Projekt bazy danych

Podstawy Baz Danych 2021/22

Paweł Steczkiewicz, Szymon Słota, Mikołaj Wielgos

Spis treści

Funkcje

Opis użytkowników

- **klient indywidualny** może składać zamówienia, rezerwować stoliki oraz generować swoje dane (raporty, historie, rabaty)
- **klient firmowy** może rezerwować stoliki dla siebie jak i swoich pracowników oraz generować dane dla całej firmy (raporty, historie, rabaty)
- **pracownik** może przygotowywać menu, produkty wymagające importu, przeglądać dane dotyczące klientów (historia zamówień, rabaty, raporty), wystawiać faktury, zmieniać ceny produktów, generować raporty oraz statystyki, przydzielać klientom karty lojalnościowe

Funkcje użytkowników

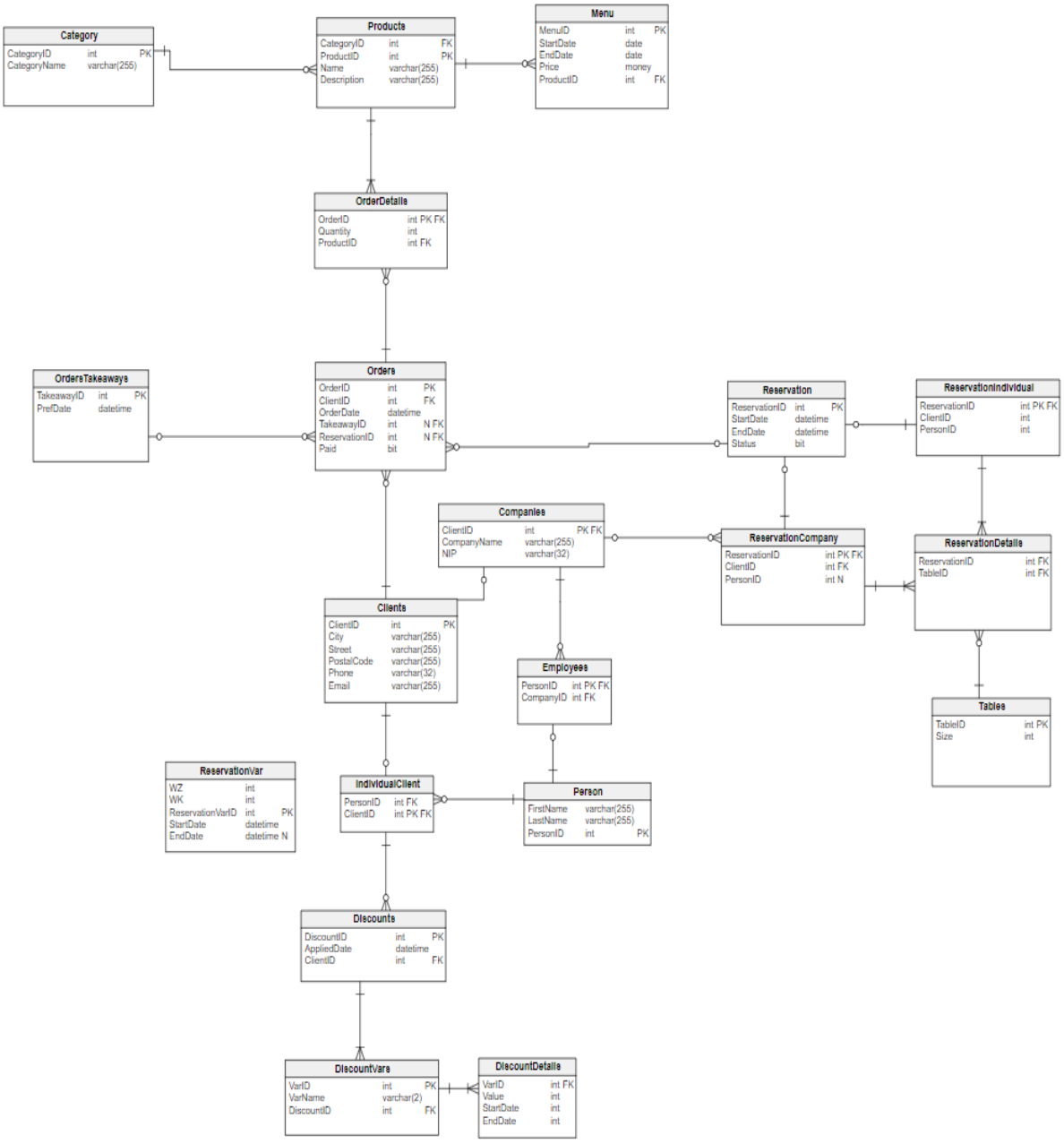
- **Klient indywidualny:**
    1. Złożenie zamówienia
    2. Zarezerwowanie stolika wraz z zamówieniem
    3. Wylistowanie historii własnych zamówień
    4. Wylistowanie własnych rabatów
    5. Wygenerowanie raportów dotyczących własnych zamówień
    6. Wygenerowanie raportów dotyczących własnych rabatów
- **Klient firmowy:**
    1. Wygenerowanie raportów dotyczących własnych zamówień
    2. Wygenerowanie raportów dotyczących własnych rabatów
    3. Złożenie rezerwacji na firmę
    4. Złożenie rezerwacji dla pracownika swojej firmy
    5. Wylistowanie historii zamówień firmy
- **Pracownik:**
    1. Wylistowanie przysługujących aktualnie klientowi rabatów
    2. Przygotowanie listy produktów wymagających importu
    3. Wylistowanie historii zamówień danego klienta
    4. Wylistowanie historii zamówień danej firmy
    5. Wystawienie faktury
    6. Wystawienie faktury zbiorczej
    7. Ustalenie aktualnego menu

8. Akceptacja zamówień
9. Akceptacja rezerwacji
10. Zmiana wartości Z1,K1,R1,K2,R2,D1,WK,WZ
11. Wylistowanie stanu stolików w danym czasie
12. Zmiana cen produktów
13. Generowanie raportów miesięcznych
14. Generowanie raportów tygodniowych
15. Generowanie raportów dotyczących rezerwacji
16. Generowanie raportów dotyczących rabatów
17. Generowanie raportów dotyczących menu
18. Generowanie statystyk zamówień dla klientów indywidualnych
19. Generowanie statystyk zamówień dla firm
20. Generowanie raportów dotyczących zamówień
21. Dodanie pozycji do menu
22. Usunięcie pozycji z menu
23. Dodanie produktu
24. Dodanie kategorii

Funkcje systemowe

1. Wyliczenie kosztu zamówienia
2. Sprawdzenie czy są dostępne stoliki w danym czasie
3. Sprawdzenie poprawności menu
4. Sprawdzenie czy klient może zarezerwować stolik
5. Przydzielanie rabatów

Schemat bazy danych


Category
CategoryID int PK
CategoryName varchar(255)
Products
CategoryID int FK
ProductID int PK
Name varchar(255)
Description varchar(255)
Menu
MenuID int PK
StartDate date
EndDate date
Price money
ProductID int FK
OrderDetails
OrderID int PK FK
Quantity int
ProductID int FK
OrdersTakeaways
TakeawayID int PK
PrefDate datetime
Orders
OrderID int PK
ClientID int FK
OrderDate datetime
TakeawayID int N FK
ReservationID int N FK
Paid bit
Reservation
ReservationID int PK
StartDate datetime
EndDate datetime
Status bit
ReservationIndividual
ReservationID int PK FK
ClientID int
PersonID int
Companies
ClientID int PK FK
CompanyName varchar(255)
NIP varchar(32)
ReservationCompany
ReservationID int PK FK
ClientID int FK
PersonID int N
ReservationDetails
ReservationID int FK
TableID int FK
Clients
ClientID int PK
City varchar(255)
Street varchar(255)
PostalCode varchar(255)
Phone varchar(32)
Email varchar(255)
Employees
PersonID int PK FK
CompanyID int FK
Tables
TableID int PK
Size int
ReservationVar
WZ int
WK int
ReservationVarID int PK
StartDate datetime
EndDate datetime N
IndividualClient
PersonID int FK
ClientID int PK FK
Person
FirstName varchar(255)
LastName varchar(255)
PersonID int PK
Discounts
DiscountID int PK
AppliedDate datetime
ClientID int FK
DiscountVars
VarID int PK
VarName varchar(2)
DiscountID int FK
DiscountDetails
VarID int FK
Value int
StartDate int
EndDate int

Opis tabel

Tabela Category

Reprezentacja kategorii pozycji w menu.

Klucz główny: CategoryID
Nazwę kategorii: CategoryName

```
CREATE TABLE Category
(
   CategoryID   int          NOT NULL,
   CategoryName varchar(255) NOT NULL,
   CONSTRAINT Category_pk PRIMARY KEY (CategoryID)
);
```

Tabela Clients

Reprezentacja klientów.

Klucz główny: ClientID
Adres e-mailowy: Email
Miasto klienta: City
Ulica: Street
Kod pocztowy: PostalCode
Numer telefonu: Phone
Warunki integralności:

- kod pocztowy postaci XX-XXX, gdzie X to cyfra od 0 do 9
  `CHECK (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]')`

- adres e-mailowy zawiera ‘@’
  `CONSTRAINT ValidEmail CHECK (Email LIKE '%@%')`

- adres e-mailowy unikalny
  `CONSTRAINT UniqueEmail UNIQUE (Email)`

- numer telefonu składa się tylko z cyfr od 0 do 9
  `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]')`

- numer telefonu unikalny
  `CONSTRAINT UniquePhone UNIQUE(Phone)`

```
CREATE TABLE Clients
(
   ClientID   int          NOT NULL,
   City       varchar(255) NOT NULL,
   Street     varchar(255) NOT NULL,
   PostalCode varchar(255) NOT NULL,
   Phone      varchar(32)  NOT NULL,
   Email      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 UniqueEmail UNIQUE (Email),
   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 UniquePhone UNIQUE (Phone),
   CONSTRAINT Clients_pk PRIMARY KEY (ClientID)
);
```

Tabela Companies

Reprezentacja firm.

Klucz główny i obcy: ClientID
Nazwę firmy: CompanyName
Numer NIP: NIP
Warunki integralności:

- numer nip składa się z cyfr od 0 do 9
  `CHECK (NIP LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')`

- numer nip unikalny
  `CONSTRAINT UniqueNIP UNIQUE (NIP)`

```
CREATE TABLE Companies
(
   ClientID    int          NOT NULL,
   CompanyName varchar(255) NOT NULL,
   NIP         varchar(32)  NOT NULL,
   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]'),
   CONSTRAINT UniqueNIP UNIQUE (NIP),
   CONSTRAINT Companies_pk PRIMARY KEY (ClientID)
)

ALTER TABLE Companies
   ADD CONSTRAINT Companies_Clients
       FOREIGN KEY (ClientID)
           REFERENCES Clients (ClientID)
```

Tabela DiscountDetails

Prezentacja szczegółowych danych na temat zniżek.

Klucz obcy: VarID
Wartość zniżki: Value
Datę rozpoczęcia zniżki: StartDate
Datę ważności zniżki: EndDate
Warunki integralności:

- EndDate musi być później od StartDate
  `CHECK (EndDate > StartDate)`

- Value zniżki między 0 a 1
  `CHECK (Value > 0 AND Value < 1)`

```
CREATE TABLE DiscountDetails
(
   VarID     int      NOT NULL,
   Value     float(2) NOT NULL,
   StartDate int      NOT NULL,
   EndDate   int      NOT NULL,
   CONSTRAINT ValidDate CHECK (EndDate > StartDate),
   CONSTRAINT ValidValue CHECK (Value > 0 AND Value < 1)
)

ALTER TABLE DiscountDetails
   ADD CONSTRAINT DiscountDetails_DiscountVars
       FOREIGN KEY (VarID)
           REFERENCES DiscountVars (VarID)
```

Tabela DiscountVars

Reprezentacja rodzajów zniżek.

Klucz główny: VarID
Klucz obcy: DiscountID
Nazwę zniżki: VarName

```
CREATE TABLE DiscountVars
(
   VarID      int         NOT NULL,
   VarName    varchar(2) NOT NULL,
   DiscountID int         NOT NULL,
   CONSTRAINT DiscountVars_pk PRIMARY KEY (VarID)
)

ALTER TABLE DiscountVars
   ADD CONSTRAINT DiscountVars_Discounts
       FOREIGN KEY (DiscountID)
           REFERENCES Discounts (DiscountID)
```

Tabela Discounts

Reprezentuje przyznane zniżki.

Klucz główny: DiscountID
Klucz obcy: ClientID
Datę przyznania danej zniżki: AppliedDate

```
CREATE TABLE Discounts
(
   DiscountID  int      NOT NULL,
   AppliedDate datetime NOT NULL,
   ClientID    int      NOT NULL,
   CONSTRAINT Discounts_pk PRIMARY KEY (DiscountID)
)

ALTER TABLE Discounts
   ADD CONSTRAINT Discounts_IndividualClient
       FOREIGN KEY (ClientID)
           REFERENCES IndividualClient (ClientID)
```

Tabela Employees

Reprezentuje pracowników firm będących klientami.

Klucz główny i obcy: PersonID
Drugi klucz obcy : CompanyID

```
CREATE TABLE Employees
(
   PersonID  int NOT NULL,
   CompanyID int NOT NULL,
   CONSTRAINT Employees_pk PRIMARY KEY (PersonID)
)

ALTER TABLE Employees
   ADD CONSTRAINT Employees_Companies
       FOREIGN KEY (CompanyID)
           REFERENCES Companies (ClientID)

ALTER TABLE Employees
   ADD CONSTRAINT Person_Employees
       FOREIGN KEY (PersonID)
           REFERENCES Person (PersonID)
```

Tabela IndividualClient

Reprezentacja indywidualnego klienta.

Klucz główny i obcy: ClientID
Drugi klucz obcy: PersonID

```
CREATE TABLE IndividualClient
(
   PersonID int NOT NULL,
   ClientID int NOT NULL,
   CONSTRAINT IndividualClient_pk PRIMARY KEY (ClientID)
)

ALTER TABLE IndividualClient
   ADD CONSTRAINT IndividualClient_Clients
       FOREIGN KEY (ClientID)
           REFERENCES Clients (ClientID)

ALTER TABLE IndividualClient
   ADD CONSTRAINT Person_IndividualClient
       FOREIGN KEY (PersonID)
           REFERENCES Person (PersonID)
```

Tabela Menu

Reprezentacja szczegółów poszczególnych menu.

Klucz główny: MenuID
Klucz obcy: ProductID
ID Menu: MenuID
ID produktu: ProductID
Cena produktu: Price
Data rozpoczęcia: StartDate
Data zakończenia: EndDate
Warunki integralności:

- cena nieujemna
  `CHECK (Price > 0)`

- data zakończenia późniejsza od daty rozpoczęcia
  `CHECK (EndDate > StartDate)`

```
CREATE TABLE Menu
(
   MenuID    int   NOT NULL,
   StartDate date  NOT NULL,
   EndDate   date  NULL,
   Price     money NOT NULL,
   ProductID int   NOT NULL,
   CONSTRAINT ValidMenuDate CHECK (EndDate > StartDate),
   CONSTRAINT ValidPrice CHECK (Price > 0),
   CONSTRAINT Menu_pk PRIMARY KEY (MenuID)
)

ALTER TABLE Menu
   ADD CONSTRAINT Menu_Products
       FOREIGN KEY (ProductID)
           REFERENCES Products (ProductID)
```

Tabela OrdersDetails

Reprezentacja szczegółów zamówień.

Klucz główny: OrderID
ID zamówienia: OrderID
ID produktu: ProductID
Ilość zamawianego produktu: Quantity
Warunki integralności:

- ilość zamawianego produktu > 0
  `CHECK (Quantity > 0)`

```
CREATE TABLE OrderDetails
(
   OrderID   int NOT NULL,
   Quantity  int NOT NULL,
   ProductID int NOT NULL,
   CONSTRAINT ValidQuantity CHECK (Quantity > 0),
   CONSTRAINT OrderDetails_pk PRIMARY KEY (OrderID)
)

ALTER TABLE OrderDetails
   ADD CONSTRAINT OrderDetails_Orders
       FOREIGN KEY (OrderID)
           REFERENCES Orders (OrderID)

ALTER TABLE OrderDetails
   ADD CONSTRAINT OrderDetails_Products
       FOREIGN KEY (ProductID)
           REFERENCES Products (ProductID)
```

Tabela Orders

Reprezentacja zamówień.

Klucz główny: OrderID
Klucz obcy: ReservationID, TakeawayID, ClientID
ID zamówienia: OrderID
ID klienta: ClientID
Data zamówienia: OrderDate
ID zamówienia na wynos: TakeawayID
ID rezerwacji: ReservationID
Określenie czy zamówienie zostało opłacone: Paid

```
CREATE TABLE Orders
(
   OrderID       int      NOT NULL,
   ClientID      int      NOT NULL,
   OrderDate     datetime NOT NULL,
   TakeawayID    int      NULL,
   ReservationID int      NULL,
   Paid          bit      NOT NULL,
   CONSTRAINT Orders_pk PRIMARY KEY (OrderID)
)

ALTER TABLE Orders
   ADD CONSTRAINT Reservation_Orders
       FOREIGN KEY (ReservationID)
           REFERENCES Reservation (ReservationID)

ALTER TABLE Orders
   ADD CONSTRAINT Orders_Clients
       FOREIGN KEY (ClientID)
           REFERENCES Clients (ClientID)

ALTER TABLE Orders
   ADD CONSTRAINT Orders_OrdersTakeaways
       FOREIGN KEY (TakeawayID)
           REFERENCES OrdersTakeaways (TakeawayID)
```

Tabela OrdersTakeaways

Reprezentacja zamówień na wynos.

Klucz główny: TakeawayID
ID zamówienia na wynos: TakeawayID
Preferowana data odbioru: PrefDate
Warunki integralności:

- Preferowana data odbioru późniejsza od daty aktualnej
  `CHECK (PrefDate > GETDATE())`

```
CREATE TABLE OrdersTakeaways
(
   TakeawayID int      NOT NULL,
   PrefDate   datetime NOT NULL,
   CONSTRAINT ValidPrefDate CHECK (PrefDate > GETDATE()),
   CONSTRAINT OrdersTakeaways_pk PRIMARY KEY (TakeawayID)
)
```

Tabela Person

Reprezentacja osób w bazie danych.

Klucz główny: PersonID
ID osoby: ProductID
Imię: FirstName
Nazwisko: LastName

```
CREATE TABLE Person
(
   FirstName varchar(255) NOT NULL,
   LastName  varchar(255) NOT NULL,
   PersonID  int          NOT NULL,
   CONSTRAINT Person_pk PRIMARY KEY (PersonID)
)
```

Tabela Products

Reprezentacja pozycji w menu w bazie danych.

Klucz główny: ProductID
Klucz obcy: CategoryID
ID produktu: ProductID
Nazwa produktu: Name
Opis produktu: Description
Wartości domyślne:

- Description: ‘brak opisu’
  `DEFAULT 'brak opisu'`

```
CREATE TABLE Products
(
   CategoryID  int          NOT NULL,
   ProductID   int          NOT NULL,
   Name        varchar(255) NOT NULL,
   Description varchar(255) NOT NULL DEFAULT 'brak opisu',
   CONSTRAINT MenuItems_pk PRIMARY KEY (ProductID)
)


ALTER TABLE Products
   ADD CONSTRAINT Category_MenuItems
       FOREIGN KEY (CategoryID)
           REFERENCES Category (CategoryID)
```

Tabela Reservation

Reprezentacja wszystkich rezerwacji w bazie danych.

Klucz główny: ReservationID
ID rezerwacji: ReservationID
Data rozpoczęcia rezerwacji: StartDate
Data zakończenia rezerwacji: EndDate
Status zamówienia: Status
Warunki integralności:

- Data zakończenia większa od daty rozpoczęcia
  `CHECK (EndDate > StartDate)`

```
CREATE TABLE Reservation
(
   ReservationID int         NOT NULL,
   StartDate     datetime    NOT NULL,
   EndDate       datetime    NOT NULL,
   Status        varchar(16) NOT NULL,
   CONSTRAINT ValidReservationDate CHECK (EndDate > StartDate),
   CONSTRAINT Reservation_pk PRIMARY KEY (ReservationID)
)
```

Tabela ReservationCompany

Reprezentacja rezerwacji dla firm.

Klucz główny: ReservationID
Klucz obcy: ClientID,ReservationID
ID rezerwacji: ReservationID
ID klienta: ClientID
ID osoby na którą jest rezerwacja: PersonID

```
CREATE TABLE ReservationCompany
(
   ReservationID int NOT NULL,
   ClientID      int NOT NULL,
   PersonID      int NULL,
   CONSTRAINT ReservationCompany_pk PRIMARY KEY (ReservationID)
)

ALTER TABLE ReservationCompany
   ADD CONSTRAINT Companies_ReservationCompany
       FOREIGN KEY (ClientID)
           REFERENCES Companies (ClientID)

ALTER TABLE ReservationCompany
   ADD CONSTRAINT ReservationCompany_Reservation
       FOREIGN KEY (ReservationID)
           REFERENCES Reservation (ReservationID)
```

Tabela ReservationDetails

Reprezentacja szczegółów rezerwacji.

Klucz obcy: TableID, ReservationID
ID rezerwacji: ReservationID
ID stolika: TableID

```
CREATE TABLE ReservationDetails
(
   ReservationID int NOT NULL,
   TableID       int NOT NULL
)

ALTER TABLE ReservationDetails
   ADD CONSTRAINT Tables_ReservationDetails
       FOREIGN KEY (TableID)
           REFERENCES Tables (TableID)

ALTER TABLE ReservationDetails
   ADD CONSTRAINT ReservationDetails_ReservationIndividual
       FOREIGN KEY (ReservationID)
           REFERENCES ReservationIndividual (ReservationID)

ALTER TABLE ReservationDetails
   ADD CONSTRAINT ReservationDetails_ReservationCompany
       FOREIGN KEY (ReservationID)
           REFERENCES ReservationCompany (ReservationID)
```

Tabela ReservationIndividual

Reprezentacja rezerwacji dla klientów indywidualnych.

Klucz główny: ReservationID
Klucz obcy: ReservationID
ID rezerwacji: ReservationID
ID klienta: ClientID
ID osoby na którą jest rezerwacja: PersonID

```
CREATE TABLE ReservationIndividual
(
   ReservationID int NOT NULL,
   ClientID      int NOT NULL,
   PersonID      int NOT NULL,
   CONSTRAINT Reservations_pk PRIMARY KEY (ReservationID)
)

ALTER TABLE ReservationIndividual
   ADD CONSTRAINT ReservationIndividual_Reservation
       FOREIGN KEY (ReservationID)
           REFERENCES Reservation (ReservationID)
```

Tabela ReservationVar

Reprezentacja zmiennych WZ i WK.

Klucz główny: ReservationVarID
ID zestawu zmiennych: ReservationVarID
Zmienna WZ: WZ
Zmienna WK: WK
Data rozpoczęcia używania zmiennych: StartDate
Data zakończenia używania zmiennych: EndDate
Warunki integralności:

- WZ oraz WK większe od 0
  `CHECK (WZ > 0 AND WK > 0)`

- Data zakończenia większa od daty rozpoczęcia
  `CHECK (ISNULL(EndDate,'9999-12-31 23:59:59') > StartDate)`

```
CREATE TABLE ReservationVar
(
   WZ               int      NOT NULL,
   WK               int      NOT NULL,
   ReservationVarID int      NOT NULL,
   StartDate        datetime NOT NULL,
   EndDate          datetime NULL,
   CONSTRAINT ValidReservationVar CHECK (WZ > 0 AND WK > 0 AND
ISNULL(EndDate, '9999-12-31 23:59:59') > StartDate ),
   CONSTRAINT ReservationVar_pk PRIMARY KEY (ReservationVarID)
)
```

Tabela Tables

Reprezentacja stolików w bazie danych.

Klucz główny: TableID
ID stolika: TableID
Rozmiar stolika (na ile osób): Size

```
CREATE TABLE Tables (
  TableID int  NOT NULL,
  Size int  NOT NULL,
  CONSTRAINT ValidSize CHECK (Size>0),
  CONSTRAINT Tables_pk PRIMARY KEY (TableID)
)
```

Widoki

Widok CurrentMenu

Wyświetla aktualne menu.

```
CREATE VIEW CurrentMenu AS
SELECT P.Name, M.Price
FROM Products P
        INNER JOIN Menu M
                   ON M.ProductID = P.ProductID
WHERE (GETDATE() BETWEEN M.StartDate AND M.EndDate)
go
```

Widok MealsMenuInfo

Wyświetla statystyki dań w danym menu (do jakiego menu należy danie, czas trwania danego menu, id produktu oraz ile razy zostało zamówione).

```
CREATE VIEW MealsMenuInfo AS
SELECT DISTINCT M.MenuID,
                M.StartDate,
                M.EndDate,
                M.ProductID,
                ISNULL((SELECT SUM(Quantity)
                        FROM Products P
                                 INNER JOIN OrderDetails OD on P.ProductID =
OD.ProductID AND P.ProductID = M.ProductID
                                 INNER JOIN Orders O on OD.OrderID = O.OrderID
                        WHERE (OrderDate BETWEEN M.StartDate AND M.EndDate)
                        GROUP BY P.Name), 0) times_sold
FROM Menu M
go
```

Widok OrdersToPay

Wyświetla zamówienia które nie zostały jeszcze opłacone przez klientów.

```
CREATE VIEW OrdersToPay AS
SELECT O.OrderID, O.ClientID, O.OrderDate
FROM ORDERS O
WHERE O.PAID = 0
go
```

Widok ClientsStats

Wyświetla informacje o kliencie, ilości jego zamówień, łącznej wartości ze wszystkich zamówień które złożył.

```
CREATE VIEW ClientStats AS
SELECT C.ClientID,
      City + ' ' + Street + ' ' + PostalCode   address,
      Phone,
      Email,
      Count(O.OrderID)                         times_ordered,
      ISNULL((SELECT value_ordered
              FROM (SELECT ClientID, SUM(val) value_ordered
                    FROM (SELECT O.ClientID
ClientID,
                                 OD.Quantity * (
                                     Select Price
                                     From Menu M2
                                     WHERE M2.MenuID = O.MenuID
                                       AND M2.ProductID = OD.ProductID) val
                          FROM OrderDetails OD
                                   INNER JOIN Orders O on O.OrderID =
OD.OrderID) out
                    GROUP BY ClientID) a
              WHERE ClientID = C.ClientID), 0) value_ordered
FROM Clients C
        LEFT JOIN Orders O
                  ON C.ClientID = O.ClientID
GROUP BY C.ClientID, City + ' ' + Street + ' ' + PostalCode, Phone, Email
go
```

Widok MealsInfo

Wyświetla informacje o wszystkich posiłkach, ich kategoriach oraz opisie.

```
CREATE VIEW MealsInfo
AS
SELECT C.CategoryName, P.Name, P.Description
FROM Products AS P
        INNER JOIN
    Category C on P.CategoryID = C.CategoryID
go
```

Widok MealsSoldInfo

Wyświetla dania oraz ile razy poszczególne dania zostały zamówione.

```
CREATE VIEW MealsSoldInfo
AS
SELECT P.Name, COUNT(OD.ProductID) times_sold
FROM Products AS P
        INNER JOIN
    OrderDetails AS OD ON P.ProductID = OD.ProductID
GROUP BY P.Name
go
```

Widok MealsSoldMonthly

Wyświetla dania oraz ile razy poszczególne dania zostały zamówione z podziałem na lata oraz miesiące.

```
CREATE VIEW MealsSoldInfoMonthly
AS
SELECT YEAR(O.OrderDate) year,
      MONTH(O.OrderDate) month,
      P.Name,
      ISNULL(COUNT(OD.ProductID),0) times_sold
FROM Products AS P
        LEFT JOIN
    OrderDetails AS OD ON P.ProductID = OD.ProductID
       LEFT JOIN Orders O on O.OrderID = OD.OrderID
GROUP BY YEAR(O.OrderDate), MONTH(O.OrderDate),P.Name
go
```

Widok MealsSoldWeekly

Wyświetla dania oraz ile razy poszczególne dania zostały zamówione z podziałem na lata oraz tygodnie.

```
CREATE VIEW MealsSoldInfoWeekly
AS
SELECT YEAR(O.OrderDate)              year,
      DATEPART(week, O.OrderDate)    week,
      P.Name,
      ISNULL(COUNT(OD.ProductID), 0) times_sold
FROM Products AS P
        LEFT JOIN
    OrderDetails AS OD ON P.ProductID = OD.ProductID
        LEFT JOIN Orders O on O.OrderID = OD.OrderID
GROUP BY YEAR(O.OrderDate), DATEPART(week, O.OrderDate), P.Name
go
```

Widok OrdersInfo

Wyświetla informacje o zamówieniu, klienta który je złożył, datę jego złożenia, czy zostało opłacone oraz wartość zamówienia.

```
CREATE VIEW OrdersInfo AS
SELECT OrderID,
      (SELECT SUM(Quantity * M.Price) value
       FROM OrderDetails OD2
                INNER JOIN Menu M on OD2.ProductID = M.ProductID
       WHERE OD2.OrderID = Orders.OrderID
       GROUP BY OD2.OrderID) value,
      ClientID,
      OrderDate,
      Paid
FROM Orders
go
```

Widok OwingClients

Wyświetla informacje o klientach którzy nie opłacili jeszcze zamówień oraz ile są nam winni.

```
CREATE VIEW OwingClients AS
SELECT OTP.ClientID client_id, sum(OI.order_value) money_to_pay
FROM OrdersToPay OTP
        INNER JOIN OrdersInfo OI on OTP.ClientID = OI.ClientID
GROUP BY OTP.ClientID
go
```

Widok PendingTakeaways

Wyświetla zamówienia na wynos które nie zostały jeszcze dostarczone.

```
CREATE VIEW PendingTakeaways AS
SELECT O.OrderID, O.ClientID
FROM ORDERS O
        INNER JOIN OrdersTakeaways OT ON
   OT.TakeawayID = O.TakeawayID
WHERE O.Paid = 1
 AND OT.PrefDate > GETDATE()
go
```

Widok DiscountInfoWeekly

Wyświetla ilość zniżek permanentnych oraz czasowych przyznanych w danym roku oraz tygodniu..

```
CREATE VIEW DiscountInfoWeekly
AS
SELECT YEAR(AppliedDate)           year,
      DATEPART(week, AppliedDate) week,
      DiscountType,
      COUNT(D.DiscountID)         amount_applied
FROM Discounts D
GROUP BY YEAR(AppliedDate), DATEPART(week, AppliedDate), DiscountType
go
```

Widok DiscountInfo

Wyświetla informacje na temat wszystkich przyznanych zniżek.

```
CREATE VIEW DiscountInfo AS
   SELECT DiscountID,
          C.ClientID,
          P.FirstName,
          P.LastName,
          DiscountType,
          AppliedDate,
          dbo.udfGetDiscountValue(DiscountID) value
   FROM Discounts D
   INNER JOIN Clients C on C.ClientID = D.ClientID
   INNER JOIN IndividualClient IC on C.ClientID = IC.ClientID
   INNER JOIN Person P on IC.PersonID = P.PersonID
go
```

Widok DiscountInfoMonthly

Wyświetla ilość zniżek permanentnych oraz czasowych przyznanych w danym roku oraz miesiącu.

```
CREATE VIEW DiscountInfoMonthly
AS
SELECT YEAR(AppliedDate)   year,
      MONTH(AppliedDate)  month,
      DiscountType,
      COUNT(D.DiscountID) amount_applied
FROM Discounts D
GROUP BY YEAR(AppliedDate), MONTH(AppliedDate), DiscountType
go
```

Widok TablesMonthly

Wyświetla statystyki miesięczne dotyczące rezerwacji stolików (w każdym miesiącu ile razy dany stolik został zarezerwowany oraz informacje o tym stoliku).

```
CREATE VIEW TablesMonthly AS
SELECT YEAR(R2.StartDate)  as year,
      MONTH(R2.StartDate) as month,
      T.TableID           as table_id,
      T.Size              as table_size,
      COUNT(RD.TableID)   as how_many_times_reserved
FROM Tables T
        INNER JOIN ReservationDetails RD on T.TableID = RD.TableID
        INNER JOIN Reservation R2 on RD.ReservationID = R2.ReservationID
GROUP BY YEAR(R2.StartDate), MONTH(R2.StartDate), T.TableID, T.Size
go
```

Widok TablesWeekly

Wyświetla statystyki tygodniowe dotyczące rezerwacji stolików(w każdym tygodniu ile razy dany stolik został zarezerwowany oraz informacje o tym stoliku).

```
CREATE VIEW TablesWeekly AS
SELECT YEAR(R2.StartDate)           as year,
      DATEPART(week, R2.StartDate) as week,
      T.TableID                    as table_id,
      T.Size                       as table_size,
      COUNT(RD.TableID)            as how_many_times_reserved
FROM Tables T
        INNER JOIN ReservationDetails RD on T.TableID = RD.TableID
        INNER JOIN Reservation R2 on RD.ReservationID = R2.ReservationID
GROUP BY YEAR(R2.StartDate), DATEPART(week, R2.StartDate), T.TableID, T.Size
go
```

Widok OrderStatsMonthly

Wyświetla ilość zamówień, ich wartość z podziałem na rok oraz miesiąc.

```
CREATE VIEW OrderStatsMonthly AS
SELECT YEAR(O.OrderDate)  as year,
      MONTH(O.OrderDate) as month,
      COUNT(*)           as order_count,
      SUM(order_value)   as value
FROM (SELECT OrderID,
             OrderDate,
             ISNULL((SELECT order_value
                     FROM (SELECT OrderID,
                                  SUM(val) order_value
                           FROM (SELECT O.OrderID OrderID,
                                        OD.Quantity *
                                        (Select Price
                                         From Menu M2
                                         WHERE M2.MenuID = O.MenuID
                                           AND M2.ProductID = OD.ProductID) val
                                 FROM OrderDetails OD
                                          INNER JOIN Orders O
                                                     on O.OrderID = OD.OrderID)
out
                           GROUP BY OrderID) out
                     WHERE OrderID = OI.OrderID), 0) order_value
      FROM Orders OI) O
GROUP BY YEAR(O.OrderDate), MONTH(O.OrderDate)
go
```

Widok OrderStatsWeekly

Wyświetla ilość zamówień, ich wartość z podziałem na rok oraz tydzień.

```
CREATE VIEW OrderStatsWeekly AS
SELECT YEAR(O.OrderDate)           as year,
      DATEPART(week, O.OrderDate) as week,
      COUNT(*)                    as order_count,
      SUM(order_value) as value
FROM (SELECT OrderID,
      OrderDate,
      (SELECT order_value
       FROM (SELECT OrderID,
                    SUM(val) order_value
             FROM (SELECT O.OrderID
OrderID,
                          OD.Quantity *
                          (Select Price From Menu M2 WHERE M2.MenuID = O.MenuID
AND M2.ProductID = OD.ProductID) val
                   FROM OrderDetails OD
                            INNER JOIN Orders O
                                       on O.OrderID = OD.OrderID) out
             GROUP BY OrderID
            ) out
       WHERE OrderID = OI.OrderID) order_value
FROM Orders OI) O
GROUP BY YEAR(O.OrderDate), DATEPART(week, O.OrderDate)
go
```

Widok PendingReservation

Wyświetla zamówienia oczekujące na potwierdzenie.

```
CREATE VIEW PendingReservations AS
   SELECT *
   FROM Reservation
   WHERE Status LIKE 'Pending'
go
```

Widok ReservationInfo

Wyświetla informacje o rezerwacjach (ile stolików oraz czas trwania rezerwacji)

```
CREATE VIEW ReservationInfo AS
SELECT R.ReservationID, TableID, StartDate, EndDate
FROM Reservation R
        LEFT OUTER JOIN ReservationDetails RD
                        on RD.ReservationID = R.ReservationID
WHERE Status NOT LIKE 'Cancelled'
go
```

Procedury

Procedura AddCategory

Przyjmuje nazwę kategorii jako argument, w rezultacie dodaje nową kategorię do tabeli Category.

```
CREATE PROCEDURE uspAddCategory
@CategoryName varchar(255)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          IF EXISTS(
              SELECT *
              FROM Category
              WHERE @CategoryName = CategoryName
           )
              BEGIN
               ;
               THROW 52000, N'Kategoria jest już dodana', 1
           end
          DECLARE @CategoryID INT
          SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
          FROM Category
          INSERT INTO Category(CategoryID, CategoryName)
          VALUES(@CategoryID, @CategoryName);
      END TRY
      BEGIN CATCH
          DECLARE @msg nvarchar(2048) =
                          N'Błąd dodawania kategorii: ' + ERROR_MESSAGE();
          THROW 52000, @msg, 1;
      END CATCH
END
go
```

Procedura AddProduct

Procedura przyjmuje nazwę produktu oraz nazwę kategorii jako argumenty, w rezultacie dodaje produkt do tabeli Products.

```
CREATE PROCEDURE uspAddProduct @Name varchar(255),
                               @CategoryName varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
                SELECT *
                FROM Products
                WHERE Name = @Name
           )
           BEGIN
               ;
               THROW 52000, N'Potrawa jest już dodana', 1
           END
       IF NOT EXISTS(
                SELECT *
                FROM Category
                WHERE CategoryName = @CategoryName
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiej kategorii', 1
           END
       DECLARE @CategoryID INT
       SELECT @CategoryID = CategoryID
       FROM Category
       WHERE CategoryName = @CategoryName
       DECLARE @ProductID INT
       SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
       FROM Products
       INSERT INTO Products(ProductID, Name, CategoryID)
       VALUES (@ProductID, @Name, @CategoryID);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodania potrawy: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
go
```

Procedura AddEmployee

Procedura przyjmuje jako argumenty nazwę firmy oraz imię i nazwisko pracownika, w rezultacie dodaje go do tabeli Employees.

```
CREATE PROCEDURE uspAddEmployee @CompanyName varchar(255),
                          @FirstName varchar(255),
                          @LastName varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
               SELECT *
               FROM Employees
               INNER JOIN Person P on Employees.PersonID = P.PersonID
               INNER JOIN Companies C on Employees.CompanyID = C.ClientID
               WHERE @FirstName = P.FirstName
               AND @LastName = P.LastName
               AND @CompanyName = C.CompanyName
           )
           BEGIN
               ;
               THROW 52000, N'Pracownik jest już w bazie', 1
           END
       IF NOT EXISTS(
           SELECT *
           FROM Companies
           WHERE @CompanyName = CompanyName
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiej firmy', 1
           end
       DECLARE @PersonID INT
       DECLARE @CompanyID INT
       SELECT @PersonID = ISNULL(MAX(PersonID), 0) + 1
       FROM Person
       SELECT @CompanyID = ClientID
       FROM Companies
       WHERE @CompanyName = CompanyName
       INSERT INTO Person(FirstName, LastName, PersonID)
       VALUES (@FirstName, @LastName, @PersonID)
       INSERT INTO Employees(PersonID, CompanyID)
       VALUES (@PersonID, @CompanyID)
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodania pracownika: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura AddProductToMenu

Procedura przyjmuje nazwę produktu, jego cenę oraz ID Menu do którego ma zostać dodany i w rezultacie dodaje ten produkt do odpowiedniego menu w tabeli Menu.

```
CREATE PROCEDURE uspAddProductToMenu @Name varchar(255),
                                     @Price money,
                                     @MenuID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Products
                WHERE Name = @Name
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiej potrawy', 1
           END
       IF NOT EXISTS(
                SELECT *
                FROM Menu
                WHERE MenuID = @MenuID
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiego menu', 1
           END
       DECLARE @ProductID INT
       SELECT @ProductID = ProductID
       FROM Products
       WHERE Name = @Name
       DECLARE @StartDate date
       SELECT TOP 1 @StartDate = StartDate
       FROM Menu
       WHERE MenuID = @MenuID
       DECLARE @EndDate date
       SELECT TOP 1 @EndDate = EndDate
       FROM Menu
       WHERE MenuID = @MenuID
       INSERT INTO Menu(MenuID, StartDate, EndDate, Price, ProductID)
       VALUES (@MenuID, @StartDate, @EndDate, @Price, @ProductID);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodania potrawy do menu: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
go
```

Procedura AddClient

Procedura przyjmuje adres oraz rodzaj klienta jako argumenty, w zależności od wybranego typu klienta powinna przyjąć również nazwę firmy i NIP lub nazwę klienta indywidualnego i w rezultacie dodaje klienta do odpowiednich tabel.

```
CREATE PROCEDURE uspAddClient @City varchar(255),
                        @Street varchar(255),
                        @PostalCode varchar(255),
                        @Phone varchar(32),
                        @Email varchar(255),
                        @ClientType varchar(1),
                        @CompanyName varchar(255),
                        @NIP varchar(32),
                        @FirstName varchar(255),
                        @LastName varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
               SELECT *
               FROM Clients
               WHERE Phone = @Phone
           )
           BEGIN
               ;
               THROW 52000, N'Numer telefonu jest już w bazie', 1
           END
       IF EXISTS(
               SELECT *
               FROM Clients
               WHERE Email = @Email
           )
           BEGIN
               ;
               THROW 52000, N'Email jest już w bazie', 1
           END
       IF EXISTS(
               SELECT *
               FROM Companies
               WHERE @CompanyName = CompanyName
           )
           BEGIN
               ;
               THROW 52000, N'Firma jest już w bazie', 1
           END
       IF EXISTS(
               SELECT *
               FROM Companies
```

```
            WHERE @NIP = NIP
        )
        BEGIN
            ;
            THROW 52000, N'NIP jest już w bazie', 1
        END
    DECLARE @ClientID INT
    SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
    FROM Clients
    DECLARE @PersonID INT
    SELECT @PersonID = ISNULL(MAX(PersonID), 0) + 1
    FROM Person
    INSERT INTO Clients(ClientID, City, Street, PostalCode, Phone, Email)
    VALUES (@ClientID, @City, @Street, @PostalCode, @Phone, @Email);
    IF @ClientType = 'C'
        INSERT INTO Companies(ClientID, CompanyName, NIP)
        VALUES (@ClientID, @CompanyName, @NIP)
    IF @ClientType = 'I'
        INSERT INTO Person(FirstName, LastName, PersonID)
        VALUES (@FirstName, @LastName, @PersonID)
    INSERT INTO IndividualClient(PersonID, ClientID)
    VALUES (@PersonID, @ClientID)
  END TRY
  BEGIN CATCH
    DECLARE @msg nvarchar(2048)
        =N'Błąd dodania klienta: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1
  END CATCH
END
go
```

Procedura AddOrder

Procedura przyjmuje ID klienta, informację czy zamówienie jest z rezerwacją lub na wynos oraz czy jest opłacone i w zależności od tego jakie parametry wybraliśmy przyjmuje dodatkowo datę odbioru zamówienia na wynos lub datę rozpoczęcia i zakończenia rezerwacji, w rezultacie dodaje zamówienie oraz ewentualną rezerwację lub zamówienie na wynos do odpowiednich tabel.

```
CREATE PROCEDURE uspAddOrder @ClientID int,
                             @Takeaway bit,
                             @Reservation bit,
                             @Paid bit,
                             @PrefDate datetime,
                             @StartDate datetime,
                             @EndDate datetime
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF ISNULL(@PrefDate,'9999-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data odbioru zamówienia na wynos', 1
           END
       IF @EndDate < GETDATE() OR @StartDate<GETDATE()
           BEGIN
               ;
               THROW 52000, 'Niepoprawna data rezerwacji', 1
           end
       DECLARE @OrderID INT
       Declare @ReservationIDIns INT = null
       Declare @TakeAwayIDIns INT = null
       Declare @ReservationID INT = null
       SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1
       FROM Reservation
       Declare @TakeawayID INT = null
       SELECT @TakeawayID = ISNULL(MAX(TakeawayID), 0) + 1
       FROM OrdersTakeaways
       DECLARE @CurrentMenuID int
       SELECT TOP 1 @CurrentMenuId = MenuID
       FROM Menu M
       WHERE GETDATE() BETWEEN M.StartDate AND M.EndDate
       SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
       FROM Orders
       IF (@Takeaway = 1)
           BEGIN
               SET @TakeAwayIDIns = @TakeawayID
               INSERT INTO OrdersTakeaways(TakeawayID, PrefDate)
               VALUES (@TakeawayID, @PrefDate)
           END
       IF (@Reservation = 1)
```

```
            BEGIN
                SET @ReservationIDIns = @ReservationID
                INSERT INTO Reservation(ReservationID, StartDate, EndDate, Status)
                VALUES (@ReservationID, @StartDate, @EndDate, 'Pending')
            END
        INSERT INTO Orders(OrderID, ClientID, OrderDate, TakeawayID, ReservationID, Paid,
MenuID)
        VALUES (@OrderID, @ClientID, GETDATE(), @TakeawayIDIns, @ReservationIDIns, @Paid,
@CurrentMenuID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania zamówienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura AddProductToOrder

Procedura przyjmuje w argumencie ID zamówienia, ilość oraz nazwę produktu, następnie dodaje te informacje do zamówienia.

```
CREATE PROCEDURE uspAddProductToOrder @OrderID int,
                                      @Quantity int,
                                      @ProductName varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
               SELECT *
               FROM Products
               WHERE Name = @ProductName
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiej potrawy', 1
           END
       IF NOT EXISTS(
               SELECT *
               FROM Orders
               WHERE OrderID = @OrderID
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiego zamowienia', 1
           END
       IF NOT EXISTS(
           SELECT *
           FROM CurrentMenu
           WHERE Name = @ProductName
           )
           BEGIN
               ;
               THROW 52000, 'Nie mozna zamowic tego produktu, gdyz nie ma go obecnie
w menu', 1
           end
       IF EXISTS (
           SELECT *
           FROM Products P
           INNER JOIN Category C on P.CategoryID = C.CategoryID
           WHERE @ProductName = P.Name
           )
           BEGIN
               ;
```

```
                DECLARE @OrderDate DATE
                SELECT @OrderDate = OrderDate
                FROM Orders
                WHERE OrderID = @OrderID
                IF DATEPART(WEEKDAY ,@OrderDate) != 4 AND DATEPART(WEEKDAY ,
@OrderDate) != 5 AND DATEPART(WEEKDAY ,@OrderDate) != 6
                BEGIN
                    ;
                    THROW 52000, N'Nieprawidłowa data złożenia zamówienia na owoce
morza', 1
                end
            end
        DECLARE @ProductID INT
        SELECT @ProductID = ProductID
        FROM Products
        WHERE Name = @ProductName
        INSERT INTO OrderDetails(OrderID, Quantity, ProductID)
        VALUES (@OrderID,@Quantity,@ProductID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania produktu do zamowienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

Procedura AddTableToReservation

Procedura przyjmuje w argumencie ID rezerwacji oraz ID stolika, następnie do danej rezerwacji przypisuje ID stolika.

```
CREATE PROCEDURE uspAddTableToReservation
@ReservationID int,
@TableID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Tables
                WHERE TableID = @TableID
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiego stolika', 1
           END
       IF NOT EXISTS(
                SELECT *
                FROM Orders
                WHERE ReservationID = @ReservationID
           )
           BEGIN
               ;
               THROW 52000, 'Nie ma takiej rezerwacji', 1
           END
       INSERT INTO ReservationDetails(ReservationID, TableID)
       VALUES (@ReservationID,@TableID)
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodania stolika do rezerwacji: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1
   END CATCH
END
```

Procedura AddTable

Procedura dodaje nowy stolik do tabeli Tables z wielkością podaną w argumencie

```
CREATE PROCEDURE uspAddTable
@Size int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          DECLARE @TableID INT
          SELECT @TableID = ISNULL(MAX(TableID), 0) + 1
          FROM Tables
          INSERT INTO Tables(TableID, Size)
          VALUES(@TableID, @Size);
      END TRY
      BEGIN CATCH
          DECLARE @msg nvarchar(2048)
                          =N'Błąd dodawania stolika: ' + ERROR_MESSAGE();
          THROW 5200, @msg, 1
      END CATCH
END
```

Procedura ModifyTable

Procedura w argumencie przyjmuje id stolika oraz wielkość stolika, następnie wyszukuje stolik o danym ID i zmienia jego wielkość na ten podany w argumencie.

```
CREATE PROCEDURE uspModifyTable
@TableID int,
@Size int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
				SELECT * FROM Tables
				WHERE TableID=@TableID
				)
		BEGIN;
				THROW 52000, 'Nie ma takiego stolika.', 1
		END
		IF @Size < 2
		BEGIN;
				THROW 52000, N'Stolik musi mieć przynajmniej 2
miejsca.', 1
		END
		IF @Size IS NOT NULL
		BEGIN
				UPDATE Tables
					SET Size = @Size
					WHERE Tables.TableID=@TableID
		END
	END TRY
	BEGIN CATCH
		DECLARE @msg nvarchar(2048)
					=N'Błąd edytowania stolika: ' +
ERROR_MESSAGE();
		THROW 52000, @msg, 1
	END CATCH
END
```

Procedura AddReservation

Procedura przyjmuje w argumencie id clienta, czas startu oraz czas zakończenia i status, następnie dodaje nową rezerwacje.

```
CREATE PROCEDURE uspAddReservation
@ClientID int,
@StartDate datetime,
@EndDate datetime,
@Status bit
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          IF NOT EXISTS(
              SELECT *
              FROM Clients
              WHERE ClientID = @ClientID
           )
              BEGIN
               ;
               THROW 52000, 'Nie ma takiego klienta', 1
           end
          DECLARE @ReservationID INT
          DECLARE @PersonID INT
          SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1
     FROM Reservation
          IF EXISTS(
              SELECT *
              FROM Companies
              WHERE ClientID = @ClientID
           )
          BEGIN
               INSERT INTO ReservationCompany(ReservationID, ClientID, PersonID)
               VALUES (@ReservationID,@ClientID,null)
           end
       ELSE BEGIN
           SELECT @PersonID = PersonID
           FROM IndividualClient
           WHERE ClientID = @ClientID
           INSERT INTO ReservationIndividual(ReservationID, ClientID, PersonID)
           VALUES (@ReservationID,@ClientID,@PersonID)
       end
          INSERT INTO Reservation(ReservationID, StartDate, EndDate, Status)
          VALUES(@ReservationID, @StartDate, @EndDate, @Status);
      END TRY
      BEGIN CATCH
          DECLARE @errorMsg nvarchar(2048)
                          =N'Błąd dodania rezerwacji: ' + ERROR_MESSAGE();
          THROW 52000, @errorMsg, 1
      END CATCH
END
```

Procedura ChangeReservationStatus

Przyjmuje id rezerwacji oraz status jako argument, w rezultacie zmienia status danej rezerwacji na ten w argumencie.

```
CREATE PROCEDURE uspChangeReservationStatus
@ReservationID int,
@Status varchar(255)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          BEGIN
                  UPDATE Reservation
                          SET Status = @Status
                          WHERE Reservation.ReservationID=@ReservationID
          END
      END TRY
      BEGIN CATCH
          DECLARE @msg nvarchar(2048)
                          =N'Błąd edytowania rezerwacji: ' + ERROR_MESSAGE();
          THROW 52000, @msg, 1
      END CATCH
END
```

Procedura ChangeOrderPaymentStatus

Przyjmuje OrderID oraz prawda/fałsz jako argument, następnie zmienia status danego zamówienia na ten podany w argumencie.

```
CREATE PROCEDURE uspChangeOrderPaymentStatus
@OrderID int,
@Paid bit
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          BEGIN
                  UPDATE Orders
                          SET Paid = @Paid
                          WHERE Orders.OrderID = @OrderID
          END
      END TRY
      BEGIN CATCH
          DECLARE @msg nvarchar(2048)
                          =N'Błąd zmiany statusu platnosci: ' +
ERROR_MESSAGE();
          THROW 52000, @msg, 1
      END CATCH
END
```

Procedura RemoveCategory

Przyjmuje nazwę kategorii jako argument, następnie szuka kategorii, jeżeli istnieje to ją usuwa, inaczej wyrzuca błąd.

```
CREATE PROCEDURE uspRemoveCategory
@CategoryName varchar(255)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
          IF NOT EXISTS(
              SELECT *
              FROM Category
              WHERE CategoryName = @CategoryName
           )
              BEGIN
               ;
               THROW 52000,'Nie ma takiej kategorii!',1
           end
          DELETE FROM Category
              WHERE CategoryName = @CategoryName
      END TRY
      BEGIN CATCH
          DECLARE @msg nvarchar(2048) =
                          N'Błąd usuwania kategorii: ' + ERROR_MESSAGE();
          THROW 52000, @msg, 1;
      END CATCH
END
```

Funkcje

Funkcja GetOrdersAboveXValue

Zwraca tabelę z zamówieniami powyżej przyjętej jako argument wartości.

```
CREATE FUNCTION udfGetOrdersAboveXValue(@input int)
   RETURNS table AS
       RETURN
       SELECT OI.OrderID, OI.ClientID, OI.order_value
       FROM OrdersInfo OI
       WHERE OI.order_value > @input
go
```

Funkcja GetMenuItemsById

Zwraca tabelę z pozycjami danego menu (przekazujemy id).

```
CREATE FUNCTION udfGetMenuItemsById(@id int)
   RETURNS TABLE AS
       RETURN
       SELECT M.MenuID, M.StartDate, M.EndDate, P.Name, M.Price
       FROM Products P
                INNER JOIN Menu M
                           ON M.ProductID = P.ProductID
       WHERE (M.MenuID = @id)
go
```

Funkcja GetMenuItemsByDate

Zwraca tabelę z pozycjami danego menu (przekazujemy date).

```
CREATE FUNCTION udfGetMenuItemsByDate(@date date)
   RETURNS TABLE AS
       RETURN
       SELECT M.MenuID, M.StartDate, M.EndDate, P.Name, M.Price
       FROM Products P
                INNER JOIN Menu M
                           ON M.ProductID = P.ProductID
       WHERE @date BETWEEN M.StartDate AND M.EndDate
go
```

Funkcja GetMealsSoldAtLeastXTimes

Zwraca pozycje, które sprzedały się więcej niż przyjętą jako argument liczbę razy.

```
CREATE FUNCTION udfGetMealsSoldAtLeastXTimes(@input int)
   RETURNS table AS
       RETURN
       SELECT MSI.Name, MSI.times_sold
       FROM MealsSoldInfo MSI
       WHERE MSI.times_sold > @input
go
```

Funkcja GetValueOfOrdersOnDay

Zwraca wartość zamówień podczas przyjętego jako argument dnia.

```
CREATE FUNCTION udfgetValueOfOrdersOnDay(@date date)
   RETURNS int
AS
BEGIN
   RETURN (SELECT SUM(OI.order_value * (1 -
dbo.udfGetDiscountValue(O.AppliedDiscount)))
           FROM OrdersInfo OI
                    INNER JOIN Orders O on OI.OrderID = O.OrderID
           WHERE YEAR(@date) = YEAR(OI.OrderDate)
             AND MONTH(@date) = MONTH(OI.OrderDate)
             AND DAY(@date) = DAY(OI.OrderDate))
END
go
```

Funkcja GetValueOfOrdersInMonth

Zwraca wartość zamówień przyjętego jako argument miesiąca (przekazujemy miesiąc i rok).

```
CREATE FUNCTION udfgetValueOfOrdersInMonth(@year int, @month int)
   RETURNS int
AS
BEGIN
   RETURN (SELECT SUM(OI.order_value * (1 -
dbo.udfGetDiscountValue(O.AppliedDiscount)))
           FROM OrdersInfo OI
                    INNER JOIN Orders O on OI.OrderID = O.OrderID
           WHERE @year = YEAR(OI.OrderDate)
             AND @month = MONTH(OI.OrderDate))
END
go
```

Funkcja GetAvgPriceOfMenu

Zwraca średnią cenę produktów z menu o ID podawanym jako argument.

```
CREATE FUNCTION udfGetAvgPriceOfMenu(@MenuID int)
   RETURNS money
AS
BEGIN
   RETURN (SELECT AVG(M.Price) FROM Menu M WHERE MenuID = @MenuID)
END
go
```

Funkcja GetBestMeals

Zwraca daną liczbę najczęściej kupowanych produktów.

```
CREATE FUNCTION udfGetBestMeals(@input int)
   RETURNS table AS
       RETURN
       SELECT DISTINCT TOP (@input) P.Name, MMI.times_sold
       FROM Products P
                INNER JOIN MealsMenuInfo MMI on P.ProductID = MMI.ProductID
       ORDER BY MMI.times_sold
go
```

Funkcja GetClientsOrderedMoreThanXTimes

Zwraca klientów którzy zamówili co najmniej daną liczbę razy.

```
CREATE FUNCTION udfGetClientsOrderedMoreThanXTimes(@amount int)
   RETURNS TABLE AS
       RETURN
       SELECT *
       FROM ClientStats
       WHERE times_ordered > @amount
go
```

Funkcja GetClientsOrderedMoreThanXValue

Zwraca klientów którzy złożyli zamówienia o łącznej wartości przekraczającej wartość przyjętą jako argument.

```
CREATE FUNCTION udfGetClientsOrderedMoreThanXValue(@value float)
   RETURNS TABLE AS
       RETURN
       SELECT *
       FROM ClientStats
       WHERE ClientStats.value_ordered > @value
go
```

Funkcja GetClientsWhoOweMoreThanX

Zwraca klientów, którzy mają u nas długi na przyjętą jako argument liczbę lub więcej.

```
CREATE FUNCTION udfGetClientsWhoOweMoreThanX(@val int)
   RETURNS table AS
       RETURN
       SELECT client_id, money_to_pay FROM OwingClients
WHERE money_to_pay > @val
go
```

Funkcja GetDiscountValue

W argumencie przyjmuje ID zniżki, następnie zwraca jej wartość, jeżeli nie znajdzie to zwraca 0.

```
CREATE FUNCTION udfGetDiscountValue(@id int)
RETURNS FLOAT AS
   BEGIN
       DECLARE @val float;
       DECLARE @type varchar(32);
       DECLARE @setid int;
       SET @type = (SELECT DiscountType FROM Discounts WHERE DiscountID=@id)
       SET @setid = (SELECT SetID  FROM Discounts WHERE DiscountID=@id)
       IF @type IS NULL
           begin
               SET @val = 0
           end
       IF @type = 'temp'
           BEGIN
               SET @val = (SELECT R2 FROM DiscountVars DV WHERE SetID=@setid)
           end
       IF @type = 'perm'
           BEGIN
               SET @val = (SELECT R1 FROM DiscountVars DV WHERE SetID=@setid)
           end
       RETURN @val
   END
go
```

Funkcja GetEmployeesOfCompany

W argumencie przyjmuje nazwę firmy, następnie zwraca pracowników tej firmy, jeżeli firma nie istnieje zwróci null.

```
CREATE FUNCTION udfGetEmployeesOfCompany(@CompanyName varchar(255))
   RETURNS table AS
       RETURN
       SELECT P.Firstname, P.Lastname
       FROM Person P
                INNER JOIN Employees E on P.PersonID = E.PersonID
                INNER JOIN Companies C on E.CompanyID = C.ClientID
       WHERE @CompanyName = CompanyName
go
```

Funkcja GetMaxPriceOfMenu

Przyjmuje w argumencie id menu, następnie zwraca najdroższą cenę produktu z danego menu.

```
CREATE FUNCTION udfGetMaxPriceOfMenu(@MenuID int)
   RETURNS money
AS
BEGIN
   RETURN (SELECT TOP 1 MAX(M.Price) FROM Menu M WHERE MenuID = @MenuID)
END
go
```

Funkcja GetMealsSoldAtLeastXTimes

W argumencie przyjmuje liczbę, następnie zwraca posiłki sprzedane co najmniej daną liczbę razy.

```
CREATE FUNCTION udfGetMealsSoldAtLeastXTimes(@input int)
   RETURNS table AS
       RETURN
       SELECT MSI.Name, MSI.times_sold
       FROM MealsSoldInfo MSI
       WHERE MSI.times_sold > @input
go
```

Funkcja GetMinPriceOfMenu

Przyjmuje jako argument ID menu, następnie zwraca najtańszą cenę produktu z tego menu.

```
CREATE FUNCTION udfGetMinPriceOfMenu(@MenuID int)
   RETURNS money
AS
BEGIN
   RETURN (SELECT TOP 1 MIN(M.Price) FROM Menu M WHERE MenuID = @MenuID)
END
go
```

Funkcja GetBestTempDiscountByClientID

Przyjmuje ID klienta jako argument, następnie zwraca najkorzystniejszą zniżkę czasową (aktualną) danego klienta.

```
CREATE FUNCTION udfGetBestTempDiscountByClientID(@id int)
   RETURNS FLOAT AS
BEGIN
   DECLARE @val float;
   SET @val = (SELECT TOP 1 DV.R1
               FROM Discounts
                        INNER JOIN DiscountVars DV on Discounts.SetID = DV.SetID
               WHERE ClientID = @id
                 AND DiscountType = 'temp'
                 AND GETDATE() BETWEEN AppliedDate AND DATEADD(day, DV.D1,
AppliedDate)
               ORDER BY 1 DESC)
   IF @val IS NULL
       BEGIN
           RETURN 0
       end
   RETURN @val
END
go
```

Funkcja GetBestPermDiscountByClientID

Przyjmuje ID klienta, następnie zwraca najkorzystniejszą zniżkę wieczystą danego klienta.

```
CREATE FUNCTION udfGetBestPermDiscountByClientID(@id int)
   RETURNS FLOAT AS
BEGIN
   DECLARE @val float;
   SET @val = (SELECT TOP 1 DV.R1
               FROM Discounts
                        INNER JOIN DiscountVars DV on Discounts.SetID = DV.SetID
               WHERE ClientID = @id
                 AND DiscountType = 'perm'
                 AND GETDATE() BETWEEN AppliedDate AND DATEADD(day, DV.D1,
AppliedDate)
               ORDER BY 1 DESC)
   IF @val IS NULL
       BEGIN
           RETURN 0
       end
   RETURN @val
END
go
```

Funkcja MenuIsCorrect

Przyjmuje menuID jako argument następnie, zwraca boolean poprawności danego menu (min. połowa dań wymieniona).

```
CREATE FUNCTION udfMenuIsCorrect(@id int)
   RETURNS int
AS
BEGIN
   DECLARE @sameItems int
   SET @sameItems = (SELECT COUNT(*)
   FROM (
            SELECT ProductID
            FROM Menu
            WHERE MenuID = (@id - 1) --prev table
            INTERSECT
            SELECT ProductID
            FROM Menu
            WHERE MenuID = @id --table to check
        ) out)
   DECLARE @minAmountToChange int
   SET @minAmountToChange = (SELECT COUNT(*) FROM Menu WHERE MenuID=(@id -1))/2
   IF @sameItems <= @minAmountToChange
   BEGIN
       return 1
   end
   return 0
END
go
```

Funkcja GetOrderValue

Przyjmuje ID zamówienia jako argument, następnie zwraca wartość zamówienia z uwzględnieniem zniżki.

```
CREATE FUNCTION udfgetOrderValue(@id int)
   RETURNS money
AS
BEGIN
   RETURN (SELECT order_value*(1-dbo.udfGetOrderDiscountValue(@id))
FROM OrdersInfo
WHERE OrderID = @id)
END
go
```

Funkcja GetOrderDiscountValue

Przyjmuje ID zamówienia jako argument, następnie zwraca wartość zniżki przypisanej do danego zamówienia.

```
CREATE FUNCTION udfGetOrderDiscountValue(@orderid int)
   RETURNS FLOAT AS
BEGIN
   DECLARE @id float;
   SET @id = (SELECT AppliedDiscount
       FROM Orders
       WHERE OrderID=@orderid)
   IF @id IS NULL
       BEGIN
           RETURN 0
       end
   RETURN dbo.udfGetDiscountValue (@id)
END
go
```

Triggery

Trigger ProperTimeVar

Trigger sprawdza czy dodana nowa zmienna zniżki D1 (ilość dni, podczas których trwa zniżka czasowa) jest prawidłowa (większa od 0).

```
create trigger TR_ProperTimeVar on DiscountVars
for insert
as
BEGIN
      if (select COUNT(*) from inserted) > 1
      begin
          RAISERROR('Dodawaj rabaty pojedynczo! ', 16, 1)
          ROLLBACK TRANSACTION
      end
      else if (select D1 from inserted) <= 0
      BEGIN
          RAISERROR('Wprowadzono niepoprawny rabat D1', 16, 1)
          ROLLBACK TRANSACTION
      END
END
go
```

Trigger DeleteOrderDetails

Trigger usuwa szczegóły zamówienia z tabeli OrderDetails, jeżeli powiązana z nim rezerwacja została anulowana.

```
CREATE TRIGGER TR_DeleteOrderDetails
ON OrderDetails
FOR DELETE
AS
BEGIN
  SET NOCOUNT ON;
  DELETE FROM OrderDetails
  WHERE OrderID in (
      select O.OrderID from Orders O
      inner join Reservation R on O.ReservationID = R.ReservationID
      where R.Status = 'Canceled'
      )
end
go
```

Trigger ProperDiscountVar

Trigger sprawdza czy dodana nowa zmienna zniżki (R1, R2) są z zakresu 0-1 (0%-100%).

```
create trigger TR_ProperDiscountVar on DiscountVars
for insert
as
BEGIN
	if (select COUNT(*) from inserted) > 1
	begin
		RAISERROR('Dodawaj rabaty pojedynczo! ', 16, 1)
		ROLLBACK TRANSACTION
	end
	else if (select R1 from inserted) not between 0 and 1
	BEGIN
		RAISERROR('Wprowadzono niepoprawny rabat R1', 16, 1)
		ROLLBACK TRANSACTION
	END
	else if (select R2 from inserted) not between 0 and 1
	BEGIN
		RAISERROR('Wprowadzono niepoprawny rabat R2', 16, 1)
		ROLLBACK TRANSACTION
	END
END
go
```

Trigger ProperMinOrdersVar

Trigger sprawdza czy dodana nowa zmienna zniżki Z1 (ilość minimalna zamówień by otrzymać zniżkę) jest prawidłowa (większa od 0).

```
create trigger TR_ProperMinOrdersVar on DiscountVars
for insert
as
BEGIN
      if (select COUNT(*) from inserted) > 1
      begin
          RAISERROR('Dodawaj rabaty pojedynczo! ', 16, 1)
          ROLLBACK TRANSACTION
      end
      else if (select Z1 from inserted) <= 0
      BEGIN
          RAISERROR('Wprowadzono niepoprawny rabat Z1', 16, 1)
          ROLLBACK TRANSACTION
      END
END
go
```

Trigger SeaFoodCheckMonday

Trigger ten blokuje zamówienia, które ze względu na znajdujące się w nim owoce morza, winno być złożone maksymalnie do poniedziałku poprzedzającego zamówienie.

```
CREATE TRIGGER TR_SeaFoodCheckMonday
ON OrderDetails
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
	IF EXISTS(
	SELECT * FROM inserted AS I
	INNER JOIN Orders AS O ON O.OrderID=I.OrderID
	INNER JOIN Products AS P ON P.ProductID=I.ProductID
	INNER JOIN OrdersTakeaways AS OT ON O.TakeawayID=OT.TakeawayID
	INNER JOIN Reservation AS R ON O.ReservationID=R.ReservationID
	WHERE (DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Thursday'
	AND DATEDIFF(day, O.OrderDate, OT.PrefDate)<=2
	AND CategoryID = 6)
	OR (DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Friday'
	AND DATEDIFF(day, O.OrderDate, OT.PrefDate)<=3
	AND CategoryID = 6)
	OR (DATENAME(WEEKDAY, OT.PrefDate) LIKE 'Saturday'
	AND DATEDIFF(day, O.OrderDate, OT.PrefDate)<=4
	AND CategoryID = 6)
	OR (DATENAME(WEEKDAY, R.StartDate) LIKE 'Thursday'
	AND DATEDIFF(day, O.OrderDate, R.StartDate)<=2
	AND CategoryID = 6)
	OR (DATENAME(WEEKDAY, R.StartDate) LIKE 'Friday'
	AND DATEDIFF(day, O.OrderDate, R.StartDate)<=3
	AND CategoryID = 6)
	OR (DATENAME(WEEKDAY, R.StartDate) LIKE 'Saturday'
	AND DATEDIFF(day, O.OrderDate, R.StartDate)<=4
	AND CategoryID = 6)
	)
	BEGIN;
		THROW 50001, 'Takie zamówienie winno być złożone maksymalnie do
poniedziałku poprzedzającego zamówienie. ', 1
	END
END
```

Indeksy

Indeks Category_pk

Ustawienie indeksu na CategoryID w tabeli Category.

```
CREATE UNIQUE INDEX Category_pk
ON Category (CategoryID)`
```

Indeks Clients_pk

Ustawienie indeksu na ClientID w tabeli Clients.

```
CREATE UNIQUE INDEX Clients_pk
ON Clients (ClientID)
```

Indeks UniquePhone

Ustawienie indeksu na Phone w tabeli Clients.

```
CREATE UNIQUE INDEX UniquePhone
ON Clients (Phone)
```

Indeks UniqueEmail

Ustawienie indeksu na Email w tabeli Clients.

```
CREATE UNIQUE INDEX UniqueEmail
ON Clients (Email)
```

Indeks Companies_pk

Ustawienie indeksu na ClientID w tabeli Companies.

```
CREATE UNIQUE INDEX Companies_pk
ON Companies (ClientID)
```

Indeks UniqueNIP

Ustawienie indeksu na NIP w tabeli Companies.

```
CREATE UNIQUE INDEX UniqueNIP
ON Companies (NIP)
```

Indeks Discounts_pk

Ustawienie indeksu na DiscountID w tabeli Discounts.

```
CREATE UNIQUE INDEX Discounts_pk
ON Discounts (DiscountID)
```

Indeks DiscountVars_pk

Ustawienie indeksu na SetID w tabeli DiscountVars.

```
CREATE UNIQUE INDEX DiscountVars_pk
ON DiscountVars (SetID)
```

Indeks Employees_pk

Ustawienie indeksu na PersonID w tabeli Employees.

```
CREATE UNIQUE INDEX Employees_pk
ON Employees (PersonID)
```

Indeks IndividualClient_pk

Ustawienie indeksu na ClientID w tabeli IndividualClient.

```
CREATE UNIQUE INDEX IndividualClient_pk
ON IndividualClient (ClientID)
```

Indeks Menu_pkk

Ustawienie indeksów na MenuID i ProductID w tabeli Menu.

```
CREATE UNIQUE INDEX Menu_pkk
ON Menu (MenuID, ProductID)
```

Indeks OrderDetails_pk

Ustawienie indeksów na OrderID i ProductID w tabeli OrderDetails.

```
CREATE UNIQUE INDEX OrderDetails_pk
ON OrderDetails (OrderID, ProductID)
```

Indeks Orders_pk

Ustawienie indeksu na OrderID w tabeli Orders.

```
CREATE UNIQUE INDEX Orders_pk
ON Orders (OrderID)
```

Indeks OrdersTakeaways_pk

Ustawienie indeksu na TakeawayID w tabeli OrdersTakeaways.

```
CREATE UNIQUE INDEX OrdersTakeaways_pk
ON OrdersTakeaways (TakeawayID)
```

Indeks Person_pk

Ustawienie indeksu na PersonID w tabeli Person.

```
CREATE UNIQUE INDEX Person_pk
ON Person (PersonID)
```

Indeks MenuItems_pk

Ustawienie indeksu na ProductID w tabeli Products.

```
CREATE UNIQUE INDEX MenuItems_pk
ON Products (ProductID)
```

Indeks Reservation_pk

Ustawienie indeksu na ReservationID w tabeli Reservation.

```
CREATE UNIQUE INDEX Reservation_pk
ON Reservation (ReservationID)
```

Indeks ReservationCompany_pk

Ustawienie indeksu na ReservationID w tabeli ReservationCompany.

```
CREATE UNIQUE INDEX ReservationCompany_pk
ON ReservationCompany (ReservationID)
```

Indeks ReservationDetails_pk

Ustawienie indeksów na ReservationID i TableID w tabeli ReservationDetails.

```
CREATE UNIQUE INDEX ReservationDetails_pk
ON ReservationDetails (ReservationID, TableID)
```

Indeks Reservations_pk

Ustawienie indeksu na ReservationID w tabeli ReservationIndividual.

```
CREATE UNIQUE INDEX Reservations_pk
ON ReservationIndividual (ReservationID)
```

Indeks ReservationVar_pk

Ustawienie indeksu na ReservationVarID w tabeli ReservationVar.

```
CREATE UNIQUE INDEX ReservationVar_pk
ON ReservationVar (ReservationVarID)
```

Uprawnienia

Uprawnienia ról w bazie danych	
Rola	**Opis**
Pracownik	Pracownik restauracji zajmujący się przyjmowaniem, sprawdzaniem, aktualizowaniem zamówień online jak i stacjonarnych. Ma możliwość dodawania rezerwacji, zamówień, produktów do zamówień, zmiany menu, dodawać stoliki, generować raporty.
Moderator	Ma dostęp do wszystkiego powyżej. Pracownik restauracji zajmujący się moderowaniem użytkowników systemu, może zmieniać ich dane, aktualizować, usuwać klientów z bazy, nadawać ręcznie im zniżki.
Administrator	Ma dostęp do każdej funkcji systemu bazodanowego.
Kod realizujący	
Pracownik	

```
CREATE ROLE worker

GRANT SELECT ON ClientStats to worker
GRANT SELECT ON CurrentMenu to worker
GRANT SELECT ON DiscountInfo to worker
GRANT SELECT ON DiscountInfoWeekly to worker
GRANT SELECT ON DiscountInfoMonthly to worker
GRANT SELECT ON MealsInfo to worker
GRANT SELECT ON MealsMenuInfo to worker
GRANT SELECT ON MealsSoldInfo to worker
GRANT SELECT ON MealsSoldInfoMonthly to worker
GRANT SELECT ON MealsSoldInfoWeekly to worker
GRANT SELECT ON OrdersInfo to worker
GRANT SELECT ON OrderStatsMonthly to worker
GRANT SELECT ON OrderStatsWeekly to worker
GRANT SELECT ON OrdersToPay to worker
GRANT SELECT ON OwingClients to worker
GRANT SELECT ON PendingReservations to worker
GRANT SELECT ON PendingTakeaways to worker
GRANT SELECT ON ReservationInfo to worker
```

	GRANT SELECT ON TablesMonthly to worker GRANT SELECT ON TablesWeekly to worker GRANT EXECUTE ON uspAddCategory to worker GRANT EXECUTE ON uspAddProduct to worker GRANT EXECUTE ON uspAddProductToMenu to worker GRANT EXECUTE ON uspAddProductToOrder to worker GRANT EXECUTE ON uspChangeReservationStatus to worker GRANT EXECUTE ON uspChangeOrderPaymentStatus to worker GRANT EXECUTE ON uspAddTable to worker
Moderator	CREATE ROLE moderator GRANT SELECT ON ClientStats to moderator GRANT SELECT ON CurrentMenu to moderator GRANT SELECT ON DiscountInfo to moderator GRANT SELECT ON DiscountInfoWeekly to moderator GRANT SELECT ON DiscountInfoMonthly to moderator GRANT SELECT ON MealsInfo to moderator GRANT SELECT ON MealsMenuInfo to moderator GRANT SELECT ON MealsSoldInfo to moderator GRANT SELECT ON MealsSoldInfoMonthly to moderator GRANT SELECT ON MealsSoldInfoWeekly to moderator GRANT SELECT ON OrdersInfo to moderator GRANT SELECT ON OrderStatsMonthly to moderator GRANT SELECT ON OrderStatsWeekly to moderator GRANT SELECT ON OrdersToPay to moderator GRANT SELECT ON OwingClients to moderator GRANT SELECT ON PendingReservations to moderator GRANT SELECT ON PendingTakeaways to moderator GRANT SELECT ON ReservationInfo to moderator GRANT SELECT ON TablesMonthly to moderator GRANT SELECT ON TablesWeekly to moderator GRANT EXECUTE ON uspAddCategory to moderator GRANT EXECUTE ON uspAddProduct to moderator GRANT EXECUTE ON uspAddProductToMenu to moderator GRANT EXECUTE ON uspAddProductToOrder to moderator GRANT EXECUTE ON uspChangeReservationStatus to moderator GRANT EXECUTE ON uspChangeOrderPaymentStatus to moderator GRANT EXECUTE ON uspAddTable to moderator GRANT EXECUTE ON uspAddClient to moderator GRANT EXECUTE ON uspAddEmployee to moderator GRANT SELECT, INSERT, DELETE ON Companies to moderator

	GRANT SELECT, INSERT, DELETE, UPDATE ON Clients to moderator GRANT SELECT, INSERT, DELETE, UPDATE ON Employees to moderator GRANT SELECT, INSERT, DELETE, UPDATE ON IndividualClient to moderator GRANT SELECT, INSERT, DELETE, UPDATE ON Person to moderator
Administrator	CREATE ROLE admin grant all privileges ON u_steczkie.dbo TO admin