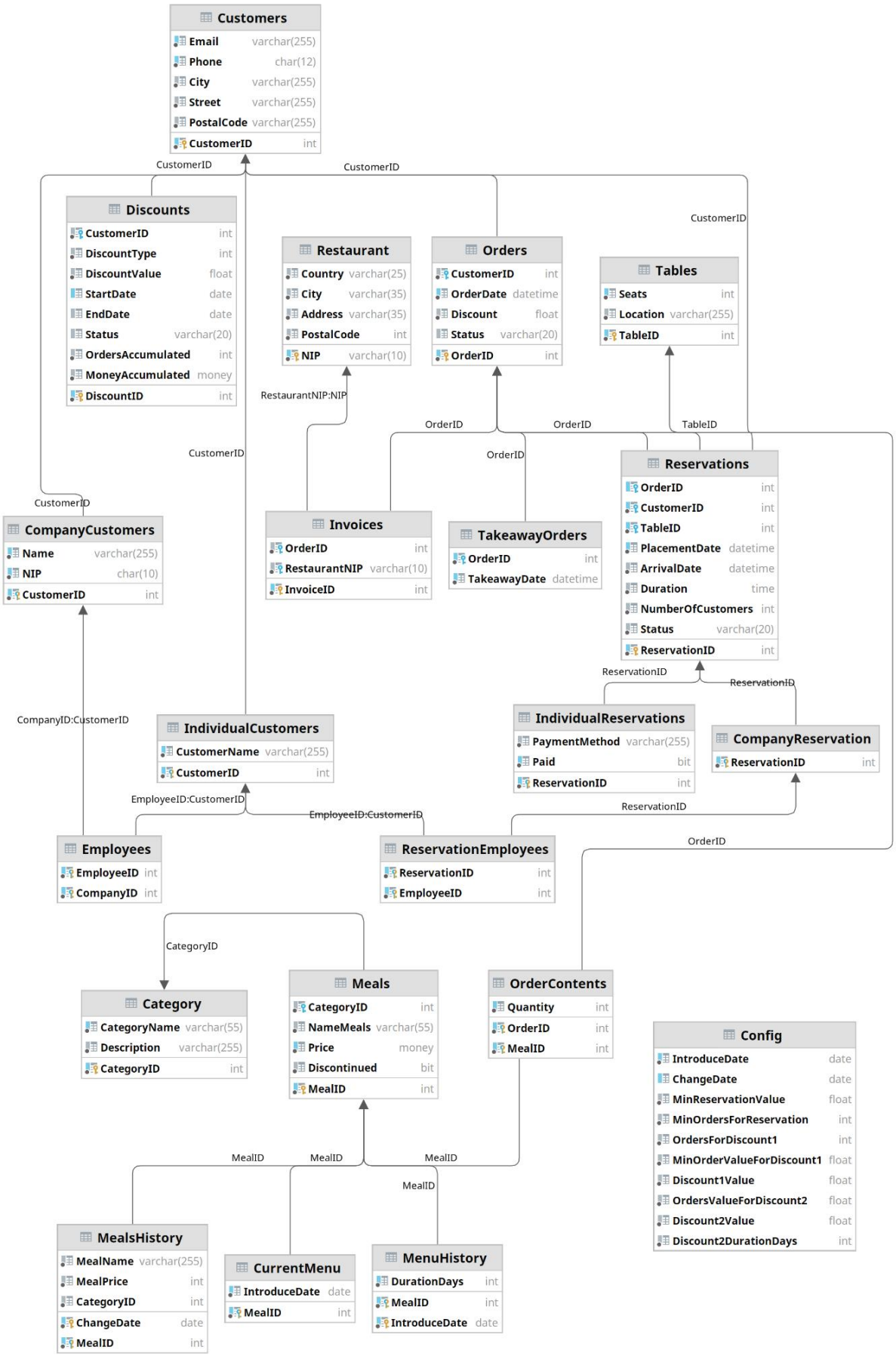
Podstawy baz danych

Projekt systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm

Natalia Luberda
Kaja Dudek
Jakub Pryc

Spis treści

Schemat bazy danych



Opis użytkowników

- klient indywidualny
    1. Ma możliwość składania zamówień
    2. Rezerwacji stolika przy jednoczesnym złożeniu zamówienia
    3. Generowania raportów dotyczących zamówień oraz rabatów

- klient firmowy
    1. Ma możliwość składania zamówień
    2. Rezerwacji stolików na firmę
    3. Rezerwacji stolików dla konkretnych pracowników firmy (imiennie)
    4. Generowania raportów dotyczących zamówień oraz rabatów

- restauracja
    1. Generowanie nowego menu
    2. Zmiana wartości danych konfiguracyjnych (WZ, WK itp.)
    3. Zmiana cen produktów
    4. Wypisanie listy wolnych stolików w tym czasie
    5. Generowanie raportów miesięcznych
    6. Generowanie raportów dniowych
    7. Generowanie raportów rocznych
    8. Generowanie raportów dot. najlepiej sprzedających się dań
    9. Generowanie raportów dot. średniej ceny menu
    10. Generowanie raportów dot. aktywnego menu w podanym czasie
    11. Generowanie raportów dot. zniżek klientów w danym miesiącu
    12. Generowanie raportów dot. zniżek klientów w danym tygodniu
    13. Generowanie miesięcznych i tygodniowych raportów dotyczących zarezerwowanych stolików
    14. Dodanie kategorii
    15. Dodanie posiłków
    16. Wystawienie faktury
    17. Wystawienie faktury zbiorczej

Opis tabel

wraz z kodem je generującym oraz opisem warunków integralności

Tabela Category

Tabela kategorii dań dostępnych w restauracji

```
CREATE TABLE Category
(
    CategoryID   int          NOT NULL IDENTITY (1, 1),
    CategoryName varchar(55)  NOT NULL,
    Description  varchar(255) NOT NULL,
    CONSTRAINT CategoryCheck CHECK (CategoryName not like '%[^a-zA-Z ]%'),
    CONSTRAINT CategoryPK PRIMARY KEY (CategoryID)
)
```

Klucz główny: CategoryID - int

Nazwa kategorii: CategoryName – varchar, składa się jedynie z różnej wielkości liter i spacji

Opis kategorii: Description – varchar

Tabela CompanyCustomers

Tabela firm

```
CREATE TABLE CompanyCustomers
(
    CustomerID int          NOT NULL,
    Name       varchar(255) NOT NULL,
    NIP        char(10)     NOT NULL,

    CONSTRAINT CompanyClients_pk PRIMARY KEY (CustomerID),
    CONSTRAINT ValidNIP CHECK ( NIP LIKE
                                '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9]')
)

ALTER TABLE CompanyCustomers
    ADD CONSTRAINT Customers_fk
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
```

Klucz główny i obcy: CustomerID - int

Nazwa firmy: Name – varchar

NIP firmy: NIP – char, składa się z 10 cyfr

Tabela CompanyReservation

Tabela rezerwacji złożonych przez firmy

```
CREATE TABLE CompanyReservation
(
    ReservationID int NOT NULL,

    CONSTRAINT CompanyReservation_pk PRIMARY KEY (ReservationID),
)

ALTER TABLE CompanyReservation
    ADD CONSTRAINT CompanyToGeneral_fk
        FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID)
```

Klucz główny i obcy: ReservationID - int

Tabela Config

Tabela z danymi konfiguracyjnymi

```
CREATE TABLE Config
(
    IntroduceDate             date  NOT NULL,
    ChangeDate                date,
    MinReservationValue       float NOT NULL,
    MinOrdersForReservation   int   NOT NULL,
    OrdersForDiscount1        int   NOT NULL,
    MinOrderValueForDiscount1 float NOT NULL,
    Discount1Value            float NOT NULL,
    OrdersValueForDiscount2   float NOT NULL,
    Discount2Value            float NOT NULL,
    Discount2DurationDays     int   NOT NULL,

    CONSTRAINT ConfigPK PRIMARY KEY (IntroduceDate, ChangeDate),
    CONSTRAINT MinReservationValuePositive CHECK (MinReservationValue > 0),
    CONSTRAINT MinOrdersForReservationPositive CHECK
(MinOrdersForReservation > 0),

    CONSTRAINT OrdersForDiscount1Positive CHECK (OrdersForDiscount1 > 0),
    CONSTRAINT MinOrderValueForDiscount1Positive CHECK
(MinOrderValueForDiscount1 > 0),
    CONSTRAINT ValidDiscount1Value CHECK (Discount1Value BETWEEN 0 AND 1),

    CONSTRAINT OrdersValueForDiscount2Positive CHECK
(OrdersValueForDiscount2 > 0),
    CONSTRAINT ValidDiscount2Value CHECK (Discount2Value BETWEEN 0 AND 1),
    CONSTRAINT Discount2DurationDaysPositive CHECK (Discount2DurationDays >
0),
)
```

Data wprowadzenia obowiązujących danych: IntroduceDate – date

Data zmiany obowiązujących danych: ChangeDate – date, nieobowiązkowa

Minimalna wartość zamówienia umożliwiająca klientowi indywidualnemu rezerwację stolika: MinReservationValue – float, większa niż 0

Minimalna ilość złożonych zamówień umożliwiająca klientowi indywidualnemu rezerwację stolika: MinOrdersForReservation – int, większa niż 0

Minimalna ilość zamówień potrzebna do uzyskania zniżku typu 1: OrdersForDiscount1 – int, większa niż 0

Minimalna wartość tych zamówień: MinOrderValueForDiscount1 – float, większa niż 0

Wartość zniżki typu 1: Discount1Value – float, wartość w przedziale od 0 do 1

Minimalna łączna wartość zamówień potrzebna do uzyskania zniżki typu 2: OrdersValueForDiscount2 – float, większa niż 0

Wartość zniżki typu 2: Discount2Value – float, wartość w przedziale od 0 do 1

Czas trwania zniżki typu2: Discount2DurationDays – int, większa niż 0

Tabela CurrentMenu

Tabela zawierająca aktualne menu

```
CREATE TABLE CurrentMenu
(
    IntroduceDate date NOT NULL,
    MealID        int  NOT NULL,
    CONSTRAINT CurrentMenu_pk PRIMARY KEY (MealID)
)

ALTER TABLE CurrentMenu
    ADD CONSTRAINT MenuPosition
        FOREIGN KEY (MealID)
            REFERENCES Meals (MealID)
```

Klucz główny i obcy: MealID – int

Data wprowadzenia tego menu: IntroduceDate - date

Tabela Customers

Tabela klientów

```
CREATE TABLE Customers
(
    CustomerID int          NOT NULL IDENTITY (1, 1),
    Email      varchar(255) NOT NULL,
    Phone      char(12)     NOT NULL,
    City       varchar(255) NOT NULL,
    Street     varchar(255) NOT NULL,
    PostalCode varchar(255) NOT NULL,

    CONSTRAINT ValidPostalCode CHECK (PostalCode LIKE
                                      '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT ValidEmail CHECK (Email LIKE '%@%.%'),
    CONSTRAINT ValidPhone CHECK (Phone LIKE
                                 '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9][0-9]'),
    CONSTRAINT Customers_pk PRIMARY KEY (CustomerID)
)
```

Klucz główny: CustomerID – int

Email: Email – varchar, musi zawierać @ i kropkę.

Nr telefonu: Phone – char, musi składać się z plusa na początku i 11 cyfr

Miasto: City – varchar

Ulica: Street - varchar

Kod pocztowy: PostalCode – varchar, musi składać się z 2 cyfr, myślnika i kolejnych 3 cyfr

Tabela Discounts

Tabela zniżek i ich historii

```
CREATE TABLE Discounts
(
   DiscountID        int         NOT NULL IDENTITY (1, 1),
   CustomerID        int         NOT NULL,
   DiscountType      int         NOT NULL,
   DiscountValue     float       NOT NULL,
   StartDate         date        NOT NULL,
   EndDate           date,
   Status            varchar(20) NOT NULL,
   OrdersAccumulated int         NOT NULL,
   MoneyAccumulated  money       NOT NULL,

   CONSTRAINT DiscountsPK PRIMARY KEY (DiscountID),
   CONSTRAINT ValidDiscountType CHECK (DiscountType BETWEEN 1 AND 2),
   CONSTRAINT ValidDiscountValue CHECK (DiscountValue BETWEEN 0 AND 1),
   CONSTRAINT EndDateForDiscount2 CHECK (DiscountType = 1 OR EndDate IS NOT
NULL),
   CONSTRAINT ValidDateRelation CHECK (EndDate > StartDate),
   CONSTRAINT StatusEnum CHECK (Status IN ('Counting', 'Active',
'Deactivated'))
)


ALTER TABLE Discounts
    ADD CONSTRAINT DiscountCustomer_fk
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
```

Klucz główny: DiscountID – int

Klucz obcy: CustomerID - int

Typ zniżki: DiscountType – int, wartość 1 (dla typu pierwszego) lub 2 (dla zniżki typu drugiego)

Wartość zniżki: DiscountValue – float, wartości pomiędzy 0 a 1

Data rozpoczęcia obowiązywania danej zniżki: StartDate – date

Data zakończenia obowiązywania danej zniżki: EndDate – date, późniejsza niż StartDate, może być równa NULL wtedy i tylko wtedy, gdy obowiązuje zniżka typu pierwszego

Status zniżki: Status - ENUM(‘Counting’, ‘Active’, ‘Deactivated’)

Liczba naliczonych zamówień do zniżki 1: OrdersAccumulated - int

Liczba naliczonej wartości zamówień do zniżki 2: MoneyAccumulated - money

Tabela Employees

Tabela pracowników firm

```
CREATE TABLE Employees
(
    EmployeeID int NOT NULL,
    CompanyID  int NOT NULL,

    CONSTRAINT Employees_pk PRIMARY KEY (EmployeeID, CompanyID)
)

ALTER TABLE Employees
    ADD CONSTRAINT EmployeeCustomer_fk
        FOREIGN KEY (EmployeeID) REFERENCES IndividualCustomers
(CustomerID)

ALTER TABLE Employees
    ADD CONSTRAINT EmployeeCompany_fk
        FOREIGN KEY (CompanyID) REFERENCES CompanyCustomers (CustomerID)
```

Klucz główny i obcy: EmployeeID – int

Klucz obcy: CompanyID - int

Tabela IndividualCustomers

Tabela klientów indywidualnych

```
CREATE TABLE IndividualCustomers
(
    CustomerID   int          NOT NULL,
    CustomerName varchar(255) NOT NULL,

    CONSTRAINT CustomersPK PRIMARY KEY (CustomerID),
)

ALTER TABLE IndividualCustomers
    ADD CONSTRAINT Customers__fk
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
```

Klucz główny i obcy: CustomerID – int

Nazwa klienta: CustomerName - varchar

Tabela IndividualReservations

Tabela rezerwacji klientów indywidualnych

```
CREATE TABLE IndividualReservations
(
    ReservationID int          NOT NULL,
    PaymentMethod varchar(255) NOT NULL,
    Paid          bit          NOT NULL,

    CONSTRAINT IndividualReservationsPK PRIMARY KEY (ReservationID),
)

ALTER TABLE IndividualReservations
    ADD CONSTRAINT IndividualToGeneral_fk
        FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID)
```

Klucz główny i obcy: ReservationID – int

Metoda płatności: PaymentMethod – varchar

Czy zamówienie zostało opłacone: Paid - bit

Tabela Invoices

Tabela faktur

```
CREATE TABLE Invoices
(
    InvoiceID     int         NOT NULL IDENTITY (1, 1),
    OrderID       int         NOT NULL,
    RestaurantNIP varchar(10) NOT NULL,

    CONSTRAINT InvoicesPK PRIMARY KEY (InvoiceID)
)

ALTER TABLE Invoices
    ADD CONSTRAINT RestaurantNIP_fk
        FOREIGN KEY (RestaurantNIP) REFERENCES Restaurant (NIP)

ALTER TABLE Invoices
    ADD CONSTRAINT InvoiceOrder_fk
        FOREIGN KEY (OrderID) REFERENCES Orders (OrderID)
```

Klucz główny: InvoiceID – int

Klucze obce: RestaurantNIP – varchar, OrderID - int

Tabela Meals

Tabela dań

```
CREATE TABLE Meals
(
    MealID       int          NOT NULL IDENTITY (1, 1),
    CategoryID   int          NOT NULL,
    NameMeals    varchar(55) NOT NULL,
    Price        money        NOT NULL,
    Discontinued bit          NOT NULL,
    CONSTRAINT ErrorPrice CHECK (Price > 0),
    CONSTRAINT NameMealsCheck CHECK (NameMeals not like '%[^a-zA-Z ]%'),
    CONSTRAINT MealsPK PRIMARY KEY (MealID)
)

ALTER TABLE Meals
    ADD CONSTRAINT CategoryMenuItems
        FOREIGN KEY (CategoryID)
            REFERENCES Category (CategoryID)
```

Klucz główny: MealID – int

Klucz obcy: CategoryID - int

Nazwa posiłku: NameMeals – varchar, różnej wielkości litery i spacje

Cena posiłku: Price – money, większa niż 0

Czy został wycofany: Discontinued - bit

Tabela MealsHistory

Tabela wszystkich, nawet nieaktualnych dań

```
CREATE TABLE MealsHistory
(
    ChangeDate date         NOT NULL,
    MealID     int          NOT NULL,
    MealName   varchar(255) NOT NULL,
    MealPrice  money        NOT NULL,
    CategoryID int          NOT NULL,

    CONSTRAINT MealsHistoryPK PRIMARY KEY (ChangeDate, MealID)
)

ALTER TABLE MealsHistory
    ADD CONSTRAINT MealRecordToCurrent_fk
        FOREIGN KEY (MealID) REFERENCES Meals (MealID)
```

Klucz główny: ChangeDate i MealID – date i int

Klucz obcy: MealID – int

Nazwa posiłku: MealName - varchar

Cena posiłku: MealPrice - money

Kategoria: CategoryID - int

Tabela MenuHistory

Tabela wszystkich menu

```
CREATE TABLE MenuHistory
(
    MealID        int  NOT NULL,
    IntroduceDate date NOT NULL,
    DurationDays  int  NOT NULL,

    CONSTRAINT MenuHistoryPK PRIMARY KEY (MealID, IntroduceDate),
    CONSTRAINT DurationDaysPositive CHECK (DurationDays > 0)
)

ALTER TABLE MenuHistory
    ADD CONSTRAINT MenuRecordToCurrent_fk
        FOREIGN KEY (MealID) REFERENCES Meals (MealID)
```

Klucz główny: MealID i IntroduceDate – int, date

Klucz obcy: MealID – int

Czas trwania danego menu: DurationDays - int

Tabela OrderContents

Tabela szczegółów zamówienia

```
CREATE TABLE OrderContents
(
    OrderID  int NOT NULL,
    MealID   int NOT NULL,
    Quantity int NOT NULL,
    CONSTRAINT ValidQuantity CHECK (Quantity > 0),
    CONSTRAINT OrderContentsPK PRIMARY KEY (OrderID, MealID)
)

ALTER TABLE OrderContents
    ADD CONSTRAINT OrdersNumber
        FOREIGN KEY (OrderID)
            REFERENCES Orders (OrderID)

ALTER TABLE OrderContents
    ADD CONSTRAINT OrdersMeals
        FOREIGN KEY (MealID)
            REFERENCES Meals (MealID)
```

Klucz główny: OrderID i MealID – int, int

Klucz obcy: OrderID – int, MealID – int

Ilość danej pozycji z zamówienia: Quantity – int, większa niż 0

Tabela Orders

Tabela zamówień

```
CREATE TABLE Orders
(
   OrderID    int          NOT NULL IDENTITY (1, 1),
   CustomerID int          NOT NULL,
   OrderDate  datetime     NOT NULL,
   Discount   float        NOT NULL,
   Status     varchar(20)  NOT NULL,

   CONSTRAINT OrdersPK PRIMARY KEY (OrderID),
   CONSTRAINT ValidDiscount CHECK (Discount BETWEEN 0 AND 1),
   CONSTRAINT OrderStatusEnum CHECK (Status IN ('Pending', 'Complete'))
)


ALTER TABLE Orders
    ADD CONSTRAINT OrderCustomers_fk
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
```

Klucz główny: OrderID – int

Klucz obcy: CustomerID - int

Data złożenia zamówienia: OrderDate - date

Obowiązująca korzystna zniżka: Discount – float, wartości pomiędzy 0 a 1

Status zamówienia: Status - ENUM(‘Pending’, ‘Complete’)

Tabela ReservationEmployees

Tabela rezerwacji na firmę

```
CREATE TABLE ReservationEmployees
(
    ReservationID int NOT NULL,
    EmployeeID    int NOT NULL,

    CONSTRAINT ReservationEmployees_pk PRIMARY KEY (ReservationID,
EmployeeID),
)

ALTER TABLE ReservationEmployees
    ADD CONSTRAINT ReservationEmployee_fk
        FOREIGN KEY (EmployeeID) REFERENCES IndividualCustomers
(CustomerID)

ALTER TABLE ReservationEmployees
    ADD CONSTRAINT ReservationID_fk
        FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID)
```

Klucz główny i obcy: ReservationID i EmployeeID – int, int

Tabela Reservations

Tabela rezerwacji

```
CREATE TABLE Reservations
(
    ReservationID     int          NOT NULL IDENTITY (1, 1),
    OrderID           int          NOT NULL,
    CustomerID        int          NOT NULL,
    TableID           int          NOT NULL,
    PlacementDate     datetime     NOT NULL,
    ArrivalDate       datetime     NOT NULL,
    Duration          time         NOT NULL,
    NumberOfCustomers int          NOT NULL,
    Status            varchar(20) NOT NULL,

    CONSTRAINT ReservationsPK PRIMARY KEY (ReservationID),
    CONSTRAINT PositiveNumberOfCustomers CHECK (NumberOfCustomers > 1),
    CONSTRAINT ReservationStatusEnum CHECK (Status IN ('Pending',
'Complete'))

)

ALTER TABLE Reservations
    ADD CONSTRAINT ReservationOrder_fk
        FOREIGN KEY (OrderID) REFERENCES Orders (OrderID)

ALTER TABLE Reservations
    ADD CONSTRAINT ReservationCustomer_fk
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)

ALTER TABLE Reservations
    ADD CONSTRAINT ReservationTable_fk
        FOREIGN KEY (TableID) REFERENCES Tables (TableID)
```

Klucz główny: ReservationID – int

Klucze obce: OrderID – int, CustomerID – int, TableID – int

Data i godzina złożenia rezerwacji: PlacementDate - datetime

Data i godzina rezerwacji stolika: ArrivalDate - datetime

Czas trwania rezerwacji: Duration – time

Ilość osób w rezerwacji: NumberOfCustomers – int, większa niż 1

Status rezerwacji: Status - ENUM(‘Pending’, ‘Complete’)

Tabela Restaurant

Tabela restauracji

```
(
    NIP         varchar(10) NOT NULL,
    Country     varchar(25) NOT NULL,
    City        varchar(35) NOT NULL,
    Address     varchar(35) NOT NULL,
    PostalCode varchar(6)  NOT NULL,
    CONSTRAINT NIPCheck CHECK (NIP like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9][0-9]'),
    CONSTRAINT PostalCodeCheck CHECK (PostalCode like '[0-9][0-9]-[0-9][0-
9][0-9]'),
    CONSTRAINT AddressCheck CHECK (Address not like '%[^a-zA-Z0-9. ]%'),
    CONSTRAINT CityCheck CHECK (City not like '%[^a-zA-Z ]%'),
    CONSTRAINT CountryCheck CHECK (Country not like '%[^a-zA-Z ]%'),
    CONSTRAINT RestaurantPK PRIMARY KEY (NIP)
)
```

Klucz główny: NIP – varchar, 10 cyfr

Kraj, w którym znajduje się restauracja: Country – varchar, różnej wielkości litery i spacje

Miasto: City – varchar, litery i spacje

Adres: Address – varchar, litery, spacje, cyfry i kropka

Kod pocztowy: PostalCode – varchar, 2 cyfry i myślnik, a następnie znów 3 cyfry

Tabela Tables

Tabela stolików

```
CREATE TABLE Tables
(
    TableID  int          NOT NULL IDENTITY (1, 1),
    Seats    int          NOT NULL,
    Location varchar(255) NOT NULL,

    CONSTRAINT Tables_pk PRIMARY KEY (TableID),
    CONSTRAINT PositiveSeats CHECK (Seats > 0)
)
```

Klucz główny: TableID – int

Ilość miejsc przy stoliku: Seats – int, większa niż 0

Lokalizacja (wewnątrz, na zewnątrz): Location - varchar

Tabela TakeawayOrders

Tabela zamówień na wynos

```
CREATE TABLE TakeawayOrders
(
    OrderID      int      NOT NULL,
    TakeawayDate datetime NOT NULL,
)

ALTER TABLE TakeawayOrders
    ADD CONSTRAINT OrderNumber
        FOREIGN KEY (OrderID)
            REFERENCES Orders (OrderID)
```

Klucz obcy: OrderID – int

Data i godzina odbioru: TakeawayDate - datetime

Funkcje

Funkcja getAvgCurrentMenuPrice

Zwraca średnią wartość aktualnego menu

```
CREATE FUNCTION getAvgCurrentMenuPrice()
RETURNS float AS
    BEGIN
        RETURN
            (SELECT AVG(M.Price)
             FROM CurrentMenu CR
             LEFT JOIN Meals M on M.MealID = CR.MealID)
    end
go
```

Funkcja getMealsSoldAtLeastXTimes

Zwraca dania sprzedane co najmniej X razy

```
CREATE FUNCTION getMealsSoldAtLeastXTimes(@val int)
RETURNS TABLE AS
    RETURN
    SELECT COUNT(OC.MealID) as NumberSold, M.NameMeals
    FROM Meals M
    INNER JOIN OrderContents OC on M.MealID = OC.MealID
GROUP BY M.NameMeals
go
```

Funkcja getMenuItemsByDate

Zwraca menu aktywne danego dnia wraz z jego zawartością

```
CREATE FUNCTION getMenuItemsByDate(@date date)
RETURNS TABLE AS
    RETURN
    SELECT M.NameMeals, C.CategoryName, M.Price, MH.IntroduceDate
    FROM Meals M
    INNER JOIN MenuHistory MH ON M.MealID = MH.MealID
    INNER JOIN Category C on C.CategoryID = M.CategoryID
WHERE DATEDIFF(day, @date, MH.IntroduceDate) < MH.DurationDays
  AND DATEDIFF(day, @date, MH.IntroduceDate) >=0
go
```

Funkcja getOrdersWithHigherValue

Zwraca tabelę zamówień o wartości wyższej niż podana wartość

```
CREATE FUNCTION getOrdersWithHigherValue(@val float)
RETURNS TABLE AS
    RETURN
    SELECT O.OrderID, SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
OrderPrice
    FROM Orders O
    INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
    INNER JOIN Meals M on M.MealID = OC.MealID
GROUP BY O.OrderID
HAVING SUM(M.Price * OC.Quantity * (1 - O.Discount)) > @val
go
```

Funkcja getOrderValue

Zwraca wartość zamówienia o podanym ID

```
CREATE FUNCTION getOrderValue(@id int)
    RETURNS float AS
    BEGIN
        RETURN
            (SELECT SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
OrderPrice
            FROM Orders O
            INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
            INNER JOIN Meals M on M.MealID = OC.MealID
            GROUP BY O.OrderID
        HAVING O.OrderID = @id)
    end
go
```

Funkcja getThisDayOrdersValue

Zwraca tabelę zamówień i ich cen złożonych danego dnia

```
CREATE FUNCTION getThisDayOrdersValue(@date date)
RETURNS TABLE AS
    RETURN
    SELECT O.OrderID, SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
OrderPrice
    FROM Orders O
    INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
    INNER JOIN Meals M on M.MealID = OC.MealID
GROUP BY O.OrderID, OrderDate
HAVING O.OrderDate = @date
go
```

Funkcja getThisMonthOrdersValue

Zwraca wartość zamówień złożonych danego miesiąca

```
CREATE FUNCTION getThisMonthOrdersValue(@date date)
RETURNS TABLE AS
    RETURN
    SELECT O.OrderID, SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
OrderPrice
    FROM Orders O
    INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
    INNER JOIN Meals M on M.MealID = OC.MealID
GROUP BY O.OrderID, OrderDate
HAVING MONTH(O.OrderDate) = MONTH(@date)
go
```

Funkcja getThisYearTotalIncome

Zwraca wartość całkowitego rocznego przychodu w podanym roku

```
CREATE FUNCTION getThisYearTotalIncome(@date date)
RETURNS float AS
    BEGIN
        RETURN
            (SELECT SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
TotalValue
            FROM Meals M
            INNER JOIN OrderContents OC on M.MealID = OC.MealID
            INNER JOIN Orders O on O.OrderID = OC.OrderID
            WHERE YEAR(@date) = YEAR(O.OrderDate))
    end
go
```

Funkcja getIndividualClientsWithMostReservations

Zwraca X klientów firmowych i indywidualnych o największej liczbie dokonanych rezerwacji w danym miesiącu

```
CREATE FUNCTION getIndividualClientsWithMostReservations(@val int, @date
date)
RETURNS TABLE AS
    RETURN
    SELECT DISTINCT TOP (@val)  C.CustomerID, C.Street, C.City,
C.PostalCode, C.Phone, COUNT(R.ReservationID) as Reservations
    FROM Reservations R
    LEFT JOIN Customers C on C.CustomerID = R.CustomerID
    LEFT JOIN Orders O on O.CustomerID = C.CustomerID
    GROUP BY C.Street, C.City, C.PostalCode, C.Phone, C.CustomerID,
R.PlacementDate
    HAVING MONTH(R.PlacementDate) = MONTH(@date) AND YEAR(R.PlacementDate)
= YEAR(@date)
ORDER BY COUNT(R.ReservationID)
go
```

Funkcja getWeeklyReservations

Zwraca raport o tygodniowych rezerwacjach

```
CREATE FUNCTION getWeeklyReservations(@date date)
RETURNS TABLE AS
    RETURN
    SELECT DISTINCT C.CustomerID, C.Street, C.City, C.PostalCode, C.Phone,
COUNT(R.ReservationID) as Reservations
    FROM Reservations R
    LEFT JOIN Customers C on C.CustomerID = R.CustomerID
    LEFT JOIN Orders O on O.CustomerID = C.CustomerID
    GROUP BY C.Street, C.City, C.PostalCode, C.Phone, C.CustomerID,
R.PlacementDate
    HAVING DATEPART(week, R.PlacementDate) = DATEPART(week, @date) AND
YEAR(R.PlacementDate) = YEAR(@date)
go
```

Funkcja getClientOrders

Zwraca raport o zamówieniach klienta

```
CREATE FUNCTION getClientOrders(@customerID int)
    RETURNS TABLE AS
        RETURN
        SELECT O.OrderID, O.OrderDate, SUM(M.Price * OC.Quantity * (1-
Discount)) as Price
        FROM Orders O
        LEFT JOIN OrderContents OC on O.OrderID = OC.OrderID
        LEFT JOIN Meals M on OC.MealID = M.MealID
        GROUP BY O.OrderID, O.OrderDate, O.CustomerID
        HAVING O.CustomerID = @customerID
```

Funkcja getClientDiscounts

Zwraca raport o wszystkich zniżkach klientów w danym miesiącu

```
CREATE FUNCTION getClientDiscounts(@date date)
    RETURNS TABLE AS
        RETURN
        SELECT DISTINCT DiscountType, DiscountValue, StartDate, EndDate,
Status
        FROM Discounts
    WHERE (MONTH(StartDate) = MONTH(@date) AND YEAR(StartDate) =
YEAR(@date) ) OR
          (MONTH(EndDate) = MONTH(@date) AND YEAR(EndDate) = YEAR(@date))
go
```

Funkcja getWeeklyClientDiscounts

Zwraca raport o wszystkich zniżkach klientów w danym tygodniu

```
CREATE FUNCTION getWeeklyClientDiscounts(@date date)
    RETURNS TABLE AS
        RETURN
        SELECT DISTINCT DiscountType, DiscountValue, StartDate, EndDate,
Status
        FROM Discounts
    WHERE (DATEPART(week, StartDate) = DATEPART(week, @date) AND
YEAR(StartDate) = YEAR(@date) ) OR
          (DATEPART(week, EndDate) = DATEPART(week, @date) AND
YEAR(EndDate) = YEAR(@date))
go
```

Funkcja getFreeTables

Zwraca listę wolnych stolików dla podanego przedziału czasowego oraz liczby miejsc

```
CREATE FUNCTION FreeTables(@StartTime datetime, @Duration time, @Seats int)
RETURNS TABLE AS
   RETURN

   SELECT Tables.TableID FROM (SELECT DISTINCT Tables.TableID FROM Tables
           INNER JOIN Reservations R2 ON Tables.TableID = R2.TableID
           WHERE ((ArrivalDate < @StartTime AND
                   @StartTime < ArrivalDate + CAST(Duration AS datetime))
                      OR
                  (ArrivalDate < @StartTime + CAST(@Duration AS datetime)
AND
                   @StartTime + CAST(@Duration AS datetime) < ArrivalDate +
CAST(Duration AS datetime))
                      OR
                   @StartTime < ArrivalDate AND
                   ArrivalDate < @StartTime + CAST(@Duration AS datetime)
                  OR Seats < @Seats) AND R2.Status='Pending') AS Taken
   RIGHT JOIN Tables ON Tables.TableID=Taken.TableID
   WHERE Taken.TableID IS NULL
GO
```

Procedury

Procedura ActivateDiscount

Aktywuje podaną zniżkę

```
CREATE PROCEDURE ActivateDiscount

@DiscountID int

AS

   BEGIN

       BEGIN TRY

           IF NOT EXISTS(SELECT DiscountID FROM Discounts WHERE
DiscountID=@DiscountID)

               BEGIN

                   THROW 52000, N'Znizka nie istnieje', 1

               END

           SET NOCOUNT ON


           DECLARE @EndDate datetime

           DECLARE @Type int

           DECLARE @DiscountValue float


           SELECT @Type=DiscountType FROM Discounts WHERE
DiscountID=@DiscountID


           IF @Type=1

               BEGIN

                   SET @DiscountValue=(SELECT Discount1Value FROM Config
WHERE ChangeDate IS NULL)

               END


           IF @Type=2

               BEGIN

                   SET @EndDate=DATEADD(DAY, (SELECT Discount2DurationDays
FROM Config WHERE ChangeDate IS NULL), GETDATE())

                   SET @DiscountValue=(SELECT Discount2Value FROM Config
WHERE ChangeDate IS NULL)

               END
```

```
            UPDATE Discounts
            SET Status='Active', StartDate=GETDATE(), EndDate=@EndDate,
DiscountValue=@DiscountValue
            WHERE @DiscountID=DiscountID


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

Procedura AddCategory

Dodaje nową kategorię do tabeli Category

```
CREATE PROCEDURE AddCategory
    @CategoryName varchar(55),
    @Description varchar(255)
AS
    BEGIN
        BEGIN TRY
            IF EXISTS(SELECT * FROM Category WHERE @CategoryName =
CategoryName)
                BEGIN
                    THROW 52000, N'Kategoria jest już dodana', 1
                END
            SET NOCOUNT ON;
            INSERT INTO Category
            VALUES (@CategoryName, @Description)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodwania kategorii: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1;
        END CATCH
    END
go
```

Procedura AddCustomer

Dodaje nowego klienta wraz z informacjami o nim do tabeli Customers, CompanyCustomers/IndividualCustomers, Employees, w przypadku klienta indywidualnego inicjuje naliczanie rabatów

```
CREATE PROCEDURE AddCustomer
   @Email varchar(255),
   @Phone char(12),
   @City varchar(255),
   @Street varchar(255),
   @PostalCode varchar(255),
   @Name varchar(255),
   @Type varchar(10),
   @IsEmployee bit,
   @EmployeeCompany varchar(255),
   @NIP char(10)
AS
   BEGIN
       BEGIN TRY
           IF @Type!='Individual' AND @Type!='Company'
               BEGIN
                   THROW 52000, N'Niepoprawny typ klienta', 1
               END

           IF @Type='Individual' AND @IsEmployee=1 AND
              NOT EXISTS(SELECT * FROM CompanyCustomers WHERE
Name=@EmployeeCompany)
               BEGIN
                   THROW 52000, N'Niepoprawna firma pracownika', 1
               END

           IF @Type='Individual' AND EXISTS(SELECT * FROM
IndividualCustomers WHERE @Name = CustomerName)
               BEGIN
                   THROW 52000, N'Klient indywidualny już w bazie', 1
               END

           IF @Type='Company' AND EXISTS(SELECT * FROM CompanyCustomers
WHERE @Name = Name)
               BEGIN
                   THROW 52000, N'Klient firmowy już w bazie', 1
               END
           DECLARE @CustomerID int

           SET NOCOUNT ON;
           INSERT INTO Customers
           VALUES (@Email, @Phone, @City, @Street, @PostalCode)

           SET @CustomerID = SCOPE_IDENTITY()

           IF @Type='Company'
               BEGIN
                   INSERT INTO CompanyCustomers
                   VALUES (@CustomerID, @Name, @NIP)
               END
```

```
          IF @Type='Individual'
              BEGIN
                  INSERT INTO IndividualCustomers
                  VALUES (@CustomerID, @Name)
                  EXEC AddDiscount 1, @CustomerID
                  EXEC AddDiscount 2, @CustomerID

                  IF @IsEmployee=1
                      BEGIN
                          DECLARE @CompanyID int
                          SELECT @CompanyID = CustomerID FROM
CompanyCustomers WHERE Name=@EmployeeCompany

                          INSERT INTO Employees
                          VALUES (@CustomerID, @CompanyID)
                      END
              END
       END TRY
       BEGIN CATCH
           DECLARE @msg nvarchar(2048) = N'Błąd dodwania klienta: ' +
ERROR_MESSAGE();
           THROW 52000, @msg, 1;
       END CATCH
   END
GO
```

Procedura AddDiscount

Dodaje rabat o status Counting danemu klientowi

```
CREATE PROCEDURE AddDiscount
@Type int,
@CustomerID int
AS
   BEGIN
       BEGIN TRY
           IF @Type NOT IN (1, 2)
               BEGIN
                   THROW 52000, N'Zly typ znizki', 1
               END
           IF NOT EXISTS(SELECT CustomerID FROM Customers WHERE
CustomerID=@CustomerID)
               BEGIN
                   THROW 52000, N'Brak podanego klienta', 1
               END
           SET NOCOUNT ON

           INSERT INTO Discounts
           VALUES (@CustomerID, @Type, 0, NULL, NULL, 'Counting', 0, 0)

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

Procedura AddEmployeeToReservation

Dodaje rezerwację dokonaną przez pracownika firmy do tabeli ReservationEmployees

```
CREATE PROCEDURE AddEmployeeToReservation
@EmployeeName varchar(255),
@ReservationID int
AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT * FROM IndividualCustomers WHERE
CustomerName=@EmployeeName)
               BEGIN
                   THROW 52000, N'Brak pracownika w bazie klientów', 1
               END
           DECLARE @EmployeeID int = (SELECT CustomerID FROM
IndividualCustomers WHERE CustomerName=@EmployeeName)
           DECLARE @CompanyID int = (SELECT CustomerID FROM Reservations
WHERE ReservationID=@ReservationID)
           IF NOT EXISTS(SELECT * FROM Employees WHERE
EmployeeID=@EmployeeID AND CompanyID=@CompanyID)
               BEGIN
                   THROW 52000, N'Podana osoba nie jest pracownikiem
podanej firmy', 1
               END
           IF NOT EXISTS(SELECT * FROM CompanyReservation WHERE
ReservationID=@ReservationID)
               BEGIN
                   THROW 52000, N'Podana razerwacja nie jest na firmę', 1
               END

           INSERT INTO ReservationEmployees
           VALUES (@ReservationID, @EmployeeID)
       END TRY
       BEGIN CATCH
           DECLARE @msg nvarchar(2048) = N'Błąd dodawania pracownika do
rezerwacji: ' + ERROR_MESSAGE();
           THROW 52000, @msg, 1;
       END CATCH
   END
GO
```

Procedura AddMeal

Dodaje posiłek wraz z ceną do tabeli Meals

```
CREATE PROCEDURE AddMeal
    @MealName varchar(55),
    @Price int,
    @CategoryName varchar(255)
AS
    BEGIN
        BEGIN TRY
            IF EXISTS(SELECT * FROM Meals WHERE @MealName = NameMeals)
                BEGIN
                    THROW 52000, N'Posiłek jest już dodany', 1
                END
            IF NOT EXISTS(SELECT * FROM Category WHERE @CategoryName =
CategoryName)
                BEGIN
                    THROW 52000, N'Nie ma takiej kategorii ', 1
                END
            DECLARE @CategoryID int
            SELECT @CategoryID = CategoryID FROM Category WHERE
CategoryName = @CategoryName

            SET NOCOUNT ON;
            INSERT INTO Meals
            VALUES (@CategoryID, @MealName, @Price, 0)
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd dodwania posiłku: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1;
        END CATCH
    END
go
```

Procedura AddTable

Dodaje stolik wraz z ilością miejsc i lokalizacją do tabeli Tables

```
CREATE PROCEDURE AddTable
@Seats int,
@Location varchar(255)
AS
    BEGIN
        INSERT INTO Tables
        VALUES (@Seats, @Location)
    END
go
```

Procedura StartEmptyOrder

Dodaje nowe, puste zamówienie do tabeli Orders

```
CREATE PROCEDURE StartEmptyOrder
@CustomerID int,
@TakeawayDate datetime = NULL
AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT * FROM Customers WHERE
CustomerID=@CustomerID)
               BEGIN
                   THROW 52000, N'Brak klienta w bazie', 1
               END
           SET NOCOUNT ON

           DECLARE @DiscountID int
           DECLARE @DiscountValue float
           DECLARE @DiscountType int
           DECLARE @OrderID int
           IF EXISTS(SELECT * FROM Discounts WHERE Status='Active' AND
CustomerID=@CustomerID)
               BEGIN
                   SELECT TOP 1 @DiscountID = DiscountID, @DiscountValue =
DiscountValue, @DiscountType = DiscountType FROM Discounts
                           WHERE Status='Active' AND CustomerID=@CustomerID
                           ORDER BY DiscountValue DESC
               END
           ELSE
               SET @DiscountValue = 0
               SET @DiscountType = 0

           INSERT INTO Orders
           VALUES (@CustomerID, GETDATE(), @DiscountValue, 'Pending')

           SET @OrderID = SCOPE_IDENTITY()

           IF @TakeawayDate IS NOT NULL
               BEGIN
                   INSERT INTO TakeawayOrders
                   VALUES (@OrderID, @TakeawayDate)
               END

           RETURN @OrderID

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

Procedura AddToOrder

Dodaje pozycję do zamówienia do tabeli OrderContents

```
CREATE PROCEDURE AddToOrder
@OrderID int,
@MealName varchar(55),
@Quantity int
AS
    BEGIN
        BEGIN TRY
            IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID=@OrderID)
                BEGIN
                    THROW 52000, N'Brak zamówienia w bazie', 1
                END

            IF NOT EXISTS(SELECT * FROM Meals WHERE NameMeals=@MealName)
                BEGIN
                    THROW 52000, N'Brak posiłku w bazie', 1
                END
            SET NOCOUNT ON

            DECLARE @MealID int
            SELECT @MealID = MealID FROM Meals WHERE NameMeals=@MealName

            INSERT INTO OrderContents
            VALUES (@OrderID, @MealID, @Quantity)

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

Procedura CancelReservation

Anuluje podaną rezerwację

```
CREATE PROCEDURE CancelReservation
@ReservationID int
AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT * FROM Reservations WHERE
ReservationID=@ReservationID)
               BEGIN
                   THROW 52000, N'Brak podanej rezerwacji', 1
               END

           IF (SELECT Status FROM Reservations WHERE
ReservationID=@ReservationID)='Complete'
               BEGIN
                   THROW 52000, N'Rezerwacja już rozpatrzona', 1
               END
           SET NOCOUNT ON
           IF EXISTS(SELECT * FROM IndividualReservations WHERE
ReservationID=@ReservationID)
               BEGIN
                   DELETE FROM IndividualReservations
                   WHERE ReservationID=@ReservationID
               END

           ELSE IF EXISTS(SELECT * FROM CompanyReservation WHERE
ReservationID=@ReservationID)
               BEGIN
                   DELETE FROM ReservationEmployees
                   WHERE ReservationID=@ReservationID
                   DELETE FROM CompanyReservation
                   WHERE ReservationID=@ReservationID
               END

           DECLARE @OrderID int = (SELECT OrderID FROM Reservations WHERE
ReservationID=@ReservationID)

           DELETE FROM OrderContents
           WHERE OrderID=@OrderID

           DELETE FROM Orders
           WHERE OrderID=@OrderID

           DELETE FROM Reservations
           WHERE ReservationID=@ReservationID

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

Procedura CompleteOrder

Oznacza zamówienie jako zakończone oraz odpowiednio nalicza, deaktywuje i aktywuje rabaty

```
CREATE PROCEDURE CompleteOrder
@OrderID int
AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT OrderID FROM Orders WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Zamówienie nie istnieje', 1
               END
           SET NOCOUNT ON

           IF (SELECT Status FROM Orders WHERE OrderID=@OrderID)='Complete'
               BEGIN
                   THROW 52000, N'Zamówienie już potwierdzone', 1
               END

           DECLARE @CustomerID int
           SELECT @CustomerID=CustomerID FROM Orders WHERE OrderID=@OrderID

           DECLARE @Discount1ID int = (SELECT DiscountID FROM Discounts
WHERE CustomerID=@CustomerID AND DiscountType=1)
           DECLARE @Discount2ID int = (SELECT DiscountID FROM Discounts
WHERE CustomerID=@CustomerID AND DiscountType=2)

           UPDATE Orders
           SET Status='Complete'
           WHERE @OrderID=OrderID

           IF (SELECT EndDate FROM Discounts WHERE DiscountID=@Discount2ID)
> GETDATE()
               BEGIN
                   UPDATE Discounts
                   SET Status='Deactivated'
                   WHERE DiscountID=@Discount2ID
                   EXEC AddDiscount 2, @CustomerID
               END

           IF (SELECT SUM(M.Price * OC.Quantity * (1 - O.Discount)) as
OrderPrice
               FROM Orders O
               INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
               INNER JOIN Meals M on M.MealID = OC.MealID
               WHERE O.OrderID = @OrderID
               GROUP BY O.OrderID) >= (SELECT MinOrderValueForDiscount1
FROM Config WHERE ChangeDate IS NULL)
               BEGIN
                   UPDATE Discounts
                   SET OrdersAccumulated += 1
                   WHERE DiscountID=@Discount1ID AND Status='Counting'
               END
```

```
            UPDATE Discounts
            SET MoneyAccumulated += (SELECT SUM(M.Price * OC.Quantity * (1 -
O.Discount)) as OrderPrice
                                                    FROM Orders O
                                                    INNER JOIN
OrderContents OC on O.OrderID = OC.OrderID
                                                    INNER JOIN Meals M
on M.MealID = OC.MealID
                                                    WHERE O.OrderID =
@OrderID
                                                    GROUP BY O.OrderID)
            WHERE DiscountID=@Discount2ID AND Status='Counting'

            IF (SELECT OrdersAccumulated FROM Discounts WHERE
DiscountID=@Discount1ID) >= (SELECT OrdersForDiscount1 FROM Config WHERE
ChangeDate IS NULL)
                BEGIN
                    EXEC ActivateDiscount @Discount1ID
                END

            IF (SELECT MoneyAccumulated FROM Discounts WHERE
DiscountID=@Discount2ID) >= (SELECT OrdersValueForDiscount2 FROM Config
WHERE ChangeDate IS NULL)
                BEGIN
                    EXEC ActivateDiscount @Discount2ID
                END

            DECLARE @RestaurantNIP varchar(10) = (SELECT TOP 1 NIP FROM
Restaurant)
            INSERT INTO Invoices
            VALUES (@OrderID, @RestaurantNIP)

            IF EXISTS(SELECT * FROM Reservations WHERE OrderID=@OrderID)
                BEGIN
                    UPDATE Reservations
                    SET Status='Complete'
                    WHERE OrderID=@OrderID
                END

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

Procedura CreateCompanyReservation

Dodaje rezerwację na firmę do tabel CompanyReservation i Reservations, zwraca ID rezerwacji

```
CREATE PROCEDURE CreateCompanyReservation
@OrderID int,
@ArrivalDate datetime,
@Duration time,
@Seats int
AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Brak podanego zamówienia w bazie', 1
               END
           IF EXISTS(SELECT * FROM Reservations WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Istnieje już rezerwacja do podanego
zamówienia', 1
               END

           DECLARE @CompanyID int = (SELECT CustomerID FROM Orders WHERE
OrderID=@OrderID)

           IF NOT EXISTS(SELECT * FROM CompanyCustomers WHERE
CustomerID=@CompanyID)
               BEGIN
                   THROW 52000, N'Podane zamównienie nie zostało złożone
przez firmę', 1
               END

           IF EXISTS(SELECT * FROM TakeawayOrders WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Podane zamówienie zostało złożone na
wynos', 1
               END
           IF NOT EXISTS(SELECT * FROM getFreeTables(@ArrivalDate,
@Duration, @Seats))
               BEGIN
                   THROW 52000, N'Brak wolnych stolików', 1
               END

           DECLARE @TableID int
           SELECT TOP 1 @TableID = TableID FROM getFreeTables(@ArrivalDate,
@Duration, @Seats)

           INSERT INTO Reservations
           VALUES (@OrderID, @CompanyID, @TableID, GETDATE(), @ArrivalDate,
@Duration, @Seats, 'Pending')

           INSERT INTO CompanyReservation
           VALUES (SCOPE_IDENTITY())

           RETURN SCOPE_IDENTITY()

       END TRY
```

```
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd tworzenia rezerwacji: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1;
        END CATCH
    END
GO
```

Procedura CreateIndividualReservation

Dodaje rezerwację klienta indywidualnego do tabeli Reservations i IndividualReservations, zwraca ID rezerwacji

```
CREATE PROCEDURE CreateIndividualReservation
@OrderID int,
@ArrivalDate datetime,
@Duration time,
@Seats int,
@PaymentMethod varchar(255),
@Paid bit

AS
   BEGIN
       BEGIN TRY
           IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Brak podanego zamówienia w bazie', 1
               END
           IF EXISTS(SELECT * FROM Reservations WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Istnieje już rezerwacja do podanego
zamówienia', 1
               END
           DECLARE @CustomerID int = (SELECT CustomerID FROM Orders WHERE
OrderID=@OrderID)
           IF NOT EXISTS(SELECT * FROM IndividualCustomers WHERE
CustomerID=@CustomerID)
               BEGIN
                   THROW 52000, N'Podane zamównienie nie zostało złożone
przez indywidualnego klienta', 1
               END

           IF EXISTS(SELECT * FROM TakeawayOrders WHERE OrderID=@OrderID)
               BEGIN
                   THROW 52000, N'Podane zamówienie zostało złożone na
wynos', 1
               END
           IF NOT EXISTS(SELECT * FROM getFreeTables(@ArrivalDate,
@Duration, @Seats))
               BEGIN
                   THROW 52000, N'Brak wolnych stolików', 1
               END

           SET NOCOUNT ON


           DECLARE @TableID int
           SELECT TOP 1 @TableID = TableID FROM getFreeTables(@ArrivalDate,
@Duration, @Seats)

           INSERT INTO Reservations
           VALUES (@OrderID, @CustomerID, @TableID, GETDATE(),
@ArrivalDate, @Duration, @Seats, 'Pending')

           INSERT INTO IndividualReservations
           VALUES (SCOPE_IDENTITY(), @PaymentMethod, @Paid)

       END TRY
```

```
        BEGIN CATCH
            DECLARE @msg nvarchar(2048) = N'Błąd tworzenia rezerwacji: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1;
        END CATCH
    END
GO
```

Procedura CreateMenu

Ustawia podaną tabelę ID posiłków jako nowe menu, stare zapisuje do tabeli MenuHistory

```
CREATE PROCEDURE CreateMenu
@NewMenu Menu READONLY
AS
   BEGIN
       BEGIN TRY

           IF DATEDIFF(DAY, CAST(GETDATE() AS date), (SELECT TOP 1
IntroduceDate FROM CurrentMenu)) < 14
               BEGIN
                   THROW 52000, N'Zbyt malo czasu od poprzedniej zmiany
menu', 1
               END
           DECLARE @MenuLength int = (SELECT COUNT(*) FROM CurrentMenu)
           IF (SELECT COUNT(*) FROM @NewMenu INNER JOIN CurrentMenu ON
CurrentMenu.MealID=[@NewMenu].MealID) > @MenuLength / 2
               BEGIN
                   THROW 52000, N'Mniej niż połowa posiłków zmieniona w
stosunku do starego menu', 1
               END

           SET NOCOUNT ON
           DECLARE @DurationTime int = DATEDIFF(DAY, CAST(GETDATE() AS
date), (SELECT TOP 1 IntroduceDate FROM CurrentMenu))

           INSERT INTO MenuHistory
           SELECT MealID, IntroduceDate, @DurationTime FROM CurrentMenu

           DELETE FROM CurrentMenu

           INSERT INTO CurrentMenu
           SELECT CAST(GETDATE() AS date), MealID FROM @NewMenu

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

Procedura UpdateConfig

Umożliwia dokonywanie zmian wartości konfiguracyjnych takich jak: minimalna wartość zamówienia WZ, minimalna ilość zamówień WK, ustalona liczba zamówień Z1, określona kwota K1, procent zniżki R1, łączna kwota K2, jednorazowa zniżka R2, liczba dni D1

```
CREATE PROCEDURE UpdateConfig
@NewMinReservationValue float = NULL,
@NewMinOrdersForReservation int = NULL,
@NewOrdersForDiscount1 int = NULL,
@NewMinOrderValueForDiscount1 float = NULL,
@NewDiscount1Value float = NULL,
@NewOrdersValueForDiscount2 float = NULL,
@NewDiscount2Value float = NULL,
@NewDiscount2DurationDays int = NULL

AS
BEGIN
    BEGIN TRY
        DECLARE @MinReservationValue float
        DECLARE @MinOrdersForReservation int
        DECLARE @OrdersForDiscount1 int
        DECLARE @MinOrderValueForDiscount1 float
        DECLARE @Discount1Value float
        DECLARE @OrdersValueForDiscount2 float
        DECLARE @Discount2Value float
        DECLARE @Discount2DurationDays int

        SELECT TOP 1 @MinReservationValue = MinReservationValue,
                     @MinOrdersForReservation = MinOrdersForReservation,
                     @OrdersForDiscount1 = OrdersForDiscount1,
                     @MinOrderValueForDiscount1 =
MinOrderValueForDiscount1,
                     @Discount1Value = Discount1Value,
                     @OrdersValueForDiscount2 = OrdersValueForDiscount2,
                     @Discount2Value = Discount2Value,
                     @Discount2DurationDays = Discount2DurationDays
                     FROM Config
        ORDER BY IntroduceDate DESC

        UPDATE Config
        SET ChangeDate = GETDATE()
        WHERE IntroduceDate=(SELECT TOP 1 IntroduceDate FROM Config ORDER
BY IntroduceDate DESC)

        IF @NewMinReservationValue IS NOT NULL
            BEGIN
                SET @MinReservationValue = @NewMinReservationValue
            END

        IF @NewMinOrdersForReservation IS NOT NULL
            BEGIN
                SET @MinOrdersForReservation = @NewMinOrdersForReservation
            END

        IF @NewOrdersForDiscount1 IS NOT NULL
            BEGIN
                SET @OrdersForDiscount1 = @NewOrdersForDiscount1
```

```
            END

        IF @NewMinOrderValueForDiscount1 IS NOT NULL
            BEGIN
                SET @MinOrderValueForDiscount1 =
@NewMinOrderValueForDiscount1
            END

        IF @NewDiscount1Value IS NOT NULL
            BEGIN
                SET @Discount1Value = @NewDiscount1Value
            END

        IF @NewOrdersValueForDiscount2 IS NOT NULL
            BEGIN
                SET @OrdersValueForDiscount2 = @NewOrdersValueForDiscount2
            END

        IF @NewDiscount2Value IS NOT NULL
            BEGIN
                SET @Discount2Value = @NewDiscount2Value
            END

        IF @NewDiscount2DurationDays IS NOT NULL
            BEGIN
                SET @Discount2DurationDays = @NewDiscount2DurationDays
            END

        INSERT INTO Config
        VALUES (GETDATE(), NULL,
                @MinReservationValue,
                @MinOrdersForReservation,
                @OrdersForDiscount1,
                @MinOrderValueForDiscount1,
                @Discount1Value,
                @OrdersValueForDiscount2,
                @Discount2Value,
                @Discount2DurationDays)

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

Procedura UpdateMeal

Umożliwia zmianę parametrów posiłku lub całkowite wycofanie go, dawne parametry zapisuje w tabeli MealsHistory

```
CREATE PROCEDURE UpdateMeal
@MealName varchar(255),
@NewCategory varchar(255) = NULL,
@NewMealName varchar(255) = NULL,
@NewPrice money = NULL,
@Discontinued bit = 0
AS
   BEGIN
       BEGIN TRY
           DECLARE @MealID int = (SELECT MealID FROM Meals WHERE
NameMeals=@MealName)
           IF NOT EXISTS(SELECT * FROM Category WHERE
CategoryName=@NewCategory)
               BEGIN
                   THROW 52000, N'Brak podanej kategorii', 1
               END
           IF (SELECT Discontinued FROM Meals WHERE MealID=@MealID)=1
               BEGIN
                   THROW 52000, N'Posiłek wycofany', 1
               END
           IF @NewCategory IS NULL AND @NewMealName IS NULL AND @NewPrice
IS NULL AND @Discontinued=0
               BEGIN
                   THROW 52000, N'Brak zmian', 1
               END

           SET NOCOUNT ON
           INSERT INTO MealsHistory
           SELECT CAST(GETDATE() AS date), @MealID, NameMeals, Price,
CategoryID FROM Meals WHERE MealID=@MealID

           IF @NewCategory IS NOT NULL
               BEGIN
                   UPDATE Meals
                   SET CategoryID=(SELECT CategoryID FROM Category WHERE
CategoryName=@NewCategory)
                   WHERE MealID=@MealID
               END

           IF @NewMealName IS NOT NULL
               BEGIN
                   UPDATE Meals
                   SET NameMeals=@NewMealName
                   WHERE MealID=@MealID
               END

           IF @NewPrice IS NOT NULL
               BEGIN
                   UPDATE Meals
                   SET Price=@NewPrice
                   WHERE MealID=@MealID
```

```
            END

        IF @Discontinued=1
            BEGIN
                UPDATE Meals
                SET Discontinued=1
                WHERE MealID=@MealID
            END

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

Widoki

Widok Menu

Wyświetla aktualne menu

```
CREATE VIEW Menu AS
SELECT M.Name, M.Price, C.CategoryName FROM Meals M
INNER JOIN CurrentMenu CM ON M.MealID = CM.MealID
INNER JOIN Categories C ON C.CategoryID = M.CategoryID
```

Widok ReservedSeatsPerDay

Wyświetla ilość zarezerwowanych miejsc każdego dnia

```
CREATE VIEW ReservedSeatsPerDay AS
SELECT YEAR(ArrivalDate) AS Y, MONTH(ArrivalDate) AS M, DAY(ArrivalDate) AS
D, SUM(NumberOfCustomers)
GROUP BY Y, M, D
```

Widok AverageReservedSeats

Wyświetla średnią ilość zarezerwowanych miejsc każdego dnia

```
CREATE VIEW AverageReservedSeats AS
SELECT YEAR(ArrivalDate) AS Y, MONTH(ArrivalDate) AS M, DAY(ArrivalDate) AS
D, AVG(NumberOfCustomers)
GROUP BY Y, M, D
```

Widok OrdersForToday

Wyświetla zamówienia złożone dnia dzisiejszego

```
CREATE VIEW OrdersForToday AS
SELECT  M.Name, M.Price,AC.Quantity, o.OrderDate FROM Orders AS o
    INNER JOIN OrderContents AC ON AC.OrderID = o.OrderID
        INNER JOIN Meals AS M ON M.MealID = AC.MealID
WHERE DATEDIFF(DAY,GETDATE(),o.OrderDate) = 0
```

Widok MealInfo

Wyświetla informacje o daniu (kategoria, nazwa, opis kategorii)

```
CREATE VIEW MealInfo AS
    SELECT C.CategoryName, M.Name, C.Description FROM Meals AS M
        INNER JOIN Categories C on M.CategoryID = C.CategoryID
GROUP BY MealID
```

Widok TotalValues

Wyświetla

```
create view TotalValues as
SELECT O.OrderID,C.CustomerID,SUM(OC.Quantity * M.Price) * (1-ISNULL(
    (SELECT TOP 1 CAST(D.Value AS FLOAT)/100 FROM Discount D
    WHERE D.StartingDate <= GETDATE()
    ORDER BY D.StartingDate DESC))) AS TotalValue FROM Customers C
        INNER JOIN Orders O ON C.CustomerID = O.CustomerID
            INNER JOIN OrderContents OC ON O.OrderID = OC.OrderID
                INNER JOIN Meals M ON OC.MealID = M.MealID
                    LEFT JOIN Discounts D ON C.CustomerID = D.CustomerID
GROUP BY C.CustomerID, O.OrderID, D.StartingDate, D.Type
```

Widok NumberOfOrders

Wyświetla liczbę złożonych zamówień

```
CREATE VIEW NumberOfOrders AS
SELECT C.customerID, COUNT(O.OrderID) AS NumbersOrders FROM Customers C
    INNER JOIN Orders O on C.CustomerID = O.CustomerID
GROUP BY C.CustomerID
```

Widok RankOfMeals

Wyświetla liczbę sprzedanych dań (w ciągu ostatnich 2 tyg) wraz z ich nazwą

```
CREATE VIEW RankOfMeals as
SELECT M.Name, COUNT(OC.Quantity) as OrdersQuantity FROM Orders O
    INNER JOIN OrderContents OC on O.OrderID = OC.OrderID
        INNER JOIN Meals M on M.MealID = OC.MealID
WHERE DATEDIFF(DAY,GETDATE(),O.OrderDate) <= 14
GROUP BY M.MealID,M.Name
```

Widok OrdersToPay

Wyświetla nieopłacone zamówienia

```
CREATE VIEW OrdersToPay AS
SELECT O.OrderID, R.CustomerID, O.OrderDate,FROM Orders O
    INNER JOIN Reservations R ON O.OrderID = R.OrderID
        INNER JOIN IndividualReservation IR WHERE IR.ReservationID =
R.ReservationID
WHERE IR.Prepaid = 0
```

Widok OrdersInfo

Wyświetla informacje o zamówieniu (ID, zamówione dania, ID klienta)

```
CREATE VIEW OrdersInfo AS
SELECT O.OrderID,M.Name,O.CustomerID FROM Orders O
    INNER JOIN OrderContents OC ON OC.OrderID = O.OrderID
        INNER JOIN Meals M ON M.MealID = OC.MealID
GROUP BY O.OrderID,O.CustomerID,M.Name
```

Widok LastVisibleMeals

Wyświetla wycofane dania posortowane po dacie dokonanych zmian

```
CREATE VIEW LastVisableMeals AS
SELECT M.Name,M.Price,ChangeDate FROM MealsHistory
    INNER JOIN Meals M ON M.MealID = OC.MealID
ORDER BY ChangeDate DESC
GROUP By M.MealID
```

Widok MaxPrice

Wyświetla danie z historii dań o największej cenie

```
CREATE VIEW MaxPrice AS
SELECT MAX(Price),MealID FROM MealsHistory
GROUP BY MealID
```

Widok MinPrice

Wyświetla danie z historii dań o najmniejszej cenie

```
CREATE VIEW MinPrice AS
SELECT MIN(Price),MealID FROM MealsHistory
GROUP BY MealID
```

Widok AvgPrice

Wyświetla średnią cenę dań z historii dań

```
CREATE VIEW AvgPrice AS
SELECT AVG(Price),MealID FROM MealsHistory
GROUP BY MealID
```

Widok DiscountPerYear

Wyświetla zniżki (i ich typ) wykorzystane w danym roku

```
CREATE VIEW DiscountPerYear AS
    SELECT DISTINCT DiscountID,
           Type,
           YEAR(OrderDate) as Year
    FROM Discounts D
    INNER JOIN IndividualCustomers I on I.CustomerID = D.CustomerID
    INNER JOIN Customer C on I.CustomerID = C.CustomerID
    INNER JOIN Orders O on C.CustomerID = O.CustomerID
    GROUP BY Year, Type
go
```

Widok DiscountMonthly

Wyświetla wszystkie zniżki (ich typ) przyznane w każdym miesiącu

```
CREATE VIEW DiscountMonthly AS
    SELECT Type,
           YEAR(OrderDate) as Year,
           MONTH(OrderDate) as Month
    FROM Discounts D
    INNER JOIN IndividualCustomers I on I.CustomerID = D.CustomerID
    INNER JOIN Customer C on I.CustomerID = C.CustomerID
    INNER JOIN Orders O on C.CustomerID = O.CustomerID
    GROUP BY Year, Month, Type
go
```

Widok DiscountCountMonthly

Wyświetla ilość zniżek przyznanych w danym roku i miesiącu

```
CREATE VIEW DiscountCountMonthly AS
    SELECT Type,
           YEAR(OrderDate) as Year,
           MONTH(OrderDate) as Month
           COUNT(D.DiscountID) as DiscountCount
    FROM Discounts D
    INNER JOIN IndividualCustomers I on I.CustomerID = D.CustomerID
    INNER JOIN Customer C on I.CustomerID = C.CustomerID
    INNER JOIN Orders O on C.CustomerID = O.CustomerID
    GROUP BY Year, Month, Type
go
```

Widok CustomerInfo

Wyświetla informacje o kliencie i ilości jego zamówień

```
CREATE VIEW CustomerInfo AS
    SELECT C.CustomerID,
           C.City + ', ' + C.PostalCode + ', ' + C.Street as address,
           COUNT(O.OrderID)
    FROM Customers C
    INNER JOIN Orders O on O.CustomerID = C.CustomerID
    GROUP BY C.CustomerID, C.City + ', ' + C.PostalCode + ', ' + C.Street
go
```

Widok CompanyCustomerOrders

Wyświetla ile zamówień złożyli poszczególni pracownicy firm

```
CREATE VIEW CompanyCustomersOrders AS
    SELECT C.CustomerID,
           COUNT(O.OrderID)
    FROM Customers C
    INNER JOIN Orders O on O.CustomerID = C.CustomerID
    LEFT JOIN CompanyCustomers CC on CC.CustomerID = C.CustomerID
    GROUP BY C.CustomerID
go
```

Indeksy

- Category_index:

Indeks dla tabeli Category, zawierający kolumny CategoryName i CategoryID. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny CategoryName lub CategoryID. Może być przydatny np. do szybkiego wyświetlenia wszystkich potraw z danej kategorii.

```
create index Category_index
    on Category (CategoryName, CategoryID)
go
```

- CompanyCustomers_index:

Indeks dla tabeli CompanyCustomers, zawierający kolumny NIP, Name i CustomerID. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny NIP, Name lub CustomerID. Może być przydatny np. do szybkiego wyświetlenia wszystkich faktur wystawionych dla danej firmy.

```
create index CompanyCustomers_index
    on CompanyCustomers (NIP, Name, CustomerID)
go
```

- Config_index:

Indeks dla tabeli Config, zawierający kolumny IntroduceDate i ChangeDate.
Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny IntroduceDate lub ChangeDate. Może być przydatny np. do szybkiego wyświetlenia wszystkich zmian wprowadzonych w danym okresie.

```
create index Config_index
    on Config (IntroduceDate,ChangeDate)
go
```

- CurrentMenu_index:

Indeks dla tabeli CurrentMenu, zawierający kolumnę IntroduceDate.
Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny IntroduceDate. Może być przydatny np. do szybkiego wyświetlenia aktualnego menu w danym dniu.

```
create index CurrentMenu_index
    on CurrentMenu (IntroduceDate)
go
```

- Customer_index:

Indeks dla tabeli Customers, zawierający kolumny Phone, Email i CustomerID.
Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny Phone, Email lub CustomerID. Może być przydatny np. do szybkiego wyświetlenia wszystkich zamówień danego klienta.

```
create index Customer_index
    on Customers (Phone,Email,CustomerID)
go
```

- Discounts_index:

Indeks dla tabeli Discounts, zawierający kolumny StartDate, Status i DicountID. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny StartDate, Status lub DiscountID. Może się przydać do np. szybkiego znalezienia aktualnej zniżki.

```
create index Discounts_index
    on Discounts (Status,StartDate, DiscountID)
go
```

- Customer_info:

Indeks dla tabeli IndividualCustomers, zawierający kolumny CustomerName i CustomerID. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny CustomerName lub CustomerID. Może się przydać do szybkiego wyszukania danego kilenta.

```
create index Customer_info
    on IndividualCustomers (CustomerName,CustomerID)
go
```

- Paid_index:

Indeks dla tabeli IndividualReservations, zawierający kolumnę Paid. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny Paid. Może się przydać do znalezienia np. nie opłaconych rezerwacji.

```
create index Paid_index
    on IndividualReservations (Paid)
go
```

- Price_meals_index:

Indeks dla tabeli Meals, zawierający kolumnę Price.
Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny Price. Może się przydać do wyszukania potrawy w danym zakresie cenowym.

```
create index Price_meals_index
    on Meals (Price)
go
```

- Meals_Change_index:

Indeks dla tabeli MealsHistory, zawierający kolumnę ChangeDate. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny ChangeDate. Może się przydać w zapytaniach o np. osotanią datę gdy dane posiłki były w menu.

```
create index Meals_Change_index
    on MealsHistory (ChangeDate)
go
```

- Menu_date_index:

Indeks dla tabeli MenuHistory, zawierający kolumny IntroduceDate i DurationDays. Ten indeks może poprawić wydajność zapytań, które filtrują lub sortują dane według kolumny Introduce.

```
create index Menu_date_index
    on MenuHistory (IntroduceDate,DurationDays)
go
```

- Order_quantity_index:

Indeks dla tabeli OrderContents, zawierający kolumny Quantity i MealID. Indeks ilości zamówień to sposób szybkiego wyszukania ilości danego posiłku, który został zamówiony w restauracji. Może być używany do zarządzania zapasami, analizowania danych sprzedaży lub generowania raportów.

```
create index Order_quantity_index
    on OrderContents (Quantity, MealID)
go
```

- Order_date_index:

Indeks dla tabeli Orders, zawierający kolumnę OrderDate.Może być używany do generowania raportów o liczbie zamówień złożonych w danym dniu lub analizowania trendów w zachowaniu klientów w czasie.

```
create index Order_date_index
    on Orders (OrderDate)
go
```

- Reservation_date_index:

Indeks dla tabeli Reservations, zawierający kolumny PlacementDate, OrderID, TableID i CustomerID. Może być używany do generowania raportów o liczbie rezerwacji złożonych w danym dniu lub analizowania trendów w zachowaniu klientów w czasie.

```
create index Reservation_date_index
    on Reservations (PlacementDate, OrderID, TableID, CustomerID)
go
```

- Seats_index:

Indeks dla tabeli Tables, zawierający kolumnę Seats. Może być używany do różnych celów, takich jak planowanie rozmieszczenia gości czy analizowanie wolnych miejsc przy poszczególnych stołach.

```
create index Seats_index
    on Tables (Seats)
go
```

- Takeaway_Orders_index:

Indeks dla tabeli TakeawayOrders, zawierający kolumny TakeawayDate i OrderID. Indeks zamówień na wynos to sposób szybkiego wyszukania zamówień na podstawie daty odbioru. Na przykład, jeśli restauracja chce dowiedzieć się, ile zamówień na wynos zostało złożonych w danym dniu, może użyć indeksu zamówień na wynos, aby szybko znaleźć odpowiednie informacje.

```
create index Takeaway_Orders_index
    on TakeawayOrders (TakeawayDate, OrderID)
go
```

Triggery

Trigger Tables_Person_Reservation - to procedura triggerowa, która jest aktywowana po wstawieniu lub aktualizacji danych w tabeli Reservations. Sprawdza, czy liczba gości przy rezerwacji jest większa niż liczba miejsc siedzących przy stole. Jeśli tak, zostaje rzucony błąd 5011.

```
CREATE trigger Tables_Person_Rser_TR
    on Reservations
    for insert, update
    as
    begin
        SET NOCOUNT ON;
            if(SELECT Top 1 ReservationID from
            (SELECT COUNT(T.Seats) as PersonNumber,R.ReservationID,
R.NumberOfCustomers as Number
            from Reservations R inner join Tables T on R.TableID =
T.TableID
            GROUP BY R.ReservationID,R.NumberOfCustomers) as PR where
PersonNumber > Number)
            is not null
        begin
            throw 5011,'There is more people than seats', 1;
        end
end
go
```

Trigger Reservations_TR - to procedura triggerowa, która jest aktywowana po wstawieniu lub aktualizacji danych w tabeli Reservations. Sprawdza, czy liczba gości przy rezerwacji jest większa niż jeden oraz czy wartość zamówienia lub liczba zamówień klienta jest wystarczająco duża, aby zarezerwować stolik. Jeśli nie, zostaje rzucony błąd 52000.

```
create trigger Tables_Person_Rser_TR
    on Reservations
    for insert, update
    as
    begin
        SET NOCOUNT ON;
            if(SELECT Top 1 ReservationID from
            (SELECT COUNT(T.Seats) as PersonNumber,R.ReservationID,
R.NumberOfCustomers as Number
            from Reservations R inner join Tables T on R.TableID = T.TableID
            GROUP BY R.ReservationID,R.NumberOfCustomers) as PR where
PersonNumber > Number)
            is not null
        begin
            throw 2011,'There is more people than seats', 1;
        end
end
```

Trigger Sea_Food_Check_TR - to procedura triggerowa, która jest aktywowana po wstawieniu danych do tabeli OrderContents. Sprawdza, czy w kategorii "SeaFood" zostało zamówione danie na dwa lub więcej dni przed rezerwacją lub zamówieniem na wynos w dniu wtorku, piątku lub soboty. Jeśli tak, zostaje rzucony błąd 52201.

```
CREATE trigger Sea_Food_Check_TR
    on OrderContents
    for insert
    as begin
    set NOCOUNT ON;

    IF(Select Top 1 O.OrderID
       From OrderContents inner join Meals M on M.MealID =
OrderContents.MealID
       inner join Category C on C.CategoryID = M.CategoryID
       inner join Orders O on OrderContents.OrderID = O.OrderID
       inner join TakeawayOrders T on O.OrderID = T.OrderID
       INNER JOIN Reservations R on O.OrderID = R.OrderID
       INNER JOIN TakeawayOrders TaO on O.OrderID = TaO.OrderID
       where CategoryName like 'SeaFood'
        and ((datename(weekday, O.OrderDate) like 'Thursday' and
datediff(day, O.OrderDate, R.ArrivalDate) <= 2)
           or (datename(weekday, O.OrderDate) like 'Friday' and
datediff(day, O.OrderDate, R.ArrivalDate) <= 3)
           or (datename(weekday, O.OrderDate) like 'Saturday' and
datediff(day, O.OrderDate, R.ArrivalDate) <= 4)
           or (datename(weekday, O.OrderDate) like 'Thursday' and
datediff(day, O.OrderDate, TaO.TakeawayDate) <= 2)
           or (datename(weekday, O.OrderDate) like 'Friday' and
datediff(day, O.OrderDate, TaO.TakeawayDate) <= 3)
           or (datename(weekday, O.OrderDate) like 'Saturday' and
datediff(day, O.OrderDate, TaO.TakeawayDate) <= 4)
           )
    ) is not null
    BEGIN;
        THROW 52201, 'This meals have to be ordered early', 1
    END
end
go
```

Trigger Invalid_Menu_Tr- to procedura triggerowa, która jest aktywowana po aktualizacji danych w tabeli OrderContents. Sprawdza, czy danie, które jest próbowane dodać do zamówienia, istnieje w bieżącym menu. Jeśli nie, zostaje rzucony błąd 52392.

```
CREATE TRIGGER Invalid_Menu_TR
ON OrderContents
AFTER UPDATE
AS
BEGIN
    IF EXISTS (SELECT TOP 1 OC.MealID, CM.MealID AS OrderMeal
               FROM OrderContents OC
               LEFT JOIN CurrentMenu CM ON OC.MealID = CM.MealID
               WHERE CM.MealID IS NULL)
    BEGIN
        THROW 52392, 'The dish is included in an unfinished order.', 1;
    END
END
```

Trigger Correct_Discount_TR- to procedura triggerowa, która jest aktywowana po wstawieniu danych do tabeli Discounts. Sprawdza, czy wartość rabatu jest pomiędzy 0 a 1. Jeśli nie, zostaje rzucony błąd 16 z komunikatem "DiscountValue is not correction".

```
create trigger Correct_Discount_TR on Discounts
for insert
as
    begin
    if (select DiscountValue from inserted) < 0
        or (select DiscountValue from inserted) > 1
    BEGIN
    RAISERROR('Discount1Valuefrom is not correction', 16, 1)
    END
    end
go
```

Trigger Change_Menu_TR to procedura triggerowa, która jest aktywowana po aktualizacji danych w tabeli CurrentMenu. Pobiera identyfikator aktualnie dodawanego dania oraz datę ostatniego wprowadzenia dania do menu z tabeli MenuHistory. Następnie sprawdza, czy danie zostało już dodane do menu w dacie ostatniego wprowadzenia. Jeśli tak, zostaje rzucony błąd 5231 z komunikatem "Meal cannot be added to the menu because it has already been served". Trigger ten może być używany do zapobiegania sytuacji, w której danie, które już było podawane w menu, jest próbowane ponownie dodawać do menu. Może to być szczególnie przydatne w restauracji, która chce zapewnić, że menu jest zawsze aktualne i uniknąć sytuacji, w której dania są podawane w kilku menu pod rząd.

```
CREATE TRIGGER Change_Menu_TR
ON CurrentMenu
AFTER UPDATE
AS
BEGIN
    DECLARE @current_meal_id INT;

    DECLARE @last_introduce_date DATE;

    SET @current_meal_id = (SELECT MealID FROM inserted);

    set @last_introduce_date = (select Top 1 IntroduceDate from MenuHistory
order by IntroduceDate desc)

    begin

    IF EXISTS (SELECT 1 FROM MenuHistory WHERE MealID = @current_meal_id
and MenuHistory.IntroduceDate = @last_introduce_date)

        Throw 5231,'Meal cannot be added to the menu because it has already
been served', 1
    END
END
go
```

Trigger Config_TR - to procedura triggerowa, która jest aktywowana po wstawieniu lub aktualizacji danych w tabeli Config. Sprawdza, czy data ostatniej zmiany w konfiguracji jest wcześniejsza niż data wprowadzenia zmiany do konfiguracji. Jeśli tak, rzuca błąd 5534 z komunikatem "Dates configuration is not correct!".

```
CREATE trigger Config_TR
    on Config
    for insert,update
    as
    begin
        if(Select TOP 1 ChangeDate from Config
            where (DATEDIFF(day,IntroduceDate,ChangeDate) < 0) AND
ChangeDate IS NOT NULL) is not null
--          if (SELECT TOP 1 ChangeDate FROM Config WHERE )
        begin
            throw 5534, 'Dates configuration is not correct!',1
        end
end
go
```

Uprawnienia i role

1. Rola Worker:

- GRANT EXECUTE ON AddEmployeeToReservation TO Worker; - Ten grant umożliwia pracownikowi wykonanie procedury składowanej AddEmployeeToReservation. Jest to używane przez pracowników danej firmy do dodawania pracowników do dokonanych rezerwacji.

```
GRANT EXECUTE ON AddEmployeeToReservation TO Worker;
```

- GRANT SELECT ON StartEmptyOrder TO Worker - Ten grant umożliwia pracownikowi wybór z tabeli StartEmptyOrder. Jest to używane przez pracowników do pobierania informacji o zamówieniach, które są w toku.

```
GRANT SELECT ON StartEmptyOrder TO Worker;
```

- GRANT SELECT ON Orders TO Worker - Ten grant umożliwia pracownikowi wybór z tabeli zamówień. Jest to używane przez pracowników do pobierania informacji o złożonych zamówieniach.

```
GRANT SELECT ON Orders TO Worker;
```

- GRANT EXECUTE ON GetOrderValue TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury GetOrderValue.

```
GRANT EXECUTE ON GetOrderValue TO Worker;
```

- GRANT SELECT ON GetClientOrders TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wyświetlanie (odczyt) danych z procedury GetClientOrders.

```
GRANT SELECT ON GetClientOrders TO Worker;
```

- GRANT SELECT, UPDATE ON Reservations TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wyświetlanie (odczyt) oraz modyfikowanie (aktualizowanie) danych z tabeli Reservations.

```
GRANT SELECT, UPDATE ON Reservations TO Worker;
```

- GRANT SELECT ON Tables TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wyświetlanie (odczyt) danych z tabeli Tables.

```
GRANT SELECT ON Tables TO Worker;
```

- GRANT EXECUTE ON ActivateDiscount TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury ActivateDiscount.

```
GRANT EXECUTE ON ActivateDiscount TO Worker;
```

- GRANT EXECUTE ON AddCategory TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury AddCategory.

```
GRANT EXECUTE ON AddCategory TO Worker;
```

- GRANT SELECT ON OrdersInfo TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "OrdersInfo" w bazie danych restauracji. Oznacza to, że pracownik może wyświetlić sobie pozycję dań danego zamówienia.

```
grant select on OrdersInfo to worker
```

- "GRANT SELECT ON OrdersToPay TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "OrdersToPay" w bazie danych restauracji. Oznacza to, że pracownik może wyświetlić sobie zamówienia do zapłaty

```
grant select on OrdersToPay to worker
```

- "GRANT SELECT ON CustomerInfo TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "CustomerInfo" w bazie danych restauracji. Oznacza to, że użytkownik może wyświetlać dane klientów restauracji.

```
grant select on CustomerInfo to worker
```

- "GRANT SELECT ON OrdersForToday TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "OrdersForToday" w bazie danych restauracji. Oznacza to, że użytkownik może wyświetlać zamówienia danego dnia.

```
grant select on OrdersForToday to worker
```

- "GRANT SELECT ON NumberOfOrders TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "NumberOfOrders" w bazie danych restauracji. Oznacza to, że użytkownik może wyświetlać liczbę zamówień.

```
grant select on NumberOfOrders to worker
```

- "GRANT SELECT ON SeeCurrentMenu TO worker" - Ta komenda pozwala użytkownikowi o nazwie "worker" na wykonywanie operacji SELECT na tabeli "SeeCurrentMenu" w bazie danych restauracji. Oznacza to, że użytkownik może wyświetlać dane menu na obencą datę.

```
grant select on SeeCurrentMenu to worker
```

- GRANT EXECUTE ON AddCustomer TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury AddCustomer.

```
GRANT EXECUTE ON AddCustomer TO Worker;
```

- GRANT EXECUTE ON AddMeal TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury AddMeal.

```
GRANT EXECUTE ON AddMeal TO Worker;
```

- GRANT EXECUTE ON AddToOrder TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury AddToOrder.

```
GRANT EXECUTE ON AddToOrder TO Worker;
```

- GRANT EXECUTE ON CancelReservation TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury CancelReservation.

```
GRANT EXECUTE ON CancelReservation TO Worker;
```

- GRANT EXECUTE ON CompleteOrder TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury CompleteOrder.

```
GRANT EXECUTE ON CompleteOrder TO Worker;
```

- GRANT EXECUTE ON CreateCompanyReservation TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury CreateCompanyReservation.

```
GRANT EXECUTE ON CreateCompanyReservation TO Worker;
```

- GRANT EXECUTE ON CreateIndividualReservation TO Worker - to polecenie pozwala użytkownikowi o nazwie "Worker" na wykonywanie procedury CreateIndividualReservation

```
GRANT EXECUTE ON CreateIndividualReservation TO Worker;
```

- GRANT SELECT ON Config TO Worker - pozwala pracownikowi na wyświetlenie danych z tabeli Config.

```
GRANT SELECT ON Config TO Worker;
```

- GRANT SELECT ON Discounts TO Worker - pozwala pracownikowi na wyświetlenie danych z tabeli Discounts.

```
GRANT SELECT ON Discounts TO Worker;
```

- GRANT SELECT ON Reservations TO Worker - pozwala pracownikowi na wyświetlenie danych z tabeli Reservations.

```
GRANT SELECT ON Reservations TO Worker;
```

- GRANT SELECT, UPDATE ON CurrentMenu TO Worker - pozwala pracownikowi na wyświetlenie oraz aktualizowanie danych z tabeli CurrentMenu.

```
GRANT SELECT, UPDATE ON CurrentMenu TO Worker;
```

- GRANT SELECT, UPDATE ON OrderContents TO Worker - pozwala pracownikowi na wyświetlenie oraz aktualizowanie danych z tabeli OrderContents.

```
GRANT SELECT, UPDATE ON OrderContents TO Worker;
```

- GRANT SELECT ON CompanyCustomers TO Worker - pozwala pracownikowi na wyświetlenie danych z tabeli CompanyCustomers.

```
GRANT SELECT ON CompanyCustomers TO Worker;
```

- GRANT SELECT ON Customers TO Worker - pozwala pracownikowi na wyświetlenie danych z tabeli Customers.

```
GRANT SELECT ON Customers TO Worker;
```

- GRANT EXECUTE ON CreateMenu TO Worker - pozwala pracownikowi na wykonanie procedury CreateMenu.

```
GRANT EXECUTE ON CreateMenu TO Worker;
```

- grant select on LastVisibleMeals to worker – przy tworzeniu menu pomaga użytkownikowi na wybranie odpowiednich posiłków.

```
grant select on LastVisibleMeals to Worker
```

- GRANT SELECT ON GetFreeTables TO Worker - pozwala pracownikowi na wyświetlenie danych z procedury GetFreeTables.

```
GRANT SELECT ON GetFreeTables TO Worker;
```

2. Rola Moderator:

- Grant execute on AddTable to Moderator – nadaje uprawnienie moderatorowi do dodawania stolików.

```
grant execute on AddTable to Moderator
```

- grant execute on AddCategory to Moderator - nadaje uprawnienie moderatorowi do dodawania kategorii w menu.

```
grant execute on AddCategory to Moderator
```

- "GRANT SELECT ON RankOfMeals TO Moderator" - Ten widok pozwala przechowywać informację o popularności poszczególnych dań w restauracji.

```
grant select on RankOfMeals to Moderator
```

- "GRANT SELECT ON CustomerInfo TO Moderator" - Ten widok przechowuje informacje o klientach restauracji takie jak adres, numery telefonów itp.

```
grant select on CustomerInfo to Moderator
```

- "GRANT SELECT ON MealInfo TO Moderator" – ten iwdok przechowuje informacje o daniach oferowanych w restauracji, takie jak składniki, cena, alergeny itp.

```
grant select on MealInfo to Moderator
```

- "GRANT SELECT ON LastVisibleMeals TO Moderator" – ten widok przechowuje informację o ostatnio wyświetlanych dań w menu restauracji.

```
grant select on LastVisibleMeals to Moderator
```

- grant execute on AddDiscount to Moderator – nadaje uprawnienie moderatorowi do dodawania zniżek.

```
grant execute on AddDiscount to Moderator
```

- grant execute on AddMeal to Moderator– nadaje uprawnienie moderatorowi do dodawania posiłków serwowanych w restauracji.

```
grant execute on AddMeal to Moderator
```

- grant execute on CreateMenu to Moderator– nadaje uprawnienie moderatorowi do tworzenia menu.

```
grant execute on CreateMenu to Moderator
```

- grant execute on ActivateDiscount to Moderator– nadaje uprawnienie moderatorowi do aktywacji zniżek.

```
grant execute on ActivateDiscount to Moderator
```

- grant execute on CancelReservation to Moderator– nadaje uprawnienie moderatorowi do anulowania rezerwacji.

```
grant execute on CancelReservation to Moderator
```

- grant execute on CreateCompanyReservation to Moderator – nadaje uprawnienie moderatorowi do tworzenia firmowych rezerwacji.

```
grant execute on CreateCompanyReservation  to Moderator
```

- grant execute on CreateIndividualReservation to Moderator– nadaje uprawnienie moderatorowi do indywidualnych rezerwacji

```
grant execute on CreateIndividualReservation to Moderator
```

- grant execute on AddToOrder to Moderator– nadaje uprawnienie moderatorowi do dodawania pozycji do zamówienia z menu.

```
grant execute on AddToOrder to Moderator
```

- grant select on CurrentMenu to Moderator– nadaje uprawnienie moderatorowi do dostępu do tabeli z aktualnym menu.

```
grant select on CurrentMenu to Moderator
```

- "GRANT SELECT ON AvgPrice TO Moderator" – ta informacja o średniej cenie dań oferowanych w restauracji, co mogłoby być przydatne dla menadżera lub moderatora do ustalenia cen w menu lub do planowania budżetu.

```
grant select on AvgPrice to Moderator
```

- grant select on Reservations to Moderator - nadaje uprawnienie moderatorowi do dostępu do tabeli z rezerwacjami.

```
grant select on Reservations to Moderator
```

- grant select on MenuHistory to Moderator nadaje uprawnienie moderatorowi do dostępu do tabeli z dawnymi menu.

```
grant select on MenuHistory to Moderator
```

- grant select on MealsHistory to Moderator nadaje uprawnienie moderatorowi do dostępu do tabeli z informacjami o datach występowania posiłków.

```
grant select on MealsHistory to Moderator
```

- grant select on Meals to Moderator nadaje uprawnienie moderatorowi do dostępu do tabeli z daniami.

```
grant select on Meals to Moderator
```

- grant select on Config to Moderator nadaje uprawnienie moderatorowi do dostępu do tabeli ze zmianami i danymi zniżkami, datami zmian, etc.

```
grant select on Config to Moderator
```

3. Rola Customer:

- GRANT SELECT ON CurrentMenu TO Customer - pozwala użytkownikowi o nazwie Customer na wybieranie danych z tabeli CurrentMenu.

```
GRANT SELECT ON CurrentMenu TO Customer
```

- GRANT EXECUTE ON AddToOrder TO Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury AddToOrder.

```
GRANT EXECUTE ON AddToOrder TO Customer
```

- GRANT EXECUTE ON CreateIndividualReservation TO Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury CreateIndividualReservation.

```
GRANT EXECUTE ON CreateIndividualReservation TO Customer
```

- GRANT EXECUTE ON CreateCompanyReservation TO Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury CreateCompanyReservation.

```
GRANT EXECUTE ON CreateCompanyReservation TO Customer
```

- GRANT EXECUTE ON getOrderValue to Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury getOrderValue.

```
GRANT EXECUTE ON getOrderValue to Customer
```

- GRANT SELECT ON getFreeTables to Customer - pozwala użytkownikowi o nazwie Customer na wybieranie danych z procedury getFreeTables.

```
GRANT SELECT ON getFreeTables to Customer
```

- GRANT SELECT ON Config to Customer - pozwala użytkownikowi o nazwie Customer na wybieranie danych z tabeli Config.

```
GRANT SELECT on Config to Customer
```

- GRANT EXECUTE ON StartEmptyOrder to Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury StartEmptyOrder.

```
GRANT EXECUTE on StartEmptyOrder to Customer
```

- GRANT EXECUTE ON AddEmployeeToReservation to Customer - pozwala użytkownikowi o nazwie Customer na wykonywanie procedury AddEmployeeToReservation.

```
GRANT EXECUTE on AddEmployeeToReservation to Customer
```

4. Rola Menadżer:

- grant execute on getAvgCurrentMenuPrice to Manager – pozwala uzyskać menadżerowi średnią cenę posiłków.

```
grant execute on getAvgCurrentMenuPrice to Manager
```

- grant select on getIndividualClientsWithMostReservations to Manager – pokazuje menadżerowi klientów zamawiających najwięcej razy.

```
grant select on getIndividualClientsWithMostReservations  to
Manager
```

- grant select on getMealsSoldAtLeastXTimes to Manager – pokazuje ilość sprzedanych posiłków i pozwala oszacować popyt.

```
grant select on getMealsSoldAtLeastXTimes to Manager
```

- grant select on getThisDayOrdersValue to Manager – daje dostęp do dziennych wpływów z zamówień

```
grant select on getThisDayOrdersValue to Manager
```

- grant select on getOrdersWithHigherValue to Manager – daje dostęp do największych złożonych zamówień

```
grant select on getOrdersWithHigherValue  to Manager
```

- grant select on getThisMonthOrdersValue to Manager – daje dostęp do miesięcznego wpływu finansowego z zamówień

```
grant select on getThisMonthOrdersValue  to Manager
```

- grant select on getWeeklyReservations to Manager- daje menadżerowi dostęp do tygodniowych rezerwacji w celu stworzenia faktury.

```
grant select on getWeeklyReservations to Manager
```

- grant select on getWeeklyClientDiscounts to Manager -daje menadżerowi dostęp do tygodniowych zniżek w celu stworzenia np. raportu.

```
grant select on getWeeklyClientDiscounts to Manager
```

- "GRANT SELECT ON ReservedSeatsPerDay TO Manager" – ten widok przechowuje informację o liczbie zarezerwowanych miejsc dla każdego dnia, co pozwalałoby menedżerowi na planowanie i przydzielanie personelu oraz zarządzanie przestrzenią.

```
grant select on ReservedSeatsPerDay to Manager
```

- "GRANT SELECT ON RankOfMeals TO Manager" – ten widok przechowuje informację o popularności poszczególnych dań w restauracji, co pozwalałoby menedżerowi na decyzje dotyczące menu i planowanie zakupów.

```
grant select on RankOfMeals to Manager
```

- "GRANT SELECT ON DiscountPerYear TO Manager" – przechowuje informację o rocznych promocjach i zniżkach w restauracji, co pozwalałoby menedżerowi na planowanie budżetu i podejmowanie decyzji marketingowych.

```
grant select on DiscountPerYear to Manager
```

- "GRANT SELECT ON CustomerInfo TO Manager" – przechowuje informacje o klientach restauracji takie jak adres, numery telefonów itp. Menedżer mógłby z niej korzystać do segmentacji klientów oraz do planowanie działań marketingowych

```
grant select on CustomerInfo to Manager
```

- "GRANT SELECT ON OrdersForToday TO Manager" – przechowuje informację o zamówieniach na dzisiaj, co pozwalałoby menedżerowi na planowanie pracy personelu i dostawy produktów.

```
grant select on OrdersForToday to Manager
```

- "GRANT SELECT ON DiscountMonthly TO Manager" – ten widok przechowuje informację o miesięcznych promocjach i zniżkach w restauracji, co pozwalałoby menedżerowi na planowanie budżetu i podejmowanie decyzji marketingowych.

```
grant select on DiscountMonthly to Manager
```

- grant execute on getThisYearTotalIncome to Manager - daje dostęp do rocznego wpływu finansowego z zamówień

```
grant execute on getThisYearTotalIncome to Manager
```