Projekt bazy danych

Podstawy Baz Danych 2022/2023

Adam Mytnik, Adrian Madej, Jakub Sus

Spis treści

1. Użytkownicy

- Klient indywidualny

  Ma możliwość składania zamówień, rezerwowania stolików, jest w stanie sprawdzić swoją historię zakupów, dostępne rabaty.

- Firma

  Ma możliwość składania zamówień, rezerwowania stolików. Może generować dane potrzebne do faktur

- Pracownik

  Ma możliwość generowania menu, przeglądania statystyk klientów, wystawiania faktur, przyjmowania zamówień, dodawania zniżek klientom

Funkcje użytkowników

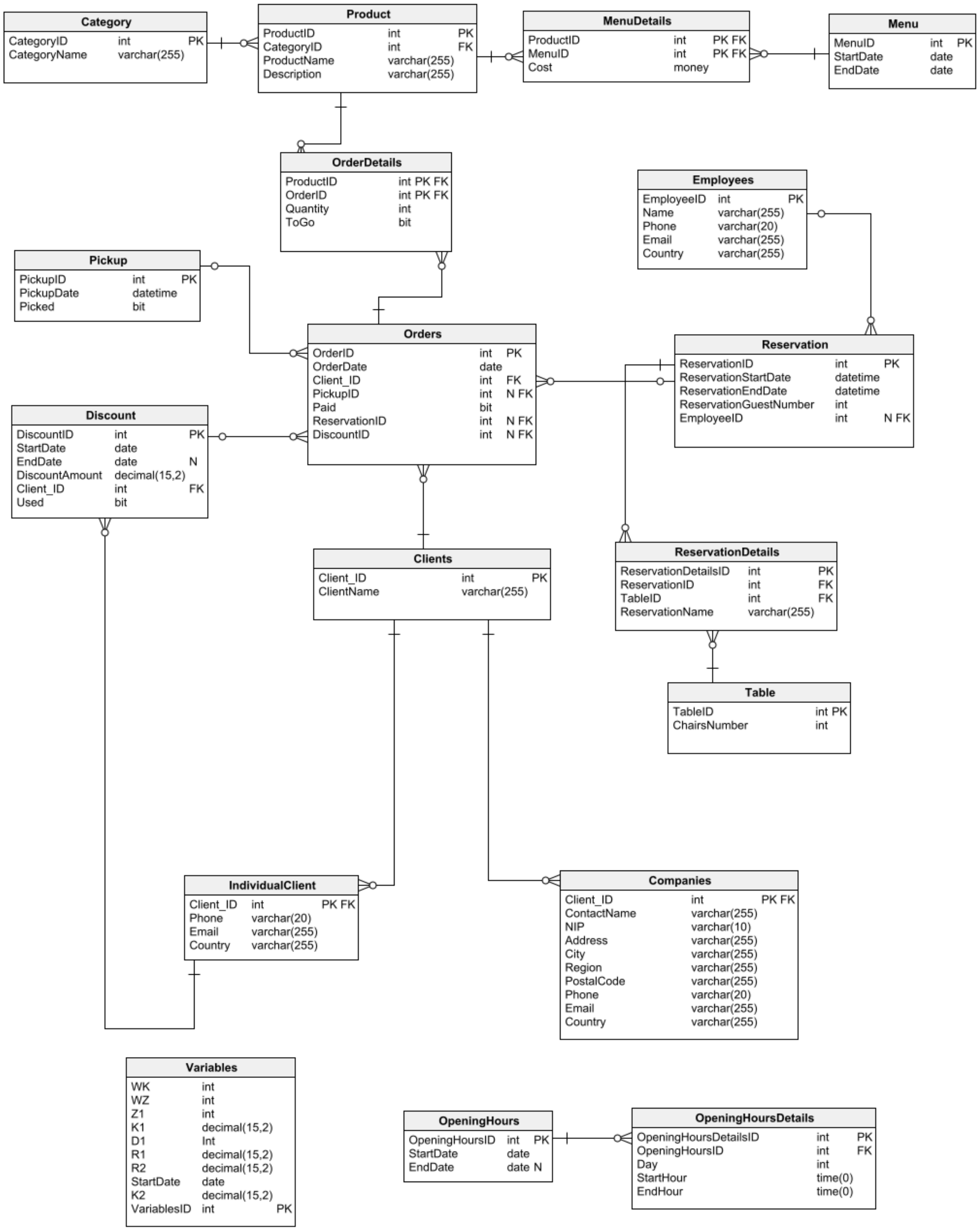
- Klient indywidualny, firma
  1. Składanie zamówienia na wynos
  2. Składanie zamówienia z rezerwacją stolika
  3. Wyświetlanie historii zamówień
  4. Wyświetlanie dostępnych rabatów

- Pracownik
  1. Dodawanie zamówień
  2. Wystawianie faktur
  3. Potwierdzenie rezerwacji
  4. Wyświetlanie zamówień klientów
  5. Dodanie produktu
  6. Dodanie kategorii
  7. Dodawanie produktu do menu
  8. Generowanie raportów tygodniowych
  9. Generowanie raportów miesięcznych
  10. Potwierdzenie rezerwacji
  11. Dodawanie klientów
  12. Przydzielanie zniżek

Funkcje Systemowe

1. Wyliczanie kosztu zamówienia
2. Sprawdzenie dostępności stolików
3. Sprawdzenie czy klient może zarezerwować stolik

2. Schemat bazy



Category

CategoryID	int	PK
CategoryName	varchar(255)	

Product

ProductID	int	PK
CategoryID	int	FK
ProductName	varchar(255)	
Description	varchar(255)	

MenuDetails

ProductID	int	PK FK
MenuID	int	PK FK
Cost	money	

Menu

MenuID	int	PK
StartDate	date	
EndDate	date	

OrderDetails

ProductID	int	PK FK
OrderID	int	PK FK
Quantity	int	
ToGo	bit	

Employees

EmployeeID	int	PK
Name	varchar(255)	
Phone	varchar(20)	
Email	varchar(255)	
Country	varchar(255)	

Pickup

PickupID	int	PK
PickupDate	datetime	
Picked	bit	

Orders

OrderID	int	PK
OrderDate	date	
Client_ID	int	FK
PickupID	int	N FK
Paid	bit	
ReservationID	int	N FK
DiscountID	int	N FK

Reservation

ReservationID	int	PK
ReservationStartDate	datetime	
ReservationEndDate	datetime	
ReservationGuestNumber	int	
EmployeeID	int	N FK

Discount

DiscountID	int	PK
StartDate	date	
EndDate	date	N
DiscountAmount	decimal(15,2)	
Client_ID	int	FK
Used	bit	

Clients

Client_ID	int	PK
ClientName	varchar(255)	

ReservationDetails

ReservationDetailsID	int	PK
ReservationID	int	FK
TableID	int	FK
ReservationName	varchar(255)	

Table

TableID	int	PK
ChairsNumber	int	

IndividualClient

Client_ID	int	PK FK
Phone	varchar(20)	
Email	varchar(255)	
Country	varchar(255)	

Companies

Client_ID	int	PK FK
ContactName	varchar(255)	
NIP	varchar(10)	
Address	varchar(255)	
City	varchar(255)	
Region	varchar(255)	
PostalCode	varchar(255)	
Phone	varchar(20)	
Email	varchar(255)	
Country	varchar(255)	

Variables

WK	int	
WZ	int	
Z1	int	
K1	decimal(15,2)	
D1	Int	
R1	decimal(15,2)	
R2	decimal(15,2)	
StartDate	date	
K2	decimal(15,2)	
VariablesID	int	PK

OpeningHours

OpeningHoursID	int	PK
StartDate	date	
EndDate	date	N

OpeningHoursDetails

OpeningHoursDetailsID	int	PK
OpeningHoursID	int	FK
Day	int	
StartHour	time(0)	
EndHour	time(0)	

3. Tabele

1.Category

(Kategoria potrawy)
Klucz główny: CategoryID
Nazwa kategorii: CategoryName

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

2.Clients

(Ogólna reprezentacja klienta)
Klucz główny: Client_ID
Nazwa klienta: ClientName

```
CREATE TABLE Clients (
    Client_ID int  NOT NULL IDENTITY,
    ClientName varchar(255)  NOT NULL,
    CONSTRAINT Clients_pk PRIMARY KEY  (Client_ID)
);
```

3.Companies

(Reprezentacja klienta-firmy)
Klucz główny i obcy: Client_ID
Nazwa osoby w firmie odpowiedzialnej za kontakt: ContactName
NIP: NIP
Adres: Address
Miasto: City
Region: Region
Państwo: Country
Kod pocztowy: PostalCode
Numer telefonu: Phone
Adres E-mail: Email

```
CREATE TABLE Companies (
    Client_ID int  NOT NULL,
    ContactName varchar(255)  NOT NULL,
    NIP varchar(10)  NOT NULL,
    Address varchar(255)  NOT NULL,
    City varchar(255)  NOT NULL,
    Region varchar(255)  NOT NULL,
    PostalCode varchar(255)  NOT NULL,
    Phone varchar(20)  NOT NULL,
    Email varchar(255)  NOT NULL,
    Country varchar(255)  NOT NULL,
    CONSTRAINT NIPUnique UNIQUE (NIP),
    CONSTRAINT PhoneValid 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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland'  ),
    CONSTRAINT NIPValid CHECK ((NIP LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' AND Country = 'Poland') OR
Country != 'Poland' ),
    CONSTRAINT EmailValid CHECK (Email LIKE '%@%.%'),
    CONSTRAINT PostalCodeValid CHECK ( (PostalCode LIKE
'[0-9][0-9]-[0-9][0-9][0-9]'  AND Country = 'Poland') OR Country != 'Poland' ),
    CONSTRAINT Companies_pk PRIMARY KEY  (Client_ID)
);

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

4.IndividualClient

(Reprezentacja klienta indywidualnego)
Klucz główny i obcy: Client_ID
Numer telefonu: Phone
Adres E-mail: Email
Państwo: Country

```
CREATE TABLE IndividualClient (
    Client_ID int  NOT NULL,
    Phone varchar(20)  NOT NULL,
    Email varchar(255)  NOT NULL,
    Country varchar(255)  NOT NULL,
    CONSTRAINT UniqueEmail UNIQUE (Email),
    CONSTRAINT IndividualClientPhoneValid 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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland'  ),
    CONSTRAINT IndividualClientEmailValid CHECK (Email LIKE '%@%.%'),
    CONSTRAINT IndividualClient_pk PRIMARY KEY  (Client_ID)
);

ALTER TABLE IndividualClient ADD CONSTRAINT IndividualCient_Clients
    FOREIGN KEY (Client_ID)
    REFERENCES Clients (Client_ID);
```

5.Discount

(Zniżki przydzielone klientom)
Klucz główny: DiscountID
Klucz obcy: Client_ID
Data rozpoczęcia obowiązywania rabatu: StartDate
Data końca obowiązywania rabatu: EndDate
Procent Rabatu(zapisywany jako liczba przecinkowa): DiscountAmount
Czy rabat już wykorzystany: Used

```sql
CREATE TABLE Discount (
    DiscountID int  NOT NULL IDENTITY,
    StartDate date  NOT NULL,
    EndDate date  NULL,
    DiscountAmount decimal(15,2)  NOT NULL,
    Client_ID int  NOT NULL,
    Used bit  NOT NULL,
    CONSTRAINT DateValid CHECK (ISNULL(EndDate,'9999-12-31 23:59:59') >
StartDate),
    CONSTRAINT Discount_pk PRIMARY KEY  (DiscountID)
);

ALTER TABLE Discount ADD CONSTRAINT IndividualClient_Discount
    FOREIGN KEY (Client_ID)
    REFERENCES IndividualClient (Client_ID);
```

6.Employees

(Reprezentacja pracownika)
Klucz główny: EmployeeID
Nazwa pracownika: Name
Numer telefonu: Phone
Adres E-mail: Email
Państwo: Country

```sql
CREATE TABLE Employees (
    EmployeeID int  NOT NULL IDENTITY,
    Name varchar(255)  NOT NULL,
    Phone varchar(20)  NOT NULL,
    Email varchar(255)  NOT NULL,
    Country varchar(255)  NOT NULL,
    CONSTRAINT EmployeeEmailValid CHECK (Email LIKE '%@%.%'),
    CONSTRAINT EmployeePhoneValid 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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland' ),
    CONSTRAINT Employees_pk PRIMARY KEY  (EmployeeID)
);
```

7.Menu

(Reprezentacja Menu)
Klucz główny: MenuID
Data rozpoczęcia obowiązywania menu: StartDate
Data końca obowiązywania menu: EndDate

```
CREATE TABLE Menu (
    MenuID int  NOT NULL IDENTITY,
    StartDate date  NOT NULL,
    EndDate date  NOT NULL,
    CONSTRAINT ValidDate CHECK (EndDate >= StartDate),
    CONSTRAINT Menu_pk PRIMARY KEY  (MenuID)
);
```

8.MenuDetails

(Zawiera potrawy należące do danego menu)
Klucz główny i obcy: MenuID, ProductID
Cena: Cost

```sql
CREATE TABLE MenuDetails (
    ProductID int  NOT NULL,
    MenuID int  NOT NULL,
    Cost money  NOT NULL,
    CONSTRAINT CostValid CHECK ((Cost >= 0)),
    CONSTRAINT MenuDetails_pk PRIMARY KEY  (ProductID,MenuID)
);

ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Menu
    FOREIGN KEY (MenuID)
    REFERENCES Menu (MenuID);

ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Product
    FOREIGN KEY (ProductID)
    REFERENCES Product (ProductID);
```

9.OpeningHours

(Zawiera informacje o godzinach otwarcia)
Klucz główny: OpeningHoursID
Data rozpoczęcia obowiązywania godzin otwarcia: StartDate
Data końca obowiązywania godzin otwarcia: EndDate

```
CREATE TABLE OpeningHours (
    OpeningHoursID int  NOT NULL IDENTITY,
    StartDate date  NOT NULL,
    EndDate date  NULL,
    CONSTRAINT OpeningHoursDateValid CHECK (ISNULL(EndDate,'9999-12-31
23:59:59') > StartDate),
    CONSTRAINT OpeningHours_pk PRIMARY KEY  (OpeningHoursID)
);
```

10.OpeningHoursDetails

(Zawiera informacje o godzinach otwarcia w poszczególne dni tygodnia dla wybranych dni kalendarzowych)
Klucz główny: OpeningHoursDetailsID
Klucz obcy: OpeningHoursID
Dzień tygodnia(1-7): Day
Godzina otwarcia: StartHour
Godzina zamknięcia: EndHour

```sql
CREATE TABLE OpeningHoursDetails (
    OpeningHoursDetailsID int  NOT NULL,
    OpeningHoursID int  NOT NULL,
    Day int  NOT NULL,
    StartHour time(0)  NOT NULL,
    EndHour time(0)  NOT NULL,
    CONSTRAINT DayValid CHECK (Day >= 1 AND Day <= 7),
    CONSTRAINT OpeningHoursDetails_pk PRIMARY KEY  (OpeningHoursDetailsID)
);

ALTER TABLE OpeningHoursDetails ADD CONSTRAINT OpeningHoursDetails_OpeningHours
    FOREIGN KEY (OpeningHoursID)
    REFERENCES OpeningHours (OpeningHoursID);
```

11.OrderDetails

(Produkty oraz ich ilość w poszczególnych zamówieniach)
Klucz główny i obcy: ProductID, OrderID
Ilość danego produktu: Quantity
Czy dany produkt powinien być podany na wynos: ToGo

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

ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID)
    ON DELETE  CASCADE;

ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Product
    FOREIGN KEY (ProductID)
    REFERENCES Product (ProductID);
```

12.Orders

(Zawiera informacje o zamówieniach)
Klucz główny: OrderID
Klucze obce: Client_ID, PickupID, ReservationID, DiscountID
Data złożenia zamówienia: OrderDate
Czy zamówienie zostało opłacone: Paid
Czy dany produkt powinien być podany na wynos: ToGo

```sql
CREATE TABLE Orders (
    OrderID int  NOT NULL IDENTITY,
    OrderDate date  NOT NULL,
    Client_ID int  NOT NULL,
    PickupID int  NULL,
    Paid bit  NOT NULL,
    ReservationID int  NULL,
    DiscountID int  NULL,
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);

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

ALTER TABLE Orders ADD CONSTRAINT Orders_Discount
    FOREIGN KEY (DiscountID)
    REFERENCES Discount (DiscountID);

ALTER TABLE Orders ADD CONSTRAINT Orders_OrderPickup
    FOREIGN KEY (PickupID)
    REFERENCES Pickup (PickupID);

ALTER TABLE Orders ADD CONSTRAINT Orders_Reservation
    FOREIGN KEY (ReservationID)
    REFERENCES Reservation (ReservationID);
```

13.Pickup

(Przechowuje informacje na temat produktów zakupionych na wynos do późniejszego odbioru)
Klucz główny: PickupID
Przewidywana data odbioru (wraz z godziną): PickupDate
Czy zamówienie zostało odebrane: Picked

```sql
CREATE TABLE Pickup (
    PickupID int  NOT NULL IDENTITY,
    PickupDate datetime  NOT NULL,
    Picked bit  NOT NULL,
    CONSTRAINT PickupDateValid CHECK (PickupDate > GETDATE()),
    CONSTRAINT Pickup_pk PRIMARY KEY  (PickupID)
);
```

14.Product

(Przechowuje informacje na temat dostępnych dań w restauracji)
Klucz główny: ProductID
Klucz obcy: CategoryID
Nazwa produktu: ProductName
Opis produktu: Description

```
CREATE TABLE Product (
    ProductID int  NOT NULL IDENTITY,
    CategoryID int  NOT NULL,
    ProductName varchar(255)  NOT NULL,
    Description varchar(255)  NOT NULL,
    CONSTRAINT Product_pk PRIMARY KEY  (ProductID)
);

ALTER TABLE Product ADD CONSTRAINT Product_Category
    FOREIGN KEY (CategoryID)
    REFERENCES Category (CategoryID);
```

15.Reservation

(Przechowuje informacje na temat poszczególnej rezerwacji)

Klucz główny: ReservationID

Klucz obcy: EmployeeID

Data początku rezerwacji (wraz z godziną): ReservationStartDate

Data końca rezerwacji (wraz z godziną): ReservationEndDate

Ilość gości: ReservationGuestNumber

```
CREATE TABLE Reservation (
    ReservationID int  NOT NULL IDENTITY,
    ReservationStartDate datetime  NOT NULL,
    ReservationEndDate datetime  NOT NULL,
    ReservationGuestNumber int  NOT NULL,
    EmployeeID int  NULL,
    CONSTRAINT ReservationDateValid CHECK (ReservationEndDate >
ReservationStartDate ),
    CONSTRAINT ReservationGuestNumberValid CHECK (ReservationGuestNumber >= 2),
    CONSTRAINT Reservation_pk PRIMARY KEY  (ReservationID),
    CONSTRAINT ReservationStartDateAfterNow CHECK (ReservationStartDate >=
GETDATE()),
);

ALTER TABLE Reservation ADD CONSTRAINT Reservation_Employees
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

16.Reservation Details

(Przechowuje informacje na temat przyporządkowania stolików do konkretnych rezerwacji)
Klucz główny: ReservationDetailsID
Klucze obce: ReservationID, TableID
Nazwa rezerwacji: ReservationName

```
CREATE TABLE ReservationDetails (
    ReservationDetailsID int  NOT NULL IDENTITY,
    ReservationID int  NOT NULL,
    TableID int  NOT NULL,
    ReservationName varchar(255)  NOT NULL,
    CONSTRAINT ReservationDetails_pk PRIMARY KEY  (ReservationDetailsID)
);

ALTER TABLE ReservationDetails ADD CONSTRAINT ReservationDetails_Reservation
    FOREIGN KEY (ReservationID)
    REFERENCES Reservation (ReservationID)
    ON DELETE  CASCADE;

ALTER TABLE ReservationDetails ADD CONSTRAINT ReservationDetails_Table
    FOREIGN KEY (TableID)
    REFERENCES "Table" (TableID);
```

17.Table

(Przechowuje informacje na temat dostępnych ilości miejsc przy określonym stoliku)
Klucz główny: TableID
Ilość dostępnych krzeseł przy danym stoliku: CharisNumber

```
CREATE TABLE "Table" (
    TableID int  NOT NULL IDENTITY,
    ChairsNumber int  NOT NULL,
    CONSTRAINT ChairsNumberValid CHECK ((ChairsNumber >= 2) and (ChairsNumber
<= 12) ),
    CONSTRAINT TableID PRIMARY KEY  (TableID)
);
```

18.Variables

(Przechowuje stałe, które są uwzględniane do rezerwacji oraz przy udzielaniu rabatów)

Klucz główny: VariablesID

Ilość zamówień wymagana do rezerwacji stolika: WK

Minimalna wartość zamówienia wymagana do rezerwacji stolika: WZ

Ilość zamówień wymagana do zniżki nr 1: Z1

Minimalna kwota zamówienia, wymagana do przyznania zniżki nr 1: K1

Ilość dni, przez które obowiązuje zniżka nr 2: D1

Wartość zniżki nr 1 (zapisywana jako liczba przecinkowa): R1

Wartość zniżki nr 2 (zapisywana jako liczba przecinkowa): R2

Czas od którego obowiązują stałe znajdujące się w Variables: StartDate

Minimalna kwota zamówienia, wymagana do przyznania zniżki nr 2: K2

```
CREATE TABLE Variables (
    WK int  NOT NULL,
    WZ int  NOT NULL,
    Z1 int  NOT NULL,
    K1 decimal(15,2)  NOT NULL,
    D1 Int  NOT NULL,
    R1 decimal(15,2)  NOT NULL,
    R2 decimal(15,2)  NOT NULL,
    StartDate date  NOT NULL,
    K2 decimal(15,2)  NOT NULL,
    VariablesID int  NOT NULL IDENTITY,
    CONSTRAINT ValuesValid CHECK ((WK > 0 AND WZ > 0 AND Z1 > 0 AND K1 > 0 AND
D1 > 0 AND R1 > 0 AND R1 <= 1 AND R2>0 AND R2 <=1)),
    CONSTRAINT Variables_pk PRIMARY KEY  (VariablesID)
);
```

4. Więzy integralności

1. Menu details

Koszt produktu musi być liczbą nieujemną

```
CHECK (Cost >= 0)
```

2. Menu

Menu musi trwać w przedziale czasowym

```
CHECK (EndDate >= StartDate)
```

3. Order details

Ilość zamówionych produktów musi być liczbą naturalną

```
 CHECK( Quantity > 0)
```

4. Pickup

Odbiór zamówienia może nastąpić dopiero po jego złożeniu

```
 CHECK(PickupDate > NOW())
```

5. Reservation

Rezerwacja musi trwać określony czas, jest możliwa tylko w przypadku gdy liczba gości jest nie mniejsza niż 2

```
CHECK(ReservationEndDate > ReservationStartDate)
CHECK (ReservationGuestNumber) >= 2
CHECK ReservationStartDate >= GETDATE()
```

6. Employees

Weryfikacja adresu Email oraz numeru telefonu

```
CHECK (Email LIKE '%@%.%')
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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland'
```

7. IndividualClient

Weryfikacja adresu e-mail oraz numeru telefonu

```
CHECK (Email LIKE '%@%.%')
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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland'
UNIQUE(Email)
```

8. Companies

Weryfikacja adresu e- mail, NIP, kodu pocztowego, NIP musi być unikalny

```
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]' OR Phone LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') AND Country = 'Poland') OR
Country != 'Poland'
CHECK (NIP LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' AND
Country = 'Poland') OR Country != 'Poland'
UNIQUE (NIP)
CHECK (Email LIKE '%@%.%')
CHECK (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'  AND Country =
'Poland') OR Country != 'Poland'
```

9. Discount

Przyznana zniżka musi być na określony czas

```
CHECK (ISNULL(EndDate,'9999-12-31 23:59:59') > StartDate)
```

10. Variables

Weryfikacja określonych stałych

```
CHECK(WK > 0 AND WZ > 0 AND Z1 > 0 AND K1 > 0 AND D1 > 0 AND R1 > 0 AND R1
<= 1 AND R2>0 AND R2 <=1)
```

5. Widoki

1.CurrentMenu

(Wyświetla pełne informacje o pozycjach z menu: nazwę, cenę, kategorię, opis)

```
Create view CurrentMenu as
Select P.ProductName, M.Cost, C.CategoryName, P.Description, MenuID from
MenuDetails M inner join Product P on M.ProductID = P.ProductID inner join
Category C on P.CategoryID = C.CategoryID where M.MenuID in (select MenuID
from Menu where CAST( GETDATE() AS Date ) >= StartDate AND CAST( GETDATE()
AS Date ) <= EndDate)
```

2.MealsCatalog

(Wyświetla informacje o wszystkich możliwych daniach)

```sql
Create view MealsCatalog as
Select P.ProductName, P.Description, C.CategoryName from Product as P inner
join Category C on C.CategoryID = P.CategoryID
```

3. ShowDiscounts

(Wyświetla informacje na podstawie których można przyznać rabat)

```
Create view ShowDiscounts as
Select WZ as orderValue, WK orderAmount, Z1 as nbOrOrders1, K1 as amount1,
R1 as discount1, K2 as amount2, R2 as discount2, D1 as days from Variables
```

4.IndividualClientInfo

(Wyświetla informacje I statystyki klientów)

```
Create view IndividualClientInfo as
select C.Client_ID, C.ClientName, IC.Phone, IC.Email, IC.Country,
count(O.OrderID) as ilosc_zamowien from Clients as C
      inner join Orders as O on C.Client_ID = O.Client_ID inner join
IndividualClient IC on C.Client_ID = IC.Client_ID
      Group by C.Client_ID, C.ClientName, IC.Phone, IC.Email, IC.Country
```

5. UnconfirmedReservation

(Wyświetla niepotwierdzone rezerwacje)

```sql
Create view UnconfirmedReservation as
Select * from Reservation where EmployeeID is null;
```

6. TodayReservations

(Wyświetla dzisiejsze potwierdzone rezerwacje)

```sql
Create view TodayReservations as
Select ReservationStartDate, ReservationEndDate, ReservationGuestNumber
from Reservation where EmployeeID is not null and convert(date,
ReservationStart) = convert(date, getdate());
```

7. OrdersToPay

(Wyświetla nieopłacone zamówienia)

```sql
Create view OrdersToPay as
Select OrderID, Client_ID from Orders where paid = 0
```

8. PendingPickup

(Wyświetla zamówienia na wynos które nie zostały odebrane)

```sql
Create view PendingPickup as
Select O.OrderID, P.PickupID from Pickup as P
inner join Orders as O on O.PickupID = P.PickupID where O.Paid = 1 and
Convert(date, PickupDate) = Convert(date, GETDATE());
```

9. ProductsSold

(Wyświetla informacje dotyczące ilości sprzedaży poszczególnych produktów)

```sql
Create view ProductsSold as
Select ProductID, Sum(Quantity) as ilość from OrderDetails group by
ProductID
```

10. ProductsSoldDaily

(Wyświetl informacje na temat miesięcznej sprzedaży produktów)

```
Create view ProductsSoldDaily as
Select OD.ProductID, Sum(OD.Quantity) as quantity, day(O.OrderDate) as day,
month(O.OrderDate) as month, year(O.OrderDate) as year  from OrderDetails
OD inner join Orders O on OD.OrderID = O.OrderID group by OD.ProductID,
day(O.OrderDate), month(O.OrderDate), year(O.OrderDate)
```

11. ProductsSoldMonthly

(Wyświetla informacje na temat miesięcznej sprzedaży produktów)

```
Create view ProductsSoldMonthly as
Select OD.ProductID, Sum(OD.Quantity) as quantity, month(O.OrderDate) as
month, year(O.OrderDate) as year  from OrderDetails OD inner join Orders O
on OD.OrderID = O.OrderID group by OD.ProductID, month(O.OrderDate),
year(O.OrderDate)
```

12. ProductsSoldAnnually

(Wyświetla ilość produktów sprzedawanych rocznie)

```
Create view ProductsSoldAnnually as
Select OD.ProductID, Sum(OD.Quantity) as quantity,  year(O.OrderDate) as
year from OrderDetails OD inner join Orders O on OD.OrderID = O.OrderID
group by OD.ProductID, year(O.OrderDate)
```

13. AnnualIncome

(Wyświetla roczny przychód)

```sql
Create view AnnuallIncome as
Select Sum(sale) as Income, year from
(Select Sum(IIF(O.DiscountID is not null,
(OD.Quantity * MD.cost * (1 - D.DiscountAmount)), (OD.Quantity * MD.cost)))
as sale,year(O.OrderDate) as year from OrderDetails OD
inner join Orders O on OD.OrderID = O.OrderID
inner join Product P on OD. ProductID = P.ProductID
inner join MenuDetails MD on P.ProductID = MD.ProductID
left join Discount D on D.DiscountID = O.DiscountID
group by OD.Quantity, MD.Cost, year(O.OrderDate)) as sy
group by year;
```

14. OrdersInfo

(Wyświetla ceny zamówień)

```
Create view OrdersInfo as
SELECT OrderID,
(SELECT Sum(IIF(O.DiscountID is not null, (OD2.Quantity * MD.Cost * (1 -
DiscountAmount)), (OD2.Quantity * MD.Cost))) value FROM OrderDetails OD2
inner join Orders O on O.OrderID = OD2.OrderID
INNER JOIN Product P on OD2.ProductID = P.ProductID
INNER JOIN MenuDetails MD on P.ProductID = MD.ProductID
INNER JOIN Menu M on M.MenuID = MD.MenuID
LEFT JOIN Discount D on O.DiscountID = D.DiscountID
 where O.OrderID = Orders.OrderID group by O.OrderID) value,
Client_ID, OrderDate, Paid FROM Orders;
```

15. DiscountsInfo

(Wyświetla informacje o dostępnych zniżkach jednorazowych dla klientów)

```
Create view DiscountsInfo as
select ClientName, DiscountAmount, StartDate, EndDate from Discount inner
join IndividualClient IC on IC.Client_ID = Discount.Client_ID inner join
Clients C on C.Client_ID = IC.Client_ID where Used = 0
```

16. CurrentVariables

(Wyświetla aktualne zmienne, na podstawie których można przyznawać rabaty)

```
Create view CurrentVariables as
Select top 1 * from variables order by StartDate desc
```

17. WeeklyTableReservations

(Wyświetla ilość tygodniowo zarezerwowanych miejsc przy stolikach)

```
Create view WeeklyTableReservations as
      select Sum(ChairsNumber) as chairs, COUNT(*) as tables,
DATEPART(week, ReservationStartDate) as nbOfWeek,
year(ReservationStartDate) as year from Reservation
      inner join ReservationDetails RD on Reservation.ReservationID =
RD.ReservationID
      inner join [Table] on RD.TableID = [Table].TableID
      group by DATEPART(week, ReservationStartDate),
year(ReservationStartDate)
```

18. MonthlyTableReservations

(Wyświetla ilość miesięcznie zarezerwowanych miejsc przy stolikach)

```
Create view MonthlyTableReservations as
      select Sum(ChairsNumber) as chairs, month(ReservationStartDate) as
month, year(ReservationStartDate) as year from Reservation
      inner join ReservationDetails RD on Reservation.ReservationID =
RD.ReservationID
      inner join [Table] on RD.TableID = [Table].TableID
      group by  month(ReservationStartDate), year(ReservationStartDate)
```

6. Procedury

1.AddCategory

(Dodaje nową kategorię produktów)

```
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
       INSERT INTO Category(CategoryName) VALUES(@CategoryName);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) = N'Błąd dodawania kategorii: ' +
ERROR_MESSAGE(); THROW 52000, @msg, 1;
   END CATCH
END
go
```

2.AddProduct

(Dodaje nowy produkt)

```
CREATE PROCEDURE uspAddProduct
@Name varchar(255),
@Description varchar(255),
@CategoryName varchar(255) AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
      IF EXISTS( SELECT * FROM Product WHERE ProductName = @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
      Set @CategoryID = (select CategoryID from Category where CategoryName
= @CategoryName)
      INSERT INTO Product(ProductName, CategoryID, Description) VALUES
(@Name, @CategoryID, @Description);
  END TRY
  BEGIN CATCH
      DECLARE @msg nvarchar(2048) =N'Błąd dodania potrawy: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
END
```

3.AddProductToMenuById

(Dodaje produkt do menu na podstawie menu id)

```
 CREATE PROCEDURE uspAddProductToMenuById
@Name varchar(255),
@Cost money,
@MenuID int AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT * FROM Product WHERE ProductName = @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 Product
WHERE ProductName = @Name
       INSERT INTO MenuDetails(ProductID , MenuID, Cost)
       VALUES (@ProductID, @MenuID, @Cost);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) =N'Błąd dodania potrawy do menu: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
```

4.AddProductToMenuByDate

(Dodaje produkt do menu na podstawie daty menu)

```
CREATE PROCEDURE uspAddProductToMenuByDate
@Name varchar(255),
@Date Date,
@Cost money AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS( SELECT * FROM Product WHERE ProductName = @Name )
           BEGIN
               THROW 52000, 'Nie ma takiej potrawy', 1
           END
       DECLARE @ProductID INT;
       SELECT @ProductID = ProductID FROM Product WHERE ProductName = @Name
       DECLARE @MenuID INT
       SELECT @MenuID = MenuID FROM Menu WHERE StartDate >= @Date and
EndDate <= @Date;
       INSERT INTO MenuDetails(ProductID , MenuID, Cost)
       VALUES (@ProductID, @MenuID, @Cost);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) =N'Błąd dodania potrawy do menu: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
```

5.AddOrder

(Dodaje nowe zamówienie)

```sql
CREATE PROCEDURE uspAddOrder @ClientID int,
@Paid bit,
@PickupDate datetime,
@StartDate datetime,
@EndDate datetime,
@ReservationGuestNumber int AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
      IF ISNULL(@PickupDate,'9999-01-01') < GETDATE()
          BEGIN
              THROW 52000, N'Niepoprawna data odbioru zamówienia na wynos',
1
          END
      IF @PickupDate is not null
          BEGIN
           IF not exists(select StartHour, EndHour from OpeningHoursDetails
             where OpeningHoursID =
                   (select top 1 OpeningHoursID
                   from OpeningHours
                   where @PickupDate BETWEEN StartDate and ISNULL(EndDate,
'9999-12-31 23:59:59'))
             and Day = datepart(weekday, CONVERT(VARCHAR(8), @PickupDate,
108))
             and CONVERT(VARCHAR(8), @PickupDate, 108) between StartHour
and EndHour)
             BEGIN
               THROW 52000, N'Niepoprawna data odbioru zamówienia na
wynos', 1
             end
          end
      IF ISNULL(@EndDate,'9999-01-01') < GETDATE() OR
ISNULL(@StartDate,'9999-01-01') < GETDATE()
          BEGIN
              THROW 52000, N'Niepoprawna data rezerwacji', 1
          END
      IF @EndDate is not null
          BEGIN
```

```sql
           if exists(select * from IndividualClient where Client_ID =
@ClientID) and ((select Count(*) from OrdersInfo where Client_ID =
@ClientID) < (select wk from CurrentVariables))
               begin
                   THROW 52000, N'Klient ma za mało zamówień aby móc wykonać
rezerwacje', 1
               end
           IF not exists(select StartHour, EndHour from OpeningHoursDetails
             where OpeningHoursID =
                   (select top 1 OpeningHoursID
                   from OpeningHours
                   where @StartDate BETWEEN StartDate and ISNULL(EndDate,
'9999-12-31 23:59:59'))
             and Day = datepart(weekday, CONVERT(VARCHAR(8), @StartDate,
108))
             and CONVERT(VARCHAR(8), @StartDate, 108) between StartHour and
EndHour
             and CONVERT(VARCHAR(8), @EndDate, 108) between StartHour and
EndHour)
             BEGIN
               THROW 52000, N'Niepoprawna data rezerwacji', 1
             end
          end
      Declare @ReservationIDIns INT = null
      Declare @PickupIDIns INT = null
      Declare @Discount Decimal(15,2) = null

      DECLARE @CurrentMenuID int
      SELECT TOP 1 @CurrentMenuId = MenuID FROM Menu M WHERE GETDATE()
BETWEEN M.StartDate AND M.EndDate
      IF (@PickupDate is not null)
          BEGIN
              INSERT INTO Pickup(PickupDate, Picked) VALUES (@PickupDate,
0)
              SET @PickupIDIns = SCOPE_IDENTITY();
          END
      IF (@StartDate is not null)
          BEGIN
              EXEC uspAddReservation @StartDate, @EndDate,
@ReservationGuestNumber
               SET @ReservationIDIns = IDENT_CURRENT('Reservation')
          END
      IF EXISTS(SELECT * FROM IndividualClient WHERE Client_ID =  @ClientID
)
           BEGIN
               SET @Discount = [dbo].udfGetBestDiscount(@ClientID)
           END
      INSERT INTO Orders(OrderDate, Client_ID, PickupID, Paid,
ReservationID, DiscountID)
      VALUES (GETDATE(),  @ClientID, @PickupIDIns, @Paid,
@ReservationIDIns, @Discount)
  END TRY
  BEGIN CATCH
      DECLARE @msg nvarchar(2048) =N'Błąd dodawania zamówienia: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
  END CATCH
END
go

grant execute on uspAddOrder to moderator
go

grant execute on uspAddOrder to worker
go
```

6.AddProductToOrder

(Dodaj produkt do zamówienia)

```
CREATE PROCEDURE uspAddProductToOrder
@OrderID int,
@Quantity int,
@ProductName varchar(255),
@ToGo bit AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
      IF NOT EXISTS( SELECT * FROM Product WHERE ProductName = @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
      DECLARE @temp datetime;
      DECLARE @menuIDToCheck int;
      IF (select PickupDate from Pickup P, Orders O where O.PickupID =
P.PickupID and OrderID = @OrderID) is not null
          BEGIN
               set @temp = (select PickupDate from Pickup P, Orders O where
O.PickupID = P.PickupID and OrderID = @OrderID)
               select @menuIDToCheck = MenuID from Menu where @temp between
StartDate and EndDate
          end
      ELSE
          BEGIN
               IF (select P.ReservationStartDate from Reservation P, Orders
O where O.ReservationID = P.ReservationID and OrderID = @OrderID) is not
null
                         BEGIN
                             set @temp = (select P.ReservationStartDate
from Reservation P, Orders O where O.ReservationID = P.ReservationID and
OrderID = @OrderID)
               select @menuIDToCheck = MenuID from Menu where @temp between
StartDate and EndDate
```

```
                          end
                ELSE
                    BEGIN
                        SET @menuIDToCheck = (select MenuID from Menu where
GETDATE() between StartDate and EndDate)
                    end
          end
      IF @menuIDToCheck is null
           BEGIN
               THROW 52000, N'Nie mozna zamowic tego produktu, gdyz menu na
dany dzień nie zsotało jeszcze dodane', 1
           end
      IF NOT EXISTS( SELECT * FROM udfGetMenuItemsById(@menuIDToCheck)
WHERE ProductName = @ProductName )
          BEGIN
              THROW 52000, N'Nie mozna zamowic tego produktu, gdyz nie ma
go w menu na dany dzień', 1
          END
      IF EXISTS( select * from Product where ProductName = @ProductName and
CategoryID = (select CategoryID from Category where CategoryName = 'Ryby'))
          BEGIN
          DECLARE @DateOfExecutingOrder datetime;
          set @DateOfExecutingOrder = null;
          IF (Select PickupID from Orders where OrderID = @OrderID) is not
null
              BEGIN
                  SET @DateOfExecutingOrder = (select PickupDate from
Pickup where PickupID = (Select PickupID from Orders where OrderID =
@OrderID))
              end
          IF (Select ReservationID from Orders where OrderID = @OrderID) is
not null
              BEGIN
                  SET @DateOfExecutingOrder = (select ReservationStartDate
from Reservation where ReservationID = (Select ReservationID from Orders
where OrderID = @OrderID))
              end
          IF @DateOfExecutingOrder is null
              BEGIN
                   THROW 52000, N'Brak daty odbioru owoców morza', 1
              end
          IF DATEPART(WEEKDAY ,@DateOfExecutingOrder) != 4 AND
DATEPART(WEEKDAY , @DateOfExecutingOrder) != 5 AND DATEPART(WEEKDAY
```

```sql
,@DateOfExecutingOrder) != 6
              BEGIN
                  THROW 52000, N'Nieprawidłowa data złożenia zamówienia na
owoce morza', 1
              end
           IF DATEPART(WEEKDAY , GETDATE()) != 1 and DATEPART(WEEKDAY ,
GETDATE()) != 7 and DATEPART(week, GETDATE()) = DATEPART(week,
@DateOfExecutingOrder)
              begin
                 THROW 52000, N'Nieprawidłowa data złożenia zamówienia na
owoce morza', 1
              end
          END
      DECLARE @ProductID INT
      SELECT @ProductID = ProductID FROM Product WHERE ProductName =
@ProductName
      INSERT INTO OrderDetails(OrderID, Quantity, ProductID, ToGo)
      VALUES (@OrderID,@Quantity,@ProductID, @ToGo)
  END TRY
  BEGIN CATCH
      DECLARE @msg nvarchar(2048) =N'Błąd dodania produktu do zamowienia: '
+ ERROR_MESSAGE(); THROW 52000, @msg, 1
  END CATCH
END
```

7. AddEmployee

(Dodaje nowego pracownika)

```
create procedure uspAddEmployee
	@Name varchar(255),
	@Phone varchar(20),
	@EMail varchar(255),
	@Country varchar(255)
as
	BEGIN try
	insert into Employees (Name, Phone, Email, Country)
	values (@Name, @Phone, @EMail, @Country)
	END try
	Begin catch
	DECLARE @msg nvarchar(2048) =N'Błąd dodania pracownika: ' +
ERROR_MESSAGE();
	THROW 52000, @msg, 1
	end catch
go
```

8. AddIndiviudalClient

(Dodaje indywidualnego klienta)

```sql
create procedure uspAddIndividualClient
      @ClientName varchar(255),
      @Phone varchar(20),
      @Email varchar(255),
      @Country varchar(255)
as
      Begin try
      insert into Clients (ClientName)
      values (@ClientName)

      declare @id int
      set @id = (select Max(Client_ID) from Clients)
      insert into IndividualClient (Client_ID, Phone, Email, Country)
      values (@id, @Phone, @Email, @Country)
      End try
      Begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania klienta: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      End catch
go
```

9. AddCompany

(Dodaje nową firmę)

```
create procedure uspAddCompany
      @ClientName varchar(255),
      @ContactName varchar(255),
      @NIP varchar(10),
      @Address varchar(255),
      @City varchar(255),
      @Region varchar(255),
      @PostalCode varchar(255),
      @Phone varchar(20),
      @EMail varchar(255),
      @Country varchar(255)
as
      begin try
      insert into Clients (ClientName)
      values (@ClientName)

      declare @id int
      set @id = (select Max(Client_ID) from Clients)

      insert into Companies (Client_ID, ContactName, NIP, Address, City,
Region, PostalCode, Phone, Email, Country)
      values (@id, @ContactName, @NIP, @Address, @City, @Region,
@PostalCode, @Phone, @EMail, @Country)
      end try
      Begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania klienta: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      End catch
go
```

10. AddTable

(Dodaje nowy stół)

```
create procedure uspAddTable
      @ChairNumbers int
as
      begin try
      insert into [Table] (ChairsNumber)
      values (@ChairNumbers)
      end try
      begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania stolika: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      end catch
go
```

11. AddReservation

(Dodaje nową rezerwację)

```
create procedure uspAddReservation
      @ReservationStartTime datetime,
      @ReservationEndTime datetime,
      @ReservationGuestNumber int
as
      begin try

      Declare @AllChairs int
      Declare @TakenChairs int

      set @AllChairs = (select Sum(ChairsNumber) from [Table])
      set @TakenChairs = (select Sum(ChairsNumber) from [Table] inner join
ReservationDetails RD on [Table].TableID = RD.TableID
      inner join Reservation R2 on R2.ReservationID = RD.ReservationID
where ReservationStartDate between @ReservationStartTime and
@ReservationEndTime
      or ReservationEndDate between @ReservationStartTime and
ReservationEndDate)
      if(@AllChairs - @TakenChairs < @ReservationGuestNumber)
      begin
            THROW 52000, N'Zbyt duża ilość gości, brak miejsc', 1
      end
   insert into Reservation (ReservationStartDate, ReservationEndDate,
ReservationGuestNumber, EmployeeID)
   values (@ReservationStartTime, @ReservationEndTime,
@ReservationGuestNumber, null)
      end try
      begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania rezerwacji: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      end catch
go
```

12. AddReservationDetails

(Dodaje ReservationDetails do istniejącej rezerwacji)

```sql
create procedure uspAddReservationDetails
      @ReservationID int,
      @TableID int,
      @ReservationName varchar(255)
as
      begin try
      Declare @NbOfReservations int
      Declare @StartTime datetime
      Declare @EndTime datetime

      set @StartTime = (select ReservationStartDate from Reservation where
ReservationID = @ReservationID)
      set @EndTime = (select ReservationEndDate from Reservation where
ReservationID = @ReservationID)
      set @NbOfReservations = (select count(*) from ReservationDetails RD
      inner join Reservation R on RD.ReservationID = R.ReservationID where
(R.ReservationStartDate between @StartTime and @EndTime
      or R.ReservationEndDate between @StartTime and @EndTime) and
RD.TableID = @TableID)
      if(@NbOfReservations > 0)
      begin
            THROW 52000, N'Ten stolik jest już zajęty', 1
      end
      insert into ReservationDetails (ReservationID, TableID,
ReservationName)
      values (@ReservationID, @TableID, @ReservationName)
      end try
      begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania rezerwacji: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      end catch
go
```

13. ConfirmReservation

(Potwierdza rezerwację)

```
create procedure uspConfirmReservation
      @ReservationID int,
      @EmployeeID int
      as
      begin try
      update Reservation
      set EmployeeID = @EmployeeID
      where ReservationID = @ReservationID
      end try
      begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd potwierdzania: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      end catch
```

14. ConfirmPickUp

(Potwierdza odebranie zamówienia)

```
      create procedure uspConfirmPickUp
      @PickupID int
      as
      begin try
      update Pickup
      set Picked = 1
      where PickupID = @PickupID
      end try

      begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd potwierdzania: ' +
ERROR_MESSAGE();
      THROW 52000, @msg, 1
      end catch
```

15.AddDiscountsToClient

(Dodaje zniżkę dla klienta)

```sql
create procedure uspAddDiscountsToClient
  @ClientID int
as
  Begin try
  IF not exists(select * from IndividualClient where
IndividualClient.Client_ID = @ClientID)
      BEGIN
           THROW 52000, N'Klient nie istnieje lub nie jest klientem
indywidualnym', 1
       end
  IF not exists(select * from Discount where Discount.Client_ID = @ClientID
and DiscountAmount = (select R1 from CurrentVariables))
      begin
           DECLARE @NumOfOrdersForR1 int
          select @NumOfOrdersForR1 = count(OrdersInfo.OrderID) from
OrdersInfo where OrdersInfo.Client_ID = @ClientID and OrdersInfo.value >=
(select K1 from CurrentVariables)
           if @NumOfOrdersForR1 >= (select Z1 from CurrentVariables)
          begin
               insert into Discount(StartDate, EndDate, DiscountAmount,
Client_ID) values (getdate(), null, (select R1 from CurrentVariables),
@ClientID)
           end
      end
  IF not exists(select * from Discount where Discount.Client_ID = @ClientID
and DiscountAmount = (select R2 from CurrentVariables))
      begin
           DECLARE @NumOfOrdersForR22 int
          select @NumOfOrdersForR22 = sum(OrdersInfo.value) from OrdersInfo
where OrdersInfo.Client_ID = @ClientID
           if @NumOfOrdersForR22 >= (select K2 from CurrentVariables)
          begin
               insert into Discount(StartDate, EndDate, DiscountAmount,
Client_ID) values (getdate(), dateadd(day, 7, getdate()), (select R2 from
CurrentVariables), @ClientID)
           end
      end
  IF exists(select * from Discount where Discount.Client_ID = @ClientID and
DiscountAmount = (select R2 from CurrentVariables))
      begin
          declare @DateOfRecentR2 date
          select top 1 @DateOfRecentR2 = StartDate from Discount where
Discount.Client_ID = @ClientID order by StartDate DESC;
           DECLARE @NumOfOrdersForR2 int
          select @NumOfOrdersForR2 = sum(OrdersInfo.value) from OrdersInfo
where OrdersInfo.Client_ID = @ClientID and OrdersInfo.OrderDate >
@DateOfRecentR2
           if @NumOfOrdersForR2 >= (select K2 from CurrentVariables)
          begin
               insert into Discount(StartDate, EndDate, DiscountAmount,
Client_ID) values (getdate(), dateadd(day, 7, getdate()), (select R2 from
CurrentVariables), @ClientID)
           end
      end
  End try
  Begin catch
      DECLARE @msg nvarchar(2048) =N'Błąd dodania Discount: ' +
ERROR_MESSAGE();
  THROW 52000, @msg, 1
  end catch
```

16. AddMenu

(Dodaje nowe menu)

```
ALTER PROCEDURE uspAddMenu
    @startDate Date,
    @dishesNumber int
AS
BEGIN
    DECLARE @isGettingRandomRecordsFinished BIT
    SET @isGettingRandomRecordsFinished = 1
    CREATE TABLE #newMenuCandidateProducts (ProductID INT, CategoryID INT,
ProductName varchar(255),

Description varchar(255))
    WHILE @isGettingRandomRecordsFinished = 1
    BEGIN
        INSERT INTO #newMenuCandidateProducts SELECT top (@dishesNumber) *
FROM Product ORDER BY newid()
          DECLARE @numberOfRepetedItems INT
        SET @numberOfRepetedItems = 0
          SELECT @numberOfRepetedItems = COUNT(*) FROM CurrentMenu as CM
                                                  WHERE EXISTS(SELECT *
FROM #newMenuCandidateProducts as tmp

WHERE tmp.ProductName LIKE CM.ProductName)
          IF @numberOfRepetedItems > @dishesNumber / 2
                SET @isGettingRandomRecordsFinished = 1
          ELSE
                SET @isGettingRandomRecordsFinished = 0
    END
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS( SELECT * FROM Menu WHERE @startDate = StartDate )
            BEGIN ;
            THROW 52000, N'Menu z tą samą początkową datą jest już dodane',
1
            END
        INSERT INTO Menu(StartDate, EndDate) VALUES(@startDate,
DATEADD(week, 2, @startDate));
    END TRY
    BEGIN CATCH
```

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

    DECLARE @MenuID INT
    SELECT @MenuID = MenuID FROM Menu WHERE StartDate = @startDate

    INSERT INTO MenuDetails SELECT tmp.ProductID, @MenuID, (SELECT TOP 1
Cost FROM MenuDetails as MD WHERE tmp.ProductID = MD.ProductID ORDER BY
MenuID DESC)
        FROM #newMenuCandidateProducts as tmp;

END
Go
```

17. AddVariables

(Dodaje nowe zmienne na podstawie których możliwe są do przyznania rabaty)

```
create procedure uspAddVariables
   @WK int,
   @WZ int,
   @Z1 int,
   @K1 decimal(15,2),
   @D1 int,
   @R1 decimal(15,2),
   @R2 decimal(15,2),
   @StartDate date,
   @K2 decimal(15, 2)
as
   begin
       insert into Variables (WK, WZ, Z1, K1, D1, R1, R2, StartDate, K2)
       values (@WK, @WZ, @Z1, @K1, @D1, @R1, @R2, @StartDate, @K2)
   end
go
```

7. Funkcje

1.GetMenuById

(Zwraca menu na podstawie id)

```
CREATE FUNCTION udfGetMenuItemsById(@id int)
   RETURNS TABLE AS
       RETURN SELECT M.MenuID, M.StartDate, M.EndDate, (select ProductName
from Product where MD.ProductID = Product.ProductID) as ProductName,
MD.Cost
       from Menu M inner join MenuDetails MD on M.MenuID = MD.MenuID WHERE
M.MenuID = @id;
```

2.GetMenuByDate

(Zwraca menu na podstawie daty)

```
CREATE FUNCTION udfGetMenuItemsByDate(@date date)
RETURNS TABLE AS
RETURN
SELECT M.MenuID, M.StartDate, M.EndDate, P.ProductName, MD.Cost FROM
Menu M inner join MenuDetails MD on M.MenuID = MD.MenuID
INNER JOIN Product P on P.ProductID = MD.ProductID WHERE @date BETWEEN
M.StartDate AND M.EndDate
```

3.GetReceipt

(Zwraca paragon)

```sql
   CREATE FUNCTION udfGetReceipt(@id int)
RETURNS TABLE AS
RETURN
SELECT P.ProductName, OD.Quantity,
     OD.Quantity*(select MD.Cost from MenuDetails MD
         inner join Menu M on M.MenuID = MD.MenuID where O.OrderDate
BETWEEN M.StartDate and M.EndDate and MD.ProductID= P.ProductID) as Cost
from Orders O inner join OrderDetails OD ON OD.OrderID = O.OrderID
Inner join Product as P ON P.ProductID = OD.ProductID where O.OrderId = @id
```

4.GetBestDiscount

(Zwraca zniżkę która jest aktualnie największa)

```
CREATE FUNCTION udfGetBestDiscount(@id int)
RETURNS int AS
BEGIN
DECLARE @val int;
SET @val = null
SET @val = (SELECT TOP 1 D.DiscountID FROM Discount where ((GETDATE()
BETWEEN StartDate AND EndDate) or (GETDATE() > StartDate AND EndDate is
null)) and used = 0 ORDER BY D.DiscountAmount DESC)
RETURN @val
END
```

5.GetOrderValue

(Zwraca wartość określonego zamówienia)

```sql
CREATE FUNCTION udfGetOrderValue(@id int)
RETURNS money AS
BEGIN
RETURN (SELECT value from OrdersInfo where OrderID = @id)
END
```

6.ShowClientDiscounts

(Zwraca dostępne zniżki dla klienta)

```sql
CREATE FUNCTION udfShowClientDiscounts(@id int)
RETURNS TABLE AS
RETURN
SELECT StartDate, EndDate, DiscountAmount from Discount where Used = false
and Client_ID = @id;
```

7.ShowClientHistory

(Zwraca historię zamówień klienta)

```
CREATE FUNCTION udfShowClientHistory (@ClientID INT)
RETURNS TABLE
AS
RETURN
  SELECT OrderID,value FROM OrdersInfo WHERE Client_ID = @ClientID;
```

8. GetInvoiceByOrderId

(Zwraca dane potrzebne do wystawienia faktury)

```
CREATE FUNCTION udfGetInvoiceByOrderId(@OrderId int)
  RETURNS TABLE AS
      RETURN SELECT (select dbo.[udfGetOrderValue](@OrderId)) / 1.23 as
orderValueNetto,
                    (select dbo.[udfGetOrderValue](@OrderId)) - (select
dbo.[udfGetOrderValue](@OrderId)) / 1.23 as VAT,
                    Co.ContactName, Co.Country, Co.NIP, Co.City,
Co.Address, Co.PostalCode  FROM Orders
          INNER JOIN Clients C on C.Client_ID = Orders.Client_ID
          INNER JOIN Companies Co on C.Client_ID = Co.Client_ID
                                                            WHERE OrderID
= @OrderId;
```

8. Triggery

1.ApplyDiscount

(Dodaje do zamówienia najwyższą zniżkę dostępną dla klienta przy składaniu zamówienia)

```
create trigger TR_applyDiscount
   ON Orders
   for INSERT
   as
   BEGIN
       Declare
       @ClientID int,
       @DiscountID int
       SET NOCOUNT ON;
       SELECT @ClientID= INSERTED.[Client_ID],
       @DiscountID= INSERTED.[DiscountID] FROM INSERTED
       IF exists(select * from IndividualClient where
IndividualClient.Client_ID = @ClientID)
           Begin
               if (Select EndDate from Discount where Discount.DiscountID =
@DiscountID) is not null
                   begin
                       update Discount set Used = 1 where DiscountID =
@DiscountID;
                   end
           end
   end
```

2.AfterDeleteOrderUnuseDiscount

(Po usunięciu zamówienia zwraca klientowi zużytą zniżkę)

```
CREATE TRIGGER TR_AfterDeleteOrderUnuseDiscount on Orders
AFTER DELETE
AS DECLARE
@DiscountID INT
select @DiscountID = d.DiscountID from deleted d
if (select Used from Discount D where D.DiscountID = @DiscountID) = 1
   BEGIN
       update  Discount set Used = 0 where DiscountID = @DiscountID
   end
```

3.AfterDeleteOrderDeleteReservationOrPickup

(Po usunięciu zamówienia usuwa rezerwację lub zamówienie na wynos)

```
CREATE TRIGGER TR_AfterDeleteOrderDeleteReservationOrPickup on Orders
AFTER DELETE
AS DECLARE
@ReservationID INT,
@PickupID INT
select @ReservationID = d.ReservationID from deleted d
select @PickupID = d.PickupID from deleted d
if @ReservationID is not null
  BEGIN
      delete from ReservationDetails where ReservationID = @ReservationID;
      delete from Reservation where ReservationID = @ReservationID;
  End
if @PickupID is not null
   Begin
       delete from Pickup where PickupID = @PickupID;
   end
```

9. Role

1. Pracownik

Pracownik zajmuje się obsługą zamówień online i stacjonarnych. Przyjmuje on rezerwacje, generuje raporty. Ma możliwość dodawania nowych klientów, tworzenia nowego menu

```sql
create role worker

GRANT SELECT ON CurrentMenu TO worker
GRANT SELECT ON MealsCatalog TO worker
GRANT SELECT ON ShowDiscounts TO worker
GRANT SELECT ON IndividualClientInfo TO worker
GRANT SELECT ON UnconfirmedReservation TO worker
GRANT SELECT ON TodayReservations TO worker
GRANT SELECT ON OrdersToPay TO worker
GRANT SELECT ON PendingPickup TO worker
GRANT SELECT ON ProductsSold TO worker
GRANT SELECT ON ProductsSoldDaily TO worker
GRANT SELECT ON ProductsSoldMonthly TO worker
GRANT SELECT ON ProductsSoldAnnually TO worker
GRANT SELECT ON AnnualIncome TO worker
GRANT SELECT ON OrdersInfo TO worker
GRANT SELECT ON DiscountsInfo TO worker
GRANT SELECT ON CurrentVariables TO worker
GRANT SELECT ON WeeklyTableReservations TO worker
GRANT SELECT ON MonthlyTableReservations TO worker

GRANT EXECUTE ON uspAddOrder TO worker
GRANT EXECUTE ON uspAddProductToOrder TO worker
GRANT EXECUTE ON uspAddIndividualClient TO worker
GRANT EXECUTE ON uspAddCompany TO worker
GRANT EXECUTE ON uspAddReservation TO worker
GRANT EXECUTE ON uspAddReservationDetails TO worker
GRANT EXECUTE On uspConfirmReservation TO worker
GRANT EXECUTE ON uspConfirmPickUp TO worker
GRANT EXECUTE ON uspAddDiscountsToClient TO worker
GRANT SELECT ON udfShowClientDiscounts TO worker
GRANT SELECT ON udfGetInvoiceByOrderId TO worker
GRANT EXECUTE ON uspAddMenu TO worker
```

2. Moderator

Moderator rozszerza uprawnienia pracownika. Ma on możliwość dodawania nowych produktów, kategorii, aktualizowania bazy pracowników firmy. Ponadto jest w stanie edytować bazę klientów. Ma możliwość zmieniać stałe na podstawie, których przyznawane są rabaty. Ma możliwość edytowania godzin otwarcia restauracji.

```
CREATE ROLE moderator

GRANT SELECT ON CurrentMenu TO moderator
GRANT SELECT ON MealsCatalog TO moderator
GRANT SELECT ON ShowDiscounts TO moderator
GRANT SELECT ON IndividualClientInfo TO moderator
GRANT SELECT ON UnconfirmedReservation TO moderator
GRANT SELECT ON TodayReservations TO moderator
GRANT SELECT ON OrdersToPay TO moderator
GRANT SELECT ON PendingPickup TO moderator
GRANT SELECT ON ProductsSold TO moderator
GRANT SELECT ON ProductsSoldDaily TO moderator
GRANT SELECT ON ProductsSoldMonthly TO moderator
GRANT SELECT ON ProductsSoldAnnually TO moderator
GRANT SELECT ON AnnualIncome TO moderator
GRANT SELECT ON OrdersInfo TO moderator
GRANT SELECT ON DiscountsInfo TO moderator
GRANT SELECT ON CurrentVariables TO moderator
GRANT SELECT ON WeeklyTableReservations TO moderator
GRANT SELECT ON MonthlyTableReservations TO moderator

GRANT EXECUTE ON uspAddCategory TO moderator
GRANT EXECUTE ON uspAddProduct TO moderator
GRANT EXECUTE ON uspAddProductToMenuByDate TO moderator
GRANT EXECUTE ON uspAddProductToMenuByDate TO moderator
GRANT EXECUTE ON uspAddOrder  TO moderator
GRANT EXECUTE ON uspAddProductToOrder TO moderator
GRANT EXECUTE ON uspAddEmployee TO moderator
GRANT EXECUTE ON uspAddIndividualClient TO moderator
GRANT EXECUTE ON uspAddCompany TO moderator
GRANT EXECUTE ON uspAddTable TO moderator
GRANT EXECUTE ON uspAddReservation TO moderator
GRANT EXECUTE ON uspAddReservationDetails TO moderator
GRANT EXECUTE On uspConfirmReservation TO moderator
GRANT EXECUTE ON uspConfirmPickUp TO moderator
```

```sql
GRANT EXECUTE ON uspAddMenu TO moderator
GRANT EXECUTE ON uspAddDiscountsToClient TO moderator
GRANT SELECT ON udfShowClientDiscounts TO moderator
GRANT SELECT ON udfGetInvoiceByOrderId TO moderator
GRANT EXECUTE ON uspAddVariables To moderator

GRANT SELECT, INSERT, DELETE ON Employees TO moderator
GRANT SELECT, INSERT, DELETE ON Reservation TO moderator
GRANT SELECT, INSERT, DELETE ON ReservationDetails TO moderator
GRANT SELECT, INSERT, DELETE ON [Table] TO moderator
GRANT SELECT, INSERT, DELETE ON Clients TO moderator
GRANT SELECT, INSERT, DELETE ON IndividualClientInfo TO moderator
GRANT SELECT, INSERT, DELETE ON Companies TO moderator
GRANT SELECT, INSERT, DELETE On OpeningHours TO moderator
GRANT SELECT, INSERT, DELETE On OpeningHoursDetails TO moderator
```

3. Admin

Ma możliwość edytowania, dodawania tabel, widoków, funkcji

```
CREATE ROLE admin
GRANT ALL PRIVILEGES ON u_sus.dbo TO admin
```

10. Indeksy

1. Indeks OpeningHours_pk

```sql
CREATE UNIQUE INDEX OpeningHours_pk
ON OpeningHours (OpeningHoursID)`
```

2. Indeks OpeningHoursDetails_pk

```sql
CREATE UNIQUE INDEX OpeningHoursDetails_pk
ON OpeningHoursDetails (OpeningHoursDetailsID)`
```

3. Indeks Companies_pk

```sql
CREATE UNIQUE INDEX Companies_pk
ON Companies (Client_ID)`
```

4. Indeks NIPUnique

```sql
CREATE UNIQUE INDEX NIPUnique
ON Companies (NIP)`
```

5. Indeks Clients_pk

```sql
CREATE UNIQUE INDEX Clients_pk
ON Clients (Client_ID)`
```

6. Indeks IndividualClient_pk

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

7. Indeks UniqueEmail

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

8. Indeks Discount_pk

```
CREATE UNIQUE INDEX Discount_pk
ON Discount (DiscountID)`
```

9. Indeks Orders_pk

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

10. Indeks Reservation_pk

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

11. Indeks ReservationDetails_pk

```
CREATE UNIQUE INDEX ReservationDetails_pk
ON ReservationDetails (ReservationDetailsID)`
```

12. Indeks TableID

```
CREATE UNIQUE INDEX TableID
ON [Table] (TableID)`
```

13. Indeks Employees_pk

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

14. Indeks Pickup_pk

```
CREATE UNIQUE INDEX Pickup_pk
ON Pickup (PickupID)`
```

15. Indeks OrderDetails_pk

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

16. Indeks NameValid

```
CREATE UNIQUE INDEX NameValid
ON Product (ProductName)`
```

17. Indeks Product_pk

```
CREATE UNIQUE INDEX Product_pk
ON Product (ProductID)`
```

18. Indeks Category_pk

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

19. Indeks CategoryName_pk

```
CREATE UNIQUE INDEX CategoryName_pk
ON Category (CategoryName)`
```

20. Indeks MenuDetails_pk

```
CREATE UNIQUE INDEX MenuDetails_pk
ON MenuDetails (ProductID, MenuID)`
```

21. Indeks Menu_pk

```
CREATE UNIQUE INDEX Menu_pk
ON Menu (MenuID)`
```