO  PermanentDiscounts(ClientID,ConstID,EnterDate)
            VALUES (@ClientID, @CurrentConstID, NULL)
            SET @ConstID = @@IDENTITY
END
```

33. GrantPermanentDiscountToClient

przydzielenie klientowi zniżki dożywotniej na którą już uzbierał.

```
CREATE PROCEDURE [dbo].[sp_GrantPermanentDiscountToClient]
            @ClientID int
AS
BEGIN
            SET NOCOUNT ON;

            IF EXISTS(SELECT 'X' FROM PermanentDiscounts WHERE ClientID = @ClientID AND EnterDate IS NULL)
            BEGIN

              DECLARE
              @Z1 int,
              @K1 money,
              @R1 float,
              @ConstID int,
              @Result int

            SET @ConstID = (SELECT ConstID FROM PermanentDiscounts WHERE ClientID = @ClientID)
              SET @Z1 = (SELECT Z1 FROM PermanentDiscountsParameters WHERE ConstID = @ConstID)
              SET @K1 = (SELECT K1 FROM PermanentDiscountsParameters WHERE ConstID = @ConstID)
              SET @R1 = (SELECT R1 FROM PermanentDiscountsParameters WHERE ConstID = @ConstID)

              SET @Result = (
              SELECT COUNT(*)
              FROM (
                          SELECT OD.OrderID, ValueDiscounted AS Summary
                          FROM IndividualClientsOrdersRaport OD
                          INNER JOIN IndividualClientsReservations ICR ON OD.OrderID = ICR.OrderID
```

```
                        WHERE ICR.ClientID = @ClientID
                        AND ValueDiscounted >= @K1
                ) AS T
                )

                IF(@Result >= @Z1)
                BEGIN
                UPDATE PermanentDiscounts
                SET EnterDate = GETDATE()
                WHERE ClientID = @ClientID
                END
               END
      END
```

34. CheckPermanentDiscount

Wywoływana po każdym złożeniu zamówienia przez klienta. Sprawdzam czy ma już przyznany program zniżek permanentnych, jeśli tak to sprawdzam czy mogę już mu ją przyznać, jeśli tak to nic nie robię. Jeśli nie to przydzielam mu program zniżki permanentnej na którą zaczyna zbierać.

```
CREATE PROCEDURE [dbo].[sp_CheckPermanentDiscount] @ClientID int
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN
   DECLARE @ConstID INT
   IF NOT EXISTS(SELECT 'X' FROM PermanentDiscounts WHERE ClientID = @ClientID)
                BEGIN
         exec sp_BindPermanentDiscountToClient @ClientID,@ConstID OUTPUT
                END
   exec sp_GrantPermanentDiscountToClient @ClientID
        END
END
```

35. CheckTemporaryDiscount

Wywoływana po każdym złożeniu zamówienia przez klienta. Sprawdzam czy ma już przyznany program zniżek czasowych, jeśli tak to sprawdzam czy mogę już mu ją przyznać, jeśli tak to sprawdzam czy nie upłynął jej termin, jeśli tak to wstawiam kolejny program. Jeśli nie to przydzielam mu program zniżki czasowej na którą zaczyna zbierać.

```
CREATE PROCEDURE [dbo].[sp_CheckTemporaryDiscount] @ClientID int
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN


        DECLARE @ConstID INT
        IF NOT EXISTS(SELECT 'X' FROM TemporaryDiscounts WHERE ClientID = @ClientID)
                BEGIN
                 exec sp_BindTemporaryDiscountToClient @ClientID,@ConstID OUTPUT
                END
        ELSE

        BEGIN
        DECLARE @EndsDate datetime
```

```
            SET @EndsDate =  (SELECT EndsDate FROM TemporaryDiscounts WHERE ClientID = @ClientID ORDER BY StartDate
    DESC)
            IF(@EndsDate IS NOT NULL AND @EndsDate < GETDATE())
            BEGIN
            exec sp_BindTemporaryDiscountToClient @ClientID,@ConstID OUTPUT
            END

            END
            exec sp_GrantPermanentDiscountToClient @ClientID
            END
    END
```

36. BindTemporaryDiscountToClient

Przydzielenie programu zniżek czasowych klientowi indywidualnemu, na który zaczyna zbierać.

```
CREATE PROCEDURE [dbo].[sp_BindTemporaryDiscountToClient]
  @ClientID int,
  @ConstID int OUTPUT
AS
BEGIN
        SET NOCOUNT ON;

  DECLARE @CurrentConstID int

  SET @CurrentConstID = (SELECT TOP 1 TDP.ConstID FROM TemporaryDiscountsParameters TDP ORDER BY TDP.EnterDate
DESC)

  INSERT INTO TemporaryDiscounts(ClientID,ConstID,StartDate,EndsDate)
  VALUES (@ClientID, @CurrentConstID, NULL,NULL)
  SET @ConstID = @@IDENTITY
END
```

37. GrantTemporaryDiscount

Przydzielenie klientowi zniżki czasowej

```
CREATE PROCEDURE [dbo].[sp_GrantTemporaryDiscountToClient] @ClientID int
AS
BEGIN
        SET NOCOUNT ON;
  IF EXISTS(SELECT 'X' FROM TemporaryDiscounts WHERE ClientID = @ClientID AND StartDate IS NULL AND EndsDate IS NULL)
    BEGIN
                DECLARE
        @K2 money,
        @R2 float,
        @D1 int,
        @ConstID int,
        @TDiscountID int,
        @PrevDiscountEndsDate datetime,
        @Result money

                SET @ConstID = (SELECT ConstID
                  FROM TemporaryDiscounts
                  WHERE ClientID = @ClientID
                 AND StartDate IS NULL
                 AND EndsDate IS NULL)


                SET @K2 = (SELECT K2 FROM TemporaryDiscountsParameters WHERE ConstID = @ConstID)
                SET @R2 = (SELECT R2 FROM TemporaryDiscountsParameters WHERE ConstID = @ConstID)
                SET @D1 = (SELECT D1 FROM TemporaryDiscountsParameters WHERE ConstID = @ConstID)
                SET @TDiscountID = (SELECT TOP 1 TDiscountID
                        FROM TemporaryDiscounts
                        WHERE ClientID = @ClientID
                         AND StartDate IS NOT NULL
                        AND EndsDate IS NOT NULL
                ORDER BY EndsDate DESC)
                IF (@TDiscountID IS NOT NULL)
        BEGIN
                SET @PrevDiscountEndsDate = (SELECT TOP 1 EndsDate FROM TemporaryDiscounts WHERE ClientID = @ClientID
AND StartDate IS NOT NULL AND EndsDate IS NOT NULL ORDER BY EndsDate DESC)
                SET @Result = (
        SELECT SUM(Summary)
        FROM (
                        SELECT OD.OrderID, ValueDiscounted AS Summary
                        FROM IndividualClientsOrdersRaport OD
                        INNER JOIN IndividualClientsReservations ICR ON OD.OrderID = ICR.OrderID
                        INNER JOIN Reservations R ON ICR.ReservationID = R.ReservationID
                        WHERE ICR.ClientID = @ClientID AND R.RequiredDate > @PrevDiscountEndsDate
                        AND ValueDiscounted >= @K2
                 ) AS T
                )
                IF (@Result >= @K2)
        BEGIN
                 UPDATE TemporaryDiscounts
                 SET StartDate = GETDATE()
                   WHERE ClientID = @ClientID
                AND StartDate IS NULL
                AND EndsDate IS NULL

                 UPDATE TemporaryDiscounts
                 SET EndsDate = DATEADD(DAY, @D1, GETDATE())
                 WHERE ClientID = @ClientID
                AND EndsDate IS NULL
        END
        END
                ELSE
        BEGIN
                SET @Result = (
        SELECT SUM(Summary)
        FROM (
                        SELECT OD.OrderID, ValueDiscounted AS Summary
                        FROM IndividualClientsOrdersRaport OD
                           INNER JOIN IndividualClientsReservations ICR ON OD.OrderID = ICR.OrderID
                        WHERE ICR.ClientID = @ClientID
                        AND ValueDiscounted >= @K2
                 ) AS T
                )
                IF (@Result >= @K2)
        BEGIN
                 UPDATE TemporaryDiscounts
                 SET StartDate = GETDATE()
                 WHERE ClientID = @ClientID
                AND StartDate IS NULL
                AND EndsDate IS NULL
                 UPDATE TemporaryDiscounts
                 SET EndsDate = DATEADD(DAY, @D1, GETDATE())
                 WHERE ClientID = @ClientID
                AND EndsDate IS NULL
        END

        END
    END
END
```

38. sp_ClientReservationsRaport

generowanie raportu dla konkretnego klienta.

```
CREATE PROCEDURE [dbo].[sp_ClientReservationsRaport]
        @ClientID  int
AS
SELECT TOP 100 percent *
FROM (
        SELECT 'Ind' AS ClientType, R.ReservationID, ICR.ClientID, ICR.OrderID, ICR.NumberOfPpl,
                YEAR(R.ReservationDate) AS Year, MONTH(R.ReservationDate) AS Month, DAY(R.ReservationDate) AS Day,
                RS.StatusName AS Status
        FROM IndividualClientsReservations ICR
        INNER JOIN Reservations R ON ICR.ReservationID = R.ReservationID
        INNER JOIN ReservationsStatuses RS ON R.StatusID = RS.StatusID
        UNION
        SELECT 'Com' AS ClientType, R.ReservationID, CR.ClientID, CRT.OrderID, CRT.NumberOfPpl,
                YEAR(R.ReservationDate) AS Year, MONTH(R.ReservationDate) AS Month, DAY(R.ReservationDate) AS Day,
                RS.StatusName AS Status
        FROM CompaniesReservations CR
        INNER JOIN Reservations R ON CR.ReservationID = R.ReservationID
        INNER JOIN ReservationsStatuses RS ON R.StatusID = RS.StatusID
        INNER JOIN CompaniesReservationsTables CRT ON CR.ReservationID = CRT.ReservationID
        ) AS T
        WHERE ClientID = @ClientID
ORDER BY 6,7,8
```

6. Funkcje

1. isTableAvailable

Zwraca 1 jeśli jest dostępny stolik na podaną liczbę ludzi w podanym terminie, 0 w przeciwnym przypadku.

```
CREATE FUNCTION FUNC_isTableAvailable
        (
        @RequiredDate datetime,
        @NumberOfPeople int
        )
        RETURNS int
AS
BEGIN
        IF EXISTS(
        SELECT T.TableID FROM Tables AS T
        WHERE T.Seats >= @NumberOfPeople
        AND T.TableID NOT IN(
        SELECT ICR.TableID AS TID FROM IndividualClientsReservations AS ICR
        INNER JOIN Reservations R ON R.ReservationID = ICR.ReservationID
        WHERE ICR.TableID IS NOT NULL AND DAY(R.RequiredDate) = DAY(@RequiredDate) AND
MONTH(R.RequiredDate) = MONTH(@RequiredDate) AND YEAR(R.RequiredDate) = YEAR(@RequiredDate)
        UNION
        SELECT CRT.TableID AS TID FROM CompaniesReservationsTables AS CRT
        INNER JOIN Reservations R ON R.ReservationID = CRT.ReservationID
        WHERE CRT.TableID IS NOT NULL AND DAY(R.RequiredDate) = DAY(@RequiredDate) AND
MONTH(R.RequiredDate) = MONTH(@RequiredDate) AND YEAR(R.RequiredDate) = YEAR(@RequiredDate)
        )
        )
        BEGIN
        RETURN 1
        END
        ELSE
        BEGIN
        RETURN 0
```

```
            END
        END
```

2. **ShouldMenuBeChanged**
   Zwraca 1 jeśli menu powinno być zmienione, 0 w przeciwnym przypadku.

```
CREATE FUNCTION [dbo].[func_shouldMenuBeChanged]()
        RETURNS bit
AS
BEGIN
        DECLARE @lastChangeDate date

        SET @lastChangeDate = (
          SELECT TOP 1 FromDate
          FROM ProductsAvailability
          GROUP BY FromDate
          HAVING COUNT(FromDate) >= 6
          ORDER BY 1 DESC
        )

        IF(ABS(DATEDIFF(DAY, GETDATE(), @lastChangeDate)) >= 14)
        BEGIN
        RETURN 1;
        END
        RETURN 0;
END
```

7. Indeksy

1. **orderID_index**

```
CREATE INDEX orderID_index ON OrderDetails(OrderID)
```

2. **productID_index**

```
CREATE INDEX productID_index ON ProductsAvailability(productID)
```

3. **IndividualClientsReservations_clientID_index**

```
CREATE INDEX IndividualClientsReservations_clientID_index ON
IndividualClientsReservations(ClientID)
```

4. **IndividualClientsReservations_OrderID_index**

```
CREATE INDEX IndividualClientsReservations_OrderID_index ON
IndividualClientsReservations(OrderID)
```

5. **TemporaryDiscounts_ClientID_index**

```
CREATE INDEX TemporaryDiscounts_ClientID_index ON TemporaryDiscounts(ClientID)
```

6. **Countries_CountryName_index**

```
CREATE INDEX Countries_CountryName_index ON Countries(CountryName)
```

7. **Cities_CityName_index**

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

8. **CompaniesReservations_clientID_index**

```
CREATE INDEX CompaniesReservations_clientID_index ON CompaniesReservations(ClientID)
```

9. **Products_categoryID_index**

```
CREATE INDEX Products_categoryID_index ON Products(categoryID)
```

8. Triggery

1. **tr_IndividualClientReservations_INSERT**
   Po złożeniu rezerwacji przez klienta indywidualnego sprawdzam czy może

zostać mu przyznana zniżka oraz czy istnieje dostępny stolik z podaną liczbą miejsc.

```
CREATE TRIGGER tr_IndividualClientReservations_INSERT
ON IndividualClientsReservatons
AFTER INSERT
AS
        SET NOCOUNT ON;
        DECLARE @RequiredDate datetime;
        SET @RequiredDate = (SELECT R.RequiredDate FROM Reservations R
        INNER JOIN inserted i ON i.ReservationID = R.ReservationID)
        IF EXISTS(
        SELECT * FROM inserted AS i
        WHERE FUNC_isTableAvailable(@RequiredDate,i.NumberOfPpl) = 0
        )
  BEGIN
        ;THROW 50001, 'Brak wolnych miejsc na podanym terminie rezerwacji.',1
  END

        SET @ClientID = (SELECT ClientID FROM inserted)
        exec sp_CheckTemporaryDiscount @ClientID
        exec sp_CheckPermanentDiscount @ClientID
GO
```

2. tr_TemporaryDiscounts_INSERT

Data dodania do tabeli temporaryDiscounts musi być wcześniejsza od daty zakończenia.

```
CREATE TRIGGER tr_TemporaryDiscounts_INSERT
ON TemporaryDiscounts
AFTER INSERT
AS
  SET NOCOUNT ON;
  IF EXISTS(
    SELECT * FROM inserted AS i
        WHERE i.StartDate >= i.EndsDate
        )
  BEGIN
        THROW 50001 , 'Data zakończenia musi być późniejsza niż data rozpoczęcia!', 1
  END
GO
```

3. tr_Orders_INSERT

```
CREATE TRIGGER tr_Orders_INSERT
ON Orders
AFTER INSERT
AS
        SET NOCOUNT ON;
        IF EXISTS(
        SELECT * FROM inserted AS i
        WHERE i.OrderDate >= i.RequiredDate
        )
        BEGIN
```

```
		;THROW 50001, 'Data wykonania zamówienia musi następowac po dacie złożenia zamówienia!', 1
		END
GO
```

4. tr_GlobalConst_INSERT

```
CREATE TRIGGER tr_GlobalConst_INSERT
ON GlobalConst
AFTER INSERT
AS
	SET NOCOUNT ON;
	IF NOT EXISTS(
		SELECT * FROM inserted AS i
		WHERE i.dateTo IS NULL
		OR i.dateTo > i.dateFrom
		)
	BEGIN
		;THROW 50001, 'Data zakończenia obowiązywania warunku musi nastąpić po dacie wprowadzenia!', 1
	END
	IF EXISTS (
		SELECT * FROM inserted AS i
		WHERE i.dateFrom < GETDATE()
		)
	BEGIN
		;THROW 50001, 'Nie można wprowadzć warunków wstecz (data wprowadzenia warunku poprzedza datę
dzisiejszą)!', 1
	END
GO
```

5. tr_Reservations_INSERT

```
CREATE TRIGGER tr_Reservations_INSERT
ON Reservations
AFTER INSERT
AS
  SET NOCOUNT ON;
	IF EXISTS(
    SELECT *
	FROM inserted i
	WHERE i.ReservationDate < GETDATE()
  )
  BEGIN
	THROW 50001 , 'Nie można ustawić daty zamówienia na wcześniejszą niż dzisiaj!', 1
  END

	IF EXISTS(
    SELECT *
	FROM inserted i
	WHERE i.RequiredDate < GETDATE()
  )
  BEGIN
```

```
          THROW 50001 , 'Nie można ustawić daty zamówienia na wcześniejszą niż dzisiaj!', 1
   END

   IF EXISTS(
     SELECT *
          FROM inserted i
          WHERE i.ReservationDate > i.RequiredDate
   )
   BEGIN
          THROW 50001 , 'Nie można zrobić rezerwacji na datę wcześniejszą, niż data rezerwowania!', 1
   END
GO
```

6. tr_ProductsAvailability_INSERT

```
CREATE TRIGGER tr_ProductsAvailability_INSERT
ON ProductsAvailability
AFTER INSERT
AS
   SET NOCOUNT ON;
   IF EXISTS(
             SELECT *
             FROM inserted i
             WHERE i.FromDate <= GETDATE()
    )
   BEGIN
      THROW 50001 , 'Nie można wprowadzić produktu od daty wcześniejszej niż jutro!', 1
   END

   DECLARE @toDate date;
   SET @toDate = (SELECT ToDate FROM inserted)

   IF (@toDate IS NOT NULL)
   BEGIN
             IF EXISTS(
               SELECT *
               FROM inserted i
               WHERE i.FromDate > i.ToDate
             )
   BEGIN
       THROW 50001 , 'Data wejścia produktu do menu musi być wcześniejsza, niż data usunięcia tego produktu z menu!', 1
   END
END
```

9. Role

1. Admin

administrator systemu

```
CREATE ROLE Admin AUTHORIZATION dbo
GRANT all to Admin
```

2. IndividualClient
klient indywidualny

```
CREATE ROLE IndividualClient AUTHORIZATION dbo
GRANT EXECUTE ON sp_ClientReservationsRaport to IndividualClient
GRANT EXECUTE ON Menu to IndividualClient
```

3. Company
klient biznesowy

```
CREATE ROLE Company AUTHORIZATION dbo
GRANT EXECUTE ON sp_ClientReservationsRaport to Company
GRANT EXECUTE ON Menu to Company
```

4. Employee
pracownik

```
CREATE ROLE Employee AUTHORIZATION dbo
GRANT SELECT ON WReservations to Employee
GRANT SELECT ON AReservations to Employee
GRANT SELECT ON CReservations to Employee
GRANT SELECT ON PReservations to Employee
GRANT SELECT ON ReservationsWithNoAssignedTables to Employee
GRANT SELECT ON TodaysReservations to Employee
GRANT SELECT ON TodaysNotConfirmedReservations to Employee
GRANT SELECT ON TodaysOrders to Employee
GRANT SELECT ON TodaysReservedTables to Employee
GRANT SELECT ON CurrentlyAvailableTables to Employee
GRANT SELECT ON IndividualClientsActiveTemporaryDiscounts to Employee
GRANT SELECT ON IndividualClientsActivePermanentDiscounts to Employee
GRANT SELECT ON ReservationsRaport to Employee
GRANT SELECT ON MenuRaport to Employee
GRANT SELECT ON PermanentDiscountsRaport to Employee
GRANT SELECT ON TemporaryDiscountsRaport to Employee
GRANT SELECT ON IndividualClientsOrdersRaport to Employee
GRANT SELECT ON CompaniesOrdersRaport to Employee
GRANT EXECUTE ON sp_InsertIndividualClient to Employee
GRANT EXECUTE ON sp_InsertCompany to Employee
GRANT EXECUTE ON sp_AddEmployeeToOrder  to Employee
GRANT EXECUTE ON sp_ConfirmReservation to Employee
GRANT EXECUTE ON sp_PayForReservation to Employee
GRANT EXECUTE ON sp_CancelReservation to Employee
GRANT EXECUTE ON sp_AddProductToOrder to Employee
GRANT EXECUTE ON sp_RemoveProductToOrder  to Employee
GRANT EXECUTE ON sp_ChangeProductQuantityInOrder to Employee
GRANT EXECUTE ON sp_MarkOrderAsRealized to Employee
GRANT EXECUTE ON sp_MarkOrderAsNotRealized to Employee
GRANT EXECUTE ON sp_FindTable to Employee
GRANT EXECUTE ON sp_AddTableToOneOfCompaniesTables to Employee
GRANT EXECUTE ON sp_AddTableToReservation to Employee
GRANT EXECUTE ON sp_CreateOrderInPlace to Employee
GRANT EXECUTE ON sp_AddOrderToCompaniesReservationTable to Employee
GRANT EXECUTE ON sp_AddCompanyReservationTable to Employee
```

5. **ShiftManager**
   menedżer zmiany, oprócz uprawnień podanych poniżej posiada także uprawnienia Employee.

```
CREATE ROLE ShiftManager AUTHORIZATION dbo
GRANT SELECT ON RealizedTodaysOrders to ShiftManager
GRANT SELECT ON RealizedThisWeekOrders to ShiftManager
GRANT SELECT ON RealizedThisMonthOrders to ShiftManager
GRANT SELECT ON RealizedThisYearOrders to ShiftManager
GRANT SELECT ON TodaysReservationsValues to ShiftManager
GRANT SELECT ON ThisMonthReservationsValues to ShiftManager
GRANT SELECT ON ThisYearReservationsValues to ShiftManager
GRANT SELECT ON IndividualClientsList to ShiftManager
GRANT SELECT ON CompaniesList to ShiftManager
GRANT EXECUTE ON sp_InsertProductAvailability to ShiftManager
```

6. **Owner**
   właściciel, oprócz uprawnień podanych poniżej posiada także uprawnienia ShiftManager'a oraz Employee.

```
CREATE ROLE Owner AUTHORIZATION dbo
GRANT SELECT ON ActualConstantsValues to Owner
GRANT SELECT ON RealisedOrdersPerEmployee to Owner
GRANT SELECT ON Top5MostFrequentlyPurchasedProducts to Owner
GRANT SELECT ON Top5MostFrequentlyOrderingIndividualClients to Owner
GRANT SELECT ON Top5MostFrequentlyOrderingCompanies to Owner
GRANT SELECT ON Top5MostFrequentlyOrderingIndividualClientWithPayment to Owner
GRANT SELECT ON Top5MostFrequentlyOrderingCompaniesWithPayment to Owner
GRANT SELECT ON Top5MostExpensiveOrdersFromIndividualClients to Owner
GRANT SELECT ON Top5MostExpensiveOrdersFromCompanies to Owner
GRANT EXECUTE ON sp_InsertGlobalConst to Owner
GRANT EXECUTE ON sp_InsertPermanentDiscountParemeters to Owner
GRANT EXECUTE ON sp_InsertTemporaryDiscountParemeters to Owner
GRANT EXECUTE ON sp_AddEmployee to Owner
GRANT EXECUTE ON sp_FindProduct to Owner
GRANT EXECUTE ON sp_AddNewTable to Owner
```