Autorzy:

Maciej Nessel

Andrzej Starzyk

Jakub Radek

Prowadzący:

dr inż. Robert Marcjan

DOKUMENTACJA BAZY DANYCH

Projekt: Restauracja

2021/2022

Projekt bazy danych systemu zarządzania działalnością firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.

1. Użytkownicy w systemie

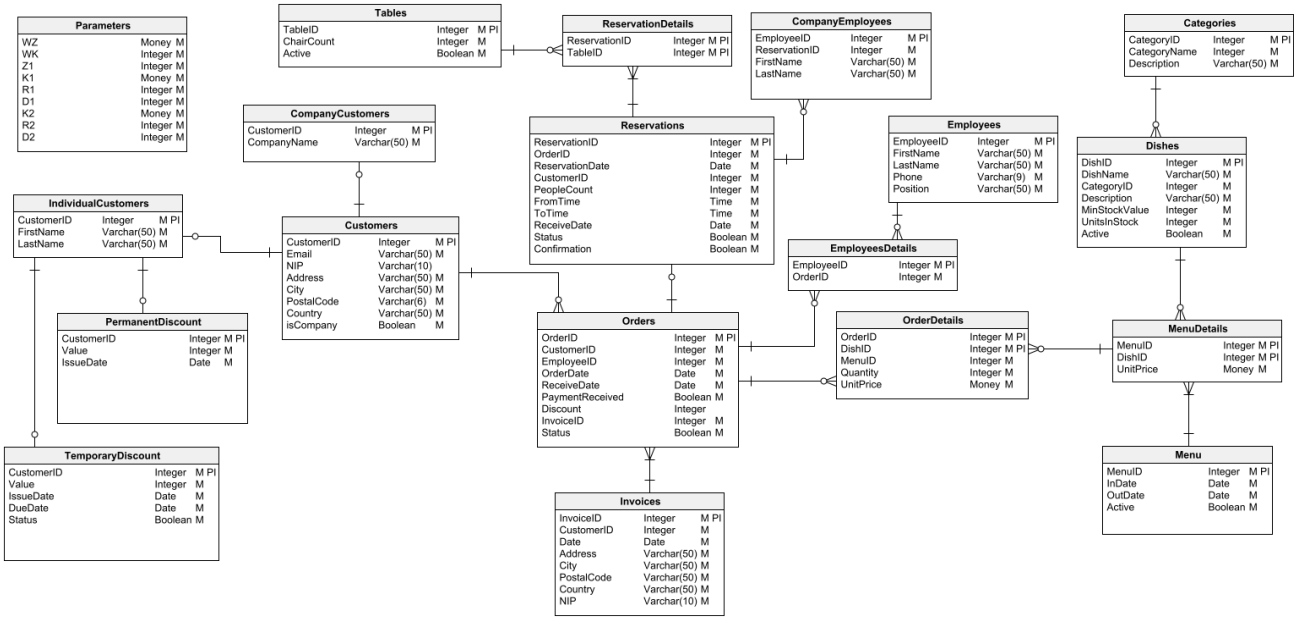
- Administrator systemu
- Klienci indywidualni
- Firmy
- Pracownicy
- Manager

2. Funkcje realizowane przez system

- Obsługa menu
    - Pracownicy i manager:
        - System pozwala na ustalenie menu na dowolny obszar czasu, dostęp do tej funkcji posiada manager.
        - Aby menu zostało aktywowane musi ono zaczynać obowiązywać w dzień następujący po ostatnim dniu na który istnieje ustalone menu.
        - Menu musi zostać zmienione o 50% co najmniej raz na dwa tygodnie, system informuje użytkownika kiedy menu spełnia warunki i może być aktywne.
        - System przechowuje informacje dotyczące historii menu na przestrzeni czasu oraz ceny potraw dostępnych w danym przedziale czasowym.
- Składanie zamówień:
    - Klienci indywidualni oraz firmy:
        - Mogą zamówić dowolną ilość dowolnego dania z menu, o ile jest ono dostępne.
        - Podczas składania zamówienia mają dostęp do potraw, które znajdą się w menu w dniu, w którym dany klient chce odebrać swoje zamówienie
        - Mogą składać zamówienia na miejscu lub przez formularz internetowy
        - Klienci indywidualni mogą zarezerwować stolik i miejsca przy nim, jeśli posiadają konto. Posiadanie konta wiąże się z podaniem adresu email.
        - Firmy mają możliwość złożenia zamówienia na firmę bądź indywidualnych pracowników danej firmy.
        - Klienci powinni otrzymać potwierdzenie zamówienia i ewentualnej rezerwacji, przy zamówieniu online należy utworzyć nowe konto bądź wykorzystać istniejące.
        - W przypadku zamawiania owoców morza, należy takie zamówienie złożyć w dniach czwartek-piątek-sobota. Odebrać je będzie można najwcześniej we wtorek w następnym tygodniu.
        - Jeśli klient indywidualny ma ważny rabat, to może go wykorzystać.

        - Podczas zamawiania na miejscu klient ma możliwość zarejestrowania się do systemu tak jak podczas transakcji online co pozwala mu generować rabaty, nie jest to natomiast konieczne.
- Generowanie faktur:
    - Klienci mogą zażądać wystawienie faktury na pojedyńcze zamówienie lub faktury zbiorczej za cały miesiąc.
- Generowanie raportów:
    - Pracownicy powinni móc otrzymać tygodniowe i miesięczne zestawienie dotyczące zarezerwowanych stolików, rabatów, statystyk zamówień wszystkich klientów.
    - Klienci powinni móc otrzymywać zestawienie dotyczące złożonych przez nich zamówień i przysługujących im rabatów.
- Funkcje administratora danych:
    - Edycja bazy danych, w tym dodawanie oraz usuwanie pracowników.

3. Schemat bazy danych:



Vertabelo

4. Tabele oraz warunki integralności:

1.Tabela **Invoices** - Faktury

```
CREATE TABLE Invoices (
    InvoiceID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    CustomerID int  NOT NULL,
    Date date  NOT NULL,
    Address varchar(50)  NOT NULL,
    City varchar(50)  NOT NULL,
    PostalCode varchar(50)  NOT NULL,
    Country varchar(50)  NOT NULL,
    NIP varchar(10)  NOT NULL
);
```

2.Tabela **Categories** - Kategorie dań

```
CREATE TABLE Categories (
    CategoryID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    CategoryName varchar(50)  NOT NULL UNIQUE,
    Description varchar(50)  NOT NULL
);
```

3.Tabela **Tables** - Stoły w restauracji

```
CREATE TABLE Tables (
    TableID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    ChairCount int  NOT NULL,
    Active bit  NOT NULL DEFAULT 1,
    CHECK(ChairCount > 0)
);
```

4.Tabela **Customers** - wspólne informacje dla klientów indywidualnych oraz firmowych

```
CREATE TABLE Customers (
    CustomerID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    Email varchar(50)  NOT NULL,
    NIP varchar(10)  NULL,
    Address varchar(50)  NOT NULL,
    City varchar(50)  NOT NULL,
    PostalCode varchar(6)  NOT NULL,
    Country varchar(50)  NOT NULL,
    isCompany bit  NOT NULL,
);
```

5.Tabela **Dishes** - Dania serwowane przez restaurację

```
CREATE TABLE Dishes (
   DishID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
   DishName varchar(50)  NOT NULL,
   CategoryID int  NOT NULL FOREIGN KEY REFERENCES Categories(CategoryID),
   Description varchar(50)  NOT NULL,
   MinStockValue int  NOT NULL,
   UnitsInStock int  NOT NULL,
   Active bit NOT NULL DEFAULT 1,
   CHECK(MinStockValue > 0),
   CHECK(UnitsInStock >=0)
);
```

6.Tabela **Employees** - informacje o pracownikach restauracji

```
CREATE TABLE Employees (
    EmployeeID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    FirstName varchar(50)  NOT NULL,
    LastName varchar(50)  NOT NULL,
    Phone varchar(9)  NOT NULL,
    Position varchar(50) NOT NULL
);
```

7.Tabela **Orders** - informacje o zamówieniach

```
CREATE TABLE Orders (
    OrderID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    CustomerID int  NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
    InvoiceID int  FOREIGN KEY REFERENCES Invoices(InvoiceID),
    EmployeeID int  NOT NULL,
    OrderDate date  NOT NULL,
    ReceiveDate date,
    PaymentReceived bit  NOT NULL,
    Discount float(2)  NULL,
    Status bit NOT NULL DEFAULT 1,
    CHECK(OrderDate <= ReceiveDate)
);
```

8.Tabela **Reservations** - informacje o rezerwacjach

```
CREATE TABLE Reservations (
    ReservationID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    OrderID int Unique NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
    ReservationDate date  NOT NULL,
    CustomerID int  NOT NULL,
    PeopleCount int  NOT NULL,
    FromTime time  NOT NULL,
    ToTime time  NOT NULL,
    ReceiveDate date  NOT NULL,
    Status bit NOT NULL DEFAULT 1,
    Confirmation bit NOT NULL DEFAULT 0
);
```

9.Tabela **CompanyEmployees** - informacje o pracownikach firmy, dla których zostaną wydane dania z rezerwacji firmowej

```
CREATE TABLE CompanyEmployees (
    CompanyEmployeeID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    ReservationID int  NOT NULL FOREIGN KEY REFERENCES
Reservations(ReservationID),
    FirstName varchar(50)  NOT NULL,
    LastName varchar(50)  NOT NULL
);
```

10.Tabela **EmployeesDetails** - pracownicy którzy są przypisani do danego zamówienia

```
CREATE TABLE EmployeesDetails (
    EmployeeID int  NOT NULL FOREIGN KEY REFERENCES
Employees(EmployeeID),
    OrderID int  NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
     CONSTRAINT PK_EmployeesDetails PRIMARY KEY(EmployeeID, OrderID)
);
```

11.Tabela **IndividualCustomers** - szczegółowe informacje klientów indywidualnych

```
CREATE TABLE IndividualCustomers (
    CustomerID int  NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES
Customers(CustomerID),
    FirstName varchar(50)  NOT NULL,
    LastName varchar(50)  NOT NULL
);
```

12.Tabela **Menu** - informacje o menu

```
CREATE TABLE Menu (
    MenuID int  NOT NULL PRIMARY KEY IDENTITY(1, 1),
    InDate date  NOT NULL,
    OutDate date  NOT NULL,
    Active bit NOT NULL
);
```

13.Tabela **MenuDetails** - informacje dotyczące dań, które wchodzą w skład danego menu

```
CREATE TABLE MenuDetails (
    MenuID int  NOT NULL,
    DishID int  NOT NULL,
    UnitPrice money  NOT NULL
    PRIMARY KEY CLUSTERED ( MenuID, DishID ),
    FOREIGN KEY ( MenuID ) REFERENCES [Menu] ( MenuID ) ON UPDATE  NO
ACTION  ON DELETE  CASCADE,
    FOREIGN KEY ( DishID ) REFERENCES [Dishes] ( DishID ) ON UPDATE  NO
ACTION  ON DELETE  CASCADE,
    CHECK(UnitPrice > 0)
);
```

14.Tabela **OrderDetails** - informacje dotyczące szczegółów zamówienia

```
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL FOREIGN KEY REFERENCES Orders(OrderID),
    DishID int  NOT NULL,
    MenuID int NOT NULL,
    Quantity int  NOT NULL,
    UnitPrice money  NOT NULL,
    CHECK(Quantity > 0),
    CHECK(UnitPrice > 0),
    PRIMARY KEY CLUSTERED ( OrderID, DishID ),
    FOREIGN KEY ( MenuID, DishID ) REFERENCES [MenuDetails] ( MenuID,
DishID) ON UPDATE  NO ACTION  ON DELETE  CASCADE,
);
```

15.Tabela **PermanentDiscount** - informacje o stałych zniżkach dla klientów

```
CREATE TABLE PermanentDiscount (
    CustomerID int  NOT NULL FOREIGN KEY REFERENCES
IndividualCustomers(CustomerID),
    Value int  NOT NULL,
    IssueDate date  NOT NULL,
    CHECK(IssueDate <= getdate()),
    CHECK(Value >= 0)
    CHECK(Value <= 100) );
```

16.Tabela **ReservationDetails** - informacje o stołach które są przypisane do zamówień

```
CREATE TABLE ReservationDetails (
    ReservationID int  NOT NULL FOREIGN KEY REFERENCES
Reservations(ReservationID),
    TableID int  NOT NULL FOREIGN KEY REFERENCES Tables(TableID),
    CONSTRAINT PK_ReservationDetail PRIMARY KEY(ReservationID, TableID)
);
```

17.Tabela **TemporaryDiscount** - informacje o tymczasowych zniżkach dla klientów

```
CREATE TABLE TemporaryDiscount (
    CustomerID int  NOT NULL FOREIGN KEY REFERENCES
IndividualCustomers(CustomerID),
    Value int  NOT NULL,
    IssueDate date  NOT NULL,
    DueDate date  NOT NULL,
    Status bit NOT NULL DEFAULT 1,
    CHECK(IssueDate <= getdate()),
    CHECK(DueDate >= getdate()),
    CHECK(Value > 0)
);
```

18.Tabela **CompanyCustomers** - szczegółowe informacje dotyczące klientów firmowych

```
CREATE TABLE CompanyCustomers (
    CustomerID int  NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
    CompanyName varchar(50)  NOT NULL UNIQUE
);
```

19.Tabela **Parameters** - parametry

```
CREATE TABLE Parameters (
WZ money NOT NULL,
WK int NOT NULL,
Z1 int NOT NULL,
K1 money NOT NULL,
R1 int NOT NULL,
D1 int NOT NULL,
K2 money NOT NULL,
R2 int NOT NULL );
```

5. Procedury

1. Dodawanie nowego klienta indywidualnego:

```
CREATE PROCEDURE AddNewIndividualCustomer
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
@NIP VARCHAR(10) = NULL,
@Email VARCHAR(50),
@Address VARCHAR(50),
@City VARCHAR(50),
@PostalCode VARCHAR(6),
@Country VARCHAR(50),
@isCompany BIT = 0
AS
BEGIN
BEGIN TRANSACTION
INSERT INTO Customers (Email, Address, City, NIP, PostalCode, Country, isCompany)
Values(@Email, @Address, @City, @NIP, @PostalCode, @Country, @isCompany);

IF @@ERROR <> 0
      BEGIN
            RAISERROR ('Not added!', 16, -1)
            ROLLBACK
      END

INSERT INTO IndividualCustomers(CustomerID, FirstName, LastName)
Values( @@IDENTITY, @FirstName, @LastName);

IF @@ERROR <> 0
      BEGIN
            RAISERROR ('Not added!', 16, -1)
            ROLLBACK
     END

COMMIT;
END
```

2. Dodawanie nowego klienta firmowego:

```
CREATE PROCEDURE AddNewCompanyCustomer
@CompanyName VARCHAR(50),
@NIP VARCHAR(10) = NULL,
@Email VARCHAR(50),
@Address VARCHAR(50),
@City VARCHAR(50),
@PostalCode VARCHAR(6),
@Country VARCHAR(50),
@isCompany BIT = 1
AS
BEGIN
BEGIN TRANSACTION
INSERT INTO Customers (Email, Address, City, NIP, PostalCode, Country,
isCompany) Values(@Email, @Address, @City, @NIP, @PostalCode, @Country,
@isCompany);
IF @@ERROR <> 0
      BEGIN
            RAISERROR ('Not added!', 16, -1)
            ROLLBACK
      END
INSERT INTO CompanyCustomers(CustomerID, CompanyName) Values( @@IDENTITY,
@CompanyName);
IF @@ERROR <> 0
      BEGIN
            RAISERROR ('Not added!', 16, -1)
            ROLLBACK
      END
COMMIT;
END
```

3. Dodawanie nowego pracownika:

```
CREATE PROCEDURE AddNewEmployee
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
@Phone VARCHAR(9),
@Position VARCHAR(50)
AS
BEGIN
INSERT INTO Employees(FirstName, LastName, Phone, Position) Values
(@FirstName, @LastName, @Phone, @Position);
END
```

4. Dodanie nowego dania:

```
CREATE PROCEDURE AddDish
@DishName Varchar(50),
@CategoryID INT,
@Description Varchar(50),
@MinStockValue INT,
@UnitsInStock INT,
@Active BIT = 1
AS
BEGIN
IF NOT EXISTS (SELECT CategoryID FROM Categories WHERE CategoryID = @CategoryID)
      THROW 50001, 'No such categoryID.', 1
INSERT INTO Dishes(DishName, CategoryID, Description, MinStockValue,
UnitsInStock) Values (@DishName, @CategoryID, @Description, @MinStockValue,
@UnitsInStock)
END
```

5. Dodanie nowej kategorii dań:

```
CREATE PROCEDURE AddCategory
@CategoryName Varchar(50),
@Description Varchar(50)
AS
BEGIN
INSERT INTO Categories(CategoryName, Description) Values (@CategoryName,
@Description)
END
```

6. Dodanie nowego menu:

```
CREATE PROCEDURE AddMenu
@InDate Date,
@OutDate Date
AS
BEGIN
INSERT INTO Menu(InDate, OutDate, Active) Values (@InDate, @OutDate, 0)
END
```

7. Dodanie dania do menu

```
CREATE PROCEDURE AddDishToMenu
@MenuID INT,
@DishID INT,
@UnitPrice Money
AS
BEGIN
IF NOT EXISTS (SELECT MenuID FROM Menu WHERE MenuID = @MenuID)
      THROW 50001, 'No such MenuID', 1
IF NOT EXISTS (SELECT DishID FROM Dishes WHERE DishID = @DishID)
      THROW 50001, 'No such DishID', 1
INSERT INTO MenuDetails(MenuID, DishID, UnitPrice) Values (@MenuID, @DishID,
@UnitPrice)
END
```

8. Dodanie nowego zamówienia:

```
CREATE PROCEDURE AddNewOrder
@CustomerID INT,
@EmployeeID INT,
@PaymentReceived BIT,
@OrderDate DATE,
@ReceiveDate DATE = NULL,
@Discount FLOAT(2) = NULL,
@InvoiceID INT = NULL
AS
BEGIN
INSERT INTO Orders(CustomerID, EmployeeID, OrderDate, ReceiveDate,
PaymentReceived, Discount, InvoiceID) Values (@CustomerID, @EmployeeID,
@OrderDate, @ReceiveDate, @PaymentReceived, @Discount, @InvoiceID)
END
```

9. Dodanie szczegółów zamówienia:

```
CREATE PROCEDURE AddNewOrderDetails
@OrderID INT,
@DishID INT,
@MenuID INT,
@Quantity INT,
@UnitPrice Money
AS
BEGIN
INSERT INTO OrderDetails(OrderID, DishID, MenuID, Quantity, UnitPrice)
Values (@OrderID, @DishID, @MenuID, @Quantity, @UnitPrice)
END
```

10. Dodanie nowego stolika:

```
CREATE PROCEDURE AddNewTable
@ChairCount INT,
@Active BIT
AS
BEGIN
INSERT INTO Tables (ChairCount, Active) Values(@ChairCount, @Active);
END
```

11. Dodanie rezerwacji:

```
CREATE PROCEDURE AddNewReservation
@OrderID INT,
@ReservationDate Date,
@CustomerID INT,
@PeopleCount INT,
@FromTime Time,
@ToTime Time,
@ReceiveDate Date
AS
BEGIN
IF NOT EXISTS (SELECT CustomerID FROM Customers WHERE CustomerID =
@CustomerID)
    THROW 50001, 'No such customer.', 1
INSERT INTO Reservations(OrderID, ReservationDate, CustomerID,
PeopleCount, FromTime, ToTime, ReceiveDate)
VALUES (@OrderID, @ReservationDate, @CustomerID, @PeopleCount, @FromTime,
@ToTime, @ReceiveDate)
END
```

12. Dodanie stolika do rezerwacji:

```
CREATE PROCEDURE AddTableToReservation
@ReservationID INT,
@TableID INT
AS
BEGIN
INSERT INTO ReservationDetails(ReservationID, TableID)
VALUES (@ReservationID, @TableID)
END
```

13. Dodanie nowej faktury

```
CREATE PROCEDURE AddMonthlyInvoice
@CustomerID INT,
@Month INT
AS
BEGIN
	BEGIN TRANSACTION
	IF @Month not BETWEEN 1 AND 12
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'Incorrect month', 1;
	END
	DECLARE @OrderID INT, @InvoiceID INT, @Today DATE = GETDATE(),
	@Address VARCHAR(50) = (SELECT Address FROM Customers
				WHERE CustomerID = @CustomerID),
	@City VARCHAR(50) = (SELECT City FROM Customers
				WHERE CustomerID = @CustomerID),
	@PostalCode VARCHAR(50) = (SELECT PostalCode FROM Customers
				WHERE CustomerID = @CustomerID),
	@Country VARCHAR(50) = (SELECT Country FROM Customers
				WHERE CustomerID = @CustomerID),
	@NIP VARCHAR(50) = (SELECT NIP FROM Customers
				WHERE CustomerID = @CustomerID);

	EXEC AddNewInvoice @CustomerID, @Today, @Address, @City, @PostalCode,
@Country, @NIP;
	UPDATE Orders
	SET InvoiceID = @@IDENTITY
	WHERE OrderID in (SELECT O.OrderID
				FROM Orders AS O
				WHERE Status = 1 AND PaymentReceived = 1
				AND MONTH(ReceiveDate) = @Month
				AND CustomerID = @CustomerID);
	COMMIT TRANSACTION

END
```

14. Dodanie stolików do rezerwacji w zależności od liczby osób i dostępnych stolików

```
CREATE PROCEDURE AddTablesToReservation
@ReservationID INT
AS
DECLARE @ReservationDate DATE = (SELECT ReceiveDate FROM Reservations WHERE
ReservationID = @ReservationID)
DECLARE @FromTime TIME = (SELECT FromTime FROM Reservations
                          WHERE ReservationID = @ReservationID)
DECLARE @ToTime TIME = (SELECT ToTime FROM Reservations
                        WHERE ReservationID = @ReservationID)
DECLARE @PeopleCount INT = (SELECT PeopleCount FROM Reservations
                            WHERE ReservationID = @ReservationID)
BEGIN
IF (SELECT SUM(ChairCount) FROM Tables WHERE TableID
IN (SELECT * FROM dbo.TablesAvaliable(@ReservationDate, @FromTime, @ToTime)))
< @PeopleCount
        THROW 50001, 'Not enough tables.', 1

DECLARE @ActualTable INT
SET @ActualTable =
      (SELECT TOP 1 TableID FROM TABLES
       WHERE  ChairCount >= @PeopleCount AND TableID IN
            (SELECT * FROM dbo.TablesAvaliable(@ReservationDate, @FromTime,
@ToTime))
       ORDER BY ChairCount)

IF(@ActualTable!=NULL)
     INSERT INTO ReservationDetails(ReservationID, TableID) VALUES
(@ReservationID,
      @ActualTable)
ELSE
    BEGIN
    DECLARE @Cnt INT = 0
    WHILE (@Cnt < @PeopleCount)
         BEGIN
          SET @ActualTable =
              (SELECT TOP 1 TableID FROM TABLES
               WHERE TableID IN (
              SELECT * FROM dbo.TablesAvaliable(@ReservationDate, @FromTime,
@ToTime))
```

```
            ORDER BY ChairCount DESC)
             INSERT INTO ReservationDetails(ReservationID, TableID)
                 VALUES (@ReservationID, @ActualTable)
          SET @Cnt = @Cnt + (SELECT ChairCount FROM Tables WHERE TableID =
@ActualTable)
          END
    END
END
```

15. Składanie zamówienia

Procedura weryfikuje najpierw poprawność podanych argumentów. Następnie dodaje nowe zamówienie, a do niego odpowiednie dania. Na końcu dodaje nową rezerwację i stoliki do niej, zniżki oraz klientów indywidualnych, dla których firma składa rezerwacje.

```
CREATE PROCEDURE PlaceOrder
@CustomerID INT,
@dishes VARCHAR(250),
@ReceiveDate DATE,
@PaymentReceived BIT,
@EmployeeID INT,
@PeopleCount INT = NULL,
@FromTime TIME = NULL,
@ToTime TIME = NULL,
@CompanyEmployees VARCHAR(250) = NULL
AS
BEGIN
	BEGIN TRANSACTION
	DECLARE @OrderID int, @Discount INT, @Today date = getdate(), @TotalPrice INT = 0;

	IF NOT EXISTS (SELECT CustomerID FROM Customers WHERE CustomerID = @CustomerID)
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'No such customer.', 1;
	END



	DECLARE @MenuID INT = (SELECT MenuID
				FROM Menu
				WHERE InDate <= @ReceiveDate
				AND OutDate >= @ReceiveDate
				AND Active = 1),
		@isCompany BIT = (SELECT isCompany FROM Customers
					WHERE CustomerID = @CustomerID);

	IF @MenuID is NULL
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'No such menu.', 1;
	END

	SET @Discount = dbo.CalculateDiscount(@CustomerID);

	SET DATEFIRST 1;
```

```
	DECLARE @Quantity int, @DishID int, @UnitPrice money, @FromCursor VARCHAR(250);

	DECLARE DishesOrdered CURSOR FOR SELECT * FROM STRING_SPLIT(@dishes, ',');

	OPEN DishesOrdered;
	FETCH NEXT FROM DishesOrdered INTO @FromCursor
	WHILE @@FETCH_STATUS = 0
		BEGIN
			SET @DishID = CAST(@FromCursor AS INT);
			IF @DishID NOT IN (SELECT DishID FROM MenuDetails WHERE MenuID =
@MenuID)
			BEGIN
				ROLLBACK TRANSACTION;
				THROW 50001, 'Dish not in menu.', 1;
			END


			IF @DishID in (SELECT DishID
						FROM Categories
						JOIN Dishes ON Categories.CategoryID =
Dishes.DishID
						WHERE CategoryName = 'Owoce morza')
			BEGIN
				IF not DATEPART(weekday, @Today) BETWEEN 4 and 6
				BEGIN
					ROLLBACK TRANSACTION;
					THROW 50001, 'Seafood cannot be ordered today.', 1
				END

				IF DATEPART(week, @ReceiveDate) = DATEPART(week, @Today) and
@ReceiveDate <= DATEADD(DAY, 8 - DATEPART(weekday, @TODAY), @Today)
				BEGIN
					ROLLBACK TRANSACTION;
					THROW 50001, 'Seafood cannot be ordered for this day.', 1
				END
			END
			FETCH NEXT FROM DishesOrdered INTO @FromCursor;
			IF @@FETCH_STATUS <> 0
			BEGIN
				ROLLBACK TRANSACTION;
				THROW 50001, 'Quantity has not been specified.', 1;
			END

			SET @Quantity = CAST(@FromCursor AS INT);
			SET @UnitPrice = (SELECT UnitPrice FROM MenuDetails WHERE DishID =
```

```
@DishID and MenuID = @MenuID);
                  SET @TotalPrice += @UnitPrice * @Quantity * (CONVERT(DECIMAL, 100 -
@Discount)/100)
                  FETCH NEXT FROM DishesOrdered INTO @FromCursor
            END
      CLOSE DishesOrdered;
      DEALLOCATE DishesOrdered;

      IF (@TotalPrice < (SELECT WZ FROM Parameters) or (SELECT COUNT(*) FROM Orders WHERE
CustomerID = @CustomerID) < (SELECT WK FROM Parameters)) and @PaymentReceived = 0
      BEGIN
            ROLLBACK TRANSACTION;
            THROW 50001, 'Cannot pay later.', 1;
      END

      EXEC AddNewOrder @CustomerID, @EmployeeID,  @PaymentReceived,  @Today,
@ReceiveDate,  @Discount
      SET @OrderID = @@IDENTITY;

      UPDATE TemporaryDiscount
      SET Status = 0
      WHERE CustomerID = @CustomerID and Status = 1;

      DECLARE DishesOrdered CURSOR FOR SELECT * FROM STRING_SPLIT(@dishes, ',');
      OPEN DishesOrdered;
      FETCH NEXT FROM DishesOrdered INTO @FromCursor
      WHILE @@FETCH_STATUS = 0
            BEGIN
                  SET @DishID = CAST(@FromCursor AS INT);
                  FETCH NEXT FROM DishesOrdered INTO @FromCursor;
                  SET @Quantity = CAST(@FromCursor AS INT);
                  SET @UnitPrice = (SELECT UnitPrice FROM MenuDetails WHERE DishID =
@DishID and MenuID = @MenuID);
                  EXEC AddNewOrderDetails @OrderID,  @DishID, @MenuID, @Quantity,
@UnitPrice
                  FETCH NEXT FROM DishesOrdered INTO @FromCursor
            END
      CLOSE DishesOrdered;
      DEALLOCATE DishesOrdered;

      DECLARE @ReservationID INT;
      IF @PeopleCount is not NULL and @FromTime is not NULL and @ToTime is not NULL
      BEGIN
            IF (SELECT SUM(ChairCount) FROM Tables
                  WHERE TableID IN (SELECT * FROM dbo.TablesAvaliable(@ReceiveDate,
@FromTime, @ToTime))) < @PeopleCount
```

```
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50001, 'No such tables.', 1;
            END
            EXEC AddNewReservation @OrderID, @Today, @CustomerID, @PeopleCount,
@FromTime, @ToTime, @ReceiveDate
            SET @ReservationID = @@Identity
            EXEC AddTablesToReservation @ReservationID
      END

      IF @isCompany = 1
            BEGIN
                  DECLARE @Name VARCHAR(50), @Surname VARCHAR(50);
                  DECLARE CompanyEmployees CURSOR FOR SELECT * FROM
string_split(@CompanyEmployees,',');
                  OPEN CompanyEmployees;
                  FETCH NEXT FROM CompanyEmployees INTO @Name
                  WHILE @@FETCH_STATUS = 0
                  BEGIN
                        FETCH NEXT FROM CompanyEmployees INTO @Surname
                        IF @@FETCH_STATUS <> 0
                        BEGIN
                              ROLLBACK TRANSACTION;
                              THROW 50001, 'Surname has not been specified', 1;
                        END
                        INSERT INTO CompanyEmployees VALUES(@ReservationID, @Name,
@Surname)
                        FETCH NEXT FROM CompanyEmployees INTO @Name
                  END
                  CLOSE CompanyEmployees;
                  DEALLOCATE CompanyEmployees;
            END
      ELSE
            BEGIN
                  DECLARE @MinOrdersForPerm INT = (SELECT Z1 FROM Parameters),
                              @MinValueForPerm MONEY = (SELECT K1 FROM Parameters),
                              @MinValueForTemp MONEY = (SELECT K2 FROM Parameters),
                              @Duration INT = (SELECT D1 FROM Parameters),
                              @PermValue INT = (SELECT R1 FROM Parameters),
                              @TempValue INT = (SELECT R2 FROM Parameters);

                  DECLARE @CustomerOrders TABLE(CustomerID INT, Total MONEY);
                  INSERT INTO @CustomerOrders
                        SELECT CustomerID,
SUM(OD.Quantity*OD.UnitPrice*((convert(decimal, 100-O.Discount))/100))
                        FROM Orders O
```

```
                        INNER JOIN OrderDetails as OD on O.OrderID = OD.OrderID
                        WHERE O.Status = 1 and CustomerID = @CustomerID
                        GROUP BY O.OrderID, O.CustomerID

                    IF @CustomerID not in (SELECT CustomerID FROM PermanentDiscount) and
                       @MinOrdersForPerm <= (SELECT COUNT(*)
                                                FROM @CustomerOrders
                                                WHERE Total > @MinValueForPerm
                                                GROUP BY CustomerID)
                        BEGIN
                              INSERT INTO PermanentDiscount(CustomerID, Value, IssueDate)
VALUES(@CustomerID, @PermValue, @Today)
                        END

                    IF @TotalPrice > @MinValueForTemp
                        BEGIN
                              DECLARE @Date DATE = DATEADD(DAY, 7, @Today)
                              INSERT INTO TemporaryDiscount(CustomerID, Value, IssueDate,
DueDate) VALUES(@CustomerID, @TempValue, @Today, @Date)
                        END
            END
      COMMIT TRANSACTION
END
```

16. Aktywacja menu

```
CREATE PROCEDURE ActivateMenu
@MenuID INT
AS
BEGIN
	BEGIN TRANSACTION;
	IF (SELECT Active FROM MENU WHERE MenuID = @MenuID) = 1
	BEGIN
		ROLLBACK TRANSACTION;
		RETURN;
	END
	IF not EXISTS(SELECT * FROM Menu WHERE Active = 1)
	BEGIN
		UPDATE Menu
		SET Active = 1
		WHERE MenuID = @MenuID;
		COMMIT TRANSACTION;
		RETURN;
	END

	DECLARE @Current INT, @Previous INT,
			@MenuInDate DATE = (SELECT InDate From Menu Where MenuID =
@MenuID),
			@LastActiveOutDate DATE = (SELECT TOP 1 OutDate FROM Menu
WHERE Active = 1 ORDER BY OutDate DESC);
	IF DATEADD(DAY, -1, @MenuInDate) > @LastActiveOutDate
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'Missing menu.', 1;
	END
	IF DATEADD(DAY, -1, @MenuInDate) < @LastActiveOutDate
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'There is already an active menu.', 1;
	END
	SET @Current = @MenuID;
	DECLARE PreviousMenu CURSOR FOR
		SELECT MenuID
		FROM Menu
		WHERE Active = 1 and OutDate > DATEADD(DAY, -14, @MenuInDate)
		ORDER BY InDate DESC
	OPEN PreviousMenu
	FETCH NEXT FROM PreviousMenu INTO @Previous;
	WHILE @@FETCH_STATUS = 0
	BEGIN
```

```
            DECLARE @OldNOfDishes INT = (SELECT COUNT(*) FROM MenuDetails WHERE
MenuID = @Previous),
                        @NewNOfDishes INT = (SELECT COUNT(*) FROM MenuDetails
WHERE MenuID = @Current),
                        @Removed INT =
                        (SELECT COUNT(*)
                         FROM (SELECT DishID
                                 FROM MenuDetails
                                 WHERE MenuID = @Previous
                                 EXCEPT
                                 SELECT DishID
                                 FROM MenuDetails
                                 WHERE MenuID = @Current) AS diff),
                        @Added INT =
                        (SELECT COUNT(*)
                         FROM (SELECT DishID
                                 FROM MenuDetails
                                 WHERE MenuID = @Current
                                 EXCEPT
                                 SELECT DishID
                                 FROM MenuDetails
                                 WHERE MenuID = @Previous) AS diff);

            IF (@Removed >= @OldNOfDishes / 2) and (@Added >= @OldNOfDishes / 2)
            BEGIN
                  UPDATE Menu
                  SET Active = 1
                  WHERE MenuID = @MenuID;
                  CLOSE PreviousMenu;
                  DEALLOCATE PreviousMenu;
                  COMMIT TRANSACTION;
                  RETURN;
            END
            SET @Current = @Previous;
            FETCH NEXT FROM PreviousMenu INTO @Previous;
      END
      CLOSE PreviousMenu;
      DEALLOCATE PreviousMenu;
      PRINT 'Menu has not been activated';
      COMMIT TRANSACTION;
END
```

6. Funkcje

1. Zwraca stoliki, które można zarezerwować w podanym czasie

```
CREATE FUNCTION TablesAvaliable (@ReservationDate DATE, @FromTime TIME,
@ToTime TIME)
RETURNS TABLE AS
RETURN (SELECT TableID FROM Tables
    WHERE Active=1 AND TableID NOT IN (
          SELECT RD.TableID
          FROM ReservationDetails RD
          INNER JOIN Reservations R ON R.ReservationID = RD.ReservationID
         WHERE (R.FromTime>=@FromTime AND R.FromTime<=@ToTime)
         OR (R.ToTime>=@FromTime AND R.ToTime<=@ToTime)
         OR ((R.FromTime<=@FromTime AND R.ToTime>=@ToTime))))
```

2. Oblicza zniżkę dla danego klienta

```
CREATE FUNCTION CalculateDiscount(@CustomerID int)
RETURNS INT AS
BEGIN
DECLARE @DISCOUNT INT  = 0;

SET @Discount = 0;

IF @CustomerID in (SELECT CustomerID FROM PermanentDiscount)
    SET @Discount += (SELECT Value
                      FROM PermanentDiscount
                      WHERE CustomerID = @CustomerID)

IF @CustomerID in (SELECT CustomerID FROM TemporaryDiscount WHERE DueDate
>= getdate())
   SET @Discount += (SELECT Value
                     FROM TemporaryDiscount
                     WHERE CustomerID = @CustomerID
                     AND DueDate >= getdate()
                     AND STATUS = 1)
RETURN @Discount;
```

7. Widoki

1. **CurrentMenu** - aktualne menu

```
Create view CurrentMenu as
select D.DishName, D.Description, MD.UnitPrice
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where format(getdate(), 'yyyy-MM-dd') >= M.InDate and getdate() <=
M.OutDate and M.Active = 1
```

2. **MenuWeekAgo** - Menu tydzień temu

```
Create view MenuWeekAgo as
select D.DishName, D.Description, MD.UnitPrice
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where dateadd(day,-7, getdate()) >= M.InDate and dateadd(day,-7,
getdate()) <= M.OutDate and M.Active = 1
```

3. **MenuYearAgo** - Menu rok temu

```
Create view MenuYearAgo as
select D.DishName, D.Description, MD.UnitPrice
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where dateadd(year,-1, getdate()) >= M.InDate and dateadd(year,-1,
getdate()) <= M.OutDate and M.Active = 1
```

4. **DishesToOrder** - Dania których jest za mało i trzeba domówić

```
Create view DishesToOrder as
select D.DishName, D.Description, MD.UnitPrice
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where D.MinStockValue > D.UnitsInStock and M.InDate <= getdate() and
M.OutDate >= getdate()
```

5. **DishesCategories** - Wszystkie dania wraz z kategoriami

```
Create view DishesCategories as
select D.DishName, C.CategoryName, C.Description
from Dishes as D
inner join Categories as C on D.CategoryID = C.CategoryID
```

6. **DishesPricesHistory** - Wszystkie dania wraz z historią ich cen i występowania w menu na przestrzeni czasu

```
Create view DishesPricesHistory as
select D.DishName, MD.UnitPrice, M.InDate, M.OutDate
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
```

7. **DishesPricesHistoryThisYear** - Wszystkie dania wraz z historią ich cen i występowania w menu na przestrzeni aktualnego roku

```
Create view DishesPricesHistoryThisYear as
select D.DishName, MD.UnitPrice, M.InDate, M.OutDate
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where datediff(year , M.InDate, getdate()) = 0 and datediff(year ,
M.OutDate, getdate()) = 0
```

8. **DishesPricesHistoryLastYear** - Wszystkie dania wraz z historią ich cen i występowania w menu na przestrzeni ostatniego roku

```
Create view DishesPricesHistoryLastYear as
select D.DishName, MD.UnitPrice, M.InDate, M.OutDate
from Menu as M
inner join MenuDetails as MD on MD.MenuID = M.MenuID
inner join Dishes as D on D.DishID = MD.DishID
where datediff(year , M.InDate, getdate()) = 1 and datediff(year ,
M.OutDate, getdate()) = 1
```

9. **OrdersToDo** - Zamówienia do wykonania

```
Create view OrdersToDo as
select O.OrderID, isnull(concat(IC.Firstname, ' ', IC.Lastname),
CC.CompanyName) as 'Customer', O.OrderDate, O.ReceiveDate
from Orders as O
inner join Customers as C on O.CustomerID = C.CustomerID
inner join CompanyCustomers as CC on C.CustomerID = CC.CustomerID
inner join IndividualCustomers as IC on C.CustomerID = IC.CustomerID
where ReceiveDate > getdate() and status = 1
```

10. **EmployeeAssingments** - Przypisanie pracowników do zamówień i daty przypisań

```
Create view EmployeeAssingments as
select concat(E.Firstname, ' ', E.Lastname) as 'Employee', O.OrderID,
O.ReceiveDate
from Orders as O
inner join EmployeesDetails as ED on O.OrderID = ED.OrderID
inner join Employees as E on ED.EmployeeID = E.EmployeeID
where ReceiveDate > getdate() and status = 1
```

11. **TodaysOrdersValue** - Przychód z zamówień z dzisiaj na każdego klienta

```
Create view TodaysOrdersValue as
select O.OrderID, SUM(OD.UnitPrice*OD.Quantity) as 'Value', O.CustomerID
from Orders as O
inner join OrderDetails as OD on O.OrderID = OD.OrderID
where format(getdate(), 'yyyy-MM-dd') = O.ReceiveDate
group by O.OrderID, O.CustomerID
```

12. **DishesPopularity** - Popularność dań oraz ich przychód na przestrzeni czasu

```
Create view DishesPopularity as
select D.DishID, D.DishName, isnull(SUM(OD.Quantity), 0) as 'Quantity'
from OrderDetails as OD
inner join MenuDetails as MD on OD.DishID = MD.DishID
right join Dishes as D on D.DishID = MD.DishID
group by OD.DishID, D.DishID, D.DishName
```

13. **DishesPopularityThisYear** - Popularność dań oraz ich przychód w tym roku

```
Create view DishesPopularityThisYear as
select D.DishID, D.DishName, isnull(SUM(OD.Quantity), 0) as 'Quantity',
isnull(SUM(OD.Quantity*OD.UnitPrice),0) as 'Income'
from OrderDetails as OD
inner join MenuDetails as MD on OD.DishID = MD.DishID
right join Dishes as D on D.DishID = MD.DishID
inner join Orders as O on O.OrderID = OD.OrderID
where datediff(year , O.ReceiveDate, getdate()) = 0 and O.status = 1
group by OD.DishID, D.DishID, D.DishName
```

14. **DishesPopularityLastYear** - Popularność dań oraz ich przychód w ostatnim roku

```
Create view DishesPopularityLastYear as
select D.DishID, D.DishName, isnull(SUM(OD.Quantity), 0) as 'Quantity',
isnull(SUM(OD.Quantity*OD.UnitPrice),0) as 'Income'
from OrderDetails as OD
inner join MenuDetails as MD on OD.DishID = MD.DishID
right join Dishes as D on D.DishID = MD.DishID
inner join Orders as O on O.OrderID = OD.OrderID
where datediff(year , O.ReceiveDate, getdate()) = 1 and O.status = 1
group by OD.DishID, D.DishID, D.DishName
```

15. **NotConfirmedReservations** - Rezerwacje bez potwierdzenia

```
Create view NotConfirmedReservations as
select *
from Reservations as R
where getdate() <= R.ReceiveDate and Confirmation = 0
```

16. **ReservationsToday** - Rezerwacje na dzisiaj

```
Create view ReservationsToday as
select *
from Reservations as R
where format(getdate(), 'yyyy-MM-dd') = R.ReceiveDate and Confirmation =
1
```

17. **ReservationsThisWeek** - Rezerwacje na tydzień do przodu

```
Create view ReservationsThisWeek as
select *
from Reservations as R
where dateadd(day, 7, getdate()) >= R.ReceiveDate and R.ReceiveDate >=
getdate() and Confirmation = 1
```

18. **ReservationsThisMonth** - Rezerwacje na miesiąc do przodu

```
Create view ReservationsThisMonth as
select *
from Reservations as R
where dateadd(month, 1, getdate()) >= R.ReceiveDate and R.ReceiveDate >=
getdate() and Confirmation = 1
```

19. **ReservedTables** - Stoliki zarezerwowane na przyszłość

```
Create view ReservedTables as
select T.TableID, T.ChairCount, R.ReceiveDate, R.FromTime, R.ToTime
from Tables as T
inner join ReservationDetails as RD on RD.TableID = T.TableID
inner join Reservations as R on R.ReservationID = RD.ReservationID
where getdate() <= R.ReceiveDate and Status = 1 and Confirmation = 1
```

20. **ReservedTablesLastWeek** - Stoliki zarezerwowane w ciągu ostatniego tygodnia

```
Create view ReservedTablesLastWeek as
select T.TableID, T.ChairCount, R.ReceiveDate, R.FromTime, R.ToTime
from Tables as T
inner join ReservationDetails as RD on RD.TableID = T.TableID
inner join Reservations as R on R.ReservationID = RD.ReservationID
where dateadd(day,-7, getdate()) >= R.ReceiveDate and Status = 1 and
Confirmation = 1
```

21. **ReservedTablesLastMonth** - Stoliki zarezerwowane w ciągu ostatniego miesiąca

```
Create view ReservedTablesLastMonth as
select T.TableID, T.ChairCount, R.ReceiveDate, R.FromTime, R.ToTime
from Tables as T
inner join ReservationDetails as RD on RD.TableID = T.TableID
inner join Reservations as R on R.ReservationID = RD.ReservationID
where dateadd(month,-1, getdate()) >= R.ReceiveDate and Status = 1 and
Confirmation = 1
```

22. **Discounts** - Przyznane rabaty

```
Create view Discounts as
select CustomerID, Value, IssueDate
from TemporaryDiscount
union
select CustomerID, Value, IssueDate
from PermanentDiscount
```

23. **DiscountsLastWeek** - Przyznane rabaty w ciągu ostatniego tygodnia

```
Create view DiscountsLastWeek as
select CustomerID, Value, IssueDate
from TemporaryDiscount
where dateadd(day,-7, getdate()) >= IssueDate
union
select CustomerID, Value, IssueDate
from PermanentDiscount
where dateadd(day,-7, getdate()) >= IssueDate
```

24. **DiscountsLastMonth** - Przyznane rabaty w ciągu ostatniego miesiąca

```
Create view DiscountsLastMonth as
select CustomerID, Value, IssueDate
from TemporaryDiscount
where dateadd(month,-1, getdate()) >= IssueDate
union
select CustomerID, Value, IssueDate
from PermanentDiscount
where dateadd(month,-1, getdate()) >= IssueDate
```

25. **WhoOrderedWhat** - Kiedy klienci złożyli zamówienia i za jaką cenę

```
Create view WhoOrderedWhat as
select O.CustomerID, isnull(concat(IC.Firstname, ' ', IC.Lastname),
CC.CompanyName) as 'Customer', OD.Quantity*((convert(decimal,
100-O.Discount))/100) as 'Price'
from Orders O
inner join OrderDetails as OD on O.OrderID = OD.OrderID
inner join Customers as C on O.CustomerID = C.CustomerID
inner join CompanyCustomers as CC on C.CustomerID = CC.CustomerID
inner join IndividualCustomers as IC on C.CustomerID = IC.CustomerID
where O.Status = 1
```

26. **WhoOrderedWhatLastWeek** - Kiedy klienci złożyli zamówienia i za jaką cenę w ciągu ostatniego tygodnia

```
Create view WhoOrderedWhatLastWeek as
select O.CustomerID, isnull(concat(IC.Firstname, ' ', IC.Lastname),
CC.CompanyName) as 'Customer',
OD.Quantity*OD.UnitPrice*((convert(decimal, 100-O.Discount))/100) as
'Price'
from Orders O
inner join OrderDetails as OD on O.OrderID = OD.OrderID
inner join Customers as C on O.CustomerID = C.CustomerID
inner join CompanyCustomers as CC on C.CustomerID = CC.CustomerID
inner join IndividualCustomers as IC on C.CustomerID = IC.CustomerID
where dateadd(day,-7, getdate()) >= O.OrderDate and O.Status = 1
```

27. **WhoOrderedWhatLastMonth** - Kiedy klienci złożyli zamówienia i za jaką cenę w ciągu ostatniego miesiąca

```
Create view WhoOrderedWhatLastMonth as
select O.CustomerID, isnull(concat(IC.Firstname, ' ', IC.Lastname),
CC.CompanyName) as 'Customer',
OD.Quantity*OD.UnitPrice*((convert(decimal, 100-O.Discount))/100) as
'Price'
from Orders O
inner join OrderDetails as OD on O.OrderID = OD.OrderID
inner join Customers as C on O.CustomerID = C.CustomerID
inner join CompanyCustomers as CC on C.CustomerID = CC.CustomerID
inner join IndividualCustomers as IC on C.CustomerID = IC.CustomerID
where dateadd(month,-1, getdate()) >= O.OrderDate and O.Status = 1
```

28. **WhatDishesToOrder** - Które potrawy należy domówić na następny dzień ponad ich minimalną ilość i ile

```
Create view WhatDishesToOrder as
select D.DishID, D.DishName, sum(OD.Quantity) - D.MinStockValue as "Amount"
from Dishes D
inner join MenuDetails as MD on D.DishID = MD.DishID
inner join OrderDetails as OD on MD.DishID = OD.DishID
inner join Orders as O on OD.OrderID = O.OrderID
inner join Reservations as R on O.OrderID = R.OrderID
where O.Status = 1 and R.Status = 1 and R.Confirmation = 1 and datediff(day,
R.ReceiveDate, getdate()) = -1
group by D.DishID, D.DishName, D.MinStockValue
```