Podstawy Baz Danych

Dokumentacja do projektu „Restauracja” - czyli systemu wspomagania działalności firmy świadczącej usługi gastronomiczne

Prowadzący
Dr Inż. Robert Marcjan

Autorzy
Damian Tworek, Jakub Kościelniak, Maciej Jedynak

1. Role

1.1. Administrator systemu

Administrator posiada całkowity dostęp do bazy danych, spełnia funkcję architekta informacji, projektanta bazy danych oraz administratora bazy danych.
Admin może zrobić wszystkie rzeczy, co osoby poniżej, a także nadawać uprawnienia administratora

1.2. Obsługa

Obsługa - przydzielamy każdemu pracownikowi zakres jego odpowiedzialności oraz to przed kim odpowiada. Osoba, przed którą odpowiada daje jej zakres autonomii w kwestii

podejmowania niektórych ważnych decyzji. Może potwierdzić rezerwację stolika i zdecydować o złożeniu zamówienia.

1.3. Menadżer

Menedżer to członek obsługi, przed którym inni pracownicy odpowiadają. Może wygenerować raport miesięczny lub tygodniowy dotyczący rezerwacji stolików, rabatów, menu - dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień.

1.4. Właściciel

Właściciel jest członkiem obsługi oraz menedżerem. Ponadto może nadawać uprawnienia właściciela i obsługi oraz przed nikim nie odpowiada.

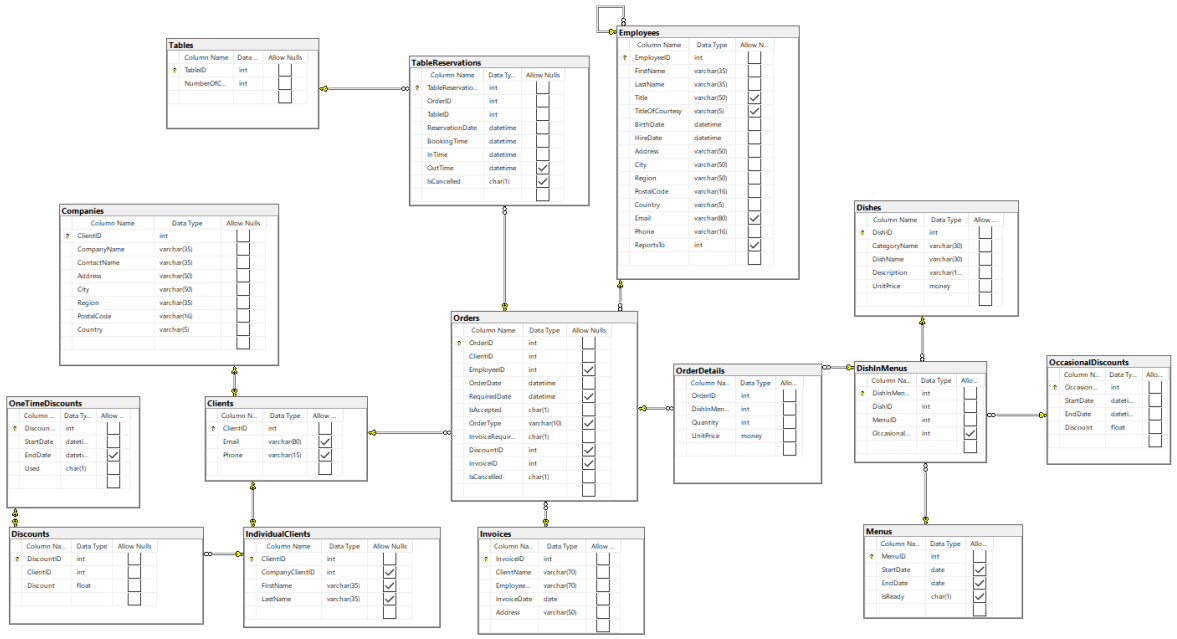
1.5. Klient indywidualny

Klient indywidualny może zamówić pozycje z menu na miejscu lub na wynos (które może być zlecone na miejscu lub wypełniając formularz internetowy), zarezerwować stolik dla co najmniej dwóch osób wypełniając formularz internetowy, zlecić zamówienie na przyszłość na pewną minimalną ustaloną kwotę. Może anulować każdą z tych akcji. Dodatkowo, jako członek firmy może generować raporty dotyczące zamówień oraz rabatów. Klientom indywidualnym przysługują rabaty.

1.6. Firma

Firma może zamówić pozycje z menu na miejscu lub na wynos (które może być zlecone na miejscu lub wypełniając formularz internetowy), zarezerwować stolik dla co najmniej dwóch osób wypełniając formularz internetowy, zlecić zamówienie na przyszłość na pewną minimalną ustaloną kwotę, wygenerować raporty dotyczące zamówień. Może anulować każdą z tych akcji.

2. Schemat Bazy Danych



3. Tabele

3.1. Tabela **Tables**

- TableID - identyfikator danego stolika, stanowi on klucz główny tabeli Table,
- NumberOfChairs - informacja o ilości miejsc przy danym stoliku.

```
create table Tables
(
   TableID        int identity
       constraint PK_Table
           primary key,
   NumberOfChairs int not null
       constraint NumberOfChairsCheck
           check ([NumberOfChairs] > 0)
)
go
```

3.2. Tabela **TableReservations**

- TableReservationID – klucz główny tabeli TableReservations,
- OrderID – identyfikator zamówienia, klucz obcy tabeli Orders,
- TableID – identyfikator, klucz obcy tabeli Tables
- InTime - godzina rozpoczęcia rezerwacji,
- ReservationDate – data i godzina, na którą stolik został zarezerwowany
- BookingTime – przewidywany czas rezerwacji (długość wizyty)
- InTime – czas przysiąścia do stolika
- OutTime- czas odejścia od stolika
- IsCancelled – informacja o tym, czy rezerwacja została anulowana

```
create table TableReservations
(
   TableReservationID int identity
       constraint [PK_Table Reservation]
           primary key,
   OrderID            int      not null
       constraint [FK_Table Reservation_Orders]
           references Orders,
   TableID            int      not null
       constraint [FK_Table Reservation_Table]
           references Tables,
   ReservationDate    datetime not null,
   BookingTime        datetime not null,
   InTime             datetime not null,
   OutTime            datetime,
   IsCancelled        char
       constraint IsCancelledTR
           check ([IsCancelled] = 'n' OR [IsCancelled] = 'y'),
   constraint [CK_Table Reservation]
       check ([InTime] < [OutTime])
)
go
```

3.3. Tabela **Employees**

- EmployeeID - klucz główny tabeli Employees,
- FirstName - Imię pracownika,
- LastName - nazwisko pracownika,
- Title - spełniana funkcja,
- TitleOfCourtesy - posiadany tytuł,

- BirthDate - data urodzenia pracownika,
- HireDate - data zatrudnienia pracownika,
- Address - adres pracownika,
- City - nazwa miasta, w którym mieszka pracownik,
- Region - nazwa regionu, w którym mieszka pracownik,
- PostalCode - kod pocztowy,
- Email - mail kontaktowy do pracownika, przy czym nie jest on wymagany,
- Phone - numer telefonu kontaktowego do pracownika,
- ReportsTo - informacja o tym, komu podlega dany pracownik.

```
create table Employees
(
  EmployeeID int identity
     constraint PK_Employees
        primary key,
  FirstName varchar(35) not null,
  LastName varchar(35) not null,
  Title varchar(50),
  TitleOfCourtesy varchar(5),
  BirthDate datetime not null,
  HireDate datetime not null,
  Address varchar(50) not null,
  City varchar(50) not null,
  Region varchar(50) not null,
  PostalCode varchar(16) not null,
  Country varchar(5) not null,
  Email varchar(80),
  Phone varchar(16) not null,
  ReportsTo int
     constraint FK_Employees_Employees1
        references Employees
     constraint CK_Employees_1
        check ([ReportsTo]>=1),
  constraint CK_Employees
     check ([BirthDate]<[HireDate])
)
go
```

3.4. Tabela **Discounts**

W tej tabeli znajdują się zarówno zniżki na całe życie - 3% procentowe jak i zniżki jednorazowe 5%. Zniżkę na całe życie dostaje się, gdy zrobimy 10 zamówień o wartości przekraczającej 30 zł.

- DiscountID - unikatowy identyfikator zniżki, klucz główny tabeli Discounts,
- ClientID - identyfikator klienta, klucz obcy,
- Discount - procentowa wartość zniżki.

```
create table Discounts
(
   DiscountID int identity
       constraint PK_Discounts
           primary key,
   ClientID   int   not null
       constraint FK_Discounts_IndividualClients
           references IndividualClients,
   Discount   float not null
       constraint CK_Discounts_1
           check ([Discount] > 0 AND [Discount] <= 1)
)
```

3.5. Tabela **OneTimeDiscounts**

W tabeli OneTimeDiscounts znajdują się zniżki jednorazowe 5%.
Zniżkę dostaje się gdy wyda się 1000 zł w restauracji.

- DiscountID - unikatowy identyfikator zniżki, klucz główny tabeli Discounts,
- StartDate - data, informacja o tym kiedy rozpoczyna się okres czasu, w którym można skorzystać ze zniżki,
- EndDate - data, informacja o tym kiedy kończy się okres czasu, w którym można skorzystać ze zniżki, może być na czas nieokreślony wtedy null,
- Used - pole określające, czy rabat został wykorzystany.

```
create table OneTimeDiscounts
(
   DiscountID int      not null
       constraint PK_OneTimeDiscounts_1
           primary key
       constraint FK_OneTimeDiscounts_Discounts
           references Discounts,
   StartDate  datetime not null,
   EndDate    datetime,
```

```
   Used       char     not null
       constraint UsedDiscount
           check ([Used] = 'n' OR [Used] = 'y'),
   constraint DateOneTimeDiscounts
       check ([StartDate] < [EndDate])
)
```

3.6. Tabela **Clients**

- ClientID - unikatowy identyfikator klienta, klucz główny tabeli Clients,
- Email - email firmy,
- Phone - telefon do osoby reprezentującej firmę.

```
create table Clients
(
  ClientID int identity
     constraint PK_Clients_1
        primary key,
  Email varchar(80),
  Phone varchar(15)
)
go
```

3.7. Tabela **Individual Clients**

- ClientID - unikatowy identyfikator klienta, klucz obcy,
- CompanyClientID – identyfikator klienta przyznany firmie, oznaczający że dany klient indywidualny jest jej pracownikiem, nie jest wymagany
- FirstName - imię zamawiającego klienta lub pracownika firmy,
- LastName - nazwisko zamawiającego klienta lub pracownika firmy.

```
create table IndividualClients
(
  ClientID int not null
     constraint [PK_Individual Clients]
        primary key
     constraint FK_IndividualClients_Clients
        references Clients,
  CompanyClientID int,
```

```
  FirstName varchar(35),
  LastName varchar(35)
)
```

3.8. Tabela **Companies**

- ClientID - identyfikator firmy, klucz obcy,
- CompanyName - nazwa firmy,
- ContactName - imię i nazwisko osoby reprezentującej firmę,
- Address - ulica i numer zamieszkania firmy,
- City - miasto firmy,
- Region - region firmy,
- PostalCode - kod pocztowy firmy,
- Country - kraj firmy.

```
create table Companies
(
  ClientID int not null
     constraint PK_Companies
        primary key
     constraint FK_Companies_Clients
        references Clients,
  CompanyName varchar(35) not null,
  ContactName varchar(35) not null,
  Address varchar(50) not null,
  City varchar(50) not null,
  Region varchar(35) not null,
  PostalCode varchar(16) not null,
  Country varchar(5) not null
)
go
```

3.9. Tabela **Dishes**

- DishID - unikatowy identyfikator dania, klucz główny tabeli Dishes,
- CategoryName - kategoria do której należy danie,
- DishName - nazwa dania,
- Description - opis dania,

- UnitPrice - cena dania.

```
create table Dishes
(
   DishID       int identity
       constraint PK_Dishes
           primary key,
   CategoryName varchar(30)  not null,
   DishName     varchar(30)  not null,
   Description  varchar(100) not null,
   UnitPrice    money        not null
       constraint UnitPriceDish
           check ([UnitPrice] >= 0)
)
```

3.10. Tabela **Menus**

- MenuID - klucz główny tabeli Menu,
- StartDate - informacja o tym, kiedy Menu staje się aktualne,
- EndDate - informacja o tym, kiedy Menu przestaje być aktualne,
- IsReady - informuje czy menu jest gotowe do publikacji.

```
create table Menus
(
   MenuID    int identity
       constraint PK_Menu
           primary key,
   StartDate date,
   EndDate   date,
   IsReady   char
       constraint CK_Menus
           check ([IsReady] = 'n' OR [IsReady] = 'y'),
   constraint CK_Menu
       check ([StartDate] < [EndDate])
)
```

3.11. Tabela **DishInMenus**

- DishInMenuID – klucz główny tabeli DishInMenus,
- DishID – identyfikator dania, klucz obcy,
- MenuID – identyfikator menu, klucz obcy,
- OccasionalDiscountID - identyfikator zniżki na danie, klucz obcy.

```
create table DishInMenus
(
      DishInMenuID int identity
            constraint PK_Details
                  primary key,
      DishID int not null
            constraint FK_Details_Dishes
                  references Dishes,
      MenuID int not null
            constraint FK_Details_Menu
                  references Menus,
      OccasionalDiscountID int
            constraint FK_DishInMenus_OccasionalDiscounts
                  references OccasionalDiscounts
)
go
```

3.12. Tabela **OrderDetails**

- OrderID - identyfikator zamówienia,
- DishInMenuID - klucz obcy tabeli DishInMenu,
- Quantity - liczba dań zamówień,
- UnitPrice - uwzględnia zniżkę przyznaną od restauracji na danie - z tabeli OccasionalDiscounts.

```
create table OrderDetails
(
   OrderID      int   not null
       constraint [FK_Order Information_Orders]
           references Orders,
   DishInMenuID int   not null
       constraint [FK_Order Information_Details]
           references DishInMenus,
   Quantity     int   not null
       constraint QuantityCheck
           check ([Quantity] > 0),
   UnitPrice    money not null
```

```
        constraint UnitPriceCheck
            check ([UnitPrice] >= 0)
)
```

3.13. Tabela **Orders**

- OrderID - identyfikator zamówienia,
- ClientID - identyfikator klienta składającego zamówienie, może być null,
- EmployeeID - identyfikator pracownika przypisanego do zamówienia,
- OrderDate - data, kiedy zamówienie zostało złożone,
- RequiredDate - informacja o tym na kiedy zamówienie ma być zrealizowane,
- IsAccepted - czy zamówienie zostało zaakceptowane przez obsługę
- OrderType - typ zamówienia; na wynos/na miejscu lub dostarczane do klienta,
- InvoiceRequired - czy klient chce żeby to zamówienie było w fakturze,
- DiscountID - identyfikator zniżki kllienta,
- InvoiceID – identyfikator faktury, klucz obcy,
- IsCancelled - czy zamówienie zostało anulowane.

```
create table Orders
(
   OrderID         int identity
       constraint PK_ClientOrders
           primary key,
   ClientID        int      not null
       constraint FK_Orders_Clients
           references Clients,
   EmployeeID      int
       constraint FK_ClientOrders_Employees1
           references Employees,
   OrderDate       datetime not null,
   RequiredDate    datetime,
   IsAccepted      char     not null
       constraint IsAccepted
           check ([IsAccepted] = 'n' OR [IsAccepted] = 'y'),
   OrderType       varchar(10)
       constraint OrderTypeCheck
           check ([OrderType] = 'out' OR [OrderType] = 'in'),
   InvoiceRequired char     not null
       check ([InvoiceRequired] = 'n' OR [InvoiceRequired] = 'y'),
   DiscountID      int,
```

```
    InvoiceID       int
        constraint FK_Orders_Invoices
            references Invoices,
    IsCancelled     char     not null
        constraint IsCancelledCheck
            check ([IsCancelled] = 'n' OR [IsCancelled] = 'y'),
    constraint CK_Orders
        check (dateadd(minute, -1, [OrderDate]) <= [RequiredDate])
)
```

3.14. Tabela **OccasionalDiscounts**

W tej tabeli znajdują się zniżki na dania w menu przyznawane od restauracji.

- OccasionalDiscountID- identyfikator zamówienia,
- StartDate - od kiedy zniżka się zaczyna,
- EndDate - do kiedy ta zniżka trwa,
- Discount - wartość zniżki.

```
create table OccasionalDiscounts
(
    OccasionalDiscountID int identity
        constraint PK_OccasionalDiscounts
            primary key,
    StartDate            datetime not null,
    EndDate              datetime not null,
    Discount             float    not null
        constraint CK_OccasionalDiscountsDiscount
            check ([Discount] > 0 AND [Discount] <= 1),
    constraint CK_OccasionalDiscountsDate
        check ([StartDate] < [EndDate])
)
```

3.15. Tabela **Invoices**

W tej tabeli przechowywane są informacje dotyczące wygenerowanych faktur.

- InvoiceID - identyfikator faktury,
- ClientName - nazwa clienta indywidualnego bądź firmy,
- EmployeeName - imię i nazwisko pracownika odpowiedzialnego za wygenerowanie faktury,
- InvoiceDate - data wygenerowanie faktury,

- Address - aktualny adres klienta.

```
CREATE TABLE Invoices(
	InvoiceID int IDENTITY(1,1) NOT NULL,
	ClientName varchar(70) NOT NULL,
	EmployeeName varchar(70) NOT NULL,
	InvoiceDate date NOT NULL,
	Address varchar(50) NOT NULL,
 CONSTRAINT [PK_Invoices] PRIMARY KEY CLUSTERED
(InvoiceID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
```

---

4. Warunki integralności

4.1. Więzy integralności

4.1.1. Companies:

Każde z pól musi być obowiązkowo wypełnione. Możemy przecież jako sprzedawcy wymagać, że firma jest weryfikowalna, możemy jej zaufać. Niebezpieczeństwo utraty produktów, ewentualnych strat finansowych, kiedy sprzedajemy produkty firmom znikąd jest zbyt duże by ryzykować - albo grają w otwarte karty i możemy pociągnąć ich w razie czego do odpowiedzialności, albo w ogóle nie powinny znaleźć się w bazie potencjalnych klientów. To samo powinno dotyczyć klientów indywidualnych działających z ramienia firmy. Dlatego uważam, że każde pole powinno być obligatoryjne, żadne pole nie powinno przyjmować wartości ‘null’. Kiedy dodajemy informacje do tej tabeli, uzupełniamy wszystkie pola. Oczywiście - każde pole z wyjątkiem PK możemy potem modyfikować.

4.1.2. IndividualClients:

Zakładamy że klient indywidualny nie musi przekazywać nam dokładnych danych o sobie. Zbieramy ilość zamówień i ich łączną cenę, żeby wyliczyć zniżki. Jeden klient może mieć wiele zniżek. Istotne jest jednak, że ilość informacji, które zbieramy na temat klienta zależy od tego, czy:

- kupuje produkty na miejscu (także na wynos, nie potrzebne są informacje inne niż ID) lub decydujemy się na dostawę do niego (wtedy wymagana jest dodatkowo informacja o numerze telefonu oraz całościowy adres)
- działa z ramienia firmy (imię i nazwisko oraz ID firmy, wtedy reszta pól jest taka sama jak firmy, dla której działa) lub wyłącznie dla siebie (jak wyżej).

4.1.3. Discounts:

W tej tabeli znajdują się zarówno zniżki na całe życie - 3% procentowe jak i zniżki jednorazowe 5%. Zniżkę na całe życie dostaje się, gdy zrobimy 10 zamówień o wartości przekraczającej 30 zł.

4.1.4. OneTimeDiscounts

W tabeli OneTimeDiscounts znajdują się zniżki jednorazowe 5%.
Zniżkę dostaje się gdy wyda się 1000 zł w restauracji.

4.1.5. ClientOrders:

W zależności od rodzaju zamówienia:
- na miejscu i na wynos - relewantny jest tylko OrderDate, RequiredDate jest nullem
- zamówienie na odległość - istnieją wartości zarówno OrderDate jak i RequiredDate

ClientID i/lub CompanyID muszą mieć wartości inne niż null.

4.1.6. Order Details:

Jest to tabela odpowiedzialna za magazynowanie informacji dotyczących zamówień. Zarówno obniżka jak i ilość może nie istnieć (brak informacji na ten temat) lub wynosić zero (procent lub właściwych jednostek).

4.1.7. Table Reservation:

Każde zamówienie ma dwa typy: na miejscu/na wynos lub dostawa, dla pierwszych można zarezerwować stoliki. Dla każdej rezerwacji ważna jest data rezerwacji (kiedy zarezerwowano) i czas rezerwacji (jak długo trwa). Rezerwować może zarówno firma lub klient indywidualny. Dla każdego zamówienia z rezerwacją można zarezerwować wiele stolików.

4.1.8. Employees:

Standardowe informacje o pracownikach. Zakładamy, że mamy prawie wszystkie informacje o pracownikach, gdyż nie chcemy zatrudniać “podejrzanych” osób. Każdy pracownik może obsługiwać wiele osób, może podlegać jednej osobie i być szefem dla wielu osób.

4.1.9. Dishes

Każde danie ma swój czas dostępności, cenę, identyfikator, nazwę, ilość jednostek w magazynie, które nie mogą być nullami. Każde zamówienie może mieć wiele dań. Rzeczy takie jak opis i obrazek nie są konieczne dla powstania nowego dania. Sprawdzenie czy menu może zostać zmienione będziemy realizować za pomocą funkcji związanej z czasem dostępności dania.

5. Funkcje

5.1. Funkcja IsSpecificTableAvailableAtSpecificTime

Przyjmując jako argumenty przedział czasu do sprawdzenia i numer stolika - zwraca informację o tym, czy w tym przedziale czasu stolik jest wolny.

```
CREATE FUNCTION IsSpecificTableAvailableAtSpecificTime
(
 @tableid int,
 @starttime datetime,
 @endtime datetime = NULL
)
RETURNS char
 AS
 BEGIN

     SET @endtime = ISNULL(@endtime, DATEADD(mi, 30, @starttime))

     IF @tableid IN (SELECT TableID
       FROM Tables T
       WHERE T.TableID NOT IN (SELECT TableID
                       FROM TableReservations
                       WHERE IsCancelled = 'n' AND
                         (OutTime IS NOT NULL AND (
                           (@starttime BETWEEN InTime AND OutTime)
                          OR (@endtime BETWEEN InTime AND OutTime)
                          OR (InTime BETWEEN @starttime AND @endtime)
                          OR (OutTime BETWEEN @starttime AND @endtime)))
                          OR (OutTime IS NULL AND @starttime <= InTime)))
         BEGIN
             RETURN 'n'
         END
     RETURN 'y'
 END
GO
```

5.2. Funkcja OrderOverallPrice

Funkcja wylicza ile należy zapłacić za zamówienie, uwzględniając użyte zniżki.

```
CREATE FUNCTION OrderOverallPrice(@orderNumber INT)
      RETURNS MONEY
  AS
  BEGIN
      DECLARE @Price MONEY = (SELECT (SELECT SUM(Quantity * UnitPrice)
              FROM Orders O
              JOIN OrderDetails OD ON O.OrderID = OD.OrderID
              WHERE O.OrderID = @orderNumber
              GROUP BY O.OrderID) )
    DECLARE @DiscountID INT = (SELECT DiscountID FROM Orders WHERE OrderID=@orderNumber)
    DECLARE @Discount FLOAT = (SELECT Discount FROM Discounts WHERE DiscountID=@DiscountID)
      IF @Discount IS NULL
          SET @Discount = 0

      RETURN @Price * (1-@Discount)
END
```

5.3 Funkcja TotalMoneySpentForClient

Funkcja wylicza ile klient wydał pieniędzy w naszej restauracji.

```
CREATE FUNCTION TotalMoneySpentForClient
   (
   @ClientID INT
   ) RETURNS MONEY
AS
BEGIN
  DECLARE @TotalMoneySpentForClient MONEY  = (SELECT SUM(dbo.OrderOverallPrice(OrderID))
                                              FROM Orders
                                              WHERE ClientID = @ClientID AND O.IsAccepted = 'y' AND
O.IsCancelled = 'n')
  IF @TotalMoneySpentForClient IS NULL
      RETURN 0
  RETURN @TotalMoneySpentForClient
END
```

5.4. Funkcja NumberOfChangedDishes

Funkcja sprawdza ile dań zostało zmienionych między dwoma menuID.

```
CREATE FUNCTION NumberOfChangedDishes(@menuID1 int, @menuID2 int)
   RETURNS int
AS
BEGIN
   DECLARE @number int = (SELECT COUNT()
                          from (SELECT DishID
                                from DishInMenus
                                WHERE MenuID = @menuID1
                                INTERSECT
                                SELECT DishID
                                from DishInMenus
                                WHERE MenuID = @menuID2) as T)
   DECLARE @number1 int = (Select COUNT()
                           from DishInMenus
                           WHERE MenuID = @menuID1)
   Return @number1 - @number
END
```

6. Widoki

6.1. Widok DishesToday

Widok dań dostępnych dzisiaj w restauracji.

```
CREATE VIEW DishesToday
AS
SELECT MenuID,
      DishInMenuID,
      DishName,
      CategoryName,
      UnitPrice,
      ISNULL(OD.Discount, 0) Discount,
      UnitPrice * (1 - ISNULL(OD.Discount, 0)) PriceAfterDiscount
FROM DishInMenus DIM
        JOIN Dishes D on D.DishID = DIM.DishID
        LEFT OUTER JOIN OccasionalDiscounts OD on OD.OccasionalDiscountID = DIM.OccasionalDiscountID
WHERE EXISTS(
            SELECT MenuID
            FROM Menus
            WHERE GETDATE() BETWEEN StartDate AND EndDate)
GO
```

6.2. Widok DishesInMenuView

Ile razy danie występowało we wszystkich Menu.

```
CREATE VIEW DishesInMenusView
AS
SELECT d.DishID, d.Dishname, COUNT(*) amount
FROM Dishes d
JOIN DishInMenus di ON d.DishId = di.DishID
GROUP BY d.DishID, d.DishName
```

6.3. Widok CancelledReservationsByCompanies

Wypisuje informacje o anulowanych rezerwacjach dokonanych przez firmy.

```
CREATE VIEW CancelledReservationsByCompanies
AS
SELECT c.ClientID, c.CompanyName, c.ContactName, tr.ReservationDate, tr.TableID
FROM Companies c, Orders o, TableReservations tr
WHERE tr.IsCancelled = 'Y'
    AND tr.OrderID = o.OrderID
    AND o.ClientID = c.ClientID
```

6.4. Widok CancelledReservationsByIndividualClients

Wypisuje informacje o anulowanych rezerwacjach dokonanych przez klientów indywidualnych.

```
CREATE VIEW CancelledReservationsByIndividualClients
AS
SELECT ic.ClientID, ic.FirstName, ic.LastName, tr.ReservationDate, tr.TableID
FROM IndividualClients ic, Orders o, TableReservations tr
WHERE tr.IsCancelled = 'Y'
  AND tr.OrderID = o.OrderID
  AND o.ClientID = ic.ClientID
```

6.5. Widok CancelledOrdersByIndividualClients

Wyszczególnia informacje o anulowanych zamówieniach klientów indywidualnych.

```
CREATE VIEW CancelledOrdersByIndividualClients
AS
SELECT ic.ClientID, ic.FirstName, ic.LastName, o.OrderID
FROM IndividualClients ic, Orders o
WHERE o.IsCancelled = 'Y'
  AND o.ClientID = ic.ClientID
```

6.6. Widok CancelledOrdersByCompanies

Wyszczególnia informacje o anulowanych zamówieniach firm.

```
CREATE VIEW CancelledOrdersByCompanies
AS
SELECT c.ClientID, c.CompanyName, c.ContactName, o.OrderID
FROM Companies c, Orders o
WHERE o.IsCancelled = 'Y'
AND o.ClientID = c.ClientID
```

6.7 Widok OverallSalesPerClient

Zwraca dla każdego klienta ile wydał pieniędzy.

```
CREATE VIEW OverallSalesPerClient
AS
   SELECT ClientID, dbo.TotalMoneySpentForClient(ClientID) AmountOfMoneySpentByClient
   FROM Orders
```

7. Procedury

7.1. Procedura MakeOrder

Procedura MakeOrder składa zamówienie. Stworzy instancję klienta, jeśli nie zostało podane ID klienta. Zapewnia, że zamówienie nie jest składane w przeszłości oraz, że została poprawna podana zniżka.

```
CREATE PROCEDURE MakeOrder
@RequiredDate datetime,
@OrderType varchar(10),
@InvoiceRequired char(1),
@ClientID int = NULL,
@CompanyID int = NULL,
@FirstName varchar(35) = NULL,
@LastName varchar(35) = NULL,
@Email varchar(80) = NULL,
@Phone varchar(15) = NULL,
@DiscountID int = NULL

AS
BEGIN

   IF DATEADD(minute, -1, GETDATE()) > @RequiredDate
       RAISERROR('RequiredDate must not be in past', 16, 1)

   IF @@ERROR != 0
       RETURN -1


   IF @ClientID IS NULL
       BEGIN
       EXEC AddIndividualClient @CompanyID,@FirstName,@LastName,@Email,@Phone
```

```
        set @ClientID = (SELECT MAX(ClientID) FROM Clients)
        PRINT 'Added client of ID: ' + STR(@ClientID)
        END

    IF @DiscountID is not null
        BEGIN
        DECLARE @ReusableDiscountID int = (SELECT DiscountID
                                           FROM Discounts
                                           WHERE DiscountID = @DiscountID
                                           AND DiscountID not in(
                                               SELECT DiscountID
                                               FROM OneTimeDiscounts
                                               WHERE OneTimeDiscounts.DiscountID = @DiscountID
                                               ))

        DECLARE @OneTimeDiscountID int = (SELECT DiscountID
        FROM OneTimeDiscounts
        WHERE DiscountID=@DiscountID AND Used ='n' AND (GETDATE() BETWEEN StartDate and EndDate))

        IF (@ReusableDiscountID is null AND @OneTimeDiscountID is null)
            RAISERROR('Selected Discount is not present in database.', 18, 1)

        DECLARE @DiscountBelongsToClientID int = (SELECT ClientID FROM Discounts WHERE DiscountID=@DiscountID)

        IF @DiscountBelongsToClientID != @ClientID
            RAISERROR('Selected Discount does not belong to Client.', 18, 1)
        END

    IF @@ERROR != 0
        RETURN -1

    INSERT INTO Orders(ClientID, EmployeeID, OrderDate, RequiredDate, IsAccepted, OrderType, InvoiceRequired,
DiscountID, InvoiceID, IsCancelled)
    VALUES (@ClientID, NULL, GETDATE(), @RequiredDate, 'n', @OrderType, @InvoiceRequired, @DiscountID, NULL, 'n')
END
```

7.2. Procedura AddDishToOrder

Procedura AddDishToOrder sprawdza czy zamówienie jest poprawne. Nie pozwoli złożyć zamówienia na owoce morza w dzień realizacji będący innym niż czwartek, piątek lub sobota. Co więcej, zamówienie na to musi być wykonane do poniedziałku poprzedzającego zamówienie. Zapewnia również poprawność ceny jednostkowej dania, uwzględniając dostępną zniżkę restauracyjną na danie. W końcu dodaje danie do zamówienia.

```
CREATE PROCEDURE [dbo].[AddDishToOrder] @OrderID int,
                                        @DishInMenuID int,
                                        @Quantity int
AS
BEGIN
    DECLARE @OrderDate datetime = (SELECT OrderDate FROM Orders WHERE OrderID = @OrderID)
    DECLARE @RequiredDate datetime = (SELECT RequiredDate FROM Orders WHERE OrderID = @OrderID)

    DECLARE @isCancelled char = (SELECT IsCancelled FROM Orders WHERE OrderID = @OrderID)
    IF @isCancelled = 'y'
        RAISERROR ('The Order is cancelled', 18, 1)
```

```
    DECLARE @DishAvailable int = (SELECT COUNT(DishInMenuID)
                                  FROM DishInMenus
                                           JOIN Menus M on DishInMenus.MenuID = M.MenuID
                                  WHERE DishInMenuID = @DishInMenuID
                                    AND M.IsReady = 'y'
                                    AND @RequiredDate BETWEEN M.StartDate AND M.EndDate)
    IF @DishAvailable = 0
        RAISERROR ('The Dish is not available', 18, 1)


    DECLARE @DishCategoryName varchar(30) = (SELECT CategoryName
                                             FROM DishInMenus
                                                      JOIN Dishes D on DishInMenus.DishID = D.DishID
                                             WHERE DishInMenuID = @DishInMenuID)


    DECLARE @dayOfWeekRequired int = (SELECT (DATEPART(weekday, @RequiredDate)))

    DECLARE @dayDifference int = (DATEDIFF(day, @OrderDate, @RequiredDate))

    IF @DishCategoryName = 'Seafood'
        BEGIN
            IF @dayOfWeekRequired not in (5, 6, 7) -- Thursday, Friday, Saturday
                RAISERROR ('You cannot order seafood in days other than Thursday, Friday or Saturday', 16, 1)


            IF @dayOfWeekRequired = 5
                IF @dayDifference < 3
                    RAISERROR ('You must place the order by the Monday preceding the order', 18, 1)

            IF @dayOfWeekRequired = 6
                IF @dayDifference < 4
                    RAISERROR ('You must place the order by the Monday preceding the order', 18, 1)
            IF @dayOfWeekRequired = 7
                IF @dayDifference < 5
                    RAISERROR ('You must place the order by the Monday preceding the order', 18, 1)
        END

    IF @@ERROR != 0
        RETURN -1

    DECLARE @OccasionalDiscount float = (SELECT Discount
                                         FROM DishInMenus DIM
                                                  JOIN OccasionalDiscounts OD
                                                       on DIM.OccasionalDiscountID = OD.OccasionalDiscountID
                                         WHERE DishInMenuID = @DishInMenuID
                                           AND @RequiredDate BETWEEN OD.StartDate AND OD.EndDate)
    IF @OccasionalDiscount is null
        set @OccasionalDiscount = 0

    DECLARE @UnitPrice money =((SELECT UnitPrice
                                FROM DishInMenus
                                         JOIN Dishes D on DishInMenus.DishID = D.DishID
                                WHERE DishInMenuID = @DishInMenuID
                               ) * 1 - (@OccasionalDiscount))

    INSERT INTO OrderDetails(OrderID, DishInMenuID, Quantity, UnitPrice)
    VALUES (@OrderID, @DishInMenuID, @Quantity, @UnitPrice)

END
go
```

7.3. Procedura AcceptOrder

Akceptuje zamówienie. Sprawdza czy klient wykorzystał zniżkę do zamówienia. Na koniec wywołuje procedurę, która sprawdza czy należy się zniżka klientowi.

```
CREATE PROCEDURE AcceptOrder
@OrderID int,
@EmployeeID int
AS
BEGIN
   DECLARE @IsEmployeeID int = (SELECT EmployeeID FROM Employees WHERE EmployeeID = @EmployeeID)
   IF @IsEmployeeID is null
       RAISERROR('Provided EmployeeID not present in database', 18, 1)


   DECLARE @IsOrderID int = (SELECT OrderID FROM Orders WHERE OrderID = @OrderID)
   If @IsOrderID is null
       RAISERROR ('Provided OrderID not present in database', 16, 1)

   DECLARE @Cancelled char = (SELECT IsCancelled FROM Orders WHERE OrderID=@OrderID)
   IF @Cancelled = 'y'
       RAISERROR('This OrderID was cancelled.', 16, 1)

   DECLARE @AlreadyAccepted char = (SELECT IsAccepted FROM Orders WHERE OrderID=@OrderID)
   IF @AlreadyAccepted ='y'
       RAISERROR('This OrderID was already accepted', 16, 1)

   IF @@ERROR != 0
       RETURN -1

   DECLARE @DiscountID int = (SELECT DiscountID FROM Orders WHERE OrderID=@OrderID)

   DECLARE @ClientID int = (SELECT ClientID FROM Orders WHERE OrderID=@OrderID)
   DECLARE @isIndividualClient int = (SELECT CLientID FROM IndividualClients WHERE ClientID=@ClientID)
   IF @isIndividualClient is not null
       BEGIN
           DECLARE @OneTimeDiscount int = (SELECT D.DiscountID
                                           FROM OneTimeDiscounts OTD
                                                    JOIN Discounts D on OTD.DiscountID = D.DiscountID
                                           WHERE OneTimeDiscounts.DiscountID = @DiscountID
                                             AND Used = 'N')

           IF @OneTimeDiscount IS NOT NULL
               BEGIN
                   UPDATE OneTimeDiscounts
                   SET Used = 'y'
                   WHERE DiscountID = @DiscountID
               end
       END

   UPDATE Orders
   SET IsAccepted = 'y', EmployeeID = @EmployeeID
   WHERE OrderID = @OrderID

   IF @isIndividualClient is not null
       BEGIN
```

```
        DECLARE @MoneySpentBeforeOrder money = (SELECT SUM(dbo.OrderOverallPrice(OrderID))
                                                FROM Orders
                                                WHERE ClientID = @ClientID AND IsCancelled = 'n'
                                                   AND  IsAccepted = 'y' AND OrderID != @OrderID
        )
        IF @MoneySpentBeforeOrder is null
            set @MoneySpentBeforeOrder = 0

        EXEC AddDiscountsIfDeserved @ClientID, @MoneySpentBeforeOrder
    END
end
go
```

7.3. Procedura AddDiscountsIfDeserved

Procedura sprawdza czy klientowi należą się jakieś zniżki. Jeśli tak - dodaje je.

```
CREATE PROCEDURE AddDiscountsIfDeserved (@ClientID int, @TotalMoneyBeforeOrder money)
AS
BEGIN
DECLARE
   @nOfValidOrdersToReusable int = (SELECT COUNT(dbo.OrderOverallPrice(OrderID)) nOfValidResuable
                                    FROM Orders
                                    WHERE ClientID = @ClientID
                                      AND dbo.OrderOverallPrice(OrderID) > 30)
   IF @nOfValidOrdersToReusable != 0 AND @nOfValidOrdersToReusable = 10
       BEGIN
           INSERT INTO Discounts(ClientID, Discount)
           VALUES (@ClientID, 0.03)
       END


  DECLARE @moneySpent int = (SELECT dbo.TotalMoneySpentForClient(@ClientID))


   DECLARE @nOfDiscountsDeserved int = (SELECT (@moneySpent / 1000 - @TotalMoneyBeforeOrder / 1000))
   IF @nOfDiscountsDeserved is null
       set @nOfDiscountsDeserved = 0


   DECLARE @cnt INT = 0;

   WHILE @cnt < @nOfDiscountsDeserved
       BEGIN
           DECLARE @EndDate datetime = (SELECT DATEADD(day, 7, GETDATE()))
           DECLARE @StartDate datetime = GETDATE()
           EXEC dbo.AddOneTimeDiscount @ClientID, @StartDate, @EndDate, 0.05
           SET @cnt = @cnt + 1;
       END
```

```
END
```

7.4. Procedura AddIndividualClient

Procedura dodaje indywidualnego klienta. Dodaje również instancję do tabeli Clients, żeby zapewnić spójność

```
CREATE PROCEDURE AddIndividualClient
   @CompanyID int = NULL,
   @FirstName varchar(35) = NULL,
   @LastName varchar(35) = NULL,
   @Email varchar(80) = NULL,
   @Phone varchar(15) = NULL
AS
SET NOCOUNT ON
BEGIN
   INSERT INTO Clients(Email, Phone)
   VALUES (@Email, @Phone)

   IF ((@FirstName is null AND @LastName is not null) OR (@FirstName is not null AND @LastName is null))
       RAISERROR('Please provide full name.', 18, 1)


   DECLARE @ClientID int = (SELECT MAX(ClientID) FROM Clients)

   IF @CompanyID is not null
       BEGIN
           DECLARE @IsCompanyID int = (SELECT ClientID FROM Companies WHERE ClientID=@CompanyID)
           IF (@IsCompanyID is null)
               RAISERROR('The CompanyID is not in database', 18, 1)
       END

   IF @@ERROR != 0
       RETURN -1

   INSERT INTO IndividualClients(ClientID, CompanyClientID, FirstName, LastName)
   VALUES (@ClientID, @CompanyID, @FirstName, @LastName)
END
```

7.5. Procedura AddOneTimeDiscount

Dodaje jednorazową zniżkę. Zapewnia również spójność z tabelą Discounts.

```
CREATE PROCEDURE AddOneTimeDiscount
   @ClientID int,
   @StartDate datetime,
   @EndDate datetime = NULL,
   @Discount float
```

```
AS
   BEGIN

       SET NOCOUNT ON;

       INSERT INTO Discounts(ClientID, Discount)
       VALUES (@ClientID, @Discount)

       DECLARE @DiscountID int = (SELECT MAX(DiscountID) FROM Discounts)

       INSERT INTO OneTimeDiscounts(DiscountID, StartDate, EndDate, Used)
       VALUES (@DiscountID, @StartDate, @EndDate, 'n')

   END
```

7.6. Procedura GetFreeTables

Zwraca tabelę wolnych stolików w podanym przedziale czasu.

```
CREATE PROCEDURE GetFreeTables
(
    @start datetime,
    @end datetime
)
AS
    BEGIN
        SELECT *
        FROM Tables T
        WHERE T.TableID NOT IN (SELECT TableID
                        FROM TableReservations
                        WHERE IsCancelled = 'n' AND
                          (OutTime IS NOT NULL AND (
                            (@start BETWEEN InTime AND OutTime)
                           OR (@end BETWEEN InTime AND OutTime)
                           OR (InTime BETWEEN @start AND @end)
                           OR (OutTime BETWEEN @start AND @end)))
                           OR (OutTime IS NULL AND @start <= InTime))
    END
GO
```

7.7. Procedura GenerateReport

Generuje raport dotyczący wydatków klienta w zadanym przedziale czasu.

```
CREATE PROCEDURE GenerateReport
(
```

```
   @clientid int,
   @start datetime,
   @end datetime
)
AS
   SET NOCOUNT ON
   BEGIN
       SELECT o.OrderID, o.OrderDate, o.RequiredDate, (SELECT Discount FROM Discounts WHERE
DiscountID = o.DiscountID) ClientDiscount, ord.Quantity, ord.UnitPrice, ocd.Discount
RestaurantDiscount
       FROM Orders o JOIN OrderDetails ord ON o.OrderID = ord.OrderID
           JOIN DishInMenus dim ON ord.DishInMenuID = dim.DishInMenuID
           JOIN OccasionalDiscounts ocd ON dim.OccasionalDiscountID = ocd.OccasionalDiscountID
       WHERE o.ClientID = @clientid AND o.OrderDate BETWEEN @start AND @end
   END
GO
```

7.8. Procedura GenerateTablesReportOverCertainTimePeriod

Wyświetla historię rezerwacji stolików przez klienta w zadanym przedziale czasu.

```
CREATE PROCEDURE GenerateTablesReportOverCertainTimePeriod
(
    @start datetime,
    @end datetime,
    @clientid int
)
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @date datetime = GETDATE()

    IF @end > @date
        BEGIN
            SET @end = @date
        END

    SELECT tr.TableReservationID, t.TableID, t.NumberOfChairs
    FROM Orders o, TableReservations tr, Tables t
    WHERE o.ClientID = @clientid
      AND tr.OrderID = o.OrderID
      AND t.TableID = tr.TableID
      AND tr.ReservationDate BETWEEN @start AND @end

    END
GO
```

7.9. Procedura CancelledReservationsByIndividualClientsOverCertainTime

Wyświetla informacje na temat klientów, którzy anulowali swoje rezerwacje w zadanym przedziale czasu.

```
CREATE PROCEDURE CancelledReservationsByIndividualClientsOverCertainTime
(
    @start datetime,
    @end datetime
)
AS
    SET NOCOUNT ON
    BEGIN
        SELECT ic.ClientID, ic.FirstName, ic.LastName, tr.ReservationDate, tr.TableID
        FROM IndividualClients ic, Orders o, TableReservations tr
        WHERE tr.IsCancelled = 'Y'
        AND tr.OrderID = o.OrderID
        AND o.ClientID = ic.ClientID
        AND tr.ReservationDate BETWEEN @start AND @end
    END
```

7.10. Procedura MakeReservation

Dokonuje rezerwacji dla klienta z uwzględnieniem tego czy jest ona na teraz i na miejscu czy wybiega w przyszłość.

```
CREATE PROCEDURE MakeReservation
(
   @ClientID int,
   @OrderID int,
   @OrderDate datetime,
   @TableID int,
   @ReservationDate datetime,
   @OutTime datetime = NULL
)
AS
   SET NOCOUNT ON
   BEGIN

       DECLARE @OrderPrice money
       EXEC @OrderPrice = OrderOverallPrice @OrderID

       DECLARE @AmountOfOrders int = (SELECT COUNT(OrderID) FROM Orders WHERE ClientID = @ClientID AND IsCancelled = 'n')

       IF (@OrderPrice >= 50.00 AND @AmountOfOrders >= 5) OR @OrderDate = @ReservationDate
           BEGIN

               IF (@OrderPrice >= 50.00 AND @AmountOfOrders >= 5) AND @OrderDate <> @ReservationDate
                   BEGIN
                       SET @OutTime = ISNULL(@OutTime, DATEADD(hour, 1, @ReservationDate))
                   END

               DECLARE @IsFree char
               EXEC @IsFree = dbo.IsSpecificTableAvailableAtSpecificTime @TableID, @ReservationDate, @OutTime

               IF @IsFree = 'y'
                   BEGIN
                       INSERT INTO TableReservations(OrderID, TableID, ReservationDate, BookingTime, InTime, OutTime, IsCancelled)
                       VALUES(@OrderID, @TableID, @ReservationDate, GETDATE(), @ReservationDate, NULL, 'n')
```

```
                END
            ELSE
                BEGIN
                    RAISERROR('Specific table is unsuitable for reservation', 18, 1)
                    IF @@ERROR != 0
                        RETURN -1

                END
        END
    ELSE
        BEGIN
            RAISERROR('Client does not necessary conditions for reservation', 18, 1)
            IF @@ERROR != 0
                RETURN -1
        END
END
GO
```

7.11. Procedura CancelReservationDueToClientEatingTakeaway

Anuluje rezerwację w sytuacji, gdy klient, który w czasie rzeczywistym dokonał zamówienia wychodzi i decyduje się zabrać je na wynos pozostawiając przy tym wolne miejsce.

```
CREATE PROCEDURE CancelReservationDueToClientEatingTakeAway
(
    @orderid int
)
AS
    SET NOCOUNT ON
    BEGIN
        UPDATE TableReservations
        SET IsCancelled = 'Y', OutTime = GETDATE()
        WHERE OrderID = @orderid
    END
```

7.12. Procedura CancelledReservationsByCompaniesOverCertainTime

Zwraca informacje na temat zamówień, które zostały anulowane przez firmy w danym przedziale czasu.

```
CREATE PROCEDURE CancelledReservationsByCompaniesOverCertainTime
(
    @start datetime,
    @end datetime
)
AS
    SET NOCOUNT ON
```

```
    BEGIN
        SELECT c.ClientID, c.CompanyName, c.ContactName, tr.ReservationDate, tr.TableID
        FROM Companies c, Orders o, TableReservations tr
        WHERE tr.IsCancelled = 'Y'
        AND tr.OrderID = o.OrderID
        AND o.ClientID = c.ClientID
        AND tr.ReservationDate BETWEEN @start AND @end
   END
```

7.13. Procedura ShowOccasionalDiscountsAtCertainTime

Zwraca informacje o zniżkach od restauracji w określonym dniu.

```
CREATE PROCEDURE ShowOccasionalDiscountsAtCertainTime
(
    @date datetime
)
AS
    SET NOCOUNT ON
    BEGIN
        SELECT * FROM OccasionalDiscounts WHERE @date BETWEEN StartDate AND EndDate
    END
```

7.14. Procedura ShowDiscountsAvailableToClientAtGivenTime

Zwraca informacje o rabatach przysługujących klientom indywidualnym (bez okazyjnych rabatów udzielanych przez resturację)..

```
CREATE PROCEDURE ShowDiscountsAvailableToClientAtGivenTime
(
    @clientid int,
    @date datetime
)
AS
    SET NOCOUNT ON
    BEGIN
        SELECT DiscountID, Discount
        FROM Discounts
        WHERE DiscountID NOT IN (SELECT DiscountID
                                 FROM OneTimeDiscounts
                                 WHERE StartDate > @date OR EndDate < @date OR Used = 'Y')
            AND ClientID = @clientid
 END
```

7.15.Procedura ShowInvoiceComponents

Pokazuje informacje zawarte w wystawionej już fakturze.

```
CREATE PROCEDURE ShowInvoiceComponents
(
   @invoiceid int
)
AS
   SET NOCOUNT ON
   BEGIN
       SELECT o.OrderID, o.OrderDate, o.RequiredDate,
(SELECT Discount FROM Discounts WHERE DiscountID = o.DiscountID) ClientDiscount, ord.Quantity, ord.UnitPrice,
ocd.Discount RestaurantDiscount
       FROM Orders o JOIN OrderDetails ord ON o.OrderID = ord.OrderID
           JOIN DishInMenus dim ON ord.DishInMenuID = dim.DishInMenuID
           JOIN OccasionalDiscounts ocd ON dim.OccasionalDiscountID = ocd.OccasionalDiscountID
       WHERE o.InvoiceID = @invoiceid
   END
```

7.16. Procedura GenerateInvoice

Generuje fakturę i wyświetla stosowne informacje.

```
CREATE PROCEDURE GenerateInvoice
(
   @clientid int,
   @clfirstname varchar(35) = NULL,
   @cllastname varchar(35) = NULL,
   @companyname varchar(35) = NULL,
   @address varchar(50),
   @empfirstname varchar(35),
   @emplastname varchar(35),
   @start datetime,
   @end datetime
)
AS
   SET NOCOUNT ON
   BEGIN
       DECLARE @clientname varchar(70)

       IF @companyname IS NULL AND (@clfirstname IS NULL OR @cllastname IS NULL)
           BEGIN
                RAISERROR('Not enough names were given', 18, 1)
           END

       IF @companyname IS NULL
           BEGIN
               SET @clientname = CONCAT(@clfirstname, ' ', @cllastname)
           END
       ELSE
           BEGIN
               SET @clientname = @companyname
           END


       DECLARE @invoiceid int = (SELECT MAX(InvoiceID) FROM Invoices) + 1

       DECLARE @invoicedate datetime = GETDATE()

       IF @end > @invoicedate
           BEGIN
               SET @end = @invoicedate
           END
```

```
        INSERT INTO Invoices(ClientName, EmployeeName, InvoiceDate, Address)
        VALUES(@clientname, CONCAT(@empfirstname,' ', @emplastname), @invoicedate, @address)

        UPDATE Orders
        SET InvoiceID = @invoiceid
        WHERE OrderID IN (SELECT OrderID
                          FROM Orders
                          WHERE ClientID = @clientid
                            AND @start <= OrderDate AND @end >= OrderDate
                            AND InvoiceID IS NULL)

        EXEC GenerateReport @clientid, @start, @end

    END
GO
```

7.17. Procedura UpdateInOutTimeReservation

Procedura aktualizuje rezerwacje stolika.

```
CREATE PROCEDURE UpdateInOutTimeReservation
    @intime datetime,
    @outtime datetime,
    @orderid int
AS
SET NOCOUNT ON
BEGIN
    UPDATE TableReservations
    SET InTime = @intime, Outtime = @outtime
    WHERE OrderID = @orderid
END
GO
```

7.18. Procedura DishesAt

Mówi nam jakie są dania w podanym czasie i jakie są uwzględniając dostępne zniżki restauracyjne.

```
CREATE PROCEDURE DishesAt(@DateTime datetime)
AS

SELECT MenuID,
      DishInMenuID,
      DishName,
      CategoryName,
      UnitPrice,
      ISNULL(OD.Discount, 0) Discount,
      UnitPrice * (1 - ISNULL(OD.Discount, 0)) PriceAfterDiscount
FROM DishInMenus DIM
```

```
        JOIN Dishes D ON D.DishID = DIM.DishID
        LEFT OUTER JOIN OccasionalDiscounts OD ON OD.OccasionalDiscountID = DIM.OccasionalDiscountID AND
                                                  @DateTime BETWEEN  OD.StartDate AND OD.EndDate
WHERE EXISTS(
             SELECT MenuID
             FROM Menus
             WHERE @DateTime BETWEEN StartDate AND EndDate)
GO
```

7.19. Procedura DeleteClient

Procedura DeleteClient usuwa klienta.

```
CREATE PROCEDURE DeleteClient
@ClientID int
AS
   BEGIN
       IF (@ClientID is null)
           RAISERROR('The CompanyID is not in database', 18, 1)

       IF @@ERROR != 0
           RETURN -1

       DECLARE @IndividualClientID int = (SELECT ClientID FROM IndividualClients WHERE ClientID=@ClientID)

       IF (@IndividualClientID is not null)
           DELETE FROM IndividualClients
           WHERE ClientID = @IndividualClientID

       DECLARE @CompanyID int = (SELECT ClientID FROM Companies WHERE ClientID=@ClientID)

       IF (@CompanyID is not null)
           DELETE FROM Companies
           WHERE ClientID = @CompanyID

       DELETE FROM Clients
       WHERE ClientID = @ClientID
   END
go
```

7.20. Procedura DeleteDishFromOrder

Procedura usuwa danie z zamówienia.

```
CREATE PROCEDURE DeleteDishFromOrder
@OrderID int,
@DishInMenuID int
AS
   DECLARE @IsOrderID int = (SELECT OrderID FROM Orders WHERE OrderID=@OrderID)

   IF @IsOrderID IS NULL
       RAISERROR('The OrderID is not present in database', 18, 1)
   IF @@ERROR != 0
```

```
        RETURN -1

    DECLARE @IsDishInMenuID int = (SELECT DishInMenuID FROM OrderDetails WHERE DishInMenuID=@DishInMenuID
                                                                            AND OrderID=@OrderID)

    IF @IsDishInMenuID IS NULL
        RAISERROR('The DishMenuID is not present in Order', 18, 1)
    IF @@ERROR != 0
        RETURN -1

    DECLARE @cancelled char = (SELECT IsCancelled FROM Orders WHERE OrderID=@OrderID)
    DECLARE @accepted char = (SELECT IsAccepted FROM Orders WHERE OrderID=@OrderID)


    IF @cancelled IS NULL
        RAISERROR('The order is cancelled', 18, 1)
    IF @accepted IS NULL
        RAISERROR('The order was already accepted.', 18, 1)
    IF @@ERROR != 0
        RETURN -1


    DELETE FROM OrderDetails
    WHERE OrderID=@OrderID AND DishInMenuID=@DishInMenuID
go
```

7.21. Procedura InspectOrder

Procedura InspectOrder zwraca wgląd w zamówienie.

```
CREATE PROCEDURE InspectOrder(@OrderID int)
AS
    BEGIN

    DECLARE @DiscountID int = (SELECT DiscountID FROM Orders WHERE OrderID=@OrderID)
    DECLARE @Discount float

    IF (@DiscountID IS NOT NULL)
        SET @Discount = (SELECT Discount FROM Discounts WHERE DiscountID=@DiscountID)
    ELSE
        SET @Discount = 0


    SELECT O.OrderID, O.ClientID, C.Phone, EmployeeID, OrderDate,
           RequiredDate, IsAccepted,IsCancelled, (@Discount) ClientOrderDiscount,
           DIM.DishInMenuID, D.DishName, Quantity, D.UnitPrice, Occ.Discount DiscountForDish,
           (dbo.OrderOverallPrice(@OrderID)) OrderOverallPrice
    FROM Orders O
    JOIN OrderDetails OD ON O.OrderID = OD.OrderID
    JOIN DishInMenus DIM ON OD.DishInMenuID = DIM.DishInMenuID
    JOIN Dishes D ON DIM.DishID = D.DishID
    JOIN Clients C ON O.ClientID = C.ClientID
    LEFT OUTER JOIN OccasionalDiscounts Occ ON DIM.OccasionalDiscountID =
```

```
Occ.OccasionalDiscountID
   WHERE OD.OrderID = @OrderID
END
```

7.22. Procedura AddDishToMenu

Procedura dodaje danie do menu.

```
CREATE PROCEDURE AddDishToMenu
(
@menuID int,
@DishID int
)
AS
BEGIN
    SET NOCOUNT ON
    IF @DishID not in (SELECT DishID FROM Dishes WHERE DishID = @DishID)
        RAISERROR('Dish not in database', 16, 1)

    IF @@ERROR != 0
        RETURN - 1

    If (@DishID NOT IN (SELECT DishID FROM DishInMenus WHERE MenuID = @menuID))
        BEGIN
               INSERT INTO DishInMenus(DishID, MenuID, OccasionalDiscountID)
               VALUES (@DishID, @menuID, NULL)
        END

END
```

7.23. Procedura AcceptMenu

Procedura pozwala zaakceptować menu, jeśli spełnione są warunki zadania.

```
CREATE PROCEDURE AcceptMenu
(
@LastMenuID int,
@NewMenuID int
)
AS
BEGIN
    DECLARE @lastmenudishesnumber int = (SELECT COUNT(*) FROM DishInMenus WHERE MenuID =@LastMenuID)
    DECLARE @disheschanged int = dbo.NumberOfChangedDishes(@NewmenuID, @LastMenuID)

    IF @disheschanged > @lastmenudishesnumber/2
        BEGIN
               UPDATE Menus SET IsReady = 'Y' WHERE @NewMenuID = MenuID
        END
END
```

7.24. Procedura InspectMenu

Ukazuje szczegóły wybranego menu; odnośnie dań, zniżek okazyjnych od restauracji oraz ich czasu trwania.

```
CREATE PROCEDURE InspectMenu
@MenuID int
AS
BEGIN
   SELECT DIM.MenuID, M.StartDate, M.EndDate, DishName, CategoryName, UnitPrice, ISNULL(Discount, 0)
Discount, UnitPrice*(1-ISNULL(Discount, 0)) PriceAfterDiscount, OD.StartDate, OD.EndDate
   FROM DishInMenus DIM
   LEFT OUTER JOIN Dishes D on DIM.DishID = D.DishID
   LEFT OUTER JOIN Menus M on DIM.MenuID = M.MenuID
   LEFT OUTER JOIN OccasionalDiscounts OD on DIM.OccasionalDiscountID = OD.OccasionalDiscountID
   WHERE DIM.MenuID = @MenuID
END
```

7.25. Procedura AddOcasionalDiscountToDishInMenu

Procedura przypisuje okresową zniżkę do dania w menu.

```
ALTER PROCEDURE AddOccasionalDiscountToDishInMenu
@DishInMenusID int,
@Discount float,
@Startdate datetime,
@Enddate datetime

AS
SET NOCOUNT ON
BEGIN

    INSERT INTO OccasionalDiscounts(StartDate, EndDate, Discount)
    VALUES (@Startdate, @Enddate, @Discount)

    DECLARE @new int = (SELECT TOP 1 OccasionalDiscountID
                        FROM OccasionalDiscounts
                        ORDER BY OccasionalDiscountID)

    UPDATE DishInMenus
       SET OccasionalDiscountID = @new WHERE DishInMenuID = @DishInMenusID


END
```

8.Triggery

8.1 Trigger CancelOrderReservation

Trigger anuluje rezerwację stolika jeśli anulowana została rezerwacja zamówienia.

```
CREATE TRIGGER CancelOrderReservation
ON Orders FOR UPDATE
AS
   BEGIN
       IF UPDATE(isCancelled)
           BEGIN
           IF (SELECT isCancelled FROM inserted) = 'y'
               BEGIN
                   DECLARE @OrderID int = (SELECT OrderID FROM inserted)
                   UPDATE TableReservations SET isCancelled = 'y'
                   WHERE OrderID = @OrderID
               END
           END
   END
```

8.2 Trigger CancelOrderInvoice

Trigger anuluje wymaganą fakturę, jeśli anulowano zamówienie.

```
CREATE TRIGGER CancelOrderAndInvoice
   ON Orders
   FOR UPDATE
   AS
BEGIN
   IF UPDATE(isCancelled)
       BEGIN
       IF (SELECT isCancelled FROM inserted) = 'y'
              BEGIN
                     DECLARE @InvoiceID AS int
                     SET @InvoiceId = (SELECT InvoiceID FROM inserted)
                     UPDATE Orders
                     SET InvoiceRequired = 'n'
                     WHERE InvoiceID = @InvoiceID
              END
       END
END
```

9.Indeksy

Indeksy nieklastrowe zostały utworzone dla każdego klucza obcego w każdej tabeli. Rozwiązanie takie jest optymalne, gdyż nie przewidujemy częstego usuwania danych z bazy - zamiast tego używamy pozycji IsCancelled.

9.1. Indeks CompanyClientID w tabeli IndividualClients

```
CREATE NONCLUSTERED INDEX [CompanyClientIDIndex] ON [dbo].[IndividualClients] (CompanyClientID);
```

9.2. Indeks ClientD w tabeli Discounts

```
CREATE NONCLUSTERED INDEX [ClientIDIndex] ON [dbo].[Discounts] (ClientID);
```

9.3. Indeks ClientID w tabeli Orders

```
CREATE NONCLUSTERED INDEX [ClientIDIndex] ON [dbo].[Orders] (ClientID);
```

9.4. Indeks EmployeeID w tabeli Orders

```
CREATE NONCLUSTERED INDEX [EmployeeIDIndex] ON [dbo].[Orders] (EmployeeID);
```

9.5. Indeks OrderID w tabeli TableReservation

```
CREATE NONCLUSTERED INDEX [OrderIDIndex] ON [dbo].[TableReservations] (OrderID)
```

9.6. Indeks TableID w tabeli TableReservation

```
CREATE NONCLUSTERED INDEX [TableIDIndex] ON [dbo].[TableReservations] (TableID);
```

9.7. Indeks DishInMenutID w tabeli OrderDetails

```
CREATE NONCLUSTERED INDEX [DishInMenuIDIndex] ON [dbo].[OrderDetails] (DishInMenuID);
```

9.8. Indeks OrderID w tabeli OrderDetails

```
CREATE NONCLUSTERED INDEX [OrderIDIndex] ON [dbo].[OrderDetails] (OrderID);
```

9.9. Indeks DishID w tabeli DishInMenus

```
CREATE NONCLUSTERED INDEX [DishIDIndex] ON [dbo].[DishInMenus] (DishID);
```

9.10. Indeks MenuID w tabeli DishInMenus

```
CREATE NONCLUSTERED INDEX [MenuIDIndex] ON [dbo].[DishInMenus] (MenuID);
```