Magdalena Skarbińska
Sebastian Misztal
Piotr Magiera

Projekt bazy danych dla restauracji

Spis treści

Funkcjonalność

I. Klienci indywidualni

a. składanie zamówienia na miejscu lub na wynos na stronie www
b. dostęp do menu
c. możliwość rezerwacji stolika (dla co najmniej 2 osób)
d. rezerwacja stolika z warunkami
e. potwierdzenie zamówienia
f. prośba o miesięczną fakturę
g. podgląd do historii zamówień

II. Firmy

a. składanie zamówienia na miejscu lub na wynos na stronie www
b. dostęp do menu
c. rezerwacja stolików (na firmę lub na pracownika)
d. potwierdzenie zamówienia
e. prośba o miesięczną fakturę
f. pogląd do historii zamówień

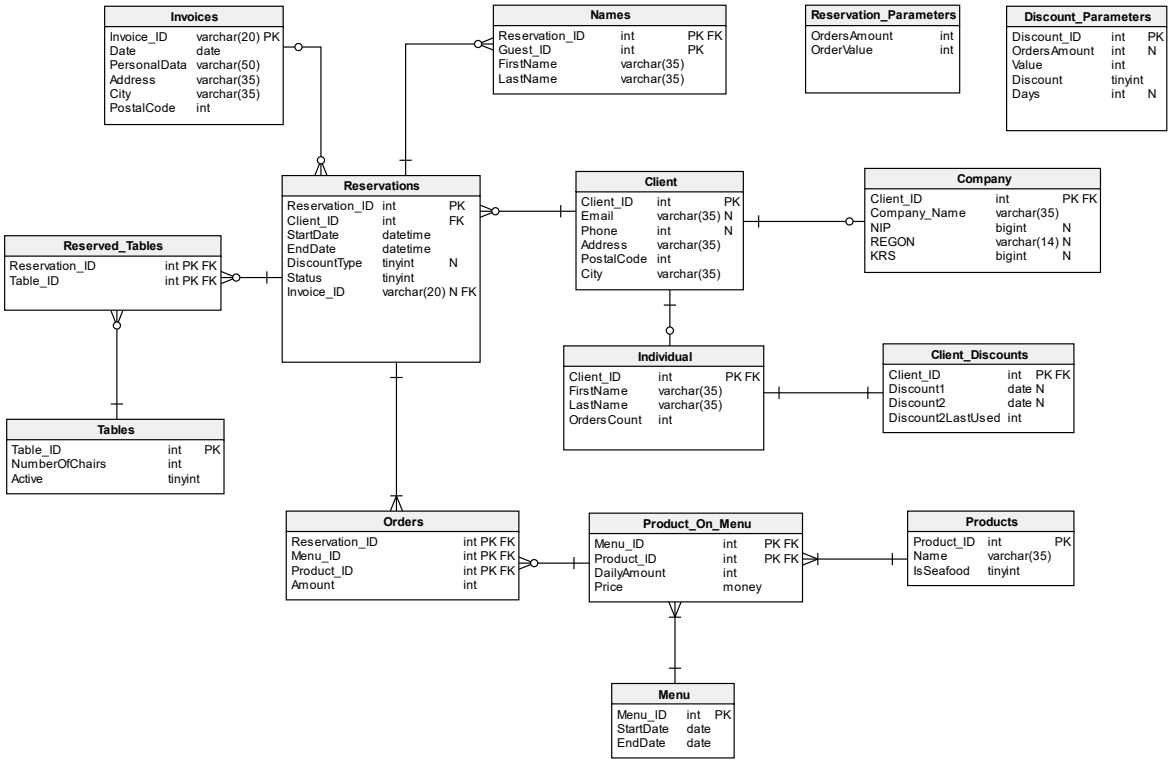
III. Obsługa

a. wystawienie faktury dla zamówienia
b. wystawienie faktur zbiorczych
c. akceptacja i wskazanie stolika
d. sprawdzanie statusu zamówienia (płatność)
e. przyjmowanie zamówienia osobiście

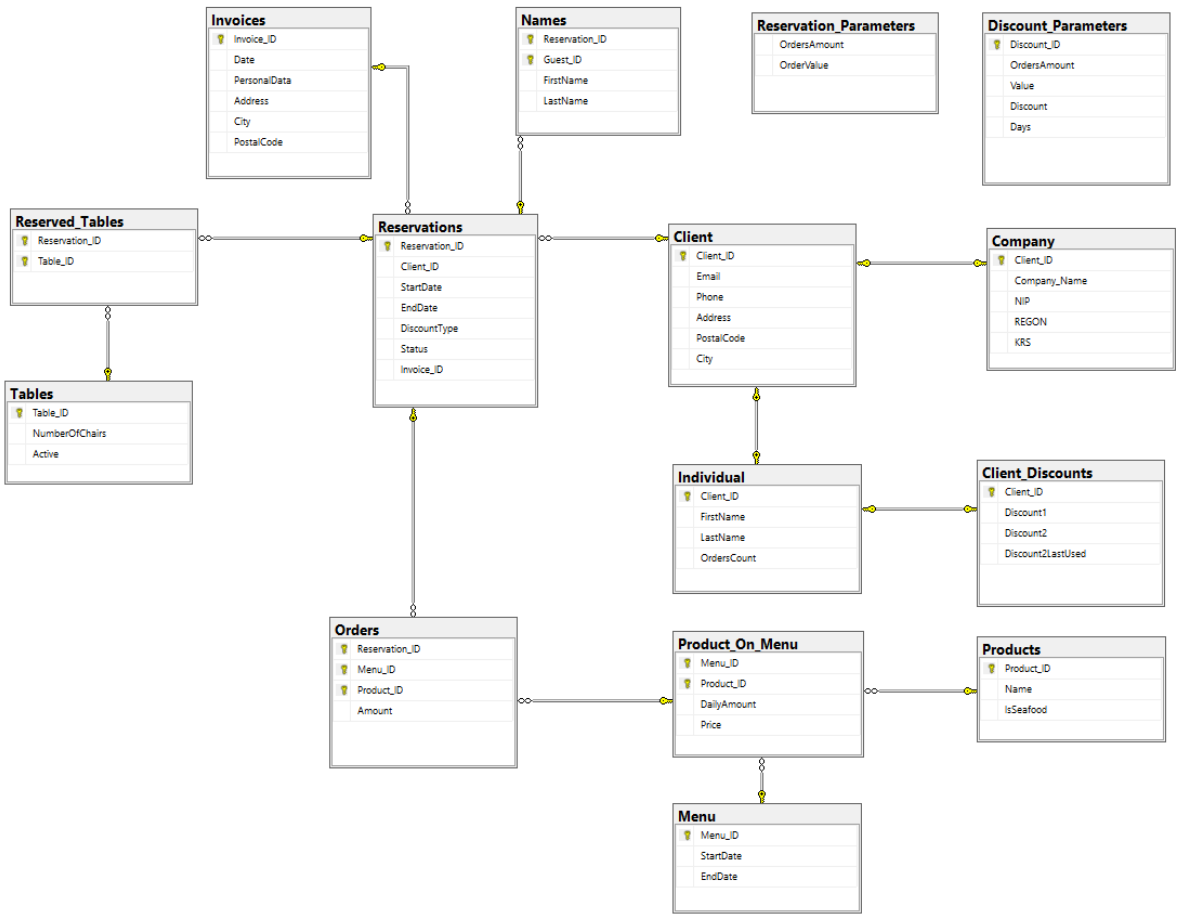
IV. Szef restauracji

a. ustalanie menu
b. generowanie raportu miesięcznego – przychód, średnia wartość zamówienia, liczba rezerwacji, ile każdego produktu zostało sprzedane
c. aktualizacja danych

Schemat bazy danych



Vertabelo



Tabele i warunki integralności

I. Client

Tabela przechowująca informacje o klientach korzystających z systemu.

Client_ID – unikalny numer klienta służący identyfikacji.
Email – adres mailowy.
Phone – telefon kontaktowy.
Address – ulica i numer domu.
PostalCode – kod pocztowy.
City – miasto zamieszkania.

```
1.  CREATE TABLE Client (
2.      Client_ID int  NOT NULL,
3.      Email varchar(35)  NULL,
4.      Phone int  NULL,
5.      Address varchar(35)  NOT NULL,
6.      PostalCode int  NOT NULL,
7.      City varchar(35)  NOT NULL,
8.      CONSTRAINT Email_check CHECK (Email like '%[a-zA-Z0-9][@][a-zA-Z0-9]%[.][a-zA-
    Z0-9]%'),
9.      CONSTRAINT Phone_check CHECK (CAST(Phone as nvarchar) like  '[0-9][0-9][0-9][0-
    9][0-9][0-9][0-9][0-9][0-9]'),
10.     CONSTRAINT PostalCode_check CHECK (PostalCode like '[0-9][0-9][0-9][0-9][0-
    9]'),
11.     CONSTRAINT Address_check CHECK (Address not like '%[^a-zA-Z0-9. ]%'),
12.     CONSTRAINT City_check CHECK (City not like '%[^a-zA-Z ]%'),
13.     CONSTRAINT Client_pk PRIMARY KEY  (Client_ID)
14. );
```

II. Client_Discounts

Tabela przechowująca dane o zniżkach dostępnych dla danego klienta.

Client_ID – numer identyfikacyjny klienta, którego dotyczą zniżki.
Discount1 – data, od której dostępna jest pierwsza zniżka (ustawiana tylko raz).
Discount2 – data, od której dostępna jest druga zniżka (do wykorzystania przez liczbę dni z tabeli Discount_Parameters).
Discount2LastUsed – numer identyfikacyjny zamówienia, w którym ostatnio została użyta druga zniżka.

```
1.  CREATE TABLE Client_Discounts (
2.      Client_ID int  NOT NULL,
3.      Discount1 date  NULL,
4.      Discount2 date  NULL,
5.      Discount2LastUsed int  NOT NULL,
6.      CONSTRAINT Discount2LastUsed_Check CHECK (Discount2LastUsed >= 0),
7.      CONSTRAINT Client_Discounts_pk PRIMARY KEY  (Client_ID)
8.  );
```

III. Company

Tabela przechowująca dane firm będących klientami restauracji.

Client_ID – numer identyfikacyjny klienta firmowego.
Company_Name – nazwa firmy.
NIP, REGON, KRS – identyfikatory potrzebne do faktur.

```
1.  CREATE TABLE Company (
2.      Client_ID int  NOT NULL,
3.      Company_Name varchar(35)  NOT NULL,
4.      NIP bigint  NULL,
5.      REGON varchar(14)  NULL,
6.      KRS bigint  NULL,
7.      CONSTRAINT NIP_check CHECK (CAST(NIP as nvarchar) like '[0-9][0-9][0-9][0-9][0-
    9][0-9][0-9][0-9][0-9][0-9]'),
8.      CONSTRAINT REGON_check CHECK (REGON like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
    9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
9.      CONSTRAINT KRS_check CHECK (CAST(KRS as nvarchar)  like '[0-9][0-9][0-9][0-
    9][0-9][0-9][0-9][0-9][0-9][0-9]'),
10.     CONSTRAINT Company_pk PRIMARY KEY  (Client_ID)
11. );
```

IV. Discount_Parameters

Tabela przechowująca szczegółowe dane o zniżkach.

Discount_ID – numer zniżki.
OrdersAmount – minimalna liczba zamówień konieczna do uzyskania zniżki.
Value – minimalna wartość każdego zamówienia konieczna do uzyskania zniżki (0 w przypadku zniżki nr 1).
Discount – wartość zniżki w procentach.
Days – długość trwania zniżki (NULL w przypadku nieskończonej liczby dni).

```
1.  CREATE TABLE Discount_Parameters (
2.      Discount_ID int  NOT NULL,
3.      OrdersAmount int  NULL,
4.      Value int  NOT NULL,
5.      Discount tinyint  NOT NULL,
6.      Days int  NULL,
7.      CONSTRAINT OrdersAmount_Check2 CHECK (OrdersAmount > 0),
8.      CONSTRAINT Value_Check CHECK (Value > 0),
9.      CONSTRAINT Discount_Check CHECK (Discount > 0 and Discount <= 100),
10.     CONSTRAINT Days_Check CHECK (Days > 0),
11.     CONSTRAINT Discount_Parameters_pk PRIMARY KEY  (Discount_ID)
12. );
```

V. Individual

Tabela z informacjami o kliencie indywidualnym.

Client_ID – numer identyfikacyjny klienta.
FirstName – imię klienta.
LastName – nazwisko klienta.
OrdersCount – liczba dotychczasowych zamówień klienta (używana do określenia, czy może dokonać rezerwacji).

```
1.  CREATE TABLE Individual (
2.      Client_ID int  NOT NULL,
3.      FirstName varchar(35)  NOT NULL,
4.      LastName varchar(35)  NOT NULL,
5.      OrdersCount int  NOT NULL,
6.      CONSTRAINT OrdersCount_Check CHECK (OrdersCount >= 0),
7.      CONSTRAINT Individual_pk PRIMARY KEY  (Client_ID)
8.  );
```

VI. Invoices

Tabela z informacjami o wydanych fakturach.

Invoice_ID – numer identyfikacyjny faktury.
Date – data wydania faktury.
PersonalData – dane osoby, na którą wystawiana jest faktura.
Address – ulica i numer domu.
PostalCode – kod pocztowy.
City – miasto zamieszkania.

```
1.  CREATE TABLE Invoices (
2.      Invoice_ID varchar(20)  NOT NULL,
3.      Date date  NOT NULL,
4.      PersonalData varchar(50)  NOT NULL,
5.      Address varchar(35)  NOT NULL,
6.      City varchar(35)  NOT NULL,
7.      PostalCode int  NOT NULL,
8.      CONSTRAINT Invoice_ID_Check CHECK (Invoice_ID not like '%[^0-9/]%'),
9.      CONSTRAINT Postal_Code_Check_2 CHECK (PostalCode like '[0-9][0-9][0-9][0-9][0-
    9]'),
10.     CONSTRAINT Address_Check_2 CHECK (Address not like '%[^a-zA-Z0-9. ]%'),
11.     CONSTRAINT City_Check_2 CHECK (City not like '%[^a-zA-Z ]%'),
12.     CONSTRAINT Invoices_pk PRIMARY KEY  (Invoice_ID)
13. );
```

VII. Menu

Tabela przechowująca informacje o danym menu.

Menu_ID – numer identyfikacyjny menu.
StartDate – data początku obowiązywania menu.
EndDate – data końca obowiązywania menu.

```
1.  CREATE TABLE Menu (
2.      Menu_ID int  NOT NULL,
3.      StartDate date  NOT NULL,
4.      EndDate date  NOT NULL,
5.      CONSTRAINT Date_Check CHECK (StartDate <= EndDate),
6.      CONSTRAINT Menu_pk PRIMARY KEY  (Menu_ID)
7.  );
```

VIII. Names

Tabela z informacjami o danych osobowych osób, na które firma złożyła rezerwację.

First_Name – imię danej osoby.
Last_Name – nazwisko danej osoby.

```
1.  CREATE TABLE Names (
2.      Reservation_ID int  NOT NULL,
3.      Guest_ID int  NOT NULL,
4.      FirstName varchar(35)  NOT NULL,
5.      LastName varchar(35)  NOT NULL,
6.      CONSTRAINT Names_pk PRIMARY KEY  (Guest_ID,Reservation_ID)
7.  );
```

IX. Orders

Tabela przechowująca informacje o zamówieniach.

Reservation_ID – numer identyfikacyjny rezerwacji.
Menu_ID – numer identyfikacyjny menu, z którego zostało złożone zamówienie.
Product_ID – numer identyfikacyjny zamawianego produktu.
Amount – ilość zamawianego produktu.

```
1.  CREATE TABLE Orders (
2.      Reservation_ID int  NOT NULL,
3.      Menu_ID int  NOT NULL,
4.      Product_ID int  NOT NULL,
5.      Amount int  NOT NULL,
6.      CONSTRAINT Amount_check CHECK (Amount > 0),
7.      CONSTRAINT Orders_pk PRIMARY KEY  (Reservation_ID,Menu_ID,Product_ID)
8.  );
```

X. Product_On_Menu

Tabela z informacjami o produktach w menu.

Menu_ID – numer identyfikacyjny menu, w którym dany produkt się znajduje.
Product_ID – numer identyfikacyjny produktu.
DailyAmount – dzienny limit danego produktu.
Price – cena produktu w danym menu.

```
1.  CREATE TABLE Product_On_Menu (
2.      Menu_ID int  NOT NULL,
3.      Product_ID int  NOT NULL,
4.      DailyAmount int  NOT NULL,
5.      Price money  NOT NULL,
6.      CONSTRAINT DailyAmount_Check CHECK (DailyAmount > 0),
7.      CONSTRAINT Price_Check CHECK (Price > 0),
8.      CONSTRAINT Product_On_Menu_pk PRIMARY KEY  (Menu_ID,Product_ID)
9.  );
```

XI. Products

Tabela z informacjami o szczegółach dotyczących danego produktu (dania).

Product_ID – numer identyfikacyjny produktu.
Name – nazwa produktu.
IsSeafood – informacja o tym, czy produkt jest z kategorii „owoce morza” (0 – nie, 1 – tak).

```
1.  CREATE TABLE Products (
2.      Product_ID int IDENTITY  NOT NULL,
3.      Name varchar(35)  NOT NULL,
4.      IsSeafood tinyint  NOT NULL,
5.      CONSTRAINT Name_Check CHECK (Name not like '%[^a-zA-Z ]%'),
6.      CONSTRAINT IsSeafood_Check CHECK (IsSeafood in (0, 1)),
7.      CONSTRAINT Products_pk PRIMARY KEY  (Product_ID)
8.  );
```

XII. Reservation_Parameters

Tabela przechowująca szczegóły dotyczące warunków składania rezerwacji online.

OrdersAmount – liczba poprzednich zamówień wymagana do złożenia rezerwacji.
OrderValue – minimalna wartość zamówień opisanych powyżej.

```
9.  CREATE TABLE Reservation_Parameters (
10.     OrdersAmount int  NOT NULL,
11.     OrderValue int  NOT NULL,
12.     CONSTRAINT OrdersAmount_Check CHECK (OrdersAmount > 0),
13.     CONSTRAINT OrderValue_Check CHECK (OrderValue > 0)
14. );
15.
```

XIII. Reservations

Tabela z informacjami o rezerwacjach dla danych klientów.

Reservation_ID – numer identyfikacyjny rezerwacji.
Client_ID – numer identyfikacyjny klienta.
StartDate – data i godzina początku rezerwacji.
EndDate – data i godzina końca rezerwacji.
DiscountType – numer zniżki przypisanej do zamówienia.
Status – status rezerwacji gdzie: 0 - nieopłacone na wynos , 1 - opłacone na wynos, 2- nieopłacone na miejscu, 3 - opłacone na miejscu, 4 - anulowane nieoddane, 5 - anulowane oddane, 6 – zapłacone zakończone.

```
16. CREATE TABLE Reservations (
17.     Reservation_ID int  NOT NULL,
18.     Client_ID int  NOT NULL,
19.     StartDate datetime  NOT NULL,
20.     EndDate datetime  NOT NULL,
21.     DiscountType tinyint  NULL,
22.     Status tinyint  NOT NULL,
23.     Invoice_ID varchar(20) NULL,
24.     CONSTRAINT Date_Check2 CHECK (StartDate <= EndDate),
25.     CONSTRAINT DiscountType_Check CHECK (DiscountType in (NULL, 0, 1)),
26.     CONSTRAINT Status_Check CHECK (Status in (0, 1, 2, 3, 4, 5, 6)),
27.     CONSTRAINT Reservations_pk PRIMARY KEY  (Reservation_ID)
28. );
29.
```

XIV. Reserved_Tables

Tabela z informacjami o zarezerwowanych stolikach

Reservation_ID – numer identyfikacyjny rezerwacji odnoszącej się do danego stolika.
Table_ID – numer identyfikacyjny stolika.

```
30. CREATE TABLE Reserved_Tables (
31.     Reservation_ID int  NOT NULL,
32.     Table_ID int  NOT NULL,
33.     CONSTRAINT Reserved_Tables_pk PRIMARY KEY  (Reservation_ID,Table_ID)
34. );
```

XV. Tables

Tabela z informacjami o stolikach w restauracji.

Table_ID – numer identyfikacyjny stolika.
NumberOfChairs – liczba krzeseł przy danym stoliku.
Active – informacja o tym, czy stolik jest obecnie w użyci (czy nie został wycofany, zniszczony etc.).

```
35. CREATE TABLE Tables (
36.     Table_ID int  NOT NULL,
37.     NumberOfChairs int  NOT NULL,
38.     Active tinyint  NOT NULL,
39.     CONSTRAINT Active_check CHECK (Active in (0, 1)),
40.     CONSTRAINT Chairs_check CHECK (NumberOfChairs > 0),
41.     CONSTRAINT Tables_pk PRIMARY KEY  (Table_ID)
42. );
43.
```

XVI. Relacje między tabelami

```
1.  -- foreign keys
2.  -- Reference: Client_Discounts_Clients (table: Client_Discounts)
3.  ALTER TABLE Client_Discounts ADD CONSTRAINT Client_Discounts_Clients
4.      FOREIGN KEY (Client_ID)
5.      REFERENCES Individual (Client_ID);
6.
7.  -- Reference: Company_Client (table: Company)
8.  ALTER TABLE Company ADD CONSTRAINT Company_Client
9.      FOREIGN KEY (Client_ID)
10.     REFERENCES Client (Client_ID);
11.
12. -- Reference: Individual_Client (table: Individual)
13. ALTER TABLE Individual ADD CONSTRAINT Individual_Client
14.     FOREIGN KEY (Client_ID)
15.     REFERENCES Client (Client_ID);
16.
17. -- Reference: Menu_Menu_date (table: Product_On_Menu)
18. ALTER TABLE Product_On_Menu ADD CONSTRAINT Menu_Menu_date
19.     FOREIGN KEY (Menu_ID)
20.     REFERENCES Menu (Menu_ID);
21.
22. -- Reference: Orders_Product_On_Menu (table: Orders)
23. ALTER TABLE Orders ADD CONSTRAINT Orders_Product_On_Menu
24.     FOREIGN KEY (Menu_ID,Product_ID)
25.     REFERENCES Product_On_Menu (Menu_ID,Product_ID);
26.
27. -- Reference: Orders_Reservations (table: Orders)
28. ALTER TABLE Orders ADD CONSTRAINT Orders_Reservations
29.     FOREIGN KEY (Reservation_ID)
30.     REFERENCES Reservations (Reservation_ID);
31.
32. -- Reference: Product_On_Menu_Products (table: Product_On_Menu)
33. ALTER TABLE Product_On_Menu ADD CONSTRAINT Product_On_Menu_Products
34.     FOREIGN KEY (Product_ID)
35.     REFERENCES Products (Product_ID);
36.
```

```
-- Reference: Reservations_Client (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Client
    FOREIGN KEY (Client_ID)
    REFERENCES Client (Client_ID);

-- Reference: Reservations_Invoices (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Invoices
    FOREIGN KEY (Invoice_ID)
    REFERENCES Invoices (Invoice_ID);

-- Reference: Reservations_Names (table: Names)
ALTER TABLE Names ADD CONSTRAINT Reservations_Names
    FOREIGN KEY (Reservation_ID)
    REFERENCES Reservations (Reservation_ID);

-- Reference: Reserved_Tables_Reservations (table: Reserved_Tables)
ALTER TABLE Reserved_Tables ADD CONSTRAINT Reserved_Tables_Reservations
    FOREIGN KEY (Reservation_ID)
    REFERENCES Reservations (Reservation_ID);

-- Reference: Reserved_Tables_Tables (table: Reserved_Tables)
ALTER TABLE Reserved_Tables ADD CONSTRAINT Reserved_Tables_Tables
    FOREIGN KEY (Table_ID)
    REFERENCES Tables (Table_ID);
```

Widoki

CurrentMenu

```
CREATE VIEW [dbo].[CurrentMenu] as
SELECT POM.Product_ID, POM.Menu_ID, M.StartDate, M.EndDate, POM.Price
FROM   dbo.Menu AS M
INNER JOIN dbo.Product_On_Menu AS POM ON M.Menu_ID = POM.Menu_ID
WHERE (CAST(M.EndDate AS DATE) >= CAST(GETDATE() AS DATE))
```

OrdersForToday

```
CREATE VIEW [dbo].[OrdersForToday] as
select p.Name, o.Amount, r.StartDate
from Orders o
join Products p on p.Product_ID = o.Product_ID
left join Reservations r on r.Reservation_ID = o.Reservation_ID
where r.Status in(0,2)
and CAST( r.StartDate AS DATE) = CAST( GETDATE() AS DATE);
```

Procedury

Od autorów:
Mamy świadomość, że użycie identity na ID większości tabel uprościłoby kod (brak konieczności używania funkcji getHighest(…)), jednak z uwagi na ograniczony zasób czasu zaimplementowaliśmy identity tylko dla tabeli Products.

addDishToProducts

Użyto IDENTITY

```
1.  CREATE PROCEDURE [dbo].[addDishToProducts]
2.  @Name AS varchar(35),
3.  @isSeaFood AS tinyint
4.  AS
5.  BEGIN
6.      BEGIN TRY
7.          IF EXISTS
8.              (
9.              SELECT * FROM Products
10.             WHERE @Name = Name
11.             )
12.         BEGIN
13.             ;THROW 52000, 'Product with this name is already in database.',1
14.         END
15.
16.         INSERT INTO Products(Name, IsSeafood)
17.         VALUES (@Name, @isSeaFood)
18.     END TRY
19.     BEGIN CATCH
20.     DECLARE @errorMsg nvarchar(2048)
21.                 = 'Cannot add dish to products. Error: ' + ERROR_MESSAGE();
22.             THROW 52000, @errorMSg, 1;
23.     END CATCH
24.     END;
```

AddOrderToReservation

```
1.  CREATE PROCEDURE [dbo].[AddOrderToReservation]
2.  @Reservation_ID int,
3.  @Product_ID int,
4.  @Amount int
5.  as
6.  begin
7.      set nocount on
8.      begin try
9.          insert into Orders
10.         (
11.             Reservation_ID,
12.             Menu_ID,
13.             Product_ID,
14.             Amount
15.         )
16.         values
17.         (
18.             @Reservation_ID,
19.             (select pom.Menu_ID from Product_On_Menu pom
20.                 join Menu m on m.Menu_ID = pom.Menu_ID
```

```
                where (select StartDate from Reservations where Reservation_ID =
@Reservation_ID)
                between m.StartDate and m.EndDate),
            @Product_ID,
            @Amount
        )
    end try
    begin catch
        declare @errorMsg nvarchar(2048)
        = 'Cannot add order to reservation. Error message: '
        + ERROR_MESSAGE();
    ;throw 52000, @errorMsg, 1
    end catch
end
```

AddPersonToReservation

```
CREATE PROCEDURE [dbo].[AddPersonToReservation]
@Reservation_ID int,
@FirstName varchar(35),
@Lastname varchar(35)
as
begin
    set nocount on
    begin try
        insert into Names
        (
            Reservation_ID,
            FirstName,
            LastName
        )
        values
        (
            @Reservation_ID,
            @FirstName,
            @Lastname
        )
    end try
    begin catch
        declare @errorMsg nvarchar(2048)
        = 'Cannot add person to reservation. Error message: '
        + ERROR_MESSAGE();
    ;throw 52000, @errorMsg, 1
    end catch
end
```

addProductToMenu

```
CREATE PROCEDURE [dbo].[addProductToMenu]
@ProductID AS int,
@DailyAmount AS int,
@Price AS money,
@StartDate AS date,
@EndDate AS date,
@MenuID AS int
AS
    BEGIN
        BEGIN TRY

        IF EXISTS
            (
            SELECT * FROM Product_On_Menu
            JOIN Menu m on m.Menu_ID = Product_On_Menu.Menu_ID
```

```
            where StartDate = @StartDate and Product_ID = @ProductID
            )
        BEGIN
            ;THROW 52000, 'Product with this id already exists in menu with this start date.',1
        END

        IF NOT(@StartDate < @EndDate)
        RETURN 1;

        INSERT INTO Menu(Menu_ID, StartDate, EndDate)
        VALUES (@MenuID , @StartDate, @EndDate)

        INSERT INTO Product_On_Menu(Menu_ID, Product_ID, DailyAmount, Price)
        VALUES (@MenuID , @ProductID, @DailyAmount, @Price)

        END TRY
        BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
                = 'Cannot add dish to menu. Error: ' + ERROR_MESSAGE();
            THROW 52000, @errorMSg, 1;
        END CATCH

    END
```

AddReservationToInvoice

```
CREATE PROCEDURE [dbo].[AddReservationToInvoice]
@Reservation_ID as int,
@Invoice_ID as int
as
    begin
        begin try
            if exists
            (select * from Reservations where Reservation_ID=@Reservation_ID and Invoice_ID is not null)
            begin
                ; throw 52000, 'reservation was already added to invoice',1
            end
            update Reservations
            set Invoice_ID = @Invoice_ID
            WHERE Reservation_ID = @Reservation_ID
        end try

        begin catch
            DECLARE @errorMsg nvarchar(2048)='Cannot asign reservation to invoice. Error: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1;
        END CATCH
    end
```

AddTableToReservation

```
CREATE PROCEDURE [dbo].[AddTableToReservation]
@Reservation_ID int,
@Table_ID int
as
begin
    set nocount on
    begin try
```

```
        if not exists
            (
                select * from Reservations
                where Reservation_ID = @Reservation_ID
            )
            begin
                ;throw 52000, 'Reservation does not exist.', 1
            end

        if not exists
            (
                select * from Tables
                where Table_ID = @Table_ID
            )
            begin
                ;throw 52000, 'Table does not exist or is not active.', 1
            end
        DECLARE @StartDate DATETIME;
        DECLARE @EndDate DATETIME;
        SELECT @StartDate=R2.StartDate, @EndDate=R2.EndDate FROM Reservations R2
    WHERE R2.Reservation_ID=@Reservation_ID;
        if exists
            (
                select * from Reserved_Tables rt
                JOIN Reservations R2 on rt.Reservation_ID = R2.Reservation_ID
                where ((R2.StartDate <= @StartDate
                AND R2.EndDate >= @StartDate)
                OR
                (R2.StartDate <= @EndDate
                AND R2.EndDate >= @EndDate)
                OR
                (R2.StartDate >= @StartDate
                AND R2.EndDate <= @EndDate))
                AND R2.Reservation_ID <> @Reservation_ID
                AND rt.Table_ID = @Table_ID
            )
            begin
                ;throw 52000, 'Table is not available at that time.', 1
            end

        insert into Reserved_Tables
        (
            Reservation_ID,
            Table_ID
        )
        values
        (
            @Reservation_ID,
            @Table_ID
        )
    end try
    begin catch
        declare @errorMsg nvarchar(2048)
        = 'Cannot add table to reservation. Error message: '
        + ERROR_MESSAGE();
    ;throw 52000, @errorMsg, 1
    end catch
end
```

cancelReservation

```
CREATE PROCEDURE [dbo].[cancelReservation]
@ReservationID AS int
AS
```

```
    BEGIN
        DECLARE @Status int;
        SET @Status = [dbo].getActualStatus(@ReservationID)
        IF (@Status IN(0,2))
            UPDATE Reservations
            SET Status = 5
            WHERE Reservation_ID = @ReservationID
        ELSE
            IF (@Status IN (1,3))
                UPDATE Reservations
                SET Status = 4
                WHERE Reservation_ID = @ReservationID
    END
```

ChangeReservationStatus

```
CREATE PROCEDURE [dbo].[ChangeReservationStatus]
@Reservation_ID int,
@Status tinyint
as
begin
    begin try
        if not exists
            (
                select * from Reservations
                where Reservation_ID = @Reservation_ID
            )
            begin
                ;throw 52000, 'Reservation does not exist.', 1
            end

            update Reservations
                set Status = @Status
                where Reservation_ID = @Reservation_ID
    end try
    begin catch
        declare @errorMsg nvarchar(2048)
        = 'Cannot change reservation status. Error message: '
        + ERROR_MESSAGE();
    ;throw 52000, @errorMsg, 1
    end catch
end
```

createCompanyClient

```
CREATE PROCEDURE [dbo].[createCompanyClient]
@Email AS varchar(35) = NULL,
@Phone AS int = NULL,
@Address AS varchar(35),
@PostalCode AS int,
@City AS varchar(35),

@CompanyName AS varchar(35),
@NIP AS bigint = NULL,
@REGON AS varchar(14) = NULL,
@KRS AS bigint = NULL
AS
    BEGIN TRANSACTION

        BEGIN TRY
            DECLARE @ClientID int;
            SET @ClientID = [dbo].getHighestClientID()
            EXEC [dbo].createNewClient @Email, @Phone, @Address, @PostalCode, @City
```

```
19.
20.
21.             IF EXISTS
22.                 (
23.                 select * from Company
24.                 where @NIP = NIP
25.                 )
26.                 AND @NIP != NULL
27.             BEGIN
28.                 ROLLBACK TRANSACTION
29.                 ;THROW 52000, 'Nip already in database.',1
30.             END
31.
32.             IF EXISTS
33.                 (
34.                 select * from Company
35.                 where @REGON = REGON
36.                 )
37.                 AND @REGON != NULL
38.             BEGIN
39.                 ROLLBACK TRANSACTION
40.                 ;THROW 52000, 'regon already in database.',1
41.             END
42.
43.             IF EXISTS
44.                 (
45.                 select * from Company
46.                 where @KRS = KRS
47.                 )
48.                 AND @KRS != NULL
49.             BEGIN
50.                 ROLLBACK TRANSACTION
51.                 ;THROW 52000, 'krs already in database.',1
52.             END
53.
54.             INSERT INTO Company(Client_ID, Company_Name, NIP, REGON, KRS)
55.             VALUES (@ClientID + 1, @CompanyName, @NIP, @REGON, @KRS)
56.
57.         END TRY
58.         BEGIN CATCH
59.         DECLARE @errorMsg nvarchar(2048)
60.                 = 'Cannot add company client. Error: ' + ERROR_MESSAGE();
61.             ROLLBACK TRANSACTION;
62.             THROW 52000, @errorMSg, 1;
63.         END CATCH
64.
65.     COMMIT TRANSACTION
```

createIndividualClient

```
1.  CREATE PROCEDURE [dbo].[createIndividualClient]
2.  @Email AS varchar(35) = NULL,
3.  @Phone AS int = NULL,
4.  @Address AS varchar(35),
5.  @PostalCode AS int,
6.  @City AS varchar(35),
7.
8.  @FirstName AS varchar(35),
9.  @LastName AS varchar(35)
10. AS
11.     BEGIN TRANSACTION
12.         BEGIN TRY
13.             DECLARE @ClientID int;
14.             SET @ClientID = [dbo].getHighestClientID();
```

```
            EXEC [dbo].[createNewClient] @Email, @Phone, @Address, @PostalCode, @City

            INSERT INTO Individual(Client_ID, FirstName, LastName, OrdersCount)
            VALUES (@ClientID + 1, @FirstName, @LastName, 0)

            INSERT INTO Client_Discounts(Client_ID, Discount1, Discount2, Discount2LastUsed)
            VALUES (@ClientID+1, NULL, NULL, 0)
        END TRY

        BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                = 'Cannot add individual client. Error: ' + ERROR_MESSAGE();
            ROLLBACK TRANSACTION;
            THROW 52000, @errorMSg, 1;
        END CATCH
    COMMIT TRANSACTION;
```

CreateInvoice

```
CREATE PROCEDURE [dbo].[CreateInvoice]
@CreationDate as date,
@Invoice_ID as varchar(20),
@PersonalData as varchar(50),
@Address as varchar(35),
@City as varchar(35),
@PostalCode as int
as
    begin transaction

        begin try
            insert into Invoices
            (
            Invoice_ID,
            Date,
            PersonalData,
            Address,
            City,
            PostalCode
            )
            values
            (
            @Invoice_ID,
            @CreationDate,
            @PersonalData,
            @Address,
            @City,
            @PostalCode
            )
        end try

        begin catch
            declare @errorMsg nvarchar(2048) = 'Cannot create invoice. Error message: '+ERROR_MESSAGE();
            rollback transaction;
            ;throw 52000, @errorMsg, 1;
        end catch

    commit transaction
```

createNewClient

```
CREATE PROCEDURE [dbo].[createNewClient]
@Email AS varchar(35) = NULL,
@Phone AS int = NULL,
@Address AS varchar(35),
@PostalCode AS int,
@City AS varchar(35)
AS
    BEGIN TRANSACTION
        BEGIN TRY
            IF EXISTS
                (
                SELECT * FROM Client
                WHERE Email = @Email
                )
            BEGIN
                ROLLBACK TRANSACTION;
                ;THROW 52000, 'Email already exists.',1
            END

            IF EXISTS
                (
                SELECT * FROM Client
                WHERE Phone = @Phone
                )
            BEGIN
                ROLLBACK TRANSACTION;
                ;THROW 52000, 'Phone already exists.',1
            END


            DECLARE @ClientID int;
            SET @ClientID = [dbo].getHighestClientID();
            INSERT INTO Client(Client_ID, Email, Phone, Address, PostalCode, City)
            VALUES(@ClientID+1, @Email, @Phone, @Address, @PostalCode, @City);

        END TRY
        BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                = 'Cannot add client. Error: ' + ERROR_MESSAGE();
            ROLLBACK TRANSACTION;
            THROW 52000, @errorMSg, 1;
        END CATCH
    COMMIT TRANSACTION
```

createReservation

```
CREATE PROCEDURE [dbo].[createReservation]
@ClientID AS int,
@StartDate AS datetime,
@EndDate AS datetime,
@DiscountType AS tinyint,
@Status AS tinyint
AS
    BEGIN TRANSACTION

        BEGIN TRY
            DECLARE @ReservationID int;
            SET @ReservationID = [dbo].getHighestReservationID()
            INSERT INTO Reservations(Reservation_ID, Client_ID, StartDate, EndDate,
DiscountType, Status)
```

```
            VALUES (@ReservationID + 1, @ClientID, @StartDate, @EndDate,
    @DiscountType, @Status)
        END TRY

        BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                    = 'Cannot add reservation . Error: ' + ERROR_MESSAGE();
            ROLLBACK TRANSACTION;
            THROW 52000, @errorMSg, 1;

        END CATCH

    COMMIT TRANSACTION
```

editDatesOfMenuSet

```
CREATE PROCEDURE [dbo].[editDatesOfMenuSet]
@MenuID AS int,
@StartDate AS date,
@EndDate AS date
AS
    BEGIN
        IF (@MenuID <= [dbo].getHighestMenuID() AND @StartDate < @EndDate)
            UPDATE Menu
            SET StartDate = @StartDate, EndDate = @EndDate
            WHERE Menu_ID = @MenuID
    END
```

EditProductAmountInReservation

```
CREATE PROCEDURE [dbo].[EditProductAmountInReservation]
@Reservation_ID int,
@Product_ID int,
@Amount int
as
begin
    begin try
        if not exists
            (
                select * from Reservations
                where Reservation_ID = @Reservation_ID
            )
            begin
                ;throw 52000, 'Reservation does not exist.', 1
            end
        if not exists
            (
                select * from Orders o
                where o.Reservation_ID = @Reservation_ID
            )
            begin
                ;throw 52000, 'Order does not exist.', 1
            end
            update Orders
                set Amount = @Amount
                where Reservation_ID = @Reservation_ID and Product_ID = @Product_ID
    end try
    begin catch
        declare @errorMsg nvarchar(2048)
        = 'Cannot change order amount. Error message: '
        + ERROR_MESSAGE();
    ;throw 52000, @errorMsg, 1
    end catch
```

```
34. end
```

endReservation

```
1.  CREATE PROCEDURE [dbo].[endReservation]
2.  @ReservationID AS int
3.  AS
4.      BEGIN
5.      BEGIN TRY
6.          IF NOT EXISTS
7.              (
8.              SELECT * FROM Reservations
9.              WHERE @ReservationID = Reservation_ID
10.             )
11.         BEGiN
12.             ;THROW 52000, 'No reservation with given reservation id',1
13.         END
14.
15.        UPDATE Reservations
16.        SET Status = 6
17.        WHERE Reservation_ID = @ReservationID
18.     END TRY
19.
20.     BEGIN CATCH
21.     DECLARE @errorMsg nvarchar(2048)
22.         ='Cannot end reservation. Error: ' + ERROR_MESSAGE();
23.         THROW 52000, @errorMsg, 1;
24.     END CATCH
25.
26. END
```

OrdersForTodaySortedByDate

```
1.  CREATE PROCEDURE [dbo].[OrdersForTodaySortedByDate] as
2.      select p.Name, o.Amount, r.StartDate
3.      from Orders o
4.      join Products p on p.Product_ID = o.Product_ID
5.      left join Reservations r on r.Reservation_ID = o.Reservation_ID
6.      where r.Status in(0,2)
7.      and CAST( r.StartDate AS DATE) = CAST( GETDATE() AS DATE)
8.      order by r.StartDate asc
```

SelectSumOfAllProductsBetween

```
1.  CREATE PROCEDURE [dbo].[SelectSumOfAllProductsBetween] @StartDate Date,
2.  @EndDate Date
3.  AS
4.      SELECT P.Name, SUM(O.Amount) AS ilosc, SUM(O.Amount * PoM.Price)
5.      FROM Products P
6.      JOIN Product_On_Menu PoM ON Pom.Product_ID = P.Product_ID
7.      JOIN Orders O ON O.Product_ID = PoM.Product_ID AND O.Menu_ID = PoM.Menu_ID
8.      JOIN Reservations R ON R.Reservation_ID = O.Reservation_ID
9.      WHERE CAST(R.StartDate AS DATE) >= @StartDate
10.     AND CAST(R.EndDate AS DATE) <= @EndDate
11.     GROUP BY P.Name
```

ShowAllFreeTablesBetweenDates

```
1.  CREATE PROCEDURE [dbo].[ShowAllFreeTablesBetweenDates]
```

```
@InputStartDate datetime,
@InputEndDate datetime
as
 select t.Table_ID, t.NumberOfChairs
 from Tables t
 join Reserved_Tables rt on rt.Table_ID = t.Table_ID
 join Reservations r on r.Reservation_ID = rt.Reservation_ID
 where t.Active=1 and r.EndDate <= @InputStartDate and r.StartDate >=
@InputEndDate;
```

Funkcje

generateReportMoney

```
CREATE FUNCTION [dbo].[generateReportMoney](
@StartDate AS date
)
RETURNS table
AS
    return(
            SELECT SUM(O.Amount*POM.Price) income, AVG(O.Amount*POM.Price)
avg_order_value, COUNT(R.Reservation_ID) order_count
            FROM Reservations R
            JOIN Orders O on O.Reservation_ID = R.Reservation_ID
            JOIN Product_On_Menu POM on POM.Product_ID = O.Product_ID and
POM.Menu_ID = O.Menu_ID
            WHERE R.Status = 6 and CAST(R.EndDate as date) >= @StartDate and
CAST(R.EndDate as date) <= dateadd(month, 1, @StartDate))
```

generateReportProducts

```
CREATE FUNCTION [dbo].[generateReportProducts](
@StartDate AS date
)
RETURNS table
AS
    return(
            SELECT P.Name id_of_product, SUM(O.Amount) as units_sold
            FROM Reservations R
            JOIN Orders O on O.Reservation_ID = R.Reservation_ID
            JOIN Product_On_Menu POM on POM.Product_ID = O.Product_ID and
POM.Menu_ID = O.Menu_ID
            JOIN Products P on P.Product_ID = POM.Product_ID
            WHERE R.Status = 6 and CAST(R.EndDate as date) >= @StartDate and
CAST(R.EndDate as date) <= dateadd(month, 1, @StartDate)
            GROUP BY P.Product_ID, P.Name
        )
```

getFreeTablesBetweenDates

```
CREATE FUNCTION [dbo].[getFreeTablesBetweenDates](
@StartDate AS datetime,
@EndDate AS datetime
)
RETURNS TABLE
AS
    return(
            SELECT T.Table_ID
```

```
            FROM Tables T
            WHERE T.Active = 1

            EXCEPT (
            SELECT DISTINCT T.Table_ID
            FROM Tables T
            JOIN Reserved_Tables RT on T.Table_ID = RT.Table_ID
            JOIN Reservations R2 on RT.Reservation_ID = R2.Reservation_ID
            WHERE (R2.StartDate<= @StartDate AND R2.EndDate >= @StartDate)
               OR (R2.StartDate <= @EndDate AND R2.EndDate >= @EndDate)
            )
    )
```

menuUntil

```
CREATE FUNCTION [dbo].[menuUntil](@EndDate DATE)
RETURNS TABLE
AS
    return(SELECT c.Product_ID, c.Menu_ID, c.StartDate, c.EndDate
    FROM currentMenu c
    WHERE c.StartDate <= @EndDate)
```

CountOfAllFreeTablesBetweenDates

```
CREATE FUNCTION [dbo].[CountOfAllFreeTablesBetweenDates]
(@InputStartDate datetime,
 @InputEndDate datetime)
returns tinyint
as
begin
    return
         (select count(*)
         from Tables t
         join Reserved_Tables rt on rt.Table_ID = t.Table_ID
         join Reservations r on r.Reservation_ID = rt.Reservation_ID
         where t.Active=1 and r.EndDate <= @InputStartDate and r.StartDate >=
   @InputEndDate);
 end
```

getActualStatus

```
CREATE FUNCTION [dbo].[getActualStatus](
    @ReservationID AS int
)
RETURNS int
AS
    BEGIN
        return (
            SELECT R.Status
            FROM Reservations R
            WHERE R.Reservation_ID = @ReservationID
            )
    END
```

getAmountOfFreeTables

```
CREATE FUNCTION [dbo].[getAmountOfFreeTables](
    @StartDate AS datetime,
    @EndDate AS datetime,
```

```
    @MinimalSize AS int
)
RETURNS int
AS
    BEGIN
        return(
            SELECT COUNT(*)
            FROM [dbo].getFreeTablesBetweenDates(@StartDate, @EndDate) FT
            JOIN Tables T ON T.Table_ID = FT.Table_ID
            WHERE T.NumberOfChairs >= @MinimalSize
            )
    END
```

getFullReservationCost

```
CREATE FUNCTION [dbo].[getFullReservationCost](@ReservationID int)
RETURNS INT
AS
BEGIN
    return(SELECT SUM(o.Amount * POM.Price) FROM Orders o
        JOIN Product_On_Menu POM ON POM.Menu_ID = o.Menu_ID AND POM.Product_ID =
    o.Product_ID
        WHERE o.Reservation_ID = @ReservationID)
END
```

getHighestClientID

```
CREATE FUNCTION [dbo].[getHighestClientID] ()
RETURNS int
AS
BEGIN
   return (
       SELECT TOP 1 C.Client_ID
        FROM Client C
        ORDER BY C.Client_ID DESC
       )
END
```

getHighestEmployeeID

```
CREATE FUNCTION [dbo].[getHighestEmployeeID]()
RETURNS int
AS
BEGIN
   return (
        SELECT TOP 1 N.Guest_ID
        FROM Names N
        ORDER BY N.Guest_ID DESC
       )
END
```

getHighestIndividual

```
CREATE FUNCTION [dbo].[getHighestIndividual] ()
RETURNS int
AS
BEGIN
   return (
        SELECT TOP 1 C.Client_ID
```

```
7.          FROM Individual C
8.          ORDER BY C.Client_ID DESC
9.         )
10. END
```

getHighestMenuID

```
1.  CREATE FUNCTION [dbo].[getHighestMenuID]()
2.  RETURNS int
3.  AS
4.      BEGIN
5.          return (
6.              SELECT TOP 1 PoM.Menu_ID
7.              FROM Product_On_Menu PoM
8.              ORDER BY PoM.Menu_ID DESC
9.              )
10.     END
```

getHighestReservationID

```
1.  CREATE FUNCTION [dbo].[getHighestReservationID]()
2.  RETURNS int
3.  AS
4.  BEGIN
5.     return (
6.          SELECT TOP 1 R.Reservation_ID
7.          FROM Reservations R
8.          ORDER BY R.Reservation_ID DESC
9.         )
10. END
```

Triggery

TrgCancelReservation

Trigger inkrementujący licznik zamówień dla rezerwacji spełniających wymagania dla pierwszej zniżki.

```
1.  CREATE TRIGGER [dbo].[trgCancelReservation] ON [dbo].[Reservations]
2.  AFTER UPDATE
3.  AS
4.  BEGIN
5.      DECLARE @ClientID int;
6.      DECLARE @ReservationID int;
7.      DECLARE @Status tinyint;
8.
9.      IF ((SELECT COUNT(*) FROM inserted) = 0)
10.         THROW 51000, 'empty_Update', 1;
11.
12.     IF ((SELECT COUNT(*) FROM inserted) > 0)
13.         BEGIN
14.             SELECT @ClientID = i.Client_ID, @ReservationID = i.Reservation_ID,
    @Status = i.Status FROM inserted i;
15.             IF((SELECT COUNT(*) FROM [dbo].[Individual]) > 0
16.                     AND @Status = 6
```

```
                    AND [dbo].getFullReservationCost(@ReservationID) > (SELECT TOP
    1 d.Value FROM [dbo].[Discount_Parameters] d))
                UPDATE Individual
                SET OrdersCount += 1
                WHERE Client_ID = @ClientID;
        END
END
```

Indeksy

Reservation_Status_Index

```
CREATE INDEX Reservation_Status_Index on Reservations (Status);
```

Reserved_Tables_Table

```
CREATE INDEX Reserved_Tables_Table on Reserved_Tables(Table_ID);
```

Reserved_Tables_Reservation

```
CREATE INDEX Reserved_Tables_Reservation on Reserved_Tables(Reservation_ID);
```

Role

Manager

```
CREATE ROLE manager

GRANT EXECUTE to manager
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER to manager
```

Customer

```
CREATE ROLE customer

GRANT EXECUTE ON [dbo].[AddPersonToReservation] to customer
GRANT EXECUTE ON [dbo].[cancelReservation]to customer
GRANT EXECUTE ON [dbo].[createCompanyClient]to customer
GRANT EXECUTE ON [dbo].[createIndividualClient]to customer
GRANT SELECT ON dbo.CurrentMenu to customer
```

Staff

```
CREATE ROLE staff

GRANT EXECUTE ON [dbo].[addDishToProducts] to Staff
GRANT EXECUTE ON [dbo].[AddOrderToReservation] to staff
GRANT EXECUTE ON [dbo].[AddPersonToReservation] to staff
GRANT EXECUTE ON [dbo].[addProductToMenu]to staff
GRANT EXECUTE ON [dbo].[AddReservationToInvoice]to staff
GRANT EXECUTE ON [dbo].[AddTableToReservation]to staff
```

```
GRANT EXECUTE ON [dbo].[cancelReservation]to staff
GRANT EXECUTE ON [dbo].[ChangeReservationStatus]to staff
GRANT EXECUTE ON [dbo].[createCompanyClient]to staff
GRANT EXECUTE ON [dbo].[createIndividualClient]to staff
GRANT EXECUTE ON [dbo].[CreateInvoice]to staff
GRANT EXECUTE ON [dbo].[createNewClient]to staff
GRANT EXECUTE ON [dbo].[createReservation]to staff
GRANT EXECUTE ON [dbo].[editDatesOfMenuSet]to staff
GRANT EXECUTE ON [dbo].[EditProductAmountInReservation]to staff
GRANT EXECUTE ON [dbo].[endReservation]to staff
GRANT EXECUTE ON [dbo].[OrdersForTodaySortedByDate]to staff
GRANT EXECUTE ON [dbo].[SelectSumOfAllProductsBetween]to staff
GRANT EXECUTE ON [dbo].[ShowAllFreeTablesBetweenDates]to staff


GRANT SELECT ON dbo.CurrentMenu to staff
GRANT SELECT ON dbo.OrdersForToday to staff

GRANT SELECT ON [dbo].[getFreeTablesBetweenDates] to staff
GRANT SELECT ON [dbo].[menuUntil] to staff

GRANT EXECUTE ON [dbo].[CountOfAllFreeTablesBetweenDates] to staff
GRANT EXECUTE ON [dbo].[getActualStatus] to staff
GRANT EXECUTE ON [dbo].[getAmountOfFreeTables] to staff
GRANT EXECUTE ON [dbo].[getFullReservationCost] to staff
```