Projekt systemu bazodanowego

Jakub Łubkowski, Paweł Jaśkowiec

Spis treści:

Opis:

Użytkownicy:

- Administrator systemu
- Kierownik restauracji
- Pracownik restauracji
- Firma
- Pracownik firmy
- Klient indywidualny
- Dostawca

Funkcje realizowane przez system dla poszczególnych użytkowników:

Administrator

- **dostęp do danych użytkowników - brak możliwości usunięcia,**
- **tworzenie/edycja/usuwanie kont kierownika/pracownika/firmy i jej pracowników/klienta/dostawcy,**
    - wprowadzanie danych do systemu,
    - edytowanie danych w systemie,

Kierownik restauracji

- **edytowanie menu,**
    - usuwanie ręcznie pozycji z menu,
    - dodawanie ręcznie pozycji do menu,
- **generowanie raportów,**
    - wybieranie jakich danych ma dotyczyć raport,
    - wybieranie okresu z jakiego generowany jest raport,

- **akceptowanie rezerwacji,**
    - weryfikacja poprawności rezerwacji,
    - potwierdzenie rezerwacji,
- **wystawianie rachunków,**
    - generowanie faktury na zamówienie,
    - generowanie paragonów,

Pracownik restauracji

- **akceptowanie rezerwacji,**
    - potwierdzenie rezerwacji,
- **generowanie faktury na zamówienie,**
- **wystawianie rachunków,**
    - generowanie faktury na zamówienie,
    - generowanie paragonów,

Firma

- **rezerwacja stolików na firmę,**
    - wybór liczby miejsc do rezerwacji,
    - ustawienie daty i godziny rezerwacji,
    - edytowanie rezerwacji przed potwierdzeniem,
- **rezerwacja stolików dla pracowników firmy,**
    - wybór liczby miejsc do rezerwacji,
    - ustawienie daty i godziny rezerwacji,
    - wprowadzenie danych osobowych pracowników,
    - edytowanie rezerwacji przed potwierdzeniem,
- **opłacenie rachunku,**

Pracownik firmy

- **rezerwacja stolika jako pracownik firmy,**
    - wybór liczby miejsc do rezerwacji,
    - ustawienie daty i godziny rezerwacji,
    - edytowanie rezerwacji przed potwierdzeniem,
- **rezygnowanie z dokonanych rezerwacji,**

Klient indywidualny

- **rezerwacja stolika jako klient indywidualny,**
    - wybór liczby miejsc do rezerwacji,
    - ustawienie daty i godziny rezerwacji,
    - złożenie zamówienia,
    - edytowanie rezerwacji przed potwierdzeniem,
- **opłacenie rachunku,**
- **podanie danych osobowych,**
- **edytowanie danych osobowych,**

Funkcje realizowane przez system:

Menu

- **połowa pozycji menu zmienia się co 2 tygodnie,**

Stoliki

- **stoliki posiadają przypisaną liczbę miejsc,**
- **przy rezerwacji przydzielany jest stolik, bądź stoliki o odpowiedniej pojemności,**

Raporty

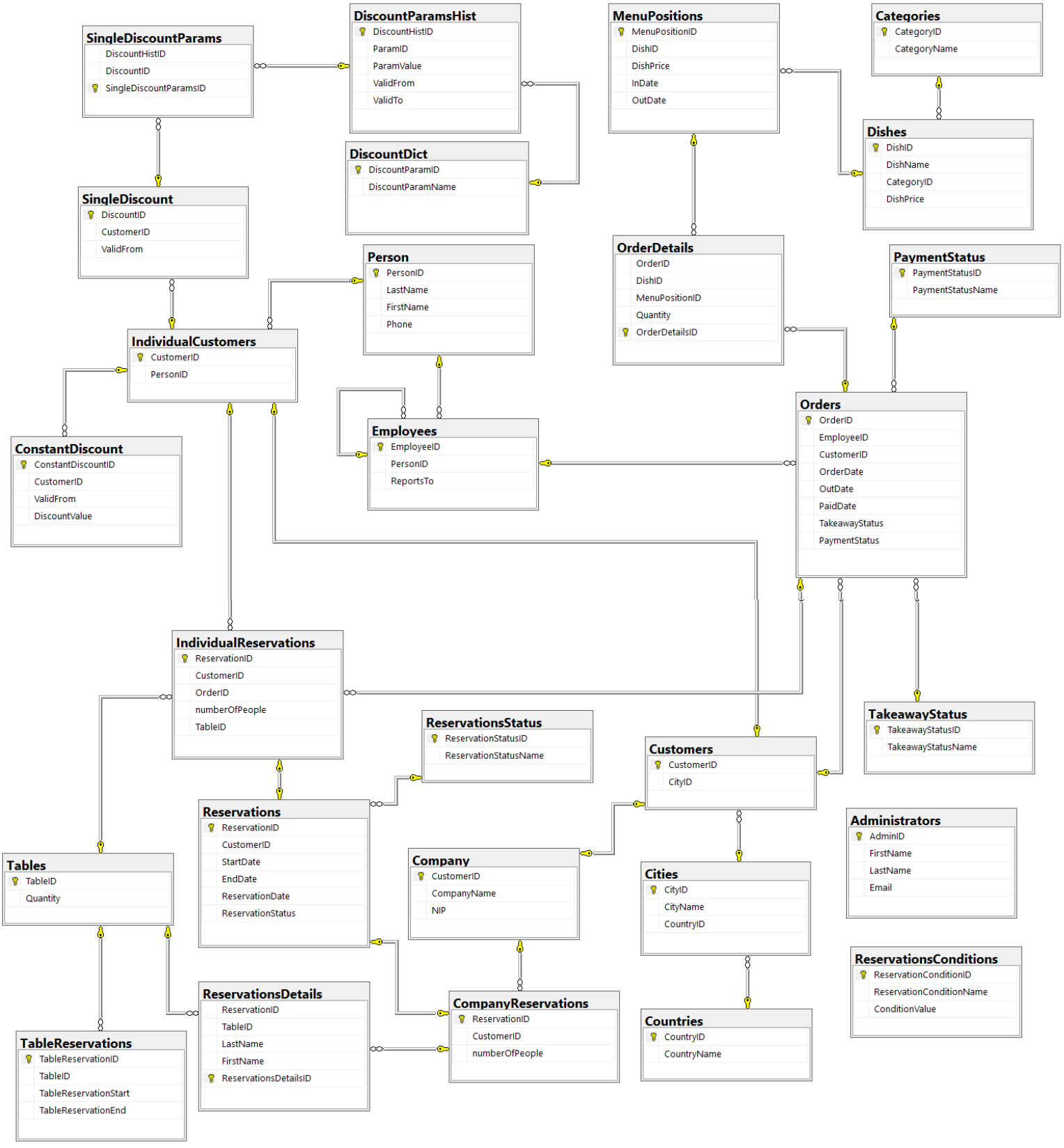
- **system generuje raporty tygodniowe / miesięczne:**
    - dotyczące rezerwacji stolików, rabatów, menu,

    - dla klientów indywidualnych / pracowników firm dotyczących zamówień, rabatów oraz statystyk zamówienia,
    - dla firm dotyczących kwot oraz czasu składania zamówień,

Rabaty

- **system przyznaje rabat w wysokości 5% klientom, którzy wykonali 15 zamówień za kwotę co najmniej 35 zł,**
- **system przyznaje rabat w wysokości 10% na okres 5 dni dla klientów którzy wykonali zamówienia z ostatniego okresu na łączną kwotę min. 1000 zł**

DIAGRAM


SingleDiscountParams
DiscountHistID
DiscountID
SingleDiscountParamsID
DiscountParamsHist
DiscountHistID
ParamID
ParamValue
ValidFrom
ValidTo
DiscountDict
DiscountParamID
DiscountParamName
MenuPositions
MenuPositionID
DishID
DishPrice
InDate
OutDate
Categories
CategoryID
CategoryName
Dishes
DishID
DishName
CategoryID
DishPrice
SingleDiscount
DiscountID
CustomerID
ValidFrom
Person
PersonID
LastName
FirstName
Phone
OrderDetails
OrderID
DishID
MenuPositionID
Quantity
OrderDetailsID
PaymentStatus
PaymentStatusID
PaymentStatusName
IndividualCustomers
CustomerID
PersonID
ConstantDiscount
ConstantDiscountID
CustomerID
ValidFrom
DiscountValue
Employees
EmployeeID
PersonID
ReportsTo
Orders
OrderID
EmployeeID
CustomerID
OrderDate
OutDate
PaidDate
TakeawayStatus
PaymentStatus
IndividualReservations
ReservationID
CustomerID
OrderID
numberOfPeople
TableID
ReservationsStatus
ReservationStatusID
ReservationStatusName
TakeawayStatus
TakeawayStatusID
TakeawayStatusName
Customers
CustomerID
CityID
Reservations
ReservationID
CustomerID
StartDate
EndDate
ReservationDate
ReservationStatus
Administrators
AdminID
FirstName
LastName
Email
Tables
TableID
Quantity
Company
CustomerID
CompanyName
NIP
Cities
CityID
CityName
CountryID
ReservationsConditions
ReservationConditionID
ReservationConditionName
ConditionValue
ReservationsDetails
ReservationID
TableID
LastName
FirstName
ReservationsDetailsID
CompanyReservations
ReservationID
CustomerID
numberOfPeople
Countries
CountryID
CountryName
TableReservations
TableReservationID
TableID
TableReservationStart
TableReservationEnd

Opisy tabel i warunki integralności:

Tabela Countries: reprezentacja krajów w bazie danych

Klucz główny: **CountryID**
nazwa kraju: **CountryName**
Warunki integralności:

- CountryName unikalne

```
   CountryName varchar(50) not null
       unique
```

```
create table Countries
(
   CountryID   int         not null
       constraint Countries_pk
           primary key,
   CountryName varchar(50) not null
       unique
)
go
```

Tabela Cities: reprezentacja miast w bazie danych

Klucz główny: **CityID**
Klucz obcy: **CountryID**
nazwa miasta: **CityName**
Warunki integralności:

- CityName unikalne

```
   CityName  varchar(50) not null
       unique,
```

```
create table Cities
(
   CityID    int         not null
       constraint Cities_pk
           primary key,
   CityName  varchar(50) not null
       unique,
   CountryID int         not null
       constraint Cities_Countries
           references Countries
)
go
```

Tabela Customers: reprezentacja klientów w bazie danych

Klucz główny: **CustomerID**
Klucz obcy: **CityID**

```
create table Customers
(
   CustomerID int not null
       constraint Customers_pk
           primary key,
   CityID     int not null
       constraint Customer_Cities
           references Cities
)
go
```

Tabela Company: reprezentacja firmy w bazie danych zawierająca jej nazwę oraz numer identyfikacji podatkowej

Klucz główny: **CustomerID**
Klucz obcy: **CustomerID**
nazwa firmy: **CustomerName**
numer NIP: **NIP**
Warunki integralności:

- nazwa firmy unikalna,

```
    CompanyName nvarchar(50) not null
        unique,
```

- NIP unikalny, składający się jedynie z cyfr,

```
NIP          nvarchar(50) not null
        unique
        constraint CK_Company_NIP
            check (isnumeric([NIP]) = 1)
```

```
create table Company
(
    CustomerID  int           not null
        constraint Company_pk
            primary key
        constraint Company_Customer
            references Customers,
    CompanyName nvarchar(50) not null
        unique,
    NIP          nvarchar(50) not null
        unique
        constraint CK_Company_NIP
            check (isnumeric([NIP]) = 1)
)
go
```

Tabela IndividualCustomers: reprezentacja klienta indywidualnego w bazie danych

Klucz główny: **CustomerID**
Klucz obcy: **CustomerID, PersonID**

```
create table IndividualCustomers
(
   CustomerID int not null
       constraint IndividualCustomers_pk
           primary key
       constraint IndividualCustomer_Customer
           references Customers,
   PersonID   int not null
       constraint Person_IndividualCustomer
           references Person
)
go
```

Tabela Reservations: reprezentacja rezerwacji złożonych przez klientów

Klucz główny: **ReservationID**
Klucz obcy: **ReservationStatus**
ID klienta składającego rezerwację: **CustomerID**
data złożenia rezerwacji: **ReservationDate**
początek rezerwacji: **StartDate**
koniec rezerwacji: **EndDate**
Warunki integralności:

- **StartDate** musi być datą późniejsza niż data rezerwacji,

```
   constraint CK_Reservations_StartDate
       check ([StartDate] >= [ReservationDate])
```

- **EndDate** musi być datą późniejszą niż **StartDate**, która różni się jedynie godziną,

```
   constraint CK_Reservations_EndDate
       check ([EndDate] > [StartDate] AND datepart(year,
[EndDate]) = datepart(year, [StartDate]) AND
              datepart(month, [EndDate]) =
datepart(month, [StartDate]) AND
              datepart(day, [EndDate]) = datepart(day,
[StartDate])),
```

```
create table Reservations
(
   ReservationID     int      not null
       constraint Reservations_pk
           primary key,
   CustomerID        int      not null,
   StartDate         datetime not null,
   EndDate           datetime not null,
   ReservationDate   datetime not null,
```

```
   ReservationStatus int      not null
       constraint Reservation_ReservationStatus
           references ReservationsStatus,
   constraint CK_Reservations_EndDate
       check ([EndDate] > [StartDate] AND datepart(year,
[EndDate]) = datepart(year, [StartDate]) AND
             datepart(month, [EndDate]) =
datepart(month, [StartDate]) AND
             datepart(day, [EndDate]) = datepart(day,
[StartDate])),
   constraint CK_Reservations_StartDate
       check ([StartDate] >= [ReservationDate])
)
go
```

Tabela CompanyReservations: reprezentacja rezerwacji złożonych przez firmy

Klucz główny: **ReservationID**
Klucz obcy: **CustomerID, ReservationID**
Liczba osób dla danej rezerwacji: **numberOfPeople**
Warunki integralności:

- **numberOfPeople** większe od zera,

```
        constraint CK_CP_nOfPeople
            check ([numberOfPeople] > 0)
```

```
create table CompanyReservations
(
   ReservationID  int not null
       constraint CompanyReservations_pk
           primary key
       constraint Reservation_CompanyReservation
           references Reservations,
   CustomerID     int not null
       constraint CompanyReservation_Company
           references Company,
   numberOfPeople int not null
       constraint CK_CP_nOfPeople
           check ([numberOfPeople] > 0)
)
go
```

Tabela IndividualReservations: reprezentacja rezerwacji złożonych przez klientów indywidualnych

Klucz główny: **ReservationID**
Klucz obcy: **CustomerID, ReservationID, OrderID, TableID**
Ilość osób dla danej rezerwacji: **numberOfPeople**
Warunki integralności:

- **numberOfPeople** większe od zera,

```
        constraint CK_IndiRes_nOfPeople
            check ([numberOfPeople] > 0),
```

```
create table IndividualReservations
(
   ReservationID  int not null
       constraint IndividualReservations_pk
           primary key
       constraint Reservation_IndividualReservation
           references Reservations,
   CustomerID     int not null
       constraint
IndividualReservation_IndividualCustomer
           references IndividualCustomers,
   OrderID        int not null
       constraint IndividualReservation_Orders
           references Orders,
   numberOfPeople int not null
       constraint CK_IndiRes_nOfPeople
           check ([numberOfPeople] > 0),
   TableID        int
       constraint Tables_IndividualReservations
           references Tables
)
go
```

Tabela ReservationsStatus: reprezentuje rodzaje statusu rezerwacji

Klucz główny: **ReservationStatusID**
nazwa statusu: **ReservationStatusName**
Warunki integralności:

- **ReservationStatusName** unikalne,

```
    ReservationStatusName nvarchar(50) not null
        unique
```

```
create table ReservationsStatus
(
    ReservationStatusID   int          not null
        constraint ReservationsStatus_pk
            primary key,
    ReservationStatusName nvarchar(50) not null
        unique
)
go
```

Tabela ReservationsDetails: reprezentuje szczegóły dotyczące jakie osoby siedzą przy stoliku o danym ID dla danej rezerwacji

Klucz główny: **ReservationsDetailsID**
Klucz obcy: **ReservationID**, **TableID**
Nazwisko: **LastName**
Imię: **FirstName**

```
create table ReservationsDetails
(
   ReservationID         int          not null
       constraint
ReservationsDetails_CompanyReservations
           references CompanyReservations,
   TableID               int
       constraint Tables_ReservationsDetails
           references Tables,
   LastName              nvarchar(50) not null,
   FirstName             nvarchar(50) not null,
   ReservationsDetailsID int identity
       constraint ReservationsDetails_pk
           primary key
)
go

create unique index
ReservationsDetails_ReservationsDetailsID_uindex
   on ReservationsDetails (ReservationsDetailsID)
go
```

Tabela Tables: reprezentuje pojemność stołu o danym ID

Klucz główny: **TableID**
pojemność stołu: **Quantity**
Warunki integralności:

- **Quantity** większe od zera,

```
        constraint CK_Tables_Quantity
            check ([Quantity] > 0)
```

```
create table Tables
(
   TableID  int not null
       constraint Tables_pk
           primary key,
   Quantity int not null
       constraint CK_Tables_Quantity
           check ([Quantity] > 0)
)
go
```

Tabela TableReservations: reprezentuje historię rezerwacji stołów oraz przedziały czasu kiedy są zarezerwowane,

o danym ID jest zarezerwowany
Klucz główny: **TableReservationID**
Klucz obcy: **TableID**
początek rezerwacji stołu: **TableReservationStart**
koniec rezerwacji stołu: **TableReservationEnd**
Warunki integralności:

- **TableReservationEnd** musi być datą późniejszą niż **TableReservationStart**, która różni się jedynie godziną,

```
constraint CK_TableRes_TableResEnd
       check ([TableReservationEnd] >
[TableReservationStart] AND
             datepart(year, [TableReservationEnd]) =
datepart(year, [TableReservationStart]) AND
             datepart(month, [TableReservationEnd])
= datepart(month, [TableReservationStart]) AND
             datepart(day, [TableReservationEnd]) =
datepart(day, [TableReservationStart]))
```

```
create table TableReservations
(
   TableReservationID    int      not null
       constraint TableReservations_pk
           primary key,
   TableID               int      not null
       constraint TableReservation_Tables
           references Tables,
   TableReservationStart datetime not null,
   TableReservationEnd   datetime not null,
   constraint CK_TableRes_TableResEnd
       check ([TableReservationEnd] >
```

```
[TableReservationStart] AND
              datepart(year, [TableReservationEnd]) =
datepart(year, [TableReservationStart]) AND
              datepart(month, [TableReservationEnd])
= datepart(month, [TableReservationStart]) AND
              datepart(day, [TableReservationEnd]) =
datepart(day, [TableReservationStart]))
)
go
```

Tabela Person: reprezentuje zbiór wspólnych danych osobowych osób fizycznych

Klucz główny: **PersonID**
nazwisko: **LastName**
imię: **FirstName**
numer telefonu: **Phone**
Warunki integralności:

- **Phone** składający się jedynie z cyfr,

```
        constraint CK_Person_Phone
            check (isnumeric([Phone]) = 1)
```

```
create table Person
(
   PersonID  int          not null
       constraint Person_pk
           primary key,
   LastName  nvarchar(50) not null,
   FirstName nvarchar(50) not null,
   Phone     nvarchar(50) not null
       constraint CK_Person_Phone
           check (isnumeric([Phone]) = 1)
)
go
```

Tabela Employees: reprezentuje zbiór informacji o pracownikach restauracji

Klucz główny: **EmployeeID**
Klucz obcy: **PersonID, ReportsTo**
Warunki integralności:

- jeśli **ReportsTo** nie jest null'em to jest różne od **EmployeeID**,

```
   constraint CK_Employees_ReportsTo
       check ([ReportsTo] <> [EmployeeID] OR
[ReportsTo] IS NULL)
```

```
create table Employees
(
   EmployeeID int not null
       constraint Employees_pk
           primary key,
   PersonID   int not null
       constraint Employees_Person
           references Person,
   ReportsTo  int
       constraint Employees_Employees
           references Employees,
   constraint CK_Employees_ReportsTo
       check ([ReportsTo] <> [EmployeeID] OR
[ReportsTo] IS NULL)
)
go
```

Tabela Administrators: reprezentuje zbiór informacji administratorach systemu,

Klucz główny: **AdminID**
imię: **FirstName**
nazwisko: **LastName**
adres email: **Email**
Warunki integralności:

- **Email** musi zawierać znak “@”,

```
        constraint CK_Administrators_Email
            check ([Email] like '%@%')
```

```
create table Administrators
(
   AdminID   int          not null
       constraint Administrators_pk
           primary key,
   FirstName varchar(50) not null,
   LastName  varchar(50) not null,
   Email     varchar(50) not null
       constraint CK_Administrators_Email
           check ([Email] like '%@%')
)
go
```

Tabela ConstantDiscount: reprezentuje zbiór informacji o stałych zniżkach dla danego klienta indywidualnego

Klucz główny: **ConstantDiscountID**
Klucz obcy: **CustomerID**
Data rozpoczęcia zniżki: **ValidFrom**
Wartość zniżki: **DiscountValue**
Warunki integralności:

- **ValidFrom** musi być datą większa od daty aktualnej (tzn. daty kiedy wprowadzana jest zniżka),

```
        constraint CK_CD_ValidFrom
            check ([ValidFrom] >= getdate()),
```

- **DiscountValue** reprezentuje procent w postaci liczby zmiennoprzecinkowej z max 2 liczbami po przecinku (np: 0.25 - 25%) na przedziale od 0 do 1,

```
        constraint CK_CD_DiscValue
            check ([DiscountValue] > 0 AND
[DiscountValue] < 1)
```

```
create table ConstantDiscount
(
   ConstantDiscountID int      not null
       constraint ConstantDiscount_pk
           primary key,
   CustomerID         int      not null
       constraint IndividualCustomer_ConstantDiscount
           references IndividualCustomers,
   ValidFrom          datetime not null
       constraint CK_CD_ValidFrom
```

```
            check ([ValidFrom] >= getdate()),
    DiscountValue       real      not null
        constraint CK_CD_DiscValue
            check ([DiscountValue] > 0 AND
[DiscountValue] < 1)
)
go
```

Tabela SingleDiscount: reprezentacja okresowych zniżek dla klienta indywidualnego

Klucz główny: **DiscountID**
Klucz obcy: **CustomerID**
data rozpoczęcia zniżki: **ValidFrom**
Warunki integralności:

- **ValidFrom** musi być datą większa od daty aktualnej (tzn. daty kiedy wprowadzana jest zniżka),

```
      constraint CK_SingleDisc_ValidFrom
        check (ValidFrom >= GETDATE())
```

```
create table SingleDiscount
(
  DiscountID int      not null
      constraint SingleDiscount_pk
          primary key,
  CustomerID int      not null
      constraint
SingleDiscount_IndividualCustomer
          references IndividualCustomers,
  ValidFrom  datetime not null,
      constraint CK_SingleDisc_ValidFrom
        check (ValidFrom >= GETDATE())
)
go
```

Tabela SingleDiscountParams: tabela łącząca tabelę **SingleDiscount** z tabelą **DiscountParamsHist**,

Klucz główny: **DiscountID**, **DiscountHistID,**
Klucz obcy: **DiscountID**, **DiscountHistID**,

```
create table SingleDiscountParams
(
  DiscountHistID int not null,
  DiscountID     int not null,
  constraint SingleDiscountParams_pk
      primary key (DiscountHistID, DiscountID)
)
go

alter table SingleDiscountParams add constraint
   DiscountParams_DiscountParamsHist
foreign key (DiscountHistID)
references SingleDiscount (DiscountID);

alter table SingleDiscountParams add constraint
   DiscountParams_SingleDiscount
foreign key (DiscountID)
references SingleDiscount (DiscountID);
```

Tabela DiscountParamsHist: reprezentuje zbiór informacji o historii zniżek okresowych dla klienta indywidualnego

Klucz główny: **DiscountHistID**
Klucz obcy: **ParamID**
Data początku zniżki okresowej: **ValidFrom**
Data końca zniżki okresowej: **ValidTo**
Wartość parametru: **ParamValue**
Warunki integralności:

- **ValidFrom** musi być datą większa od daty aktualnej (tzn. daty kiedy wprowadzana jest zniżka),

```
  constraint CK_DiscPH_ValidFrom
      check ([ValidFrom] >= GETDATE()),
```

- jeśli **ValidTo** zawiera datę to musi zawierać datę późniejszą niż data pola **ValidFrom**,

```
  constraint CK_DiscPH_ValidTo
      check ([ValidTo] IS NULL OR [ValidTo] >
[ValidFrom])
```

- **ParamValue** reprezentuje procent w postaci liczby zmiennoprzecinkowej z max 2 liczbami po przecinku (np: 0.25 - 25%) na przedziale od 0 do 1,

```
    constraint CK_DiscPH_ParamValue
       check ([ParamValue] > 0 AND [ParamValue] < 1),
```

```
create table DiscountParamsHist
(
  DiscountHistID int      not null
      constraint DiscountParamsHist_pk
          primary key,
  ParamID        int      not null
      constraint DiscountDict_DiscountParamsHist
          references DiscountDict,
```

```
  ParamValue      real      not null
      constraint CK_DiscPH_ParamValue
          check ([ParamValue] > 0 AND
[ParamValue] < 1),
  ValidFrom       datetime not null,
  ValidTo         datetime,
  constraint CK_DiscPH_ValidFrom
      check ([ValidFrom] >= GETDATE()),
  constraint CK_DiscPH_ValidTo
      check ([ValidTo] IS NULL OR [ValidTo] >
[ValidFrom])
)
go

alter table DiscountParamsHist
   add constraint
       DiscountDict_DiscountParamsHist
       foreign key (ParamID)
           references DiscountDict
(DiscountParamID);
```

Tabela DiscountDict: reprezentuje zbiór parametrów opisujących zniżki okresowego dla klienta indywidualnego

Klucz główny: **DiscountParamID**
nazwa parametru: **DiscountParamName**

```
create table DiscountDict
(
   DiscountParamID   int          not null,
   DiscountParamName nvarchar(50) not null,
   constraint DiscountDict_pk
       primary key (DiscountParamID)
)
go
```

Tabela Categories: reprezentacja kategorii sprzedawanych pozycji,

Klucz główny: **CategoryID**
Nazwa kategorii: **CategoryName**
Warunki integralności:

- **CategoryName** unikalne,

```
    CategoryName varchar(50) not null unique,
```

```
create table Categories
(
    CategoryID   int          not null,
    CategoryName varchar(50) not null unique,
    constraint Categories_pk
        primary key (CategoryID),
    unique (CategoryName)
)
go
```

Tabela Dishes: reprezentacja sprzedawanych pozycji,

Klucz główny: **DishID**
Klucz obcy: **CategoryID**
Nazwa pozycji: **DishName**
Cena pozycji: **DishPrice**
Warunki integralności:

- **DishPrice** większe od zera,

```
    constraint CK_Dishes_Price
        check ([DishPrice] > 0)
```

- **DishName** unikalne,

```
    DishName    varchar(50) not null unique,
```

```
create table Dishes
(
    DishID      int          not null,
    DishName    varchar(50) not null unique,
    CategoryID int          not null,
    DishPrice  money        not null,
    constraint Dishes_pk
        primary key (DishID),
    unique (DishName),
    constraint Products_Categories
        foreign key (CategoryID) references
Categories (CategoryID)
)
go

alter table Dishes
    add constraint CK_Dishes_Price
        check ([DishPrice] > 0)
go
```

Tabela MenuPositions: reprezentacja sprzedawanych pozycji z datami kiedy znajdują się w naszym menu

Klucz główny: **MenuPositionID**
Klucz obcy: **DishID**
Aktualna cena pozycji w menu: **DishPrice**
Data wstawienia pozycji do menu: **InDate**
Data usunięcia pozycji z menu: **OutDate**
Warunki integralności:

- **OutDate** jeśli nie jest null'em, to musi być datą późniejsza niż **InDate**,

```
   constraint CK_MenuPos_OutDate
       check ([OutDate] IS NULL OR [OutDate] >
[InDate])
```

- **DishPrice** większe od zera,

```
   constraint CK_MenuPos_DPrice
       check ([DishPrice] > 0)
```

```
create table MenuPositions
(
   MenuPositionID int      not null,
   DishID         int      not null,
   DishPrice      money    not null,
   InDate         datetime not null,
   OutDate        datetime,
   constraint MenuPositions_pk
       primary key (MenuPositionID),
   constraint Menu_Products
       foreign key (DishID) references Dishes
)
go

alter table MenuPositions
```

```
   add constraint CK_MenuPos_DPrice
       check ([DishPrice] > 0)
go

alter table MenuPositions
   add constraint CK_MenuPos_OutDate
       check ([OutDate] IS NULL OR [OutDate] >
[InDate])
go
```

Tabela OrderDetails: reprezentacja szczegółów dotyczących danego zamówienia

Klucz główny: **OrderID**
Klucz obcy: **OrderID, MenuPositionID**
ID dania: **DishID**
Ilość zakupionej pozycji: **Quantity**
Naliczona zniżka: **Discount**
Warunki integralności:

- Quantity jest wartością dodatnią

```
    constraint CK_OrderDet_Quantity
        check ([Quantity] > 0)
```

```
create table OrderDetails
(
    OrderID        int not null,
    DishID         int not null,
    MenuPositionID int not null,
    Quantity       int not null,
    OrderDetailsID int identity,
    constraint OrderDetails_pk
        primary key (OrderDetailsID),
    constraint OrderDetails_MenuPosition
        foreign key (MenuPositionID) references
MenuPositions,
    constraint OrderDetails_Orders
        foreign key (OrderID) references Orders
)
go

create unique index
OrderDetails_OrderDetailsID_uindex
    on OrderDetails (OrderDetailsID)
go
```

```
alter table OrderDetails
   add constraint CK_OrderDet_Quantity
       check ([Quantity] > 0)
go
```

Tabela Orders: reprezentacja ogólnych informacji dotyczących danego zamówienia

Klucz główny: **OrderID**
Klucz obcy: **EmployeeID, CustomerID, TakeawayStatus, PaymentStatus**
Data złożenia zamówienia: **OrderDate**
Data realizacji zamówienia: **OutDate**
Data płatności za zamówienie: **PaidDate**

Warunki integralności:

- **OutDate** jeśli nie jest null'em, musi być datą późniejsza niż **OrderDate**,

```
   constraint CK_Orders_OutDate
       check ([OutDate] IS NULL OR [OutDate] >
[OrderDate])
```

- **PaidDate** jeśli nie jest null'em, musi być datą nie wcześniejszą niż **OrderDate**,

```
   constraint CK_Orders_PaidDate
       check ([PaidDate] IS NULL OR [PaidDate] >=
[OrderDate])
```

```
create table Orders
(
   OrderID        int      not null,
   EmployeeID     int      not null,
   CustomerID     int      not null,
   OrderDate      datetime not null,
   OutDate        datetime,
   PaidDate       datetime,
   TakeawayStatus int      not null,
   PaymentStatus  int      not null,
   constraint Orders_pk
       primary key (OrderID),
```

```
   constraint Orders_Customer
       foreign key (CustomerID) references Customers,
   constraint Orders_Employees
       foreign key (EmployeeID) references Employees,
   constraint Orders_PaymentStatus
       foreign key (PaymentStatus) references
PaymentStatus,
   constraint Orders_TakeawayStatus
       foreign key (TakeawayStatus) references
TakeawayStatus
)
go

alter table Orders
   add constraint CK_Orders_OutDate
       check ([OutDate] IS NULL OR [OutDate] >
[OrderDate])
go

alter table Orders
   add constraint CK_Orders_PaidDate
       check ([PaidDate] IS NULL OR [PaidDate] >=
[OrderDate])
go
```

Tabela PaymentStatus: reprezentuje rodzaje statusu płatności

Klucz główny: **PaymentStatusID**
Nazwa statusu: **PaymentStatusName**
Warunki integralności:

- PaymentStatusName musi być unikalne

```
    PaymentStatusName nvarchar(50) not null
unique,
```

```
create table PaymentStatus
(
    PaymentStatusID   int           not null,
    PaymentStatusName nvarchar(50) not null
unique,
    constraint PaymentStatus_pk
        primary key (PaymentStatusID)
)
go
```

Tabela TakeawayStatus: reprezentuje rodzaje statusu zamówienia (na wynos lub stacjonarnie)

Klucz główny: **TakeawayStatusID**
Nazwa statusu: **TakeawayStatusName**
Warunki integralności:

- TakeawayStatusName musi być unikalne

```
TakeawayStatusName nvarchar(50) not null,
   unique (TakeawayStatusName)
```

```
create table TakeawayStatus
(
   TakeawayStatusID   int           not null,
   TakeawayStatusName nvarchar(50) not null,
   constraint TakeawayStatus_pk
       primary key (TakeawayStatusID),
   unique (TakeawayStatusName)
)
go
```

Tabela ReservationsConditions: reprezentuje warunki niezbędne do spełnienia przy składaniu rezerwacji,

Klucz główny: **ReservationConditionID**
Nazwa warunku rezerwacji: **ReservationConditionName**
Wartość warunku rezerwacji: **ConditionValue**
Warunki integralności:

- **ConditionValue** nie mniejsze od zera,

```
    constraint CK_ResConditions_ConditionValue
       check ([ConditionValue] >= 0)
```

- **ReservationConditionName** unikalne,

```
   ReservationConditionName nvarchar(50) not
null,
   unique (ReservationConditionName)
```

```
create table ReservationsConditions
(
   ReservationConditionID   int          not null,
   ReservationConditionName nvarchar(50) not null,
   ConditionValue           int          not null,
   constraint ReservationsConditions_pk
       primary key (ReservationConditionID),
   unique (ReservationConditionName)
)
go

alter table ReservationsConditions
   add constraint CK_ResConditions_ConditionValue
       check ([ConditionValue] >= 0)
go
```

Widoki:

Widok allMenuPositions

Pokazuje wszystkie pozycje które wystąpiły bądź wystąpią w menu wraz z ich datami wejścia i wyjścia z menu

```
CREATE VIEW allMenuPositions AS
SELECT DISTINCT MP.MenuPositionID,
                MP.DishID,
                D.DishName,
                Mp.DishPrice,
                Mp.InDate,
                Mp.OutDate
FROM MenuPositions AS MP
        INNER JOIN Dishes D on D.DishID = MP.DishID
go
```

Widok allDishes

Pokazuje wszystkie dania wraz z nazwą i kategorią

```
CREATE VIEW allDishes AS
SELECT DISTINCT D.DishID,
                D.DishName,
                C.CategoryName
FROM Dishes D
        INNER JOIN Categories C on C.CategoryID =
D.CategoryID
go
```

Widok allReservations

pokazuje wszystkie złożone rezerwacje

```
CREATE VIEW allReservations AS
SELECT R.ReservationID,
      R.CustomerID,
      RS.ReservationStatusName,
      R.StartDate,
      'Individual'       AS Customer,
      Year(R.StartDate)  as Year,
      MONTH(R.StartDate) as Month
FROM Reservations AS R
        INNER JOIN IndividualReservations IR on
R.ReservationID = IR.ReservationID
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
UNION
SELECT R.ReservationID,
      R.CustomerID,
      RS.ReservationStatusName,
      R.StartDate,
      'Company'          AS Customer,
      Year(R.StartDate)  as Year,
      MONTH(R.StartDate) as Month
FROM Reservations AS R
        INNER JOIN CompanyReservations CR on
R.ReservationID = CR.ReservationID
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
go
```

Widok allIndividualReservations

Pokazuje wszystkie rezerwacje klientów indywidualnych

```
CREATE VIEW allIndividualReservations AS
SELECT R.ReservationID,
      R.CustomerID,
      RS.ReservationStatusName,
      R.StartDate,
      R.EndDate,
      YEAR(R.StartDate)  as Year,
      MONTH(R.StartDate) as Month
FROM Reservations AS R
        INNER JOIN IndividualReservations IR on
R.ReservationID = IR.ReservationID
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
go
```

Widok allCompaniesReservations

Pokazuje wszystkie rezerwacje firmowe

```
CREATE VIEW allCompaniesReservations AS
SELECT R.ReservationID,
      R.CustomerID,
      C.CompanyName,
      RS.ReservationStatusName,
      R.StartDate,
      R.EndDate,
      YEAR(R.StartDate)  as Year,
      MONTH(R.StartDate) as Month
FROM Reservations AS R
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
        INNER JOIN CompanyReservations CR on
R.ReservationID = CR.ReservationID
        INNER JOIN Company C on C.CustomerID =
CR.CustomerID
go
```

Widok allReservationsParticipants

Wyświetla wszystkich uczestników danych rezerwacji firmowych

```
CREATE VIEW allReservationsParticipants AS
SELECT CR.ReservationID,
      RD.TableID,
      RD.LastName + ' ' + RD.FirstName AS
[Participant name]
FROM CompanyReservations AS CR
        INNER JOIN ReservationsDetails RD on
CR.ReservationID = RD.ReservationID
        INNER JOIN Reservations R on R.ReservationID
= CR.ReservationID
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
WHERE RS.ReservationStatusName = 'confirmed'
  OR RS.ReservationStatusName = 'completed'
go
```

Widok allTableStatistics

Wyświetla statystyki dotyczące rezerwacji danych stolików (pojemność stolika i jak często był wybierany przy rezerwacjach),

```
create view allTableStatistics as
select T.TableID,
      T.Quantity,
      (select count(*)
       from TableReservations TR
       where T.TableID = TR.TableID) as Used
from Tables T
group by T.TableID, T.Quantity
go
```

Widok allUnpaidOrders

Wyświetla zamówienia które nie zostały opłacone

```
CREATE VIEW allUnpaidOrders AS
SELECT O.OrderID,
      O.OrderDate,
      O.CustomerID
FROM Orders AS O
        INNER JOIN PaymentStatus PS on
PS.PaymentStatusID = O.PaymentStatus
WHERE PS.PaymentStatusName = 'unpaid'
go
```

Widok allWaitingReservations

Wyświetla wszystkie rezerwacje oczekujące na potwierdzenie

```
CREATE VIEW allWaitingReservations AS
SELECT R.ReservationID,
      R.CustomerID,
      R.ReservationDate,
      R.StartDate,
      R.EndDate
FROM Reservations AS R
        INNER JOIN ReservationsStatus RS on
RS.ReservationStatusID = R.ReservationStatus
WHERE ReservationStatusName = 'pending'
go
```

Widok **AverageOrderPricesForCustomers**

Wyświetla średnią wartość zamówień klientów wraz z informacją czy jest to klient indywidualny lub firma

```
CREATE VIEW AverageOrderPricesForCustomers AS
Select COS.CustomerID,
      P.LastName + ' ' + P.FirstName as Customer,
      Round(AVG(COS.sum), 2)         as AVG,
      'Person'                       as
[Company/Person]
FROM CustomersOrdersSum COS
        INNER JOIN IndividualCustomers IC on
COS.CustomerID = IC.CustomerID
        INNER JOIN Person P on P.PersonID =
IC.PersonID
GROUP BY COS.CustomerID, P.LastName, P.FirstName
UNION
Select COS.CustomerID,
      C.CompanyName           as Customer,
      Round(AVG(COS.sum), 2) as AVG,
      'Company'               as [Company/Person]
FROM CustomersOrdersSum COS
        INNER JOIN Company C on COS.CustomerID =
C.CustomerID
GROUP BY COS.CustomerID, C.CompanyName
go
```

Widok **CustomersNumbersOfOrders**

Wyświetla ilość zamówień danego klienta

```
CREATE VIEW CustomersNumbersOfOrders AS
select C.CustomerID,
      P.LastName + ' ' + P.FirstName as Customer,
      count(O.OrderID)               as [Number of Orders],
      'Person'                       as [Company/Person]
FROM Customers C
        INNER JOIN Orders O on C.CustomerID = O.CustomerID
        INNER JOIN IndividualCustomers IC on C.CustomerID =
IC.CustomerID
        INNER JOIN Person P on P.PersonID = IC.PersonID
GROUP BY C.CustomerID, P.LastName + ' ' + P.FirstName
UNION
SELECT C.CustomerID,
      C2.CompanyName   as Customer,
      count(O.OrderID) as [Number of Orders],
      'Company'        as [Company/Person]
FROM Customers C
        INNER JOIN Orders O on C.CustomerID = O.CustomerID
        INNER JOIN Company C2 on C.CustomerID = C2.CustomerID
GROUP BY C.CustomerID, C2.CompanyName
go
```

Widok CustomersOrdersSum

Wyświetla wartość zamówień

```
CREATE VIEW CustomersOrdersSum AS
select C.CustomerID,
      O.OrderID,
      ROUND(SUM(MP.DishPrice * OD.Quantity * (1 -
isnull(dbo.udfGetDiscount(O.OrderID, C.CustomerID), 0))), 2)
as sum
FROM Customers C
        INNER JOIN IndividualCustomers IC on C.CustomerID =
IC.CustomerID
        INNER JOIN Person P on P.PersonID = IC.PersonID
        INNER JOIN Orders O on C.CustomerID = O.CustomerID
        INNER JOIN OrderDetails OD on O.OrderID = OD.OrderID
        INNER JOIN MenuPositions MP on MP.MenuPositionID =
OD.MenuPositionID
GROUP BY C.CustomerID, O.OrderID
UNION
select C.CustomerID,
      O.OrderID,
      ROUND(SUM(MP.DishPrice * OD.Quantity), 2) as sum
FROM Customers C
        INNER JOIN Company C2 on C.CustomerID = C2.CustomerID
        INNER JOIN Orders O on C.CustomerID = O.CustomerID
        INNER JOIN OrderDetails OD on O.OrderID = OD.OrderID
        INNER JOIN MenuPositions MP on MP.MenuPositionID =
OD.MenuPositionID
GROUP BY C.CustomerID, O.OrderID
go
```

Widok CustomersOrders

Wyświetla wartość zamówień wraz z imieniem i nazwiskiem klienta indywidualnego bądź nazwą firmy, datą zamówienia, rokiem oraz miesiącem

```
CREATE VIEW CustomersOrders AS
SELECT O.OrderID,
      IC.CustomerID,
      P.LastName + ' ' + P.FirstName AS [Customer],
      ROUND(SUM(MP.DishPrice * OD.Quantity * (1 -
isnull(dbo.udfGetDiscount(O.OrderID, C.CustomerID), 0))),
            2)                          AS [Order price],
      O.OrderDate,
      'Person'                          as [Company/Person],
      YEAR(O.OrderDate)                 as Year,
      MONTH(O.OrderDate)                as Month
FROM Orders AS O
        INNER JOIN OrderDetails OD on O.OrderID =
OD.OrderID
        INNER JOIN MenuPositions MP on MP.MenuPositionID
= OD.MenuPositionID
        INNER JOIN Customers C on C.CustomerID =
O.CustomerID
        INNER JOIN IndividualCustomers IC on C.CustomerID
= IC.CustomerID
        INNER JOIN Person P on P.PersonID = IC.PersonID
GROUP BY O.OrderID, IC.CustomerID, P.LastName + ' ' +
P.FirstName, O.OrderDate
UNION
SELECT O.OrderID,
      C.CustomerID,
      C2.CompanyName                            AS
```

```
[Customer],
      ROUND(SUM(MP.DishPrice * OD.Quantity), 2) AS [Order
price],
      O.OrderDate,
      'Company'                                  as
[Company/Person],
      YEAR(O.OrderDate)                          as Year,
      MONTH(O.OrderDate)                         as Month
FROM Orders AS O
        INNER JOIN OrderDetails OD on O.OrderID =
OD.OrderID
        INNER JOIN MenuPositions MP on MP.MenuPositionID
= OD.MenuPositionID
        INNER JOIN Customers C on C.CustomerID =
O.CustomerID
        INNER JOIN Company C2 on C.CustomerID =
C2.CustomerID
GROUP BY O.OrderID, C.CustomerID, C2.CompanyName,
O.OrderDate
```

Widok CustomersStatistics

Wyświetla statystyki danego klienta indywidualnego/firmy takie jak ilość złożonych zamówień bądź średnia wartość zamówienia

```
CREATE VIEW CustomersStatistics AS
SELECT C.CustomerID,
      cNOO.Customer,
      cNoo.[Company/Person],
      cNOO.[Number of Orders],
      aOPFC.AVG as [Average Order Price]
FROM Customers C
        INNER JOIN averageOrderPricesForCustomers aOPFC
on C.CustomerID = aOPFC.CustomerID
        INNER JOIN customersNumbersOfOrders cNOO on
C.CustomerID = cNOO.CustomerID
go
```

Widok currentMenu

Wyświetla dania z aktualnego menu

```
CREATE VIEW currentMenu AS
SELECT MP.DishID,
       D.DishName,
       MP.DishPrice,
       MP.InDate,
       C.CategoryName
FROM MenuPositions AS MP
         INNER JOIN Dishes D on D.DishID = MP.DishID
         INNER JOIN Categories C on C.CategoryID =
D.CategoryID
WHERE MP.OutDate IS NULL
  OR MP.OutDate > GETDATE()
go
```

Widok DishesInMenuQuant

Wyświetla ile razy dane danie wystąpiło w menu

```
CREATE VIEW DishesInMenuQuant as
SELECT D.DishName,
      (select count(*)
       from Dishes D2
                INNER JOIN MenuPositions MP on
D2.DishID = MP.DishID
       where D.DishID = D2.DishID) as DishInMenu
from Dishes D
go
```

Widok individualCustomersDiscounts

Wyświetla zniżki klientów indywidualnych

```
CREATE VIEW individualCustomersDiscounts AS
SELECT SD.CustomerID,
      P.LastName + ' ' + P.FirstName AS [Customer Name],
      DD.DiscountParamName,
      DPH.ParamValue,
      DPH.ValidFrom
FROM DiscountParamsHist AS DPH
        INNER JOIN DiscountDict DD on DD.DiscountParamID
= DPH.ParamID
        INNER JOIN SingleDiscountParams SDP on
DPH.DiscountHistID = SDP.DiscountHistID
        INNER JOIN SingleDiscount SD on SD.DiscountID =
SDP.DiscountID
        INNER JOIN IndividualCustomers IC on
IC.CustomerID = SD.CustomerID
        INNER JOIN Person P on P.PersonID = IC.PersonID
UNION
SELECT I.CustomerID,
      P2.LastName + ' ' + P2.FirstName AS [Customer
Name],
      'Constant Discount',
      CD.DiscountValue,
      CD.ValidFrom
FROM ConstantDiscount AS CD
        INNER JOIN IndividualCustomers I on I.CustomerID
= CD.CustomerID
        INNER JOIN Person P2 on P2.PersonID = I.PersonID
go
```

Widok QuantityOfOrderedDishes

Wyświetla ilość zamówień danego dania

```
CREATE VIEW QuantityOfOrderedDishes as
SELECT D.DishName,
      (select count(*)
       from Dishes D2
                INNER JOIN MenuPositions MP on
D2.DishID = MP.DishID
                INNER JOIN OrderDetails OD on
MP.MenuPositionID = OD.MenuPositionID
       where D.DishID = D2.DishID) as DishQuant
from Dishes D
go
```

Widok whereCustomersAreFrom

Wyświetla informację ilu klientów pochodzi z danego kraju

```
create view whereCustomersAreFrom as
select C.CountryName,
      (select count(*)
       from Countries CS
                inner join Cities C2 on CS.CountryID
= C2.CountryID
                inner join Customers C3 on C2.CityID
= C3.CityID
       where CS.CountryID = C.CountryID
      ) as CustomersQuantity
from Countries C
go
```

Widok ordersInvoice

Wyświetla informacje potrzebne do faktur

(OrderID, CustomerID, MenuPositionID, DishPrice, Quantity, Discount, Total Price, OrderDate, Year, Month)

```
CREATE view ordersInvoice as
select OD.OrderID,
      O.CustomerID,
      OD.MenuPositionID,
      MP.DishPrice,
      OD.Quantity,
      ROUND(dbo.udfGetDiscount(O.OrderID,
O.CustomerID), 2) as discount,
      CO.[Order price],
      O.OrderDate,
      year(O.OrderDate)
as Year,
      Month(O.OrderDate)
as Month
from orders as O
        INNER JOIN OrderDetails OD on O.OrderID =
OD.OrderID
        INNER JOIN CustomersOrders CO on O.OrderID =
CO.OrderID and OD.OrderID = CO.OrderID
        INNER JOIN MenuPositions MP on
MP.MenuPositionID = OD.MenuPositionID
go
```

Funkcje:

Funkcja udfGetBestSellingDishes

Zwraca top x najlepiej sprzedających się dań

```
create function udfGetBestSellingDishes(@x int)
   RETURNS table AS RETURN
   select TOP (@x) * from QuantityOfOrderedDishes
   order by DishQuant desc
go
```

Funkcja udfGetCompaniesStatistics

Zwraca statystyki firm dotyczące ilości złożonych zamówień oraz średniej wartości zamówienia

```
CREATE FUNCTION udfGetCompaniesStatistics()
  RETURNS table AS
      RETURN
      SELECT CustomerID, Customer, [Company/Person],
[Number of Orders], [Average Order Price]
      FROM CustomersStatistics CS
      WHERE CS.[Company/Person] = 'Company'
go
```

Funkcja udfGetIndividualCustomerStatistics

Zwraca statystyki klientów indywidualnych dotyczące ilości złożonych zamówień oraz średniej wartości zamówienia

```
CREATE FUNCTION udfGetIndividualCustomerStatistics()
     RETURNS table AS
      RETURN
      SELECT CustomerID, Customer, [Company/Person],
[Number of Orders], [Average Order Price]
      FROM CustomersStatistics CS
      WHERE CS.[Company/Person] = 'Person'
go
```

Funkcja udfGetCustomerStatisticsById

Zwraca statystyki klienta o podanym numerze id dotyczące ilości złożonych zamówień oraz średniej wartości zamówienia

```
CREATE FUNCTION udfGetCustomerStatisticsById(@id int)
   RETURNS table AS
       RETURN
       select customerid, customer, [company/person],
[number of orders], [average order price]
        from CustomersStatistics CS
        where CS.CustomerID = @id
go
```

Funkcja udfGetDiscount

Zwraca aktualną w danym zamówieniu (order_id) największą zniżkę klienta (customer_id)

```
CREATE FUNCTION udfGetDiscount(@order_id int, @customer_id int)
RETURNS FLOAT AS
  BEGIN
      DECLARE @val1 float;
      DECLARE @val2 float;
      DECLARE @maxval float;
      SET @val1 = ISNULL((
          select DiscountValue
          from ConstantDiscount CD
          INNER JOIN Orders O on @order_id = O.OrderID
          where (CD.CustomerID = @customer_id and O.OrderDate >=
CD.ValidFrom)
          ),0)
      SET @val2 = ISNULL((
          select DPH.ParamValue
          from SingleDiscount SD
          INNER JOIN SingleDiscountParams SDP on SD.DiscountID =
SDP.DiscountID
          INNER JOIN DiscountParamsHist DPH on SDP.DiscountHistID
= DPH.DiscountHistID
          INNER JOIN Orders O on @order_id = O.OrderID
          where (SD.CustomerID = @customer_id and O.OrderDate >=
DPH.ValidFrom and O.OrderDate <= ISNULL((DPH.ValidTo),getdate()))
          ),0)
      if(@val1 >= @val2)
          set @maxval = @val1
      else
          set @maxval = @val2

      RETURN @maxval
  END
go
```

Funkcja **udfGetDiscountsFrom**

Zwraca zniżki klientów z okresu `input` dni do tyłu

```
CREATE FUNCTION udfGetDiscountsFrom(@input int)
  RETURNS table AS
      RETURN
      SELECT customerid, [customer name],
discountparamname, paramvalue, validfrom
      FROM individualCustomersDiscounts ICD
      WHERE ValidFrom >= DATEADD(day,-@input,
GETDATE())
go
```

Funkcja udfGetDishesByCategory

Zwraca dania w danej kategorii

```
CREATE FUNCTION udfGetDishesByCategory(@cat
varchar(50))
     RETURNS table AS
      RETURN
      SELECT dishid, dishname, categoryname from
allDishes
       where CategoryName = (@cat)
go
```

Funkcja **udfGetDishesFrom**

Zwraca pozycję które weszły do menu w przeciągu ostatnich `input` dni

```
CREATE FUNCTION udfGetDishesFrom(@input int)
  RETURNS table AS
      RETURN
      SELECT menupositionid, dishid, dishname,
dishprice, indate, outdate
      FROM allMenuPositions MP
      WHERE MP.InDate >= DATEADD(day,-@input,
GETDATE())
go
```

Funkcja udfGetMenuPositionsWithPrice

Zwraca pozycję w menu które wystąpiły w określonej cenie (price)

```
CREATE FUNCTION udfGetMenuPositionsWithPrice(@price
money)
     RETURNS table AS
      RETURN
      select menupositionid, dishid, dishname,
dishprice, indate, outdate from allMenuPositions aMP
       where aMP.DishPrice = @price
go
```

Funkcja udfGetMenuPositionsWithPriceHigherThan

Zwraca pozycję w menu które wystąpiły w cenie większej niż (price)

```
CREATE FUNCTION
udfGetMenuPositionsWithPriceHigherThan(@price money)
     RETURNS table AS
      RETURN
      select menupositionid, dishid, dishname,
dishprice, indate, outdate from allMenuPositions aMP
       where aMP.DishPrice > @price
go
```

Funkcja udfGetMenuPositionsWithPriceLowerThan

Zwraca pozycję w menu które wystąpiły w cenie mniejszej niż (price)

```
CREATE FUNCTION
udfGetMenuPositionsWithPriceLowerThan(@price money)
     RETURNS table AS
      RETURN
      select menupositionid, dishid, dishname,
dishprice, indate, outdate from allMenuPositions aMP
       where aMP.DishPrice < @price
go
```

Funkcja udfGetOrderPrice

Zwraca wartość danego zamówienia (order_id)

```
CREATE FUNCTION udfGetOrderPrice(@order_id int)
RETURNS FLOAT AS
  BEGIN
      DECLARE @sum float;

      set @sum = (
          select [Order price]
          from CustomersOrders
          where OrderID = @order_id
          )

      RETURN @sum
  END
go
```

Funkcja udfGetOrdersDone

Zwraca ilość zamówień danego klienta (customer_id)

```
CREATE FUNCTION udfGetOrdersDone(@customer_id int)
RETURNS FLOAT AS
  BEGIN
      DECLARE @sum float;

      set @sum = (
          select [number of orders]
          from CustomersNumbersOfOrders
          where CustomerID = @customer_id
          )

      RETURN @sum
  END
go
```

Funkcja udfGetOrdersInYear

Zwraca zamówienia złożone w danym roku (year)

```
CREATE FUNCTION udfGetOrdersInYear(@year int)
  RETURNS table AS
      RETURN
      SELECT ORDERID, CUSTOMERID, CUSTOMER, [ORDER
PRICE], ORDERDATE, [COMPANY/PERSON], YEAR, MONTH
      FROM CustomersOrders CO
      WHERE CO.Year = @year
go
```

Funkcja udfGetOrdersInYearAndMonth

Zwraca zamówienia złożone w danym roku (year) i miesiącu (month)

```
CREATE FUNCTION udfGetOrdersInYearAndMonth(@year
int,@month int)
  RETURNS table AS
      RETURN
      SELECT ORDERID, CUSTOMERID, CUSTOMER, [ORDER
PRICE], ORDERDATE, [COMPANY/PERSON], YEAR, MONTH
      FROM CustomersOrders CO
      WHERE CO.Year = @year and CO.Month = @month
go
```

Funkcja udfGetReservationsFrom

Zwraca rezerwacje z ostatnich `input` dni

```
CREATE FUNCTION udfGetReservationsFrom(@input int)
  RETURNS table AS
      RETURN
      SELECT R.ReservationID, CustomerID,
ReservationStatusName, StartDate, Customer
      FROM allReservations R
      WHERE R.StartDate >= DATEADD(day,-@input,
GETDATE())
go
```

Funkcja udfGetReservationsInYear

Zwraca rezerwacje z danego roku (year)

```
CREATE FUNCTION udfGetReservationsInYear(@year int)
  RETURNS table AS
      RETURN
      SELECT R.ReservationID, CustomerID,
ReservationStatusName, StartDate, Customer
      FROM allReservations R
      WHERE R.Year = @year
go
```

Funkcja udfGetReservationsInYearAndMonth

Zwraca rezerwacje z danego roku (year) oraz miesiąca (month)

```
CREATE FUNCTION
udfGetReservationsInYearAndMonth(@year int,@month
int)
  RETURNS table AS
      RETURN
      SELECT R.ReservationID, CustomerID,
ReservationStatusName, StartDate, Customer
      FROM allReservations R
      WHERE R.Year = @year and R.Month = @month
go
```

Funkcja udfGetInvoiceByOrderID

Wyświetla szczegóły do faktury dla zamówienia o numerze @id

```
CREATE FUNCTION udfGetInvoiceByOrderID(@id int)
  RETURNS table AS
      RETURN
      select OrderID,
             CustomerID,
             MenuPositionID,
             DishPrice,
             Quantity,
             discount,
             [Order price],
             OrderDate,
             Year,
             Month
      from ordersInvoice OI
       where OI.OrderID = @id
go
```

Funkcja udfGetInvoiceByCustomerIDandMonth

Wyświetla szczegóły do miesięcznej faktury dla klienta o numerze ID @customerID w danym miesiącu @month

```
CREATE FUNCTION
udfGetInvoiceByCustomerIDandMonth(@customerID int,
@month int)
  RETURNS table AS
      RETURN
      select OrderID,
             CustomerID,
             MenuPositionID,
             DishPrice,
             Quantity,
             discount,
             [Order price],
             OrderDate,
             Year,
             Month
      from ordersInvoice OI
       where OI.CustomerID = @customerID and @month =
Month
go
```

Procedury:

Procedura uspInsertCategory

Dodaje kategorię o podanej nazwie “categoryName”

```
CREATE PROCEDURE uspInsertCategory @categoryName
varchar(50)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS( SELECT * FROM Categories
                WHERE CategoryName = @categoryName )
            BEGIN;
                THROW 52000, N'category already
exist', 1
            END
       DECLARE @CategoryID INT
       SELECT @CategoryID = ISNULL(MAX(CategoryID),
0) + 1
       FROM Categories
       INSERT INTO Categories(CategoryID,
CategoryName)
       VALUES (@CategoryID, @categoryName);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspRemoveCategory

Usuwa kategorię o podanej nazwie “categoryName”, jeśli nie posiada ona przypisanych dań,

```
CREATE PROCEDURE uspRemoveCategory @categoryName
varchar(50)
AS
BEGIN
   SET NOCOUNT ON
  BEGIN TRY
      IF NOT EXISTS( SELECT * FROM Categories
              WHERE CategoryName = @categoryName )
          BEGIN;
              THROW 52000, N'category doesnt exist', 1
          END
      DECLARE @CatId int
      SELECT @CatId = CategoryID from Categories
      where CategoryName = @categoryName
      IF EXISTS( SELECT * FROM Dishes
              WHERE CategoryID = @CatId )
          BEGIN;
              THROW 52000, N'Kategoria posiada przypisane
dania.', 1
          END
      DELETE FROM Categories
      WHERE CategoryName = @categoryName
  END TRY
  BEGIN CATCH
      DECLARE @msg nvarchar(2048)
          =N'error: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
end
go
```

Procedura uspInsertCity

Dodaje miasto o nazwie “cityName” z kraju o ID “CountryID”

```
CREATE PROCEDURE [dbo].[uspInsertCity] @cityName
nvarchar(50), @CountryID int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       DECLARE @CityID INT
       SELECT @CityID = ISNULL(MAX(@CityID), 0) + 1
from Cities
       INSERT INTO Cities(cityid, cityname,
countryid)
       VALUES (@CityID, @cityName, @CountryID)
       SET @CityID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) =
                'Bład przy dodawaniu miasta:' +
CHAR(13) + CHAR(10) +
                ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

Procedura uspRemoveCity

Usuwa miasto o nazwie o ID “CityID”, jeśli żadne użytkownik nie pochodzi z tego miasta,

```
CREATE PROCEDURE uspRemoveCity @CityID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Cities
                WHERE CityID = @CityID
            )
            BEGIN
                ;
                THROW 52000, N'City doesnt exist', 1
            END
       IF EXISTS(
               SELECT *
               FROM Customers
               WHERE CityID = @CityID
           )
           BEGIN
               ;
               THROW 52000, N'Miasto w użyciu', 1
           END
       DELETE FROM Cities
       WHERE CityID = @CityID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertCompany

Dodaje firmę o nazwie “companyName” oraz NIPie “NIP”
z miasta o ID “CityID” do tabeli Company i Customers

```
CREATE PROCEDURE [dbo].[uspInsertCompany]
@companyName nvarchar(50), @NIP nvarchar(50), @CityID
int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @CustomerID INT
        SELECT @CustomerID = ISNULL(MAX(CustomerID),
0) + 1 from Customers
        INSERT INTO Company(CustomerID, CompanyName,
NIP)
        VALUES (@CustomerID, @companyName, @NIP)
        INSERT INTO Customers(CustomerID, CityID)
        VALUES (@CustomerID, @CityID)
        SET @CustomerID = @@IDENTITY
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) =
                'Bład przy dodawaniu firmy:' +
CHAR(13) + CHAR(10) +
                ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura uspRemoveCompany

Usuwa firmę o ID “CustomerID” z tabeli Company oraz Customers, jeśli dana firma nie posiada historii zamówień

```
CREATE PROCEDURE uspRemoveCompany @CustomerID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Company
                WHERE @CustomerID = CustomerID
            )
            BEGIN
                ;
                THROW 52000, N'Company doesnt exist', 1
            END
       IF EXISTS(
               SELECT *
               FROM CompanyReservations
               WHERE CustomerID = @CustomerID
           )
           BEGIN
               ;
               THROW 52000, N'Firma posiada historię
rezerwacji', 1
           END
       DELETE FROM Company
       WHERE @CustomerID = CustomerID
       EXEC uspRemoveCustomer @CustomerID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertCountry

Dodaje państwo o nazwie “countryName” do tabeli Countries

```
CREATE PROCEDURE [dbo].[uspInsertCountry]
@countryName nvarchar(50)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @CountryID INT
        SELECT @CountryID = ISNULL(MAX(CountryID), 0)
+ 1 from Countries
        INSERT INTO Countries(CountryID, CountryName)
        VALUES (@CountryID, @countryName)
        SET @CountryID = @@IDENTITY
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) =
                'Błąd przy dodawaniu państwa:' +
CHAR(13) + CHAR(10) +
                ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura uspRemoveCountry

Usuwa państwo o ID “CountryID” z tabeli Countries oraz miasta leżące w danym państwie

```
CREATE PROCEDURE uspRemoveCountry @CountryID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Countries
                WHERE CountryID = @CountryID
            )
            BEGIN
                ;
               THROW 52000, N'Country doesnt exist', 1
            END
       DELETE from Cities
       where CountryID = @CountryID
       DELETE FROM Countries
       WHERE CountryID = @CountryID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertDish

Dodaje danie o danej nazwie “dishName” z kategorii “catName” o cenie “dishPrice” do tabeli Dishes

```
CREATE PROCEDURE uspInsertDish @dishName varchar(50), @catName
varchar(50), @dishPrice money
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS( SELECT * FROM Dishes
                WHERE DishName = @dishName )
           BEGIN;
               THROW 52000, N'dish already exist', 1
           END
       IF NOT EXISTS( SELECT *
                FROM Categories
                WHERE CategoryName = @catName )
           BEGIN;
               THROW 52000, 'no such category', 1
           END
       DECLARE @CategoryID INT
       SELECT @CategoryID = CategoryID
       FROM Categories
       WHERE CategoryName = @catName
       DECLARE @ProductID INT
       SELECT @ProductID = ISNULL(MAX(DishID), 0) + 1
       FROM Dishes
       INSERT INTO Dishes(DishID, DishName, CategoryID, DishPrice)
       VALUES (@ProductID, @dishName, @CategoryID, @dishPrice);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspRemoveDish

Usuwa danie o danej nazwie “dishName” z tabeli Dishes, jeśli dane danie nie znajduje się w MenuPositions,

```
CREATE PROCEDURE uspRemoveDish @dishName varchar(50)
AS
BEGIN
   SET NOCOUNT ON
  BEGIN TRY
      IF NOT EXISTS( SELECT * FROM Dishes
               WHERE DishName = @dishName )
          BEGIN;
              THROW 52000, N'dish doesnt exist', 1
          END
      DECLARE @DisID int
      select @DisID = DishID from Dishes
      where DishName = @dishName
      IF EXISTS( SELECT * FROM MenuPositions
               WHERE DishID = @DisID )
          BEGIN;
              THROW 52000, N'Danie w użyciu', 1
          END
      DELETE FROM Dishes
      WHERE DishName = @dishName
  END TRY
  BEGIN CATCH
      DECLARE @msg nvarchar(2048)
          =N'error: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
end
go
```

Procedura uspInsertEmployee

Dodaje pracownika o danym imieniu “firstname”, nazwisku “lastname”, numerze telefonu “phone” z przełożonym o ID “reportsto” do tabeli Employees oraz Person

```
CREATE PROCEDURE [dbo].[uspInsertEmployee] @firstname
nvarchar(50), @lastname nvarchar(50), @phone nvarchar(50),
                                          @reportsto int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       DECLARE @EmployeeID INT
       SELECT @EmployeeID = ISNULL(MAX(EmployeeID), 0) + 1
from Employees
       DECLARE @PersonID INT
       SELECT @PersonID = ISNULL(MAX(PersonID), 0) + 1 from
Person
       EXEC uspInsertPerson @firstname, @lastname, @phone,
@PersonID
       INSERT INTO Employees(EmployeeID, PersonID, ReportsTo)
       VALUES (@EmployeeID, @PersonID, @reportsto)
       SET @EmployeeID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) =
               'Blad przy dodawaniu pracownika:' + CHAR(13) +
CHAR(10) +
               ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

Procedura uspRemoveEmployee

Usuwa pracownika o danym ID “EmployeeID” z tabeli Employees oraz Person, jeśli dany pracownik nie posiada historii obsłużonych zamówień,

```
CREATE PROCEDURE uspRemoveEmployee @EmployeeID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT *
                FROM Employees
                WHERE EmployeeID = @EmployeeID )
            BEGIN;
                THROW 52000, N'Employee doesnt exist', 1
            END
       IF EXISTS( SELECT *
               FROM Orders
               WHERE EmployeeID = @EmployeeID )
           BEGIN;
               THROW 52000, N'Pracownik posiada historię
zamówień', 1
           END
       DECLARE @PersonID int
       SET @PersonID = (SELECT PersonID FROM Employees
                WHERE EmployeeID = @EmployeeID )
       DELETE FROM Employees
       WHERE EmployeeID = @EmployeeID
       EXEC uspRemovePerson @PersonID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertIndividualCustomer

Dodaje klienta o danym imieniu “firstname”, nazwisku “lastname”, numerze telefonu “phone” z miasta “CityID” do tabeli IndividualCustomers, Customers oraz Person

```
CREATE PROCEDURE [dbo].[uspInsertIndividualCustomer]
@firstname nvarchar(50), @lastname nvarchar(50), @phone
nvarchar(50), @CityID int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       DECLARE @CustomerID INT
       SELECT @CustomerID = ISNULL(MAX(CustomerID), 0) +
1 from Customers
       DECLARE @PersonID INT
       SELECT @PersonID = ISNULL(MAX(PersonID), 0) + 1
from Person
       EXEC uspInsertPerson @firstname, @lastname,
@phone, @PersonID
     INSERT INTO Customers(CustomerID, CityID)
       VALUES (@CustomerID, @CityID)
       INSERT INTO IndividualCustomers(CustomerID,
PersonID)
       VALUES (@CustomerID, @PersonID)
   SET @CustomerID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) =
       'Blad przy dodawaniu klienta indywidualnego:' +
CHAR(13) + CHAR(10) +
ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

Procedura uspRemoveIndividualCustomer

Usuwa klienta indywidualnego o ID “CustomerID” z tabeli IndividualCustomers, Customers oraz Person, jeśli dany klient indywidualny nie posiada historii zamówień,

```
CREATE PROCEDURE uspRemoveIndividualCustomer @CustomerID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT * FROM IndividualCustomers
                WHERE @CustomerID = CustomerID )
           BEGIN;
               THROW 52000, N'Individual customer doesnt
exist', 1
           END
       IF EXISTS( SELECT *
               FROM IndividualReservations
               WHERE CustomerID = @CustomerID )
          BEGIN;
               THROW 52000, N'Użytkownik posiada historię
rezerwacji', 1
          END
       DECLARE @PersonID int
       SET @PersonID = (SELECT PersonID
                FROM IndividualCustomers
                WHERE @CustomerID = CustomerID)
       DELETE FROM IndividualCustomers
       WHERE @CustomerID = CustomerID
       EXEC uspRemovePerson @PersonID
       EXEC uspRemoveCustomer @CustomerID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertPerson

Dodaje dane osoby o imieniu “firstname”, nazwisku “lastname”, numerze telefonu “phone” i “PersonID” do tabeli Person

```
CREATE PROCEDURE uspInsertPerson @firstname
nvarchar(50), @lastname nvarchar(50), @phone
nvarchar(50), @PersonID int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF (
           len(@phone) != 9
           )
           BEGIN
               ;
               THROW 52000, 'Zła długość numeru
telefonu', 1
           END
       INSERT INTO Person(PersonID, LastName,
FirstName, Phone)
       VALUES (@PersonID, @lastname, @firstname,
@phone);
       SET @PersonID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = 'Bład przy
dodawaniu osoby:' +
                                     CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

Procedura uspRemovePerson

Usuwa dane o osobie o ID “PersonID” z tabeli Person

```
CREATE PROCEDURE uspRemovePerson  @PersonID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Person
                WHERE @PersonID = PersonID
            )
            BEGIN
                ;
                THROW 52000, N'Person doesnt exist', 1
            END
       DELETE FROM Person
       WHERE @PersonID = PersonID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspRemoveCustomer

Usuwa klienta z tablicy Customers o ID “CustomerID”

```
CREATE PROCEDURE uspRemoveCustomer @CustomerID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Customers
                WHERE @CustomerID = CustomerID
            )
            BEGIN
                ;
                THROW 52000, N'Customer doesnt exist',
1
            END
       DELETE FROM Customers
       WHERE @CustomerID = CustomerID
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
end
go
```

Procedura uspInsertMenuPosition

Dodaje do tabeli MenuPosition danie o ID “DishID” z datą końca “OutDate”

```
CREATE PROCEDURE uspInsertMenuPosition @DishID int,
@OutDate datetime
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Dishes
                WHERE DishID = @DishID
           )
           BEGIN
                ;
                THROW 52000, 'Nie ma takiej
potrawy', 1
           END
       DECLARE @DishPrice money
       SELECT @DishPrice = DishPrice
       FROM Dishes
       WHERE DishID = @DishID
       DECLARE @MenuPosID int
       SELECT @MenuPosID =
ISNULL(MAX(MenuPositionID), 0) + 1
       from MenuPositions
       IF EXISTS(
                SELECT *
                FROM MenuPositions
                WHERE DishID = @DishID
                  and (OutDate is null or OutDate >
GETDATE())
```

```
            )
            BEGIN
                ;
                THROW 52000, 'Taka potrawa jest już
w Menu', 1
            END
        INSERT INTO MenuPositions(MenuPositionID,
DishID, DishPrice, InDate, OutDate)
        VALUES (@MenuPosID, @DishID, @DishPrice,
GETDATE(), @OutDate);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy do menu: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura uspInsertOrder

Dodaje zamówienie i dotyczące go dane do tabeli Orders oraz OrderDetails

```
CREATE PROCEDURE uspInsertOrder @EmployeeID int,
                                @CustomerID int,
                                @OutDate datetime,
                                @PaidDate datetime,
                                @TakeawayStatus int,
                                @PaymentStatus int,
                                @OrderedFood
OrderedFood READONLY
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(SELECT *
                FROM @OrderedFood O
                         inner join MenuPositions M
on O.MenuPositionID = M.MenuPositionID
                         inner join Dishes D on
M.DishID = D.DishID
                where D.CategoryID = 10)
       BEGIN
       IF NOT EXISTS(
                SELECT *
                FROM @OrderedFood O
                         inner join MenuPositions M
on O.MenuPositionID = M.MenuPositionID
                         inner join Dishes D on
M.DishID = D.DishID
                where D.CategoryID = 10
                  and (DATENAME(WEEKDAY, @OutDate) =
'Thursday' or DATENAME(WEEKDAY, @OutDate) =
```

```
'Friday' or
                        DATENAME(WEEKDAY, @OutDate) =
'Saturday')
                    and ((3 <= DATEDIFF(day,
GETDATE(), @OutDate) and
                        DATEDIFF(day, GETDATE(),
@OutDate) <= 5 and
                        DATENAME(WEEKDAY, GETDATE())
!= 'Tuesday' and
                        DATENAME(WEEKDAY, GETDATE())
!= 'Wednesday') or (DATEDIFF(day, GETDATE(),
@OutDate) > 5)))
            BEGIN;
                THROW 52000, N'Zamówienie złożone za
późno', 1
            END
        END
        DECLARE @OrderID INT
        SELECT @OrderID = ISNULL(MAX(OrderID), 0) +
1
        FROM Orders
        INSERT INTO Orders(OrderID, EmployeeID,
CustomerID, OrderDate, OutDate, PaidDate,
TakeawayStatus, PaymentStatus)
        VALUES (@OrderID, @EmployeeID, @CustomerID,
GETDATE(), @OutDate, @PaidDate, @TakeawayStatus,
@PaymentStatus)
        DECLARE @RowCnt int
        SELECT @RowCnt = COUNT(0) FROM @OrderedFood;
        DECLARE @RowNumber int
        SET @RowNumber = 1
        WHILE @RowNumber <= @RowCnt
            BEGIN
                DECLARE @MenuPosID int
                SELECT @MenuPosID = MenuPositionID
                from @OrderedFood
                WHERE OrderedFoodID = @RowNumber
```

```
                DECLARE @Quant int
                SELECT @Quant = Quantity
                from @OrderedFood
                WHERE OrderedFoodID = @RowNumber
                EXEC uspInsertOrderDetailsToOrder
@OrderID, @MenuPosID, @Quant
                SET @RowNumber = @RowNumber + 1
            END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania zamówienia: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura uspInsertOrderDetailsToOrder

Dodaje szczegóły dotyczące zamówenia o ID “OrderID” tzn. jakie danie oraz w jakiej ilości zamówiono

```
CREATE PROCEDURE uspInsertOrderDetailsToOrder @OrderID int,
                                              @MenuPositionID int,
                                              @Quantity int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT *
                FROM MenuPositions
                WHERE MenuPositionID = @MenuPositionID )
           BEGIN;
               THROW 52000, 'Nie ma takiej potrawy', 1
           END
       IF NOT EXISTS( SELECT * FROM Orders
                WHERE OrderID = @OrderID )
           BEGIN;
               THROW 52000, 'Nie ma takiego zamowienia', 1
           END

           DECLARE @DishID INT
           SELECT @DishID = DishID
           FROM MenuPositions
           WHERE MenuPositionID = @MenuPositionID
           INSERT INTO OrderDetails(OrderID, DishID,
MenuPositionID, Quantity)
           VALUES (@OrderID, @DishID, @MenuPositionID, @Quantity)
       END TRY
       BEGIN CATCH
           DECLARE @msg nvarchar(2048)
               =N'Błąd dodawania szczegółów zamówienia: ' +
ERROR_MESSAGE();
           THROW 52000, @msg, 1
       END CATCH
END
go
```

Procedura uspInsertPersonToTable

Przypisuje stół o ID “TableID” do osoby o ReservationsDetailsID równym “ResDetID”

```
CREATE PROCEDURE uspInsertPersonToTable @TableID int,
                                        @ResDetID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT *
                FROM ReservationsDetails
                WHERE ReservationsDetailsID = @ResDetID )
           BEGIN;
               THROW 52000, 'Nie ma takiej rezerwacji', 1
           END
       IF NOT EXISTS( SELECT *
                FROM Tables
                WHERE TableID = @TableID )
           BEGIN;
               THROW 52000, 'Nie ma takiego stolika', 1
           END
       BEGIN
           update ReservationsDetails
           set TableID = @TableID
           where ReservationsDetailsID = @ResDetID
       end
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'error: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura uspInsertReservation

Procedura obsługuje system rezerwacji dla klienta indywidualnego oraz firmy uzpełniając dla klienta indywidualnego tabele Reservations, IndividualReservations oraz Order a dla firmy tabele Reservations, CompanyReservations oraz ReservationsDetails odpowiednimi danymi

```
CREATE PROCEDURE uspInsertReservation @EmployeeID int,
                                      @CustomerID int,
                                      @StartDate datetime,
                                      @EndDate datetime,
                                      @numberOfPeople int,
                                      @People People
READONLY,
                                      @OrderedFood
OrderedFood READONLY
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Customers
                WHERE CustomerID = @CustomerID
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego klienta', 1
            end

       DECLARE @peopleCond int
       SELECT @peopleCond = ConditionValue
       from ReservationsConditions
       WHERE ReservationConditionID = 1
       IF (@numberOfPeople < @peopleCond)
```

```
            BEGIN
                ;
                THROW 52000, 'Za malo osob', 1
            end

        DECLARE @ReservationID INT
        SELECT @ReservationID = ISNULL(MAX(ReservationID),
0) + 1
        FROM Reservations

        DECLARE @OrderID INT
        SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
        FROM Orders

        IF EXISTS(
                SELECT *
                FROM IndividualCustomers

                WHERE CustomerID = @CustomerID
            )
            BEGIN;
                DECLARE @OrdersDone int
                SET @OrdersDone =
dbo.udfGetOrdersDone(@CustomerID)
                DECLARE @conditionOrdersDone int
                select @conditionOrdersDone =
ConditionValue
                from ReservationsConditions
                where ReservationConditionID = 3

                IF (@conditionOrdersDone > @OrdersDone)
                    BEGIN;
                        THROW 52000, 'Za malo zlozonych
zamowien', 1
                    END
                IF EXISTS(SELECT *
                          FROM @OrderedFood O
                                   inner join
MenuPositions M on O.MenuPositionID = M.MenuPositionID
                                   inner join Dishes D on
```

```
M.DishID = D.DishID
                        where D.CategoryID = 10)
                BEGIN
                    IF NOT EXISTS(
                            SELECT *
                            FROM @OrderedFood O
                                    inner join
MenuPositions M on O.MenuPositionID = M.MenuPositionID
                                    inner join Dishes
D on M.DishID = D.DishID
                            where D.CategoryID = 10
and (DATENAME(WEEKDAY, @StartDate) = 'Thursday' or
DATENAME(WEEKDAY, @StartDate) = 'Friday' or
DATENAME(WEEKDAY, @StartDate) = 'Saturday')
and ((3 <= DATEDIFF(day, GETDATE(), @StartDate) and
DATEDIFF(day, GETDATE(), @StartDate) < 5 and
DATENAME(WEEKDAY, @StartDate) != 'Tuesday' and
DATENAME(WEEKDAY, @StartDate) != 'Wednesday') or
(DATEDIFF(day, GETDATE(), @StartDate) >= 5)))
                            BEGIN;
                                THROW 52000, N'Rezerwacja
złożona za późno', 1
                            END
                    END
                DECLARE @Disc1 float
                SELECT @Disc1 = isnull(DiscountValue, 0)
                from ConstantDiscount
                WHERE CustomerID = @CustomerID
                  and ValidFrom <= GETDATE()
                DECLARE @Disc2 float
                SELECT @Disc2 = isnull(ParamValue, 0)
                from DiscountParamsHist DPH
                         inner join SingleDiscountParams
SDP on DPH.DiscountHistID = SDP.DiscountHistID
                         inner join SingleDiscount SD on
SDP.DiscountID = SD.DiscountID
                WHERE CustomerID = @CustomerID
                DECLARE @Disc float
                SET @Disc1 = isnull(@Disc1, 0)
                SET @Disc2 = isnull(@Disc2, 0)
```

```
                if (@Disc1 >= @Disc2)
                    set @Disc = @Disc1
                else
                    set @Disc = @Disc2

                DECLARE @RowC int
                SELECT @RowC = COUNT(0) FROM @OrderedFood;
                DECLARE @RowN int
                SET @RowN = 1
                DECLARE @Sum money
                SET @Sum = 0

                WHILE @RowN <= @RowC
                    BEGIN
                        DECLARE @MenuPosID int
                        SELECT @MenuPosID = MenuPositionID
                        from @OrderedFood
                        WHERE OrderedFoodID = @RowN

                        DECLARE @Price money
                        SELECT @Price = DishPrice
                        FROM MenuPositions
                        WHERE MenuPositionID = @MenuPosID

                        DECLARE @Quant int
                        SELECT @Quant = Quantity
                        from @OrderedFood
                        WHERE OrderedFoodID = @RowN
                        SET @Sum = @Sum + isnull(@Price *
(1 - @Disc) * @Quant, 0)
                        SET @RowN = @RowN + 1
                    END

                DECLARE @MinPrice money
                SELECT @MinPrice = ConditionValue
                FROM ReservationsConditions
                WHERE ReservationConditionID = 2

                IF (@MinPrice > @Sum)
```

```
                    BEGIN;
                        THROW 52000, N'Za mała wartość
zamówienia', 1
                    END
                INSERT INTO Reservations(ReservationID,
CustomerID, StartDate, EndDate, ReservationDate,

ReservationStatus)
                VALUES (@ReservationID, @CustomerID,
@StartDate, @EndDate, GETDATE(), 3)
                EXEC uspInsertOrder @EmployeeID,
@CustomerID, @StartDate, null, 2, 2, @OrderedFood
                INSERT INTO
IndividualReservations(ReservationID, CustomerID,
OrderID, numberOfPeople, TableID)
                VALUES (@ReservationID, @CustomerID,
@OrderID, @numberOfPeople, null)
            end
        ELSE
            BEGIN
                INSERT INTO Reservations(ReservationID,
CustomerID, StartDate, EndDate, ReservationDate,
ReservationStatus)
                VALUES (@ReservationID, @CustomerID,
@StartDate, @EndDate, GETDATE(), 3)
                INSERT INTO
CompanyReservations(ReservationID, CustomerID,
numberOfPeople)
                VALUES (@ReservationID, @CustomerID,
@numberOfPeople)
                DECLARE @RowCnt int
                SELECT @RowCnt = COUNT(0) FROM @People;
                DECLARE @RowNumber int
                SET @RowNumber = 1
                WHILE @RowNumber <= @RowCnt
                    BEGIN
                        DECLARE @firstname nvarchar(50)
                        SELECT @firstName = firstName
                        from @People
                        where PeopleID = @RowNumber
```

```
                        DECLARE @lastname nvarchar(50)
                        SELECT @lastName = lastName
                        from @People
                        where PeopleID = @RowNumber
                        INSERT INTO
ReservationsDetails(ReservationID, TableID, LastName,
FirstName)
                        VALUES (@ReservationID, null,
@lastname, @firstname)
                        SET @RowNumber = @RowNumber + 1
                    END
            end
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
            =N'Błąd dodania rezerwacji: ' +
ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
    END CATCH
END
go
```

Procedura uspInsertTable

Dodaje stół o pojemności “Quantity” do tabeli Table

```
CREATE PROCEDURE uspInsertTable @Quantity int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF ( @Quantity <= 1 )
            BEGIN;
                THROW 52000, 'Za mała ilość
miejsc', 1
            END
        DECLARE @TableID INT
        SELECT @TableID = ISNULL(MAX(TableID), 0)
+ 1
        FROM Tables
        INSERT INTO Tables(TableID, Quantity)
        VALUES (@TableID, @Quantity);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania stolika: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura uspFindTableToReservation

Przypisuje rezerwacji o danym “ReservationID” stół o ID “TableID” jeśli nie jest on zajęty w czasie rezerwacji oraz posiada wystarczającą ilość miejsc, wtedy wywołuje również procedurę uspInsertTableToTableRes oraz uspConfirmReservation,

```
CREATE PROCEDURE uspFindTableToReservation @ReservationID
int, @Quantity int, @TableID int,
                                                @ResIdTable
ReservationsDetIDtoTableID READONLY
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT * FROM Reservations
                WHERE ReservationID = @ReservationID )
           BEGIN;
                THROW 52000, N'Nie ma takiej rezerwacji',
1
           END

       DECLARE @StartDate datetime
       DECLARE @EndDate datetime
       SELECT @StartDate = StartDate
       FROM Reservations
       where ReservationID = @ReservationID
       SELECT @EndDate = EndDate
       FROM Reservations
       where ReservationID = @ReservationID

       IF EXISTS( SELECT * FROM TableReservations TR
               WHERE TR.TableID = @TableID
                 and (@StartDate between
```

```
TR.TableReservationStart and TR.TableReservationEnd
                    or @EndDate between
TR.TableReservationStart and TR.TableReservationEnd))
            BEGIN;
                THROW 52000, N'Stół niedostępny w danym
terminie.', 1
            end

        DECLARE @TableQuantity int
        SELECT @TableQuantity = Quantity
        from Tables
        where TableID = @TableID
        IF (@Quantity > @TableQuantity)
            BEGIN;
                THROW 52000, N'Za mała ilość miejsc przy
stole', 1
            end


        IF EXISTS( SELECT * FROM IndividualReservations
                WHERE ReservationID = @ReservationID )
            BEGIN
                DECLARE @CheckTableNull int
                SELECT @CheckTableNull = TableID FROM
IndividualReservations
                WHERE ReservationID = @ReservationID
                IF ( @CheckTableNull is not null )
            BEGIN;
                THROW 52000, N'Rezerwacja ma przydzielony
stolik', 1
            END
                UPDATE IndividualReservations
                SET TableID = @TableID
                WHERE ReservationID = @ReservationID
                EXEC uspInsertTableToTableRes @TableID,
@StartDate, @EndDate
                EXEC uspConfirmReservation @ReservationID
            END
        ELSE
            BEGIN
```

```
                DECLARE @RowCnt int
                SELECT @RowCnt = COUNT(0) FROM
@ResIdTable;
                DECLARE @RowNumber int
                SET @RowNumber = 1
                WHILE @RowNumber <= @RowCnt
                    BEGIN
                        DECLARE @ResId int
                        SELECT @ResId = ResIdTable
                        from @ResIdTable
                        where TableResId = @RowNumber
                        update ReservationsDetails
                        set TableID = @TableID
                        where ReservationID =
@ReservationID
                        and ReservationsDetailsID = @ResId
                        SET @RowNumber = @RowNumber + 1
                    END
                EXEC uspInsertTableToTableRes @TableID,
@StartDate, @EndDate
                EXEC uspConfirmReservation @ReservationID
            end
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'error: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura uspUpdateTable

Modyfikuje pojemność stołu o ID “TableID” na równą “Quantity”

```
CREATE PROCEDURE uspUpdateTable @TableID int,
                                @Quantity int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT * FROM Tables
                WHERE TableID = @TableID )
            BEGIN;
                THROW 52000, 'Nie ma takiego stolika.', 1
            END
       IF @Quantity < 2
            BEGIN;
                THROW 52000, N'Za mało miejsc.', 1
            END
       IF @Quantity IS NOT NULL
            BEGIN
                UPDATE Tables
                SET Quantity = @Quantity
                WHERE TableID = @TableID
            END
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd edytowania stolika: ' +
            ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura uspInsertTableToTableRes

Dodaje do tabeli TableReservations stół o ID “TableID” z datami “StartDate” i “EndDate” początku i końca rezerwacji, kiedy dany stół jest zarezerwowany

```
create procedure uspInsertTableToTableRes @TableID int,
@StartDate datetime, @EndDate datetime
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       DECLARE @TableReservationID int
       SELECT @TableReservationID =
ISNULL(MAX(TableReservationID), 0) + 1
       FROM TableReservations
       INSERT INTO TableReservations(TableReservationID,
TableID, TableReservationStart, TableReservationEnd)
       VALUES (@TableReservationID, @TableID, @StartDate,
@EndDate);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodawania stolika do historii
rezerwacji: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura uspCompleteReservation

Zmienia status rezerwacji o ID “ReservationID” na “Completed”

```
CREATE PROCEDURE uspCompleteReservation
@ReservationID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM Reservations
                WHERE ReservationID = @ReservationID
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej
rezerwacji', 1
            END
        BEGIN
            UPDATE Reservations
            SET ReservationStatus = 4
            WHERE Reservations.ReservationID =
@ReservationID
        end
    end try
    begin catch
        declare @msg nvarchar(2048) = 'Error: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    end catch
end
go
```

Procedura uspDeclineReservation

Zmienia status rezerwacji o ID “ReservationID” na “Declined”

```
CREATE PROCEDURE uspDeclineReservation
@ReservationID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM Reservations
                WHERE ReservationID = @ReservationID
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej
rezerwacji', 1
            END
        BEGIN
            UPDATE Reservations
            SET ReservationStatus = 2
            WHERE Reservations.ReservationID =
@ReservationID
        END
    END TRY
    begin catch
        declare @msg nvarchar(2048) = 'Error: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    end catch
end
go
```

Procedura uspConfirmReservation

Zatwierdza rezerwację o ID “ReservationID”.

```
CREATE PROCEDURE uspConfirmReservation
@ReservationID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT * FROM Reservations
                WHERE ReservationID = @ReservationID
            )
            BEGIN;
                THROW 52000, 'Nie ma takiej
rezerwacji', 1
            END
       BEGIN
            UPDATE Reservations
            SET ReservationStatus = 1
            WHERE Reservations.ReservationID =
@ReservationID
       end
   end try
   begin catch
       declare @msg nvarchar(2048) = 'Error: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1
   end catch
end
go
```

Procedura uspClearMenu

Wyrzuca wszystkie dania znajdujące się w aktualnym Menu, tzn. ustawia ich “OutDate” na równy dacie późniejszej o jeden dzień od daty uruchomienia procedury

```
CREATE PROCEDURE uspClearMenu
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       BEGIN
           UPDATE MenuPositions
           SET OutDate = DATEADD(DAY, 1, GETDATE())
           WHERE OutDate is null
              or OutDate >= GETDATE()
       END
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd usuwania dań: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura uspCheckMenu

Wyrzuca dania znajdujące się w aktualnym Menu dłużej niż 2 tygodnie, tzn. ustawia ich “OutDate” na równy dacie uruchomienia procedury,

```
create procedure uspCheckMenu
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       BEGIN
           IF EXISTS(
               select * from MenuPositions
               WHERE DATEDIFF(day, InDate, GETDATE()) >= 14
               and(OutDate is null or OutDate > GETDATE())
               )
           BEGIN
               UPDATE MenuPositions
               SET OutDate = GETDATE()
               WHERE DATEDIFF(day, InDate, GETDATE()) >= 14
               and(OutDate is null or OutDate >= GETDATE())
           end
       END
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd usuwania dań: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Triggery:

Trigger ConfirmIRReservationAfterAddingTableToRes

Akceptuje rezerwacje indywidualną po przypisaniu jej stolika

```
CREATE TRIGGER
ConfirmIRReservationAfterAddingTableToRes
ON IndividualReservations
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;
   UPDATE Reservations SET ReservationStatus = 1
   WHERE ReservationID in (
       select ReservationID from
IndividualReservations
       where tableID is not null
       )
end
```

Trigger ConfirmReservationIfTableToRes

Akceptuje rezerwacje firmową po przypisaniu jej stolika,

```
CREATE TRIGGER ConfirmReservationIfTableToRes
ON ReservationsDetails
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;
   UPDATE Reservations
   SET ReservationStatus = 1
   WHERE ReservationID in(select ReservationID FROM
ReservationsDetails
   WHERE TableID is not null)
end
go
```

Trigger SetPaidStatusIfOrderOut

Zmienia status na “Paid” dla zamówień których data wydania jest wcześniejsza niż data aktualna oraz miały status “Unpaid”,

```
CREATE TRIGGER SetPaidStatusIfOrderOut
ON Orders
AFTER UPDATE
AS
BEGIN
   SET NOCOUNT ON;
   UPDATE Orders
   SET PaidDate = GETDATE()
   WHERE PaymentStatus = 2 and OutDate < GETDATE()
   UPDATE Orders
   SET PaymentStatus = 1
   WHERE PaymentStatus = 2 and OutDate < GETDATE()
end
go
```

Trigger DeleteDeclinedOrderDetails

Usuwa szczegóły rezerwacji, gdy rezerwacja została anulowana

```
CREATE TRIGGER DeleteDeclinedOrderDetails
ON Reservations
FOR DELETE
AS
BEGIN
   SET NOCOUNT ON;
   DELETE FROM OrderDetails
   WHERE OrderID in (
       select IndividualReservations.OrderID from
IndividualReservations
       inner join dbo.Reservations R2 on
R2.ReservationID =
IndividualReservations.ReservationID
       where R2.ReservationStatus = 2
       )
end
```

Trigger CheckMenuPositions

Uruchamia procedurę “uspCheckMenu” jeśli modyfikujemy tabelę MenuPositions

```
CREATE trigger CheckMenuPositions
   on MenuPositions
   after insert, update
   as
   begin
       set nocount on;
       EXECUTE uspCheckMenu
   end
go
```

Trigger DeclineOrCompleteReservation

Zmienia status na “Completed” lub “Decline” dla rezerwacji których “EndDate” jest datą wcześniejsza niż data aktualna oraz miały odpowiednio status “Confirmed” lub “Pending”,

```
create trigger DeclineOrCompleteReservation
   on Reservations
   after insert, update
as
begin
   set nocount on;
   update Reservations
   SET ReservationStatus = 4
   where ReservationStatus = 1 and EndDate < GETDATE()

   update Reservations
   SET ReservationStatus = 2
   where ReservationStatus = 3 and EndDate < GETDATE()
end
go
```

Indeksy:

Indeks Administrators_pk: ustawienie indexu na AdminID w tabeli Administrators:

```
create unique index Administrators_pk
   on Administrators (AdminID)
go
```

Indeks Categories_pk: ustawienie indexu na CategoryID w tabeli Categories:

```
create unique index Categories_pk
   on Categories (CategoryID)
go
```

Indeks Cities_pk: ustawienie indexu na CityID w tabeli Cities:

```
create unique index Cities_pk
   on Cities (CityID)
go
```

Indeks Cities_Ind: ustawienie indexu na CityName w tabeli Cities:

```
create unique index Cities_Ind
   on Cities (CityName, CountryID)
go
```

Indeks Company_pk: ustawienie indexu na CustomerID w tabeli Company:

```
create unique index Company_pk
   on Company (CustomerID)
go
```

Indeks CompanyReservations_pk: ustawienie indexu na ReservationID w tabeli Reservations:

```
create unique index CompanyReservations_pk
   on CompanyReservations (ReservationID)
go
```

Indeks ConstantDiscount_pk: ustawienie indexu na ConstantDiscountID w tabeli ConstantDiscount:

```
create unique index ConstantDiscount_pk
   on ConstantDiscount (ConstantDiscountID)
go
```

Indeks Countries_pk: ustawienie indexu na CountryID w tabeli Countries:

```
create unique index Countries_pk
   on Countries (CountryID)
go
```

Indeks Customers_pk: ustawienie indexu na CustomerID w tabeli Customers:

```
create unique index Customers_pk
   on Customers (CustomerID)
go
```

Indeks DiscountDict_pk: ustawienie indexu na DiscountParamID w tabeli DiscountDict:

```
create unique index DiscountDict_pk
   on DiscountDict (DiscountParamID)
go
```

Indeks DiscountParamsHist_pk: ustawienie indexu na DiscountHistID w tabeli DiscountParamsHist:

```
create unique index DiscountParamsHist_pk
   on DiscountParamsHist (DiscountHistID)
go
```

Indeks Dishes_pk: ustawienie indexu na DishID w tabeli Dishes:

```
create unique index Dishes_pk
   on Dishes (DishID)
go
```

Indeks Employees_pk: ustawienie indexu na EmployeeID w tabeli Employees:

```
create unique index Employees_pk
   on Employees (EmployeeID)
go
```

Indeks IndividualCustomers_pk: ustawienie indexu na CustomerID w tabeli IndividualCustomers:

```
create unique index IndividualCustomers_pk
   on IndividualCustomers (CustomerID)
go
```

Indeks IndividualReservations_pk: ustawienie indexu na ReservationID w tabeli IndividualReservations:

```
create unique index IndividualReservations_pk
   on IndividualReservations (ReservationID)
go
```

Indeks MenuPositions_pk: ustawienie indexu na MenuPositionID w tabeli MenuPositions:

```
create unique index MenuPositions_pk
   on MenuPositions (MenuPositionID)
go
```

Indeks OrderDetails_pk: ustawienie indexu na OrderDetailsID w tabeli OrderDetails:

```
create unique index OrderDetails_pk
   on OrderDetails (OrderDetailsID)
go
```

Indeks Orders_pk: ustawienie indexu na OrderID w tabeli Orders:

```
create unique index Orders_pk
   on Orders (OrderID)
go
```

Indeks PaymentStatus_pk: ustawienie indexu na PaymentStatusID w tabeli PaymentStatus:

```
create unique index PaymentStatus_pk
   on PaymentStatus (PaymentStatusID)
go
```

Indeks Person_pk: ustawienie indexu na PersonID w tabeli Person:

```
create unique index Person_pk
   on Person (PersonID)
go
```

Indeks Person_Ind: ustawienie indexu na (LastName, FirstName, Phone) w tabeli Person:

```
create unique index Person_Ind
   on Person (LastName, FirstName, Phone)
go
```

Indeks Reservations_pk: ustawienie indexu na ReservationID w tabeli Reservations:

```
create unique index Reservations_pk
   on Reservations (ReservationID)
go
```

Indeks ReservationsConditions_pk: ustawienie indexu na ReservationConditionID w tabeli ReservationsConditions:

```
create unique index ReservationsConditions_pk
   on ReservationsConditions (ReservationConditionID)
go
```

Indeks ReservationsDetails_pk: ustawienie indexu na ReservationsDetailsID w tabeli ReservationsDetails:

```
create unique index ReservationsDetails_pk
   on ReservationsDetails (ReservationsDetailsID)
go
```

Indeks ReservationsStatus_pk: ustawienie indexu na ReservationStatusID w tabeli ReservationsStatus:

```
create unique index ReservationsStatus_pk
   on ReservationsStatus (ReservationStatusID) go
```

Indeks SingleDiscount_pk: ustawienie indexu na DiscountID w tabeli SingleDiscount:

```
create unique index SingleDiscount_pk
   on SingleDiscount (DiscountID)
go
```

Indeks SingleDiscountParams_pk_2: ustawienie indexu na SingleDiscountParamsID w tabeli SingleDiscountParams:

```
create unique index SingleDiscountParams_pk_2
   on SingleDiscountParams (SingleDiscountParamsID)
go
```

Indeks TableReservations_pk: ustawienie indexu na TableReservationID w tabeli TableReservations:

```
create unique index TableReservations_pk
   on TableReservations (TableReservationID)
go
```

Indeks Tables_pk: ustawienie indexu na TableID w tabeli Tables:

```
create unique index Tables_pk
   on Tables (TableID)
go
```

Indeks TakeawayStatus_pk: ustawienie indexu na TakeawayStatusID w tabeli TakeawayStatus:

```
create unique index TakeawayStatus_pk
   on TakeawayStatus (TakeawayStatusID)
go
```

Uprawnienia:

Rola Administator:

```
create role Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Administrators
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Categories
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Cities
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Company
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON CompanyReservations
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON ConstantDiscount
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Countries
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Customers
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON DiscountDict
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON DiscountParamsHist
   to Administrator
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Dishes
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Employees
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON IndividualCustomers
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON
IndividualReservations
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON MenuPositions
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON OrderDetails
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Orders
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON PaymentStatus
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Person
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Reservations
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON
ReservationsConditions
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON ReservationsDetails
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON ReservationsStatus
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON SingleDiscount
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON SingleDiscountParams
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON TableReservations
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Tables
   to Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON TakeawayStatus
   to Administrator
```

Rola Pracownik:

```
create role Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
allCompaniesReservations
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON allDishes
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
allIndividualReservations to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON allMenuPositions
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON allReservations
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
allWaitingReservations to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
allTableStatistics to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON allUnpaidOrders
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
AverageOrderPricesForCustomers
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON currentMenu
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON CustomersOrders
   to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
CustomersOrdersSum to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON
CustomersStatistics to Employee
GRANT SELECT, INSERT, UPDATE, DELETE ON DishesInMenuQuant
to Employee
GRANT EXECUTE to Employee
```

Rola Manager:

```
CREATE ROLE Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Reservations to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON
IndividualReservations to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON CompanyReservations
to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON ReservationsDetails
to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON ReservationsStatus to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON tables to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON TableReservations to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON
ReservationsConditions to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Customers to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON IndividualCustomers
to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Company to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON MenuPositions to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Dishes to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Categories to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Orders to Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON OrderDetails to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON SingleDiscount to
Manager
GRANT SELECT, INSERT, UPDATE, DELETE ON ConstantDiscount to
Manager
GRANT EXECUTE to Manager
GRANT SELECT to Manager
```