Podstawy Baz Danych

Projekt: Wspomaganie działalności świadczącej usługi gastronomiczne

Anna Gut
Paweł Świder

Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. System winien być możliwy do równoległego użytkowania przez kilka tego typu firm.

Funkcje realizowane przez system

System będzie realizował następujące funkcje:

- złożenie zamówienia na wynos (klient)
- złożenie zamówienia z rezerwacją stolika (klient)
- zatwierdzenie zamówienia złożonego przez klienta(pracownik)
- sprawdzenie czy zamówienie spełnia warunek możliwy do rezerwacji stolika (pracownik)
- zapisanie rezerwacji w systemie, przyznanie rezerwacji numeru stolika, funkcja wyświetla informacje czy dana rezerwacja spełnia warunki (minimalna kwota zamówienia) (pracownik)
- wyświetlenie listy stolików w restauracji oraz ich dostępność i liczba miejsc przy stoliku (manager)
- zmiana stanu dostępności stolika oraz liczby miejsc przy stoliku (manager)
- sprawdzanie czy w danym czasie jest dostępny stolik dla danej liczby osób (pracownik)
- dodanie klienta do listy stałych klientów (pracownik/manager) (jeśli taka lista będzie)
- wyświetlenie stałych klientów (pracownik/manager)
- sprawdzenie czy klient jest stałym klientem (pracownik)
- wyświetlenie pozycji w menu (pracownik, klient)
- wyświetlenie aktywnych pozycji w menu, które zawierają podany składnik (manager)
- dodanie pozycji w menu, sprawdza czy pozycja ta nie była wcześniej niż miesiąc temu (manager)
- usunięcie pozycji z menu (manager)
- sprawdzanie spełnienia warunku o zmienianiu co najmniej połowy pozycji w menu raz na dwa tygodnie (manager)
- dodanie/usunięcie rabatu do zamówienia, sprawdza czy klient spełnia warunki danego rabatu (pracownik)
- wyświetlenie rabatów w danej restauracji (klient, pracownik)
- wyświetlenie, ile dany klient złożył zamówień i na jaką łączną kwotę (pracownik)
- Wyświetlenie, ile w ciągu ostatniego miesiąca zostało złożonych zamówień oraz na jaką kwotę przez danego klienta (pracownik)
- Wyświetlenie, ile w ciągu ostatniego kwartału zostało złożonych zamówień oraz na jaką kwotę przez danego klienta(pracownik)
- zmiana rabatów i warunków ich otrzymania (manager)
- dodanie/usunięcie jednorazowych rabatów dla danego klienta (dodaje rabat automatycznie, gdy klient spełni warunki i usuwa rabat po jego wykorzystaniu lub wygaśnięciu czasu)
- wyświetlanie przyszłych zamówień zawierających daną pozycję z menu (manager)
- naliczenie rabatu do zamówienia (pracownik, za zgodą klienta)
- wyświetlanie informacji o zamówieniu (co i ile zamówiono, koszt jednostkowy produktu, naliczone rabaty, całkowity koszt zamówienia, koszt zamówienia po odjęciu rabatu, data zamówienia (klient, pracownik, manager)
- wyświetlenie danych klienta restauracji, który złożył zamówienie (pracownik/manager) (te dwie funkcje do faktur)
- wyświetlenie obecnych rezerwacji (osobne wersje dla klientów i managera restauracji)
- wyświetlenie podstawowych danych o zamówieniach (kto, kiedy, kwota i rabaty) (manager)
- wyświetlenie informacji z jakich rabatów skorzystali klienci (manager)
- wyświetlenie jakie stoliki były rezerwowane i ile razy (manager)
- wyświetlenie informacji o tym jakie danie były zamawiane i jak często (manager)

Opis bazy danych

1) Restauracje

Baza danych zawiera informacje dotyczące funkcjonowania niezależnych od siebie firm świadczących usługi gastronomiczne. Dane opisujące restauracje oraz sposób ich funkcjonowania i warunki dotyczące przyznawania rabatów znajdują się w tabelach *Company* oraz *DiscountsData*, posiadających klucz główny *CompanyID*, będący unikalnym numerem identyfikującym firmę.

2) Zamówienia i rezerwacje

Informacje dotyczące zamówień oraz rezerwacji znajdują się w tabelach *Orders, OrderDetails, Reservations, ReservationDetails*. Do tabeli *Orders* trafiają wszystkie zamówienia złożone zarówno na miejscu, jak i te przesłane przez formularz internetowy. Rezerwacje mogą być złożone zarówno przez klientów indywidualnych, jak i przez firmy.

Przyszłe zamówienia uzyskują miano rezerwacji (trafiają do tabeli *Reservations*) tylko wtedy, gdy spełnione zostaną następujące warunki:

- w czasie planowanej rezerwacji jest wystarczająca liczba dostępnych miejsc oraz stolików

- zamówione dania znajdują się w menu, które będzie obowiązywało w restauracji w dniu planowanej rezerwacji

- łączny koszt zamówionych dań jest równy lub większy od wymaganego przez daną restaurację

- dania zawierające owoce morza zostały zamówione z odpowiednim wyprzedzeniem oraz na odpowiedni dzień tygodnia

Rezerwacja może być w każdym momencie anulowana przez managera restauracji z powodu wyczerpania się konkretnych składników wchodzących w skład zamówionych dań lub z powodu zmianiających się obostrzeń związanych z panująca pandemią

3) Menu i stoliki

W bazie danych każda modyfikacja menu restauracji (usunięcie lub dodanie potrawy) powoduje utworzenie się nowego menu uwzględniającego wprowadzone zmiany. Wszystkie informacje dotyczące menu, ich zawartości oraz okresu obowiązywania w danej restauracji zawierają tabele *Menu* oraz *MenuDetails*.

Menu musi być tworzone z uwzględnieniem poniższych warunków:

-ponad połowa pozycji w menu musi być zmieniana co najmniej raz na dwa tygodnie

- pozycja zdjęta z menu, może pojawić się w nim ponownie dopiero po miesiącu

Analogicznie każda nowa zmiana w dostępności stolików oraz ilości miejsc, które można przy nich zająć powoduje stworzenie nowego układu w bazie danych. Informacje o konkretnych układach zawierają tabele *Intervals* oraz *IntervalDetails.*

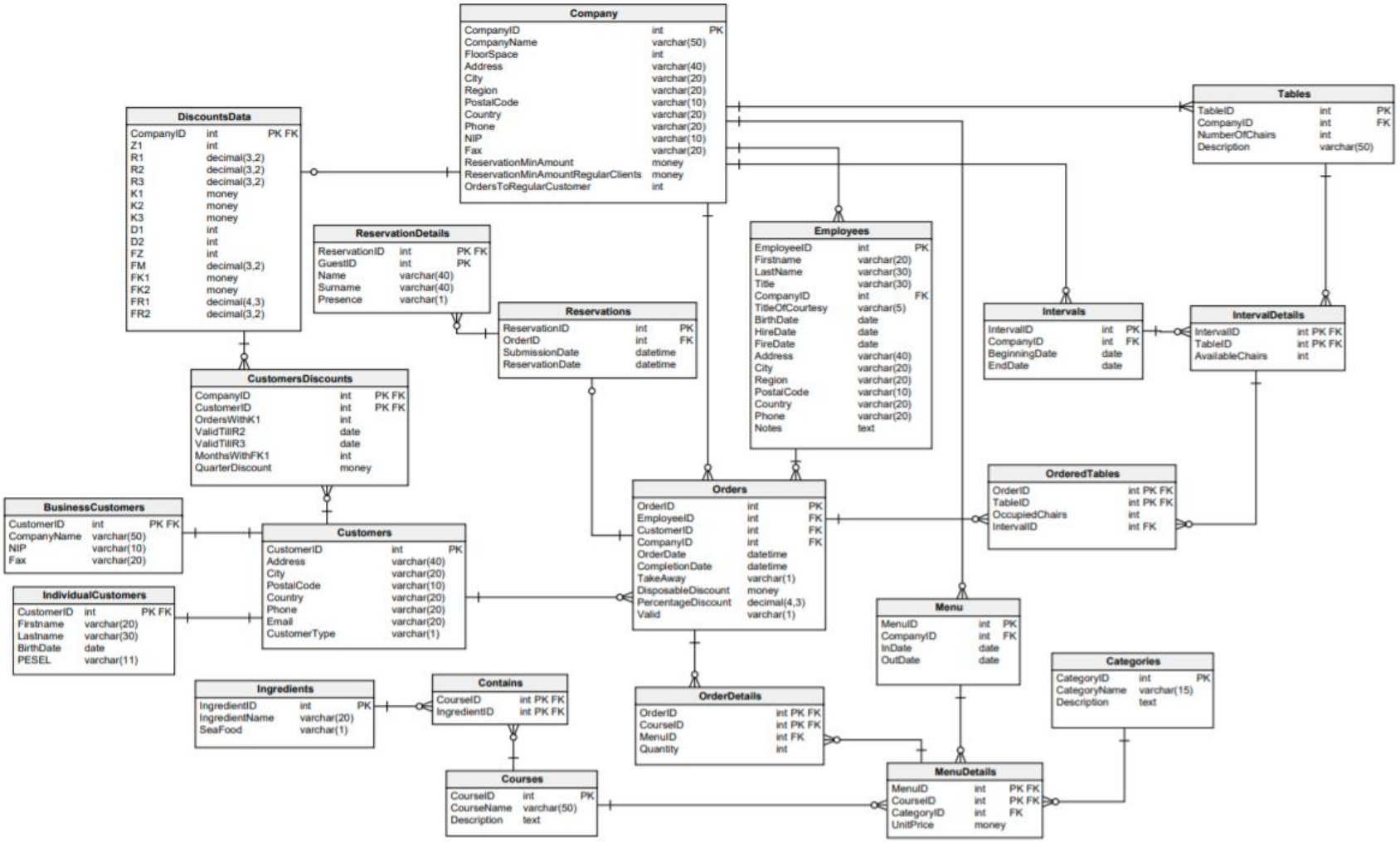
4) Rabaty

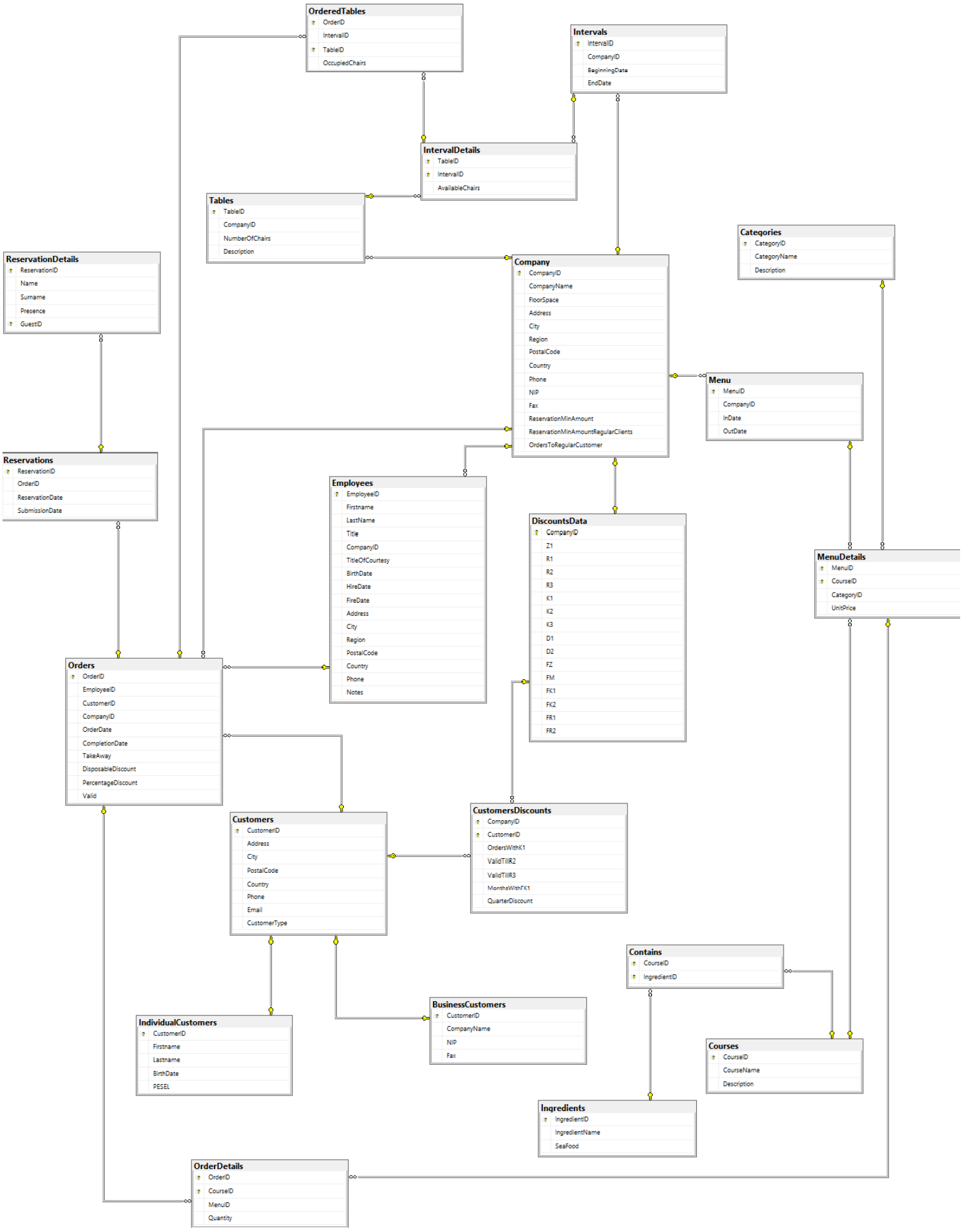
Informacje dotyczące sposobu funkcjonowania rabatów w konkretnych restauracjach zawiera tabela *DiscountsData.* Informacje dotyczące zniżek, które posiadają stali klienci w restauracjach znajdują się w tabeli *CustomersDiscounts.*

5) Stali klienci

Informacje dotyczące klientów, którzy posiadają kartę stałego klienta w którejś z restauracji korzystającej z systemu zawiera tabela *Customers.* Szczegółowe dane klientów zawierają się w tabelach *BusinessCustomers* oraz *IndividualCustomers,* w zależności od typu klienta.

Schemat bazy danych




OrderedTables
OrderID
IntervalID
TableID
OccupiedChairs
Intervals
IntervalID
CompanyID
BeginningDate
EndDate
IntervalDetails
TableID
IntervalID
AvailableChairs
Tables
TableID
CompanyID
NumberOfChairs
Description
Categories
CategoryID
CategoryName
Description
ReservationDetails
ReservationID
Name
Surname
Presence
GuestID
Company
CompanyID
CompanyName
FloorSpace
Address
City
Region
PostalCode
Country
Phone
NIP
Fax
ReservationMinAmount
ReservationMinAmountRegularClients
OrdersToRegularCustomer
Menu
MenuID
CompanyID
InDate
OutDate
Reservations
ReservationID
OrderID
ReservationDate
SubmissionDate
Employees
EmployeeID
Firstname
LastName
Title
CompanyID
TitleOfCourtesy
BirthDate
HireDate
FireDate
Address
City
Region
PostalCode
Country
Phone
Notes
DiscountsData
CompanyID
Z1
R1
R2
R3
K1
K2
K3
D1
D2
FZ
FM
FK1
FK2
FR1
FR2
MenuDetails
MenuID
CourseID
CategoryID
UnitPrice
Orders
OrderID
EmployeeID
CustomerID
CompanyID
OrderDate
CompletionDate
TakeAway
DisposableDiscount
PercentageDiscount
Valid
Customers
CustomerID
Address
City
PostalCode
Country
Phone
Email
CustomerType
CustomersDiscounts
CompanyID
CustomerID
OrdersWithK1
ValidTillR2
ValidTillR3
MonthsWithГК1
QuarterDiscount
Contains
CourseID
IngredientID
IndividualCustomers
CustomerID
Firstname
Lastname
BirthDate
PESEL
BusinessCustomers
CustomerID
CompanyName
NIP
Fax
Courses
CourseID
CourseName
Description
Ingredients
IngredientID
IngredientName
SeaFood
OrderDetails
OrderID
CourseID
MenuID
Quantity

Tabele

BusinessCustomers

Zawiera szczegółowe dane dla klientów wszystkich restauracji, będących klientami firmowymi.
Pola w tabeli:

- CustomerID (int) PK – unikalny numer identyfikujący klienta, nie może przyjmować wartości NULL
- CompanyName (varchar(50)) - nazwa firmy, nie może przyjmować wartości NULL
- NIP (varchar(10)) - numer identyfikacji podatkowej firmy, nie może przyjmować wartości NULL, musi być unikalny oraz zawierać dokładnie 10 cyfr
- Fax (varchar(20)) - numer FAX firmy, może przyjmować wartość NULL

Posiada indeks do pola CompanyName.

```
CREATE TABLE BusinessCustomers (
    CustomerID int  NOT NULL,
    CompanyName varchar(50)  NOT NULL,
    NIP varchar(10)  NOT NULL,
    Fax varchar(20) NULL,
    CONSTRAINT BusinessCustomers_pk PRIMARY KEY  (CustomerID),
       CONSTRAINT NIPUnique unique (NIP),
       CONSTRAINT NIPCheck check (NIP NOT LIKE '%^[0-9]%' AND DATALENGTH(NIP)=10),
       INDEX companyname_index (CompanyName)
);
```

Categories

Zawiera informacje dotyczące kategorii potraw w restauracjach .
Pola w tabeli:

- CategoryID (int) PK – unikalny numer identyfikujący kategorię, nie może przyjmować wartości NULL
- CategoryName (varchar(15)) – nazwa kategorii, nie może przyjmować wartości NULL, musi być unikalna
- Description (text) – opis kategorii, może przyjmować wartości NULL

```
CREATE TABLE Categories (
    CategoryID int  NOT NULL,
    CategoryName varchar(15)  NOT NULL,
    Description text NULL,
    CONSTRAINT Categories_pk PRIMARY KEY  (CategoryID),
       CONSTRAINT CN_Unique unique (CategoryName)
);
```

Company

Zawiera informacje dotyczące wszystkich firm korzystających z systemu.
Pola w tabeli:

- CompanyID (int) PK – unikalny numer identyfikujący firmę, nie może przyjmować wartości NULL
- CompanyName (varchar(50)) – nazwa restauracji, nie może przyjmować wartości NULL, musi być unikalna
- FloorSpace (int) – powierzchnia restauracji wyrażona w metrach kwadratowych, może przyjmować wartości NULL

- Address (varchar(40)) – ulica oraz numer lokalu, w którym znajduje się restauracja, nie może przyjmować wartości NULL
- City (varchar(20)) - miejscowość, w której znajduje się restauracja, nie może przyjmować wartości NULL
- Region (varchar(20)) – region, w którym znajduje się restauracja, może przyjmować wartości NULL
- PostalCode (varchar(10)) – kod pocztowy dla adresu, w którym znajduje się restauracja, nie może przyjmować wartości NULL
- Country (varchar(20)) – kraj, w którym znajduje się restauracja, nie może przyjmować wartości NULL, posiada domyślną wartość `Poland`
- Phone (varchar(20)) – numer telefonu restauracji, nie może przyjmować wartości NULL
- NIP (varchar(10)) – numer identyfikacji podatkowej dla restauracji, nie może przyjmować wartości NULL, musi być unikalny i składać się dokładnie z 10 cyfr
- Fax (varchar(20)) – numer Fax restauracji, może przyjmować wartości NULL
- ReservationMinAmount (money) – minimalna kwota rezerwacji dla klientów, którzy nie złożyli wymaganej ilość zamówień w restauracji, nie może przyjmować wartości NULL, nie może być mniejsza od 0, posiada domyślną wartość 200
- ReservationMinAmountRegularClients (money) - minimalna kwota rezerwacji dla klientów, którzy złożyli wymaganą ilość zamówień w restauracji, nie może przyjmować wartości NULL, nie może być mniejsza od 0, posiada domyślną wartość 50
- OrdersToRegularCustomer (int) – minimalna liczba zamówień, które musi złożyć klient w restauracji, aby obowiązywała go mniejsza wartość minimalnej kwoty rezerwacji, nie może przyjmować wartości NULL, nie może być mniejsza od 0, posiada domyślną wartość 5

Posiada indeks do pola CompanyName

```
CREATE TABLE Company (
    CompanyID int  NOT NULL,
    CompanyName varchar(50)  NOT NULL,
    FloorSpace int NULL,
    Address varchar(40)  NOT NULL,
    City varchar(20)  NOT NULL,
    Region varchar(20) NULL,
    PostalCode varchar(10)  NOT NULL,
    Country varchar(20)  NOT NULL DEFAULT 'Poland',
    Phone varchar(20)  NOT NULL,
    NIP varchar(10)  NOT NULL,
    Fax varchar(20) NULL,
    ReservationMinAmount money  NOT NULL DEFAULT 200,
    ReservationMinAmountRegularClients money  NOT NULL DEFAULT 50,
    OrdersToRegularCustomer int NOT NULL DEFAULT 5
    CONSTRAINT Company_pk PRIMARY KEY  (CompanyID),
       CONSTRAINT CompanyName_Unique unique(CompanyName),
       CONSTRAINT NIPUnique_1 unique(NIP),
       CONSTRAINT NIPCheck_1 check (NIP NOT LIKE '%^[0-9]%' AND DATALENGTH(NIP)=10),
       CONSTRAINT ReservationMinAmountCheck check (ReservationMinAmount>=0),
       CONSTRAINT ReservationMinAmountRegularClientsCheck check
       (ReservationMinAmountRegularClients>=0),
       CONSTRAINT OrdersToRegularCustomerCheck check (OrdersToRegularCustomer>=0),

       INDEX companyname_index (CompanyName)
);
```

Contains

Zawiera informacje o tym, jakie składniki wykorzystywane są dla dań w restauracjach.

Posiada pola:

- CourseID (int) PK – numer identyfikujący danie, nie może przyjmować wartości NULL
- IngredientID (int) PK – numer identyfikujący składnik, nie może przyjmować wartości NULL

Posiada indeks do pól CourseID oraz IngredientID

```
CREATE TABLE "Contains" (
    CourseID int  NOT NULL,
    IngredientID int  NOT NULL,
    CONSTRAINT Contains_pk PRIMARY KEY  (CourseID,IngredientID),
       INDEX course_index (CourseID),
       INDEX ingredient_index (IngredientID)
);
```

Courses

Zawiera informacje o daniach serwowanych w restauracjach.

Posiada pola:

- CourseID (int) PK – unikalny numer identyfikujący danie, nie może przyjmować wartości NULL
- CourseName (varchar(50)) – nazwa dania, nie może przyjmować wartości NULL
- Description – opis danie, może przyjmować wartości NULL

```
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    CourseName varchar(50)  NOT NULL,
    Description text NULL,
    CONSTRAINT Courses_pk PRIMARY KEY  (CourseID),
       INDEX coursename_index (CourseName)
);
```

Customers

Zawiera dane dotyczące stałych klientów dla restauracji.

Posiada pola:

- CustomerID (int) PK – unikalny numer identyfikujący stałego klienta, nie może przyjmować wartości NULL
- Address (varchar(40)) – ulica oraz numer lokalu, w którym znajduje się restauracja, może przyjmować wartości NULL
- City (varchar(20)) - miejscowość, w której znajduje się restauracja, może przyjmować wartości NULL
- Region (varchar(20)) – region, w którym znajduje się restauracja, może przyjmować wartości NULL
- PostalCode (varchar(10)) – kod pocztowy dla adresu, w którym znajduje się restauracja, może przyjmować wartości NULL
- Country (varchar(20)) – kraj, w którym znajduje się restauracja, może przyjmować wartości NULL
- Phone (varchar(20)) – numer telefonu klienta, może przyjmować wartość NULL
- Email (varchar(20)) – adres e-mail klienta, może przyjmować wartość NULL
- CustomerType (varchar(1)) – oznacza rodzaj klienta, może przyjmować wartości `B`-oznacza klienta firmowego oraz ‘I’-indywidualnego, nie może przyjmować wartości NULL

Posiada indeks do pola CustomerType

```
CREATE TABLE Customers (
    CustomerID int  NOT NULL,
    Address varchar(40)  NULL,
    City varchar(20)   NULL,
    Region varchar(20)   NULL,
    PostalCode varchar(10)  NULL,
    Country varchar(20)  NULL,
    Phone varchar(20)  NULL,
    Email varchar(20)  NULL,
    CustomerType varchar(1)  NOT NULL,
    CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID),
       CONSTRAINT CustomerTypeCheck check (CustomerType='I' OR CustomerType='B')
);
```

CustomersDiscounts

Zawiera informacje o zniżkach klientów w restauracjach.

Posiada pola:

- CompanyID (int) PK – numer identyfikujący restaurację, nie może przyjmować wartości NULL
- CustomerID (int) PK – numer identyfikujący stałego klienta, nie może przyjmować wartości NULL
- OrdersWithK1 (int) – ilość zamówień, które dokonał klient za określoną dla restauracji kwotę K1, nie może przyjmować wartości NULL, może przyjmować wartości większe lub równe 0, posiada domyślną wartość 0
- ValidTillR2 (date) – data określająca ważność jednorazowego rabatu procentowego R2 dla klienta w restauracji, który złożył zamówienia na łączną kwotę K2, posiada domyślną wartość NULL
- ValidTillR3 (date) – data określająca ważność jednorazowego rabatu procentowego R3 dla klienta w restauracji, który złożył zamówienia na łączną kwotę K3, posiada domyślną wartość NULL
- MonthsWithFK1 (int) – liczba miesięcy, przez które zostało złożonych co najmniej FZ zamówień na łączną kwotę co najmniej FK1, nie może przyjmować wartości NULL, może przyjmować wartości większe lub równe 0, posiada domyślną wartość 0
- QuarterDiscount (money) - rabat kwotowy otrzymywany za każdy kolejny kwartał, w którym złożone były zamówienia na łączną kwotę FK2, nie może przyjmować wartości NULL, może przyjmować wartości większe lub równe 0, posiada domyślną wartość 0

Posiada indeksy do pól CompanyID oraz CustomerID.

```
CREATE TABLE CustomersDiscounts (
    CompanyID int  NOT NULL,
    CustomerID int  NOT NULL,
    OrdersWithK1 int  NOT NULL DEFAULT 0,
    ValidTillR2 date   NULL DEFAULT NULL,
    ValidTillR3 date   NULL DEFAULT NULL,
    MonthsWithFK1 int   NOT NULL DEFAULT 0,
    QuarterDiscount money  NOT NULL DEFAULT 0,
    CONSTRAINT CustomersDiscount_pk PRIMARY KEY  (CompanyID,CustomerID),
       CONSTRAINT OrdersWithK1Check check (OrdersWithK1>=0),
       CONSTRAINT MonthsWithFK1Check check (MonthsWithFK1>=0),
       CONSTRAINT QuarterDiscountCheck check (QuarterDiscount>=0),
       INDEX  companyid_index (CompanyID),
       INDEX customerid_index (CustomerID)
);
```

DiscountsData

Zawiera informację o zniżkach realizowanych przez restauracje.

Posiada pola:

- CompanyID (int) PK – unikalny numer identyfikujący restaurację, nie może przyjmować wartości NULL
- Z1 (int) – liczba zamówień, które musi złożyć klient w restauracji za kwotę K1, aby otrzymać rabat procentowy R1, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 10
- R1 (decimal(3,2)) – wysokość rabatu dla klientów, którzy złożyli Z1 zamówień na łączną kwotę K1, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.03
- R2 (decimal(3,2)) - wysokość rabatu dla klientów, którzy złożyli zamówienia na łączną kwotę K2, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.05
- R3 (decimal(3,2)) - wysokość rabatu dla klientów, którzy złożyli zamówienia na łączną kwotę K3, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.05
- K1 (money) – kwota, za jaką klient musi złożyć co najmniej Z1 zamówień, aby otrzymać rabat R1, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 30
- K2 (money) – łączna kwota, za jaką klient musi złożyć zamówienia, aby otrzymać jednorazowy rabat R2, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 1000
- K3 (money) – łączna kwota, za jaką klient musi złożyć zamówienia, aby otrzymać jednorazowy rabat R3, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 5000
- D1 (int) – liczba dni, przez które ważna jest jednorazowa zniżka R2, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 7
- D2 (int) – liczba dni, przez które ważna jest jednorazowa zniżka R3, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 7
- FZ (int) – liczba zamówień, które klient musi złożyć w ciągu miesiąca za łączną kwotę FK1, aby otrzymać rabat FR1, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 5
- FM (decimal(3,2)) – maksymalny łączny rabat procentowy, który może otrzymać klient, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.04
- FK1 (money) – łączna kwota, za jaką muszą być złożone zamówienia przez klienta w ciągu miesiąca, aby otrzymał on rabat FR1, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 500
- FK2 (money) – łączna kwota, za jaką muszą być złożone zamówienia przez klienta w ciągu kwartału , aby otrzymał on rabat FR2, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 10000
- FR1 (decimal(4,3)) – wysokość rabatu procentowego za każdy miesiąc, w którym złożone zostało FZ zamówień na łączną kwotę FK1, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.001
- FR2 (decimal(3,2)) – procent z łącznej kwoty, za którą zrealizowano zamówienia, stanowiący rabat kwotowy za każdy kwartał, w którym złożono zamówienia na łączną kwotę FK2, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0.05

```
CREATE TABLE DiscountsData (
    CompanyID int  NOT NULL,
    Z1 int  NOT NULL DEFAULT 10,
    R1 decimal(3,2)  NOT NULL DEFAULT 0.03,
    R2 decimal(3,2)  NOT NULL DEFAULT 0.05,
    R3 decimal(3,2)  NOT NULL DEFAULT 0.05,
    K1 money  NOT NULL DEFAULT 30,
    K2 money  NOT NULL DEFAULT 1000,
    K3 money  NOT NULL DEFAULT 5000,
    D1 int  NOT NULL DEFAULT 7,
    D2 int  NOT NULL DEFAULT 7,
    FZ int  NOT NULL DEFAULT 5,
    FM decimal(3,2)  NOT NULL DEFAULT 0.04,
    FK1 money  NOT NULL DEFAULT 500,
    FK2 money  NOT NULL DEFAULT 10000,
    FR1 decimal(4,3)  NOT NULL DEFAULT 0.001,
    FR2 decimal(3,2)  NOT NULL DEFAULT 0.05,
    CONSTRAINT DiscountsData_pk PRIMARY KEY  (CompanyID),
       CONSTRAINT Z1Check check (Z1>=1),
       CONSTRAINT R1Check check (R1>=0 AND R1<=1),
       CONSTRAINT R2Check check (R2>=0 AND R2<=1),
       CONSTRAINT R3Check check (R3>=0 AND R3<=1),
       CONSTRAINT K1Check check (K1>=0),
       CONSTRAINT K2Check check (K2>=0),
       CONSTRAINT K3Check check (K3>=0),
       CONSTRAINT D1Check check (D1>=0),
       CONSTRAINT D2Check check (D2>=0),
       CONSTRAINT FZCheck check (FZ>=0),
       CONSTRAINT FMCheck check (FM>=0 AND FM<=1),
       CONSTRAINT FK1Check check (FK1>=0),
       CONSTRAINT FK2Check check (FK2>=0),
       CONSTRAINT FR1Check check (FR1>=0 AND FR1<=1),
       CONSTRAINT FR2Check check (FR2>=0 AND FR2<=1)
);
```

Employees

Zawiera informacje o pracownikach restauracji.

Posiada pola:

- EmployeeID (int) PK – unikalny numer identyfikujący pracownika, nie może przyjmować wartości NULL
- Firstname (varchar(20)) – imię pracownika, nie może przyjmować wartości NULL
- Lastname (varchar(30)) – nazwisko pracownika, nie może przyjmować wartości NULL
- Title (varchar(30)) - nazwa stanowiska pracownika, może przyjmować wartości NULL
- CompanyID (int) – numer identyfikujący restaurację, w której jest zatrudniony pracownik, nie może przyjmować wartości NULL
- TitleOfCourtesy (varchar(5)) – zwrot grzecznościowy, może przyjmować wartości NULL
- BirthDate (date) – data urodzenia pracownika, nie może przyjmować wartości NULL
- HireDate (date) – data zatrudnienia pracownika, nie może przyjmować wartości NULL
- FireDate (date) – data zwolnienia pracownika, może przyjmować wartości NULL
- Address (varchar(40)) – ulica oraz numer lokalu, w którym mieszka pracownik, nie może przyjmować wartości NULL
- City (varchar(20)) - miejscowość, w której mieszka pracownik, nie może przyjmować wartości NULL
- Region (varchar(20)) – region, w którym mieszka pracownik, może przyjmować wartości NULL
- PostalCode (varchar(10)) – kod pocztowy dla adresu, pod którym mieszka pracownik, nie może przyjmować wartości NULL

- Country (varchar(20)) – kraj, w którym mieszka pracownik, nie może przyjmować wartości NULL
- Phone (varchar(20)) – numer telefonu pracownika, nie może przyjmować wartość NULL
- Notes (text) - uwagi dotyczące pracownika, może przyjmować wartości NULL

Posiada indeksy do pól CompanyID oraz złączenia pól (FirstName,LastName)

```
CREATE TABLE Employees (
    EmployeeID int  NOT NULL,
    Firstname varchar(20)  NOT NULL,
    Lastname varchar(30)  NOT NULL,
    Title varchar(30) NULL,
    CompanyID int  NOT NULL,
    TitleOfCourtesy varchar(5) NULL,
    BirthDate date  NOT NULL,
    HireDate date  NOT NULL,
    FireDate date  NULL,
    Address varchar(40)  NOT NULL,
    City varchar(20)  NOT NULL,
    Region varchar(20) NULL,
    PostalCode varchar(10)  NOT NULL,
    Country varchar(20)  NOT NULL,
    Phone varchar(20)  NOT NULL,
    Notes text NULL,
    CONSTRAINT EmployeeID PRIMARY KEY  (EmployeeID),
       INDEX companyid_index (CompanyID),
       INDEX fullname_index  (Firstname,Lastname),
       INDEX fullname_index1  (Lastname,Firstname)

);
```

IndividualCustomers

Zawiera szczegółowe informacje o stałych klientach restauracji, którzy są klientami indywidualnymi.
Posiada pola:

- CustomerID (int) PK – unikalny numer identyfikujący klienta, nie może przyjmować wartości NULL
- Fisrtname (varchar(20)) – imię klienta, nie może przyjmować wartości NULL
- Lastname (varchar(30))– nazwisko klienta, nie może przyjmować wartości NULL
- BirthDate (date) – data urodzenia klienta, może przyjmować wartości NULL
- PESEL (varchar(11)) – numer pesel klienta, może przyjmować wartości NULL

Posiada indeks dla złączenia pól (Firstname, Lastname)

```
CREATE TABLE IndividualCustomers (
    CustomerID int  NOT NULL,
    Firstname varchar(20)  NOT NULL,
    Lastname varchar(30)  NOT NULL,
    BirthDate date  NULL,
    PESEL varchar(11) NULL,
    CONSTRAINT IndividualCustomers_pk PRIMARY KEY  (CustomerID),
       INDEX fullname_index (Firstname,Lastname),
       INDEX fullname_index1  (Lastname,Firstname)

);
```

Ingredients

Zawiera informacje o składnikach wykorzystywanych do dań w restauracjach.

Posiada pola:

- IngredientID (int) PK – unikalny numer identyfikujący składnik, nie może przyjmować wartości NULL
- IngredientName (varchar(20)) – nazwa składnika, nie może przyjmować wartości NULL
- SeaFood (varchar(1)) – informacja o tym, czy składnik jest owocem morza, przyjmuje wartości `Y` oraz `N`, nie może przyjmować wartości NULL, posiada domyślną wartość `N`

Posiada indeks do pola SeaFood.

```
CREATE TABLE Ingredients (
    IngredientID int  NOT NULL,
    IngredientName varchar(20)  NOT NULL,
    SeaFood varchar(1)  NOT NULL DEFAULT 'N',
    CONSTRAINT Ingredients_pk PRIMARY KEY  (IngredientID),
       CONSTRAINT SeaFoodCheck check (SeaFood='Y' OR SeaFood='N'),
       INDEX nameindex (IngredientName)

);
```

IntervalDetails

Zawiera szczegołowe informacje o czasowym układzie oraz dostępności stolików w restauracji.

Posiada pola:

- TableID (int) PK – numer identyfikujący stolik, nie może przyjmować wartości NULL
- IntervalID (int) PK – numer identyfikujący układ stolików, nie może przyjmować wartości NULL
- AvailableChairst (int) – liczba miejsc dostępna dla danego stolika w danym układzie, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0

Posiada indeksy do pól TableID oraz IntervalID.

```
CREATE TABLE IntervalDetails (
    TableID int  NOT NULL,
    IntervalID int  NOT NULL,
    AvailableChairs int  NOT NULL,
    CONSTRAINT IntervalDetails_pk PRIMARY KEY  (IntervalID,TableID),
       CONSTRAINT AvailableChairsCheck check (AvailableChairs>=0),
       INDEX tableid_index (TableID),
       INDEX intervalid_index (IntervalID)
);
```

Intervals

Zawiera informacje dotyczące układów stolików w restauracjach.

Posiada pola:

- IntervalID (int) PK – unikalny numer identyfikujący układ stolików w restauracji, nie może przyjmować wartości NULL
- CompanyID (int) – numer identyfikujący restaurację, nie może przyjmować wartości NULL
- BeginningDate (date) – data początku obowiązywania układu stolików w restauracji, nie może przyjmować wartości NULL
- EndDate (date) – data końca obowiązywania układu stolików w restauracji, może przyjmować wartości NULL, jeżeli nie jest NULLEM, to może przyjmować tylko wartości późniejsze od BeginningDate

Posiada indeks dla pola CompanyID

```
CREATE TABLE Intervals (
    IntervalID int  NOT NULL,
    CompanyID int  NOT NULL,
    BeginningDate date  NOT NULL,
    EndDate date NULL,
    CONSTRAINT Intervals_pk PRIMARY KEY  (IntervalID),
       CONSTRAINT DateCheck check (BeginningDate<=ISNULL(EndDate,BeginningDate)),
       INDEX companyid_index (CompanyID),
       INDEX beginningdate_index (BeginningDate)

);
```

Menu

Zawiera informacje o menu w restauracjach.

Posiada pola:

- MenuID (int) PK – unikalny numer identyfikujący menu, nie może przyjmować wartości NULL
- CompanyID (int) – numer identyfikujący restaurację, nie może przyjmować wartości NULL
- InDate (date) – data początku obowiązywania danego menu w restauracji, nie może przyjmować wartości NULL
- OutDate (date) – data końca obowiązywania danego menu w restauracji, może przyjmować wartości NULL, jeżeli nie jest NULLEM, to musi być późniejsza od daty InDate

Posiada indeks dla pola CompanyID.

```
CREATE TABLE Menu (
    MenuID int  NOT NULL,
    CompanyID int  NOT NULL,
    InDate date  NOT NULL,
    OutDate date NULL,
    CONSTRAINT Menu_pk PRIMARY KEY  (MenuID),
       CONSTRAINT DateCheck_1 check (InDate<=ISNULL(OutDate,InDate)),
       INDEX companyid_index (CompanyID),
       INDEX indate_index (InDate)

);
```

MenuDetails

Zawiera szczegółowe informacje o menu dla restauracji.

Posiada pola:

- MenuID (int) PK – numer identyfikujący menu dla restauracji, nie może przyjmować wartości NULL
- CourseID (int) PK – numer identyfikujący danie w menu restauracji, nie może przyjmować wartości NULL
- CategoryID (int) – numer identyfikujący kategorię, do której należy danie w danym menu, nie może przyjmować wartości NULL
- UnitPrice (money) – cena jednostkowa dla dania w danym menu, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0

Posiada indeksy do pól MenuID oraz CourseID.

```
CREATE TABLE MenuDetails (
    MenuID int  NOT NULL,
    CourseID int  NOT NULL,
    CategoryID int  NOT NULL,
    UnitPrice money  NOT NULL,
    CONSTRAINT MenuDetails_pk PRIMARY KEY  (MenuID,CourseID),
```

```
    CONSTRAINT UnitPriceCheck check (UnitPrice>=0),
    INDEX menuid_index (MenuID),
    INDEX courseid_index (CourseID)
);
```

OrderDetails

Zawiera szczegółowe informacje o złożonym zamówieniu w restauracji.

Posiada pola:

- OrderID (int) PK – numer identyfikujący zamówienie w restauracji, nie może przyjmować wartości NULL
- CourseID (int) PK – numer identyfikujący zamówione danie, nie może przyjmować wartości NULL
- MenuID (int) – numer identyfikujący menu, z którego zostało zamówione danie, nie może przyjmować wartości NULL
- Quantity (int) – liczba zamówionych sztuk danego dania, nie może przyjmować wartości NULL, przyjmuje wartości większe od 0

Posiada indeksy do pól OrderID oraz CourseID.

```
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    CourseID int  NOT NULL,
    MenuID int  NOT NULL,
    Quantity int  NOT NULL,
    CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,CourseID),
    CONSTRAINT QuantityCheck check (Quantity>0),
    INDEX orderid_index (OrderID),
    INDEX courseid_index (CourseID)
);
```

Orders

Zawiera informacje o złożonym zamówieniu.

Posiada pola:

- OrderID (int) PK – unikalny numer identyfikujący zamówienie, nie może przyjmować wartości NULL
- EmployeeID (int) –numer identyfikujący pracownika, który obsługuje zamówienie, może przyjmować wartości NULL,
- CustomerID (int) – numer identyfikujący stałego klienta, który złożył zamówienie, może przyjmować wartości NULL
- CompanyID (int) – numer identyfikujący restaurację, w której zostało złożone zamówienie, nie może przyjmować wartości NULL
- OrderDate (datetime) – data i godzina złożenia zamówienia, nie może przyjmować wartości NULL
- CompletionDate (datetime) – data zapłaty za zamówienie, może przyjmować wartość NULL, jeżeli nie jest NULLEM, to musi być późniejsza niż OrderDate
- TakeAway (varchar(1)) – informacja o tym, czy zamówienie było złożone na wynos, nie może przyjmować wartości NULL, przyjmuje wartości `Y` oraz `N`
- DisposableDiscount (money) – wartość zniżki kwotowej dla zamówienia, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0, posiada domyślną wartość 0
- PercentageDiscount (decimal(4,3)) – wartość zniżki procentowej dla zamówienia, nie może przyjmować wartości NULL, przyjmuje wartości pomiędzy 0 a 1, posiada domyślną wartość 0

- Valid (varchar(1)) – informacja o tym, czy zamówienie jest ważne(czy nie zostało anulowane), nie może przyjmować wartości NULL, przyjmuje wartości `Y` lub `N`

Posiada indeksy do pól: EmployeeID, CustomerID, CompanyID, Valid, TakeAway oraz OrderDate.

```
CREATE TABLE Orders (
    OrderID int  NOT NULL,
    EmployeeID int NULL,
    CustomerID int NULL,
    CompanyID int  NOT NULL,
    OrderDate datetime  NOT NULL,
    CompletionDate datetime NULL,
    TakeAway varchar(1)  NOT NULL,
    DisposableDiscount money  NOT NULL DEFAULT 0,
    PercentageDiscount decimal(4,3)  NOT NULL DEFAULT 0,
    Valid varchar(1) NOT NULL DEFAULT 'Y',
    CONSTRAINT OrderID PRIMARY KEY  (OrderID),
       CONSTRAINT TakeAwayCheck check (TakeAway='Y' OR TakeAway='N'),
       CONSTRAINT DisposableDiscountCheck check (DisposableDiscount>=0),
       CONSTRAINT PercentageDisscountCheck check (PercentageDiscount>=0 AND
       PercentageDiscount<=1),
       CONSTRAINT DateCheck_2 check (OrderDate<=(ISNULL(CompletionDate,OrderDate))),
       INDEX employeeid_index (EmployeeID),
       INDEX customerid_index (CustomerID),
       INDEX companyid_index (CompanyID),
       INDEX orderdate_index (OrderDate)

);
```

ReservationDetails

Zawiera szczegółowe informacje o uczestnikach złożonej rezerwacji w restauracji.

Posiada pola:

- ResetvationID (int) PK – numer identyfikujący rezerwację, nie może przyjmować wartości NULL
- GuestID (int) PK – unikalny numer identyfikujący uczestnika rezerwacji, nie może przyjmować wartości NULL
- Name (varchar(40)) – imię uczestnika rezerwacji, nie może przyjmować wartości NULL
- Surname (varchar(40)) – nazwisko uczestnika rezerwacji, nie może przyjmować wartości NULL
- Presence (varchar(1)) - informacja o tym, czy uczestnik pojawił się w restauracji w czasie rezerwacji, nie może przyjmować wartości NULL, przyjmuje wartości `Y` oraz `N`, posiada domyślną wartość `Y`

Posiada indeksy do pola Presence.

```
CREATE TABLE ReservationDetails (
    ReservationID int  NOT NULL,
    GuestID int NOT NULL,
    Name varchar(40)  NOT NULL,
    Surname varchar(40)  NOT NULL,
    Presence varchar(1)  NOT NULL DEFAULT 'Y',
    CONSTRAINT ReservationDetails_pk PRIMARY KEY  (ReservationID,GuestID),
       CONSTRAINT U_GuestID UNIQUE(GuestID),
       CONSTRAINT PresenceCheck check (Presence='Y' OR Presence='N')
);
```

Reservations

Zawiera informacje o rezerwacjach złożonych w restauracjach.

Posiada pola:

- ReservationID (int) PK – unikalny numer identyfikujący rezerwację, nie może przyjmować wartości NULL
- OrderID (int) – numer zamówienia, którym jest rezerwacja, musi być unikalny w tabeli, nie może przyjmować wartości NULL
- ReservationDate (datetime) – data i godzina rozpoczęcia realizacji złożonej rezerwacji, nie może przyjmować wartości NULL
- SubmissionDate (datetime) – data i godzina zatwierdzenia rezerwacji przez pracownika restauracji, może przyjmować wartości NULL

Posiada indeksy dla pól OrderID oraz ReservationDate.

```
CREATE TABLE Reservations (
    ReservationID int  NOT NULL,
    OrderID int  NOT NULL,
    ReservationDate datetime  NOT NULL,
    SubmissionDate datetime NULL,
    CONSTRAINT Reservations_pk PRIMARY KEY  (ReservationID),
       CONSTRAINT U_OrderID UNIQUE(OrderID),
       INDEX reservationdate_index (ReservationDate),
       INDEX orderid_index (OrderID),
       INDEX reservationdate_index (ReservationDate)
);
```

Tables

Zawiera informację o wszystkich stolikach w restauracjach.

Posiada pola:

- TableID (int) PK – unikalny numer identyfikujący stolik w restauracji, nie może przyjmować wartości NULL
- CompanyID (int) – numer identyfikujący restaurację, w której znajduje się dany stolik, nie może przyjmować wartości NULL
- NumberOfChairs (int) – maksymalna ilość miejsc przy stoliku, nie może przyjmować wartości NULL, przyjmuje wartości większe lub równe 0
- Description (text) – opis stolika lub jego położenia w restauracji, może przyjmować wartości NULL

Posiada indeks dla pola CompanyID.

```
CREATE TABLE Tables (
    TableID int  NOT NULL,
    CompanyID int  NOT NULL,
    NumberOfChairs int  NOT NULL,
    Description text NULL,
    CONSTRAINT Tables_pk PRIMARY KEY (TableID),
       CONSTRAINT NumberOfChairsCheck check (NumberOfChairs>=0),
       INDEX companyid_index (CompanyID)
);
```

Relacje pomiędzy tabelami

Relacja Contains-Ingredients

```
ALTER TABLE "Contains" ADD CONSTRAINT Contains_Ingredients
    FOREIGN KEY (IngredientID)
    REFERENCES Ingredients (IngredientID);
```

Relacja Courses-Contains

```
ALTER TABLE "Contains" ADD CONSTRAINT Courses_Contains
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

Relacja CustomersDiscounts-Customers

```
ALTER TABLE CustomersDiscounts ADD CONSTRAINT CustomersDiscount_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

Relacja CustomersDiscounts-DiscountsData

```
ALTER TABLE CustomersDiscounts ADD CONSTRAINT CustomersDiscount_DiscountsData
    FOREIGN KEY (CompanyID)
    REFERENCES DiscountsData (CompanyID);
```

Relacja Customers-BusinessCustomers

```
ALTER TABLE BusinessCustomers ADD CONSTRAINT Customers_BusinessCustomers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

Relacja Customers-IndividualCustomers

```
ALTER TABLE IndividualCustomers ADD CONSTRAINT Customers_IndividualCustomers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

Relacja DiscountsData-Company

```
ALTER TABLE DiscountsData ADD CONSTRAINT DiscountsData_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Relacja Employees-Company

```
ALTER TABLE Employees ADD CONSTRAINT Employees_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Relacja MenuDetails-Categories

```
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Categories
    FOREIGN KEY (CategoryID)
    REFERENCES Categories (CategoryID);
```

Relacja MenuDetails-Courses

```
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

Relacja MenuDetails-Menu

```
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Menu
    FOREIGN KEY (MenuID)
    REFERENCES Menu (MenuID);
```

Relacja Menu-Company

```
ALTER TABLE Menu ADD CONSTRAINT Menu_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Relacja OrderDetails-MenuDetails

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_MenuDetails
    FOREIGN KEY (MenuID,CourseID)
    REFERENCES MenuDetails (MenuID,CourseID);
```

Relacja OrderedTables-Orders

```
ALTER TABLE OrderedTables ADD CONSTRAINT OrderedTables_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

Relacja OrderedTables-IntervalDetails

```
ALTER TABLE OrderedTables ADD CONSTRAINT OrderedTables_IntervalDetails
    FOREIGN KEY (IntervalID,TableID)
    REFERENCES IntervalDetails (IntervalID,TableID);
```

Relacja Orders-Company

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Relacja Orders-Customers

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

Relacja Orders-Employees

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Employees
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

Relacja Orders-OrderDetails

```
ALTER TABLE OrderDetails ADD CONSTRAINT Orders_OrderDetails
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

Relacja ReservationDetails-Reservations

```
ALTER TABLE ReservationDetails ADD CONSTRAINT Reservation_ReservationsDetails
    FOREIGN KEY (ReservationID)
    REFERENCES Reservations (ReservationID);
```

Relacja Reservations-Orders

```
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

Relacja IntervalDetails-Tables

```
ALTER TABLE IntervalDetails ADD CONSTRAINT IntervalDetails_Tables
    FOREIGN KEY (TableID)
    REFERENCES Tables (TableID);
```

Relacja IntervalDetails-Company

```
ALTER TABLE Intervals ADD CONSTRAINT IntervalDetails_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Relacja Tables-Company

```
ALTER TABLE Tables ADD CONSTRAINT Tables_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);
```

Widoki

OrdersData

Widok zawierający informacje o zamówieniu:

- ID zamówienia
- ID Klienta który złożył zamówienie
- ID Restauracji w której złożono zamówienie
- Data zamówienia
- Łączny koszt zamówienia bez uwzględniania rabatów
- Rabat kwotowy przyznany do zamówienia
- Rabat procentowy przyznany do zamówienia
- Informacje czy zamówienie jest aktualne

```
CREATE OR ALTER VIEW OrdersData
AS
	SELECT O.OrderID, CustomerID, CompanyID, OrderDate,
sum(MD.UnitPrice*OD.Quantity) as Sum, DisposableDiscount, PercentageDiscount, Valid
	FROM Orders as O
	INNER JOIN OrderDetails as OD
	ON O.OrderID = OD.orderID
	INNER JOIN MenuDetails as MD
	ON OD.MenuID = MD.MenuId and OD.CourseId = MD.courseID
	GROUP BY O.OrderID, CustomerID, CompanyID, OrderDate, disposableDiscount,
PercentageDiscount, Valid
GO
```

Courses Availability

Widok zawierający informacje o daniach:

- ID Dania
- ID Restauracji w jakiej dostępne było dane danie
- ID Menu do jakiego należy danie
- Początek dostępności dania w Menu
- Koniec dostępności dania w Menu

```
CREATE OR ALTER VIEW CoursesAvailability
AS
	SELECT MD.CourseID,M.MenuID, M.CompanyID, M.InDate, M.OutDate
	FROM MenuDetails as MD
	INNER JOIN Menu as M
	ON MD.MenuID = M.MenuID
GO
```

ShowUsedDiscounts

Widok zawierający informacje o przyznanych rabatach:

- ID Klienta jakiemu przyznano rabat
- ID Restauracji która przyznała rabat
- ID zamówienia do którego przyznano rabat
- Kwotę rabatu jaki przyznano zamówieniu
- Wartość rabatu procentowego jaki przyznano zamówieniu

```
CREATE OR ALTER VIEW ShowUsedDiscounts
AS
	SELECT CustomerID,CompanyID,OrderID,DisposableDiscount,PercentageDiscount
	FROM Orders
	WHERE CustomerID IS NOT NULL AND Valid='Y' AND (DisposableDiscount IS NOT NULL
OR PercentageDiscount!=0)
GO
```

Funkcje

GetRestaurantCurrentMenuID

Funkcja zwracająca ID aktualnie obowiązującego menu w restauracji.

```
CREATE OR ALTER FUNCTION GetRestaurantCurrentMenuID (
	@RestaurantID int
)
	RETURNS int
AS
BEGIN
	RETURN ( [dbo].GetRestaurantMenuID(@RestaurantID, GETDATE()))
END
GO
```

GetRestaurantMenuID

Funkcja zwracająca ID menu które obowiązuje w danym czasie @DateTime.

```
CREATE OR ALTER FUNCTION GetRestaurantMenuID (
	@RestaurantID int,
	@DataTime date
)
	RETURNS int
AS
BEGIN
	RETURN (
		SELECT MenuID
		FROM Menu
		WHERE CompanyID = @RestaurantID and (OutDate is null OR OutDate >
@DataTime) and InDate < @DataTime
	);
END
GO
```

GetCompanyWithMenuID

Funkcja zwracająca ID restauracji do której należy dane menu.

```
CREATE OR ALTER FUNCTION GetCompanyWithMenuID(
	@MenuID int
)
	RETURNS int
AS
BEGIN
	RETURN (
		SELECT CompanyID FROM Menu WHERE MenuID = @MenuID
	)
END
GO
```

GetRestaurantCurrentIntervalID

Funkcja zwracająca ID aktualnie obowiązującego układu stolików w danej restauracji.

```
CREATE OR ALTER FUNCTION GetRestaurantCurrentIntervalID (
	@RestaurantID int
)
	RETURNS int
AS
BEGIN
	RETURN [dbo].GetRestaurantIntervalID(@RestaurantID, GETDATE())
END
GO
```

GetRestaurantIntervalID

Funkcja zwracająca ID układu stolików obowiązującego w danym czasie w danej restauracji.

```
CREATE OR ALTER FUNCTION GetRestaurantIntervalID (
	@RestaurantID int,
	@DataTime date
)
	RETURNS int
AS
BEGIN
	RETURN (
		SELECT IntervalID
		FROM Intervals
		WHERE CompanyID = @RestaurantID and (EndDate is null OR EndDate >
@DataTime) and BeginningDate < @DataTime
	);
END
GO
```

GetCustomerID

Funkcja zwracająca ID klienta który złożył dane zamówienie.

```
CREATE OR ALTER FUNCTION GetCustomerID(
	@OrderID int
)
	RETURNS int
AS
BEGIN
	RETURN (
		SELECT CustomerID
		FROM Orders
		WHERE OrderID=@OrderID
	);
END
GO
```

TotalOrderCost

Funkcja zwracająca łączną kwotę zamówienia, bez uwzględnienia rabatów.

```
CREATE OR ALTER FUNCTION TotalOrderCost (
	@OrderID int
)
	RETURNS int
AS
BEGIN
	RETURN (
SELECT [Sum] FROM OrdersData WHERE OrderID = @OrderID);
END
GO
```

MinReservationCost

Funkcja zwracająca minimalną kwotę zamówienia potrzebną do złożenia rezerwacji. Uwzględnia ona ilość zamówień dokonaną przez klienta w przeszłości.

```
CREATE OR ALTER FUNCTION MinReservationCost (
	@CustomerID int,
	@CompanyID int
)
	RETURNS int
AS
BEGIN
	IF	(
		SELECT count(*)
		FROM OrdersData
		WHERE CustomerID = @CustomerID and CompanyID = @CompanyID
		GROUP BY OrderID) > 5
			RETURN (SELECT ReservationMinAmount FROM Company WHERE CompanyID
= @CompanyID)
	RETURN (SELECT ReservationMinAmountRegularClients FROM Company WHERE CompanyID
= @CompanyID)
END
GO
```

ReservationConditions

Funkcja sprawdzająca czy zamówienie spełnia warunki potrzebne do złożenia rezerwacji, czyli minimalną kwotę zamówienia, jeśli zamówienie spełnia warunki funkcja zwraca 1 w przeciwnym wypadku zwraca 0.

```
CREATE OR ALTER FUNCTION ReservationConditions(
	@OrderID int
)
	RETURNS int
AS
BEGIN
	DECLARE @CompanyID int = (SELECT CompanyID FROM Orders WHERE OrderID =
@OrderID)
	DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE OrderID =
@OrderID)
	IF ([dbo].TotalOrderCost(@OrderID) >= [dbo].MinReservationCost(@CustomerID,
@CompanyID))
		RETURN 1
	RETURN 0
END
GO
```

AvailablePlace

Funkcja zwracająca ilość miejsc dostępnych w danym czasie w restauracji. Bierze pod uwagę ilość zajętych miejsc w danym czasie.

```
CREATE or ALTER FUNCTION AvailablePlace(
	@CompanyID int,
	@Time datetime
)
	RETURNS int
AS
BEGIN
	DECLARE @PlacesInRestaurant int = (
		SELECT SUM(AvailableChairs)
		FROM IntervalDetails AS ID
		INNER JOIN Intervals AS I
		ON I.IntervalID = ID.IntervalID
		WHERE I.BeginningDate <= @Time and I.EndDate >= @Time and I.CompanyID =
@CompanyID
		)
	DECLARE @OccupiedPlaces int = (
		SELECT SUM(OccupiedChairs)
		FROM OrderedTables as OT
		INNER JOIN Orders as O
		ON O.OrderID = OT.OrderID
		WHERE O.OrderDate <= @Time and O.CompletionDate >= @Time and O.CompanyID
= @CompanyID and O.Valid = 'Y' and O.TakeAway = 'N'
	)
	RETURN @PlacesInRestaurant - @OccupiedPlaces
END
GO
```

CourseInDate

Funkcja zwracająca datę pojawienia się danej potrawy w restauracji. Jeśli dana potrawa była w kilku kolejnych menu zwraca datę pojawienia się jej w najwcześniejszym menu.

```
CREATE OR ALTER FUNCTION CourseInDate(
	@CourseID int,
	@MenuID int
)
	RETURNS date
AS
BEGIN
	DECLARE @InDate date = (SELECT InDate FROM Menu WHERE MenuID = @MenuID)
	DECLARE @PreviousCourseID int = (SELECT TOP 1 CourseID FROM CoursesAvailability
			WHERE DATEDIFF(DD,OutDate,@InDate) = 1 and CompanyID =
[dbo].GetCompanyWithMenuID(@MenuID) and CourseID = @CourseID)
	IF (@PreviousCourseID is null)
		RETURN @InDate
	DECLARE @PreviousMenuID int = (SELECT TOP 1 MenuID FROM CoursesAvailability
			WHERE DATEDIFF(DD,OutDate,@InDate) = 1 and CompanyID =
[dbo].GetCompanyWithMenuID(@MenuID) and CourseID = @CourseID)
	RETURN [dbo].CourseInDate(@CourseID, @PreviousMenuID)
END
GO
```

MenuConditions

Funkcja sprawdzająca czy menu spełnia warunki:

1. Czy połowa potraw które były dostępne w menu w ciągu dwóch tygodni jest zmieniona
2. Czy nowe potrawy nie były dostępne w przeciągu miesiąca czasu przed ich wprowadzeniem

Jeśli menu spełnia warunki to funkcja zwraca 1, w przeciwnym wypadku funkcja zwraca 0.

```
CREATE OR ALTER FUNCTION MenuConditions(
	@MenuID int
)
	RETURNS int
AS
BEGIN
	DECLARE @StartDate date = (SELECT InDate FROM Menu WHERE MenuID = @MenuID)
	DECLARE @EndDate date = (SELECT OutDate FROM Menu WHERE MenuID = @MenuID)
	IF (@EndDate is null)
		SET @EndDate = GETDATE()
	IF (DATEDIFF(DD, @StartDate, @EndDate) > 14)
		RETURN 0 --jeśli menu trwa dłużej niż dwa tygodnie nie można było
zmienić potraw

	DECLARE @CoursesInMenu TABLE(CourseID int)
	INSERT INTO @CoursesInMenu SELECT CourseID FROM CoursesAvailability WHERE
MenuID = @MenuID

	IF (SELECT COUNT(*)
		FROM CoursesAvailability
		WHERE DATEDIFF(DD, OutDate, @StartDate) <= 31 and CourseID in (SELECT *
FROM @CoursesInMenu)
			and DATEDIFF(DD, OutDate, @StartDate) > 1 and CompanyID =
[dbo].GetCompanyWithMenuID(@MenuID)) > 0
		RETURN 0 -- istnieją dania które były zdjęte zbyt wcześnie

	DECLARE @MenuSize int = (SELECT COUNT(*) FROM @CoursesInMenu)
	DECLARE @CoursesWithMoreThanTwoWeeksIn int =
		(SELECT COUNT(*) FROM @CoursesInMenu WHERE
DATEDIFF(DD,[dbo].CourseInDate(CourseID,@MenuID),@EndDate) > 14)

	IF (@MenuSize < 2*@CoursesWithMoreThanTwoWeeksIn)
		RETURN 0 --połowa Menu nie jest wymieniona
	RETURN 1
END
GO
```

CanAddCourse

Funkcja sprawdzająca czy potrawa może dostać dodana do menu, czyli czy nie została zdjęcia wcześniej niż miesiąc temu.

```
CREATE OR ALTER FUNCTION CanAddCourse(
	@MenuID int,
	@CourseID int
)
	RETURNS int
AS
BEGIN
	DECLARE @StartDate date = (SELECT InDate FROM Menu WHERE MenuID = @MenuID)
	IF (SELECT COUNT(*)
		FROM CoursesAvailability
		WHERE DATEDIFF(DD, OutDate, @StartDate) <= 31 and CourseID = @CourseID
and
```

```
                    DATEDIFF(DD, OutDate, @StartDate) > 1 and CompanyID =
[dbo].GetCompanyWithMenuID(@MenuID)) > 0
                RETURN 0
        RETURN 1
END
GO
```

GetTableWithNPlaces

Funkcja zwracająca stolik z @Places wolnymi miejscami z danej restauracji w danym czasie. Jeśli nie można zwrócić takiego stolika zwracana jest wartość -1.

```
CREATE OR ALTER FUNCTION GetTableWithNPlaces(
        @Company int,
        @Places int,
        @Time datetime
)
        RETURNS int
AS
BEGIN
        DECLARE @IntervalID int = [dbo].GetRestaurantIntervalID(@Company,@Time)
        DECLARE @SearchTable int = (
                SELECT TOP 1 ID.TableID
                FROM IntervalDetails AS ID
                INNER JOIN Intervals AS I
                ON I.IntervalID = ID.IntervalID
                INNER JOIN OrderedTables AS OT
                ON ID.IntervalID = OT.IntervalID and ID.TableID = OT.TableID
                WHERE I.IntervalID = @IntervalID
                GROUP BY ID.IntervalID, ID.TableID, AvailableChairs
                HAVING AvailableChairs - SUM(OccupiedChairs)>= @Places
        )
        IF @SearchTable IS NULL
                RETURN -1
        RETURN @SearchTable
END
GO
```

ContainsSeaFoodCourse

Funkcja sprawdzająca czy danie zawiera owoce morza. Zwraca 1 jeśli danie zawiera owoce morza i 0 w przeciwnym wypadku.

```
CREATE OR ALTER FUNCTION ContainsSeaFoodCourse(
        @CourseID int
)
        RETURNS int
AS
BEGIN
        IF((SELECT COUNT(*) FROM Ingredients
        INNER JOIN [Contains] ON Ingredients.IngredientID=[Contains].IngredientID AND
CourseID=@CourseID
        WHERE SeaFood='Y')>0 )
                RETURN 1
        RETURN 0
END
GO
```

ContainsSeaFoodOrder

Funkcja sprawdzająca czy dane zamówienie zawiera owoce morza. Zwraca 1 jeśli danie zawiera owoce morza i 0 w przeciwnym wypadku.

```
CREATE OR ALTER FUNCTION ContainsSeaFoodOrder(
	@OrderID int
)
	RETURNS int
AS
BEGIN
	IF((SELECT COUNT(*) FROM OrderDetails WHERE OrderID=@OrderID AND
[dbo].ContainsSeaFoodCourse(CourseID)='Y')>0)
		RETURN 1
	RETURN 0
END
GO
```

OrdersSumFromDate

Funkcja zwracająca kwotę jaką klient wydał w danej restauracji od określonej dany. Jeśli w danym czasie nie dokonywał zamówień w restauracji to zwracana jest wartość 0.

```
CREATE OR ALTER FUNCTION OrdersSumFromDate(
	@CustomerID int,
	@CompanyID int,
	@Date datetime
)
	RETURNS money
AS
BEGIN
	RETURN (SELECT SUM(Quantity*UnitPrice) FROM Orders
	INNER JOIN OrderDetails ON Orders.OrderID=OrderDetails.OrderID
	INNER JOIN MenuDetails ON MenuDetails.MenuID=OrderDetails.MenuID AND
MenuDetails.CourseID=OrderDetails.CourseID
	WHERE OrderDate>@Date AND CustomerID=@CustomerID AND CompanyID=@CompanyID)
END
GO
```

Procedury

ShowDiscounts

Procedura wyświetlająca rabaty w danej restauracji.

```
CREATE OR ALTER PROCEDURE ShowDiscounts
	@CompanyID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT *
	FROM DiscountsData
	WHERE CompanyID=@CompanyID
END
GO
```

AddNewIndividualCustomer

Procedura dodająca nowego klienta indywidualnego do bazy. Tworzy rekordy w tabelach Customers i IndividualCustomers opisujące klienta.

```
CREATE or ALTER PROCEDURE AddNewIndividualCustomer
	@Address varchar(40),
	@City varchar(40),
	@PostalCode varchar(40),
	@Country varchar(40),
	@Phone varchar(11),
	@Email varchar(40),
	@Firstname varchar(35),
	@Lastname varchar(35),
	@BirthDate date,
	@PESEL varchar(11)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @CustomerID int=(SELECT ISNULL(MAX(CustomerID),1) FROM Customers)
	DECLARE @CustomerType varchar(1)='I'
	BEGIN TRANSACTION
	INSERT INTO Customers
(CustomerID,Address,City,PostalCode,Country,Phone,Email,CustomerType)
	VALUES (@CustomerID,
@Address,@City,@PostalCode,@Country,@Phone,@Email,@CustomerType)
	INSERT INTO IndividualCustomers (CustomerID,FIrstname,Lastname,BirthDate,PESEL)
	VALUES (@CustomerID,@Firstname,@Lastname,@BirthDate,@PESEL)
	COMMIT TRANSACTION
END
GO
```

AddNewBusinesCustomer

Procedura dodająca nowego klienta firmowego do bazy. Tworzy rekordy w tabelach Customers i BusinessCustomers opisujące klienta

```
CREATE or ALTER PROCEDURE AddNewBusinessCustomer
	@Address varchar(40),
	@City varchar(40),
	@PostalCode   varchar(40),
	@Country varchar(40),
	@Phone varchar(11),
	@Email varchar(40),
```

```
	@CompanyName varchar(40),
	@NIP varchar(10),
	@Fax varchar(20)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @CustomerID int=(SELECT ISNULL(MAX(CustomerID),1) FROM Customers)
	DECLARE @CustomerType varchar(1)='B'
	BEGIN TRANSACTION
	INSERT INTO Customers
(CustomerID,Address,City,PostalCode,Country,Phone,Email,CustomerType)
	VALUES (@CustomerID,
@Address,@City,@PostalCode,@Country,@Phone,@Email,@CustomerType)
	INSERT INTO BusinessCustomers (CustomerID,CompanyName,NIP,Fax)
	VALUES (@CustomerID,@CompanyName,@NIP,@Fax)
	COMMIT TRANSACTION
END
GO
```

ChangeCustomerData

Procedura zmieniająca adres klienta, wykorzystywana jeśli klient zmieni swój adres.

```
CREATE OR ALTER PROCEDURE ChangeCustomerData
	@CustomerID int,
	@Address varchar(40),
	@City varchar(40),
	@PostalCode   varchar(40),
	@Country varchar(40),
	@Phone varchar(11),
	@Email varchar(40)
AS
BEGIN
	SET NOCOUNT ON;
	IF @Address IS NOT NULL
		UPDATE Customers
		SET Address=@Address
		WHERE CustomerID=@CustomerID
	IF @City IS NOT NULL
		UPDATE Customers
		SET City=@City
		WHERE CustomerID=@CustomerID
	IF @PostalCode IS NOT NULL
		UPDATE Customers
		SET PostalCode=@PostalCode
		WHERE CustomerID=@CustomerID
	IF @Country IS NOT NULL
		UPDATE Customers
		SET Country=@Country
		WHERE CustomerID=@CustomerID
	IF @Phone IS NOT NULL
		UPDATE Customers
		SET Phone=@Phone
		WHERE CustomerID=@CustomerID
	IF @Email IS NOT NULL
		UPDATE Customers
		SET Email=@Email
		WHERE CustomerID=@CustomerID
END
GO
```

CreateDiscountsData

Procedura tworząca rekord w tabeli CustomersDiscounts przechowujący rabaty klienta.

```
CREATE OR ALTER PROCEDURE CreateDiscountsData
	@CompanyID int,
	@CustomerID int
AS
BEGIN
	SET NOCOUNT ON;
	INSERT INTO CustomersDiscounts (CompanyID,CustomerID)
	VALUES (@CompanyID,@CustomerID)
END
GO
```

AddCompany

Procedura dodająca restauracje do bazy danych.

```
CREATE OR ALTER PROCEDURE AddCompany
	@CompanyName varchar(40),
	@FloorSpace int,
	@Address varchar(40),
	@City varchar(15),
	@Region varchar(20),
	@PostalCode varchar(10),
	@Phone varchar(20),
	@NIP varchar(11),
	@Fax varchar(20)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @CompanyID int=(SELECT ISNULL(MAX(CompanyID),1) FROM Company)
	INSERT INTO
Company(CompanyID,CompanyName,FloorSpace,Address,City,Region,PostalCode,Phone,NIP,Fax)
	VALUES(@CompanyID,@CompanyName,@FloorSpace,@Address,@City,@Region,@PostalCode,@
Phone,@NIP,@Fax)
	INSERT INTO DiscountsData(CompanyID)
	VALUES (@CompanyID)
END
GO
```

ChangeCompanyData

Procedura zmieniająca dane restauracji

```
CREATE OR ALTER PROCEDURE ChangeCompanyData
	@CompanyID int,
	@CompanyName varchar(40),
	@FloorSpace int,
	@ReservationMinAmount int,
	@Address varchar(40),
	@City varchar(15),
	@Region varchar(20),
	@PostalCode  varchar(10),
	@Country varchar(15),
	@Phone varchar(20),
	@NIP varchar (11),
	@Fax varchar (20)
AS
BEGIN
	SET NOCOUNT ON;
	IF @CompanyName IS NOT NULL
		UPDATE Company
```

```
		SET CompanyName=@CompanyName
		WHERE CompanyID=@CompanyID
	IF @FloorSpace IS NOT NULL
		UPDATE Company
		SET FloorSpace=@FloorSpace
		WHERE CompanyID=@CompanyID
	IF @ReservationMinAmount IS NOT NULL
		UPDATE Company
		SET ReservationMinAmount=@ReservationMinAmount
		WHERE CompanyID=@CompanyID
	IF @Address IS NOT NULL
		UPDATE Company
		SET Address=@Address
		WHERE CompanyID=@CompanyID
	IF @City IS NOT NULL
		UPDATE Company
		SET City=@City
		WHERE CompanyID=@CompanyID
	IF @Region IS NOT NULL
		UPDATE Company
		SET Region=@Region
		WHERE CompanyID=@CompanyID
	IF @PostalCode IS NOT NULL
		UPDATE Company
		SET PostalCode=@PostalCode
		WHERE CompanyID=@CompanyID
	IF @Country IS NOT NULL
		UPDATE Company
		SET Country=@Country
		WHERE CompanyID=@CompanyID
	IF @Phone IS NOT NULL
		UPDATE Company
		SET Phone=@Phone
		WHERE CompanyID=@CompanyID
	IF @NIP IS NOT NULL
		UPDATE Company
		SET NIP=@NIP
		WHERE CompanyID=@CompanyID
	IF @Fax IS NOT NULL
		UPDATE Company
		SET Fax=@Fax
		WHERE CompanyID=@CompanyID
END
GO
```

FireEmployee

Procedura zwalniająca danego pracownika.

```
CREATE OR ALTER PROCEDURE FireEmployee
	@EmployeeID int,
	@FireDate date
AS
BEGIN
	SET NOCOUNT ON;
	UPDATE Employees
	SET FireDate=@FireDate
	WHERE EmployeeID=@EmployeeID
END
GO
```

AddIngredientToCourse

Procedura dodająca składnik do danego dania.

```
CREATE or ALTER PROCEDURE AddIngredientToCourse
	@CourseID int,
	@IngredientID int
AS
BEGIN
	SET NOCOUNT ON;
	IF (SELECT COUNT(*) FROM [Contains] WHERE CourseID = @CourseID AND IngredientID
= @IngredientID) = 0
		INSERT INTO [Contains](CourseID,IngredientID)
		VALUES (@CourseID,@IngredientID)
	ELSE
		PRINT 'Danie już posiada ten składnik'
END
GO
```

DeleteIngredientFromCourse

Procedura usuwająca składnik z potrawy

```
CREATE OR ALTER PROCEDURE DeleteIngredientFromCourse
	@CourseID int,
	@IngredientID int
AS
BEGIN
	SET NOCOUNT ON;
	DELETE [Contains] WHERE CourseID=@CourseID AND IngredientID=@IngredientID
END
GO
```

AddCourseToMenu

Dodaje potrawę do menu. Jeśli potrawę można dodać, w przeciwnym wypadku zwracany jest komunikat błędu.

```
CREATE or ALTER PROCEDURE AddCourseToMenu
	@MenuID int,
	@CourseID int,
	@CategoryID int,
	@UnitPrice int
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @CompanyID int=(SELECT CompanyID FROM Menu WHERE MenuID=@MenuID)
	IF ([dbo].CanAddCourse(@CompanyID,@CourseID)=1)
		INSERT INTO MenuDetails(MenuID,CourseID,CategoryID,UnitPrice)
		VALUES (@MenuID,@CourseID,@CategoryID,@UnitPrice)
	ELSE
		THROW 50005, 'Cannot add this course', 1;
END
GO
```

AddDiscountsToOrder

Procedura wlicza rabaty w zamówienie. Procedura pozwala wybrać tylko te rabaty których klient chce użyć. Sprawdza czy klientowi przysługuje dany rabat i jeśli przysługuje mu to dodaje dany rabat to zamówienia. Jeśli dany rabat był rabatem jednorazowym to modyfikuje rabaty klienta tak, aby klient nie mógł drugi raz użyć tego samego rabatu. Procedura pozwala na dodanie różnie wielu rabatów, a niespełnianie warunków do przyznania jednego nie wpływa na inne rabaty.

```
CREATE OR ALTER PROCEDURE AddDiscountsToOrder
	@OrderID int,
	@R1Disc varchar(1),
	@ValidTillR2Disc varchar(1),
	@ValidTillR3Disc varchar(1),
	@FR1Disc varchar(1),
	@FK2Disc varchar(2)
AS
BEGIN
	DECLARE @CustomerID int = (SELECT [dbo].GetCustomerID(@OrderID))
	DECLARE @CompanyID int = (SELECT CompanyID FROM Orders WHERE OrderID =
@OrderID)
	DECLARE @CustomerType int = (SELECT CustomerType FROM Customers WHERE
CustomerID = @CustomerID)
	DECLARE @OrderDate datetime = (SELECT OrderID FROM Orders WHERE OrderID =
@OrderID)
	-- naliczanie rabatu za określoną ilość zamówień
	IF @R1Disc = 'Y' AND @CustomerType='I'
	BEGIN
		DECLARE @OrdersWithK1 int = (SELECT OrdersWithK1 FROM CustomersDiscounts
			WHERE CompanyID = @CompanyID AND CustomerID = @CustomerID)
		DECLARE @R1 decimal(3,2) = (SELECT R1 FROM DiscountsData WHERE CompanyID
= @CompanyID)
		DECLARE @Z1 decimal(3,2) = (SELECT Z1 FROM DiscountsData WHERE CompanyID
= @CompanyID)
		UPDATE Orders
		SET PercentageDiscount = PercentageDiscount + CAST(@OrdersWithK1/@Z1 AS
int)*@R1
		WHERE OrderID = @OrderID
	END
	--naliczanie rabatów czasowych dla indywidualnego klienta
	IF @ValidTillR2Disc = 'Y' AND @CustomerType='I'
	BEGIN
		DECLARE @R2 decimal(3,2) = (SELECT R2 FROM DiscountsData WHERE CompanyID
= @CompanyID)
		DECLARE @Date datetime = (SELECT ValidTillR2 FROM CustomersDiscounts
			WHERE CompanyID = @CompanyID AND CustomerID = @CustomerID)
		IF @OrderDate < @Date
		BEGIN
			UPDATE Orders
				SET PercentageDiscount = PercentageDiscount + @R2
				WHERE OrderID = @OrderID
			UPDATE CustomersDiscounts
				SET ValidTillR2 = @OrderDate
				WHERE CustomerID = @CustomerID AND CompanyID = @CompanyID
		END
		ELSE
			PRINT('Rabat R2 jest nieaktywny')
	END

	--naliczanie rabatów czasowych dla indywidualnego klienta
	IF @ValidTillR3Disc = 'Y' AND @CustomerType='I'
	BEGIN
		DECLARE @R3 decimal(3,2) = (SELECT R3 FROM DiscountsData WHERE CompanyID
= @CompanyID)
		DECLARE @Date2 datetime = (SELECT ValidTillR3 FROM CustomersDiscounts
			WHERE CompanyID = @CompanyID AND CustomerID = @CustomerID)
		IF @OrderDate < @Date2
		BEGIN
			UPDATE Orders
				SET PercentageDiscount = PercentageDiscount + @R3
				WHERE OrderID = @OrderID
```

```
                        UPDATE CustomersDiscounts
                                SET ValidTillR3 = @OrderDate
                                WHERE CustomerID = @CustomerID AND CompanyID = @CompanyID
                END
                ELSE
                        PRINT('Rabat R3 jest nieaktywny')
        END

        --naliczanie rabatu procentowego dla firm
        IF @FR1Disc = 'Y' AND @CustomerType = 'B'
        BEGIN
                DECLARE @FR1 decimal(3,2) = (SELECT FR1 FROM DiscountsData WHERE
CompanyID = @CompanyID)
                DECLARE @MonthsWithFK1 int = (SELECT MonthsWithFK1 FROM
CustomersDiscounts
                        WHERE CompanyID = @CompanyID AND CustomerID = @CustomerID)
                UPDATE Orders
                        SET PercentageDiscount = PercentageDiscount + @FR1*@MonthsWithFK1
                        WHERE OrderID = @OrderID
        END

        --naliczenie rabatu kwotowego dla firm
        IF @FR1Disc = 'Y' AND @CustomerType = 'B'
        BEGIN
                DECLARE @QuarterDiscount int = (SELECT QuarterDiscount FROM
CustomersDiscounts
                        WHERE CompanyID = @CompanyID AND CustomerID = @CustomerID)
                IF @QuarterDiscount < (SELECT [Sum] FROM OrdersData WHERE OrderID =
@OrderID)
                        SET @QuarterDiscount = (SELECT [Sum] FROM OrdersData WHERE
OrderID = @OrderID)
                UPDATE Orders
                        SET DisposableDiscount =
                                DisposableDiscount + @QuarterDiscount
                UPDATE CustomersDiscounts
                        SET QuarterDiscount = 0
                        WHERE CustomerID = @CustomerID AND CompanyID = @CompanyID
        END
END
GO
```

AddNewReservation

Procedura składająca rezerwację w określonej restauracji. Procedura uniemożliwia złożenie rezerwacji na owoce morza jeśli nie jest składana za późno i termin rezerwacji to czwartek-piątek-sobota.

```
CREATE OR ALTER PROCEDURE AddNewReservation
        @ReservationDate datetime,
        @NumberOfPeople int,
        @OrderID int,
        @CustomerID int,
        @CompanyID int,
        @EmployeeID int,
        @TakeAway varchar(1)
AS
BEGIN
        SET NOCOUNT ON;
        DECLARE @ReservationID int=(SELECT ISNULL(MAX(ReservationID),0) FROM
Reservations)
        --sprawdzenie warunków w owocami morza
```

```
	IF (@NumberOfPeople>=2 AND [dbo].ReservationConditions(@OrderID)=1 AND
[dbo].AvailablePlace(@CompanyID,@ReservationDate)>=@NumberOfPeople
	AND [dbo].ContainsSeaFoodOrder(@OrderID)=1 AND
DATEPART(weekday,@ReservationDate)>=4
	AND DATEDIFF(day,(SELECT OrderDate FROM Orders WHERE
OrderID=@OrderID),@ReservationDate)>=3+DATEPART(weekday,@ReservationDate)-4)
		INSERT INTO Reservations(ReservationID,OrderID,ReservationDate)
		VALUES(@ReservationID,@OrderID,@ReservationDate)
	ELSE
		THROW 50005, N'An errOR occurred', 1;
END
GO
```

SubmitReservation

Procedura zatwierdzająca zamówienie połączone z rezerwacją jeśli pracownik firmy uzna że można je zrealizować.

```
CREATE OR ALTER PROCEDURE SubmitReservation
	@ReservationID int,
	@SubmissionDate date
AS
BEGIN
	SET NOCOUNT ON;
	UPDATE Reservations
	SET SubmissionDate=@SubmissionDate
	WHERE ReservationID=@ReservationID
END
GO
```

AddTableToOrder

Dodaje stolik do zamówienia, sprawdzając czy przy stoliku jest odpowiednia ilość wolnych miejsc. Jeśli nie można dodać stolika zwraca błąd.

```
CREATE OR ALTER PROCEDURE AddTableToOrder
	@OrderID int,
	@CompanyID int,
	@TableID int,
	@OrderedChairs int
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @OrderDate datetime = (SELECT OrderDate FROM Orders WHERE OrderID =
@OrderID)
	DECLARE @IntervalID int = [dbo].GetRestaurantIntervalID(@CompanyID, @OrderDate)
	DECLARE @ChairsInTable int = (SELECT AvailableChairs FROM IntervalDetails WHERE
IntervalID = @IntervalID AND TableID = @TableID)
	DECLARE @OccupiedChairs int = (
		SELECT SUM(OccupiedChairs)
		FROM OrderedTables as OT
		INNER JOIN Orders as O
		ON O.OrderID = OT.OrderID
		WHERE TableID = @TableID AND IntervalID = @IntervalID AND O.OrderDate >
@OrderDate AND O.CompletionDate < @OrderDate AND O.Valid = 'Y')
	IF @OrderedChairs <= @ChairsInTable - @OccupiedChairs
		INSERT INTO OrderedTables(OrderID,IntervalID,TableID,OccupiedChairs)
		VALUES(@OrderID,@IntervalID,@TableID,@OccupiedChairs)
	ELSE
		THROW 50005, 'Not enought chairs', 1
END
GO
```

MenuCourses

Procedura wyświetlająca dania należace do danego menu.

```
CREATE OR ALTER PROCEDURE MenuCourses
	@MenuID int
AS
BEGIN
	SET NOCOUNT ON;
	SELECT CourseID
	FROM MenuDetails
	WHERE MenuID = @MenuID
END
GO
```

RestaurantCustomers

Procedura wyświetla ID klientów którzy złożyli zamówienie w danej restauracji.

```
CREATE OR ALTER PROCEDURE RestaurantCustomers
	@RestaurantID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT CustomerID
	FROM CustomersDiscounts
	WHERE CompanyID = @RestaurantID
END
GO
```

RestaurantOrdersInTimeInterval

Procedura wyświetlająca ID zamówień, które restauracja realizuje w określonym czasie.

```
CREATE OR ALTER PROCEDURE RestaurantOrdersInTimeInterval
	@RestaurantID int,
	@StartTime datetime,
	@EndTime datetime
AS
BEGIN
	SET NOCOUNT ON
	SELECT OrderID
	FROM Orders
	WHERE CompanyID = @RestaurantID and (
	(OrderDate > @StartTime and OrderDate < @EndTime) OR (CompletionDate >
@StartTime and CompletionDate < @EndTime))
END
GO
```

InvitedPeople

Procedura wyświetlająca gości powiązanych z danym zamówieniem.

```
CREATE OR ALTER PROCEDURE InvitedPeople
	@ReservationID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT *
	FROM ReservationDetails
	WHERE ReservationID = @ReservationID
END
GO
```

FutureOrdersWithIngredient

Procedura zwracająca przyszłe zamówienia w których znajduje się dany składnik.

```
CREATE OR ALTER PROCEDURE FutureOrdersWithIngredient
	@RestaurantID int,
	@IngredientID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT *
	FROM Orders as O
	INNER JOIN OrderDetails as OD
	on O.OrderID = OD.OrderID
	INNER JOIN Courses as C
	on C.CourseID = OD.OrderID
	INNER JOIN Ingredients as I
	on I.IngredientID = C.CourseID
	where O.OrderDate > GETDATE() and O.CompanyID = @RestaurantID and
I.IngredientID = @IngredientID
END
GO
```

OrderedCoursesStats

Procedura zwracająca ile razy dania był zamówione w określonym przedziale czasowym.

```
CREATE OR ALTER PROCEDURE OrderedCoursesStats
	@RestaurantID int,
	@StartTime datetime,
	@EndTime datetime = GETDATE
AS
BEGIN
	SET NOCOUNT ON
	SELECT OD.CourseID, sum(OD.Quantity) as TotalQuantity
	FROM OrderDetails as OD
	INNER JOIN Orders as O
	on O.OrderID = OD.OrderID
	WHERE O.CompanyID = @RestaurantID and O.OrderDate > @StartTime and O.OrderDate
< @EndTime
	GROUP BY OD.CourseID
END
GO
```

OrderedTablesStats

Procedura zwracająca ile razy dany stolik był rezerwowany w danym przedziale czasowym.

```
CREATE OR ALTER PROCEDURE OrderedTablesStats
	@RestaurantID int,
	@StartTime datetime,
	@EndTime datetime = GETDATE
AS
BEGIN
	SET NOCOUNT ON
	SELECT OD.TableID, count(*) as TotalTableReservation
	FROM OrderedTables as OD
	INNER JOIN Orders as O
	on O.OrderID = OD.OrderID
	where O.CompanyID = @RestaurantID and O.OrderDate > @StartTime and O.OrderDate
< @EndTime
	group by OD.TableID
END
GO
```

ShowCustomerData

Procedura wyświetlająca informacje o kliencie. Procedura bierze pod uwagę typ klienta i dostosowuję do niego zwracane dane.

```
CREATE OR ALTER PROCEDURE ShowCustomerData

	@CustomerID int
AS
BEGIN
	SET NOCOUNT ON
	IF (SELECT CustomerType FROM Customers WHERE CustomerID=@CustomerID)='I'
		SELECT
Address,City,PostalCode,Country,Phone,Email,FirstName,LastName,BirthDate,PESEL
		FROM Customers,IndividualCustomers
		WHERE Customers.CustomerID=@CustomerID AND
IndividualCustomers.CustomerID=@CustomerID
	ELSE
		SELECT Address,City,PostalCode,Country,Phone,Email,CompanyName,NIP,Fax
		FROM Customers,BusinessCustomers
		WHERE Customers.CustomerID=@CustomerID AND
BusinessCustomers.CustomerID=@CustomerID
END
GO
```

ShowOrderData

Procedura wyświetlająca dane dotyczące zamówienia, m.in. jego koszt a także przypisane rabaty.

```
CREATE OR ALTER PROCEDURE ShowOrderData
	@OrderID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT *
	FROM OrdersData
	WHERE OrderID = @OrderID
END
GO
```

WhenCourseCanBeOrder

Procedura zwracająca przedziały czasowe w których można zamówić określone danie w danej restauracji.

```
CREATE OR ALTER PROCEDURE WhenCourseCanBeOrdered
	@CourseID int,
	@CompanyID int
AS
BEGIN
	SET NOCOUNT ON
	SELECT InDate, OutDate
	FROM CoursesAvailability
	WHERE CourseID = @CourseID and CompanyID = @CompanyID
END
GO
```

ShowCustomerData2

Procedura wyświetla dane klienta, który złożył określone zamówienie @OrderID, w zależności od jego typu.

```
CREATE OR ALTER PROCEDURE ShowCustomerData
	@OrderID int
AS
BEGIN
	SET NOCOUNT ON
	DECLARE @CustomerID int=[dbo].GetCustomerID(@OrderID)
	IF (SELECT CustomerType FROM Customers WHERE CustomerID=@CustomerID)='I'
		SELECT
Address,City,PostalCode,Country,Phone,Email,FirstName,LastName,BirthDate,PESEL
		FROM Customers,IndividualCustomers
		WHERE Customers.CustomerID=@CustomerID AND
IndividualCustomers.CustomerID=@CustomerID
	ELSE
		SELECT Address,City,PostalCode,Country,Phone,Email,CompanyName,NIP,Fax
		FROM Customers,BusinessCustomers
		WHERE Customers.CustomerID=@CustomerID AND
BusinessCustomers.CustomerID=@CustomerID
END
GO
```

UpdateBusinessMonthlyDiscount

Procedura służąca do aktualizacji miesięcznych w danej restauracji. Zmienia ona wartości kolumny MonthWithFK1 z tabeli CustomersDiscounts dla klientów firmowych. Jeśli klient spełnił warunki podwyższenia kwoty rabatu to ilość miesięcy z rabatem zostaje zwiększona o 1. Jeśli klient nie spełnił warunków to ilość miesięcy zostaje wyzerowana.

```
CREATE OR ALTER PROCEDURE UpdateBusinessCustomersDiscount
	@CompanyID int,
	@MonthStart date
AS
BEGIN
	DECLARE @MonthEnd date = DATEADD(M,1,@MonthStart)
	DECLARE @ProfitableCustomers TABLE(CustomerID int) --klienci który zamówili za
odpowiednią kwotę
	DECLARE @MinSum money = (SELECT FK1 FROM DiscountsData WHERE CompanyID =
@CompanyID)
	DECLARE @MinOrders int = (SELECT FZ FROM DiscountsData WHERE CompanyID =
@CompanyID)
	INSERT INTO @ProfitableCustomers
		SELECT CustomerID
		FROM OrdersData
		WHERE CompanyID = @CompanyID AND OrderDate >= @MonthStart AND OrderDate
<= @MonthEnd
		GROUP BY CustomerID
		HAVING SUM([Sum]) >= @MinSum and COUNT(*) >= @MinOrders
	UPDATE CustomersDiscounts -- update dla zamawiających klientów
		SET MonthsWithFK1 = MonthsWithFK1+1
		WHERE CustomerID IN (SELECT CustomerID FROM @ProfitableCustomers)
	UPDATE CustomersDiscounts -- update dla zamawiających klientów
		SET MonthsWithFK1 = 0
		WHERE CustomerID NOT IN (SELECT CustomerID FROM @ProfitableCustomers)
END
GO
```

UpdateBusinessQuarterDisount

Procedura aktualizująca rabaty kwartalne dla klientów firmowych. Przyznaje ona rabat kwotowy w zależności od łącznej kwoty zamówionych dań w poprzednim kwartale. Jeśli klient nie spełnił warunków rabatu to jego obecny rabat – jeśli istnieje zostanie wyzerowany.

```
CREATE OR ALTER PROCEDURE UpdateBusinessCustomersDiscount
	@CompanyID int,
	@QuarerStart date
AS
BEGIN
	DECLARE @QuarerEnd date = DATEADD(M,1,@QuarerStart)
	DECLARE @ProfitableCustomers TABLE(CustomerID int, [Sum] money) --klienci który
zamówili za odpowiednią kwotę
	DECLARE @MinSum money = (SELECT FK2 FROM DiscountsData WHERE CompanyID =
@CompanyID)
	DECLARE @FR2 decimal(3,2) = (SELECT FR2 FROM DiscountsData WHERE CompanyID =
@CompanyID)
	INSERT INTO @ProfitableCustomers
		SELECT CustomerID, SUM([Sum])
		FROM OrdersData
		WHERE CompanyID = @CompanyID AND OrderDate >= @QuarerStart AND OrderDate
<= @QuarerEnd
		GROUP BY CustomerID
		HAVING SUM([Sum]) >= @MinSum
	UPDATE CustomersDiscounts -- update dla zamawiających klientów
		SET QuarterDiscount = [Sum]*@FR2
		FROM CustomersDiscounts as CD
		LEFT OUTER JOIN @ProfitableCustomers as PC
		ON PC.CustomerID = CD.CustomerID
		WHERE CD.CustomerID IN (SELECT CustomerID FROM @ProfitableCustomers)
	UPDATE CustomersDiscounts -- update dla zamawiających klientów
		SET QuarterDiscount = 0
		WHERE CustomerID NOT IN (SELECT CustomerID FROM @ProfitableCustomers)
END
GO
```

ShowCustomerOrdersData

Procedura wyświetlająca dane dotyczące zamówień klienta w danej restauracji.

```
CREATE OR ALTER PROCEDURE ShowCustomerOrdersData
	@CustomerID int,
	@CompanyID int
AS
BEGIN
	SET NOCOUNT ON;
	SELECT OrderID, OrderDate, Sum,DisposableDiscount,PercentageDiscount, Valid
	FROM OrdersData
	WHERE CustomerID=@CustomerID AND CompanyID=@CompanyID
END
GO
```

Triggery

UpdateCustomerDiscounts

Trigger włączający się kiedy pracownik ustali datę końca zamówienia, czyli moment w którym klient płaci za zamówienie. Aktualizuje on rabaty dla klientów indywidualnych sprawdzając wszystkie warunki potrzebne występujące w tych rabatach.

```
CREATE OR ALTER TRIGGER UptadeCustomerDiscounts
ON Orders
AFTER UPDATE
AS
BEGIN
	IF UPDATE(CompletionDate)
	BEGIN
		DECLARE @OrderID int = (SELECT OrderID FROM inserted)
		DECLARE @CompanyID int = (SELECT CompanyID FROM Orders WHERE
OrderID=@OrderID)
		DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE
OrderID=@OrderID)
		IF ((SELECT CustomerID FROM CustomersDiscounts WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID) IS NULL AND @CustomerID IS NOT NULL)
			INSERT INTO CustomersDiscounts (CompanyID,CustomerID) VALUES
(@CompanyID,@CustomerID)
		IF (SELECT CustomerType FROM Customers WHERE CustomerID=@CustomerID) =
'I'
		BEGIN
			DECLARE @Cost money = (SELECT [Sum] FROM OrdersData WHERE
OrderID=@OrderID)
			IF (@Cost > (SELECT K1 FROM DiscountsData WHERE
CompanyID=@CompanyID))
				UPDATE CustomersDiscounts -- update zamówień za okreœlon¹
kwotê
					SET OrdersWithK1 = OrdersWithK1+1
					WHERE CompanyID = @CompanyID AND CustomerID =
@CustomerID

			DECLARE @ValidTillR2 datetime =
				(SELECT ValidTillR2 FROM CustomersDiscounts WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID)

			DECLARE @CostFromLastR2Discount money
			IF (@ValidTillR2 is null)
				SET @CostFromLastR2Discount =
					(SELECT SUM([Sum]) FROM OrdersData WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID)
			ELSE
				SET @CostFromLastR2Discount =
					(SELECT SUM([Sum]) FROM OrdersData WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID AND OrderDate > @ValidTillR2)
			DECLARE @K2 money = (SELECT K2 FROM DiscountsData WHERE
CompanyID=@CompanyID)
			IF (@K2 <= @CostFromLastR2Discount)
			BEGIN
				DECLARE @R2 int = (SELECT R2 FROM DiscountsData WHERE
CompanyID=@CompanyID)
				UPDATE CustomersDiscounts -- update 1 rabatu czasowego
					SET ValidTillR2 = DATEADD(day,@R2,GETDATE())
					WHERE CompanyID = @CompanyID AND CustomerID =
@CustomerID
```

```
                        END
                                                DECLARE @ValidTillR3 datetime =
                                (SELECT ValidTillR2 FROM CustomersDiscounts WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID)

                        DECLARE @CostFromLastR3Discount money
                        IF (@ValidTillR3 is null)
                                SET @CostFromLastR3Discount =
                                        (SELECT SUM([Sum]) FROM OrdersData WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID)
                        ELSE
                                SET @CostFromLastR2Discount =
                                        (SELECT SUM([Sum]) FROM OrdersData WHERE
CustomerID=@CustomerID AND CompanyID=@CompanyID AND OrderDate > @ValidTillR3)
                        DECLARE @K3 money = (SELECT K3 FROM DiscountsData WHERE
CompanyID=@CompanyID)
                        IF (@K3 <= @CostFromLastR3Discount)
                        BEGIN
                                DECLARE @R3 int = (SELECT R3 FROM DiscountsData WHERE
CompanyID=@CompanyID)
                                UPDATE CustomersDiscounts -- update 2 rabatu czasowego
                                        SET ValidTillR3 = DATEADD(day,@R3,GETDATE())
                                        WHERE CompanyID = @CompanyID AND CustomerID =
@CustomerID
                        END
                END
        END
END
GO
```

Informacje dotyczące wygenerowanych danych

Dane wypełniające tabele w bazie danych zostały wygenerowane za pomocą skryptów tworzących odpowiednie polecenia, napisanych w języku Python oraz częściowo z wykorzystaniem programu SQL Data Generator 4.

Wygenerowane zostały następujące dane:
- 2000 stałych klientów restauracji w raz z ich danymi, w tym 1000 klientów indywidualnych oraz 1000 klientów biznesowych
- 11 kategorii dań
- 50 restauracji wraz z ich danymi oraz informacjami na temat sposobu przyznawania zniżek
-65 dań
- 1000 składników
-1024 pracowników, wraz z ich danymi
-200000 zamówień, w tym 30000 rezerwacji
- 1250 stolików
-4500 układów stolików w restauracjach
-17038 różnych menu dla restauracji
-323 rekordy w tabeli przechowującej informacje o zawartości dań
-3871 rekordów w tabeli przechowującej informacje o zniżkach klientów
-112500 rekordów w tabeli przechowującej szczegółowe informacje dotyczące układu stolików w restauracji
-274035 rekordów w tabeli przechowującej szczegółowe informacje na temat dań znajdujących się w menu
- 199000 rekordów w tabeli przechowującej szczegółowe informacje na temat zamówień
- 38085 rekordów w tabeli przechowującej szczegółowe informacje na temat rezerwowanych stolików

-130000 rekordów w tabeli przechowującej szczegółowe informacje o uczestnikach rezerwacji

Uprawnienia do danych

Uprawnienia do wyświetlania określonych danych w tabelach zależą od pełnionych ról. Wyróżniamy:

1) Manager
2) Kelner
3) Klient

Manager

Ma dostęp do danych we wszystkich tabelach, dotyczących restauracji, w której pracuje.

Kelner

Ma dostęp do:

- Zamówień, które obsługuje
- Podglądu na Menu oraz kategorie dań
- Podglądu układu stolików
- Do rabatów oferowanych przez restaurację, w której pracuje
- Do danych oraz rabatów stałych klientów restauracji, w której pracuje

Klient

Ma dostęp do:

- Podglądu na Menu oraz kategorie dań
- Podgląd na składniki z których zrobione jest danie
- Podgląd na zniżki oferowane przez restauracje
- Formularza składania zamówienia