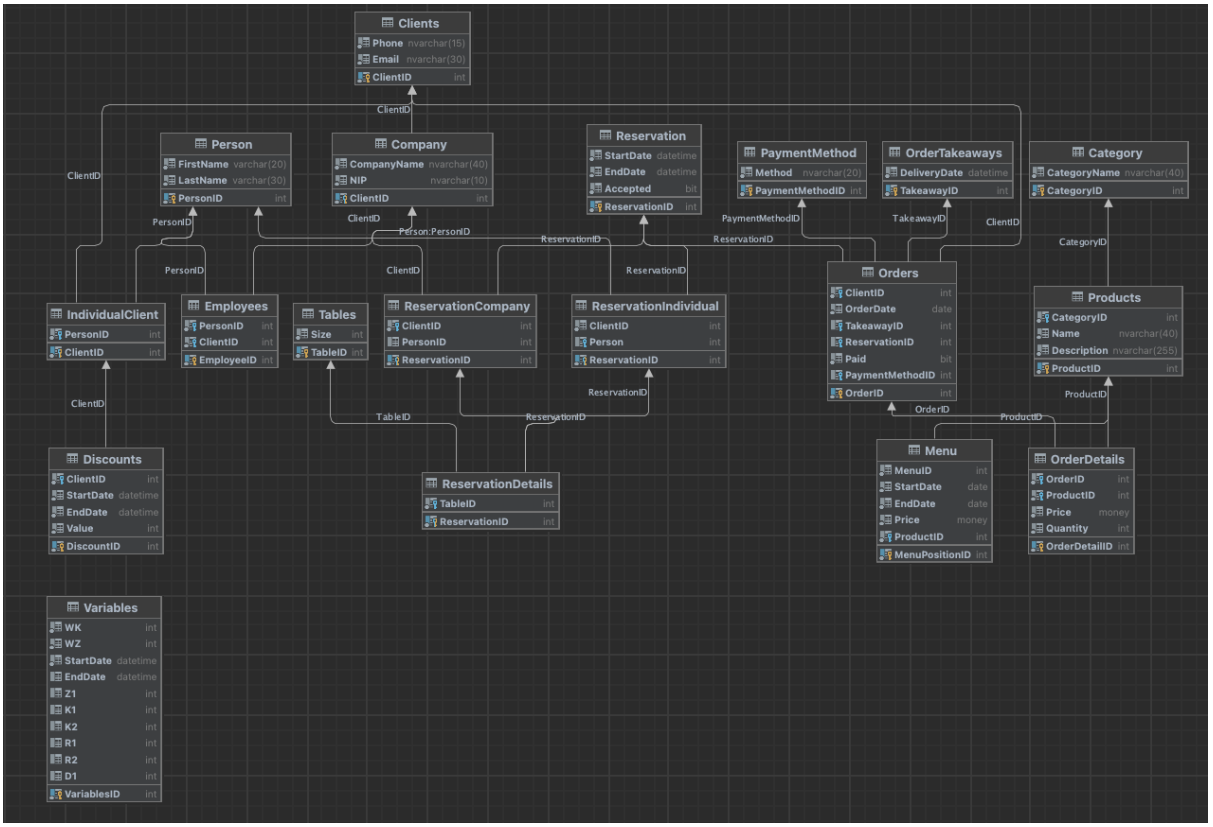
Projekt Restauracja

Podstawy baz danych

Magdalena Skrok, Jakub Żywiecki

Schemat bazy danych



FUNKCJE REALIZOWANE PRZEZ SYSTEM

1. Gość
   1.1 Ma możliwość wyświetlania aktualnego menu.
   1.2 Może składać zamówienie.
2. Pracownik
   2.1 Może zmienić status rezerwacji.
   2.2 Może dodać stolik do rezerwacji.
   2.3 Może odebrać zamówienie.
3. Menadżer
   3.1 Może sprawdzić poprawność menu.
   3.2 Może wystawić fakturę dla klienta.
   3.3 Może dodać produkty do menu.
4. Admin
   4.1 Ma dostęp do wszystkich funkcji systemu.

TABELE

1.1 Tabela Category

Zawiera kategorie pozycji z menu.

Opis tabeli:
Klucz główny: CategoryID
Nazwa kategorii: CategoryName

Kod generujący tabelę:

```
CREATE TABLE Category (
CategoryID int NOT NULL,
CategoryName nvarchar(40) NOT NULL,
CONSTRAINT Category_pk PRIMARY KEY (CategoryID)
);
```

1.2 Tabela Company

Zawiera dane dotyczące firm.

Opis tabeli:
Klucz główny i obcy: ClientID

Nazwa firmy: CompanyName
NIP firmy: NIP

Warunki integralności:
Numer NIP składa się z 12 cyfr od 0 do 9 i jest unikalny.

Kod generujący tabelę:

```
CREATE TABLE Company (
ClientID int NOT NULL,
CompanyName nvarchar(40) NOT NULL,
NIP nvarchar(10) NOT NULL,
CONSTRAINT ValidNIP CHECK (NIP LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT UniqueNIP UNIQUE (NIP),
CONSTRAINT Company_pk PRIMARY KEY (ClientID)
);
```

Relacje:

```
ALTER TABLE Company ADD CONSTRAINT Company_Clients
FOREIGN KEY (ClientID)
REFERENCES Clients (ClientID);
```

1.3 Tabela Reservation

Zawiera dane dotyczące wszystkich rezerwacji.

Opis tabeli:
Klucz główny i obcy: ReservationID
Data rozpoczęcia rezerwacji: StartDate
Data zakończenia rezerwacji: EndDate
Czy rezerwacja została zaakceptowana przez pracownika: Accepted

Warunki integralności:
Data początku rezerwacji musi poprzedzać datę zakończenia rezerwacji.

Kod generujący tabelę:

```
CREATE TABLE Reservation (
ReservationID int NOT NULL,
StartDate datetime NOT NULL,
EndDate datetime NOT NULL,
```

```
Accepted bit NOT NULL,
CONSTRAINT ValidReservationDate CHECK (EndDate > StartDate),
CONSTRAINT Reservation_pk PRIMARY KEY (ReservationID)
);
```

Relacje:

```
ALTER TABLE Reservation ADD CONSTRAINT Reservation_ReservationCompany
FOREIGN KEY (ReservationID)
REFERENCES ReservationCompany (ReservationID);
ALTER TABLE Reservation ADD CONSTRAINT
Reservation_ReservationIndividual
FOREIGN KEY (ReservationID)
REFERENCES ReservationIndividual (ReservationID);
```

1.4 Tabela ReservationCompany

Zawiera dane dotyczące rezerwacji dla firm.

Opis tabeli:
Klucz główny: ReservationID
Klucz obcy: ClientID
(Opcjonalnie) Wskazany pracownik, na którego dokonano rezerwacji: PersonID

Kod generujący tabelę:

```
CREATE TABLE ReservationCompany (
ClientID int NOT NULL,
ReservationID int NOT NULL,
PersonID int NULL,
CONSTRAINT ReservationCompany_pk PRIMARY KEY (ReservationID)
);
```

Relacje:

```
ALTER TABLE ReservationCompany ADD CONSTRAINT
ReservationCompany_Company
FOREIGN KEY (ClientID)
REFERENCES Company (ClientID);
```

1.5 Tabela ReservationDetails

Zawiera szczegóły rezerwacji.

Opis tabeli:
Klucz główny i obcy: ReservationID
Klucz obcy: TableID

Kod generujący tabelę:

```
CREATE TABLE ReservationDetails (
ReservationID int NOT NULL,
TableID int NOT NULL,
CONSTRAINT ReservationDetails_pk PRIMARY KEY (ReservationID)
);
```

Relacje:

```
ALTER TABLE ReservationDetails ADD CONSTRAINT
ReservationDetails_ReservationCompany
FOREIGN KEY (ReservationID)
REFERENCES ReservationCompany (ReservationID);
ALTER TABLE ReservationDetails ADD CONSTRAINT
ReservationDetails_ReservationIndividual
FOREIGN KEY (ReservationID)
REFERENCES ReservationIndividual (ReservationID);
ALTER TABLE ReservationDetails ADD CONSTRAINT ReservationDetails_Tables
FOREIGN KEY (TableID)
REFERENCES Tables (TableID);
```

1.6 Tabela ReservationIndividual

Zawiera dane dotyczące rezerwacji dla klientów indywidualnych.

Opis tabeli:
Klucz główny: ReservationID
ID klienta indywidualnego, na którego dokonana jest rezerwacja: ClientID
ID osoby, na którą dokonana jest rezerwacja: PersonID

Kod generujący tabelę:

```
CREATE TABLE ReservationIndividual (
ClientID int NOT NULL,
ReservationID int NOT NULL,
PersonID int NOT NULL,
CONSTRAINT ReservationIndividual_pk PRIMARY KEY (ReservationID)
);
```

1.7 Tabela Tables

Zawiera dane dotyczące stolików dostępnych w restauracji.

Opis tabeli:
Klucz główny: TableID
Rozmiar stołu: Size

Kod generujący tabelę:

```
CREATE TABLE Tables (
TableID int NOT NULL,
Size int NOT NULL,
CONSTRAINT Tables_pk PRIMARY KEY (TableID)
);
```

1.8 Tabela Variables

Zawiera dane dotyczące zmiennych w bazie danych.

Opis tabeli:
Klucz główny: VariablesID
Data rozpoczęcia: StartDate
Data zakończenia: EndDate
Minimalna liczba zamówień: WK
Minimalna wartość zamówienia: WZ

Warunki integralności:
Data zakończenia musi następować po dacie rozpoczęcia. Dodaliśmy również obsługę przypadku braku daty zakończenia.
WK oraz WZ muszą być większe od 0.

Kod generujący tabelę:

```
CREATE TABLE Variables (
VariablesID int NOT NULL,
WK int NOT NULL,
WZ int NOT NULL,
StartDate int NOT NULL,
EndDate int NULL,
CONSTRAINT ValidVariables CHECK (WZ > 0 AND WK > 0 AND ISNULL(EndDate,
'3000-01-01 23:59:59') > StartDate ),
CONSTRAINT Variables_pk PRIMARY KEY (VariablesID)
);
```

1.9 Tabela Employees

Zawiera dane dotyczące pracowników firm.

Opis tabeli:
Klucz główny: EmployeeID
Klucze obce: ClientID, PersonID

Kod generujący tabelę:

```
CREATE TABLE Employees (
PersonID int NOT NULL,
EmployeeID int NOT NULL,
ClientID int NOT NULL,
CONSTRAINT Employees_pk PRIMARY KEY (PersonID)
);
```

Relacje:

```
ALTER TABLE Employees ADD CONSTRAINT Employees_Company
FOREIGN KEY (ClientID)
REFERENCES Company (ClientID);
ALTER TABLE Employees ADD CONSTRAINT Employees_Person
FOREIGN KEY (PersonID)
REFERENCES Person (PersonID);
```

1.20 Tabela Person

Zawiera dane dotyczące osób w bazie danych.

Opis tabeli:
Klucz główny: PersonID
Imię osoby: FirstName
Nazwisko osoby: LastName

Kod generujący tabelę:

```
CREATE TABLE Person (
PersonID int NOT NULL,
FirstName varchar(20) NOT NULL,
LastName varchar(30) NOT NULL,
CONSTRAINT Person_pk PRIMARY KEY (PersonID)
);
```

1.21 Tabela Menu

Zawiera dane dotyczące menu w bazie danych.

Opis tabeli:
Klucz główny: MenuID
Klucz obcy: ProductID
Początek ważności: StartDate
Koniec ważności: EndDate
Cena: Price

Warunki integralności:
Data początku ważności musi być wcześniejsza, niż data końca ważności. Cena musi być większa od 0.

Kod generujący tabelę:

```
CREATE TABLE Menu (
MenuID int NOT NULL,
StartDate date NOT NULL,
EndDate date NOT NULL,
Price money NOT NULL CHECK (Price>0),
ProductID int NOT NULL,
CONSTRAINT dateCheck CHECK (StartDate<EndDate ),
CONSTRAINT Menu_pk PRIMARY KEY (MenuID)
);
```

Relacje:

```
ALTER TABLE Menu ADD CONSTRAINT Menu_Products
FOREIGN KEY (ProductID)
REFERENCES Products (ProductID);
```

1.22 Tabel Products

Zawiera dane dotyczące produktów w bazie danych.

Opis tabeli:
Klucz główny: ProductID
Klucz obcy: CategoryID
Nazwa produktu: Name
Opis produktu: Description

Kod generujący tabelę:

```
CREATE TABLE Products (
```

```
ProductID int NOT NULL,
CategoryID int NOT NULL,
Name nvarchar(40) NOT NULL,
Description nvarchar(255) NOT NULL,
CONSTRAINT Products_pk PRIMARY KEY (ProductID)
);
```

Relacje:

```
ALTER TABLE Products ADD CONSTRAINT Products_Category
FOREIGN KEY (CategoryID)
REFERENCES Category (CategoryID);
```

1.23 Tabela OrderDetails

Zawiera dane dotyczące szczegółów zamówienia w bazie danych.

Opis tabeli:
Klucz główny i obcy: OrderID
Cena całego zamówienia: Price
Ilość zamawianego produktu: Quantity

Warunki integralności:
Cena musi być większa od 0.

Kod generujący tabelę:

```
CREATE TABLE OrderDetails (
OrderID int NOT NULL,
ProductID int NOT NULL,
Price money NOT NULL CHECK (Price>0),
Quantity int NOT NULL,
CONSTRAINT OrderDetails_pk PRIMARY KEY (OrderID)
);
```

Relacje:

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Products
FOREIGN KEY (ProductID)
REFERENCES Products (ProductID);
```

1.24 Tabela OrderTakeaways

Zawiera dane dotyczące zamówień na wynos w bazie danych.

Opis tabeli:
Klucz główny: TakeawayID
Data odbioru zamówienia: DeliceryDate

Kod generujący tabelę:

```
CREATE TABLE OrderTakeaways (
TakeawayID int NOT NULL,
DeliveryDate datetime NOT NULL,
CONSTRAINT OrderTakeaways_pk PRIMARY KEY (TakeawayID)
);
```

1.25 Tabela Orders

Zawiera dane dotyczące szczegółów zamówienia w bazie danych.

Opis tabeli:
Klucz główny: OrderID
Klucze obce: ClientID, PaymentMethodID
(Opcjonalnie) Klucze obce: TakeawayID, ReservationID
Data złożenia zamówienia: OrderDate
Informacja czy zapłacono: Paid

Kod generujący tabelę:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
ClientID int NOT NULL,
OrderDate date NOT NULL,
TakeawayID int NULL,
ReservationID int NULL,
Paid bit NOT NULL,
PaymentMethodID int NOT NULL,
CONSTRAINT Orders_pk PRIMARY KEY (OrderID)
);
```

Relacje:

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Clients
FOREIGN KEY (ClientID)
REFERENCES Clients (ClientID);
ALTER TABLE Orders ADD CONSTRAINT Orders_OrderTakeaways
```

```
FOREIGN KEY (TakeawayID)
REFERENCES OrderTakeaways (TakeawayID);
ALTER TABLE Orders ADD CONSTRAINT Orders_PaymentMethod
FOREIGN KEY (PaymentMethodID)
REFERENCES PaymentMethod (PaymentMethodID);
ALTER TABLE Orders ADD CONSTRAINT Orders_Reservation
FOREIGN KEY (ReservationID)
REFERENCES Reservation (ReservationID);
```

1.26 Tabela PaymentMethod

Zawiera dane dotyczące metod płatności w bazie danych.

Opis tabeli:
Klucz główny: PaymentMethodID
Metoda płatności: Method

Kod generujący tabelę:

```
CREATE TABLE PaymentMethod (
PaymentMethodID int NOT NULL,
Method nvarchar(20) NOT NULL,
CONSTRAINT PaymentMethod_pk PRIMARY KEY (PaymentMethodID)
);
```

1.27 Tabela Clients

Zawiera szczegółowe dane dotyczące klientów w bazie danych.

Opis tabeli:
Klucz główny: ClientID
Numer telefonu klienta: Phone
Adres e-mail klienta: Email

Warunki integralności:
Adres e-mail musi zawierać znak ‘@’ oraz być unikalny. Telefon musi składać się z ciągu 9 cyfr oraz być unikalny.

Kod generujący tabelę:

```
CREATE TABLE Clients (
ClientID int NOT NULL,
Phone varchar(15) NOT NULL,
Email varchar(30) NOT NULL,
```

```
CONSTRAINT Phone CHECK (Phone LIKE
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT UniquePhone UNIQUE (Phone),
CONSTRAINT Email CHECK  (Email LIKE '%@%'),
CONSTRAINT UniqueEmail UNIQUE (Email),
CONSTRAINT Clients_pk PRIMARY KEY (ClientID)
);
```

1.28 Tabela Discounts

Zawiera dane dotyczące zniżek w bazie danych.

Opis tabeli:
Klucz główny: DiscountID
Klucz obcy: ClientID
Początek ważności zniżki: StartDate
Koniec ważności zniżki: EndDate
Wartość zniżki: Value

Warunki integralności:
Data początku ważności zniżki musi być wcześniejsza, niż data końca ważności zniżki. Wartość musi być większa od 0.

Kod generujący tabelę:

```
CREATE TABLE Discounts (
DiscountID int NOT NULL,
ClientID int NOT NULL,
StartDate datetime NOT NULL,
EndDate datetime NOT NULL,
Value int NOT NULL  CHECK (Value>0),
CONSTRAINT dateCheck CHECK (StartDate<EndDate),
CONSTRAINT Discounts_pk PRIMARY KEY (DiscountID)
);
```

Relacje:

```
ALTER TABLE Discounts ADD CONSTRAINT Discounts_IndividualClient
FOREIGN KEY (ClientID)
REFERENCES IndividualClient (ClientID);
```

1.29 Tabela IndividualClient

Zawiera dane dotyczące indywidualnego klienta w bazie danych.

Opis tabeli:
Klucz główny i obcy: ClientID
Klucz obcy: PersonID

Kod generujący tabelę:

```
CREATE TABLE IndividualClient (
ClientID int NOT NULL,
PersonID int NOT NULL,
CONSTRAINT IndividualClient_pk PRIMARY KEY (ClientID)
);
```

Relacje:

```
ALTER TABLE IndividualClient ADD CONSTRAINT IndividualClient_Clients
FOREIGN KEY (ClientID)
REFERENCES Clients (ClientID);
ALTER TABLE IndividualClient ADD CONSTRAINT IndividualClient_Person
FOREIGN KEY (PersonID)
REFERENCES Person (PersonID);
```

WIDOKI

2.1 Statystyka klientów

```
CREATE VIEW ClientStats AS
SELECT C.ClientID, Phone, Email, COUNT(O.OrderID) as orders_ammount,
(SELECT value FROM (SELECT SUM(val) as value FROM
(SELECT ClientID, OD.Price * (1 -
[dbo].[udfGetClientDiscountAtDate](OrderDate, ClientID)) as val FROM
OrderDetails OD INNER JOIN Orders O on O.OrderID =
OD.OrderID) as CIvv) as CIv) as total_value
FROM Clients AS C
INNER JOIN Orders O
ON C.ClientID = O.ClientID
GROUP BY C.ClientID, Phone, Email
```

2.2 Informacje o kategoriach i posiłkach

```
CREATE VIEW MealsInformations
AS
SELECT C.CategoryName, P.Name, P.Description
```

```
FROM Products AS P
INNER JOIN Category C on P.CategoryID = C.CategoryID
GO
```

2.3 Zniżki przyznane w tym tygodniu

```
CREATE VIEW DiscountsThisWeek AS
SELECT DiscountID,
C.ClientID,
P.FirstName,
P.LastName,
StartDate,
EndDate,
Value
FROM Discounts D
INNER JOIN Clients C on C.ClientID = D.ClientID
INNER JOIN IndividualClient IC on C.ClientID = IC.ClientID
INNER JOIN Person P on IC.PersonID = P.PersonID
WHERE  DatePart(week, StartDate) = DatePart(week,GETDATE())
go
```

2.4 Wszystkie przyznane zniżki

```
CREATE VIEW DiscountInfoView AS
SELECT DiscountID,
C.ClientID,
P.FirstName,
P.LastName,
StartDate,
EndDate,
Value
FROM Discounts D
INNER JOIN Clients C on C.ClientID = D.ClientID
INNER JOIN IndividualClient IC on C.ClientID = IC.ClientID
INNER JOIN Person P on IC.PersonID = P.PersonID
go
```

2.5 Zamówienia z danego dnia

```
CREATE VIEW OrdersToday AS
SELECT OrderID, ClientID, OrderDate, Paid, TakeawayID, ReservationID
FROM Orders
WHERE CONVERT(DATE, OrderDate) = CONVERT(DATE, GETDATE())
```

```
GO
```

2.6 Rezerwacje oczekujące na potwierdzenie

```
CREATE VIEW ReservationsToAccept AS
SELECT R.ReservationID, TableID, StartDate, EndDate FROM Reservation AS
R
INNER JOIN ReservationDetails RD
on RD.ReservationID = R.ReservationID
WHERE Accepted IS NULL
GO
```

2.7 Firmy, które zrobiły rezerwacje na przyszły tydzień

```
CREATE VIEW CompaniesReservationsWeekly AS
SELECT CompanyName, R.StartDate, R.EndDate FROM Company AS C
INNER JOIN ReservationCompany RC on RC.ClientID = C.ClientID
INNER JOIN Reservation R on R.ReservationID = RC.ReservationID
WHERE DatePart(week, R.StartDate) > DatePart(week,GETDATE())
GO
```

2.8 Informacje o rezerwacjach

```
CREATE VIEW ReservationsInfoView AS
SELECT R.ReservationID, TableID, StartDate, EndDate FROM Reservation AS
R
INNER JOIN ReservationDetails RD
on RD.ReservationID = R.ReservationID
GO
```

2.9 Aktualne menu

```
CREATE VIEW ActualMenu AS
SELECT P.Name, M.Price
FROM Products AS P
INNER JOIN Menu AS M
ON M.ProductID=P.ProductID
WHERE M.StartDate<=GETDATE() AND M.EndDate>=GETDATE()
GO
```

2.10 Informacje o wszystkich istniejących menu

```
CREATE VIEW Menus AS
SELECT M.MenuID, P.Name, M.Price, M.StartDate, M.EndDate
FROM Menu AS M
INNER JOIN Products AS P
ON M.ProductID=P.ProductID
ORDER BY M.MenuID
GO
```

2.11 Zamówienia

```
CREATE VIEW OrdersView AS
SELECT O.OrderID, O.OrderDate, SUM(OD.Price*OD.Quantity) as OrderValue
FROM Orders AS O
INNER JOIN OrderDetails AS OD
ON O.OrderID=OD.OrderID
GROUP BY O.OrderID, O.OrderDate
GO
```

2.12 Zamawiające firmy

```
CREATE VIEW CompaniesView AS
SELECT C.CompanyName, C.NIP, COUNT(O.OrderID) as ilosc_zamowien
FROM Company AS C
INNER JOIN CLients AS CL
ON C.ClientID=CL.ClientID
INNER JOIN Orders AS O
ON O.ClientID=CL.ClientID
GROUP BY C.CompanyName, C.NIP
GO
```

2.13 Zamówienia oczekujące na odbiór

```
CREATE VIEW WaitingsOrders AS
SELECT O.OrderID, O.OrderDate, OT.DeliveryDate
FROM Orders AS O
INNER JOIN OrderTakeaways AS OT
ON OT.TakeawayID=O.TakeawayID
WHERE O.TakeawayID IS NOT NULL AND OT.DeliveryDate>=GETDATE()
GO
```

2.14 Zarezerwowane stoliki

```
CREATE VIEW TablesReservation AS
SELECT R.ReservationID, R.StartDate, R.EndDate, T.TableID, T.Size
FROM Reservation AS R
INNER JOIN ReservationDetails AS RD
ON R.ReservationID=RD.ReservationID
INNER JOIN Tables AS T
ON T.TableID=RD.TableID
GO
```

2.15 Zbiór form płatności używanych w tym tygodniu

```
CREATE VIEW WeekPaymentMethod AS
SELECT P.Method
FROM PaymentMethod AS P
INNER JOIN Orders AS O
ON O.PaymentMethodID=P.PaymentMethodID
WHERE Paid=1 AND DatePart(week, O.OrderDate)=DatePart(week,GETDATE())
GO
```

2.16 Widok maksymalnych zniżek dla poszczególnych klientów w dniu obecnym

```
CREATE VIEW ClientsDiscounts AS
SELECT ClientID, MAX(Value) as znizka FROM Discounts
WHERE GETDATE() <= StartDate AND GETDATE() <= EndDate
GROUP BY ClientID
go
```

FUNKCJE

3.1 Zwraca tabele z zamówieniami ponad określoną wartością

```
CREATE FUNCTION udfGetOrdersAbove(@input int)
RETURNS table AS
RETURN
SELECT OrdersView.OrderID, OrdersView.OrderDate, OrdersView.OrderValue
FROM OrdersView
WHERE OrdersView.OrderValue > @input
```

```
go
```

3.2 Zwraca tabele produktów z menu po ID

```
CREATE FUNCTION udfGetMenuItemsById(@id int)
RETURNS TABLE AS
RETURN
SELECT M.MenuID, P.Name, M.Price, M.StartDate, M.EndDate
FROM Products P
INNER JOIN Menu M
ON M.ProductID = P.ProductID
WHERE (M.MenuID = @id)
go
```

3.3 Zwraca tabele produktów z menu po dacie

```
CREATE FUNCTION udfGetMenuItemsByDate(@date date)
RETURNS TABLE AS
RETURN
SELECT M.MenuID, P.Name, M.Price, M.StartDate, M.EndDate
FROM Products P
INNER JOIN Menu M
ON M.ProductID = P.ProductID
WHERE @date BETWEEN M.StartDate AND M.EndDate
go
```

3.4 Zwraca tabele produktów z menu po kategorii

```
CREATE FUNCTION udfGetMenuItemsByCategory(@CategoryName nvarchar(40))
RETURNS TABLE AS
RETURN
SELECT M.MenuID, P.Name, M.Price, M.StartDate, M.EndDate
FROM Products P
INNER JOIN Menu M
ON M.ProductID = P.ProductID
INNER JOIN Category C
ON C.CategoryID = P.CategoryID
WHERE @CategoryName = C.CategoryName
go
```

3.5 Zwraca tabele z pracownikami danej danej firmy

```
CREATE FUNCTION udfGetCompanyEmployees(@CompanyName nvarchar(40))
RETURNS table AS
RETURN
SELECT P.Firstname, P.Lastname
FROM Person P
INNER JOIN Employees E on P.PersonID = E.PersonID
INNER JOIN Company C on E.ClientID = C.ClientID
WHERE @CompanyName = CompanyName
go
```

3.6 Zwraca najwyższą cenę produktu w danym menu

```
CREATE FUNCTION udfGetMaxMenuPrice(@MenuID int)
RETURNS money
AS
BEGIN
RETURN (SELECT TOP 1 MAX(Menu.Price) FROM Menu WHERE MenuID = @MenuID)
END
go
```

3.7 Zwraca najniższą cenę produktu w danym menu

```
CREATE FUNCTION udfGetMinMenuPrice(@MenuID int)
RETURNS money
AS
BEGIN
RETURN (SELECT TOP 1 MIN(Menu.Price) FROM Menu WHERE MenuID = @MenuID)
END
go
```

3.8 Zwraca wartość zamówień podczas danego miesiąca (jako argument przyjmuje konkretną datę)

```
CREATE FUNCTION udfgetOrdersValueMonthly(@date date)
RETURNS int
AS
BEGIN
RETURN (SELECT SUM(OD.Price * OD.Quantity * (1 - CD.znizka))
FROM OrderDetails OD
INNER JOIN Orders O on OD.OrderID = O.OrderID
```

```
INNER JOIN ClientsDiscounts CD on O.ClientID = CD.ClientID
WHERE (YEAR(@date) = YEAR(O.OrderDate)
AND MONTH(@date) = MONTH(O.OrderDate)))
END
go
```

3.9 Zwraca wartość zamówień podczas danego dnia

```
CREATE FUNCTION udfgetOrdersValueDaily(@date date)
RETURNS int
AS
BEGIN
RETURN (SELECT SUM(OD.Price * OD.Quantity * (1 - CD.znizka))
FROM OrderDetails OD
INNER JOIN Orders O on OD.OrderID = O.OrderID
INNER JOIN ClientsDiscounts CD on O.ClientID = CD.ClientID
WHERE YEAR(@date) = YEAR(O.OrderDate)
AND MONTH(@date) = MONTH(O.OrderDate)
AND DAY(@date) = DAY(O.OrderDate))
END
go
```

3.10 Zwraca wartość zamówień podczas danego roku (jako argument przyjmuje konkretną datę)

```
CREATE FUNCTION udfgetOrdersValueYearly(@date date)
RETURNS int
AS
BEGIN
RETURN (SELECT SUM(OD.Price * OD.Quantity * (1 - CD.znizka))
FROM OrderDetails OD
INNER JOIN Orders O on OD.OrderID = O.OrderID
INNER JOIN ClientsDiscounts CD on O.ClientID = CD.ClientID
WHERE YEAR(@date) = YEAR(O.OrderDate))
END
go
```

3.11 Zwraca tabelę zawierającą X najczęściej kupowanych produktów

```
CREATE FUNCTION udfGetAmmountOfSoldMeals(@input int)
RETURNS table AS
```

```
RETURN
SELECT DISTINCT TOP (@input) P.Name, SUM(Quantity) as ilosc
FROM Products P
INNER JOIN OrderDetails OD on P.ProductID =
OD.ProductID
INNER JOIN Orders O on ODD.Order ID = O.OrderID
GROUP BY P.Name
ORDER BY SUM(Quantity)
go
```

3.12 Sprawdza czy menu jest poprawne

```
CREATE FUNCTION udfMenuCorrect(@id int)
RETURNS int
AS
BEGIN
DECLARE @same int
SET @same = (SELECT COUNT(*)
FROM (
SELECT ProductID
FROM Menu
WHERE MenuID = (@id - 1)
INTERSECT
SELECT ProductID
FROM Menu
WHERE MenuID = @id
) out)
DECLARE @minAmountToChange int
SET @minAmountToChange = (SELECT COUNT(*) FROM Menu WHERE MenuID=(@id
-1))/2
IF @same <= @minAmountToChange
BEGIN
return 1
END
return 0
END
go
```

3.13 Zwraca tabelę zawierającą najpopularniejsze x stolików

```
CREATE FUNCTION udfXMostPopularTables(@input int)
RETURNS TABLE AS
RETURN
```

```
SELECT DISTINCT TOP (@input) RD.TableID, COUNT(RD.ReservationID) as
ilosc_rezerwacji
FROM ReservationDetails RD
go
```

3.14 Zwraca tabelę klientów, którzy zamawiali co najmniej x razy

```
CREATE FUNCTION udfClientsOrderedMoreThanXTimes(@amount int)
RETURNS TABLE AS
RETURN
SELECT *
FROM ClientStats
WHERE orders_ammount> @amount
go
```

3.15 Zwraca tabelę klientów, którzy wydali co najmniej x pieniędzy

```
CREATE FUNCTION udfClientsSpentMoreThanX(@spent money)
RETURNS TABLE AS
RETURN
SELECT *
FROM ClientStats
WHERE total_value > @spent
Go
```

3.16 Zwraca zniżkę danego klienta w danym czasie

```
CREATE FUNCTION udfGetClientDiscountAtDate(@date date, @clientID int)
RETURNS int
AS
BEGIN
RETURN (SELECT MAX(value)
FROM Discounts
WHERE (@Date between StartDate and EndDate) and ClientID = @clientID)
END
go
```

3.17 Widok zarezerwowanych stolików w dacie

```
CREATE FUNCTION udfGetReservedTablesAtDate(@date date)
RETURNS table AS
RETURN
SELECT TableID FROM ReservationDetails
INNER JOIN Reservation R on ReservationDetails.ReservationID =
R.ReservationID
WHERE @date between R.StartDate and R.EndDate
go
```

3.18 Faktura miesięczna

```
CREATE FUNCTION udfInvoiceMonthly(@CompanyName nvarchar(30), @StartDate
date, @EndDate date)
RETURNS TABLE AS
RETURN
SELECT OrderDate, Paid, PaymentMethodID, Phone, Email, CompanyName,
NIP, SUM(Price - [dbo].[udfGetClientDiscountAtDate](OrderDate,
C2.ClientID)) as suma_zamowien FROM Orders
INNER JOIN Clients C on C.ClientID = Orders.ClientID
LEFT JOIN OrderDetails OD on Orders.OrderID = OD.OrderID
INNER JOIN Company C2 on C.ClientID = C2.ClientID
WHERE CompanyName = @CompanyName and OrderDate between @StartDate and
@EndDate
GROUP BY OrderDate, Paid, PaymentMethodID, Phone, Email, CompanyName,
NIP
GO
```

3.19 Faktura dla zamówienia

```
CREATE FUNCTION udfInvoiceForOrder(@OrderID int)
RETURNS TABLE AS
RETURN
SELECT OrderDate, Paid, PaymentMethodID, Phone, Email, CompanyName,
NIP, Price - [dbo].[udfGetClientDiscountAtDate](OrderDate, C2.ClientID)
as suma  FROM Orders
INNER JOIN Clients C on C.ClientID = Orders.ClientID
LEFT JOIN OrderDetails OD on Orders.OrderID = OD.OrderID
INNER JOIN Company C2 on C.ClientID = C2.ClientID
WHERE Orders.OrderID = @OrderID
GO
```

PROCEDURY

4.1 Dodaje nową kategorię do tabeli Categories

```
CREATE PROCEDURE pAddCategory
@CategoryName nvarchar(40)
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF EXISTS(
SELECT *
FROM Category
WHERE @CategoryName = CategoryName
)
BEGIN
;
THROW 52000, N'Kategoria jest już dodana', 1
end
DECLARE @CategoryID INT
SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
FROM Category
INSERT INTO Category(CategoryID, CategoryName)
VALUES(@CategoryID, @CategoryName);
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048) =
N'Błąd dodawania kategorii: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
END CATCH
END
Go
```

4.2 Dodaje nową firmę do Company and Clients

```
CREATE PROCEDURE pAddCompany @CompanyName nvarchar(40), @NIP
nvarchar(12),
@Phone nvarchar(15), @Email nvarchar(30)
AS
BEGIN
```

```
SET NOCOUNT ON
BEGIN TRY
IF EXISTS(
SELECT *
FROM Clients
WHERE Phone = @Phone
)
BEGIN
;
THROW 52000, N'Istnieje już klient z takim numerem telefonu', 1
END
IF EXISTS(
SELECT *
FROM Clients
WHERE Email = @Email
)
BEGIN
;
THROW 52000, N'Istnieje już klient z takim adresem email', 1
END
IF EXISTS(
SELECT *
FROM Company
WHERE NIP = @NIP
)
BEGIN
;
THROW 52000, N'Istnieje już klient z takim numerem NIP', 1
END
DECLARE @ClientID INT
SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
FROM Clients
INSERT INTO Company(ClientID, CompanyName, NIP)
VALUES (@ClientID, @CompanyName, @NIP);
INSERT INTO Clients(ClientID, Phone, Email)
VALUES (@ClientID, @Phone, @Email);
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Błąd dodania potrawy: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
END CATCH
END
```

```
Go
```

4.3 Dodaje nowy produkt do tabeli Products

```
CREATE PROCEDURE pAddProduct @Name nvarchar(40),
@CategoryName nvarchar(40), @Description nvarchar(255)
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF EXISTS(
SELECT *
FROM Products
WHERE Name = @Name
)
BEGIN
;
THROW 52000, N'Potrawa jest już dodana', 1
END
IF NOT EXISTS(
SELECT *
FROM Category
WHERE CategoryName = @CategoryName
)
BEGIN
;
THROW 52000, 'Nie ma takiej kategorii', 1
END
DECLARE @CategoryID INT
SELECT @CategoryID = CategoryID
FROM Category
WHERE CategoryName = @CategoryName
DECLARE @ProductID INT
SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
FROM Products
INSERT INTO Products(ProductID, Name, CategoryID, Description)
VALUES (@ProductID, @Name, @CategoryID, @Description);
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Błąd dodania potrawy: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
```

```
END CATCH
END
Go
```

4.4 Dodaje produkt do Menu

```
CREATE PROCEDURE pAddProductToMenu @Name nvarchar(40),
@Price money,
@MenuID int
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF NOT EXISTS(
SELECT *
FROM Products
WHERE Name = @Name
)
BEGIN
;
THROW 52000, 'Nie ma takiej potrawy', 1
END
IF NOT EXISTS(
SELECT *
FROM Menu
WHERE MenuID = @MenuID
)
BEGIN
;
THROW 52000, 'Nie ma takiego menu', 1
END
DECLARE @ProductID INT
SELECT @ProductID = ProductID
FROM Products
WHERE Name = @Name
DECLARE @StartDate date
SELECT TOP 1 @StartDate = StartDate
FROM Menu
WHERE MenuID = @MenuID
DECLARE @EndDate date
SELECT TOP 1 @EndDate = EndDate
FROM Menu
WHERE MenuID = @MenuID
```

```
IF DATEDIFF(day,GETDATE(),@StartDate)<1
BEGIN
;
THROW 5200, 'Produkt powinien być dodany z co najmniej jednodniowym
wyprzedzeniem!', 1
END
INSERT INTO Menu(MenuID, StartDate, EndDate, Price, ProductID)
VALUES (@MenuID, @StartDate, @EndDate, @Price, @ProductID);
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N 'Błąd dodania potrawy do menu: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
go
```

4.5 Dodaje stolik do rezerwacji

```
CREATE PROCEDURE pAddTableToReservation
@ReservationID int,
@TableID int
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF NOT EXISTS(
SELECT *
FROM Tables
WHERE TableID = @TableID
)
BEGIN
;
THROW 52000, 'Nie ma takiego stolika', 1
END
IF NOT EXISTS(
SELECT *
FROM Orders
WHERE ReservationID = @ReservationID
)
BEGIN
;
THROW 52000, 'Nie ma takiej rezerwacji', 1
```

```
END

DECLARE @start DATE
DECLARE @end DATE
SELECT @start= StartDate
FROM Reservation
WHERE ReservationID=@ReservationID
SELECT @end= EndDate
FROM Reservation
WHERE ReservationID=@ReservationID
IF EXISTS (
SELECT *
FROM Reservation AS R
INNER JOIN ReservationDetails AS RD
ON R.ReservationID=RD.ReservationID
WHERE RD.TableID=@TableID AND ((@start<R.EndDate AND
@start>R.StartDate)OR(@end<R.EndDate AND @end>R.StartDate))
)
BEGIN
;
THROW 52000, 'Ten stolik jest juz zarezerwowany w tym terminie', 1
END
INSERT INTO ReservationDetails(ReservationID, TableID)
VALUES (@ReservationID,@TableID)
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Błąd dodania stolika do rezerwacji: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

4.6 Dodaje nowy stolik

```
CREATE PROCEDURE pAddTable
@Size int
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF @Size<2
BEGIN
;
```

```
THROW 52000, 'Za maly stolik', 1
END
DECLARE @TableID INT
SELECT @TableID = ISNULL(MAX(TableID), 0) + 1
FROM Tables
INSERT INTO Tables(TableID, Size)
VALUES(@TableID, @Size);
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N 'Błąd dodawania stolika: ' + ERROR_MESSAGE();
THROW 5200, @msg, 1
END CATCH
END
```

4.7 Zmienia status rezerwacji

```
CREATE PROCEDURE pChangeReservationStatus
@ReservationID int,
@Accepted bit
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
BEGIN
UPDATE Reservation
SET Accepted = @Accepted
WHERE Reservation.ReservationID=@ReservationID
END
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N 'Błąd edytowania rezerwacji: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

4.8 Dodaje produkt do zamówienia

```
CREATE PROCEDURE pAddProductToOrder @OrderID int,
@Quantity int,
```

```
@ProductName nvarchar(40)
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF NOT EXISTS(
SELECT *
FROM Products
WHERE Name = @ProductName
)
BEGIN
;
THROW 52000, 'Nie ma takiego produktu', 1
END
IF NOT EXISTS(
SELECT *
FROM Orders
WHERE OrderID = @OrderID
)
BEGIN
;
THROW 52000, 'Nie ma takiego zamówienia', 1
END
DECLARE @ProductID int
SELECT @ProductID=ProductID
FROM Products
WHERE Name=@ProductName
DECLARE @Price money
SELECT @Price=Price
FROM Menu
WHERE ProductID=@ProductID
IF NOT EXISTS(
SELECT *
FROM ActualMenu
WHERE Name = @ProductName
)
BEGIN
;
THROW 52000, 'Nie można zamówić tego produktu, gdyż nie ma go aktualnie
w menu', 1
END
IF EXISTS (
SELECT *
```

```
FROM Products P
INNER JOIN Category C on P.CategoryID = C.CategoryID
WHERE @ProductName = P.Name AND C.CategoryName LIKE 'owoce morza'
)
BEGIN
;
IF DATEDIFF(DAY, GETDATE(),  @OrderDate)<3 OR  (DATEDIFF(DAY,
GETDATE(),  @OrderDate)=4 AND DAY(GETDATE())!=7)  OR  (DATEDIFF(DAY,
GETDATE(),  @OrderDate)=3 AND DAY(GETDATE())!=1)
BEGIN
;
THROW 52000, N'Nieprawidłowa data zamówienia na owoce morza', 1
END

DECLARE @OrderDate DATE
SELECT @OrderDate = OrderDate
FROM Orders
WHERE OrderID = @OrderID
F DATEPART(WEEKDAY ,@OrderDate) != 4 AND DATEPART(WEEKDAY ,
@OrderDate) != 5 AND DATEPART(WEEKDAY ,@OrderDate) != 6
BEGIN
;
THROW 52000, N'Nieprawidłowa data realizacji zamówienia na owoce
morza', 1
END
END
INSERT INTO OrderDetails(OrderID, ProductID, Price, Quantity)
VALUES (@OrderID,@ProductID,@Price*@Quantity,@Quantity)
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Błąd dodania produktu do zamówienia: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
GO
```

4.9 Usuwa kategorię

```
CREATE PROCEDURE pRemoveCategory
@CategoryName nvarchar(40)
AS
BEGIN
```

```
SET NOCOUNT ON
BEGIN TRY
IF NOT EXISTS(
SELECT *
FROM Category
WHERE CategoryName = @CategoryName
)
BEGIN
;
THROW 52000,'Nie ma takiej kategorii!',1
end
DELETE FROM Category
WHERE CategoryName = @CategoryName
END TRY
BEGIN CATCH
                    DECLARE @msg nvarchar(2048) =
N 'Błąd usuwania kategorii: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
END CATCH
END
```

4.10 Usuwa potrawę

```
CREATE PROCEDURE pRemoveProduct
@Name nvarchar(40)
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF NOT EXISTS(
SELECT *
FROM Products
WHERE Name = @Name
)
BEGIN
;
THROW 52000,'Nie ma takiego produktu!',1
end
DELETE FROM Products
WHERE Name = @Name
END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048) =
```

```
N 'Błąd usuwania produktu: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
END CATCH
END
```

4.11 Dodaje zamówienie

```
CREATE PROCEDURE pAddOrders
@ClientID int,
@Takeaway bit,
@DeliveryDate datetime = NULL,
@Reservation bit,
@StartDate datetime = NULL,
@EndDate datetime = NULL,
@Paid bit,
@PaymentMethod int,
@Products nvarchar(max)
--Produkty podawane jako [id,quantity][id,quantity]....
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY

   DECLARE @OrderDate date = GETDATE()
   DECLARE @OrderID int
   SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
   FROM Orders
   DECLARE @TakeawayID int = NULL
   DECLARE @ReservationID int = NULL

   IF @Reservation=1
   BEGIN
   EXEC [dbo].[pAddReservation] @ClientID, @StartDate, @EndDate, 1, @OrderID
   SELECT @ReservationID = ISNULL(MAX(ReservationID), 0)
   FROM Reservation
   END
   IF @Takeaway=1
   BEGIN
   SELECT @TakeawayID = ISNULL(MAX(TakeawayID), 0) + 1
   FROM OrderTakeaways
   INSERT INTO OrderTakeaways(TakeawayID, DeliveryDate)
   VALUES (@TakeawayID, @DeliveryDate)
   END
```

```
   INSERT INTO Orders(OrderID, ClientID, OrderDate, TakeawayID, ReservationID,
Paid, PaymentMethodID)
   VALUES (@OrderID, @ClientID, @OrderDate, @TakeawayID, @ReservationID, @Paid,
@PaymentMethod)

   DECLARE @len int = LEN(@Products)

   DECLARE @cnt INT = 1;
   DECLARE @OrderDetailID int
   WHILE @cnt <= @len
   BEGIN
      SET @cnt = @cnt + 1;
      DECLARE @ProductID int=SUBSTRING(@Products,@cnt,1)
      PRINT @ProductID
      SET @cnt = @cnt + 2;
      DECLARE @Quantity int=SUBSTRING(@Products,@cnt,1)
      SET @cnt = @cnt + 2;

      DECLARE @Price int
      SELECT @Price=Price
      FROM Menu
      WHERE ProductID=@ProductID

      SET @Price= @Price*@Quantity

      SET @OrderDetailID = (SELECT ISNULL(MAX(OrderDetailID)+1, 0) FROM
OrderDetails)
      INSERT INTO OrderDetails(OrderDetailID, OrderID, ProductID, Price, Quantity)
       VALUES (@OrderDetailID, @OrderID, @ProductID, @Price, @Quantity)
   END;


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

4.12 Dodaje rezerwacje

```
CREATE PROCEDURE [dbo].[pAddReservation]
@ClientID int,
@StartDate datetime,
@EndDate datetime,
@Status bit,
@OrderID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    Declare @WK int SELECT WK FROM Variables
    IF NOT EXISTS(
    SELECT * FROM [dbo].[udfClientsOrderedMoreThanXTimes](@WK)
    WHERE ClientID = @ClientID)
    BEGIN;
    THROW 52000, N'Klient nie spełnia WK', 1
    END
    Declare @WZ int SELECT WZ FROM Variables
    IF NOT EXISTS(
    SELECT * FROM Orders
    INNER JOIN OrderDetails OD on Orders.OrderID = OD.OrderID
    WHERE @OrderID = Orders.OrderID and @WZ <= OD.Price)
    BEGIN;
    THROW 52000, N'Zamówienie nie spełnia WZ', 1
    END
    IF NOT EXISTS(
    SELECT * FROM Clients
    WHERE ClientID = @ClientID)
    BEGIN;
    THROW 52000, 'Nie ma takiego klienta', 1
    END
    DECLARE @ReservationID int
    DECLARE @Person int
    SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 FROM
Reservation
    INSERT INTO Reservation(ReservationID, StartDate, EndDate,
Accepted)
    VALUES(@ReservationID, @StartDate, @EndDate, @Status);
    IF EXISTS(
    SELECT * FROM Company
    WHERE ClientID = @ClientID)
```

```
    BEGIN
    INSERT INTO ReservationCompany(ReservationID, ClientID, PersonID)
    VALUES (@ReservationID,@ClientID,null)
    END
    ELSE BEGIN
    SELECT @Person = PersonID
    FROM IndividualClient
    WHERE ClientID = @ClientID
    INSERT INTO ReservationIndividual(ReservationID, ClientID, Person)
    VALUES (@ReservationID,@ClientID,@Person)
    END
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar(2048)
    =N'Błąd dodania rezerwacji: ' + ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1
    END CATCH
END
GO
```

TRIGGERY

5.1 Sprawdza czy dodana nowa zmienna zniżki WZ jest poprawna (wartość większa od 0).

```
create trigger TR_MinOrdersVar on Variables
for insert
as
BEGIN
if (select COUNT(*) from inserted) > 1
BEGIN
RAISERROR('Dodawaj zniżki pojedynczo! ', 16, 1)
ROLLBACK TRANSACTION
END
else if (select WZ from inserted) <= 0
BEGIN
RAISERROR(' Wprowadzono niepoprawną wartość zniżki WZ', 16, 1)
ROLLBACK TRANSACTION
END
END
Go
```

5.2 Sprawdza czy dodana nowa zmienna zniżki WK jest poprawna (wartość większa od 0).

```
create trigger TR_MinOrdersVar1 on Variables
for insert
as
BEGIN
if (select COUNT(*) from inserted) > 1
BEGIN
RAISERROR('Dodawaj rabaty pojedynczo! ', 16, 1)
ROLLBACK TRANSACTION
END
else if (select WK from inserted) <= 0
BEGIN
RAISERROR('Wprowadzono niepoprawną wartość zniżki WK', 16, 1)
ROLLBACK TRANSACTION
END
END
go
```

INDEKSY

6.1 Indeks Category_pk

Ustawienie indeksu na CategoryID w tabeli Category

```
CREATE UNIQUE INDEX Category_pk
ON Category (CategoryID)
```

6.2 Indeks Products_pk

Ustawienie indeksu na ProductID w tabeli Products

```
CREATE UNIQUE INDEX Products_pk
ON Products (ProductID)
```

6.3 Indeks OrderDetails_pk

Ustawienie indeksu na OrderID w tabeli OrderDetails

```
CREATE UNIQUE INDEX OrderDetails_pk
ON OrderDetails (OrderDetailID)
```

6.4 Indeks Orders_pk

Ustawienie indeksu na OrderID w tabeli Orders

```
CREATE UNIQUE INDEX Orders_pk
ON Orders (OrderID)
```

6.5 Indeks OrderTakeaways_pk

Ustawienie indeksu na TakeawayID w tabeli OrderTakeaways

```
CREATE UNIQUE INDEX OrderTakeaways_pk
ON OrderTakeaways (TakeawayID)
```

6.6 Indeks PaymentMethod_pk

Ustawienie indeksu na PaymentMethodID w tabeli PaymentMethod

```
CREATE UNIQUE INDEX PaymentMethod_pk
ON PaymentMethod (PaymentMethodID)
```

6.7 Indeks Reservation_pk

Ustawienie indeksu na ReservationID w tabeli Reservation

```
CREATE UNIQUE INDEX Reservation_pk
ON Reservation (ReservationID)
```

6.8 Indeks ReservationIndividual_pk

Ustawienie indeksu na ClientID w tabeli ReservationIndividual

```
CREATE UNIQUE INDEX ReservationIndividual_pk
ON ReservationIndividual (ClientID)
```

6.9 Indeks ReservationCompany_pk

Ustawienie indeksu na ClientID w tabeli ReservationCompany

```
CREATE UNIQUE INDEX ReservationCompany_pk
ON ReservationCompany (ClientID)
```

6.10 Indeks ReservationDetails_pk

Ustawienie indeksu na ReservationID w tabeli ReservationDetails

```
CREATE UNIQUE INDEX ReservationDetails_pk
```

```
ON ReservationDetails (ReservationID)
```

6.11 Indeks Tables_pk

Ustawienie indeksu na TableID w tabeli Tables

```
CREATE UNIQUE INDEX Tables_pk
ON Tables(TableID)
```

6.12 Indeks Clients_pk

Ustawienie indeksu na ClientID w tabeli Clients

```
CREATE UNIQUE INDEX Clients_pk
ON Clients (ClientID)
```

6.13 Indeks Company_pk

Ustawienie indeksu na ClientID w tabeli Company

```
CREATE UNIQUE INDEX Company_pk
ON Company (ClientID)
```

6.14 Indeks IndividualClient_pk

Ustawienie indeksu na ClientID w tabeli IndividualClient

```
CREATE UNIQUE INDEX IndividualClient_pk
ON IndividualCLient(ClientID)
```

6.15 Indeks Employees_pk

Ustawienie indeksu na PersonID w tabeli Employees

```
CREATE UNIQUE INDEX Employees_pk
ON Employees(PersonID)
```

6.16 Indeks Person_pk

Ustawienie indeksu na PersonID w tabeli Person

```
CREATE UNIQUE INDEX Person_pk
ON Person(PersonID)
```

6.17 Indeks Discounts_pk

Ustawienie indeksu na DiscountID w tabeli Discounts

```
CREATE UNIQUE INDEX Discounts_pk
ON Discounts(DiscountID)
```

6.18 Indeks Variables_pk

Ustawienie indeksu na DiscountID w tabeli Variables

```
CREATE UNIQUE INDEX Variables_pk
ON Variables(DiscountID)
```

UPRAWNIENIA

KLIENT

Posiada uprawnienia do korzystania z obsługi restauracji.

```
CREATE ROLE Client
GRANT SELECT ON ActualMenu to Client
GRANT EXECUTE ON pAddOrders to Client
```

PRACOWNIK

Posiada uprawnienia niezbędne do obsługi klientów.

```
CREATE ROLE Worker
GRANT SELECT ON Category to Worker
GRANT SELECT ON Clients to Worker
GRANT SELECT ON Discounts to Worker
GRANT SELECT ON IndividualClient to Worker
GRANT SELECT ON Menu to Worker
GRANT SELECT ON OrderDetails to Worker
GRANT SELECT ON Orders to Worker
GRANT SELECT ON OrderTakeaways to Worker
GRANT SELECT ON PaymentMethod to Worker
GRANT SELECT ON Person to Worker
GRANT SELECT ON Products to Worker
GRANT SELECT ON Reservation to Worker
GRANT SELECT ON ReservationCompany to Worker
GRANT SELECT ON ReservationDetails to Worker
GRANT SELECT ON ReservationIndividual to Worker
GRANT SELECT ON Tables to Worker
GRANT SELECT ON Variables to Worker
```

```
GRANT SELECT ON ActualMenu to Worker
GRANT SELECT ON ClientsDiscounts to Worker
GRANT SELECT ON MealsInformations to Worker
GRANT SELECT ON OrdersToday to Worker
GRANT SELECT ON OrdersView to Worker
GRANT SELECT ON  ReservationsInfoView to Worker
GRANT SELECT, UPDATE ON ReservationsToAccept to Worker
GRANT SELECT ON TablesReservation to Worker
GRANT SELECT ON WaitingsOrders to Worker

GRANT SELECT ON udfInvoiceForOrder to Worker
GRANT SELECT ON udfInvoiceMonthly to Worker
GRANT EXECUTE ON pAddTableToReservation to Worker
GRANT EXECUTE ON pChangeReservationStatus to Worker
```

MENAGER

Menager może wyciągać informacje z bazy dotyczące restauracji, raporty, podsumowania.

```
GRANT SELECT ON Category to Menager
GRANT SELECT ON Clients to Menager
GRANT SELECT ON Discounts to Menager
GRANT SELECT ON IndividualClient to Menager
GRANT SELECT ON Menu to Menager
GRANT SELECT ON OrderDetails to Menager
GRANT SELECT ON Orders to Menager
GRANT SELECT ON OrderTakeaways to Menager
GRANT SELECT ON PaymentMethod to Menager
GRANT SELECT ON Person to Menager
GRANT SELECT ON Products to Menager
GRANT SELECT ON Reservation to Menager
GRANT SELECT ON ReservationCompany to Menager
GRANT SELECT ON ReservationDetails to Menager
GRANT SELECT ON ReservationIndividual to Menager
GRANT SELECT ON Tables to Menager
GRANT SELECT ON Variables to Menager

GRANT SELECT ON ActualMenu to Menager
GRANT SELECT ON ClientsDiscounts to Menager
GRANT SELECT ON MealsInformations to Menager
GRANT SELECT ON OrdersToday to Menager
```

```
GRANT SELECT ON OrdersView to Menager
GRANT SELECT ON ReservationsInfoView to Menager
GRANT SELECT, UPDATE ON ReservationsToAccept to Menager
GRANT SELECT ON TablesReservation to Menager
GRANT SELECT ON WaitingsOrders to Menager

GRANT SELECT ON udfInvoiceForOrder to Menager
GRANT SELECT ON udfInvoiceMonthly to Menager
GRANT EXECUTE ON pAddProductToMenu to Menager
GRANT EXECUTE ON pAddProductToOrder to Menager
GRANT EXECUTE ON pAddTableToReservation to Menager
GRANT EXECUTE ON pChangeReservationStatus to Menager
GRANT SELECT ON udfGetAmmountOfSoldMeals to Menager
GRANT SELECT ON udfGetCompanyEmployees to Menager
GRANT EXECUTE ON udfGetMaxMenuPrice to Menager
GRANT SELECT ON udfGetMenuItemsByCategory to Menager
GRANT SELECT ON udfGetMenuItemsByDate to Menager
GRANT SELECT ON udfGetMenuItemsById to Menager
GRANT EXECUTE ON udfGetMinMenuPrice to Menager
GRANT SELECT ON udfGetOrdersAbove to Menager
GRANT EXECUTE ON udfgetOrdersValueDaily to Menager
GRANT EXECUTE ON udfgetOrdersValueMonthly to Menager
GRANT EXECUTE ON udfgetOrdersValueYearly to Menager
GRANT EXECUTE ON udfMenuCorrect to Menager
GRANT SELECT ON udfXMostPopularTables to Menager
```

ADMINISTRATOR

Może wykonywać wszystkie operacje na bazie danych.

```
CREATE ROLE Admin
GRANT all privileges ON u_mskrok TO Admin
```