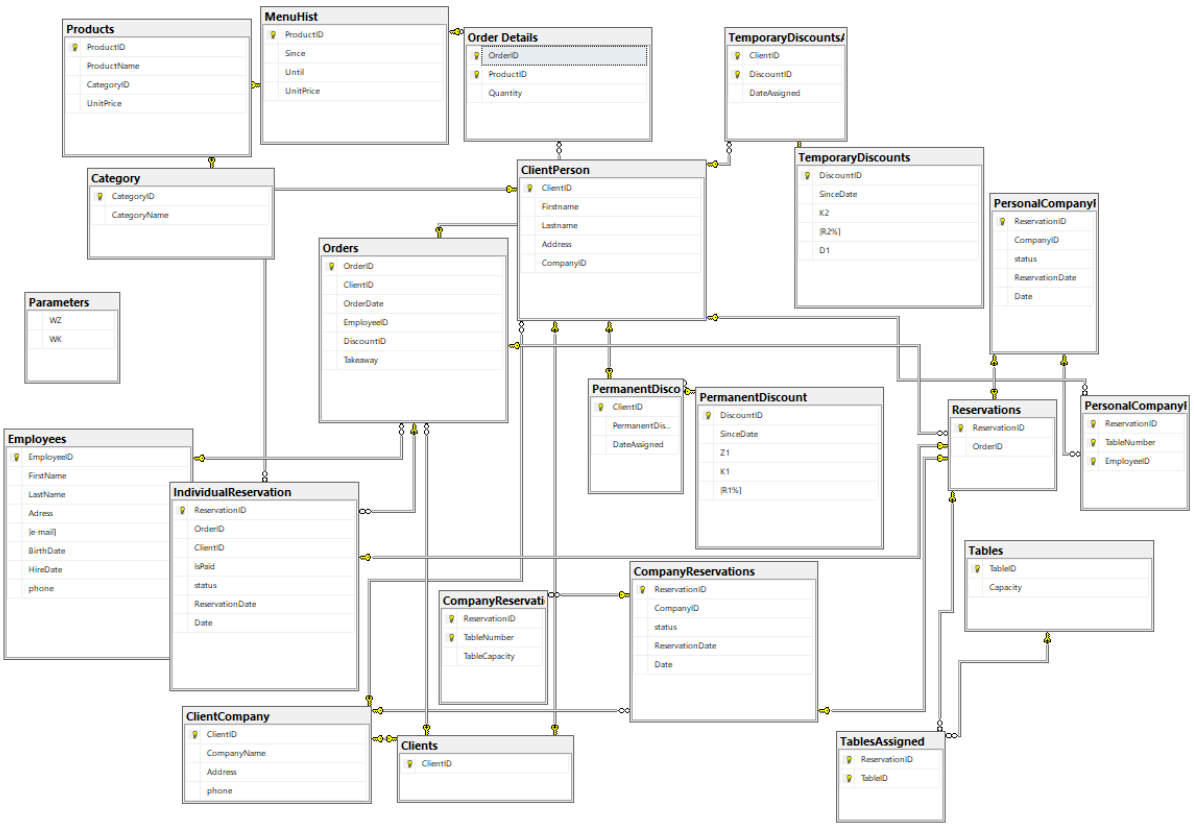
Dokumentacja dotycząca bazy restauracji

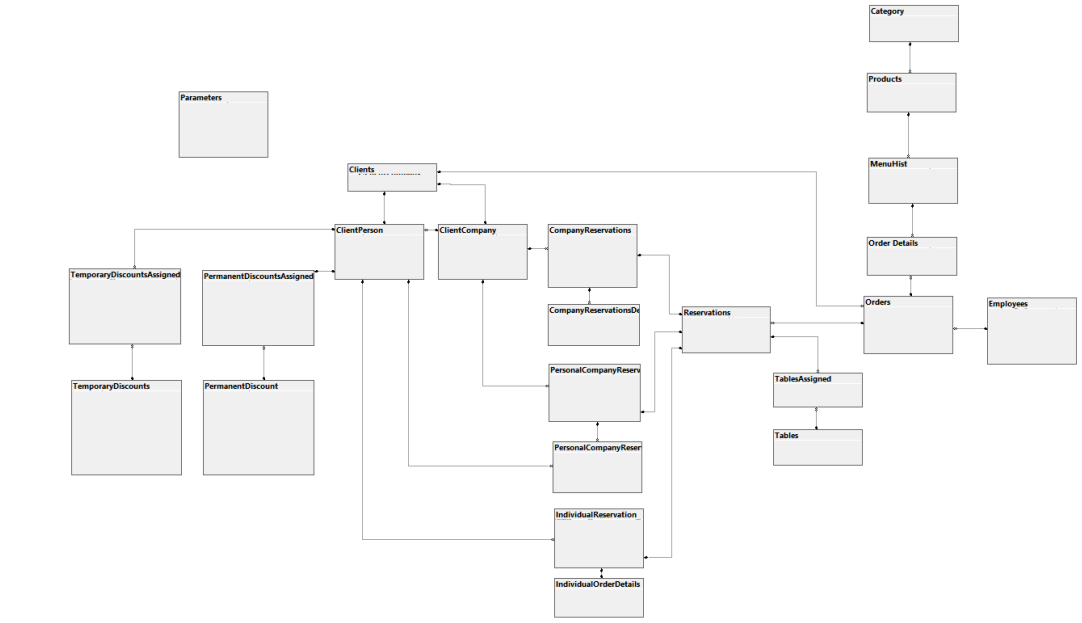
Piotr Ludynia, Hubert Obarzanek, Piotr Ignacy

Funkcje w systemie i uprawnienia poszczególnych użytkowników

- Administrator
    - Dostęp do wszystkich funkcji użytkowników oraz pełnej modyfikacji danych
    - możliwość zmiany roli pozostałych użytkowników, dodawania i usuwania nowych
    - możliwość ręcznego wykonania backupów bazy
- Zarządzający restauracją
    - ustalanie menu
    - ustalanie cen produktów
    - ustalanie minimalnej kwoty wymaganej do przyznania zniżki,
    - ustalanie minimalnej liczby zamówień wymaganej do przyznania zniżki
    - ustalanie wysokości zniżek
    - zlecenie wygenerowania przez system raportu dotyczącego zajętości stolików, zrealizowanych rabatów lub obowiązującego menu dla wybranego tygodnia/miesiąca
    - dodawanie produktów i kategorii, nowych stolików
    - posiada wszystkie uprawnienia zwykłego pracownika
    - dodawanie nowego pracownika do bazy
- Pracownik
    - autoryzacja i przyjmowanie rezerwacji
    - przyjmowanie zamówień na miejscu
    - wprowadzanie zamówień dokonywanych na miejscu do historii systemu
    - pobieranie informacji o zamówieniach z systemu
    - zlecenie wygenerowania przez system raportu dotyczącego statystyk zamówień klienta/firmy na jego prośbę. (Raportów miesięcznych i tygodniowych, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia – dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień)
    - wpisywanie danych o zamówieniu
    - zlecenie wygenerowania faktury za dane zamówienie lub za ostatni miesiąc
- Klient Indywidualny
    - rezerwacja stolika na daną liczbę osób
    - składanie zamówień na wynos z wyprzedzeniem (przez stronę internetową)
    - rejestracja jako użytkownik systemu
    - opłacanie zamówień
- Klient firmowy
    - składanie zamówień dla pracowników (imiennie)
    - składanie zamówień na firmę
    - opłacanie zamówień
- System
    - funkcja obliczająca minimalną wartość zamówienia potrzebną do rezerwacji
    - obliczanie liczby wcześniejszych zamówień danego klienta
    - przydzielanie zniżek klientom po spełnieniu warunków (odpowiednio dla zniżek jednorazowych po realizacji zamówień na zadaną łączną kwotę, lub

zniżek na wszystkie zamówienia po zrealizowaniu odpowiedniej liczby zamówień na wymaganą minimalną kwotę)
- powiadamianie zarządzającego o konieczności zmiany menu na 2 dni przed upłynięciem terminu zmiany połowy pozycji
- powiadamianie o konieczności ustalenia lub zatwierdzenia menu na dzień przed
- przekazanie zamówienia złożonego przez internet do weryfikacji przez obsługę
- automatyczne informowanie klienta o akceptacji/odrzuceniu rezerwacji złożonej przez internet po decyzji pracownika
- automatyczny backup bazy raz dziennie o godzinie 4.00, kiedy restauracja jest zamknięta
- weryfikacja dostępności żądanej liczby stolików przy składaniu zamówienia
- sprawdzenie warunków koniecznych (np. minimalna wartość zamówienia lub odpowiedniej daty przy zamówieniu obejmującym owoce morza) przy rezerwacji składanej przez użytkownika w formularzu internetowym, oraz powiadomienie go w razie niespełnienia któregoś z nich.
- przydzielanie stolika do rezerwacji
- księgowanie opłat za zamówienia
- zapisywanie historii przychodów
- dodanie nowego ClientInividuals podczas składania zamówienia gdy taki client jeszcze nie istnieje w bazie





Tabele

1. **Category** - wszystkie kategorie zamawianych produktów
    a. CategoryID (int) - klucz główny kategorii,
        i. autoinkrementacja zaczynając od 1 (IDENTITY(1, 1))
        ii. nie może być nullem
    b. CategoryName (varchar(50)) - nazwa kategorii
        i. nie może być nullem
        ii. unikalny (UNIQUE)

```
CREATE TABLE [dbo].[Category](
    [CategoryID] [int] IDENTITY(1,1) NOT NULL,
    [CategoryName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Category] PRIMARY KEY CLUSTERED
(
    [CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

2. **Products** - wszystkie produkty mogące być kiedykolwiek serwowane w restauracji
    a. ProductID (int) - klucz główny produktu
        i. nie może być nullem
        ii. autoinkrementacja zaczynając od 1 (IDENTITY(1, 1))

b. ProductName (varchar(50)) - nazwa produktu
    i. nie może być nullem
    ii. unikalny (UNIQUE)
c. CategoryID (int) - informacja jakiej kategorii jest dany produkt
    i. nie może być nullem
    ii. klucz obcy do tabeli Categories
d. UnitPrice (money) - cena danego produktu
    i. nie może być nullem
    ii. wartość musi być nieujemna (CHECK (UnitPrice >= 0))

```
CREATE TABLE [dbo].[Products](
    [ProductID] [int] IDENTITY(1,1) NOT NULL,
    [ProductName] [varchar](50) NOT NULL,
    [CategoryID] [int] NOT NULL,
    [UnitPrice] [money] NOT NULL,
 CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
    [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_Category] FOREIGN KEY
([CategoryID]) REFERENCES [dbo].[Category] ([CategoryID])
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_Category]
GO
ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_MenuHist]
FOREIGN KEY([ProductID]) REFERENCES [dbo].[MenuHist] ([ProductID])
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_MenuHist]
GO
```

3. **MenuHist** - tabela informująca w jakim okresie dany produkt był dostępny w Menu
    a. ProductID (int) - klucz obcy wskazujący na produkt
        i. nie może być null
    b. Since (date) - data, od kiedy dany produkt był/jest dostępny w Menu
        i. nie może być null
    c. Until (date) - data, do kiedy dany produkt był dostępny w Menu
        i. może być null (w takim przypadku oznacza to, że dany produkt jest dalej dostępny w Menu)
        ii. nie wcześniejsza data niż Since
    d. UnitPrice (money) - cena, w jakiej dany produkt był dostępny w Menu
        i. nie może być null
        ii. wartość musi być nieujemna (CHECK (UnitPrice >= 0))

```
CREATE TABLE [dbo].[MenuHist](
    [ProductID] [int] NOT NULL,
    [Since] [date] NOT NULL,
    [Until] [date] NULL,
    [UnitPrice] [money] NOT NULL,
 CONSTRAINT [PK_MenuHist] PRIMARY KEY CLUSTERED
(
    [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

4. **Orders** - informacje o danym zamówieniu
    a. OrderID (int) - klucz główny zamówienia
        i. nie może być null
    b. ClientID (int) - informuje który klient złożył zamówienie
        i. nie może być null
        ii. klucz obcy do tabeli Clients
    c. OrderDate (date) - data zamówienia
        i. nie może być null
    d. EmployeeID (int) - informuje, który pracownik restauracji obsługiwał zamówienie
        i. nie może być null
        ii. klucz obcy do tabeli Employees
    e. Takeaway (bit) - czy na wynos (0 - nie, 1 - tak)
        i. nie może być null
        ii. domyślna wartość 0
    f. DiscountVallue (real) - informuje o wartości zniżki w zamuwieniu
        i. domyślna wartość 0
        ii. nie może być null

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

5. **Order Details** - szczegółowe informacje o danym zamówieniu
    a. OrderID (int) - jedno z dwóch pól klucza głównego
        i. nie może być null
        ii. jednocześnie klucz obcy do tabeli Orders
    b. ProductID (int) - drugie pole klucza głównego
        i. nie może być null
        ii. klucz obcy do tabeli Products
    c. Quantity (int) - informuje ile produktów o danym ProductID było w zamówieniu

i. nie może być null
ii. wartość dodatnia (CHECK (Quantity > 0))

```
CREATE TABLE [dbo].[Order Details](
    [OrderID] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
    [Quantity] [int] NOT NULL,
 CONSTRAINT [PK_Order Details] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC,
    [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Order Details]  WITH CHECK ADD  CONSTRAINT [FK_Order Details_MenuHist]
FOREIGN KEY([ProductID]) REFERENCES [dbo].[MenuHist] ([ProductID])
GO
ALTER TABLE [dbo].[Order Details] CHECK CONSTRAINT [FK_Order Details_MenuHist]
GO
ALTER TABLE [dbo].[Order Details]  WITH CHECK ADD  CONSTRAINT [FK_Order Details_Orders]
FOREIGN KEY([OrderID]) REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[Order Details] CHECK CONSTRAINT [FK_Order Details_Orders]
GO
```

6. **Employees** - informacje o pracownikach restauracji
    a. EmployeeID (int) - klucz główny tabeli
        i. autoinkrementacja
        ii. nie może być null
    b. FirstName (varchar(50)) - imię pracownika
        i. nie może być null
    c. LastName (varchar(50)) - nazwisko pracownika
        i. nie może być null
    d. Adress (varchar(50)) - adres zamieszkania pracownika
        i. nie może być null
    e. e-mail (varchar(50)) - adres e-mail pracownika
        i. nie może być null
        ii. unikalny (UNIQUE)
    f. BirthDate (date) - data urodzenia pracownika
        i. nie może być null
        ii. z zakresu 01-01-1900 do dzisiejszej daty (CHECK (BirthDate > ‘01-01-1900’ AND BirthDate < getdate()))
    g. HireDate (date) - data zatrudnienia pracownika
        i. nie może być null
        ii. po 01-01-1900 (CHECK (HireDate > ‘01-01-1900’)
    h. phone (varchar(50)) - numer telefonu pracownika
        i. nie może być null
        ii. unikalny (UNIQUE)

```
CREATE TABLE [dbo].[Employees](
    [EmployeeID] [int] IDENTITY(1,1) NOT NULL,
    [FirstName] [varchar](50) NOT NULL,
    [LastName] [varchar](50) NOT NULL,
    [Adress] [varchar](50) NOT NULL,
    [e-mail] [varchar](50) NOT NULL,
    [BirthDate] [date] NOT NULL,
    [HireDate] [date] NOT NULL,
    [phone] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
    [EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

7. **Parameters** - zawiera ustalone przez zarządcę restauracji stałe
    a. WZ (money) - minimalna wartość zamówienia przy składaniu zamówienia przez internetowy formularz
        i. nie może być null
        ii. wartość nieujemna - CHECK (WZ >= 0)
    b. WK (int) - minimalna ilość wcześniejszych zamówień aby można było złożyć internetowe zamówienie
        i. nie może być null
        ii. wartość nieujemna - CHECK (WK >= 0)

```
CREATE TABLE [dbo].[Parameters](
    [WZ] [money] NOT NULL,
    [WK] [int] NOT NULL
) ON [PRIMARY]
GO
```

8. **Clients** - zbiór wszystkich klientów indywidualnych i firmowych
    a. ClientID (int) - klucz główny tabeli

```
CREATE TABLE [dbo].[Clients](
    [ClientID] [int] IDENTITY(1,1) NOT 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]
) ON [PRIMARY]
GO
```

9. **ClientCompany** - klienci firmowi
    a. ClientID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy do tabeli Clients
    b. CompanyName (varchar(50)) - nazwa firmy
        i. nie może być null
        ii. unikalny (UNIQUE)

c. Adress (varchar(50)) - adres firmy
    i. nie może być null
d. phone (varchar(50)) - numer telefonu do firmy
    i. nie może być null
    ii. unikalny (UNIQUE)

```
CREATE TABLE [dbo].[ClientCompany](
    [ClientID] [int] NOT NULL,
    [CompanyName] [varchar](50) NOT NULL,
    [Address] [varchar](50) NOT NULL,
    [phone] [varchar](50) NOT NULL,
 CONSTRAINT [PK_ClientCompany] 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].[ClientCompany]  WITH CHECK ADD  CONSTRAINT [FK_ClientCompany_Clients]
FOREIGN KEY([ClientID]) REFERENCES [dbo].[Clients] ([ClientID])
GO
ALTER TABLE [dbo].[ClientCompany] CHECK CONSTRAINT [FK_ClientCompany_Clients]
GO
```

10. **ClientPerson** - wszyscy klienci indywidualni
    a. ClientID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy do tabeli Clients
    b. FirstName (varchar(50)) - imię klienta
        i. nie może być null
    c. Lastname (varchar(50)) - nazwisko klienta
        i. nie może być null
    d. Address (varchar(50)) - adres klienta
        i. nie może być null
    e. CompanyID (int) - oznacza, że ta osoba jest pracownikiem firmy o danym CompanyID
        i. klucz obcy z tabeli ClientCompany
        ii. może być null (wartość NULL, oznacza, że dany klient nie pracuje dla żadnej z firm z tabeli ClientCompany)

```
CREATE TABLE [dbo].[ClientPerson](
    [ClientID] [int] NOT NULL,
    [Firstname] [varchar](50) NOT NULL,
    [Lastname] [varchar](50) NOT NULL,
    [Address] [varchar](50) NOT NULL,
    [CompanyID] [int] NULL,
 CONSTRAINT [PK_ClientPerson] 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].[ClientPerson]  WITH CHECK ADD  CONSTRAINT [FK_ClientPerson_ClientCompany]
FOREIGN KEY([CompanyID]) REFERENCES [dbo].[ClientCompany] ([ClientID])
GO
ALTER TABLE [dbo].[ClientPerson] CHECK CONSTRAINT [FK_ClientPerson_ClientCompany]
GO
ALTER TABLE [dbo].[ClientPerson]  WITH CHECK ADD  CONSTRAINT [FK_ClientPerson_Clients]
FOREIGN KEY([ClientID]) REFERENCES [dbo].[Clients] ([ClientID])
GO
ALTER TABLE [dbo].[ClientPerson] CHECK CONSTRAINT [FK_ClientPerson_Clients]
GO
```

11. **Reservation** - zbiór wszystkich rezerwacji
    a. ReservationID (int) - klucz główny tabeli
        i. nie może być null
    b. OrderID (int) - gdy podczas dokonywania rezerwacji nie składa się zamówienia, ma wartość NULL
        i. może być null
        ii. klucz obcy do tabeli Orders

```
CREATE TABLE [dbo].[Reservations](
    [ReservationID] [int] IDENTITY(1,1) NOT NULL,
    [OrderID] [int] 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]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_Orders]
FOREIGN KEY([OrderID]) REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_Orders]
GO
```

12. **IndividualReservation** - rezerwacje dla klientów indywidualnych
    a. ReservationID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy do tabeli Reservation
    b. OrderID (int) - przypisuje do danej rezerwacji numer zamówienia
        i. nie może być null
        ii. klucz obcy do tabeli Orders
    c. ClientID (int) - przypisuje każdemu zamówieniu indywidualnemu indywidualnego klienta.

i. nie może być null
ii. klucz obcy do tabeli ClientPerson

d. IsPaid (bit) - informuje czy zamówienie zostało opłacone z góry (1- opłacone, 0- nieopłacone)
i. nie może być null

e. status (bit) - informuje czy zamówienie zostało potwierdzone przez pracownika restauracji (0 - niepotwierdzone, 1 - potwierdzone)
i. nie może być null
ii. domyślnie 0

f. ReservationDate (date) - data złożenia rezerwacji
i. nie może być null
ii. domyślnie aktualna data - DEFAULT getdate()

g. Date (date) - na kiedy firma chce zarezerwować stoliki
i. nie może być null
ii. nie może być wcześniejsza niż ReservationDate - CHECK (Date >= ReservationDate)

```
CREATE TABLE [dbo].[IndividualReservation](
    [ReservationID] [int] IDENTITY(1,1) NOT NULL,
    [OrderID] [int] NOT NULL,
    [ClientID] [int] NOT NULL,
    [IsPaid] [bit] NOT NULL,
    [status] [bit] NOT NULL,
    [ReservationDate] [date] NOT NULL,
    [Date] [date] NOT NULL,
 CONSTRAINT [PK_IndividualReservation] 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].[IndividualReservation] ADD  CONSTRAINT [DF_IndividualReservation_status]  DEFAULT ((0)) FOR [status]
GO
ALTER TABLE [dbo].[IndividualReservation]  WITH CHECK ADD  CONSTRAINT [FK_IndividualReservation_ClientPerson] FOREIGN KEY([ClientID])
REFERENCES [dbo].[ClientPerson] ([ClientID])
GO
ALTER TABLE [dbo].[IndividualReservation] CHECK CONSTRAINT [FK_IndividualReservation_ClientPerson]
GO
ALTER TABLE [dbo].[IndividualReservation]  WITH CHECK ADD  CONSTRAINT [FK_IndividualReservation_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[IndividualReservation] CHECK CONSTRAINT [FK_IndividualReservation_Orders]
GO
ALTER TABLE [dbo].[IndividualReservation]  WITH CHECK ADD  CONSTRAINT [FK_IndividualReservation_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[IndividualReservation] CHECK CONSTRAINT [FK_IndividualReservation_Reservations]
GO
```

13. **PersonalCompanyReservations** - rezerwacje dla firm, gdzie firma wskazuje pracowników którzy przybędą
    a. ReservationID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy do tabeli Reservations
    b. CompanyID (int) - klucz obcy do tabeli ClientCompany, informuje jaka firma składa zamówienie
        i. nie może być null
    c. status (bit) - informuje czy zamówienie zostało potwierdzone przez obsługę restauracji (0 - niepotwierdzone, 1 - potwierdzone)
        i. nie może być null
        ii. domyślnie 0 - DEFAULT 0
    d. ReservationDate (date) - data złożenia rezerwacji

i. nie może być null
ii. domyślnie aktualna data - DEFAULT getdate()

e. Date (date) - na kiedy firma chce zarezerwować stoliki
i. nie może być null
ii. nie może być wcześniejsza niż ReservationDate - CHECK (Date >= ReservationDate)

```
CREATE TABLE [dbo].[PersonalCompanyReservations](
    [ReservationID] [int] NOT NULL,
    [CompanyID] [int] NOT NULL,
    [status] [bit] NOT NULL,
    [ReservationDate] [date] NOT NULL,
    [Date] [date] NOT NULL,
 CONSTRAINT [PK_PersonalCompanyReservations] 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].[PersonalCompanyReservations]  WITH CHECK ADD  CONSTRAINT
[FK_PersonalCompanyReservations_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[PersonalCompanyReservations] CHECK CONSTRAINT
[FK_PersonalCompanyReservations_Reservations]
GO
```

14. **PersonalCompanyReservationDetails** - szczegóły rezerwacji złożonej przez firmę dla konkretnych pracowników
    a. ReservationID (int) - jedno z 3 pól klucza głównego
        i. nie może być null
        ii. klucz obcy do tabeli PersonalCompanyReservations
    b. TableNumber (int) - drugie pole klucza głównego, numer stolika przy którym firma chce posadzić poszczególnych pracowników
        i. nie może być null
    c. EmployeeID (int) - trzecie pole klucza głównego, informuje który pracownik danej firmy ma siedzieć przy danym stoliku
        i. klucz obcy do tabeli ClientPerson
        ii. nie może być null

```
CREATE TABLE [dbo].[PersonalCompanyReservationDetails](
    [ReservationID] [int] NOT NULL,
    [TableNumber] [int] NOT NULL,
    [EmployeeID] [int] NOT NULL,
 CONSTRAINT [PK_PersonalCompanyReservationDetails] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [TableNumber] ASC,
    [EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[PersonalCompanyReservationDetails]  WITH CHECK ADD  CONSTRAINT
[FK_PersonalCompanyReservationDetails_ClientPerson] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[ClientPerson] ([ClientID])
GO
ALTER TABLE [dbo].[PersonalCompanyReservationDetails] CHECK CONSTRAINT
[FK_PersonalCompanyReservationDetails_ClientPerson]
GO
ALTER TABLE [dbo].[PersonalCompanyReservationDetails]  WITH CHECK ADD  CONSTRAINT
[FK_PersonalCompanyReservationDetails_PersonalCompanyReservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[PersonalCompanyReservations] ([ReservationID])
GO
ALTER TABLE [dbo].[PersonalCompanyReservationDetails] CHECK CONSTRAINT
[FK_PersonalCompanyReservationDetails_PersonalCompanyReservations]
GO
```

15. **CompanyReservations** - rezerwacje dla firm, bez wskazywania konkretnych pracowników
    a. ReservationID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy do tabeli Reservations
    b. CompanyID (int) - informuje która firma składa zamówienie
        i. nie może być null
        ii. klucz obcy do tabeli ClientCompany
    c. status (bit) - informuje czy rezerwacja została zatwierdzona przez obsługę restauracji (0 - niezatwierdzona, 1 - zatwierdzona)
        i. nie może być null
        ii. default 0
    d. ReservationDate (date) - data składania rezerwacji
        i. nie może być nulli
        ii. domyślnie ustawiana jako dziś - DEFAULT getdate()
    e. Date (date) - data, na kiedy firma chce zarezerwować stoliki
        i. nie może być null
        ii. nie może być wcześniejsza niż ReservationDate - CHECK (Date >= ReservationDate)

```
CREATE TABLE [dbo].[CompanyReservations](
    [ReservationID] [int] IDENTITY(1,1) NOT NULL,
    [CompanyID] [int] NOT NULL,
    [status] [bit] NOT NULL,
    [ReservationDate] [date] NOT NULL,
    [Date] [date] NOT NULL,
 CONSTRAINT [PK_CompanyReservations] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservations_ClientCompany] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[ClientCompany] ([ClientID])
GO
ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
[FK_CompanyReservations_ClientCompany]
GO
ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservations_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [FK_CompanyReservations_Reservations]
GO
```

16. **CompanyReservationsDetails** - szczegóły rezerwacji złożonej przez firmę bez wskazywania konkretnych pracowników
    a. ReservationID (int) - jeden z dwóch pól klucza głównego
        i. nie może być null
        ii. jednocześnie klucz obcy do tabeli CompanyReservations
    b. TableNumber (int) - drugie pole klucza głównego, informuje o numerze stolika który chce zarezerwować firma
        i. nie może być null
    c. TableCapacity (int) - pojemność stolika jaką chce firma
        i. musi być większa niż 0 - CHECK (TableCapacity > 0)
        ii. nie może być null

```
CREATE TABLE [dbo].[CompanyReservationsDetails](
    [ReservationID] [int] NOT NULL,
    [TableNumber] [int] IDENTITY(1,1) NOT NULL,
    [TableCapacity] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationsDetails] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [TableNumber] 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].[CompanyReservationsDetails]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservationsDetails_CompanyReservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservations] ([ReservationID])
GO
ALTER TABLE [dbo].[CompanyReservationsDetails] CHECK CONSTRAINT
[FK_CompanyReservationsDetails_CompanyReservations]
GO
```

17. **Tables** - zbiór wszystkich stolików dostępnych w restauracji
    a. TableID (int) - klucz główny, numer stolika
        i. nie może być null
    b. Capacity (int) - pojemność danego stolika
        i. musi być większa niż 0 - CHECK (Capacity > 0)
        ii. nie może być null

```
CREATE TABLE [dbo].[Tables](
    [TableID] [int] IDENTITY(1,1) NOT NULL,
    [Capacity] [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
```

18. **TablesAssigned** - gdy rezerwacja zostanie potwierdzona, przypisuje konkretne stoliki do rezerwacji
    a. ReservationID (int) - jedno z dwóch pól klucza głównego
        i. nie może być null
        ii. klucz obcy do tabeli Reservations
    b. TableID (int) - drugie pole klucza głównego
        i. nie może być null
        ii. klucz obcy do tabeli Tables

```
CREATE TABLE [dbo].[TablesAssigned](
    [ReservationID] [int] NOT NULL,
    [TableID] [int] NOT NULL,
 CONSTRAINT [PK_TablesAssigned] PRIMARY KEY CLUSTERED
(
    [ReservationID] ASC,
    [TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[TablesAssigned]  WITH CHECK ADD  CONSTRAINT [FK_TablesAssigned_Reservations]
FOREIGN KEY([ReservationID]) REFERENCES [dbo].[Reservations] ([ReservationID])
GO
ALTER TABLE [dbo].[TablesAssigned] CHECK CONSTRAINT [FK_TablesAssigned_Reservations]
GO
ALTER TABLE [dbo].[TablesAssigned]  WITH CHECK ADD  CONSTRAINT [FK_TablesAssigned_Tables]
FOREIGN KEY([TableID]) REFERENCES [dbo].[Tables] ([TableID])
GO
ALTER TABLE [dbo].[TablesAssigned] CHECK CONSTRAINT [FK_TablesAssigned_Tables]
GO
```

19. **TemporaryDiscountsAssigned** - tabela zawierająca przyporządkowania zdobytych zniżek jednorazowych klientom wraz z datą przyznania
    a. ClientID (int) - jedno z dwóch pól należących do klucza głównego tabeli
        i. nie może być null
        ii. klucz obcy z tabeli ClientPerson
    b. DiscountID (int) - drugie z pól składających się na klucz główny
        i. nie może być null

ii. klucz obcy z tabeli TemporaryDiscounts

c. DateAssigned (date) - data przyznania tymczasowej zniżki
   i. nie może być null
   ii. nie może być późniejsza niż data dzisiejsza - CHECK (DateAssigned <= getdate())

```
CREATE TABLE [dbo].[TemporaryDiscountsAssigned](
    [ClientID] [int] NOT NULL,
    [DiscountID] [int] NOT NULL,
    [DateAssigned] [date] NOT NULL,
 CONSTRAINT [PK_DiscountsAssigned] PRIMARY KEY CLUSTERED
(
    [ClientID] ASC,
    [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[TemporaryDiscountsAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_DiscountsAssigned_ClientPerson] FOREIGN KEY([ClientID])
REFERENCES [dbo].[ClientPerson] ([ClientID])
GO
ALTER TABLE [dbo].[TemporaryDiscountsAssigned] CHECK CONSTRAINT
[FK_DiscountsAssigned_ClientPerson]
GO
ALTER TABLE [dbo].[TemporaryDiscountsAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_TemporaryDiscountsAssigned_TemporaryDiscounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[TemporaryDiscounts] ([DiscountID])
GO
ALTER TABLE [dbo].[TemporaryDiscountsAssigned] CHECK CONSTRAINT
[FK_TemporaryDiscountsAssigned_TemporaryDiscounts]
GO
```

20. **TemporaryDiscounts** - tabela zawierająca dane dotyczące zniżek tymczasowych
    a. DiscountID (int) - klucz główny dla zniżki tymczasowej
       i. nie może być null
    b. SinceDate (date) - data od kiedy obowiązuje dana zniżka
       i. nie może być null
       ii. nie może być późniejsza niż data dzisiejsza - CHECK (SinceDate <= getdate())
    c. K2 (money) - łączna kwota na którą należy zrealizować zamówienia
       i. nie może być null
       ii. musi być większa od 0
    d. R2% (real) - procentowa wartość zniżki
       i. nie może być null
       ii. musi być w zakresie (0,1> - CHECK ([R2%] <= 1 AND [R2%] > 0)
    e. D1 (int) - długość ważności zniżki (w dniach)
       i. nie może być null
       ii. musi być większa od 0 - CHECK (D1 > 0)

```
CREATE TABLE [dbo].[TemporaryDiscounts](
    [DiscountID] [int] NOT NULL,
    [SinceDate] [date] NOT NULL,
    [K2] [money] NOT NULL,
    [R2%] [real] NOT NULL,
    [D1] [int] NOT NULL,
 CONSTRAINT [PK_OtherDiscounts] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

21. **PermanentDiscountsAssigned** - tabela zawierająca przyporządkowania zdobytych zniżek permanentnych klientom wraz z datą przyznania
    a. ClientID (int) - klucz główny
        i. nie może być null
        ii. klucz obcy z tabeli ClientPerson
    b. PermanentDiscountsAssigned (int) - ID przyznanej zniżki permanentnej
        i. nie może być null
        ii. klucz obcy z tabeli PermanentDiscount
    c. DateAssigned (date) - data przyznania zniżki
        i. nie może być null
        ii. nie może być późniejsza niż data dzisiejsza - CHECK (DateAssigned <= getdate())

```
CREATE TABLE [dbo].[PermanentDiscountsAssigned](
    [ClientID] [int] NOT NULL,
    [PermanentDiscountID] [int] NOT NULL,
    [DateAssigned] [date] NOT NULL,
 CONSTRAINT [PK_PermanentDiscountsAssigned] 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].[PermanentDiscountsAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscountsAssigned_ClientPerson] FOREIGN KEY([ClientID])
REFERENCES [dbo].[ClientPerson] ([ClientID])
GO
ALTER TABLE [dbo].[PermanentDiscountsAssigned] CHECK CONSTRAINT
[FK_PermanentDiscountsAssigned_ClientPerson]
GO
ALTER TABLE [dbo].[PermanentDiscountsAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscountsAssigned_PermanentDiscount] FOREIGN KEY([PermanentDiscountID])
REFERENCES [dbo].[PermanentDiscount] ([DiscountID])
GO
ALTER TABLE [dbo].[PermanentDiscountsAssigned] CHECK CONSTRAINT
[FK_PermanentDiscountsAssigned_PermanentDiscount]
GO
```

22. **PermanentDiscount** - tabela zawierająca dane dotyczące zniżek permanentnych
    a. DiscountID (int) - klucz główny, identyfikator zniżki
        i. nie może być null
    b. SinceDate (date) - data od kiedy obowiązuje dana zniżka

i. nie może być null

c. Z1 (int) - minimalna liczba zamówień aby zniżka była przyznana
   i. nie może być null
   ii. większa od 0 - CHECK (Z1 > 0)

d. K1 (money) - minimalna wartość zamówienia aby zamówienie zostało wliczone jako jedno z Z1 zamówień do zniżki
   i. nie może być null
   ii. większa od 0 - CHECK (K1 > 0)

e. R1% (real) - procentowa wartość zniżki
   i. nie może być null
   ii. w zakresie (0, 1> - CHECK ([R1%] > 0 AND [R1%] <= 1)

```
CREATE TABLE [dbo].[PermanentDiscount](
    [DiscountID] [int] IDENTITY(1,1) NOT NULL,
    [SinceDate] [date] NOT NULL,
    [Z1] [int] NOT NULL,
    [K1] [money] NOT NULL,
    [R1%] [real] NOT NULL,
 CONSTRAINT [PK_PermanentDiscount] PRIMARY KEY CLUSTERED
(
    [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Warunki integralnościowe

```sql
ALTER TABLE Products
ADD UNIQUE (ProductName)

ALTER TABLE Products
ADD CHECK (UnitPrice >= 0)

ALTER TABLE MenuHist
ADD CHECK (UnitPrice >= 0)

ALTER TABLE MenuHist
ADD CHECK (Since <= Until)

ALTER TABLE [Order Details]
ADD CHECK (Quantity > 0)

ALTER TABLE Employees
ADD UNIQUE ([e-mail])

ALTER TABLE Employees
ADD CHECK (BirthDate > '01-01-1900' AND BirthDate < getdate())

ALTER TABLE Employees
ADD CHECK (HireDate > '01-01-1900')

ALTER TABLE Employees
ADD UNIQUE (phone)

ALTER TABLE Parameters
ADD CHECK (WZ >= 0)

ALTER TABLE Parameters
ADD CHECK (WK >= 0)

ALTER TABLE ClientCompany
ADD UNIQUE (CompanyName)

ALTER TABLE ClientCompany
ADD UNIQUE (Phone)

ALTER TABLE IndividualReservation
ADD DEFAULT getdate() FOR ReservationDate

ALTER TABLE IndividualReservation
ADD CHECK (Date >= ReservationDate)
```

```sql
ALTER TABLE PersonalCompanyReservations
ADD DEFAULT getdate() FOR ReservationDate

ALTER TABLE PersonalCompanyReservations
ADD CHECK (Date >= ReservationDate)

ALTER TABLE CompanyReservations
ADD DEFAULT getdate() FOR ReservationDate

ALTER TABLE CompanyReservations
ADD CHECK (Date >= ReservationDate)

ALTER TABLE CompanyReservationsDetails
ADD CHECK (TableCapacity > 0)

ALTER TABLE TemporaryDiscountsAssigned
ADD CHECK (DateAssigned <= getdate())

ALTER TABLE TemporaryDiscounts
ADD CHECK (SinceDate <= getdate())

ALTER TABLE TemporaryDiscounts
ADD CHECK ([R2%] > 0 AND [R2%] <= 1)

ALTER TABLE TemporaryDiscounts
ADD CHECK (D1 > 0)

ALTER TABLE PermanentDiscountsAssigned
ADD CHECK (DateAssigned <= getdate())

ALTER TABLE PermanentDiscount
ADD CHECK (Z1 > 0)

ALTER TABLE PermanentDiscount
ADD CHECK (K1 > 0)

ALTER TABLE PermanentDiscount
ADD CHECK ([R1%] > 0 AND [R1%] <= 1)
```

Widoki

Widoki dotyczące menu

1. MenuView - pokazuje aktualne pozycje w menu

```
create view [dbo].[MenuView] as
select products.ProductName,category.CategoryName,menuhist.UnitPrice
from menuhist left join products on menuhist.productid=products.productid
left join category on products.categoryid=category.categoryid
where menuhist.until is null and menuhist.Since < getdate()
GO
```

2. MenuHistView - lista dostępnych produktów wraz z okresami w jakich były dostępne i cenami

```
CREATE VIEW [dbo].[MenuHistView]
AS
SELECT        dbo.MenuHist.MenuHistID, dbo.Products.ProductName,
dbo.MenuHist.Since, dbo.MenuHist.Until, dbo.MenuHist.UnitPrice
FROM            dbo.MenuHist INNER JOIN
                         dbo.Products ON dbo.MenuHist.ProductID = dbo.Products.ProductID
GO
```

Widoki dotyczące zamówień

Zamówienia firmowe

3. CompanyClientReport - raport dla klienta firmowego dotyczący szczegółów wszystkich jego zamówień pokazujący zamówione produkty, datę zamówienia, ceny produktów i ich liczbę w zamówieniu

```
CREATE VIEW [dbo].[CompanyClientReport]
AS
SELECT        dbo.ClientCompany.CompanyName, Orders.OrderID, dbo.Orders.OrderDate, dbo.Products.ProductName, dbo.[Order Details].Quantity
FROM            dbo.ClientCompany INNER JOIN
                         dbo.Orders ON dbo.ClientCompany.ClientID = dbo.Orders.ClientID INNER JOIN
                         dbo.[Order Details] ON dbo.Orders.OrderID = dbo.[Order Details].OrderID INNER JOIN
                         dbo.Products ON dbo.Products.ProductID = dbo.[Order Details].MenuHistID INNER JOIN
                         dbo.MenuHist ON dbo.Products.ProductID = dbo.MenuHist.ProductID
GO
```

4. CompanyOrdersView - lista zamówień firmowych z ich łączną wartością

```
create view [dbo].[CompanyOrdersView] as
    select OrderDate,CompanyName,sum(quantity*UnitPrice) as TotalValue from orders
    join ClientCompany on orders.ClientID=ClientCompany.ClientID
    join [Order Details] on orders.OrderID=[Order Details].OrderID
    join MenuHist on MenuHist.MenuHistID=[Order Details].MenuHistID
    group by orders.OrderID,OrderDate,CompanyName
GO
```

5. CompanyOrdersSumView - łączna wartość zamówień klientów firmowych

```
create view [dbo].[CompanyOrdersSumView] as
    select CompanyName,sum(Quantity*UnitPrice) as [Total value] from orders
    join ClientCompany on orders.ClientID=ClientCompany.ClientID
    join [Order Details] on orders.OrderID=[Order Details].OrderID
    join MenuHist on MenuHist.MenuHistID=[Order Details].MenuHistID
    group by CompanyName
GO
```

Zamówienia indywidualne

6. IndividualOrdersView - lista klientów wraz z poszczególnymi produktami zamówionymi przez nich i informacją z jakiego zamówienia jest dany produkt

```
CREATE VIEW [dbo].[IndividualOrdersView]
AS
SELECT        dbo.Orders.ClientID, dbo.ClientPerson.Firstname, dbo.ClientPerson.Lastname,
              dbo.Orders.OrderID, dbo.Products.ProductName, dbo.[Order Details].Quantity,
              dbo.Orders.DiscountValue, dbo.Orders.OrderDate,
                        dbo.Orders.Takeaway, dbo.MenuHist.UnitPrice
FROM            dbo.Orders LEFT OUTER JOIN
                        dbo.[Order Details] ON dbo.Orders.OrderID = dbo.[Order Details].OrderID INNER JOIN
                        dbo.ClientPerson ON dbo.Orders.ClientID = dbo.ClientPerson.ClientID LEFT OUTER JOIN
                        dbo.MenuHist ON dbo.[Order Details].MenuHistID = dbo.MenuHist.ProductID LEFT OUTER JOIN
                        dbo.Products ON dbo.MenuHist.ProductID = dbo.Products.ProductID
WHERE        (dbo.Products.ProductName IS NOT NULL)
GO
```

7. IndividualOrdersValuesView - widok z podsumowaniem zamówień indywidualnych wraz z wartością zamówienia po uwzględnieniu zniżki

```
CREATE VIEW [dbo].[IndividualOrdersValuesView]
AS
SELECT        ClientID, Firstname, Lastname, OrderID, OrderDate, ROUND(SUM(UnitPrice * Quantity) * (1 - DiscountValue), 2) AS OrderValue
FROM            dbo.IndividualOrdersView
GROUP BY ClientID, Firstname, Lastname, OrderID, OrderDate, DiscountValue
GO
```

Widoki dotyczące stolików

8. TableUsageView - raport o wykorzystaniu poszczególnych stolików

```
CREATE VIEW [dbo].[TableUsageView]
AS
SELECT        dbo.Tables.TableID, dbo.Tables.Capacity, dbo.Orders.OrderDate
FROM            dbo.TablesAssigned INNER JOIN
                        dbo.Reservations ON dbo.TablesAssigned.ReservationID = dbo.Reservations.ReservationID INNER JOIN
                        dbo.Tables ON dbo.TablesAssigned.TableID = dbo.Tables.TableID INNER JOIN
                        dbo.Orders ON dbo.Reservations.OrderID = dbo.Orders.OrderID
GO
```

9. ReservationTablesView - rezerwacje wraz z informacją ile i dla ilu osób stolików potrzeba (wykorzystuje 3 widoki - na zamówienia klientów indywidualnych, firmowych i firmowych imiennych (IndividualReservationTablesView, CompanyReservationTablesView, PersonalCompanyReservationTablesView).

```
CREATE VIEW [dbo].[ReservationTablesView]
AS
SELECT        ReservationID, TableCapacity
FROM            (SELECT        ReservationID, TableCapacity
                         FROM            dbo.CompanyReservationTablesView
                         UNION
                         SELECT        ReservationID, TableCapacity
                         FROM            dbo.IndividualReservationTablesView
                         UNION
                         SELECT        ReservationID, TableCapacity
                         FROM            dbo.PersonalCompanyReservationTablesView) AS AllReservationTablesInfo
GO
```

10. AssignedTablesDates - lista stolików przpisanych do danych zamówień

```
CREATE VIEW [dbo].[AssignedTablesDates]
AS
SELECT        dbo.Reservations.ReservationID, AllReservations.Date, dbo.TablesAssigned.TableID
FROM            dbo.TablesAssigned INNER JOIN
                         dbo.Reservations ON dbo.TablesAssigned.ReservationID = dbo.Reservations.ReservationID INNER JOIN
                             (SELECT        ReservationID, Date
                              FROM            dbo.IndividualReservation
                              UNION
                              SELECT        ReservationID, Date
                              FROM            dbo.CompanyReservations
                              UNION
                              SELECT        ReservationID, Date
                              FROM            dbo.PersonalCompanyReservations) AS AllReservations ON AllReservations.ReservationID = dbo.Reservations.ReservationID
GO
```

Widoki dotyczące zniżek

Zniżki tymczasowe

11. AllTemporaryDiscountsView - lista wszystkich przydzielonych zniżek tymczasowych

```
SELECT        dbo.TemporaryDiscountsAssigned.ClientID, dbo.TemporaryDiscounts.[R2%],
              dbo.TemporaryDiscountsAssigned.DateAssigned, DATEADD(DAY, dbo.TemporaryDiscounts.D1,
              dbo.TemporaryDiscountsAssigned.DateAssigned)
                          AS Expires
FROM            dbo.TemporaryDiscounts INNER JOIN
                         dbo.TemporaryDiscountsAssigned ON dbo.TemporaryDiscounts.DiscountID = dbo.TemporaryDiscountsAssigned.DiscountID
```

Zniżki permanentne

12. AllPermanentDiscountsView - lista wszystkich przydzielonych zniżek stałych

```
SELECT        dbo.PermanentDiscountsAssigned.ClientID, dbo.PermanentDiscount.[R1%],
              dbo.PermanentDiscountsAssigned.DateAssigned, NULL AS Expires
FROM            dbo.PermanentDiscount INNER JOIN
                         dbo.PermanentDiscountsAssigned ON dbo.PermanentDiscount.DiscountID = dbo.PermanentDiscountsAssigned.PermanentDiscountID
```

Widoki dotyczące rezerwacji

13. UnconfirmedTableReservations - wszystkie niepotwierdzone rezerwacje stolików

```
create view [dbo].[UnconfirmedTableReservations] as
    select ReservationID,CompanyID as ClientID, ReservationDate,Date from PersonalCompanyReservations
    where status=0
    union
    select ReservationID,ClientID,ReservationDate,Date from IndividualReservation
    where status=0
    union
    select ReservationID,CompanyID as ClientID,ReservationDate,Date from CompanyReservations
    where status=0
GO
```

Procedury

1. AddCategory - dodaje nową kategorię do tabeli

```
create procedure [dbo].[AddCategory](@name varchar(50)) as
begin
 insert into Category (CategoryName) values(@name);
 end;
 GO
```

2. AddClientCompany - dodaje nowego klienta firmowego

```
create procedure [dbo].[AddClientCompany] (
    @companyname varchar(50),
    @adres varchar(50),
    @phone varchar(50)
) as
begin
    insert into Clients default values;
    insert into ClientCompany values(@@identity,@companyname,@adres,@phone);
end
 GO
```

3. AddClientPerson - dodaje nowego klienta indywidualnego

```
create procedure [dbo].[AddClientPerson] (
    @firstname varchar(50),
    @lastname varchar(50),
    @adres varchar(50),
    @companyID int=null
) as
begin
    insert into Clients default values;
    insert into ClientPerson values(@@IDENTITY,@firstname,@lastname,@adres,@companyID);
end;
GO
```

4. AddEmployee - dodaje nowego pracownika restauracji

```
CREATE procedure [dbo].[AddEmployee](
    @firstname varchar(50),
    @lastname varchar(50),
    @adress varchar(50),
    @email varchar(50),
    @birthdate date,
    @hiredate as datetime=null,
    @phone varchar(50)
    ) as
begin
    begin try
        if @hiredate is null
            set @hiredate=getdate();
        insert into Employees values (@firstname,@lastname,@adress,@email,@birthdate,@hiredate,@phone);
    end try
    begin catch
        print 'zle dane'
    end catch
end;

GO
```

5. AddCompanyReservation - dodaje nową rezerwacje dla klienta firmowego

```
CREATE PROCEDURE [dbo].[AddCompanyReservation]
    @CompanyID INT,
    @Date DATE
AS
BEGIN
    EXEC AddOrder @clientID=@CompanyID, @orderDate=@Date;

    DECLARE @OrderID INT = (SELECT TOP 1 OrderID FROM Orders WHERE ClientID = @CompanyID
                            ORDER BY OrderID DESC)

    INSERT INTO Reservations(OrderID) VALUES (@OrderID)

    DECLARE @ReservationID INT = (SELECT ReservationID FROM Reservations WHERE OrderID = @OrderID)

    INSERT INTO CompanyReservations(ReservationID, CompanyID, ReservationDate, Date)
    VALUES (@ReservationID, @CompanyID, GETDATE(), @Date)
END
GO
```

6. AddCompanyReservationDetails - dodaje szczegóły rezerwacji dla klienta firmowego

```
CREATE PROCEDURE AddCompanyReservationDetails
    @ReservationID INT,
    @TableCapacity INT
AS
BEGIN
    INSERT INTO CompanyReservationDetails(ReservationID, TableCapacity) VALUES
    (@ReservationID, @TableCapacity)
END
GO
```

7. AddPersonalCompanyReservation - dodaje nowy wpis do Orders i Reservations i odpowiedni wpis w PersonalCompanyReservations

```
CREATE PROCEDURE [dbo].[AddPersonalCompanyReservation]
    @CompanyID INT,
    @Date DATE
AS
BEGIN
    EXEC AddOrder @clientID = @CompanyID, @orderDate = @Date;

    DECLARE @OrderID INT = (SELECT TOP 1 OrderID FROM Orders WHERE ClientID = @CompanyID
                            ORDER BY OrderID DESC)

    INSERT INTO Reservations(OrderID) VALUES (@OrderID)

    DECLARE @ReservationID INT = (SELECT ReservationID FROM Reservations WHERE OrderID = @OrderID)

    INSERT INTO PersonalCompanyReservations(ReservationID, CompanyID, ReservationDate, Date)
    VALUES (@ReservationID, @CompanyID, GETDATE(), @Date)
END
GO
```

8. AddPersonalCompanyReservationDetails - dodaje szczegóły rezerwacji (w postaci przypisania poszczególnych pracowników do stolików)

```
CREATE PROCEDURE AddPersonalCompanyReservationDetails
    @ReservationID INT,
    @TableNumber INT,
    @EmployeeID INT
AS
BEGIN
    INSERT INTO PersonalCompanyReservationDetails(ReservationID, TableNumber, EmployeeID)
    VALUES (@ReservationID, @TableNumber, @EmployeeID)
END
GO
```

9. AssignTable - przydziela konkretny stolik do zamówienia (wykonuje pracownik). Procedura sprawdza dodatkowo, czy zadany stolik nie jest już przydzielony do innego zamówienia w tym samym dniu.

```
CREATE PROCEDURE AssignTable
    @ReservationID INT,
    @TableID INT
AS
BEGIN

    DECLARE @Date DATE =
    (SELECT Date FROM IndividualReservation
     WHERE ReservationID = @ReservationID
     UNION
          SELECT Date FROM PersonalCompanyReservations
          WHERE ReservationID = @ReservationID
     UNION
          SELECT Date FROM CompanyReservations
          WHERE ReservationID = @ReservationID)
    DECLARE @TableFree BIT = ([dbo].IsTableFree(@TableID, @Date))

    IF (@TableFree = 1)
    BEGIN
        INSERT INTO TablesAssigned(ReservationID, TableID)
        VALUES (@ReservationID, @TableID)
    END
END
GO
```

10. AddMenu - dodaje nową pozycję do MenuHist

```
CREATE procedure [dbo].[AddMenu](
    @prodname varchar(50),
    @sincedate date,
    @untildate date=null,
    @unitprice money
)as
begin
    if (@sincedate > getdate())
    begin
        begin try
            declare @prodid int
            set @prodid=(select ProductID from Products where ProductName=@prodname);
            insert into MenuHist values(@prodid,@sincedate,@untildate,@unitprice);
        end try
        begin catch
            print 'bledne dane'
        end catch
    end
    else
    begin
        print 'poczatkowa data musi byc pozniejsza od dzisiejszej'
    end
end;
GO
```

11. RemoveFromMenu - usuwa zadaną pozycję z menu (korzystając z MenuHistID jako argumentu), lub jeśli zadanej pozycji nie ma w MenuHist - wypisuje odpowiedni komunikat

```
create procedure [dbo].[RemoveFromMenu](@menuhistid int, @untildate date) as
begin
    if exists (select * from MenuHist where MenuHistID = @menuhistid)
    begin
        update dbo.MenuHist
        set Until=@untildate
        where MenuHistID=@menuhistid
    end
    else
    begin
        print 'zadanej pozycji nie ma w menuhist'
    end
end
```

12. AddOrder - dodaje zamówienie

```
CREATE PROCEDURE [dbo].[AddOrder]( @clientID int, @orderDate date, @discountValue real=0, @takeaway bit=0)

AS
BEGIN
    if
        not exists (select * from clients
        inner join PermanentDiscountsAssigned on clients.ClientID = PermanentDiscountsAssigned.ClientID
        inner join PermanentDiscount on PermanentDiscountsAssigned.PermanentDiscountID = PermanentDiscount.DiscountID
        where clients.clientID = @clientID and PermanentDiscount.[R1%] = @discountValue)
        and not
        exists (select * from clients
        inner join TemporaryDiscountsAssigned on clients.ClientID = TemporaryDiscountsAssigned.ClientID
        inner join TemporaryDiscounts on TemporaryDiscountsAssigned.DiscountID = TemporaryDiscounts.DiscountID
        where clients.clientID = @clientID and TemporaryDiscounts.[R2%] = @discountValue and TemporaryDiscounts.SinceDate <= @orderDate and
        DATEADD(day, TemporaryDiscounts.D1,TemporaryDiscounts.SinceDate) >= @orderDate)
    begin
        set @discountValue = 0.0
    end
    begin
        if (@clientID in (SELECT ClientID FROM ClientCompany))
        begin
            SET @takeaway = 0
        end
        insert into Orders (ClientID,OrderDate,DiscountValue,Takeaway)
        values (@clientID,@orderDate,@discountValue,@takeaway)
    end


END
GO
```

13. AssignEmployee - Procedura przydzielająca pracownika do zamówienia - dodaje employeeID w orders

```
CREATE PROCEDURE [dbo].[AssignEmployee] (@employeeID int, @orderID int)

AS
BEGIN
    update Orders set EmployeeID = @employeeID
    where Orders.OrderID = @orderID
END
GO
```

14. AddOrderDetails - dodaje szczegóły zamówienia. Sprawdza czy produkt, który próbujemy dodać jest aktualnie dostępny w menu, a także - jeśli są to owoce morza - czy spełnione są warunki dotyczące odpowiednio wcześniejszego zamówienia.

```
CREATE PROCEDURE [dbo].[AddOrderDetails]
    @OrderID int,
    @MenuHistID int,
    @Quantity int
AS
BEGIN
    DECLARE @OrderDate DATE
    SET @OrderDate = (SELECT OrderDate FROM Orders WHERE OrderID=@OrderID)
    SET DATEFIRST 1
    IF (((SELECT CategoryName FROM Category INNER JOIN Products
        ON Products.CategoryID = Category.CategoryID INNER JOIN MenuHist
        ON Products.ProductID = MenuHist.ProductID WHERE MenuHistID = @MenuHistID) != 'Seafood') OR
        (DATEPART(WEEKDAY, @OrderDate) BETWEEN 4 AND 6 AND
        DATEDIFF(DAY, GETDATE(), @OrderDate) > DATEPART(WEEKDAY, @OrderDate)-1)) AND
        ((SELECT Until FROM MenuHist WHERE MenuHistID=@MenuHistID) IS NULL OR
        (SELECT Until FROM MenuHist WHERE MenuHistID=@MenuHistID) > GETDATE())
        AND (SELECT Since FROM MenuHist WHERE MenuHistID=@MenuHistID) < @OrderDate
    BEGIN
        INSERT INTO [Order Details](OrderID, MenuHistID, Quantity) VALUES (@OrderID, @MenuHistID, @Quantity)
    END
END
```

15. AddParameters - ustawia nowe parametry WZ, WK

```
create procedure [dbo].[AddParameters] (
    @wz money,
    @wk int
) as
begin
    begin try
        update Parameters set WZ=@wz;
        update Parameters set WK=@wk;
    end try
    begin catch
        print 'nie udalo sie ustawic parametrow'
    end catch
end;
GO
```

16. AddProduct - dodaje nowy produkt

```
create procedure [dbo].[AddProduct](
    @prodname varchar(50),
    @categoryname varchar(50)
    ) as
begin
    begin try
        declare @temp int;
        set @temp=(select CategoryID from Category where CategoryName=@categoryname);
        insert into Products (ProductName,CategoryID) values (@prodname,@temp);
    end try
    begin catch
        print 'Podana kategoria nie istnieje';
    end catch;
end;
GO
```

17. AddTable - dodaje nowy stolik

```
create procedure [dbo].[AddTable](
    @capacity int
) as
begin
    insert into Tables(Capacity) values (@capacity);
end
GO
```

18. IndividualMonthOrders - pokazuje wszystkie indywidualne zamówienia z ostatniego miesiąca, jesli podamy dodatkowo imie i nazwisko to pokaze zamównienia tylko dla tego klienta

```
CREATE procedure [dbo].[IndividualMonthOrders] (
    @ClientID INT
) as
begin
    if @ClientID is null
        begin
            select * from IndividualOrdersView
            where ( month( getdate() ) > 1 AND month( getdate() ) = month( orderdate ) + 1 AND year( getdate() ) = year( orderdate ) ) OR
            ( month( getdate() ) = 1 AND month( orderdate ) = 12 AND year( getdate() ) = year( orderdate) + 1 )
        end
    else
        begin
            select * from IndividualOrdersView
            where ( month( getdate() ) > 1 AND month( getdate() ) = month( orderdate ) + 1 AND year( getdate() ) = year( orderdate ) ) OR
            ( month( getdate() ) = 1 AND month( orderdate ) = 12 AND year( getdate() ) = year( orderdate) + 1 ) and ClientID=@ClientID
        end
end
```

19. IndividualWeekOrders - pokazuje wszystkie indywidualne zamówienia z ostatniego tygodnia, jesli podamy dodatkowo imie i nazwisko to pokaze zamównienia tylko dla tego klienta

```
CREATE procedure [dbo].[IndividualWeekOrders] (
    @ClientID INT
) as
begin
    if @ClientID is null
        begin
            select * from IndividualOrdersView
            where datediff(day,orderdate,getdate())<=7
        end
    else
        begin
            select * from IndividualOrdersView
            where datediff(day,orderdate,getdate())<=7
                and ClientID=@ClientID
        end
end
```

20. GetTablesForReservation - zwraca listę z pojemnościami stolików jakie są wymagane dla danej rezerwacji

```
CREATE PROCEDURE [dbo].[GetTablesForReservation]
    @reservationID int
AS
BEGIN
    SELECT * FROM [dbo].ReservationTablesView WHERE
    ReservationID = @reservationID
END
GO
```

21. GrantPermanentDiscount - przyznawanie zniżki stałej - sprawdzane jest czy klient spełnił warunki na zadaną zniżkę i nie posiada już innej zniżki stałej

```
CREATE PROCEDURE GrantPermanentDiscount
    @ClientID INT,
    @DiscountID INT
AS
BEGIN
    DECLARE @OrderCount INT = (SELECT COUNT(*) FROM Orders WHERE ClientID=@ClientID
    AND [dbo].GetOrderValue(OrderID) >= (SELECT K1 FROM PermanentDiscount WHERE DiscountID=@DiscountID)
    AND OrderDate > (SELECT SinceDate FROM PermanentDiscount WHERE DiscountID=@DiscountID))

    IF (@OrderCount >= (SELECT Z1 FROM PermanentDiscount WHERE DiscountID=@DiscountID)
    AND @ClientID NOT IN (SELECT ClientID FROM PermanentDiscountsAssigned) AND
    NOT EXISTS(SELECT * FROM PermanentDiscount WHERE SinceDate >
    (SELECT SinceDate FROM PermanentDiscount WHERE DiscountID=@DiscountID)) )
    BEGIN
        INSERT INTO PermanentDiscountsAssigned(ClientID, PermanentDiscountID, DateAssigned)
        VALUES (@ClientID, @DiscountID, GETDATE())
    END
END
GO
```

22. GrantTemporaryDiscount - przyznanie zniżki tymczasowej - sprawdzenie czy klient spełnił warunki na daną zniżkę

```
CREATE PROCEDURE [dbo].[GrantTemporaryDiscount]
    @ClientID INT,
    @DiscountID INT
AS
BEGIN
    DECLARE @TotalValue MONEY = (SELECT SUM([dbo].GetOrderValue(OrderID)) FROM Orders
    WHERE ClientID=@ClientID AND OrderDate > (SELECT SinceDate FROM TemporaryDiscounts
    WHERE DiscountID=@DiscountID))

    IF (@TotalValue > (SELECT K2 FROM TemporaryDiscount WHERE DiscountID = @DiscountID))
    BEGIN
        INSERT INTO TemporaryDiscountsAssigned(ClientID, DiscountID, DateAssigned)
        VALUES (@ClientID, @DiscountID, GETDATE())
    END
END
GO
```

23. SubmitIndividualReservation - potwierdza rezerwację klienta indywidualnego - sprawdza czy spełnione są warunki (WK, WZ), jeśli nie są spełnione - usuwa z Orders dane zamówienie i powiązane z danym zamówieniem wpisy w Order Details. Jeśli warunki są spełnione - zostaje dodany odpowiedni wpis w tabeli IndividualReservations i IndividualReservationDetails.

```sql
CREATE PROCEDURE [dbo].[SubmitIndividualReservation]
    @OrderID INT,
    @IsPaid BIT,
    @TableCapacity INT=NULL
AS
BEGIN
    IF ([dbo].GetOrderValue(@OrderID) > (SELECT WZ FROM Parameters) AND
        [dbo].GetClientFinishedOrdersCount((SELECT ClientID FROM Orders WHERE OrderID=@OrderID)) >
        (SELECT WK FROM Parameters))
    BEGIN
        INSERT INTO Reservations(OrderID) VALUES (@OrderID)

        DECLARE @ReservationID INT
        SET @ReservationID = (SELECT ReservationID FROM Reservations WHERE OrderID=@OrderID)
        DECLARE @ClientID INT
        SET @ClientID = (SELECT ClientID FROM Orders WHERE OrderID = @OrderID)
        DECLARE @Date DATE
        SET @Date = (SELECT OrderDate FROM Orders WHERE OrderID=@OrderID)

        INSERT INTO IndividualReservation(ReservationID, ClientID, IsPaid, ReservationDate, Date)
        VALUES (@ReservationID, @ClientID, @IsPaid, GETDATE(), @Date)

        DECLARE @Takeaway BIT
        SET @Takeaway = (SELECT Takeaway FROM Orders WHERE OrderID=@OrderID)
        IF (@Takeaway = 0)
        BEGIN
            INSERT INTO IndividualReservationDetails(ReservationID, TableCapacity)
            VALUES (@ReservationID, @TableCapacity)
        END
    END
    ELSE
    BEGIN
        DELETE FROM [Order Details] WHERE OrderID = @OrderID
        DELETE FROM Orders WHERE OrderID = @OrderID
    END
END
```

24. ConfirmReservation - zmienia status rezerwacji na 1 (potwierdzona) - używa pracownik

```
CREATE PROCEDURE ConfirmReservation
    @ReservationID INT
AS
BEGIN
    IF @ReservationID IN (SELECT ReservationID FROM IndividualReservation)
    BEGIN
        UPDATE IndividualReservation
        SET status=1
        WHERE ReservationID=@ReservationID
    END
    ELSE IF @ReservationID IN (SELECT ReservationID FROM CompanyReservations)
    BEGIN
        UPDATE CompanyReservations
        SET status=1
        WHERE ReservationID=@ReservationID
    END
    ELSE IF @ReservationID IN (SELECT ReservationID FROM PersonalCompanyReservations)
    BEGIN
        UPDATE PersonalCompanyReservations
        SET status=1
        WHERE ReservationID=@ReservationID
    END
END
GO
```

Funkcje

1. FindClientCompany - znajduje klienta firmowego i zwraca jego ClientID, jeśli nie istnieje, dodaje nowego klienta do bazy i zwraca jego ClientID

```
create function [dbo].[FindClientCompany] (
    @companyname varchar(50),
    @adres varchar(50),
    @phone varchar(50)
) returns int as
begin
    declare @clientid int;
    set @clientid=(select ClientID from ClientCompany where CompanyName=@companyname)
    if (@clientid is null)
        begin
            exec AddClientCompany @companyname,@adres,@phone
            set @clientid=@@identity
        end
    return @clientid
end
GO
```

2. FindClientPerson - znajduje klienta indywidualnego i zwraca jego ClientID, jeśli nie istnieje, dodaje nowego klienta do bazy i zwraca jego ClientID

```
create function [dbo].[FindClientPerson] (
    @imie varchar(50),
    @nazwisko varchar(50),
    @adres varchar(50),
    @companyID int=null
) returns int as
begin
    declare @clientid int;
    set @clientid=(select ClientID from ClientPerson where Firstname=@imie and Lastname=@nazwisko)
    if (@clientid is null)
        begin
            exec AddClientPerson @imie,@nazwisko,@adres,@companyID
            set @clientid=@@identity
        end
    return @clientid
end
GO
```

3. AmountOfIndividualOrders - zwraca ilość zamówień jaką złożył dany klient lub null gdy dany klient nie istnieje

```
CREATE function [dbo].[AmountOfIndividualOrders] (
    @clientid int
) returns int as
begin
    declare @count int
    set @count =(select count(*) from orders where clientid=@clientid)
    return @count
end

GO
```

4. GetOrderValue - zwraca wartość zamówienia o podanym id

```
create function OrderValue(
    @orderid int
) returns money as
begin
    declare @value money
    set @value =(select sum(UnitPrice*Quantity*(1-DiscountValue))
        from orders left join [Order Details] on orders.OrderID=[Order Details].OrderID
            left join MenuHist on [Order Details].MenuHistID=MenuHist.MenuHistID
        where orders.OrderID=@orderid)
    return @value
end
```

5. IsTableFree (przyjmuje ID stolika i datę jako argumenty) - zwraca 0 jeśli stolik jest już zarezerwowany na dany dzień, lub 1 jeśli jest wolny

```
CREATE FUNCTION IsTableFree
(
    @TableID INT,
    @Date DATE
)
RETURNS BIT
AS
BEGIN
    IF (@TableID NOT IN (SELECT TableID FROM TablesAssigned INNER JOIN
        Reservations ON Reservations.ReservationID = TablesAssigned.ReservationID INNER JOIN
        (SELECT ReservationID, Date FROM IndividualReservation UNION
         SELECT ReservationID, Date FROM PersonalCompanyReservations UNION
         SELECT ReservationID, Date FROM CompanyReservations) AS AllReservations ON
         AllReservations.ReservationID = Reservations.ReservationID
        WHERE AllReservations.Date = @Date))
    BEGIN
        RETURN 1
    END

    RETURN 0
END
GO
```

Triggery

1. change_status - po przypisaniu stolików do rezerwacji, zmienia status rezerwacji

```
create trigger [dbo].[change_status] on [dbo].[TablesAssigned]
after insert
as
begin
    declare @ReservationID int=(select ReservationID from inserted)
    declare @TablesAssigned int=(select count(*) from TablesAssigned where ReservationID=@ReservationID)
    declare @tables int
    if exists(select * from CompanyReservations)
        begin
            set @tables=(select count(*) from CompanyReservationDetails)
            if @tables=@TablesAssigned
                begin
                    update CompanyReservations set status=1 where ReservationID=@ReservationID
                end
        end
    if exists(select * from PersonalCompanyReservations)
        begin
            set @tables=(select count(*) from PersonalCompanyReservationDetails)
            if @tables=@TablesAssigned
                begin
                    update PersonalCompanyReservations set status=1 where ReservationID=@ReservationID
                end
        end
    if exists(select * from IndividualReservation)
        begin
            set @tables=(select count(*) from IndividualReservationDetails)
            if @tables=@TablesAssigned
                begin
                    update IndividualReservation set status=1 where ReservationID=@ReservationID
                end
        end
end
GO
```

Indeksy

- w MenuHist na ProductID
- Orders na ClientID
- Reservations na OrderID
- CompanyReservations na CompanyID
- PersonalCompanyReservations na CompanyID
- IndividualReservation na ClientID

UPRAWNIENIA

1. klient indywidualny
    a. możliwość odczytu aktualnego menu
    b. złożenie zamówienia i dodanie do niego pozycji
    c. rezerwacja indywidualna
    d. dodanie siebie jako klient

```
CREATE ROLE [IndividualClient]
GO
grant select on MenuView to CompanyClients
grant execute on AddClientPerson to CompanyClients
grant execute on AddOrderDetails to CompanyClients
grant execute on AddIndividualReservation to CompanyClients
grant execute on AddOrder to CompanyClients
grant execute on SubmitIndividualReservation to CompanyClients
```

2. klient firmowy
    a. dostęp do aktualnego menu
    b. złożenie zamówienia i dodanie do niego pozycji
    c. rezerwacja firmowa
    d. rezerwacja pracowników
    e. dodanie siebie jako klient
    f. dodać pracowników jako klientów

```
CREATE ROLE [CompanyClients]
GO
grant select on MenuView to CompanyClients
grant execute on AddOrder to CompanyClients
grant execute on AddOrderDetails to CompanyClients
grant execute on AddCompanyReservation to CompanyClients
grant execute on AddCompanyReservationDetails to CompanyClients
grant execute on AddClientCompany to CompanyClients
grant execute on AddClientPerson to CompanyClients
grant execute on CompanyMonthReport to CompanyClients
grant execute on CompanyWeekReport to CompanyClients
```

3. employee
    a. dostęp do menu, zamówień, stolików
    b. co więcej? Bo to chyba nie wszystko, a nie wiem
    c. potwierdzanie stolików

```
CREATE ROLE Employee

GRANT SELECT ON IndividualOrdersView TO Employee
GRANT SELECT ON IndividualOrdersValuesView TO Employee
GRANT SELECT ON CompanyOrdersSumView TO Employee
GRANT SELECT ON MenuView TO Employee
GRANT SELECT ON CompanyClientReport TO Employee
GRANT SELECT ON MenuHistMonthView TO Employee
GRANT SELECT ON MenuHistWeekView TO Employee
GRANT SELECT ON TableUsageView TO Employee
GRANT SELECT ON TableUsageWeekView TO Employee
GRANT SELECT ON TemporaryDiscountsReportMonthView TO Employee
GRANT SELECT ON TemporaryDiscountsReportWeekView TO Employee
GRANT SELECT ON NewPermanentDiscountsMonthView TO Employee
GRANT SELECT ON NewPermanentDiscountsWeekView TO Employee
GRANT SELECT ON CompanyOrdersView TO Employee
GRANT SELECT ON AllTemporaryDiscountsView TO Employee
GRANT SELECT ON AllPermanentDiscountsView TO Employee
GRANT SELECT ON UnconfirmedTableReservations TO Employee
GRANT SELECT ON CompanyOrderPricesView TO Employee
GRANT SELECT ON CompanyReservationTablesView TO Employee
GRANT SELECT ON IndividualReservationTablesView TO Employee
GRANT SELECT ON PersonalCompanyReservationTablesView TO Employee
GRANT SELECT ON ReservationTablesView TO Employee
GRANT SELECT ON IndividualClientsOrderValuesView TO Employee
GRANT SELECT ON CompanyClientsOrderValuesView TO Employee
GRANT SELECT ON MenuHistView TO Employee
GRANT SELECT ON AssignedTablesDates TO Employee
```

```
GRANT EXECUTE ON AssignEmployee TO Employee
GRANT EXECUTE ON AssignTable TO Employee
GRANT EXECUTE ON CompanyMonthReport TO Employee
GRANT EXECUTE ON CompanyWeekReport TO Employee
GRANT EXECUTE ON GetTablesForReservation TO Employee
GRANT EXECUTE ON GrantPermanentDiscount TO Employee
GRANT EXECUTE ON GrantTemporaryDiscount TO Employee
GRANT EXECUTE ON IndividualMonthOrders TO Employee
GRANT EXECUTE ON IndividualWeekOrders TO Employee

GRANT EXECUTE ON AddOrder TO Employee
GRANT EXECUTE ON AddOrderDetails TO Employee
GRANT EXECUTE ON AddClientCompany TO Employee
GRANT EXECUTE ON AddClientPerson TO Employee
GRANT SELECT ON MenuView TO Employee
```

4. Owner
    a. Pełne uprawnienia do widoków, procedur, funkcji i bazy danych