Podstawy Baz Danych
Dokumentacja do projektu “Restauracje”

Autorzy:

Piotr Sobczyński

Witold Strzeboński

Andrzej Karciński

Spis treści:

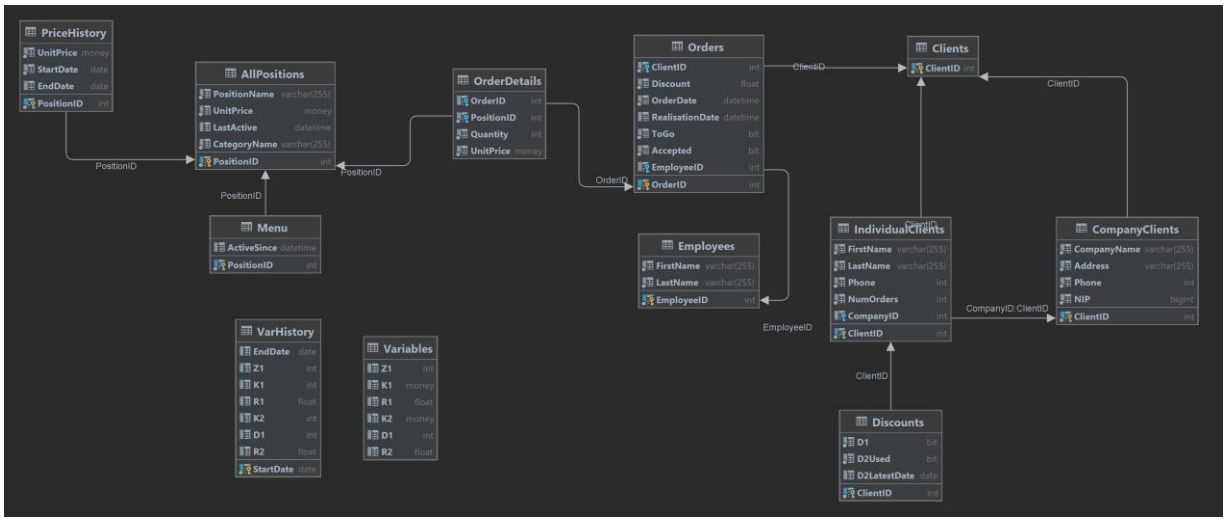
1. **Wprowadzenie**
2. **Diagram bazy danych**
3. **Tabele**
    - 3.1. AllPositions
    - 3.2. PriceHistory
    - 3.3. Menu
    - 3.4. Orders
    - 3.5. OrderDetails
    - 3.6. Employees
    - 3.7. Clients
    - 3.8. IndividualClients
    - 3.9. CompanyClients
    - 3.10. Discounts
    - 3.11. Variables
    - 3.12. VarHistory
4. **Widoki**
    - 4.1. CompanyEmployees
    - 4.2. CurrentMonthIncome
    - 4.3. CurrentMonthSales
    - 4.4. CurrentSeaFood
    - 4.5. MenuInfo
    - 4.6. MonthlyIncome
    - 4.7. MonthlyIncomeReportForEmployee
    - 4.8. MonthlyOperatedReportForEmployee
    - 4.9. NotAccepted
    - 4.10. PreviousMonthIncome
    - 4.11. PreviousMonthSales
    - 4.12. TotalReceipt
    - 4.13. OrdersByHour
    - 4.14. WeekDayLastMonth
5. **Procedury**
    - 5.1. AcceptOrder
    - 5.2. AddCompanyClient
    - 5.3. AddEmployee
    - 5.4. AddIndividualClient
    - 5.5. AddOrder
    - 5.6. AddOrderDetails
    - 5.7. AddPosition
    - 5.8. AddPositionToMenu
    - 5.9. ChangePrice
    - 5.10. ChangeVariables
6. **Funkcje**
    - 6.1. ClientDiscounts
    - 6.2. ClientOrdersHistory
    - 6.3. ClientsFromCompany
    - 6.4. EmployeeCTOperateMonth
    - 6.5. EmployeeOperateMonth
    - 6.6. MonthlyEmployeeCTOperate
    - 6.7. MonthlyEmployeeOperate

6.8. MonthlyIncomeAndAmount
6.9. OrdersOperatedByEmployee
6.10. OrdersOrderedBetween
6.11. OrdersRealizedBetween
6.12. CanUseOneTimeDiscount
6.13. IsEntitledToOneTimeDiscount
6.14. IsEntitledToReusableDiscount
6.15. PriceOfOrder
6.16. SumClientReceipt
6.17. OrdersInDayLastMonth
6.18. OrdersLastMonth
7. Triggery
7.1. UpdateMenu
7.2. AddCompanyClientToClients
7.3. AddIndividualClientToClients
8. Generowanie danych

1. Wprowadzenie

Celem projektu było stworzenie bazy danych dla restauracji. Klienci mogli składać zamówienia na miejscu bądź poprzez formularz internetowy z zamówieniem realizowanym w lokalu bądź zapakowanym do odbioru na wynos, wraz z możliwością zamówienia dań z owoców morza po wcześniejszym zgłoszeniu takiej chęci. Jest również możliwość zamówienia cateringu przez zewnętrzną firmę. Ponad to musieliśmy zaprojektować system naliczania rabatów dla stałych klientów a także rotację menu w taki sposób, aby każda pozycja zmieniała się co najmniej raz na dwa tygodnie.

2. Diagram bazy danych



3. Tabele

3.1.AllPositions

Przechowuje dane o wszystkich daniach, z których wybieramy menu na dany tydzień

PositionID – przechowuje numer dania (indeks)

PositionName – zawiera nazwę potrawy

UnitPrice – cena jednostkowa za danie

LastActive – kiedy pozycja ostatni raz była w menu

CategoryName **–** nazwa kategorii, z której pochodzi dana potrawa (są dostępne 4 kategorie: Danie glowne, Zupa, Deser, Owoce morza, Napoj)

3.2.PriceHistory

Przechowuje historię cen potraw

PositionID – indeks dania

UnitPrice – cena na jaką zmieniła się cena potrawy

StartDate – data wejścia w życie podanej zmiany

EndDate – data wraz z którą cena przestaje obowiązywać

3.3.Menu

Przechowuje aktualne menu

PositionID – indeks potrawy

ActiveSince – data od kiedy pozycja jest w menu

3.4.Orders

Przechowuje wszystkie zamówienia bez uwzględnienia tego co zostało naliczone w danym zamówieniu

OrderID – indeks zamówienia

ClientID – indeks klienta

Discount – naliczona zniżka (wartość między 0.0 a 1.0)

OrderDate – data zamówienia

RealisationDate – data realizacji

ToGo – czy zamówienie będzie realizowane na wynos

Accepted – czy zamówienie zostało zaakceptowane

Employee - jeżeli zostało zaakceptowane to przez którego pracownika

3.5.OrderDetails

Zawiera Szczegóły dotyczące każdego z zamówień

OrderID – indeks zamówienia

PositionID – indeks dania

Quantity – liczba zamówionej potrawy

UnitPrice – cena jednostkowa za porcję

3.6.Employees

Zawiera informacje dotyczące pracowników restauracji

EmployeeID – indeks pracownika

FirstName - imię pracownika

LastName – nazwisko pracownika

3.7. Clients

Tabela porządkująca dla obu rodzajów klientów

ClientID – indeks klienta

3.8. IndividualClients

Tabela zawierająca dane dotyczące klientów indywidualnych

ClientID – indeks klienta

FirstName - imię klienta

LastName – nazwisko klienta

Phone – numer telefonu (format dziewięciocyfrowy)

NumOrders – liczba zamówień dokonanych przez klienta

CompanyID - jeżeli jest pracownikiem firmy to pole zawiera indeks firmy

3.9. CompanyClients

Przechowuje dane dotyczące firm

ClientID – indeks klienta (indeks firmy)

CompanyName – nazwa firmy

Address - adres

Phone – numer telefonu (format dziewięciocyfrowy)

NIP – NIP firmy (format dziesięciocyfrowy)

3.10. Discounts

Tabela przechowująca zniżki dla klientów indywidualnych

ClientID – indeks klienta

D1 – czy klient posiada zniżkę typu D1

D2Used – czy klient posiada zniżkę typu D2

D2LatestDate – data ostatniego użycia zniżki D2

3.11. Variables

Tabela przechowująca aktualne hiperparametry

Z1 – liczba zamówień

K1 – kwota zamówienia

R1 - zniżka (wartość od 0.0 do 1.0)

K2 - łączna kwota

D1 – liczba dni

R2 - zniżka (wartość od 0.0 do 1.0)

3.12. VarHistory

Przechowuje dane historyczne hiperparametrów

StartDate – początkowa data obowiązywania hiperparametrów

EndDate – data końca obowiązywania hiperparametrów

Z1 – liczba zamówień

K1 – kwota zamówienia

R1 - zniżka (wartość od 0.0 do 1.0)

K2 - łączna kwota

D1 – liczba dni

R2 - zniżka (wartość od 0.0 do 1.0)

4. Widoki

4.1. CompanyEmployees

Widok pokazuje wszystkich klientów indywidualnych, którzy pracują w współpracującej firmie.

4.2.CurrentMonthIncome

Wyświetla przychód z bieżącego miesiąca.

4.3.CurrentMonthSales

Widok pokazuje wszystkie zamówienia wraz ze szczegółami z bieżącego miesiąca.

```
CREATE VIEW [dbo].[CurrentMonthSales] AS
SELECT O.OrderID, PositionID, Quantity, UnitPrice, OrderDate FROM OrderDetails
JOIN Orders O on O.OrderID = OrderDetails.OrderID
WHERE MONTH(OrderDate) = MONTH(GETDATE()) AND YEAR(OrderDate) = YEAR(GETDATE())
```

4.4.CurrentSeaFood

Widok pokazuje aktualnie dostępne w menu dania z owoców morza.

4.5.MenuInfo

Wyświetla menu wraz z informacjami szczegółowymi.

4.6.MonthlyIncome

Widok ukazuje wartość zamówień oraz ich ilość z podziałem na miesiące w ostatnich 12 miesięcy.

4.7.MonthlyIncomeReportForEmployee

Widok pokazuje sumę obsłużonych zamówień przez poszczególnych pracowników z podziałem na miesiące w ostatnich 12 miesiącach.

4.8.MonthlyOperatedReportForEmployee

Widok pokazuje liczbę obsłużonych zamówień przez poszczególnych pracowników z podziałem na miesiące w ostatnich 12 miesiącach.

4.9.NotAccepted

Widok wyświetla zamówienia, które nie zostały zaakceptowane.

4.10. PreviousMonthIncome

Widok pokazuje przychód z zeszłego miesiąca.

4.11. PreviousMonthSales

Widok pokazuje wszystkie zamówienia wraz ze szczegółami z poprzedniego miesiąca.

```
CREATE VIEW [dbo].[PreviousMonthSales] AS
    SELECT O.OrderID, PositionID, Quantity, UnitPrice, OrderDate FROM OrderDetails
    JOIN Orders O on O.OrderID = OrderDetails.OrderID
    WHERE MONTH(DATEADD(month, 1, OrderDate)) = MONTH(GETDATE()) AND
    YEAR(DATEADD(month, 1, OrderDate)) = YEAR(GETDATE())
```

4.12. TotalReceipt

Wyświetla wszystkie zamówienia z wyszczególnieniem ceny końcowej przed rabatem, z nałożonym rabatem a także wartość rabatu.

```
CREATE VIEW [dbo].[TotalReceipt] AS
SELECT O.OrderID,
    TMP.WithoutDiscount,
    ROUND(TMP.WithoutDiscount * (1-O.Discount), 2) as WithDiscount,
    O.Discount
    FROM Orders O
    JOIN
    (SELECT OrderID,
        SUM(Quantity * UnitPrice) as WithoutDiscount
        FROM OrderDetails
        GROUP BY OrderID) TMP ON O.OrderID = TMP.OrderID
```

4.13. OrdersByHour

Wyświetla liczbę i całkowity koszt zamówień z podziałem na godziny.

```
CREATE VIEW [dbo].[OrdersByHour] AS
    SELECT DATEPART(hour, OrderDate) AS Hour, COUNT(OrderID) AS NumberOfOrders,
    ROUND(SUM(dbo.PriceOfOrder(OrderID)), 2) AS TotalCost
    FROM Orders
    GROUP BY DATEPART(hour, OrderDate)
```

4.14. WeekDayLastMonth

Wyświetla liczbę zamówień dokonanych w ostatnim miesiącu z podziałem na dni tygodnia.

5. Procedury

5.1.AcceptOrder

AcceptOrder (@OrderID INT)

Zmienia status zamówienia na zaakceptowany.

5.2. AddCompanyClient

AddCompanyClient (@CompanyName VARCHAR(255), @Address VARCHAR (255), @Phone INT, @NIP BIGINT)

Dodaje firmę do bazy.

5.3.AddEmployee

AddEmployee (@FirstName VARCHAR (255), @LastName VARCHAR(255))

Dodaje pracownika do bazy.

```
CREATE PROCEDURE [dbo].[AddEmployee]
    @FirstName varchar(255),
    @LastName varchar(255)
AS
    SET NOCOUNT ON
    DECLARE @EmployeeID int
    SELECT @EmployeeID = MAX(EmployeeID) + 1 FROM Employees
    INSERT INTO Employees(EmployeeID, FirstName, LastName)
    VALUES (@EmployeeID, @FirstName, @LastName)
```

5.4.AddIndividualClient

AddIndividualClient (@FirstName VARCHAR (255), @LastName VARCHAR(255), @Phone INT, @CompanyID INT)

Dodaje klienta indywidualnego do bazy.

```
CREATE PROCEDURE [dbo].[AddIndividualClient]
    @FirstName varchar(255),
    @LastName varchar(255),
    @Phone int,
    @CompanyID int
AS
    SET NOCOUNT ON
    IF @CompanyID IS NOT NULL AND NOT EXISTS(SELECT * FROM CompanyClients WHERE ClientID = @CompanyID)
        RAISERROR('Nie ma takiej firmy w bazie', 16, 1)
    DECLARE @ClientID int
    SELECT @ClientID = MAX(ClientID) + 1 FROM Clients
    INSERT INTO IndividualClients(ClientID, FirstName, LastName, Phone, NumOrders, CompanyID)
    VALUES (@ClientID, @FirstName, @LastName, @Phone, 0, @CompanyID)
    INSERT INTO Discounts(ClientID, D1, D2Used, D2LatestDate)
    VALUES (@ClientID, 0, 0, null)
```

5.5.AddOrder

AddOrder (@ClientID INT, @Discount FLOAT, @RealisationDate DATETIME, @ToGo BIT, @EmployeeID INT)

Wybiera najkorzystniejszą zniżkę, ustawia datę zamówienia na aktualną, ustawia status zamówienia na niezaakceptowany i dodaje zamówienie.

5.6.AddOrderDetail

AddOrderDetail (@OrderID INT, @PositionID INT, @Quantity INT)

Dodaje pozycję z zamówienia do bazy i aktualizuje dostępne zniżki dla klienta.

```
CREATE PROCEDURE [dbo].[AddOrderDetail]
    @OrderID int,
    @PositionID int,
    @Quantity int
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
        RAISERROR(N'Nie ma takiego zamówienia w bazie', 16, 1)
    IF NOT EXISTS(SELECT * FROM Menu WHERE PositionID = @PositionID)
        RAISERROR('Nie ma takiej pozycji w menu', 16, 1)
    DECLARE @UnitPrice money
    SELECT @UnitPrice = UnitPrice FROM AllPositions WHERE PositionID = @PositionID
    INSERT INTO OrderDetails(OrderID, PositionID, Quantity, UnitPrice)
    VALUES (@OrderID, @PositionID, @Quantity, @UnitPrice)
    DECLARE @ClientID int
    SELECT @ClientID = ClientID FROM Orders WHERE OrderID = @OrderID
    IF (SELECT D1 FROM Discounts WHERE ClientID = @ClientID) = 0 AND
    dbo.IsEntitledToReusableDiscount(@ClientID) = 1
        UPDATE Discounts
        SET D1 = 1
        WHERE ClientID = @ClientID
    IF dbo.IsEntitledToOneTimeDiscount(@ClientID) = 1
        UPDATE Discounts
        SET D2Used = 0, D2LatestDate = GETDATE()
        WHERE ClientID = @ClientID
```

5.7.AddPosition

AddPosition (@PositionName VARCHAR(255), @CategoryName, VARCHAR(255), @UnitPrice MONEY)

Dodaje pozycję do bazy.

5.8. AddPositionToMenu

AddPositionToMenu(@PositonID INT, @ForceAdd BIT)

Dodaje podaną pozycję z AllPositions do aktywnego Menu. Jeżeli pozycja nie istnieje lub jest już aktywna wyrzuca błąd. Jeśli pozycja ostatnio była aktywna mniej niż 2 tygodnie to program wyrzuci błąd chyba że parametr @ForceAdd jest równy 1.

```
CREATE PROCEDURE [dbo].[AddPositionToMenu]
    @PositionID int,
    @ForceAdd bit
AS
    IF @PositionID NOT IN (SELECT PositionID FROM AllPositions
    WHERE LastActive IS NOT NULL)
        RAISERROR(N'Pozycja nie istnieje lub jest już aktywna',16,1)

    IF @ForceAdd = 1 OR (DATEDIFF(DAY, (SELECT MIN(LastActive) FROM
    AllPositions WHERE PositionID = @PositionID), GETDATE()) < 14)
        RAISERROR(N'Pozycja powtórzy się, nie minęły jeszcze 2 tygodnie',16,1)

    UPDATE AllPositions SET LastActive = NULL WHERE PositionID = @PositionID
```

5.9. SwapOldestPosition

SwapOldestPosition (@Category_name varchar(255))

Podmienia najstarszą pozycję podanej kategori z AllPositions i Menu.

Jeśli od
najstarszych pozycji nie minęły dwa tygodnie to wyrzuca błąd.

5.10. ChangePrice

ChangePrice (@PositionID INT, @UnitPrice INT)

Zmienia cenę pozycji i odnotowuje to w historii cen.

```
CREATE PROCEDURE [dbo].[ChangePrice]
    @PositionID int,
    @UnitPrice int
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT * FROM AllPositions WHERE PositionID = @PositionID)
        RAISERROR(N'Taka pozycja nie istnieje', 16, 1)
    UPDATE AllPositions
    SET UnitPrice = @UnitPrice
    WHERE PositionID = @PositionID
    UPDATE PriceHistory
    SET EndDate = GETDATE()
    WHERE PositionID = @PositionID AND EndDate IS NULL
    INSERT INTO PriceHistory(PositionID, UnitPrice, StartDate, EndDate)
    VALUES (@PositionID, @UnitPrice, DATEADD(day, 1, GETDATE()), null)
```

5.11. ChangeVariables

ChangeVariables (@Z1 INT, @K1 INT, @R1 FLOAT, @K2 INT, @D1 INT, @R2 FLOAT)

Zmienia wartości zmiennych i odnotowuje to w historii.

6. Funkcje

6.1. ClientDiscounts

dbo.ClientDiscounts (@clientID INT)

Funkcja zwraca tabelę ze szczegółami zniżek dla podanego klienta.

```
CREATE FUNCTION [dbo].[ClientDiscounts](@ClientID int)
RETURNS @Discounts TABLE (
    D1 bit,
    R1 float,
    OrdersLeft int,
    D2 bit,
    R2 float,
    ExpireDate date,
    AmountToGetNext money
)
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM IndividualClients WHERE ClientID = @ClientID)
        RETURN
    DECLARE @D1 bit
    SELECT @D1 = D1 FROM Discounts WHERE ClientID = @ClientID
    DECLARE @R1 float
    SELECT @R1 = R1 FROM Variables
    DECLARE @D2 bit
    IF dbo.CanUseOneTimeDiscount(@ClientID) = 1
        SET @D2 = 1
    ELSE
        SET @D2 = 0
    DECLARE @R2 float
    SELECT @R2 = R2 FROM Variables
    DECLARE @OrdersLeft int
    IF @D1 = 1
        SET @OrdersLeft = 0
    ELSE
    BEGIN
        DECLARE @K1 money
        SELECT @K1 = K1 FROM Variables
        DECLARE @Z1 int
        SELECT @Z1 = Z1 FROM Variables
        SELECT @OrdersLeft = COUNT(OrderID)
        FROM Orders
        WHERE ClientID = @ClientID AND Accepted = 1 AND dbo.PriceOfOrder(OrderID) >= @K1
        SET @OrdersLeft = @Z1 - @OrdersLeft
    END
    DECLARE @Date date
    SELECT @Date = D2LatestDate FROM Discounts WHERE ClientID = @ClientID
    DECLARE @Days int
    SELECT @Days = D1 FROM Variables
    DECLARE @ExpireDate date
    SET @ExpireDate = DATEADD(day, @Days, @Date)
    DECLARE @K2 money
    SELECT @K2 = K2 FROM Variables
    DECLARE @Amount money
    SELECT @Amount = ISNULL(SUM(dbo.PriceOfOrder(OrderID)), 0)
    FROM Orders
    WHERE ClientID = @ClientID AND Accepted = 1 AND OrderDate > @Date
    SET @Amount = @K2 - @Amount
    INSERT INTO @Discounts (D1, R1, OrdersLeft, D2, R2, ExpireDate, AmountToGetNext)
    VALUES (@D1, @R1, @OrdersLeft, @D2, @R2, @ExpireDate, @Amount)
    RETURN
END
```

6.2. ClientOrdersHistory

dbo.ClientOrdersHistory(@clientID INT)

Funkcja zwraca tabelę z wszystkimi zaakceptowanymi zamówieniami podanego klienta.

6.3. ClientsFromCompany

dbo.ClientsFromCompany(@companyID INT)

Funkcja zwraca tabelę z klientami indywidualnymi pracującymi dla podanej firmy.

6.4. EmployeeCTOperateMonth

dbo.EmployeeCTOperateMonth(@idMonth INT, @idEmployee INT)

Funkcja licząca liczbę zamówień obsłużonych przez podanego pracownika w podanym miesiącu zeszłego roku.

```
CREATE FUNCTION [dbo].[EmployeeCTOperateMonth] (@idMonth int, @idEmployee int)
RETURNS TABLE AS
RETURN((SELECT
        COUNT(TR.WithDiscount) total
    FROM TotalReceipt TR JOIN Orders O
    ON O.OrderID = TR.OrderID
    WHERE (YEAR(GETDATE()) <= YEAR(O.OrderDate) OR ( YEAR(GETDATE()) - 1 = YEAR(O.OrderDate)
    AND MONTH(GETDATE()) <= MONTH(O.OrderDate) ))
    AND (O.EmployeeID =@idEmployee) AND MONTH(O.OrderDate) = @idMonth))
```

6.5. EmployeeOperateMonth

dbo.EmployeeOperateMonth(@idMonth INT, @idEmployee INT)

Funkcja licząca łączną kwotę zamówień obsłużonych przez podanego pracownika w podanym miesiącu zeszłego roku.

```
CREATE FUNCTION [dbo].[EmployeeOperateMonth] (@idMonth int, @idEmployee int)
RETURNS TABLE AS
RETURN((SELECT
        SUM(TR.WithDiscount) total
    FROM TotalReceipt TR JOIN Orders O
    ON O.OrderID = TR.OrderID
    WHERE (YEAR(GETDATE()) <= YEAR(O.OrderDate) OR ( YEAR(GETDATE()) - 1 = YEAR(O.OrderDate)
    AND MONTH(GETDATE()) <= MONTH(O.OrderDate) ))
    AND (O.EmployeeID =@idEmployee) AND MONTH(O.OrderDate) = @idMonth))
```

6.6. MonthlyEmployeeCTOperate

dbo.MonthlyEmployeeCTOperate ()

Funkcja generująca dane do widoku MonthlyOperatedReportForEmployee.

6.7. MonthlyEmployeeOperate

dbo.MonthlyEmployeeOperate()

Funkcja generująca dane do widoku MonthlyIncomeReportForEmployee.

```
CREATE FUNCTION [dbo].[MonthlyEmployeeOperate] ()
RETURNS @myTable TABLE
        ([id] float,
         [Jan] float,
         [Feb] float,
         [Mar] float,
         [Apr] float,
         [May] float,
         [Jun] float,
         [Jul] float,
         [Aug] float,
         [Sep] float,
         [Oct] float,
         [Nov] float,
         [Dec] float)
AS BEGIN
    DECLARE
        @idEmployee int = 1,
        @maxEmployee int = (SELECT COUNT(*) FROM Employees),
        @jan float = 0,
        @feb float = 0,
        @mar float = 0,
        @apr float = 0,
        @may float = 0,
        @jun float = 0,
        @jul float = 0,
        @aug float = 0,
        @sep float = 0,
        @oct float = 0,
        @nov float = 0,
        @dec float = 0
    WHILE @idEmployee <= @maxEmployee

   BEGIN
       SET @jan = (SELECT * FROM dbo.EmployeeOperateMonth(1,@idEmployee))
       SET @feb = (SELECT * FROM dbo.EmployeeOperateMonth(2,@idEmployee))
       SET @mar = (SELECT * FROM dbo.EmployeeOperateMonth(3,@idEmployee))
       SET @apr = (SELECT * FROM dbo.EmployeeOperateMonth(4,@idEmployee))
       SET @may = (SELECT * FROM dbo.EmployeeOperateMonth(5,@idEmployee))
       SET @jun = (SELECT * FROM dbo.EmployeeOperateMonth(6,@idEmployee))
       SET @jul = (SELECT * FROM dbo.EmployeeOperateMonth(7,@idEmployee))
       SET @aug = (SELECT * FROM dbo.EmployeeOperateMonth(8,@idEmployee))
       SET @sep = (SELECT * FROM dbo.EmployeeOperateMonth(9,@idEmployee))
       SET @oct = (SELECT * FROM dbo.EmployeeOperateMonth(10,@idEmployee))
       SET @nov = (SELECT * FROM dbo.EmployeeOperateMonth(11,@idEmployee))
       SET @dec = (SELECT * FROM dbo.EmployeeOperateMonth(12,@idEmployee))

       INSERT INTO @myTable ([id],[Jan],[Feb],[Mar],[Apr],[May],[Jun],[Jul],[Aug],[Sep],[Oct],[Nov],[Dec])
       VALUES (@idEmployee,@jan,@feb,@mar,@apr,@may,@jun,@jul,@aug,@sep,@oct,@nov,@dec)
       SET @idEmployee = @idEmployee + 1
   END

   UPDATE @myTable SET Jan=0.0 WHERE Jan is NULL
   UPDATE @myTable SET Feb=0.0 WHERE Feb is NULL
   UPDATE @myTable SET Mar=0.0 WHERE Mar is NULL
   UPDATE @myTable SET Apr=0.0 WHERE Apr is NULL
   UPDATE @myTable SET May=0.0 WHERE May is NULL
   UPDATE @myTable SET Jun=0.0 WHERE Jun is NULL
   UPDATE @myTable SET Jul=0.0 WHERE Jul is NULL
   UPDATE @myTable SET Aug=0.0 WHERE Aug is NULL
   UPDATE @myTable SET Sep=0.0 WHERE Sep is NULL
   UPDATE @myTable SET Oct=0.0 WHERE Oct is NULL
   UPDATE @myTable SET Nov=0.0 WHERE Nov is NULL
   UPDATE @myTable SET Dec=0.0 WHERE Dec is NULL

   RETURN
END
```

6.8. MonthlyIncomeAndAmount

dbo.MonthlyIncomeAndAmount()

Funkcja generująca dane do widoku MonthlyIncome.

```
CREATE FUNCTION [dbo].[MonthlyIncomeAndAmount] ()
RETURNS @myTable TABLE
        ([Month] int,
         [Total] float,
         [Amount] int)
AS
BEGIN
    INSERT INTO @myTable ([Month], [Total], [Amount])
    SELECT tmp.mont, tmp.total, tmp.amount
    FROM
    (SELECT MONTH(O.OrderDate) mont,
        SUM(TR.WithDiscount) total,
        COUNT(TR.WithDiscount) amount
    FROM TotalReceipt TR JOIN Orders O
    ON O.OrderID = TR.OrderID
    WHERE YEAR(GETDATE()) <= YEAR(O.OrderDate) OR ( YEAR(GETDATE()) - 1 = YEAR(O.OrderDate)
    AND MONTH(GETDATE()) <= MONTH(O.OrderDate) )
    GROUP BY MONTH(O.OrderDate)) tmp

    DECLARE @idColumn int = 1
    WHILE @idColumn <= 12
    BEGIN
        IF @idColumn NOT IN (SELECT [Month] FROM @myTable)
        BEGIN
            INSERT INTO @myTable ([Month], [Total], [Amount]) VALUES (@idColumn, 0.0, 0)
        END
        SET @idColumn = @idColumn + 1
    END

    RETURN

END
```

6.9. OrdersOperatedByEmployee

dbo.OrdersOperatedByEmployee(@employeeID INT)

Funkcja zwraca tabelę z zamówieniami obsłużonymi przez podanego pracownika.

6.10. OrdersOrderedBetween

dbo. OrdersOrderedBetween(@start DATE, @end DATE)

Funkcja zwraca tabelę z zamówieniami zamówionymi pomiędzy podanymi datami.

```
CREATE FUNCTION [dbo].[OrdersOrderedBetween] (
    @start DATETIME = '2000-01-01 23:59:59.000',
    @end DATETIME = '3000-01-01 23:59:59.000')
    RETURNS TABLE AS
    RETURN (SELECT * FROM Orders
            WHERE OrderDate >= @start AND OrderDate <= @end)
```

6.11. OrdersRealizedBetween

dbo.OrdersRealisedBetween(@start DATE, @end DATE)

Funkcja zwraca tabelę z zamówieniami zrealizowanymi pomiędzy podanymi datami.

6.12. CanUseOneTimeDiscount

dbo.CanUseOneTimeDiscount(@clientID INT)

Funkcja informuje, czy podany klient może skorzystać z jednorazowej zniżki.

```
CREATE FUNCTION [dbo].[CanUseOneTimeDiscount] (@ClientID int)
RETURNS bit
AS
BEGIN
    DECLARE @D2LatestDate date
    SELECT @D2LatestDate = D2LatestDate FROM Discounts WHERE ClientID = @ClientID
    DECLARE @D2Used bit
    SELECT @D2Used = D2Used FROM Discounts WHERE ClientID = @ClientID
    DECLARE @D1 int
    SELECT @D1 = D1 FROM Variables
    IF @D2LatestDate IS NOT NULL AND @D2Used = 0 AND GETDATE() <= DATEADD(day, @D1, @D2LatestDate)
        RETURN 1
    RETURN 0
END
```

6.13. IsEntitledToOneTimeDiscount

dbo.IsEntitledToOneTimeDiscount(@clientID INT)

Funkcja informuje, czy podanemu klientowi należy się nowa jednorazowa zniżka.

```
CREATE FUNCTION [dbo].[IsEntitledToOneTimeDiscount] (@ClientID INT)
RETURNS bit
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM IndividualClients WHERE ClientID = @ClientID)
        RETURN 0
    DECLARE @K2 money
    SELECT @K2 = K2 FROM Variables
    DECLARE @Date date
    SELECT @Date = D2LatestDate FROM Discounts WHERE ClientID = @ClientID
    DECLARE @TotalCost int
    SELECT @TotalCost = SUM(dbo.PriceOfOrder(OrderID))
    FROM Orders
    WHERE ClientID = @ClientID AND Accepted = 1 AND OrderDate > @Date
    IF (@TotalCost >= @K2)
        RETURN 1
    RETURN 0
END
```

6.14. IsEntitledToReusableDiscount

dbo.IsEntitledToOneTimeDiscount(@clientID INT)

Funkcja informuje, czy podanemu klientowi należy się zniżka na wszystkie zmówienia.

```
CREATE FUNCTION [dbo].[IsEntitledToReusableDiscount] (@ClientID INT)
RETURNS bit
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM IndividualClients WHERE ClientID = @ClientID)
        RETURN 0
    DECLARE @K1 money
    SELECT @K1 = K1 FROM Variables
    DECLARE @Z1 int
    SELECT @Z1 = Z1 FROM Variables
    DECLARE @NumberOfOrders int
    SELECT @NumberOfOrders = COUNT(OrderID)
    FROM Orders
    WHERE ClientID = @ClientID AND Accepted = 1 AND dbo.PriceOfOrder(OrderID) >= @K1
    IF (@NumberOfOrders >= @Z1)
        RETURN 1
    RETURN 0
END
```

6.15. PriceOfOrder

dbo.PriceOfOrder(@OrderID INT)

Funkcja zwraca wartość podanego zamówienia z nałożonym rabatem.

6.16. SumClientReceipt

dbo.SumClientReceipt(@clientID INT)

Funkcja zwraca sumę wartości wszystkich zaakceptowanych zamówień złożonych przez podanego klienta.

```
CREATE FUNCTION [dbo].[SumClientReceipt] (@clientID INT)
RETURNS FLOAT AS
BEGIN
    RETURN (SELECT ROUND(SUM(Quantity* UnitPrice*(1-Discount)),2) AS A
        FROM Orders O JOIN OrderDetails OD ON O.OrderID = OD.OrderID
        WHERE Accepted = 1
        GROUP BY ClientID
        HAVING ClientID = @clientID)
END
```

6.17. OrdersInDayLastMonth

Dbo.OrdersInDayLastMonth(@day INT)

Funkcja zwraca liczbę zamówień w określonym dniu tygodnia w zeszłym miesiącu

```
CREATE FUNCTION OrdersInDayLastMonth (@day INT)
RETURNS INT AS
BEGIN
    RETURN (SELECT COUNT(*) FROM (
    SELECT DATEPART(WEEKDAY, RealisationDate) dt FROM Orders WHERE
    (MONTH(RealisationDate) + 1 = MONTH(GETDATE()) AND YEAR(RealisationDate) = YEAR(GETDATE()))
    OR (MONTH(RealisationDate) = 12 AND MONTH(GETDATE()) = 1 AND YEAR(RealisationDate) + 1 = YEAR(GETDATE()))
    ) TMP WHERE TMP.dt = @day)
END
```

6.18. OrdersLastMonth

Dbo.OrdersLastMonth()

Funkcja zwraca tabelę zawierającą liczbę zamówień z podziałem na dni tygodnia.

```
CREATE FUNCTION OrdersLastMonth ()
RETURNS @myTable TABLE
    ([mon] int,
     [tue] int,
     [wed] int,
     [thu] int,
     [fri] int,
     [sat] int,
     [sun] int)
AS BEGIN
    DECLARE
    @MO int = (SELECT * FROM dbo.OrdersInDayLastMonth(1)),
    @TU int = (SELECT * FROM dbo.OrdersInDayLastMonth(2)),
    @WE int = (SELECT * FROM dbo.OrdersInDayLastMonth(3)),
    @TH int = (SELECT * FROM dbo.OrdersInDayLastMonth(4)),
    @FR int = (SELECT * FROM dbo.OrdersInDayLastMonth(5)),
    @SA int = (SELECT * FROM dbo.OrdersInDayLastMonth(6)),
    @SU int = (SELECT * FROM dbo.OrdersInDayLastMonth(7))

    INSERT INTO @myTable ([mon], [tue], [wed],
     [thu], [fri], [sat], [sun])
     VALUES (@MO, @TU, @WE, @TH, @FR, @SA, @SU)
    RETURN
```

7. Triggery

7.1. UpdateMenu

Trigger zostaje aktywowany w momencie zmiany, dodania bądź usunięcia danych do tabeli AllPositions. Odpowiada za poprawne wyświetlanie aktywnego menu.

7.2. AddCompanyClientToClients

Trigger zostaje aktywowany po dodaniu nowej firmy do klientów. Dodaje nowego klienta do tabeli Clients

7.3. AddIndividualClientToClients

Trigger zostaje aktywowany po dodaniu nowego indywidualnego klienta. Dodaje nowego klienta do tabeli Clients.

8. Generowanie danych

Wszystkie dane zostały wygenerowane za pomocą strony www.mockaroo.com. Podczas generowania danych ustawialiśmy ograniczenia do generowania w taki sposób, aby dane pomiędzy tabelami były zgodne między sobą oraz z warunkami integralnościowymi.