Podstawy baz danych

Projekt:
System wspomagania firmy świadczącej usługi gastronomiczne

Jakub Szymczak, Szymon Budziak, Łukasz Wala

1. Propozycja funkcji realizowanych przez system

1.1 Menadżer restauracji:

1) wgląd i możliwość modyfikacji oferty/menu,
2) wgląd i możliwość modyfikacji listy pracowników,
3) ustalanie rabatów,
4) dostęp do raportów, faktur i statystyk generowanych przez system,
5) możliwość dodawania, usuwania oraz realizacji rezerwacji.

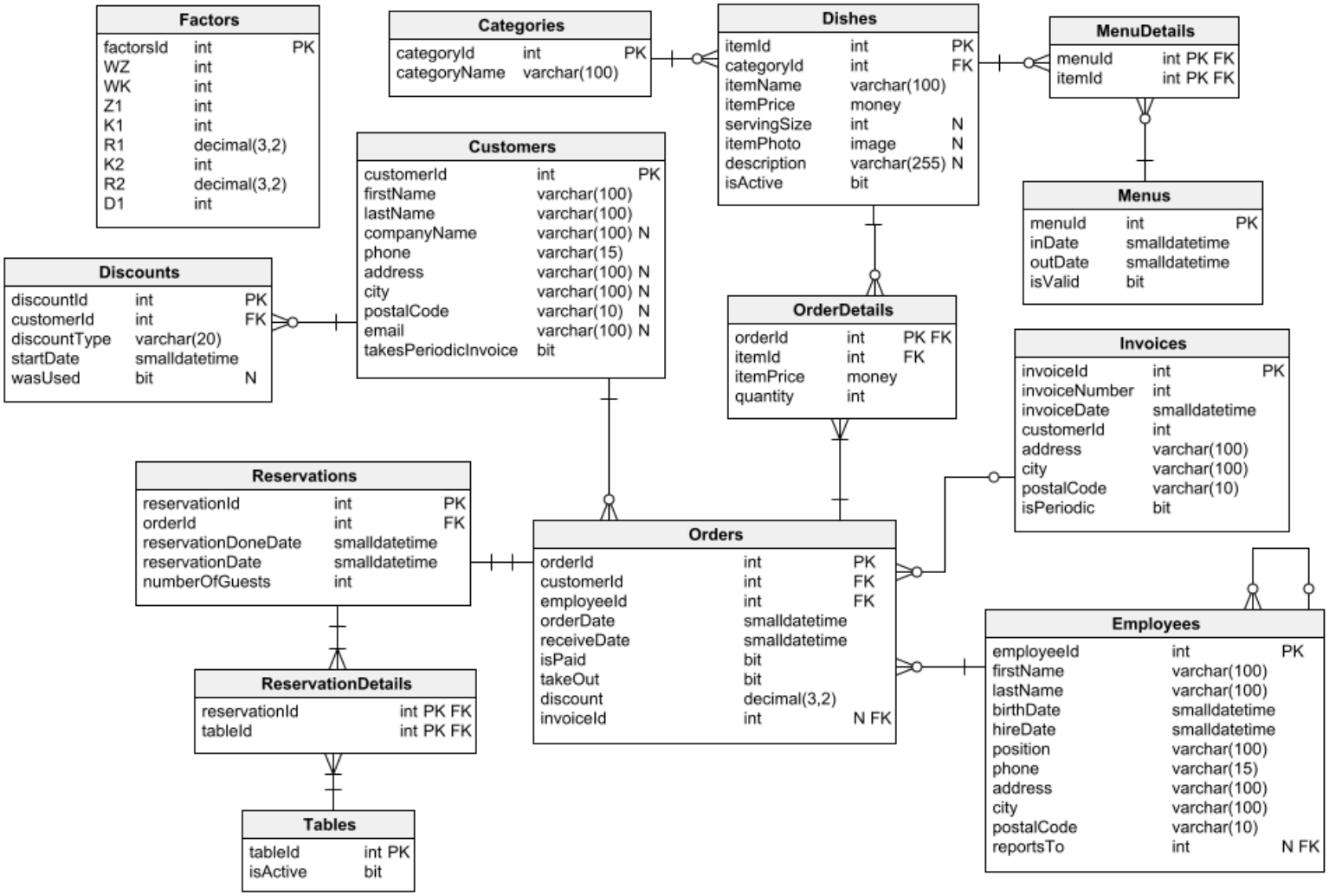
1.2 Kelner/ kasjer:

1) możliwość dodawania, usuwania oraz realizacji rezerwacji,
2) możliwość dodawania zamówień,
3) wgląd do menu.

1.3 Klient:

1) wgląd do menu,
2) możliwość rezerwacji stolików i zamówienia jedzenia na wynos.

2. Schemat bazy danych



3. Tabele

3.1 Tabela Categories

Tabela zawiera informacje na temat kategorii, do których należą elementy menu.

categoryId - identyfikator danej kategorii, autoinkrementowany,
categoryName - nazwa danej kategorii.

```
CREATE TABLE Categories (
        categoryId int  NOT NULL IDENTITY (1,1),
        categoryName varchar(100)  NOT NULL,
        CONSTRAINT Categories_pk PRIMARY KEY  (categoryId)
);
```

3.2 Tabela Tables

Tabela przechowująca informacje o stolikach.

tableId - identyfikator stolika,
isActive - wartość identyfikująca czy dany stolik jest w użytku.

```
CREATE TABLE Tables (
        tableId int  NOT NULL IDENTITY(1,1),
        isActive bit  NOT NULL DEFAULT 1,
        CONSTRAINT Tables_pk PRIMARY KEY  (tableId)
);
```

3.3 Tabela Customers

Tabela zawiera informacje na temat klientów firmy, pozwala na stwierdzenie, czy klient jest indywidualny, czy reprezentuje firmę, jego informacje kontaktowe.

customerId - identyfikator klienta, autoinkremetowany,
firstName, lastName - imię i nazwisko klienta,
companyName - nazwa firmy, jeżeli klient ją reprezentuje, NULL w innym przypadku,
phone, email - numer telefonu, email i fax klienta,
address, city, postalCode - dane adresowe klienta,
takesPeriodicInvoice - wartość identyfikująca czy klient bierze fakturę okresową.

```
CREATE TABLE Customers (
        customerId int  NOT NULL IDENTITY (1,1),
        firstName varchar(100)  NOT NULL,
        lastName varchar(100)  NOT NULL,
        companyName varchar(100)  NULL,
        phone varchar(15)  NOT NULL,
        address varchar(100)  NULL,
        city varchar(100)  NULL,
        postalCode varchar(10)  NULL,
        email varchar(100)  NULL CHECK(email IS NULL OR email LIKE '%_@%_.%_'),
        takesPeriodicInvoice bit  NOT NULL DEFAULT 0,
        CONSTRAINT Customers_pk PRIMARY KEY  (customerId)
);
```

3.4 Tabela Employees

Tabela zawiera informacje na temat pracowników firmy, ich dane kontaktowe i adresowe oraz to, komu podlegają.

employeeId - identyfikator pracownika,
firstName, lastName - imię i nazwisko pracownika,
birthDate - data urodzenia pracownika,
hireDate - data przyjęcia pracownika do pracy,
position - stanowisko pracownika w firmie,
phone - numer telefonu pracownika,
address, city, postalCode - dane adresowe pracownika,
reportsTo - pracownik, który nadzoruje danego pracownika.

```
CREATE TABLE Employees (
        employeeId int  NOT NULL IDENTITY(1,1),
        firstName varchar(100)  NOT NULL,
        lastName varchar(100)  NOT NULL,
        birthDate smalldatetime  NOT NULL CHECK(birthDate <= GETDATE()),
        hireDate smalldatetime  NOT NULL CHECK(hireDate <= GETDATE()),
        position varchar(100)  NOT NULL,
        phone varchar(15)  NOT NULL,
        address varchar(100)  NOT NULL,
        city varchar(100)  NOT NULL,
        postalCode varchar(10)  NOT NULL,
        reportsTo int NULL,
        CONSTRAINT Employees_pk PRIMARY KEY  (employeeId)
);
-- Reference: Employees_Employees (table: Employees)
ALTER TABLE Employees ADD CONSTRAINT Employees_Employees
        FOREIGN KEY (reportsTo)
        REFERENCES Employees (employeeId);
```

3.5 Tabela Dishes

Tabela przechowuje dane o produktach które mogą być zawarte w menu.

itemId - identyfikator danego produktu,
categoryId - identyfikator kategorii, do której należy produkt,
itemName - nazwa danego produktu,
itemPrice - cena danego produktu,
servingSize - wielkość produktu (waga/objętość),

itemPhoto - zdjęcie danego produktu,
description - opis danego produktu,
isActive - danie może zostać użyte w nowo utworzonym menu.

```
CREATE TABLE Dishes (
        itemId int  NOT NULL IDENTITY(1,1),
        categoryId int  NOT NULL,
        itemName varchar(100)  NOT NULL,
        itemPrice money  NOT NULL CHECK (itemPrice > 0),
        servingSize int  NULL,
        itemPhoto image  NULL,
        description varchar(255)  NULL,
    isActive bit  NOT NULL DEFAULT 0,
        CONSTRAINT Dishes_pk PRIMARY KEY  (itemId)
);

-- Reference: Menu_Categories (table: Dishes)
ALTER TABLE Dishes ADD CONSTRAINT Menu_Categories
        FOREIGN KEY (categoryId)
        REFERENCES Categories (categoryId);
```

3.6 Tabela Menus

Tabela przechowuje informacje o tym w jakich przedziałach czasowych obowiązywało dane menu.

menuId - identyfikator menu,
inDate - data, od której dany produkt jest w obecnym menu restauracji,
outDate - data, do której dany produkt jest w obecnym menu restauracji,
isValid - menu spełnia warunki poprawności menu.

```
-- Table: Menus
CREATE TABLE Menus (
        menuId int  NOT NULL IDENTITY(1,1),
        inDate smalldatetime  NOT NULL CHECK(inDate > GETDATE()),
        outDate smalldatetime  NOT NULL CHECK(outDate > GETDATE()),
    isValid bit  NOT NULL DEFAULT 0,
        CONSTRAINT Menus_pk PRIMARY KEY  (menuId)
);
```

3.7 Tabela MenuDetails

Tabela przechowuje informacje jakie dania zawierają dane menu.

menuId - identyfikator menu,
itemId - identyfikator danego produktu.

```
CREATE TABLE MenuDetails (
        menuId int  NOT NULL,
        itemId int  NOT NULL,
        CONSTRAINT MenuDetails_pk PRIMARY KEY  (menuId,itemId)
);

-- Reference: Dishes_MenuDetails (table: MenuDetails)
ALTER TABLE MenuDetails ADD CONSTRAINT Dishes_MenuDetails
        FOREIGN KEY (itemId)
```

3.8 Tabela Orders

Tabela przechowuje informacje o zamówieniach wykonanych przez klientów. Dzięki niej jesteśmy w stanie dowiedzieć się o szczegółach danego zamówienia, które zostało złożone przez danego klienta, np. jaki pracownik je obsłużył, data zamówienia, czy odbiera zamówienie na wynos oraz czy odbiera fakturę lub paragon.

orderId - identyfikator zamówienia, wartość autoinkrementowana,
customerId - identyfikator klienta,
employeeId - identyfikator pracownika,
orderDate - data złożenia zamówienia,
receiveDate - data odebrania zamówienia,
isPaid - wartość identyfikująca czy zamówienie zostało złożone,
takeOut - wartość identyfikująca czy klient odbiera zamówienie na wynos,
discount - procent jaki został nadany klientowi na całe zamówienie,
invoiceId - identyfikator faktury.

```
CREATE TABLE Orders (
        orderId int  NOT NULL IDENTITY(1,1),
        customerId int  NOT NULL,
        employeeId int  NOT NULL,
        orderDate smalldatetime  NOT NULL CHECK(orderDate <= GETDATE()),
        receiveDate smalldatetime  NOT NULL CHECK(receiveDate >= GETDATE()),
        isPaid bit  NOT NULL,
        takeOut bit  NOT NULL,
        discount decimal(3,2)  NOT NULL CHECK(discount BETWEEN 0.00 AND 1.00),
        invoiceId int  NULL,
        CONSTRAINT Orders_pk PRIMARY KEY  (orderId)
);
```

```
-- Reference: Orders_Customers (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Customers
        FOREIGN KEY (customerId)
        REFERENCES Customers (customerId);

-- Reference: Orders_Invoices (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Invoices
        FOREIGN KEY (invoiceId)
        REFERENCES Invoices (invoiceId);

-- Reference: Orders_Employees (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Employees
        FOREIGN KEY (employeeId)
        REFERENCES Employees (employeeId);
```

3.9 Tabela OrderDetails

Tabela przechowująca informacje o szczegółach danego zamówienia.

orderId - identyfikator zamówienia,
itemId - identyfikator produktu,
itemPrice - cena produktu,
quantity - ilość danego produktu.

```
CREATE TABLE OrderDetails (
        orderId int  NOT NULL,
        itemId int  NOT NULL,
        itemPrice money  NOT NULL CHECK(itemPrice > 0),
        quantity int  NOT NULL CHECK(quantity > 0),
        CONSTRAINT OrderDetails_pk PRIMARY KEY  (orderId)
);

-- Reference: Dishes_OrderDetails (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT Dishes_OrderDetails
        FOREIGN KEY (itemId)
        REFERENCES Dishes (itemId);

-- Reference: OrderDetails_Orders (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
        FOREIGN KEY (orderId)
        REFERENCES Orders (orderId);
```

3.10 Tabela Reservations

Tabela przechowuje informacje dotyczące rezerwacji dokonanych przez klientów.

reservationId - identyfikator danej rezerwacji,
orderId - identyfikator zamówienia,
reservationDoneDate - data, w której klient dokonał rezerwacji,
reservationDate - data, na którą klient dokonał rezerwacji,
numberOfGuests - ilość gości wchodzących na daną rezerwację.

```
CREATE TABLE Reservations (
        reservationId int  NOT NULL IDENTITY(1,1),
        orderId int  NOT NULL,
        reservationDoneDate smalldatetime  NOT NULL CHECK(reservationDoneDate <= GETDATE()),
        reservationDate smalldatetime  NOT NULL CHECK(reservationDate >= GETDATE()),
        numberOfGuests int  NOT NULL CHECK (numberOfGuests > 0),
        CONSTRAINT Reservations_pk PRIMARY KEY  (reservationId)
);

-- Reference: Orders_Reservations (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Orders_Reservations
        FOREIGN KEY (orderId)
        REFERENCES Orders (orderId);
```

3.11 Tabela ReservationDetails

Tabela przechowuje detale rezerwacji zawartych w tabeli Reservations.

reservationId - identyfikator danej rezerwacji,
tableId - numer identyfikacyjny stołu, na który dokonana jest rezerwacja.

```
CREATE TABLE ReservationDetails (
        reservationId int  NOT NULL,
        tableId int  NOT NULL,
        CONSTRAINT ReservationDetails_pk PRIMARY KEY  (reservationId,tableId)
);

-- Reference: ReservationDetails_Reservations (table: ReservationDetails)
ALTER TABLE ReservationDetails ADD CONSTRAINT ReservationDetails_Reservations
        FOREIGN KEY (reservationId)
        REFERENCES Reservations (reservationId);

-- Reference: ReservationDetails_Tables (table: ReservationDetails)
ALTER TABLE ReservationDetails ADD CONSTRAINT ReservationDetails_Tables
        FOREIGN KEY (tableId)
        REFERENCES Tables (tableId);
```

3.12 Tabela Factors

Tabela przechowuje stałe używane przy wyliczaniu rabatów oraz walidacji możliwości złożenia zamówienia, zawiera jeden wiersz.
factorsId - identyfikator stałych,
WZ - minimalna wartość zamówienia składanego przez internet,
WK - minimalna ilość zamówień potrzebna do złożenia zamówienia przez internet,
Z1 - liczba zamówień potrzebna do otrzymania dożywotniego rabatu,
K1 - minimalna kwota zamówienia, ażeby wliczał się do otrzymania dożywotniego rabatu,
R1 - wartość procentowa rabatu dożywotniego,
K2 - łączna kwota potrzebna do otrzymania jednorazowego rabatu,
R2 - wartość procentowa rabatu jednorazowego,
D1 - czas na zrealizowanie rabatu jednorazowego.

```
CREATE TABLE Factors (
        factorsId int  NOT NULL,
        WZ int  NOT NULL,
        WK int  NOT NULL,
        Z1 int  NOT NULL,
        K1 int  NOT NULL,
        R1 decimal(3,2)  NOT NULL CHECK(R1 BETWEEN 0.00 AND 1.00),
        K2 int  NOT NULL,
        R2 decimal(3,2)  NOT NULL CHECK(R2 BETWEEN 0.00 AND 1.00),
        D1 int  NOT NULL,
        CONSTRAINT Factors_pk PRIMARY KEY  (factorsId)
);
```

3.13 Tabela Discounts

Tabela przechowuje dane o rabatach uzyskanych przez danego klienta.
discountId - identyfikator rabatu,
customerId - identyfikator klienta,
discountType - type rabatu, może być ‘lifetime’, czyli dożywotni za zrealizowanie pewnej liczby zamówień albo ‘temporary’, czyli jednorazowy rabat za zrealizowanie zamówienia za pewną kwotę,
startDate - data od której rabat zaczyna obowiązywać,
wasUsed - czy rabat został użyty.

```
CREATE TABLE Discounts (
        discountId int  NOT NULL IDENTITY (1,1),
        customerId int  NOT NULL,
        discountType varchar(20)  NOT NULL CHECK(discountType IN ('lifetime', 'temporary')),
```

```
        startDate smalldatetime  NOT NULL CHECK(startDate <= GETDATE()),
    wasUsed bit  NULL,
        CONSTRAINT Discounts_pk PRIMARY KEY  (discountId)
);

-- Reference: Customers_Discounts (table: Discounts)
ALTER TABLE Discounts ADD CONSTRAINT Customers_Discounts
        FOREIGN KEY (customerId)
        REFERENCES Customers (customerId);
```

3.14 Tabela Invoices

Tabela przechowuje dane o fakturach wystawianych za zamówienia.
invoiceId - identyfikator faktury,
invoiceNumber - numer faktury,
invoiceDate - data wystawienia faktury,
customerId - identyfikator klienta, na którego została wystawiona faktura,
address, city, postalCode - dane adresowe klienta (na wypadek np. zmiany siedziby),
isPeriodic - czy faktura jest fakturą comiesięczną.

```
CREATE TABLE Invoices (
        invoiceId int  NOT NULL IDENTITY(1,1),
        invoiceNumber int  NOT NULL UNIQUE,
        invoiceDate smalldatetime  NOT NULL CHECK(invoiceDate <= GETDATE()),
        customerId int  NOT NULL,
        address varchar(100)  NOT NULL,
        city varchar(100)  NOT NULL,
        postalCode varchar(10)  NOT NULL,
    isPeriodic bit  NOT NULL,
        CONSTRAINT Invoices_pk PRIMARY KEY  (invoiceId)
);
```

4. Widoki

4.1 CurrentMenu

Widok zawiera obecne dania zawarte w obecnym menu. Dla każdego dania wypisuje jego ItemId, categoryId, nazwę, cenę, wielkość porcji, jego zdjęcie oraz opis.

```
CREATE VIEW CurrentMenu
AS
SELECT D.itemId, D.categoryId, D.itemName,
    D.itemPrice, D.servingSize, D.itemPhoto,
    D.description
FROM dbo.Dishes AS D
INNER JOIN dbo.MenuDetails AS MD
ON D.itemId = MD.itemId
INNER JOIN dbo.Menus AS M
ON MD.menuId = M.menuId
WHERE GETDATE() BETWEEN M.inDate AND M.outDate
```

4.2 DishesFromLastTwoWeeks

Widok zawiera dania obecne we wszystkich menu z ostatnich dwóch tygodni. Przydatne w walidacji menu.

```
CREATE VIEW DishesFromLastTwoWeeks
AS
SELECT D.itemId, itemName
FROM Dishes as D
WHERE itemId IN (SELECT MD.itemId FROM MenuDetails AS MD
    INNER JOIN Menus AS M ON M.MenuId = MD.MenuId
    WHERE MD.menuId = M.menuId AND inDate < GETDATE() AND DATEDIFF(DAY, outDate, GETDATE()) < 14)
```

4.3 UpcomingReservations

Widok zawiera dane o nadchodzących rezerwacjach, powiązanych z nimi zamówieniach, całkowitej kwocie zamówienia oraz klientach, którzy je złożyli.

```
CREATE VIEW UpcomingReservations
AS
SELECT R.reservationId, R.numberOfGuests, R.reservationDate,
    O.orderId, O.customerId, C.firstName, C.lastName,
    SUM(OD.itemPrice * OD.quantity * (1-O.discount)) AS priceToPay
FROM dbo.Reservations AS R
INNER JOIN Orders AS O
ON R.orderId = O.orderId
INNER JOIN OrderDetails AS OD
ON O.orderId = OD.orderId
INNER JOIN Customers AS C
ON O.customerId = C.customerId
```

```
WHERE R.reservationDate >= GETDATE()
GROUP BY R.reservationId, R.numberOfGuests, R.reservationDate,
    O.orderId, O.customerId, C.firstName, C.lastName
```

4.4 UpcomingOrders

Widok zawiera dane o zamówieniach, których daty odbioru lub daty związane z rezerwacją nadchodzą, całkowitych kwotach do zapłaty, tym, czy są na wynos, klientach, którzy złożyli zamówienie.

```
CREATE VIEW UpcomingOrders
AS SELECT O.orderId, O.receiveDate, O.takeOut,
    O.isPaid, SUM(OD.itemPrice * OD.quantity * (1-O.discount)) AS OrderPrice,
    C.customerId, C.firstName, C.lastName
FROM Orders AS O
INNER JOIN OrderDetails AS OD
ON O.orderId = OD.orderId
INNER JOIN Customers AS C
ON O.customerId = C.customerId
WHERE O.receiveDate >= GETDATE()
GROUP BY O.orderId, O.receiveDate, O.takeOut,
    O.isPaid, C.customerId, C.firstName, C.lastName
```

4.5 GeneralOrdersStats

Widok zawiera dane (segregowane po latach i miesiącach) o łącznym przychodzie, średniej wartości zamówienia, kwotach straconych na rabatach oraz ilości zamówień.

```
CREATE VIEW GeneralOrdersStats AS
SELECT YEAR(O.orderDate) AS 'Order year',
       MONTH(O.orderDate)                                AS 'Order month',
       SUM(OD.quantity * OD.itemPrice * (1 - O.discount)) AS 'Total income',
       AVG(OD.quantity * OD.itemPrice * (1 - O.discount)) AS 'Average order price',
       SUM(OD.quantity * OD.itemPrice * O.discount)       AS 'Price lost on discounts',
       COUNT(O.orderId)                                  AS 'Number of orders'
FROM dbo.Orders AS O
         INNER JOIN dbo.OrderDetails OD ON O.orderId = OD.orderId
GROUP BY YEAR(O.orderDate), MONTH(O.orderDate)
```

4.6 TableReservationCount

Widok zawiera dane (segregowane po latach i miesiącach) o tym ile razy dany stolik był rezerwowany w danym roku i miesiącu.

```
CREATE VIEW TableReservationCount AS
SELECT T.tableId  AS 'Table id',
        YEAR(R.reservationDate) AS 'Reservation Year',
```

```
        MONTH(R.reservationDate) AS 'Reservation Month',
        COUNT(*) AS 'Number of reservations'
FROM Tables AS T
        INNER JOIN ReservationDetails RD ON T.tableId = RD.tableId
        INNER JOIN Reservations R ON R.reservationId = RD.reservationId
GROUP BY T.tableId, YEAR(R.reservationDate), MONTH(R.reservationDate)
```

4.7 DishPurchaseCount

Widok zawiera dane (segregowane po latach i miesiącach) o tym ile razy dany przedmiot był kupowany w danym roku i miesiącu.

```
CREATE VIEW DishPurchaseCount AS
SELECT itemName, YEAR(orderDate) AS year, MONTH(orderDate) AS month, COUNT(D.itemId) AS itemCount
FROM Dishes AS D
INNER JOIN OrderDetails AS OD ON D.itemId = OD.itemId
INNER JOIN Orders AS O ON O.orderId = OD.orderId
GROUP BY YEAR(orderDate), MONTH(orderDate), D.itemId, itemName
```

4.8 CustomerSpendingsStats

Widok zawiera dane o klientach io wydanych przez nich kwotach na zamówienia.

```
CREATE VIEW CustomerSpendingsStats AS
SELECT C.customerId, C.firstName, C.lastName, C.companyName,
    SUM(OD.itemPrice * OD.quantity * (1 - O.discount)) AS totalSpendings
FROM Customers AS C
INNER JOIN Orders AS O
ON C.customerId = O.OrderId
INNER JOIN OrderDetails AS OD
ON O.orderId = OD.orderId
GROUP BY C.customerId, C.firstName, C.lastName, C.companyName
```

5. Procedury

5.1 InsertToMenu

Procedura dodaje danie do menu. Jako argumenty przyjmuje menuId menu, do którego danie jest dodawane oraz itemId danego dania.

```
CREATE PROCEDURE InsertToMenu
    @menuId INT,
    @itemId INT
AS
BEGIN
    IF (NOT EXISTS(SELECT * FROM Menus WHERE @menuId = menuId))
        BEGIN
            RAISERROR ('No such menuId', -1, -1)
            RETURN
        END
    IF (NOT EXISTS(SELECT * FROM Dishes WHERE @itemId = itemId))
        BEGIN
            RAISERROR ('No such itemId', -1, -1)
            RETURN
        END
    INSERT INTO MenuDetails(menuId, itemId)
    VALUES (@menuId, @itemId)
END
```

5.2 ValidateMenu

Procedura sprawdza poprawność menu (zawiera maksymalnie połowę dań, które znajdowały się w menu w przeciągu ostatnich dwóch tygodni), jeżeli spełnia ono warunki, ustawia flagę ***isValid***.

```
CREATE PROCEDURE ValidateMenu @menuId INT
AS
BEGIN
    IF (NOT EXISTS(SELECT menuId FROM Menus WHERE menuId = @menuId))
        BEGIN
            RAISERROR ('No such menuId', -1, -1)
            RETURN
        END
    IF (2 * (SELECT COUNT(*) FROM DishesFromLastWeeks) < (SELECT COUNT(*)
                                                          FROM MenuDetails
                                                          WHERE menuId = @menuId))
        BEGIN
            UPDATE Menus SET isValid = 1 WHERE menuId = @menuId
        END
END
```

5.3 CreateMenu

Procedura dodaje nowe menu przy założeniu, że czas aktywności menu wynosi maksymalnie dwa tygodnie(ponieważ po dwóch tygodniach połowa dań w menu musi być zmieniona).

```
CREATE PROCEDURE CreateMenu @inDate SMALLDATETIME,
                            @outDate SMALLDATETIME
AS
BEGIN
    IF (DATEDIFF(DAY, @inDate, @outDate) > 14 OR DATEDIFF(DAY, @inDate, @outDate) < 0)
        BEGIN
            RAISERROR ('Wrong outDate or inDate', -1, -1)
                        RETURN
        END
    INSERT INTO Menus(inDate, outDate)
    VALUES (@inDate, @outDate)
END
```

5.4 AddNewOrder

Procedura dodaje nowe zamówienie, jako argumenty przyjmuje customerId, employeeId, orderDate, receiveDate, isPaid, takeOut, discountType. Jeśli discountType oznacza rabat jednorazowy, to procedura zmienia pole wasUsed na 1 w rabacie, który używa.

```
CREATE PROCEDURE AddNewOrder @custemerId INT,
                             @employeeId INT,
                             @orderDate SMALLDATETIME,
                             @receiveDate SMALLDATETIME,
                             @isPaid BIT,
                             @takeOut BIT,
                             @discountType VARCHAR(20)
AS
BEGIN
    IF (@discountType NOT LIKE 'lifetime' OR @discountType NOT LIKE 'temporary')
        BEGIN
            RAISERROR ('No such discount type', -1, -1)
            RETURN
        END
    IF (@orderDate > @receiveDate)
        BEGIN
            RAISERROR ('Wrong orderDate or receiveDate', -1, -1)
            RETURN
        END
    IF (NOT EXISTS(SELECT customerId FROM Customers WHERE customerId = @custemerId))
        BEGIN
            RAISERROR ('No such customerId', -1, -1)
            RETURN
        END
    IF (NOT EXISTS(SELECT employeeId FROM Employees WHERE employeeId = @employeeId))
        BEGIN
            RAISERROR ('No such employeeId', -1, -1)
            RETURN
```

```
        END
    IF (NOT EXISTS(SELECT discountId
                   FROM Discounts
                   WHERE customerId = @custemerId
                     AND discountType = @discountType
                     AND (wasUsed = NULL OR wasUsed = 0)))
        BEGIN
            RAISERROR ('Customer does not have such discount', -1, -1)
            RETURN
        END
    DECLARE @discount DECIMAL(3, 2)
    IF (@discountType LIKE 'lifetime')
        BEGIN
            SET @discount = (SELECT R1 FROM Factors)
        END
    ELSE
        BEGIN
            DECLARE @discountId INT;
            SET @discountId = (SELECT TOP 1 discountId
                               FROM Discounts
                               WHERE customerId = @custemerId
                                 AND discountType = @discountType
                                 AND wasUsed = 0
                                 AND DATEDIFF(DAY, startDate, GETDATE()) < (SELECT D1 FROM Factors))
            UPDATE Discounts SET wasUsed = 1 WHERE @discountId = discountId
            SET @discount = (SELECT R2 FROM Factors)
        END
    INSERT INTO Orders(customerId, employeeId, orderDate, receiveDate, isPaid, takeOut, discount)
    VALUES (@custemerId, @employeeId, @orderDate, @receiveDate, @isPaid, @takeOut, @discount)
END
```

5.5 InsertIntoOrder

Procedura dodaje szczegóły zamówienia do zamówienia. Jako argumenty przyjmuje orderId zamówienia, do którego są dodawane szczegóły, itemId dania, które dodajemy do zamówienia oraz quantity, czyli ilość dodanych dań.

```
CREATE PROCEDURE InsertIntoOrder @orderId INT,
                                 @itemId INT,
                                 @quantity INT
AS
BEGIN
        IF (NOT EXISTS(SELECT orderId FROM Orders WHERE orderId = @orderId))
        BEGIN
            RAISERROR ('No such orderId', -1, -1)
            RETURN
        END
    IF (NOT EXISTS(SELECT itemId FROM Dishes WHERE itemId = @itemId AND isActive = 1))
        BEGIN
            RAISERROR ('No such itemId or dish is not active', -1, -1)
            RETURN
        END

    DECLARE @itemPrice INT
    SET @itemPrice = (SELECT itemPrice
                      FROM Dishes
                      WHERE itemId = @itemId)
    INSERT INTO OrderDetails(orderId, itemId, itemPrice, quantity)
```

```
    VALUES (@orderId, @itemId, @itemPrice, @quantity)
END
```

5.6 AddNewReservation

Procedura, która dodaje nową rezerwację, jako argumenty otrzymuje orderId zamówienia, z którym powiązana jest rezerwacja, reservationDoneDate, czyli datę utworzenia rezerwacji, reservationDate, na którą została dodana rezerwacja, oraz numberOfGuests. Procedura pozwala na rezerwację, gdy klient zapłacił za zamówienie lub spełnił warunki podane w dokumentacji.

```
CREATE PROCEDURE AddReservation @orderId INT,
                                @reservationDoneDate INT,
                                @reservationDate INT,
                                @numberOfGuests INT
AS
BEGIN
    IF (NOT EXISTS (SELECT orderId FROM Orders WHERE orderId = @orderId))
        BEGIN
            RAISERROR ('No such orderId', -1, -1)
            RETURN
        END
    IF (EXISTS (SELECT * FROM Orders WHERE orderId = @orderId AND isPaid = 1))
        BEGIN
            INSERT INTO Reservations(orderId, reservationDoneDate, reservationDate, numberOfGuests)
            VALUES (@orderId, @reservationDoneDate, @reservationDate, @numberOfGuests)
        END
    ELSE
        BEGIN
            DECLARE @customerId INT
            SET @customerId = (SELECT customerId from Orders where orderId = @orderId)
            IF ((SELECT COUNT(*)
                FROM Orders AS O
                                INNER JOIN OrderDetails AS OD
                                ON O.orderId = OD.orderID
                                WHERE O.customerId = @customerId
                                HAVING SUM((1 - O.discount) * OD.itemPrice * OD.quantity) > (SELECT
WZ FROM Factors)) > (SELECT WK FROM Factors))
            BEGIN
                INSERT INTO Reservations(orderId, reservationDoneDate, reservationDate,
numberOfGuests)
                VALUES (@orderId, @reservationDoneDate, @reservationDate, @numberOfGuests)
            END
            ELSE
            BEGIN
                RAISERROR ('Client does not meet the requirements for reservation', -1, -1)
                RETURN
            END
        END
END
```

5.7 AddReservationDetails

Procedura dodaje szczegóły do rezerwacji. Jako argumenty przyjmuje reservationId rezerwacji, do której dodajemy szczegóły oraz tableId stołu, który wiążemy z daną rezerwacją.

```
CREATE PROCEDURE AddReservationDetails @reservationId INT,
                                       @tableId INT
AS
BEGIN
     IF (NOT EXISTS(SELECT reservationId FROM Reservations WHERE reservationId = @reservationId))
        BEGIN
            RAISERROR ('No such reservationId', -1, -1)
                        RETURN
        END
    IF (NOT EXISTS(SELECT tableId FROM Tables WHERE tableId = @tableId))
        BEGIN
            RAISERROR ('No such tableId', -1, -1)
                        RETURN
        END

    INSERT INTO ReservationDetails(reservationId, tableId)
    VALUES(@reservationId, @tableId)
END
```

5.8 CreateInvoice

Procedura tworzy nową fakturę. Jako argumenty przyjmuje invoiceNumber, invoiceDate, customerId klienta, dla którego generujemy fakturę, address, city, postalCode, isPeriodic, które mówi czy dana faktura jest periodyczna oraz opcjonalne orderId, na które dodawana będzie faktura. Jeśli faktura jest periodyczna to zbiera wszystkie zamówienia klienta, do których nie została podpięta faktura.

```
CREATE PROCEDURE CreateInvoice @invoiceNumber INT,
                               @invoiceDate SMALLDATETIME,
                               @customerId INT,
                               @address VARCHAR(100),
                               @city VARCHAR(100),
                               @postalCode VARCHAR(10),
                               @isPeriodic BIT,
                               @orderId INT = NULL
AS
BEGIN
    IF (@invoiceDate > GETDATE())
    BEGIN
        RAISERROR ('Wrong date', -1, -1)
                RETURN
    END
    IF (NOT EXISTS (SELECT customerId FROM Customers WHERE customerId = @customerId))
        BEGIN
```

```
            RAISERROR ('No such customerId', -1, -1)
                        RETURN
        END
    IF (NOT EXISTS (SELECT orderId FROM Orders WHERE orderId = @orderId))
        BEGIN
            RAISERROR ('No such orderId', -1, -1)
                        RETURN
        END

    INSERT INTO Invoices(invoiceNumber, invoiceDate, customerId, address, city, postalCode,
isPeriodic)
    VALUES(@invoiceNumber, @invoiceDate, @customerId, @address, @city, @postalCode, @isPeriodic)

    DECLARE @biggestIndex INT
    SET @biggestIndex = (SELECT TOP 1 invoiceId FROM Invoices ORDER BY invoiceId DESC) + 1
    IF (@isPeriodic = 1)
    BEGIN
        UPDATE Orders SET invoiceId = @biggestIndex WHERE invoiceId = NULL AND customerId =
@customerId
    END
    ELSE
    BEGIN
        UPDATE Orders SET invoiceId = @biggestIndex WHERE invoiceId = NULL AND orderId = @orderId
    END
END
```

5.9 ShowFreeTablesAt

Procedura pokazuje niezarezerwowane stoliki w podanym czasie przy założeniu, że rezerwacja trwa ***timespan*** minut, gdzie ***timespan*** jest parametrem.

```
CREATE PROCEDURE ShowFreeTablesAt(@datetime AS smalldatetime, @timespan AS int)
AS
BEGIN
        SELECT T.tableId
        FROM Tables AS T
        WHERE T.isActive = 1 AND T.tableId NOT IN (
                SELECT T1.tableId
                FROM Tables AS T1
                INNER JOIN ReservationDetails AS RD
                ON T1.tableId = RD.tableId
                INNER JOIN Reservations AS R
                ON RD.reservationId = R.reservationId
                WHERE @datetime >= R.reservationDate AND @datetime <= DATEADD(MINUTE, @timespan,
R.reservationDate))
END
```

5.10 QualifiesForLifetimeDiscount

Procedura sprawdza, czy klient kwalifikuje się do otrzymania rabatu dożywotniego, jeżeli tak, dodaje go do tabeli ***Discounts***.

```
CREATE PROCEDURE QualifiesForLifetimeDiscount(@customerId AS int)
AS
BEGIN
```

```
        -- walidacja czy istnieje taki customer
        DECLARE @orderCount INT
        SET @orderCount = (SELECT COUNT(*) FROM (
                SELECT SUM(OD.itemPrice * OD.quantity  * (1-O.discount)) AS totalPrice
                FROM OrderDetails AS OD
                INNER JOIN Orders AS O
                ON O.orderId = OD.orderId AND O.customerId = @customerId
                GROUP BY O.OrderId
                HAVING SUM(OD.itemPrice * OD.quantity  * (1-O.discount)) >= (SELECT K1 FROM
Factors)) AS A)
        IF(@orderCount >= (SELECT Z1 FROM Factors)
                AND NOT EXISTS (
                        SELECT D.discountId FROM Discounts AS D
                        WHERE D.customerId = @customerId AND D.discountType = 'lifetime'
                        ))
        BEGIN
                INSERT INTO Discounts values(@customerId, 'lifetime', GETDATE(), NULL)
        END
END
```

5.11 QualifiesForTemporaryDiscount

Procedura sprawdza, czy klient kwalifikuje się do otrzymania rabatu tymczasowego, jeżeli tak, dodaje go do tabeli ***Discounts***.

```
CREATE PROCEDURE QualifiesForTemporaryDiscount(@customerId AS int)
AS
BEGIN
        -- walidacja czy istnieje taki customer
        DECLARE @lastTemporaryDiscount SMALLDATETIME
        SET @lastTemporaryDiscount = (
                SELECT MAX(D.startDate)
                FROM Discounts AS D
                WHERE D.customerId = @customerId AND D.discountType = 'temporary'
        )
        DECLARE @totalCost MONEY
        SET @totalCost = (
                SELECT SUM(OD.itemPrice * OD.quantity  * (1-O.discount)) AS totalPrice
                FROM OrderDetails AS OD
                INNER JOIN Orders AS O
                ON O.orderId = OD.orderId AND O.customerId = @customerId
                WHERE O.orderDate > @lastTemporaryDiscount
                GROUP BY O.OrderId
        )
        IF(@totalCost IS NULL OR @totalCost >= (SELECT K2 FROM Factors))
        BEGIN
                INSERT INTO Discounts values(@customerId, 'temporary', GETDATE(), 1)
        END
END
```

5.12 SetOrderAsPaid

Procedura ustawia flagę ***isPaid*** zamówienia na 1, do użycia, gdy takowe zostanie opłacone.

```
CREATE PROCEDURE setOrderPaid(
    @orderId INT)
AS
BEGIN
    DECLARE @checkIsPaid BIT
    SET @checkIsPaid = (SELECT isPaid FROM Orders O WHERE O.orderId = @orderId)
    IF (@checkIsPaid = 0)
        BEGIN
            DECLARE @paid BIT
            SET @paid = 1
            UPDATE Orders
            SET isPaid = @paid
        END
    ELSE
        BEGIN
            RAISERROR ('Order is already paid.', -1, -1)
        END
END
```

5.13 CancelOrder

Procedura usuwa zamówienie, jeżeli nie zostało jeszcze zrealizowane.

```
CREATE PROCEDURE CancelOrder(@orderId AS int)
AS
BEGIN
        IF (NOT EXISTS (SELECT orderId FROM Orders WHERE orderId = @orderId))
        BEGIN
                RAISERROR ('Order does not exists', -1, -1)
                RETURN
        END

        IF ( NOT ((SELECT O.receiveDate FROM Orders AS O WHERE O.orderId = @orderId) < GETDATE()))
        BEGIN
                RAISERROR ('Order already completed', -1, -1)
                RETURN
        END

        DELETE FROM Orders
        WHERE Orders.orderId = @orderId
        DELETE FROM OrderDetails
        WHERE OrderDetails.orderId = @orderId
END
```

5.14 CancelReservation

Procedura usuwa rezerwację, jeżeli nie została ona jeszcze zrealizowana oraz usuwa towarzyszące jej zamówienie.

```
CREATE PROCEDURE CancelReservation(@reservationId AS int)
AS
BEGIN
	IF (NOT EXISTS (SELECT reservationId FROM Reservations WHERE reservationId =
@reservationId))
	BEGIN
		RAISERROR ('Reservation does not exists', -1, -1)
		RETURN
	END

	IF ( NOT ((SELECT O.receiveDate FROM Orders AS O WHERE O.orderId = @reservationId) <
GETDATE()))
	BEGIN
		RAISERROR ('Reservation already completed', -1, -1)
		RETURN
	END

	DECLARE @orderId AS int
	SET @orderId = (SELECT R.orderId FROM Reservations AS R WHERE R.reservationId =
@reservationId)

	EXEC CancelOrder @orderId
	DELETE FROM Reservations
	WHERE Reservations.reservationId = @reservationId

	DELETE FROM ReservationDetails
	WHERE ReservationDetails.reservationId = @reservationId
END
```

5.15 ChangeFactors

Procedura do zmiany współczynników występujących w rabatach. Zmiany w R1 oraz R2 są możliwe tylko jeżeli wprowadzone wartości są w przedziale 0.00 i 1.00.

```
CREATE PROCEDURE ChangeFactors(
    @WZ INT, @WK INT, @Z1 INT, @K1 INT, @R1 DECIMAL(3, 2), @K2 INT, @R2 DECIMAL(3, 2), @D1 INT)
AS
BEGIN
    UPDATE Factors
    SET WZ = @WZ
    UPDATE Factors
    SET WK = @WK
    UPDATE Factors
    SET Z1 = @Z1
    UPDATE Factors
    SET K1 = @K1
    IF (@R1 BETWEEN 0.00 AND 1.00)
        BEGIN
```

```
            UPDATE Factors
            SET R1 = @R1
        END
    ELSE
        BEGIN
            RAISERROR ('R1 factor is not between 0.00 and 1.00.', -1, -1)
        END
    UPDATE Factors
    SET K2 = @K2
    IF (@R2 BETWEEN 0.00 AND 1.00)
        BEGIN
            UPDATE Factors
            SET R2 = @R2
        END
    ELSE
        BEGIN
            RAISERROR ('R2 factor is not between 0.00 and 1.00.', -1, -1)
        END
    UPDATE Factors
    SET D1 = @D1
END
```

5.16 ChangeCustomer

Procedura, która zmienia dane klienta. Zmiana odbyw się, jeżeli klient o podanym id występuje już w bazie danych.

```
CREATE PROCEDURE ChangeCustomer(
    @customerId INT,
    @firstName VARCHAR(100),
    @lastName VARCHAR(100),
    @companyName VARCHAR(100) = NULL,
    @phone VARCHAR(15),
    @address VARCHAR(100) = NULL,
    @city VARCHAR(100) = NULL,
    @postalCode VARCHAR(100) = NULL,
    @email VARCHAR(100) = NULL,
    @takesPeriodicInvoice BIT)
AS
BEGIN
    IF(NOT EXISTS (SELECT customerId FROM Customers WHERE customerId = @customerId))
        BEGIN
            RAISERROR ('No customer with such id', -1, -1)
        END
    ELSE
        BEGIN
            UPDATE Customers
            SET firstName = @firstName WHERE customerId = @customerId
            UPDATE Customers
            SET lastName = @lastName WHERE customerId = @customerId
            UPDATE Customers
            SET companyName = @companyName WHERE customerId = @customerId
            UPDATE Customers
            SET phone = @phone WHERE customerId = @customerId
            UPDATE Customers
            SET address = @address WHERE customerId = @customerId
            UPDATE Customers
            SET city = @city WHERE customerId = @customerId
            UPDATE Customers
```

```
            SET postalCode = @postalCode WHERE customerId = @customerId
            IF (@email IS NULL OR @email LIKE '%_@%_.%_')
                BEGIN
                    UPDATE Customers
                    SET email = @email WHERE customerId = @customerId
                END
            ELSE
                BEGIN
                    RAISERROR ('Email is not in email format.', -1, -1)
                END
            UPDATE Customers
            SET takesPeriodicInvoice = @takesPeriodicInvoice WHERE customerId = @customerId
        END
END
```

5.17 ChangeEmployee

Procedura, która zmienia dane pracownika. Zmiana zostanie dokonana tylko wtedy, gdy pracownik o podanym id występuje już w bazie danych.

```
CREATE PROCEDURE ChangeEmployees(
    @employeeId INT,
    @firstName VARCHAR(100),
    @lastName VARCHAR(100),
    @birthDate SMALLDATETIME,
    @hireDate SMALLDATETIME,
    @position VARCHAR(100),
    @phone VARCHAR(15),
    @address VARCHAR(100),
    @city VARCHAR(100),
    @postalCode VARCHAR(10),
    @reportsTo INT)
AS
BEGIN
        IF(NOT EXISTS (SELECT @employeeId FROM Employees WHERE employeeId = @employeeId))
        BEGIN
            RAISERROR ('No employee with such id', -1, -1)
        END
    ELSE
        BEGIN
                        UPDATE Employees
                        SET firstName = @firstName WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET lastName = @lastName WHERE @employeeId = employeeId
                        IF (@birthDate <= GETDATE())
                                BEGIN
                                        UPDATE Employees
                                        SET birthDate = @birthDate WHERE @employeeId = employeeId
                                END
                        ELSE
                                BEGIN
                                        RAISERROR ('Hire date is not before current date.', -1, -1)
                                END
                        IF (@hireDate <= GETDATE())
                                BEGIN
                                        UPDATE Employees
                                        SET hireDate = @hireDate WHERE @employeeId = employeeId
                                END
                        ELSE
```

```
                                BEGIN
                                        RAISERROR ('Hire date is not before current date.', -1, -1)
                                END
                        UPDATE Employees
                        SET position = @position WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET phone = @phone WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET address = @address WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET city = @city WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET postalCode = @postalCode WHERE @employeeId = employeeId
                        UPDATE Employees
                        SET reportsTo = @reportsTo WHERE @employeeId = employeeId
                END
END
```