Adam Biśta, Krzysztof Wysocki,
Wiktor Wilkusz

Podstawy Baz Danych - Projekt 2022 / 2023

Spis Treści

1. Wstęp
    I. Temat projektu
    II. Opis projektu
2. Schemat bazy danych
3. Tabele (warunki integralnościowe)
    I. Companies
    II. CompanyEmployees
    III. CompanyReservationParticipants
    IV. CustomersPersonalData
    V. Customers
    VI. IndividualCustomers
    VII. DiningTables
    VIII. Invoices
    IX. Menu
    X. MenuDetails
    XI. OrderDetails
    XII. Orders
    XIII. PaymentMethod
    XIV. ProductIngredients
    XV. IngredientsWarehouse
    XVI. Products
    XVII. ProductPrices
    XVIII. Categories
    XIX. Reservations
    XX. RestaurantEmployees
    XXI. EmployeesSalary
    XXII. Takeaway
    XXIII. VariablesData
    XXIV. TempDiscount
4. Widoki
    I. CurrentMenu
    II. AvailableProducts
    III. NotAvailableProducts
    IV. NotAvailableIngredients
    V. NotPaidOrders
    VI. TodayReservations
    VII. OrdersPendingForConfirmation
    VIII. OrderDetailsView
    IX. TotalOrdersProductsPricesReport
    X. AverageSalaryOfRestaurantEmployee

XI. FiveBestEmployees
XII. TotalProductsSales
XIII. TotalCategoreisSales
XIV. AvailableTables
XV. TotalReservationReport
XVI. CurrentMenuSalesStatsView
XVII. TotalCustomersDiscountsView
XVIII. OrderStatisticsView

5. Procedury
    I. AddMenu
    II. RemoveMenu
    III. AddIngredientToWarehouse
    IV. RemoveIngredientFromWarehouse
    V. AddIngredientToProduct
    VI. RemoveIngredientFromProduct
    VII. AddProductToMenu
    VIII. RemoveProductFromMenu
    IX. AddProductToOrder
    X. RemoveProductFromOrder
    XI. AddCustomerProcedure
    XII. RemoveCustomerProcedure
    XIII. updateDiscountProcedure
    XIV. AddReservationProcedure
    XV. RemoveReservationProcedure

6. Funkcje
    I. GetIngredientsForProduct
    II. GetProductsFromCategory
    III. GetProductsFromMenu
    IV. GetCurrentEmployeeSalary
    V. GetCurrentAverageSalaryForOccupation
    VI. GetTotalProductsAndAveragePriceOfMenu
    VII. GetHighestSalaryForEmployee
    VIII. RemainingDaysForMenu
    IX. RemainingFreeSeats
    X. CanAccommodateCustomers
    XI. GetDetailsOfOrder
    XII. GetOrdersAboveValue
    XIII. GetValueOfOrdersOnDay
    XIV. GetValueOfOrdersOnMonth
    XV. GetValueOfOrder
    XVI. Invoice
    XVII. CollectiveInvoice

7. Triggery

I. AddMenuOneDayInAdvance
II. CorrectSeaFoodOrderDateTrigger
III. CheckReservationSeatsTrigger
IV. CheckReservationCapacityTrigger
V. CheckDiscountAviabilityTrigger
VI. CheckIndividualReservationAviabilityTrigger

8. Indeksy
I. RestaurantEmployees_RestaurantEmployeeID_Index
II. Products_ProductID_Index
III. Menu_MenuID_Index
IV. MenuDetails_MenuID_Index
V. Orders_OrderID_Index
VI. OrderDetails_OrderID_Index
VII. Customers_CustomerID_Index

9. Role i uprawnienia
VI. Role na tabele
VII. Role na widoki
VIII. Role na funkcje
IX. Role na procedury
X. Dane które są dostępne tylko dla: Admin, Owner:

10. Przykładowe dane

1. Wstęp

I. Temat projektu

Tematem projektu jest system bazy danych wspomagający działalność firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.

II. Opis projektu

W ofercie jest żywność (np. pizza, owoce morza, spaghetti) oraz napoje bezalkoholowe (np. lemoniada, woda, sok pomarańczowy). Usługi świadczone są na miejscu oraz na wynos. Zamówienie na wynos może być zlecone na miejscu lub z wyprzedzeniem za pomocą internetowego formularza. Firma dysponuje ograniczoną liczbą stolików, w tym miejsc siedzących. Istnieje możliwość wcześniejszej rezerwacji stolika dla co najmniej dwóch osób.

Klientami są osoby indywidualne oraz firmy . Istnieje możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc.

Menu ustalane jest co najmniej dziennym wyprzedzeniem. W firmie panuje zasada, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie.

W dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Z uwagi na indywidualny import takie zamówienie winno być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.

Internetowy formularz umożliwia klientowi indywidualnemu rezerwację stolika, przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu, przy minimalnej wartości zamówienia *WZ*, w przypadku klientów, którzy dokonali wcześniej co najmniej *WK* zamówień. Informacja wraz z potwierdzeniem zamówienia oraz wskazaniem stolika wysyłana jest po akceptacji przez obsługę.

Internetowy formularz umożliwia także rezerwację stolików dla firm, w dwóch opcjach: rezerwacji stolików na firmę i/lub rezerwację stolików dla konkretnych pracowników firmy.

System umożliwia realizację programów rabatowych dla klientów indywidualnych:

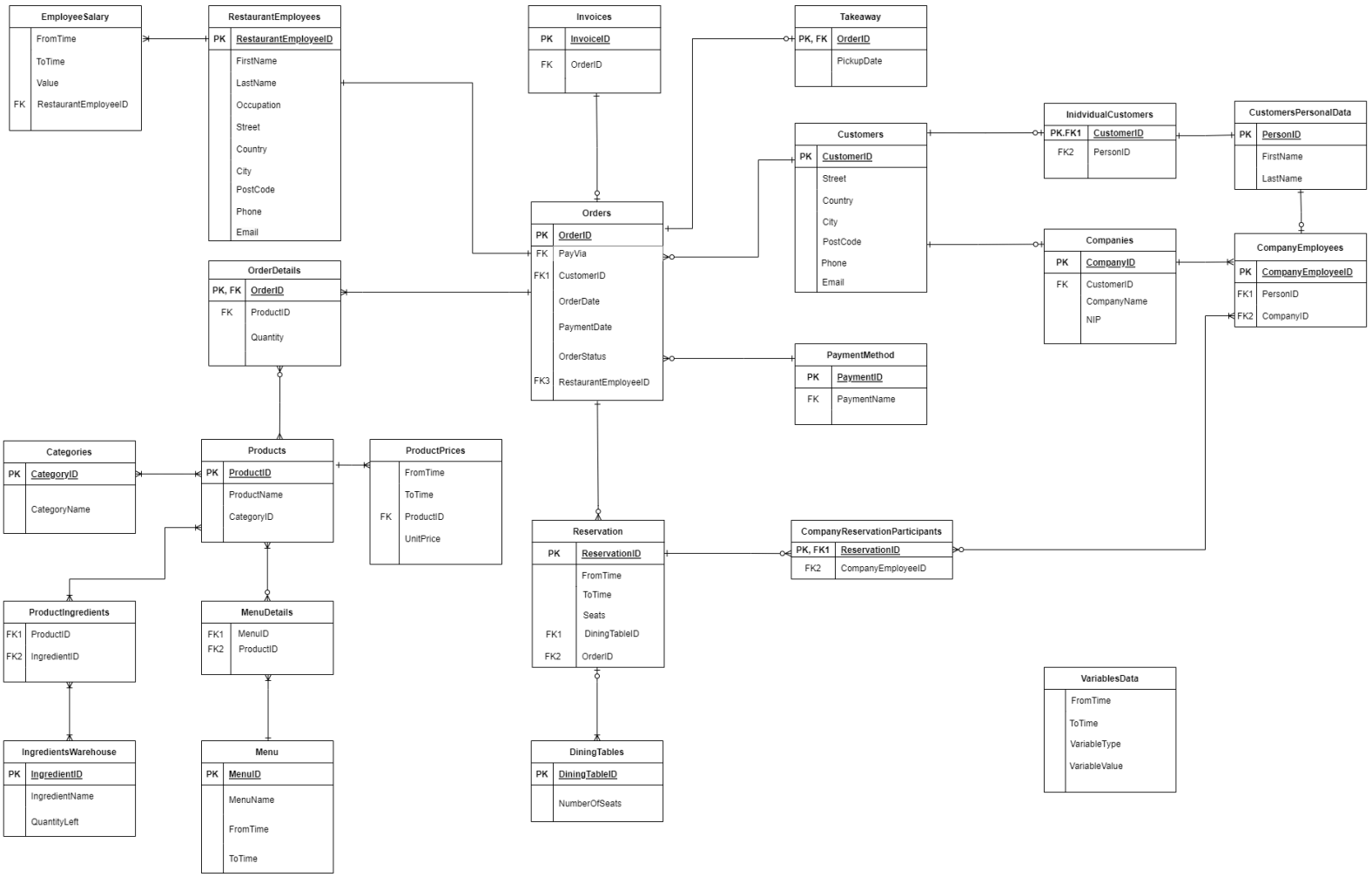
- Po realizacji ustalonej liczby zamówień *Z*1 za co najmniej określoną kwotę *K*1: *R*1% zniżki na wszystkie zamówienia;
- Po realizacji zamówień za łączną kwotę *K*2: jednorazowa zniżka *R*2% na zamówienia złożone przez D1 dni, począwszy od dnia przyznania zniżki .

Zniżki nie łączą się.

System umożliwia generowanie raportów miesięcznych i tygodniowych, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia – dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień.

2. Schemat bazy danych

(Dostępny również w załączniku)



3. Tabele (warunki integralnościowe)

I. Companies

Tabela przechowująca informacje o firmach - klientach.

CompanyID - unikalny numer identyfikacyjny firmy
CustomerID - unikalny numer identyfikacyjny klienta
CompanyName - nazwa firmy
NIP - numer NIP firmy

```
CREATE TABLE Companies
(
   CompanyID   int    NOT NULL,
   CustomerID  int    NOT NULL,
```

```
    CompanyName varchar(64) NOT NULL,
    NIP         char(10)    NOT NULL,
    CONSTRAINT Companies_pk PRIMARY KEY (CompanyID)
);
```

II. CompanyEmployees

Tabela przechowująca informacje o pracownikach firmy - klienta.

CompanyEmployeeID - unikalny numer identyfikacyjny pracownika
PersonID - unikalny numer identyfikacyjny osoby
CompanyID - unikalny numer identyfikacyjny firmy, w której pracownik jest zatrudniony

```
CREATE TABLE CompanyEmployees
(
    CompanyEmployeeID int NOT NULL,
    PersonID          int NOT NULL,
    CompanyID         int NOT NULL,
    CONSTRAINT CompanyEmployees_pk PRIMARY KEY (CompanyEmployeeID)
);
```

III. CompanyReservationParticipants

Tabela przechowująca informacje o pracownikach firmy - klienta, dla których dokonano rezerwacji stolika

ReservationID - unikalny numer identyfikacyjny rezerwacji
CompanyEmployeeID - unikalny numer identyfikacyjny pracownika

```
CREATE TABLE CompanyReservationParticipants
(
    ReservationID     int NOT NULL,
    CompanyEmployeeID int NOT NULL,
);
```

IV. CustomersPersonalData

Tabela przechowująca dane osobowe klientów indywidualnych oraz pracowników firm - klientów.

PersonID - unikalny numer identyfikacyjny osoby

FirstName - imię osoby
LastName - nazwisko osoby

```
        CREATE TABLE CustomersPersonalData
(
   PersonID  int    NOT NULL,
   FirstName varchar(64) NOT NULL,
   LastName  varchar(64) NOT NULL,
   CONSTRAINT FirstName_CustomersPersonalData_c CHECK (FirstName LIKE
'[A-Z]%'),
   CONSTRAINT LastName_CustomersPersonalData_c CHECK (LastName LIKE
'[A-Z]%'),
   CONSTRAINT CustomersPersonalData_pk PRIMARY KEY (PersonID)
);
```

V. Customers

Tabela przechowująca informacje o klientach restauracji.

CustomerID - unikalny numer identyfikacyjny klienta
Street, **Country**, **City**, **PostCode**, **Phone**, **Email** - dane adresowe / kontaktowe klienta

```
CREATE TABLE Customers
(
   CustomerID int    NOT NULL,
   Street     varchar(64) NOT NULL,
   Country    varchar(64) NOT NULL,
   City       varchar(64) NOT NULL,
   PostCode   varchar(16) NOT NULL,
   Phone      varchar(16) NOT NULL,
   Email      varchar(64) NOT NULL,
   CONSTRAINT Country_Customers_c CHECK (Country LIKE '[A-Z]%'),
   CONSTRAINT City_Customers_c CHECK (City LIKE '[A-Z]%'),
   CONSTRAINT Street_Customers_c CHECK (Street LIKE '[A-Z]%'),
   CONSTRAINT Customers_pk PRIMARY KEY (CustomerID)
);
```

VI. IndividualCustomers

Tabela przypisująca dane osobowe do klientów indywidualnych.

CustomerID - unikalny numer identyfikacyjny klienta
PersonID - unikalny numer identyfikacyjny osoby

```
CREATE TABLE IndividualCustomers
(
   CustomerID int NOT NULL,
   PersonID   int NOT NULL,
   CONSTRAINT IndividualCustomers_pk PRIMARY KEY (CustomerID)
);
```

VII. DiningTables

Tabela przechowująca informacje o stolikach.

DiningTableID - unikalny numer identyfikacyjny stolika
NumberOfSeats - liczba miejsc przy stoliku

```
CREATE TABLE DiningTables
(
   DiningTableID int NOT NULL,
   NumberOfSeats int NOT NULL,
   CONSTRAINT NumberOfSeats_DiningTables_c CHECK (NumberOfSeats > 0),
   CONSTRAINT DiningTables_pk PRIMARY KEY (DiningTableID)
);
```

VIII. Invoices

Tabela przechowująca informacje o fakturach.

InvoiceID - unikalny numer identyfikacyjny faktury
OrderID - unikalny numer identyfikacyjny zamówienia

```
CREATE TABLE Invoices
(
   InvoiceID int NOT NULL,
   OrderID   int NOT NULL,
   CONSTRAINT Invoices_pk PRIMARY KEY (InvoiceID)
);
```

IX. Menu

Tabela przechowująca informacje o menu dostępnych w danym przedziale czasu.

MenuID - unikalny numer identyfikacyjny menu
MenuName - nazwa menu
FromTime - data rozpoczęcia obowiązywania menu
ToTime - data zakończenia obowiązywania menu. Jeśli przechowuje wartość NULL, oznacza to, że dane menu obowiązuje do chwili obecnej

```
CREATE TABLE Menu
(
   MenuID   int    NOT NULL,
   MenuName varchar(64) NOT NULL,
   FromTime datetime    NOT NULL,
   ToTime   datetime    NULL DEFAULT NULL,
   CONSTRAINT Proper_Dates_Menu_c CHECK (FromTime <= ToTime OR ToTime IS
NULL),
   CONSTRAINT Menu_pk PRIMARY KEY (MenuID)
);
```

X. MenuDetails

Tabela przechowująca informacje o produktach w menu.

MenuID - unikalny numer identyfikacyjny menu
ProductID - unikalny numer identyfikacyjny produktu

```
CREATE TABLE MenuDetails
(
   MenuID    int NOT NULL,
   ProductID int NOT NULL,
);
```

XI. OrderDetails

Tabela przechowująca informacje o zawartości zamówienia.

OrderID - unikalny numer identyfikacyjny zamówienia
ProductID - unikalny numer identyfikacyjny produktu
Quantity - liczba zamówionych sztuk danego produktu

```
CREATE TABLE OrderDetails
(
   OrderID   int NOT NULL,
```

```
   ProductID int NOT NULL,
   Quantity  int     NOT NULL,
   CONSTRAINT Quantity_OrderDetails_c CHECK (Quantity >= 0),
);
```

XII. Orders

Tabela przechowująca informacje o zamówieniach

OrderID - unikalny numer identyfikacyjny zamówienia
CustomerID - unikalny numer identyfikacyjny klienta
OrderDate - data dokonania zamówienia
CollectDate - wybrana przez klienta data, na którą zamówienie ma być gotowe
PaymentDate - data dokonania płatności. Jeśli przechowuje wartość NULL, oznacza to, że nadal nie dokonano płatności
PayVia - wybrana metoda płatności
OrderStatus - status zamówienia
RestaurantEmployeeID - unikalny numer identyfikacyjny pracownika restauracji
DiscountPercent - zniżka przyznana zamówieniu, wyrażona w procentach
.

```
CREATE TABLE Orders
(
    OrderID              int     NOT NULL,
    CustomerID           int     NOT NULL,
    OrderDate            datetime    NOT NULL,
    CollectDate          datetime   NULL,
    PaymentDate          datetime    NULL DEFAULT NULL,
    PayVia               int     NULL,
    OrderStatus          varchar(64) NOT NULL,
    RestaurantEmployeeID int     NOT NULL,
    DiscountPercent      int     NOT NULL DEFAULT 0,
    CONSTRAINT Orders_pk PRIMARY KEY (OrderID)
);
```

XIII. PaymentMethod

Tabela przechowująca informacje o dostępnych metodach płatności.

PaymentID - unikalny numer identyfikacyjny metody płatności
PaymentName - nazwa metody płatności

```
CREATE TABLE PaymentMethod
(
    PaymentID   int    NOT NULL,
    PaymentName varchar(64) NOT NULL,
    CONSTRAINT PaymentMethod_pk PRIMARY KEY (PaymentID)
)
```

XIV. ProductIngredients

Tabela przypisująca składniki do produktów.

ProductID - unikalny numer identyfikacyjny produktu
IngredientID - unikalny numer identyfikacyjny składnika

```
CREATE TABLE ProductIngredients
(
    ProductID    int NOT NULL,
    IngredientID int NOT NULL,
);
```

XV. IngredientsWarehouse

Tabela przechowująca informacje o składnikach produktów.

IngredientID - unikalny numer identyfikacyjny składnika
IngredientName - nazwa składnika
QuantityLeft - dostępna liczba składnika w magazynie

```
CREATE TABLE IngredientsWarehouse
(
    IngredientID   int    NOT NULL,
    IngredientName varchar(64) NOT NULL,
    QuantityLeft   int        NOT NULL,
    CONSTRAINT QuantityLeft_IngredientsWarehouse_c CHECK (QuantityLeft >= 0),
    CONSTRAINT IngredientsWarehouse_pk PRIMARY KEY (IngredientID)
);
```

XVI. Products

Tabela przechowująca informacje o produktach.

ProductID - unikalny numer identyfikacyjny produktu
ProductName - nazwa produktu
CategoryID - unikalny numer identyfikacyjny kategorii, do której należy produkt

```
CREATE TABLE Products
(
    ProductID   int    NOT NULL,
    ProductName varchar(64) NOT NULL,
    CategoryID  int    NOT NULL,
    CONSTRAINT Products_pk PRIMARY KEY (ProductID)
);
```

XVII. ProductPrices

Tabela przechowująca informacje o cenach produktów w danym przedziale czasu.

ProductID - unikalny numer identyfikacyjny produktu
FromTime - data rozpoczęcia obowiązywania danej ceny
ToTime - data zakończenia obowiązywania ceny. Jeśli przechowuje wartość NULL, oznacza to, że dana cena obowiązuje do chwili obecnej
UnitPrice - cena jednostkowa produktu

```
CREATE TABLE ProductPrices
(
    ProductID int NOT NULL,
    FromTime  datetime NOT NULL,
    ToTime    datetime NULL DEFAULT NULL,
    UnitPrice int      NOT NULL,
    CONSTRAINT Proper_Dates_ProductPrices_c CHECK (FromTime <= ToTime OR
ToTime IS NULL),
    CONSTRAINT UnitPrice_ProductPrices_c CHECK (UnitPrice >= 0),
);
```

XVIII. Categories

Tabela przechowująca informacje kategoriach produktów.

CategoryID - unikalny numer identyfikacyjny kategorii
CategoryName - nazwa kategorii

```
CREATE TABLE Categories
```

```
(
    CategoryID   int    NOT NULL,
    CategoryName varchar(64) NOT NULL,
    CONSTRAINT Categories_pk PRIMARY KEY (CategoryID)
);
```

XIX. Reservations

Tabela przechowująca informacje o rezerwacjach stolików.

ReservationID - unikalny numer identyfikacyjny rezerwacji
FromTime - data rozpoczęcia obowiązywania rezerwacji
ToTime - data zakończenia obowiązywania rezerwacji
Seats - liczba zarezerwowanych miejsc
DiningTableID - unikalny numer identyfikacyjny zarezerwowanego stolika
OrderID - unikalny numer identyfikacyjny zamówienia

```
CREATE TABLE Reservations
(
    ReservationID int NOT NULL,
    FromTime      datetime NOT NULL,
    ToTime        datetime NOT NULL,
    Seats         int      NOT NULL,
    DiningTableID int NOT NULL,
    OrderID       int NOT NULL,
    CONSTRAINT Seats_Reservation_c CHECK (Seats <= 40 AND Seats > 0),
    CONSTRAINT Proper_Dates_Reservation_c CHECK (FromTime < ToTime),
    CONSTRAINT Reservation_pk PRIMARY KEY (ReservationID)
);
```

XX. RestaurantEmployees

Tabela przechowująca informacje o pracownikach restauracji.

RestaurantEmployeeID - unikalny numer identyfikacyjny pracownika restauracji
FirstName - imię pracownika
LastName - nazwisko pracownika
Occupation - stanowisko pracownika
Street, **Country**, **City**, **PostCode**, **Phone**, **Email** - dane adresowe / kontaktowe pracownika

```
CREATE TABLE RestaurantEmployees
```

```
(
    RestaurantEmployeeID int    NOT NULL,
    FirstName           varchar(64) NOT NULL,
    LastName            varchar(64) NOT NULL,
    Occupation          varchar(64) NOT NULL,
    Street             varchar(64) NOT NULL,
    Country             varchar(64) NOT NULL,
    City               varchar(64) NOT NULL,
    PostCode            varchar(16) NOT NULL,
    Phone              char(9)     NOT NULL,
    Email              varchar(64) NOT NULL,
    CONSTRAINT City_RestaurantEmployees_c CHECK ((City LIKE '[A-Z]%')),
    CONSTRAINT Country_RestaurantEmployees_c CHECK ((Country LIKE '[A-Z]%')),
    CONSTRAINT Street_RestaurantEmployees_c CHECK ((Street LIKE '[A-Z]%')),
    CONSTRAINT Name_Validation_RestaurantEmployees_c CHECK ((FirstName LIKE
'[A-Z]%') AND (LastName LIKE '[A-Z]%')),
    CONSTRAINT RestaurantEmployees_pk PRIMARY KEY (RestaurantEmployeeID)
);
```

XXI. EmployeesSalary

Tabela przechowująca informacje o płacy pracowników restauracji w danym przedziale czasu.

RestaurantEmployeeID - unikalny numer identyfikacyjny pracownika restauracji
FromTime - data rozpoczęcia obowiązywania płacy
ToTime - data zakończenia obowiązywania płacy. Jeśli przechowuje wartość NULL, oznacza to, że dana płaca obowiązuje do chwili obecnej
Salary - płaca pracownika

```
CREATE TABLE EmployeesSalary
(
    RestaurantEmployeeID int NOT NULL,
    FromTime            datetime NOT NULL,
    ToTime             datetime NULL DEFAULT NULL,
    Salary             int      NOT NULL,
    CONSTRAINT Salary_EmployeesSalary_c CHECK (Salary >= 0),
    CONSTRAINT Proper_Dates_EmployeesSalary_c CHECK (FromTime <= ToTime OR
ToTime IS NULL),
);
```

XXII. Takeaway

Tabela przechowująca informacje o zamówieniach na wynos.

OrderID - unikalny numer identyfikacyjny zamówienia
PickupDate - wybrana data odbioru zamówienia. Jeśli przechowuje wartość NULL - zamówienie powinno zostać wydane jak najszybciej.

```
CREATE TABLE Takeaway
(
   OrderID    int NOT NULL,
   PickupDate datetime NULL DEFAULT NULL,
   CONSTRAINT Takeaway_pk PRIMARY KEY (OrderID)
);
```

XXIII. VariablesData

Tabela przechowująca zmienne związane z obliczaniem zniżek

FromTime - data przypisania zmiennej
ToTime - maksymalna data kiedy zmienna jest aktywna
VariableType - rodzaj zmiennej (K1,D1,R1…)
VariableValue - wartość zmiennej

```
CREATE TABLE VariablesData
(
   FromTime      datetime   NOT NULL,
   ToTime        datetime   NULL DEFAULT NULL,
   VariableType  varchar(3) NOT NULL,
   VariableValue int        NOT NULL,
   CONSTRAINT Proper_Dates_VariablesData_c CHECK (FromTime <= ToTime OR
ToTime IS NULL),
   CONSTRAINT VariableValue_VariablesData_c CHECK (VariablesData.VariableValue
>= 0)
);
```

XXIV. TempDiscount

Tabela przechowująca zmienne związane z obliczaniem zniżek

FromTime - data aktywacji zniżki
ToTime - czas do kiedy zniżka jest aktywna, NULL = bezterminowo
CustomerID - id klienta na którego została nałożona zniżka
DicountPercent - wartość zmiennej wyrażonej w procentach

```
CREATE TABLE TempDiscount
(
```

```
    CustomerID       int        NOT NULL,
    FromTime         datetime   NOT NULL,
    ToTime           datetime   NULL DEFAULT NULL,
    DiscountPercent  int        NOT NULL DEFAULT 0,
    CONSTRAINT Proper_Dates_VariablesData_c CHECK (FromTime <= ToTime OR
ToTime IS NULL),
    CONSTRAINT TempDiscount_DiscountPercent_c CHECK
(TempDiscount.DiscountPercent >= 0)
);
```

4. Widoki

I. CurrentMenu

Widok wyświetlający produkty dostępne w obecnie obowiązującym menu.

```
CREATE VIEW Current_Menu_View AS
SELECT Products.ProductID ,Products.ProductName, ProductPrices.UnitPrice
FROM MenuDetails
      JOIN Products ON Products.ProductID = MenuDetails.ProductID
      JOIN ProductPrices ON ProductPrices.ProductID = Products.ProductID
WHERE MenuID = (SELECT MenuID
          FROM Menu
          WHERE ToTime IS NULL)
AND ProductPrices.ToTime IS NULL
```

II. AvailableProducts

Widok wyświetlający obecnie dostępne produkty.

```
CREATE VIEW Available_Products_View AS
SELECT Products.ProductID, ProductName
from ProductIngredients
      JOIN IngredientsWarehouse ON ProductIngredients.IngredientID =
IngredientsWarehouse.IngredientID
      JOIN Products ON ProductIngredients.ProductID = Products.ProductID
GROUP BY Products.ProductID, ProductName
HAVING MIN(QuantityLeft) > 0
```

III. NotAvailableProducts

Widok wyświetlający obecnie niedostępne produkty.

```
CREATE VIEW Not_Available_Products_View AS
SELECT Products.ProductID, ProductName
from ProductIngredients
	JOIN IngredientsWarehouse ON ProductIngredients.IngredientID =
IngredientsWarehouse.IngredientID
	JOIN Products ON ProductIngredients.ProductID = Products.ProductID
GROUP BY Products.ProductID, ProductName
HAVING MIN(QuantityLeft) = 0
```

IV. NotAvailableIngredients

Widok wyświetlający obecnie niedostępne składniki.

```
CREATE VIEW Not_Available_Ingredients_View AS
SELECT IngredientID, IngredientName
from IngredientsWarehouse
WHERE QuantityLeft = 0
```

V. NotPaidOrders

Widok wyświetlający nieopłacone zamówienia.

```
CREATE VIEW Not_Paid_Orders_View AS
SELECT CustomerID, RestaurantEmployeeID, OrderID
FROM Orders
WHERE OrderStatus like 'awaiting payment'
```

VI. TodayReservations

Widok wyświetlający rezerwacje złożone na obecny dzień.

```
CREATE VIEW Today_Reservations_View AS
SELECT FromTime, ToTime, Seats, Orders.PaymentDate,
CustomersPersonalData.FirstName, CustomersPersonalData.LastName
FROM Reservation
	JOIN Orders ON Orders.OrderID = Reservation.OrderID
	JOIN Customers ON Customers.CustomerID = Orders.CustomerID
```

```
    JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
    JOIN CustomersPersonalData ON CustomersPersonalData.PersonID =
IndividualCustomers.PersonID
WHERE FromTime = (SELECT CAST( GETDATE() AS Date ))
```

VII. OrdersPendingForConfirmation

Widok wyświetlający zamówienia oczekujące na potwierdzenie.

```
CREATE VIEW Orders_Pending_For_Confirmation_View AS
SELECT CustomerID, RestaurantEmployeeID, OrderID
FROM Orders
WHERE OrderStatus like '%not_confirmed%'
```

VIII. OrderDetailsView

Widok wyświetlający szczegółowe informacje na temat zamówień.

```
CREATE VIEW Order_Details_View
AS
SELECT  Customers.CustomerID,CustomersPersonalData.FirstName,
CustomersPersonalData.LastName,Orders.OrderID, OrderDetails.Quantity,
ProductPrices.UnitPrice, OrderDate, PaymentDate,
PaymentMethod.PaymentName, OrderStatus,
RestaurantEmployees.RestaurantEmployeeID
FROM Orders
        JOIN OrderDetails ON  OrderDetails.OrderID = Orders.OrderID
        JOIN Products ON Products.ProductID = OrderDetails.ProductID
        JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        JOIN PaymentMethod ON PaymentMethod.PaymentId = Orders.PayVia
        JOIN RestaurantEmployees ON
RestaurantEmployees.RestaurantEmployeeID = Orders.RestaurantEmployeeID
        JOIN Customers ON Customers.CustomerID = Orders.CustomerID
        JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
        JOIN CustomersPersonalData ON CustomersPersonalData.PersonID =
IndividualCustomers.PersonID
```

IX. TotalOrdersProductsPricesReport

Widok wyświetlający tygodniowy i miesięczny raport sprzedaży.

```
CREATE VIEW Total_Orders_Products_Prices_Report_View AS

SELECT TOP 1
(SELECT COUNT(OrderID) FROM Orders
WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
AND YEAR(Orders.OrderDate) = YEAR(GETDATE())) AS [total number of orders
for the last month],
(SELECT COUNT(OrderID) FROM Orders
WHERE DATEPART(WEEK,Orders.OrderDate) = DATEPART(WEEK,GETDATE()) AND
YEAR(Orders.OrderDate) = YEAR(GETDATE()))
AS [total number of orders for the last week],


(SELECT SUM(Quantity) FROM OrderDetails INNER JOIN Orders ON
Orders.OrderID = OrderDetails.OrderID
WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
AND YEAR(Orders.OrderDate) = YEAR(GETDATE())) AS [total number of sold
products for the last month],

(SELECT SUM(Quantity) FROM OrderDetails INNER JOIN Orders ON
Orders.OrderID = OrderDetails.OrderID
WHERE DATEPART(WEEK,Orders.OrderDate) = DATEPART(WEEK,GETDATE()) AND
YEAR(Orders.OrderDate) = YEAR(GETDATE()))
AS [total number of sold products for the last week],

(SELECT SUM(table2.calkowitaSuma) FROM (SELECT
Orders.OrderID,SUM(OrderDetails.Quantity*ProductPrices.UnitPrice*(1-(Orde
rs.DiscountPercent/100.0))
) as calkowitaSuma
FROM Orders INNER JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID
INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
INNER JOIN ProductPrices ON Products.ProductID = ProductPrices.ProductID
WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
AND YEAR(Orders.OrderDate) = YEAR(GETDATE())
AND Orders.OrderDate >= ProductPrices.FromTime AND (ProductPrices.ToTime
is NULL OR ProductPrices.ToTime >= Orders.OrderDate)
GROUP BY Orders.OrderID) AS table2 ) AS [total order price for the last
month],

(SELECT SUM(table2.calkowitaSuma) FROM (SELECT Orders.OrderID,
SUM(OrderDetails.Quantity*ProductPrices.UnitPrice*(1-(Orders.DiscountPerc
ent/100.0))
) as calkowitaSuma
```

```
FROM Orders INNER JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID
INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
INNER JOIN ProductPrices ON Products.ProductID = ProductPrices.ProductID
WHERE DATEPART(WEEK,Orders.OrderDate) = DATEPART(WEEK,GETDATE()) AND
YEAR(Orders.OrderDate) = YEAR(GETDATE())
AND Orders.OrderDate >= ProductPrices.FromTime AND (ProductPrices.ToTime
is NULL OR ProductPrices.ToTime >= Orders.OrderDate)
GROUP BY Orders.OrderID) AS table2 ) AS [total order price for the last
week]

FROM Orders
```

X. AverageSalaryOfRestaurantEmployee

Widok wyświetlający średnie zarobki pracowników.

```
CREATE VIEW Average_Salary_Of_Restaurant_Employee_View AS
SELECT RestaurantEmployees.RestaurantEmployeeID ,FirstName,LastName,
ROUND(AVG(Salary),2) as [srednie zarobki]
FROM RestaurantEmployees INNER JOIN EmployeesSalary ON
RestaurantEmployees.RestaurantEmployeeID =
EmployeesSalary.RestaurantEmployeeID
GROUP BY RestaurantEmployees.RestaurantEmployeeID ,FirstName,LastName
```

XI. FiveBestEmployees

Widok wyświetlający pięciu najlepszych pracowników.

```
CREATE VIEW Five_Best_Employees_View AS
SELECT TOP 5 RestaurantEmployees.RestaurantEmployeeID,
FirstName,LastName FROM RestaurantEmployees
INNER JOIN Orders ON Orders.RestaurantEmployeeID =
RestaurantEmployees.RestaurantEmployeeID
GROUP BY RestaurantEmployees.RestaurantEmployeeID, FirstName,LastName
ORDER BY COUNT(OrderID) DESC
```

XII. TotalProductsSales

Widok wyświetlający liczbę sprzedanych sztuk dla każdego produktu.

```
CREATE VIEW Total_Products_Sales_View AS
SELECT ProductName, SUM(Quantity) AS TotalOrders
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
GROUP BY ProductName
```

XIII. TotalCategoreisSales

Widok wyświetlający liczbę sprzedanych sztuk dla każdej kategorii.

```
CREATE VIEW Total_Categories_Sales_View AS
SELECT Categories.CategoryName, SUM(Quantity) AS TotalOrders
FROM OrderDetails
JOIN Products ON Products.ProductID = OrderDetails.ProductID
JOIN Categories ON Products.CategoryID = Categories.CategoryID
GROUP BY Categories.CategoryName
```

XIV. AvailableTables

Widok wyświetlający wolne stoliki w chwili obecnej..

```
CREATE VIEW Available_Tables_View AS
SELECT DISTINCT DiningTables.DiningTableID, DiningTables.NumberOfSeats
FROM Reservation
JOIN DiningTables ON DiningTables.DiningTableID =
Reservation.DiningTableID
WHERE ToTime < GETDATE()
```

XV. TotalReservationReport

Widok wyświetlający raport rezerwacji

```
CREATE VIEW Total_Reservation_Report_for_Customers_View AS
SELECT
(SELECT COUNT(*)  FROM Reservation
INNER JOIN Orders ON Reservation.OrderID = Orders.OrderID
```

```
INNER JOIN Customers ON Customers.CustomerID = Reservation.ReservationID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
AND YEAR(Orders.OrderDate) = YEAR(GETDATE())
) as [ilość dokonanych rezerwacji prywatnie w tym miesiącu],
(SELECT COUNT(*)  FROM Reservation
INNER JOIN Orders ON Reservation.OrderID = Orders.OrderID
INNER JOIN Customers ON Customers.CustomerID = Reservation.ReservationID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
WHERE DATEPART(WEEK,Orders.OrderDate) = DATEPART(WEEK,GETDATE()) AND
YEAR(Orders.OrderDate) = YEAR(GETDATE())
) as [ilosc dokonanych rezerwacji prywatnie w tym tygodniu],
(SELECT COUNT(*)  FROM Reservation
INNER JOIN Orders ON Reservation.OrderID = Orders.OrderID
INNER JOIN Customers ON Customers.CustomerID = Reservation.ReservationID
INNER JOIN Companies ON Companies.CustomerID = Customers.CustomerID
WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
AND YEAR(Orders.OrderDate) = YEAR(GETDATE())
) as [ilosc dokonanych rezerwacji na firmę w tym miesiacu],
(SELECT COUNT(*)  FROM Reservation
INNER JOIN Orders ON Reservation.OrderID = Orders.OrderID
INNER JOIN Customers ON Customers.CustomerID = Reservation.ReservationID
INNER JOIN Companies ON Companies.CustomerID = Customers.CustomerID
WHERE DATEPART(WEEK,Orders.OrderDate) = DATEPART(WEEK,GETDATE()) AND
YEAR(Orders.OrderDate) = YEAR(GETDATE())
) as [ilosc dokonanych rezerwacji na firmę w tym tygodniu]
```

XVI. CurrentMenuSalesStatsView

Widok wyświetlający statystykę sprzedaży produktów z obecnego menu zamówionych w czasie trwania obecnego menu.

```
CREATE VIEW CurrentMenuSalesStatsView
AS
SELECT Products.ProductName, COUNT(Products.ProductName) AS Total
FROM Products
LEFT JOIN OrderDetails ON OrderDetails.ProductID = Products.ProductID
LEFT JOIN  Orders ON Orders.OrderID = OrderDetails.ProductID
WHERE (OrderDate > (SELECT FromTime
                FROM Menu
                WHERE ToTime IS NULL)
OR OrderDate IS NULL)
```

```
AND Products.ProductID IN (SELECT ProductID
                    FROM MenuDetails
                    WHERE MenuID = (SELECT MenuID
                                FROM Menu
                                WHERE ToTime IS NULL))
GROUP BY Products.ProductName
```

XVII. TotalCustomersDiscountsView

Widok wyświetlający całkowity rabat przyznany każdemu z klientów..

```
CREATE VIEW TotalCustomersDiscountsView
AS
SELECT Customers.CustomerID, SUM(ISNULL((DiscountPercent / 100.0) *
(UnitPrice * Quantity), 0)) AS TotalDisocunt
FROM Customers
LEFT JOIN Orders ON Orders.CustomerID = Customers.CustomerID
LEFT JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
LEFT JOIN ProductPrices ON OrderDetails.ProductID =
ProductPrices.ProductID
WHERE ProductPrices.ToTime IS NULL
GROUP BY Customers.CustomerID
```

XVIII. OrderStatisticsView

Widok wyświetlający wybrane statystyki dotyczące zamówień:

- całkowitą liczbę zamówień
- całkowitą cenę zrealizowanych zamówień
- ilość zamówień dla klientów indywidualnych
- ilość zamówień dla klientów firmowych
- ilość zamówień nieopłaconych
- ilość zamówień nieodebranych
- data ostatnio zrealizowanego zamówienia

```
CREATE VIEW OrderStatisticsView
AS
SELECT
(SELECT COUNT(*) FROM Orders) as [całkowita liczba zamówień],
(SELECT SUM(ProductPrices.UnitPrice*Quantity*(1- (DiscountPercent/100)))
FROM Orders
INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
INNER JOIN Products ON OrderDetails.ProductID =  Products.ProductID
INNER JOIN ProductPrices ON ProductPrices.ProductID = Products.ProductID
```

```
WHERE ProductPrices.FromTime < Orders.OrderDate AND (ProductPrices.ToTime =
NULL OR ProductPrices.ToTime > Orders.OrderDate)
) as [całkowita cena zrealizowanych zamówień],
(SELECT COUNT(*) FROM Orders
INNER JOIN Customers ON Customers.CustomerID = Orders.CustomerID
INNER JOIN IndividualCustomers ON IndividualCustomers.CustomerID =
Customers.CustomerID
) as [ilość zamowien dla klientow indywidualnych],
(SELECT COUNT(*) FROM Orders
INNER JOIN Customers ON Customers.CustomerID = Orders.CustomerID
INNER JOIN Companies ON Companies.CustomerID = Customers.CustomerID
) as [ilość zamowien dla klientów firmowych],
(SELECT COUNT(*) FROM Orders WHERE PaymentDate = NULL) as [ilość zamówień
nieopłaconych],
(SELECT COUNT(*) FROM Orders WHERE CollectDate = NULL) as [ilość zamówień
nieodebranych],
(SELECT TOP 1 OrderDate FROM Orders ORDER BY OrderDate DESC) as [data
ostatnio zrealizowanego zamówienia]
GO
```

5. Procedury

I. AddMenu

Procedura dodająca nowe menu.

```
CREATE PROCEDURE AddMenuProcedure
@MenuName varchar(64),
@FromTime datetime,
@ToTime   datetime
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
      IF EXISTS(
         SELECT *
         FROM Menu
         WHERE MenuName = @MenuName
      )
                        BEGIN
                              ;
                              THROW 52000, 'Menu with provided name is already in the
database', 1
```

```
                        END

    DECLARE @MenuID INT
                SELECT @MenuID = ISNULL(MAX(MenuID), 0) + 1
                FROM Menu

        INSERT INTO Menu(MenuID, MenuName, FromTime, ToTime)
        VALUES (@MenuID, @MenuName, @FromTime, @ToTime);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                'Error adding menu: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
```

II. RemoveMenu

Procedura usuwająca menu.

```
CREATE PROCEDURE RemoveMenuProcedure
@MenuID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Menu
            WHERE @MenuID = @MenuID
        )
        BEGIN
        ;
        DELETE FROM Menu WHERE  @MenuID = MenuID
        END
    ELSE
        BEGIN
            ;
            THROW 52000, 'Menu with provided ID does not exist', 1
        END

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
```

```
            'Error removing menu: ' + ERROR_MESSAGE();
    THROW 52000, @errormsg, 1;
  END CATCH
END
```

III. AddIngredientToWarehouse

Procedura dodająca nowy składnik do magazynu.

```
CREATE PROCEDURE AddIngredientToWarehouse
@IngredientName varchar(64),
@QuantityLeft int

AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
    IF EXISTS(
      SELECT *
      FROM IngredientsWarehouse
      WHERE IngredientName = @IngredientName
    )
                    BEGIN
                        ;
                        THROW 52000, 'Ingredient with provided name is already in
the database', 1
                    END

  DECLARE @IngredientID INT
            SELECT @IngredientID = ISNULL(MAX(IngredientID), 0) + 1
            FROM IngredientsWarehouse

      INSERT INTO IngredientsWarehouse(IngredientID, IngredientName, QuantityLeft)
      VALUES (@IngredientID, @IngredientName, @QuantityLeft);

  END TRY
  BEGIN CATCH
    DECLARE @errormsg nvarchar(2048) =
            'Error adding ingredient: ' + ERROR_MESSAGE();
    THROW 52000, @errormsg, 1;
  END CATCH
END
```

IV. RemoveIngredientFromWarehouse

Procedura usuwająca składnik z magazynu.

```
CREATE PROCEDURE RemoveIngredientFromWarehouse
@IngredientID int

AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
      IF EXISTS(
         SELECT *
         FROM IngredientsWarehouse
         WHERE IngredientID = @IngredientID
      )
      BEGIN
      ;
      DELETE FROM IngredientsWarehouse WHERE  IngredientID = @IngredientID
      END
   ELSE
      BEGIN
         ;
         THROW 52000, 'Ingredient with provided ID does not exist', 1
      END

   END TRY
   BEGIN CATCH
      DECLARE @errormsg nvarchar(2048) =
               'Error removing Ingredient: ' + ERROR_MESSAGE();
      THROW 52000, @errormsg, 1;
   END CATCH
END
```

V. AddIngredientToProduct

Procedura dodająca składnik do produktu.

```
CREATE PROCEDURE AddIngredientToProduct
@ProductID int,
@IngredientID int

AS
BEGIN
```

```
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM ProductIngredients
            WHERE IngredientID = @IngredientID
                            AND ProductID = @ProductID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'This productID-ingredientID pair already
exists', 1
                            END
                    IF NOT EXISTS(
            SELECT *
            FROM Products
            WHERE ProductID = @ProductID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'Product with provided ID does not exist', 1
                            END
                    IF NOT EXISTS(
            SELECT *
            FROM IngredientsWarehouse
            WHERE IngredientID = @IngredientID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'Ingredient with provided ID does not exist',
1
                            END

            INSERT INTO ProductIngredients(ProductID, IngredientID)
            VALUES (@ProductID, @IngredientID);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                'Error adding productID-ingredientID pair: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
```

VI. RemoveIngredientFromProduct

Procedura usuwająca składnik produktu.

```
CREATE PROCEDURE RemoveIngredientFromProduct
@IngredientID int,
@ProductID int

AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
      IF EXISTS(
         SELECT *
         FROM ProductIngredients
         WHERE IngredientID = @IngredientID
                        AND ProductID = @ProductID
      )
      BEGIN
      ;
      DELETE FROM ProductIngredients WHERE  IngredientID = @IngredientID AND
ProductID = @ProductID
      END
   ELSE
      BEGIN
         ;
         THROW 52000, 'Provided productID-ingredientID pair does not exist', 1
      END

   END TRY
   BEGIN CATCH
      DECLARE @errormsg nvarchar(2048) =
               'Error removing productID-ingredientID pair: ' + ERROR_MESSAGE();
      THROW 52000, @errormsg, 1;
   END CATCH
END
```

VII. AddProductToMenu

Procedura dodająca produkt do menu.

```
CREATE PROCEDURE AddProductToMenuProcedure
@ProductID int,
@MenuID int

AS
BEGIN
```

```
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM MenuDetails
            WHERE MenuID = @MenuID
                            AND ProductID = @ProductID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'This productID-menuID pair already exists',
1
                            END
                    IF NOT EXISTS(
            SELECT *
            FROM Products
            WHERE ProductID = @ProductID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'Product with provided ID does not exist', 1
                            END
                    IF NOT EXISTS(
            SELECT *
            FROM Menu
            WHERE MenuID = @MenuID
        )
                            BEGIN
                                    ;
                                    THROW 52000, 'Menu with provided ID does not exist', 1
                            END

            INSERT INTO MenuDetails(MenuID, ProductID)
            VALUES (@MenuID, @ProductID);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                'Error adding productID-menuID pair: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
```

VIII. RemoveProductFromMenu

Procedura usuwająca produkt z menu.

```
CREATE PROCEDURE RemoveProductFromMenuProcedure
@MenuID int,
@ProductID int

AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
      IF EXISTS(
         SELECT *
         FROM MenuDetails
         WHERE MenuID = @MenuID
                        AND ProductID = @ProductID
      )
      BEGIN
      ;
      DELETE FROM MenuDetails WHERE  MenuID = @MenuID AND ProductID =
@ProductID
      END
   ELSE
      BEGIN
         ;
         THROW 52000, 'Provided productID-menuID pair does not exist', 1
      END

   END TRY
   BEGIN CATCH
      DECLARE @errormsg nvarchar(2048) =
               'Error removing productID-menuID pair: ' + ERROR_MESSAGE();
      THROW 52000, @errormsg, 1;
   END CATCH
END
```

IX. AddProductToOrder

Procedura dodające produkt do zamówienia

```
CREATE PROCEDURE AddProductToOrder
@ProductID int,
@OrderID int,
@Quantity int

AS
```

```
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM OrderDetails
            WHERE OrderID = @OrderID
            AND ProductID = @ProductID
        )
            BEGIN
                ;
                THROW 52000, 'This productID-orderID pair already exists', 1
            END
        IF NOT EXISTS(
            SELECT *
            FROM Products
            WHERE ProductID = @ProductID
        )
            BEGIN
                ;
                THROW 52000, 'Product with provided ID does not exist', 1
            END
        IF NOT EXISTS(
            SELECT *
            FROM Orders
            WHERE OrderID = @OrderID
        )
            BEGIN
                ;
                THROW 52000, 'Order with provided ID does not exist', 1
            END

    INSERT INTO OrderDetails(OrderID, ProductID,Quantity)
    VALUES (@OrderID, @ProductID,@Quantity);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Error adding productID-orderID pair: ' +
ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

X. RemoveProductFromOrder

Procedura usuwająca produkt z zamówienia

```
CREATE PROCEDURE RemoveProductFromOrder
@OrderID int,
@ProductID int,
@Quantity int

AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM OrderDetails
            WHERE OrderID = @OrderID
            AND ProductID = @ProductID
            AND Quantity = @Quantity
        )
        BEGIN
        ;
        DELETE FROM OrderDetails WHERE  OrderID = @OrderID AND ProductID =
@ProductID AND Quantity = @Quantity
        END
    ELSE
        BEGIN
            ;
            THROW 52000, 'Provided productID-orderID pair does not exist', 1
        END

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Error removing productID-orderID pair: ' +
ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

XI. AddCustomerProcedure

Procedura dodająca klienta do bazy

```
CREATE PROCEDURE AddCustomerProcedure
@Street varchar(64),
@Country varchar(64),
```

```
@City varchar(64),
@PostCode varchar(16),
@Phone varchar(16),
@Email varchar(64)

AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Customers
            WHERE Street = @Street AND Country = @Country AND City=@City AND
PostCode = @PostCode AND Phone = @Phone AND Email=@Email
        )
        BEGIN
            ;
            THROW 52000, 'Klient jest już w bazie', 1
        END

    DECLARE @CustomerID INT
    SELECT @CustomerID = ISNULL(MAX(CustomerID), 0) + 1
    FROM Customers
    INSERT INTO Customers(CustomerID,Street, Country, City, PostCode, Phone,
Email)
    VALUES (@CustomerID, @Street, @Country, @City, @PostCode, @Phone,
@Email);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Błąd dodawania klienta: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

XII. RemoveCustomerProcedure

Procedura usuwająca klienta z bazy

```
CREATE PROCEDURE RemoveCustomerProcedure
@CustomerID INT

AS
BEGIN
```

```
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Customers
            WHERE CustomerID = @CustomerID
        )
        BEGIN
        ;
        DELETE FROM Customers
        WHERE  CustomerID = @CustomerID
        END
    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Błąd usuwania klienta: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

XIII. updateDiscountProcedure

Procedura aktualizująca zniżkę

```
CREATE PROCEDURE updateDiscountProcedure
@VariableType varchar(3),
@VariableValue int

AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM VariablesData
            WHERE VariableType = @VariableType
        )
            BEGIN
                ;
                THROW 52000, 'this type of discount does not exist', 1
            END


        IF EXISTS(
```

```
            SELECT * FROM VariablesData WHERE VariableType =
@VariableType AND ToTime IS NULL
        )
        BEGIN
            ;
            UPDATE VariablesData SET ToTime = GETDATE() WHERE ToTime IS
NULL  AND VariableType = @VariableType
        END

        INSERT INTO
VariablesData(FromTime,ToTime,VariableType,VariableValue)
            VALUES (GETDATE(),NULL,@VariableType,@VariableValue);


    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Błąd aktualizacji rabatu: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

XIV. AddReservationProcedure

Procedura dodająca rezerwacje

```
CREATE PROCEDURE AddReservationProcedure
@FromTime      datetime ,
@ToTime        datetime ,
@Seats         int,
@DiningTableID int,
@OrderID       int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
        SELECT OrderID FROM Orders WHERE OrderID = @OrderID
        )
            BEGIN
                ;
                THROW 52000, 'Provided OrderID is not in the database', 1
            END
```

```
        IF NOT EXISTS(
        SELECT DiningTableID FROM DiningTables WHERE DiningTableID =
@DiningTableID
        )
            BEGIN
                ;
                THROW 52000, 'Provided DiningTableID is not in the
database', 1
            END

        IF EXISTS(
            SELECT *
            FROM Reservation
            WHERE OrderID = @OrderID AND DiningTableID = @DiningTableID
        )
            BEGIN
                ;
                THROW 52000, 'Reservation with provided OrderID and
DiningTableID is already in the database', 1
            END

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

    INSERT INTO Reservation(ReservationID, FromTime, ToTime,
Seats,DiningTableID,OrderID)
    VALUES (@ReservationID, @FromTime, @ToTime,
@Seats,@DiningTableID,@OrderID);

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Error adding menu: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
```

```
go
```

XV. RemoveReservationProcedure

Procedura usuwające rezerwacje

```
CREATE PROCEDURE RemoveReservationProcedure
```

```
@ReservationID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Reservation
            WHERE ReservationID = @ReservationID
        )
        BEGIN
        ;
        DELETE FROM Reservation WHERE  @ReservationID = ReservationID
        END
    ELSE
        BEGIN
            ;
            THROW 52000, 'Reservation with provided ID does not exist', 1
        END

    END TRY
    BEGIN CATCH
        DECLARE @errormsg nvarchar(2048) =
                    'Error removing menu: ' + ERROR_MESSAGE();
        THROW 52000, @errormsg, 1;
    END CATCH
END
go
```

6. Funkcje

I. GetIngredientsForProduct

Funkcja zwracająca tabelę składników danego produktu.

```
CREATE FUNCTION GetIngredientsForProduct(@ProductID int)
  RETURNS table
      AS
      RETURN
            SELECT IngredientName
            FROM ProductIngredients
            JOIN IngredientsWarehouse ON IngredientsWarehouse.IngredientID =
ProductIngredients.IngredientID
            WHERE ProductIngredients.ProductID = @ProductID
```

II. GetProductsFromCategory

Funkcja zwracająca tabelę produktów z danej kategorii.

```
CREATE FUNCTION GetProductsFromCategory(@CategoryID int)
   RETURNS table
         AS
         RETURN
                  SELECT Products.ProductName
                  FROM Products
                  JOIN Categories ON Categories.CategoryID = Products.CategoryID
                  WHERE Categories.CategoryID = @CategoryID
```

III. GetProductsFromMenu

Funkcja zwracająca tabelę produktów z danego menu.

```
CREATE FUNCTION GetProductsFromMenu(@MenuID int)
   RETURNS table
         AS
         RETURN
                  SELECT Products.ProductName
                  FROM Menu
                  JOIN MenuDetails ON MenuDetails.MenuID = Menu.MenuID
                  JOIN Products ON Products.ProductID = MenuDetails.ProductID
```

IV. GetCurrentEmployeeSalary

Funkcja zwracająca obecną płacę pracownika.

```
CREATE FUNCTION GetCurrentEmployeeSalary(@EmployeeID int)
RETURNS table
AS
RETURN
  SELECT Salary
  FROM EmployeesSalary
  WHERE RestaurantEmployeeID = @EmployeeID
  AND ToTime IS NULL
```

V. GetCurrentAverageSalaryForOccupation

Funkcja zwracająca obecną średnią płacę dla danego stanowiska.

```
CREATE FUNCTION GetCurrentAverageSalaryForOccupation(@Occupation varchar(64))
RETURNS table
AS
RETURN
  SELECT AVG(Salary) AS AveragePriceForOccupation
  FROM EmployeesSalary
  JOIN RestaurantEmployees ON EmployeesSalary.RestaurantEmployeeID =
RestaurantEmployees.RestaurantEmployeeID
  WHERE Occupation = @Occupation
  AND ToTime IS NULL
```

VI. GetTotalProductsAndAveragePriceOfMenu

Funkcja zwracająca łączną liczbę produktów oraz ich średnią cenę dla danego menu.

```
CREATE FUNCTION GetTotalProductsAndAveragePriceOfMenu(@MenuID int)
RETURNS table
AS
RETURN
  SELECT COUNT(UnitPrice) AS TotalProducts, AVG(UnitPrice) AS AverageUnitPrice
  FROM Menu
  JOIN MenuDetails ON Menu.MenuID = MenuDetails.MenuID
  JOIN ProductPrices ON MenuDetails.ProductID = ProductPrices.ProductID
  WHERE Menu.MenuID = @MenuID
  AND ProductPrices.ToTime IS NULL
```

VII. GetHighestSalaryForEmployee

Funkcja zwracająca największą historyczną płacę danego pracownika.

```
CREATE FUNCTION GetHighestSalaryForEmployee(@EmployeeID int)
RETURNS table
AS
RETURN
  SELECT MAX(Salary) AS MaxEmployeeSalary
  FROM EmployeesSalary
  WHERE RestaurantEmployeeID = @EmployeeID
```

VIII. RemainingDaysForMenu

Funkcja sprawdzająca ile dni pozostało na dane menu

```
CREATE FUNCTION RemainingDaysForMenu(@MenuID int)
RETURNS table
AS
RETURN
    SELECT DATEDIFF(day, GETDATE(), (SELECT
        ToTime
        FROM Menu
        WHERE MenuID = @MenuID )) as [remaining days]
GO
```

IX. RemainingFreeSeats

Funkcja sprawdzająca ile pozostało wolnych miejsc

```
CREATE FUNCTION RemainingFreeSeats()
RETURNS table
AS
RETURN
    SELECT (
    (SELECT
    SUM(NumberOfSeats)
    FROM DiningTables) -

    (SELECT
    SUM(Seats)
    FROM Reservation
    WHERE GETDATE() >= FromTime AND GETDATE() <= ToTime
    )) as [free seats]
GO
```

X. CanAccommodateCustomers

Funkcja sprawdzająca czy można zmieścić klientów w restauracji

```
CREATE FUNCTION CanAccommodateCustomers(@customers int)
RETURNS table
AS
RETURN
    SELECT  (CASE WHEN free_seats.nrOfFreeSeats >= @customers THEN 'true'
ELSE 'false' END) AS freeSpaces
    FROM (SELECT ((SELECT SUM(NumberOfSeats) FROM DiningTables) - (SELECT
SUM(Seats) FROM Reservation WHERE GETDATE() >= FromTime AND GETDATE() <=
ToTime)) as nrOfFreeSeats
```

```
    ) as free_seats
GO
```

XI. GetDetailsOfOrder

Funkcja zwracająca detale danego zamówienia

```
CREATE FUNCTION GetDetailsOfOrder(@input int)
    RETURNS table AS
        RETURN
        SELECT OrderDetails.Quantity*ProductPrices.UnitPrice as cena,
Orders.OrderID,Orders.OrderDate, Orders.OrderStatus, Orders.PayVia as
rodzaj_płatności--, RestaurantEmployees.FirstName as
imie_obsługującego_pracownika, Customers.Email as kontakt_do_klienta
        FROM Orders
        INNER JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        WHERE Orders.OrderID = @input
go
```

XII. GetOrdersAboveValue

Funkcja zwraca zamówienia powyżej danej wartości

```
CREATE FUNCTION GetOrdersAboveValue(@input int)
    RETURNS table AS
        RETURN
        SELECT Orders.OrderID, Orders.OrderStatus,
OrderDetails.Quantity*ProductPrices.UnitPrice as cena
        FROM Orders
        INNER JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        WHERE OrderDetails.Quantity*ProductPrices.UnitPrice > @input
go
```

XIII. GetValueOfOrdersOnDay

Funkcja zwracająca wartości wszystkich zamówień w danym dniu

```
CREATE FUNCTION GetValueOfOrdersOnDay(@date date)
    RETURNS table AS
        RETURN
        SELECT SUM(OrderDetails.Quantity*ProductPrices.UnitPrice) as Suma
```

```
        FROM Orders
        INNER JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        WHERE YEAR(@date) = YEAR(Orders.OrderDate)
        AND MONTH(@date) = MONTH(Orders.OrderDate)
        AND DAY(@date) = DAY(Orders.OrderDate)
go
```

XIV. GetValueOfOrdersOnMonth

Funkcja zwracająca wartości wszystkich zamówień w danym miesiącu

```
CREATE FUNCTION GetValueOfOrdersOnMonth(@date date)
    RETURNS table AS
        RETURN
        SELECT SUM(OrderDetails.Quantity*ProductPrices.UnitPrice) as Suma
        FROM Orders
        INNER JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        WHERE YEAR(@date) = YEAR(Orders.OrderDate)
        AND MONTH(@date) = MONTH(Orders.OrderDate)
go
```

XV. GetValueOfOrder

Funkcja zwracająca w wartość danego zamówienia

```
CREATE FUNCTION GetValueOfOrdersOnMonth(@date date)
    RETURNS table AS
        RETURN
        SELECT SUM(OrderDetails.Quantity*ProductPrices.UnitPrice) as Suma
        FROM Orders
        INNER JOIN OrderDetails ON OrderDetails.OrderID = Orders.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON ProductPrices.ProductID =
Products.ProductID
        WHERE YEAR(@date) = YEAR(Orders.OrderDate)
        AND MONTH(@date) = MONTH(Orders.OrderDate)
go
```

XVI. Invoice

Funkcja zwraca fakturę dla zwykłego zamówienia

```
CREATE FUNCTION invoice(@ordersID int)
RETURNS table
AS
RETURN
       SELECT Products.ProductName as[nazwa produktu],
        OrderDetails.Quantity as [ilosc],
        ProductPrices.UnitPrice as [cena produktu],
        Orders.DiscountPercent as [zniżka],
        ProductPrices.UnitPrice*(1- (Orders.DiscountPercent/100.0)) as [cena
produktu z uwzględnieniem zniżki],
        Orders.OrderDate as [data zamowienia] FROM Orders
        INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON Products.ProductID =
ProductPrices.ProductID
        WHERE ProductPrices.FromTime >= Orders.OrderDate AND
(ProductPrices.ToTime = NULL OR ProductPrices.ToTime  >= Orders.OrderDate)
        AND Orders.OrderID = @ordersID
GO
```

XVII. CollectiveInvoice

Funkcja zwraca fakturę na wszystkie zamówienia z miesiąca na daną firmę

```
CREATE FUNCTION collectiveInvoice(@companyID int)
RETURNS table
AS
RETURN
        SELECT

SUM(OrderDetails.Quantity*ProductPrices.UnitPrice*(1-(Orders.DiscountPercent
/100.0))) as [calkowita cena produktow],
        ROUND(AVG(Orders.DiscountPercent),2) as [srednia znizek],
        Orders.OrderDate as [data zamowienia],
        Companies.NIP as [NIP firmy]
        FROM Companies
        INNER JOIN  Customers ON Companies.CustomerID = Customers.CustomerID
        INNER JOIN ORDERS ON Customers.CustomerID = Orders.CustomerID
        INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
        INNER JOIN Products ON Products.ProductID = OrderDetails.ProductID
        INNER JOIN ProductPrices ON Products.ProductID =
ProductPrices.ProductID
        WHERE MONTH(Orders.OrderDate) = MONTH(GETDATE())
        AND YEAR(Orders.OrderDate) = YEAR(GETDATE())
        AND Companies.CompanyID = @companyID
        AND ProductPrices.FromTime >= Orders.OrderDate AND
(ProductPrices.ToTime = NULL OR ProductPrices.ToTime  >= Orders.OrderDate)
```

```
        GROUP BY Orders.OrderDate,Companies.NIP
GO
```

7. Triggery

I. AddMenuOneDayInAdvance

Trigger sprawdzający, czy menu jest dodawane z wyprzedzeniem co najmniej jednego dnia.

```
CREATE TRIGGER AddMenuOneDayInAdvanceTrigger
        ON Menu
        AFTER INSERT AS
BEGIN
        DECLARE @InsertedMenuID int
        SET @InsertedMenuID = (SELECT MAX(MenuID)
                               FROM Menu)
        DECLARE @InsertedDate datetime
        SET @InsertedDate = (SELECT FromTime
                             FROM Menu
                             WHERE MenuID = @InsertedMenuID)
        IF DATEDIFF(hour, GETDATE(), @InsertedDate) < 24
                BEGIN
                        PRINT ('Adding new menu failed. Menu must be inserted with at
least one day in advance')
                        DELETE FROM Menu WHERE MenuID = @InsertedMenuID
                END
END
```

II. CorrectSeaFoodOrderDateTrigger

Trigger sprawdzający, czy zamówienie zawierające owoce morza zostało złożone w odpowiednim terminie i na odpowiedni termin.

```
CREATE TRIGGER CorrectSeafoodOrderDateTrigger
ON OrderDetails
AFTER INSERT AS
BEGIN
        DECLARE @InsertedOrderID int
        SET @InsertedOrderID = (SELECT OrderID
```

```
                                FROM INSERTED)

        DECLARE @InsertedProductID int
        SET @InsertedProductID = (SELECT ProductID
                                  FROM INSERTED)

        DECLARE @InsertedProductQuantity int
        SET @InsertedProductQuantity = (SELECT Quantity
                                        FROM INSERTED)

        DECLARE @InsertedProductCategoryID int
        SET @InsertedProductCategoryID = (SELECT CategoryID
                                          FROM INSERTED
                                            JOIN Products ON INSERTED.ProductID =
Products.ProductID)

        DECLARE @CollectDate datetime
        SET @CollectDate = (SELECT CollectDate
                            FROM Orders
                            WHERE OrderID = @InsertedOrderID)

        DECLARE @CollectDateWeekday datetime
        SET @CollectDateWeekday = DATEPART(weekday, @CollectDate)

        DECLARE @OrderDate datetime
        SET @OrderDate = (SELECT OrderDate
                          FROM Orders
                        WHERE OrderID = @InsertedOrderID)

        IF @InsertedProductCategoryID = (SELECT CategoryID
                                         FROM Categories
                                           WHERE CategoryName = 'Seafood')
                BEGIN
                        IF DATEDIFF(day, @OrderDate, @CollectDate) < 3 +
(@CollectDateWeekday - 5)
                                BEGIN
                                        PRINT ('Adding product to order failed. Orders
containing Seafood can only be placed until the first Monday before the collect date.')
                                        DELETE FROM OrderDetails WHERE OrderID =
@InsertedOrderID
                                        AND ProductID = @InsertedProductID
                                        AND Quantity = @InsertedProductQuantity
                                END
                        IF @CollectDateWeekday NOT IN (5, 6, 7)
                                BEGIN
                                        PRINT ('Adding product to order failed. Orders
containing Seafood can only be collected on Thursdays, Fridays and Saturdays')
```

```
                                DELETE FROM OrderDetails WHERE OrderID =
@InsertedOrderID
                                AND ProductID = @InsertedProductID
                                AND Quantity = @InsertedProductQuantity
                            END
            END
END
```

III. CheckReservationSeatsTrigger

Trigger sprawdzający czy rezerwacja jest na minimum 2 miejsca

```
CREATE TRIGGER CheckReservationSeatsTrigger
    ON Reservation
    AFTER INSERT AS
BEGIN
    DECLARE @InsertedReservationID int
    SET @InsertedReservationID = (SELECT ReservationID
                    FROM INSERTED)
    DECLARE @InsertedSeats int
    SET @InsertedSeats = (SELECT Seats
                FROM Reservation
                WHERE ReservationID = @InsertedReservationID)
    IF  @InsertedSeats < 2
        BEGIN
            PRINT ('Adding new reservation failed. Reservation must be
inserted with at least two seats')
            DELETE FROM Reservation WHERE ReservationID =
@InsertedReservationID
        END
END
GO
```

IV. CheckReservationCapacityTrigger

Trigger sprawdzający czy ilość osób na stolik w rezerwacji nie jest za duża na stolik

```
CREATE TRIGGER CheckReservationCapacityTrigger
    ON Reservation
    AFTER INSERT AS
BEGIN
    DECLARE @InsertedReservationID int
    SET @InsertedReservationID = (SELECT ReservationID
                    FROM INSERTED)
    DECLARE @InsertedSeats int
    SET @InsertedSeats = (SELECT Seats
```

```
                FROM Reservation
                WHERE ReservationID = @InsertedReservationID)
    IF  @InsertedSeats > (SELECT DiningTables.NumberOfSeats FROM Reservation
INNER JOIN DiningTables ON DiningTables.DiningTableID =
Reservation.DiningTableID WHERE @InsertedReservationID = ReservationID)
        BEGIN
            PRINT ('Adding new reservation failed. Reservation must be
inserted with less or equal number of availabile seats')
            DELETE FROM Reservation WHERE ReservationID =
@InsertedReservationID
        END
END
GO
```

V. CheckDiscountAviabilityTrigger

Trigger sprawdzający możliwość przyznania zniżki na podstawie poprzednich zamówień oraz przypisujące daną zniżkę do klienta

```
CREATE TRIGGER CheckDiscountAvailabilityTrigger
ON Orders
AFTER INSERT AS
BEGIN
    DECLARE @InsertedOrderID int
    SET @InsertedOrderID = (SELECT OrderID
                    FROM INSERTED)

    DECLARE @InsertedCustomerID int
    SET @InsertedCustomerID = (SELECT Orders.CustomerID
                        FROM INSERTED
                        INNER JOIN Orders ON Orders.OrderID =
INSERTED.OrderID)


    DECLARE @ThisOrderZ1 int
    SET @ThisOrderZ1 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='Z1' AND
toTime = NULL)

    DECLARE @ThisOrderK1 int
    SET @ThisOrderK1 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='K1' AND
toTime = NULL)
    DECLARE @ThisOrderR1 int
    SET @ThisOrderR1 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='R1' AND
toTime = NULL)
                            --------------------------------
```

```
    DECLARE @ThisOrderK2 int
    SET @ThisOrderK2 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='K2' AND
toTime = NULL)
    DECLARE @ThisOrderR2 int
    SET @ThisOrderR2 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='R2' AND
toTime = NULL)
    DECLARE @ThisOrderD1 int
    SET @ThisOrderD1 = (SELECT VariableValue
                            FROM VariablesData WHERE VariableType='D1' AND
toTime = NULL)


    DECLARE @ThisOrderCustomerNumberOfOrders int
    SET @ThisOrderCustomerNumberOfOrders =(SELECT COUNT(Orders.OrderID)
                              FROM Customers
                           JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Customers.CustomerID
                           JOIN Orders ON Orders.CustomerID =
Customers.CustomerID
                           WHERE Orders.CustomerID = @InsertedCustomerID AND
( [dbo].GetValueOfOrder(@InsertedOrderID))>@ThisOrderK1)



    DECLARE @ThisOrderCustomerNumberValueOfOrders int
    SET @ThisOrderCustomerNumberValueOfOrders = (SELECT SUM(
[dbo].GetValueOfOrder(Orders.OrderID))
                              FROM Customers
                           INNER JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Customers.CustomerID
                           INNER JOIN Orders ON Orders.CustomerID =
Customers.CustomerID
                           WHERE Orders.CustomerID = @InsertedCustomerID)


    IF @ThisOrderCustomerNumberOfOrders >= @ThisOrderZ1
        BEGIN
                    PRINT ('Qualified for first discount')
                    UPDATE TempDiscount
                    SET DiscountPercent = @ThisOrderR1, ToTime = NULL,
FromTime = GETDATE()
                    WHERE CustomerID = @InsertedOrderID
        END

    IF @ThisOrderCustomerNumberValueOfOrders >= @ThisOrderK2
        BEGIN
```

```
                    PRINT ('Qualified for second discount')
                    UPDATE TempDiscount
                    SET DiscountPercent = @ThisOrderR1, ToTime =
DATEADD(day,@ThisOrderD1, GETDATE()), FromTime = GETDATE()
                    WHERE CustomerID = @InsertedOrderID
        END
END
GO
```

VI. CheckIndividualReservationAviabilityTrigger

Trigger sprawdzający czy dany klient indywidualny ma możliwość rezerwacji

```
CREATE TRIGGER CheckIndividualReservationAvailabilityTrigger
ON Reservation
AFTER INSERT AS
BEGIN
DECLARE @InsertedReservationID int
    SET @InsertedReservationID = (SELECT ReservationID
                    FROM INSERTED)
    DECLARE @InsertedOrderID int
    SET @InsertedOrderID = (SELECT OrderID
                     FROM INSERTED)

    DECLARE @InsertedCustomerID int
    SET @InsertedCustomerID = (SELECT Orders.CustomerID
                       FROM INSERTED
                         INNER JOIN Orders ON Orders.OrderID =
INSERTED.OrderID
                       )

    DECLARE @ThisOrderWK int
    SET @ThisOrderWK = (SELECT VariableValue
                              FROM VariablesData WHERE
VariableType='WK' AND toTime = NULL)

    DECLARE @ThisOrderWZ int
    SET @ThisOrderWZ = (SELECT VariableValue
                              FROM VariablesData WHERE
VariableType='WZ' AND toTime = NULL)
```

```
    DECLARE @ThisOrderCustomerNumberOfOrders int
    SET @ThisOrderCustomerNumberOfOrders =(SELECT
COUNT(Orders.OrderID) --liczy wszystkie zamówienia danego klienta
                            FROM Customers
                           INNER JOIN IndividualCustomers ON
IndividualCustomers.CustomerID = Customers.CustomerID
                           INNER JOIN Orders ON Orders.CustomerID =
Customers.CustomerID
                           WHERE Orders.CustomerID =
@InsertedCustomerID)

    DECLARE @ThisOrderCustomerNumberValueOfOrders int
    SET @ThisOrderCustomerNumberValueOfOrders =
(dbo.GetValueOfOrder(@InsertedOrderID) --liczy wartosc danego
zamowienia
                            )

    DECLARE @flag int
    SET @flag =0
    IF @ThisOrderCustomerNumberOfOrders >= @ThisOrderWK
        BEGIN
                    PRINT ('Qualified for reservation')
                    SET @flag =1
        END

    IF @ThisOrderCustomerNumberValueOfOrders >= @ThisOrderWZ AND
@flag=0
        BEGIN
                    PRINT ('Qualified for reservation')
                    SET @flag =1
        END
    IF  @flag=0
        BEGIN
                    PRINT ('Not qualified for reservation')
                    DELETE FROM Reservation WHERE ReservationID =
@InsertedReservationID
        END
END
GO
```

8. Indeksy

I. RestaurantEmployees_RestaurantEmployeeID_Index

```
CREATE UNIQUE NONCLUSTERED INDEX
RestaurantEmployees_RestaurantEmployeeID_Index ON RestaurantEmployees
(RestaurantEmployeeID)
```

II. Products_ProductID_Index

```
CREATE UNIQUE NONCLUSTERED INDEX Products_ProductID_Index ON Products
(ProductID)
```

III. Menu_MenuID_Index

```
CREATE UNIQUE NONCLUSTERED INDEX Menu_MenuID_Index ON Menu (MenuID)
```

IV. MenuDetails_MenuID_Index

```
CREATE NONCLUSTERED INDEX MenuDetails_MenuID_Index ON MenuDetails
(MenuID)
```

V. Orders_OrderID_Index

```
CREATE UNIQUE NONCLUSTERED INDEX Orders_OrderID_Index ON Orders
(OrderID)
```

VI. OrderDetails_OrderID_Index

```
CREATE NONCLUSTERED INDEX OrderDetails_OrderID_Index ON OrderDetails
(OrderID)
```

VII. Customers_CustomerID_Index

```
CREATE UNIQUE NONCLUSTERED INDEX Customers_CustomerID_Index ON Customers
(CustomerID)
```

9. Role i uprawnienia

Dostępne role w bazie:

I. Administrator - dostęp do wszystkiego
II. Finanse - dostęp do danych klienta, faktur, cen
III. StoreKeeper - obsługa produktów
IV. Waiter - dostęp do sekcji związanych z obsługą zamówień
V. Właściciel - dostęp do wszystkiego - bez modyfikacji funkcjonalności

VI. Role na tabele

```
GRANT SELECT ON Companies to Finanse
GRANT SELECT ON CompanyEmployees to Finanse
GRANT SELECT ON CompanyReservationParticipants to  Finanse
GRANT SELECT ON CustomersPersonalData to Finanse, Waiter
GRANT SELECT ON Customers to Finanse, Waiter
GRANT SELECT ON IndividualCustomers to Finanse, Waiter
GRANT SELECT ON DiningTables to Waiter
GRANT SELECT ON Invoices to Finanse
GRANT SELECT ON Menu to  Waiter, StoreKeeper
GRANT SELECT ON MenuDetails to  Waiter, StoreKeeper
GRANT SELECT ON OrderDetails to Finanse, Waiter
GRANT SELECT ON Orders to Finanse, Waiter
GRANT SELECT ON PaymentMethod to Finanse, Waiter
GRANT SELECT ON ProductIngredients to StoreKeeper
GRANT SELECT ON IngredientsWarehouse to StoreKeeper
GRANT SELECT ON Products to StoreKeeper, Waiter
GRANT SELECT ON ProductPrices to Finanse, Waiter
GRANT SELECT ON Categories to Waiter, StoreKeeper
GRANT SELECT ON Reservation to  Finanse, Waiter
GRANT SELECT ON RestaurantEmployees to Finanse
GRANT SELECT ON EmployeesSalary to Finanse
```

```
GRANT SELECT ON Takeaway to Waiter
GRANT SELECT ON VariablesData to Finanse
```

VII. Role na widoki

```
GRANT SELECT ON Current_Menu_View  to Waiter, StoreKeeper
GRANT SELECT ON Available_Products_View  to Waiter, StoreKeeper
GRANT SELECT ON Not_Available_Products_View  to StoreKeeper
GRANT SELECT ON Not_Available_Ingredients_View  to StoreKeeper
GRANT SELECT ON Not_Paid_Orders_View  to Finanse, Waiter
GRANT SELECT ON Today_Reservations_View  to Finanse,Waiter
GRANT SELECT ON Orders_Pending_For_Confirmation_View  to Finanse, Waiter
GRANT SELECT ON Order_Details_View to Finanse, Waiter
GRANT SELECT ON Total_Orders_Products_Prices_Report_View  to Finanse
GRANT SELECT ON Total_Products_Sales_View  to Finanse
GRANT SELECT ON Total_Categories_Sales_View  to Finanse
GRANT SELECT ON Available_Tables_View  to Waiter
GRANT SELECT ON Total_Reservation_Report_for_Customers_View to Finanse
GRANT SELECT ON CurrentMenuSalesStatsView to Finanse
GRANT SELECT ON TotalCustomersDiscountsView to Finanse
GRANT SELECT ON OrderStatisticsView to Finanse
```

VIII. Role na funkcje

```
GRANT EXECUTE ON GetDetailsOfOrder to  Finanse, Waiter
GRANT EXECUTE ON GetStateOfOrder to  Finanse, Waiter
GRANT EXECUTE ON GetDataOfEmployeee to Finanse
GRANT EXECUTE ON GetOrdersAboveValue to Finanse
GRANT EXECUTE ON GetValueOfOrdersOnDay to Finanse
GRANT EXECUTE ON GetValueOfOrdersOnMonth to Finanse
GRANT EXECUTE ON GetValueOfOrder to Finanse
GRANT EXECUTE ON GetCheapestProductInCategory to Finanse
GRANT EXECUTE ON GetMostExpensiveProductInCategory to Finanse
GRANT EXECUTE ON GetIngredientsForProduct to StoreKeeper
GRANT EXECUTE ON GetProductsFromCategory to StoreKeeper
GRANT EXECUTE ON GetProductsFromMenu to Waiter, StoreKeeper
GRANT EXECUTE ON GetCurrentEmployeeSalary to Finanse
GRANT EXECUTE ON GetCurrentAverageSalaryForOccupation to Finanse
GRANT EXECUTE ON GetTotalProductsAndAveragePriceOfMenu to Finanse
GRANT EXECUTE ON GetHighestSalaryForEmployee to Finanse
GRANT EXECUTE ON RemainingDaysForMenu to Waiter, StoreKeeper
GRANT EXECUTE ON RemainingFreeSeats to Waiter
GRANT EXECUTE ON CanAccommodateCustomers to Waiter
```

IX. Role na procedury

```
GRANT EXECUTE ON AddCategoryProcedure to StoreKeeper
GRANT EXECUTE ON RemoveCategoryProcedure to StoreKeeper
GRANT EXECUTE ON AddProductProcedure to StoreKeeper
GRANT EXECUTE ON RemoveProductProcedure to StoreKeeper
GRANT EXECUTE ON AddEmployeeProcedure to Finanse
GRANT EXECUTE ON RemoveEmployeeProcedure to Finanse
GRANT EXECUTE ON AddIngredientToWarehouse to StoreKeeper
GRANT EXECUTE ON RemoveIngredientFromWarehouse to StoreKeeper
GRANT EXECUTE ON AddIngredientToProduct to StoreKeeper
GRANT EXECUTE ON RemoveIngredientFromProduct to StoreKeeper
GRANT EXECUTE ON AddProductToOrder to Finanse, Waiter
GRANT EXECUTE ON RemoveProductFromOrder to Finanse, Waiter
GRANT EXECUTE ON AddCustomerProcedure  to Finanse
GRANT EXECUTE ON RemoveCustomerProcedure  to Finanse
GRANT EXECUTE ON updateDiscountProcedure to Finanse
GRANT EXECUTE ON AddReservationProcedure to Finanse, Waiter
GRANT EXECUTE ON RemoveReservationProcedure to Finanse, Waiter
```

X. Dane które są dostępne tylko dla: Admin, Owner:

```
AddTableProcedure
RemoveTableProcedure
AddMenuProcedure
RemoveMenuProcedure
AddProductToMenuProcedure
RemoveProductFromMenuProcedure
```

10. Przykładowe dane

W załączniku dołączono plik generujący przykładowe dane.