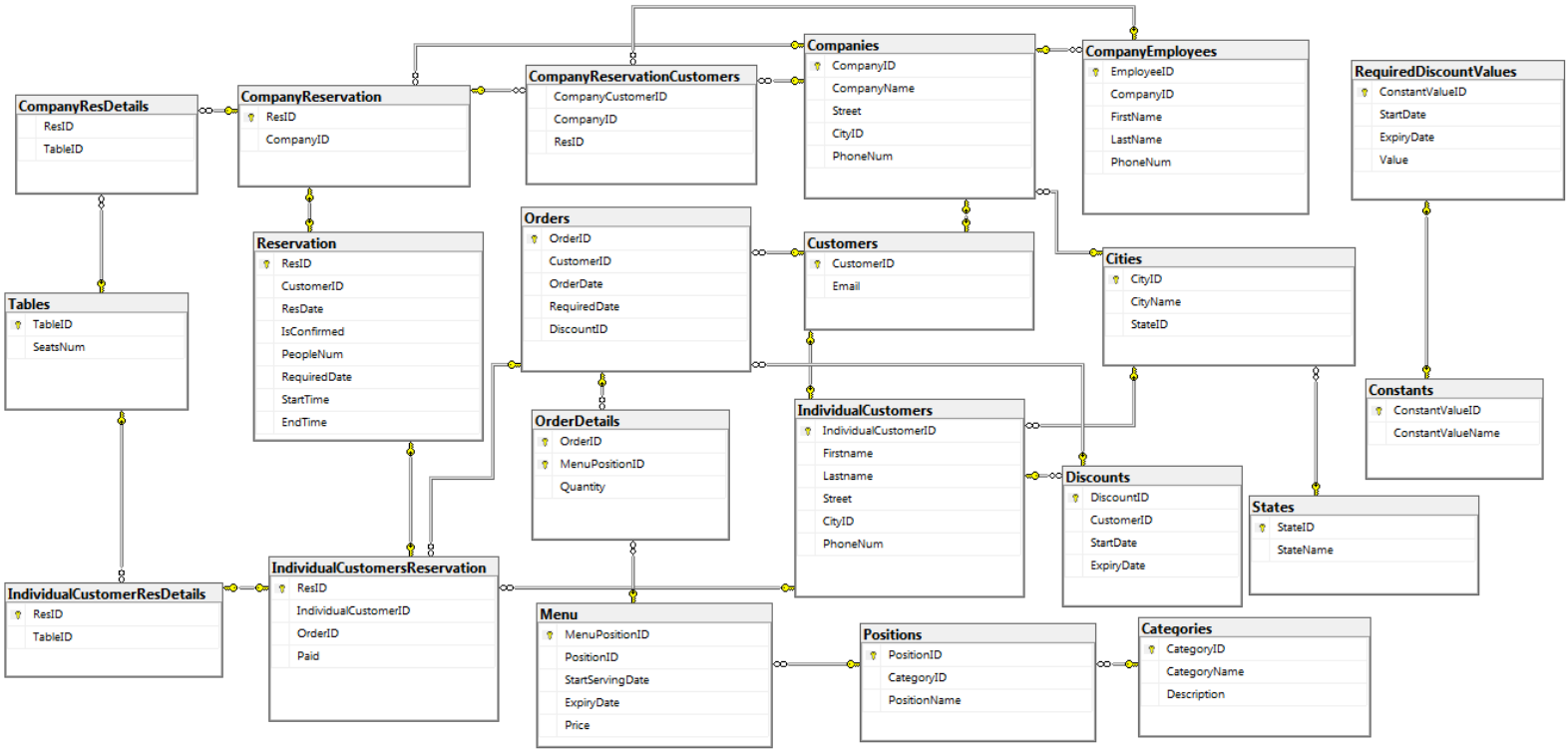
Projekt Restauracja

Podstawy baz danych

Cyprian Neugebauer, Ewelina Flisak, Paweł Konop



Spis treści

FUNKCJE SYSTEMU

1. Klient (Customer)

- składanie zamówienia online na wynos
- rezerwacja stolika przy jednoczesnym złożeniu zamówienia
- możliwość skorzystania z programów rabatowych przy zamówieniu online
- możliwość odwołania rezerwacji stolika i zamówienia, jeśli zostało złożone on-line wraz ze zwrotem kosztów (z wyjątkiem zamówienia owoców morza - brak zwrotu)

2. Kuchnia (KitchenService)

- dostęp do obecnego Menu
- podgląd na dzisiejsze zamówienia
- dodawanie nowych dań do Menu
- dodawanie nowej pozycji do Menu (do tabeli Positions)

3. Administrator (Admin)

- wszystkie funkcjonalności innych użytkowników
- dodawanie i usuwanie klientów i pracowników (osoby związane z bazą)

4. Obsługa restauracji (RestaurantService)

- możliwość wygenerowania faktury dla danego zamówienia lub zbiorczej
- możliwość wprowadzenia do bazy danych zamówienia złożonego stacjonarnie
- możliwość nałożenia dostępnych dla klienta rabatów
- przydział i aktualizacja zajętych stolików
- akceptacja rezerwacji stolika przez formularz online

5. Menadżer restauracji (RestaurantManager)

- dodawanie nowych klientów
- wgląd do statystyk
- sprawdzanie dochodów firmy
- modyfikacja stałych rabatowych (zniżki)
- dodawanie stolików

6. System (System)

- sprawdzanie czy warunek na otrzymanie rabatu został spełniony
- zaimportuj datę rezerwacji stolika
- sprawdzanie czy rezerwacja jest potwierdzona
- aktualizacja menu
- generowanie raportów (miesięczne, tygodniowe, dla klientów indywid. i firm)
- sprawdzanie warunku WK i WZ

TABELE

1. Tabela Positions

Reprezentacja wszystkich dań dostępnych w restauracji, niekonieczne aktualnie dostępnych w menu.

nazwa pola	typ danych	null/not null	przechowywana zawartość pola
PositionID	int	not null	nr ID pozycji
CategoryID	int	not null	nr ID kategorii
PositionName	varchar(50)	not null	nazwa pozycji

Warunki integralności:

A. PositionID - klucz główny do tabeli Positions

```
CONSTRAINT Positions_pk PRIMARY KEY  (PositionID)
```

B. CategoryID - klucz obcy do tabeli Categories

```
FOREIGN KEY (CategoryID) REFERENCES Categories (CategoryID);
```

C. PositionName unikalne

```
create unique index positions_name_index on Positions (PositionName)
```

```
CREATE TABLE Positions (
    PositionID int  NOT NULL,
    CategoryID int  NOT NULL,
    PositionName varchar(50)  NOT NULL,
    CONSTRAINT Positions_pk PRIMARY KEY  (PositionID)
);
create unique index positions_name_index on Positions (Name)
```

2. Tabela Categories

Reprezentacja wszystkich kategorii dań dostępnych w restauracji.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
CategoryID	int	not null	nr ID kategorii
CategoryName	varchar(50)	not null	nazwa kategorii
Description	varchar(50)	not null	opis kategorii

Warunki integralności:

A. CategoryID- klucz główny do tabeli Categories.

```
CONSTRAINT Categories_pk PRIMARY KEY  (CategoryID)
```

B. CategoryName - nazwa unikalna

```
create unique index categories_name_index on Categories (CategoryName)
```

```
CREATE TABLE Categories (
    CategoryID int  NOT NULL,
    CategoryName varchar(50)  NOT NULL,
    Description varchar(50)  NOT NULL,
    CONSTRAINT Categories_pk PRIMARY KEY  (CategoryID)
);
create unique index categories_name_index on Categories (CategoryName)
```

3. Tabela Cities

Reprezentacja miast w bazie danych.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
CityID	int	not null	nr ID miasta
CityName	varchar(50)	not null	nazwa miasta
StateID	int	not null	nr ID stanu

Warunki integralności:

A. CityID - klucz główny do tabeli Cities.

```
CONSTRAINT Cities_pk PRIMARY KEY  (CityID)
```

B. StateID - klucz obcy do tabeli States.

```
FOREIGN KEY (StateID) REFERENCES States (StateID);
```

C. CityName - nazwa miasta jest unikalna.

```
create unique index cities_name_index on Cities (Name)
```

```
CREATE TABLE Cities (
    CityID int  NOT NULL,
    CityName varchar(50)  NOT NULL,
    StateID int  NOT NULL,
    CONSTRAINT Cities_pk PRIMARY KEY  (CityID)
);
create unique index cities_name_index on Cities (Name)
```

4. Tabela Companies

Reprezentacja firm zamawiających jedzenie.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
CompanyID	int	not null	nr ID firmy
CompanyName	varchar(50)	not null	nazwa firmy
Street	varchar(50)	not null	nazwa ulicy
CityID	int	not null	nr ID miasta
PhoneNum	varchar(12)	not null	numer telefonu do firmy

Warunki integralności:

A. CompanyID - klucz główny do tabeli Companies, klucz obcy do tabeli Customers

```
CONSTRAINT Companies_pk PRIMARY KEY  (CompanyID)
```

```
FOREIGN KEY (CompanyID) REFERENCES Customers (CustomerID);
```

B. CityID - klucz obcy do tabeli Cities.

```
FOREIGN KEY (CityID) REFERENCES Cities (CityID);
```

C. PhoneNum - numer telefonu składa się z ‘+’, nr kierunkowego i 9 cyfr.

```
check([PhoneNum] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

```
CREATE TABLE Companies (
    CompanyID int  NOT NULL,
    CompanyName varchar(50)  NOT NULL,
    Street varchar(50)  NOT NULL,
    CityID int  NOT NULL,
    PhoneNum varchar(12)  NOT NULL,
    check([PhoneNum] like '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    CONSTRAINT Companies_pk PRIMARY KEY  (CompanyID));
```

5. Tabela CompanyReservationCustomers

Reprezentacja klientów z firm, którzy złożyli zamówienie.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
CompanyCustomerID	int	not null	nr ID klienta firmowego
CompanyID	int	not null	nazwa firmy
ResID	int	not null	numer rezerwacji

Warunki integralności:

A. CompanyCustomerID - klucz główny do tabeli CompanyCustomers
B. CompanyID - klucz główny do tabeli CompanyCustomers, klucz obcy do tabeli Companies

```
CONSTRAINT CompanyCustomers_pk PRIMARY KEY
(CompanyID,CompanyCustomerID));
```

```
FOREIGN KEY (CompanyID) REFERENCES Companies (CompanyID);
```

C. ResID - klucz obcy do tabeli CompanyReservation

```
FOREIGN KEY (ResID) REFERENCES CompanyReservation (ResID);
```

```
CREATE TABLE CompanyCustomers (
   CompanyCustomerID int  NOT NULL,
   CompanyID int  NOT NULL,
   ResID int  NOT NULL,
                  CONSTRAINT CompanyCustomers_pk PRIMARY KEY
(CompanyID,CompanyCustomerID));
```

6. Tabela CompanyReservation

Reprezentacja rezerwacji złożonych przez firmy, potwierdzonych przez restaurację.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ResID	int	not null	nr ID rezerwacji
CompanyID	int	not null	numer ID firmy

Warunki integralności:

A. ResID - klucz główny do tabeli CompanyReservation, klucz obcy do tabeli Reservation

```
CONSTRAINT CompanyReservation_pk PRIMARY KEY  (ResID));
```

```
FOREIGN KEY (ResID) REFERENCES Reservation (ResID);
```

B. CompanyID - klucz obcy do tabeli Companies

```
FOREIGN KEY (CompanyID) REFERENCES Companies (CompanyID);
```

```
CREATE TABLE CompanyReservation (
    ResID int  NOT NULL,
    CompanyID int  NOT NULL,
    CONSTRAINT CompanyReservation_pk PRIMARY KEY  (ResID));
```

7. Tabela Constants

Tabela zawierająca informacje o stałych dotyczących zniżek oraz ich nazwę.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ConstantValueID	int	not null	nr ID pozycji
ConstantValueName	varchar(50)	not null	nazwa stałej

Warunki integralności:

A. ConstantValueID - klucz główny do tabeli Constants

```
CONSTRAINT Constants_pk PRIMARY KEY  (ConstantValueID));
```

B. ConstantValueName - nazwa stałej unikalna.

```
create unique index constants_name_index on Constants (ConstantValueName)
```

```
CREATE TABLE Constants (
    ConstantValueID int  NOT NULL,
    ConstantValueName varchar(50)  NOT NULL,
    CONSTRAINT Constants_pk PRIMARY KEY  (ConstantValueID));

create unique index constants_name_index on Constants (ConstantValueName)
```

8. Tabela Customers

Tabela z informacjami o klientach

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
CustomerID	int	not null	nr ID klienta
Email	varchar(50)	not null	adres e-mail

Warunki integralności:

A. CustomerID - klucz główny do tabeli Customers

```
CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID)
```

B. Adres e-mail zawiera ‘@’ i jest unikalny dla każdego klienta

```
CHECK ([Email] like '%@%'),
```

```
create unique index customers_name_index
on Customers (Email)
```

```
CREATE TABLE Customers (
    CustomerID int  NOT NULL,
        Email varchar(50) NOT NULL,
        CHECK ([Email] like '%@%'),
    CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID));

create unique index customers_name_index
on Customers (Email)
```

9. Tabela IndividualCustomers

Tabela z informacjami o klientach indywidualnych.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
IndividualCustomerID	int	not null	nr ID indywidualnego klienta
FirstName	varchar(50)	not null	imię indywidualnego klienta
LastName	varchar(50)	not null	nazwisko indywidualnego klienta
Street	varchar(50)	not null	nazwa ulicy
CityID	int	not null	nr ID miasta
PhoneNum	varchar(12)	not null	nr telefonu

Warunki integralności:

A. IndividualCustomerID - klucz główny do tabeli IndividualCustomers, klucz obcy do tabeli Customers

```
CONSTRAINT IndividualCustomers_pk PRIMARY KEY  (IndividualCustomerID)
```

B. CityID - klucz obcy do tabeli Cities

```
FOREIGN KEY (CityID) REFERENCES Cities (CityID);
```

C. PhoneNum - numer telefonu składa się z ‘+’, dwóch cyfr określających nr kierunkowy oraz 9 cyfr

```
CHECK ([PhoneNum] like '+[0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]')
```

```
CREATE TABLE IndividualCustomers (
   IndividualCustomerID int  NOT NULL,
   Firstname varchar(50)  NOT NULL,
   Lastname varchar(50)  NOT NULL,
   Street varchar(50)  NOT NULL,
   CityID int  NOT NULL,
   PhoneNum varchar(9)  NOT NULL,
        CHECK ((ISNUMERIC([PhoneNum]) = (1))),
   CONSTRAINT IndividualCustomers_pk PRIMARY KEY
(IndividualCustomerID));
```

10. Tabela IndividualCustomersReservation

Dotycząca rezerwacji złożonych przez indywidualnego klienta, potwierdzonych przez restaurację.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ResID	int	not null	nr ID rezerwacji
IndividualCustomerID	int	not null	nr ID indywidualnego klienta
OrderID	int	not null	nr ID zamówienia
Paid	bit	not null	czy zamówienie zostało opłacone

Warunki integralności:

A. ResID- klucz główny do tabeli IndividualCustomersReservation, klucz obcy do Reservation

```
CONSTRAINT IndividualCustomersReservation_pk PRIMARY KEY  (ResID)
```

```
FOREIGN KEY (ResID) REFERENCES Reservation (ResID);
```

B. IndividualCustomerID - klucz obcy do tabeli IndividualCustomers

```
FOREIGN KEY (IndividualCustomerID) REFERENCES IndividualCustomers
(IndividualCustomerID);
```

C. OrderID - klucz obcy do tabeli Orders

```
FOREIGN KEY (OrderID) REFERENCES Orders (OrderID);
```

```
CREATE TABLE IndividualCustomersReservation (
    ResID int  NOT NULL,
    IndividualCustomerID int  NOT NULL,
    OrderID int  NOT NULL,
    Paid bit  NOT NULL,
    CONSTRAINT IndividualCustomersReservation_pk PRIMARY KEY  (ResID));
```

11. Tabela Menu

Tabela zawierająca serwowane dania w restauracji.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
MenuPositionID	int	not null	nr ID pozycji w menu
PositionID	int	not null	nr ID pozycji
StartServingDate	date	not null	data pojawienia się pozycji w menu
ExpiryDate	date	null	data zakończenia występowania pozycji w menu
Price	money	not null	cena za porcję

Warunki integralności:

A. MenuPositionID - klucz główny do tabeli Menu

```
CONSTRAINT Menu_pk PRIMARY KEY  (MenuPositionID)
```

B. PositionID - klucz obcy do tabeli Positions

```
FOREIGN KEY (PositionID) REFERENCES Positions (PositionID);
```

C. Data pojawienia się pozycji w menu jest datą wcześniejszą lub równą od daty zakończenia serwowania (StartServingDate <= ExpiryDate)

```
check(StartServingDate <= isnull(ExpiryDate, StartServingDate)),
```

D. Cena za porcję jest nieujemna (Price >= 0)

```
check (Price >= 0)
```

E. StartServingDate domyślnie przyjmuje aktualną datę

```
StartServingDate Date default getdate()  NOT NULL
```

```
CREATE TABLE Menu (
```

```
MenuPositionID int  NOT NULL,
PositionID int  NOT NULL,
StartServingDate Date default getdate()  NOT NULL,
ExpiryDate Date  NULL,
      check(StartServingDate <= isnull(ExpiryDate, StartServingDate)),
Price money  NOT NULL,
      check (Price >= 0),
CONSTRAINT Menu_pk PRIMARY KEY  (MenuPositionID));
```

12. Tabela OrderDetails

Tabela z dodatkowymi informacjami dotyczących zamówienia.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
OrderID	int	not null	nr ID zamówienia
MenuPositionID	int	not null	nr ID pozycji w menu
Quantity	int	not null	liczba zamówionych pozycji

Warunki integralności:

A. OrderID - klucz główny do tabeli OrderDetails, klucz obcy do tabeli Orders

```
CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,MenuPositionID)
```

```
FOREIGN KEY (OrderID) REFERENCES Orders (OrderID);
```

B. MenuPositionID- klucz główny do tabeli OrderDetails, klucz obcy do tabeli Menu

```
CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,MenuPositionID)
```

```
FOREIGN KEY (MenuPositionID) REFERENCES Menu (MenuPositionID);
```

C. Liczba zamówionych pozycji jest większa niż 0 (Quantity > 0)

```
check ([Quantity] > 0)
```

```
CREATE TABLE OrderDetails (
   OrderID int  NOT NULL,
   MenuPositionID int  NOT NULL,
   Quantity int  NOT NULL,
        check ([Quantity] > 0),
   CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,MenuPositionID));
```

13. Tabela Orders

Tabela prezentująca obecne zamówienia.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
OrderID	int	not null	nr ID zamówienia
CustomerID	int	null	nr ID klienta
OrderDate	date	not null	data złożenia zamówienia
RequiredDate	date	not null	data na kiedy ma być zrealizowane zamówienie
DiscountID	int	not null	nr ID zniżki

Warunki integralności:

A. OrderID - klucz główny do tabeli Orders

```
CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
```

B. CustomerID - klucz obcy do tabeli Customers

```
FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

C. Data złożenia zamówienia nie jest wcześniejsza niż data zrealizowania (OrderDate <= RequiredDate)

```
check(OrderDate <= RequiredDate)
```

D. OrderDate domyślnie przyjmuje aktualną datę

```
OrderDate date default getdate() NOT NULL
```

E. DiscountID - klucz obcy do tabeli Discounts

```
FOREIGN KEY (DiscountID) REFERENCES Discounts(DiscountID);
```

```
CREATE TABLE Orders (
    OrderID int  NOT NULL,
    CustomerID int  NULL,
    OrderDate date default getdate() NOT NULL,
    RequiredDate date  NOT NULL,
        check(OrderDate <= RequiredDate ),
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID),
    DiscountID int  NULL,
);
```

14. Tabela RequiredDiscountValues

Tabela-słownik z aktualnie wymaganymi wartościami dotyczących zniżek.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ConstantValueID	int	not null	nr ID stałej
StartDate	date	not null	data przyznania zniżki
ExpiryDate	date	not null	data końca zniżki
Value	int	not null	wartość stałej

Warunki integralności:

A. ConstantValueID - klucz główny, klucz obcy do tabeli Constants

```
CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
```

```
FOREIGN KEY (ConstantValueID) REFERENCES Constants (ConstantValueID);
```

B. data przyznania zniżki mniejsza bądź równa dacie końca zniżki (StartDate <= ExpiryDate)

```
check([StartDate] <= [ExpiryDate])
```

C. Wartość stałej większa od zera (Value > 0)

```
check (Value > 0)
```

D. StartDate domyślnie przyjmuje aktualną datę

```
StartDate date default getdate() NOT NULL
```

```
CREATE TABLE RequiredDiscountValues (
   ConstantValueID int  NOT NULL,
   StartDate date default getdate() NOT NULL,
   ExpiryDate date  NOT NULL,
        check([StartDate] <= [ExpiryDate]),
   Value int  NOT NULL,
        check (Value > 0),
 CONSTRAINT RequiredDiscountValues_pk PRIMARY KEY  (ConstantValueID));
```

15. Tabela Reservation

Tabela zawierająca informacje o rezerwacjach stolików zarówno klientów indywidualnych jak i rezerwacjach od firm (datę dokonania rezerwacji, czy rezerwacja jest potwierdzona, liczba osób na którą dokonano rezerwację).

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ResID	int	not null	nr ID rezerwacji
CustomerID	int	not null	nr ID klienta
ResDate	date	not null	data złożenia rezerwacji
IsConfirmed	bit	not null	czy rezerwacja potwierdzona
PeopleNum	int	not null	liczba osób na którą dokonano rezerwację
RequiredDate	date	not null	data na kiedy ma być zarezerwowany stolik
StartTime	time	not null	godzina, od której następuje przydzielenie stolika
EndTime	time	not null	godzina, do której jest rezerwacja stolika

Warunki integralności:

A. ResID - klucz główny do tabeli Reservation

```
CONSTRAINT Reservation_pk PRIMARY KEY  (ResID)
```

B. Liczba osób na którą dokonano rezerwację większa od zera (PeopleNum > 0)

```
check(PeopleNum > 0)
```

C. Data rezerwacji większa bądź równa aktualnej dacie

```
check(ResDate <= getdate())
```

```
CREATE TABLE Reservation (
   ResID int  NOT NULL,
   CustomerID int  NOT NULL,
   ResDate date  NOT NULL,
   check(ResDate <= getdate()),
   IsConfirmed bit  NOT NULL,
   PeopleNum int  NOT NULL,
        check(PeopleNum > 0),
   CONSTRAINT Reservation_pk PRIMARY KEY  (ResID));
```

16. Tabela Discounts

Tabela, która przechowuje dane jednorazowej zniżki, (przysługujące klientom indywidualnym), tj. datę jej otrzymania oraz datę końca.
(Gdy ExpiryDate jest nullem - zniżka stała, gdy ExpiryDate jest datą - zniżka jednorazowa)

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
DiscountID	int	not null	nr ID zniżki
CustomerID	int	not null	nr ID indywidualnego klienta
StartDate	date	not null	data przyznania zniżki
ExpiryDate	date	null	data końca zniżki

Warunki integralności:

A. DiscountID - klucz główny

```
CONSTRAINT Discount_pk PRIMARY KEY  (DiscountID)
```

B. CustomerID - klucz obcy do tabeli IndividualCustomers

```
FOREIGN KEY (CustomerID) REFERENCES IndividualCustomers
(IndividualCustomerID);
```

C. data przyznania zniżki mniejsza bądź równa daty jej końca (StartDate <= ExpiryDate)

```
check(StartDate <= isnull(ExpiryDate, StartDate))
```

D. StartDate większa bądź równa od aktualnej daty

```
check(StartDate >= getdate())
```

```
CREATE TABLE SingleUseDiscount (
   SingleUseDiscountID int  NOT NULL,
   CustomerID int  NOT NULL,
   StartDate date  NOT NULL,
   check(StartDate >= getdate()),
   ExpiryDate date  NOT NULL,
        check(StartDate <= ExpiryDate),
   CONSTRAINT SingleUseDiscount_pk PRIMARY KEY  (SingleUseDiscountID));
```

17. Tabela States

Tabela słownik stanów.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
StateID	int	not null	nr ID stanu
StateName	varchar(50)	not null	nazwa stanu

Warunki integralności:

A. StateID - klucz główny

```
CONSTRAINT States_pk PRIMARY KEY  (StateID)
```

B. Nazwa stanu unikalna

```
create unique index states_name_index on States (StateName)
```

```
CREATE TABLE States (
    StateID int  NOT NULL,
    StateName varchar(50)  NOT NULL,
    CONSTRAINT States_pk PRIMARY KEY  (StateID)
);

create unique index states_name_index
on States (StateName)
```

18. Tabela Tables

Tabela przechowuje informacje o stolikach, które przysługują klientom składającym rezerwację poprzez formularz WWW.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
TableID	int	not null	nr ID stolika
SeatsNum	int	not null	liczba miejsc przy stoliku

Warunki integralności:

A. TableID - klucz główny do tabeli Tables.

```
CONSTRAINT Tables_pk PRIMARY KEY  (TableID)
```

B. Liczba krzeseł przy stoliku jest dodatnia.

```
check([SeatsNum] > 0),
```

```
CREATE TABLE Tables (
    TableID int  NOT NULL,
    SeatsNum int  NOT NULL,
          check([SeatsNum] > 0),
    CONSTRAINT Tables_pk PRIMARY KEY  (TableID)
);
```

19. CompanyResDetails

Tabela przejścia tabeli Tables oraz CompanyReservation.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ResID	int	not null	nr ID rezerwacji
TableID	int	not null	nr ID stolika

Warunki integralności:

A. ResID - klucz obcy do tabeli CompanyReservation.

```
FOREIGN KEY (ResID) REFERENCES CompanyReservation(ResID);
```

B. TableID - klucz obcy do tabeli Tables.

```
FOREIGN KEY (TableID) REFERENCES Tables(TableID);
```

20. IndividualCustomerResDetails

Tabela przejścia tabeli Tables oraz IndividualCustomersReservation.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
ResID	int	not null	nr ID rezerwacji
TableID	int	not null	nr ID stolika

Warunki integralności:

C. ResID - klucz obcy do tabeli IndividualCustomersReservation.

```
FOREIGN KEY (ResID) REFERENCES IndividualCustomersReservation(ResID);
```

D. TableID - klucz obcy do tabeli Tables.

```
FOREIGN KEY (TableID) REFERENCES Tables(TableID);
```

21. Tabela CompanyEmployees

Tabela z klientami z firm.

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
EmployeeID	int	not null	nr ID pracownika
CompanyID	int	not null	nr ID firmy
FirstName	varchar(50)	not null	imię pracownika
LastName	varchar(50)	not null	nazwisko pracownika
PhoneNum	varchar(9)	not null	numer telefonu

Warunki integralności:

A. EmployeeID - klucz glówny do tabeli CompanyEmployees.

```
CONSTRAINT COmpanyEmployees_pk PRIMARY KEY  (EmployeeID)
```

B. CompanyID - klucz obcy do tabeli Companies.

```
FOREIGN KEY (CompanyID) REFERENCES Companies(ComapnyID);
```

C. PhoneNum - numer telefonu składa się z ‘+’, nr kierunkowego i 9 cyfr.

```
CHECK ([PhoneNum] LIKE '+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

```
CREATE TABLE CompanyEmployees (
    EmployeeID int  NOT NULL,
    CompanyID int  NOT NULL,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    PhoneNum varchar(9)  NOT NULL,
                    CHECK ([PhoneNum] LIKE
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),

                        CONSTRAINT CompanyCustomers_pk PRIMARY KEY
(CompanyID,CompanyCustomerID));
```

22. Tabela DiscountDetails

Tabela z ze szczegółami zniżek klientów indywidualnych

nazwa pola	typ danych	null/ not null	przechowywana zawartość pola
DiscountID	int	not null	nr ID zniżki
ConstantValueID	int	not null	nr ID wartości zniżki
Value	int	not null	wartość zniżki

Warunki integralności:

A. DiscountID - klucz główny do tabeli DiscountDetails

```
CONSTRAINT DiscountDetails_pk PRIMARY KEY  (DiscountID)
```

B. ConstantValueID - klucz obcy do tabeli RequiredDiscountValues

```
FOREIGN KEY (ConstantValueID) REFERENCES
```

```
RequiredDiscountValues(ConstantValueID);
```

WIDOKI

1. Widok CurrentMenu

Wyświetla wszystkie pozycje, które znajdują się aktualnie w menu.

```
create view CurrentMenu as
	select m.MenuPositionID, p.PositionName, price from Menu m
		inner join positions p on p.positionid = m.PositionID
		where StartServingDate <= getdate() and (ExpiryDate is null or
ExpiryDate >= getdate())
```

2. Widok TodaysReservation

Przedstawia listę rezerwacji złożonych na dzień dzisiejszy.

```
CREATE VIEW TodaysReservation AS
	SELECT RES.ResID FROM Reservation RES
	WHERE RES.ResDate = CONVERT(DATE, GETDATE())
```

3. Widok OrdersValue

Przedstawia dla każdego zamówienia jego wartość (z uwzględnieniem rabatu)

```
create view OrdersValue as(
select o.CustomerID, o.orderid, (quantity * price * (1 - CONVERT(float, value)/100)) as
orderValue from orders o
inner join orderdetails od on od.orderid = o.orderid
inner join menu m on m.MenuPositionID = od.MenuPositionID
inner join discounts d on d.DiscountID = o.DiscountID
inner join DiscountDetails dd on dd.DiscountID = d.DiscountID
union
select o.customerid, o.orderid, (quantity * price) as orderVal from orders o
```

```
inner join orderdetails od on od.orderid = o.orderid
inner join menu m on m.MenuPositionID = od.MenuPositionID
where o.DiscountID is null)
```

4. Widok EndsWithGmailCom

Wyświetla wszystkich klientów indywidualnych posługujących się pocztą gmail.

```
CREATE VIEW EndsWithGmailCom AS
	SELECT IC.FirstName + ' ' + IC.LastName as customer FROM Customers C
	INNER JOIN IndividualCustomers IC ON IC.IndividualCustomerID = C.CustomerID
	WHERE C.Email LIKE '%@gmail.com'
	UNION
	SELECT CO.CompanyName FROM Customers C
	INNER JOIN Companies CO ON CO.CompanyID = C.CustomerID
	WHERE C.Email LIKE '@gmail.com'
```

5. Widok CurrentConstants

Wyświetla aktualne stałe wartości rabatowe.

```
create view CurrentConstants as
	select r.ConstantValueID, ConstantValueName, Value from
RequiredDiscountValues r
	inner join Constants c on c.ConstantValueID = r.ConstantValueID
	where StartDate <= getdate() and isnull(ExpiryDate, getdate()) >= getdate()
```

6. Widok CategoryOrders

Wyświetla liczbę produktów zamówionych z danej kategorii

```
create view CategoryOrders as
	select categoryname, sum(Quantity) as categoryOrders from Categories c
	inner join positions p on p.CategoryID = c.CategoryID
	inner join menu m on m.MenuPositionID = p.PositionID
	inner join OrderDetails od on od.MenuPositionID = m.MenuPositionID
	inner join orders o on od.OrderID = o.orderid
	group by c.CategoryID, CategoryName
```

7. Widok NotPaidCompaniesRes

Lista firm, których rezerwacje nie zostały opłacone.

```
create view NotPaidCompaniesRes as
	select r.CustomerID, CompanyName from Reservation r
	inner join CompanyReservation cr on cr.ResID = r.ResID
	inner join Companies c on c.CompanyID = cr.CompanyID
	where IsConfirmed = 0
```

8. Widok IndividualClientsOrders

Zamówienia klientów indywidualnych.

```
create view IndividualClientsOrders as
	select FirstName + ' ' + LastName as IndividualCustomer, o.OrderID
	from IndividualCustomers as ic
	inner join Customers as c on c.CustomerID = ic.IndividualCustomerID
	inner join Orders as o on o.CustomerID = c.CustomerID
```

9. Widok CompanyClientsOrders

Zamówienia klientów firmowych.

```
create view CompanyClientsOrders as
	select CompanyName, o.OrderID
	from Companies as co
	inner join Customers as c on c.CustomerID = co.CompanyID
	inner join Orders as o on o.CustomerID = c.CustomerID
```

10. Widok CategoryAndPositionStatistics

Liczba sprzedanych produktów i ich ilość z podziałem na miesiące i lata

```
create view CategoryAndPositionStatistics AS
select year(o.OrderDate) as year, month(o.OrderDate) as month, (DATEPART(day,
o.OrderDate)-1)/7 + 1 AS week,
c.CategoryName, p.PositionName, sum(od.quantity * m.price * ISNULL(1 -
convert(float, dd.Value) / 100, 1)) AS VALUE, sum(od.Quantity) AS NUMBER from
positions p
inner join Categories c on c.CategoryID = p.CategoryID
inner join Menu m on m.positionID = p.PositionID
inner join OrderDetails od on od.MenuPositionID = m.MenuPositionID
inner join Orders o on o.OrderID = od.OrderID
left outer join Discounts d on d.DiscountID = o.DiscountID
left outer join DiscountDetails dd on dd.DiscountID = d.DiscountID
group by year(o.OrderDate), month(o.OrderDate), (DATEPART(day, o.OrderDate)-1)/7
+ 1, c.CategoryName, p.PositionName
```

11. Widok QuantityOfOrdersYearMonthForEachCustomer

Wyświetla ilość zamówień złożonych przez danego klienta z podziałem na miesiąc i rok

```
create view QuantityOfOrdersYearMonthForEachCustomer as
	select year(orderdate) as year, month(orderdate) as month, c.CustomerID,
count(*) as numOfOrders from orders o
	inner join customers c on c.CustomerID = o.CustomerID
	group by c.CustomerID, year(orderdate), month(orderdate)
```

12. Widok UnconfirmedReservations

Wyświetl niezatwierdzone rezerwacje.

```
create view UnconfirmedReservations as
        select resid from Reservation
        where IsConfirmed = 0
```

13. Widok QuantityOfResForEachTableYearMonth

Wyświetl ilość rezerwacji przy każdym stoliku w każdym miesiącu i roku

```
create view QuantityOfResForEachTableYearMonth as
select year(requireddate) as year, month(requireddate) as month, t.tableid,
count(r.resid) as resNum from tables t
inner join CompanyResDetails crd on crd.tableid = t.TableID
inner join CompanyReservation cr on cr.resid = crd.resid
inner join Reservation r on cr.resid = r.resid
group by t.tableid, r.resid, year(requireddate), month(requireddate)
union
select year(requireddate) as year, month(requireddate) as month, t.tableid,
count(r.resid) as resNum from tables t
inner join IndividualCustomerResDetails icrd on icrd.tableid = t.tableid
inner join IndividualCustomersReservation icr on icr.resid = icrd.resid
inner join reservation r on icr.resid = r.resid
group by t.tableid, r.resid, year(requireddate), month(requireddate)
```

14. Widok RevenueOfEachYearMonthWeek

Wyświetl przychód w każdym roku, miesiącu i tygodniu (z uwzględnieniem zniżek)

```
create view RevenueOfEachYearMonthWeek as
select distinct year(o1.orderdate) as YEAR, month(o1.orderdate) as MONTH,
(DATEPART(day,o1.OrderDate)-1)/7 + 1 AS WEEK, (isnull((select sum(quantity *
price) from orders o inner join orderdetails od on od.orderid = o.orderid

inner join menu m on m.MenuPositionID = od.MenuPositionID

where DiscountID is null and YEAR(o.OrderDate) = year(o1.OrderDate) and
month(o.OrderDate) = MONTH(o1.OrderDate) and
```

```
(DATEPART(day, o1.OrderDate)-1)/7 + 1 = (DATEPART(day,o.OrderDate)-1)/7 +
1), 0)

+
            isnull((select sum(quantity * price * (1 - CONVERT(float, value)/100)) from
orders o inner join orderdetails od on od.orderid = o.orderid

          inner join menu m on m.MenuPositionID = od.MenuPositionID

          inner join Discounts d on o.discountid = d.DiscountID

          inner join discountdetails dd on dd.discountid = d.DiscountID

          where YEAR(o.OrderDate) = year(o1.OrderDate) and month(o.OrderDate)
= MONTH(o1.OrderDate) and

          (DATEPART(day, o1.OrderDate)-1)/7 + 1 = (DATEPART(day,
o.OrderDate)-1)/7 + 1), 0)) AS VALUE from orders o1)
```

15. Widok QuantityOfTablesReservation

Widok, który wyświetla id stolików i sumaryczną ilość rezerwacji każdego z nich

```
create view QuantityOfTablesReservation as
select t1.tableID, year(r1.requireddate) as year, month(r1.requireddate) as month,
(DATEPART(day, r1.RequiredDate)-1)/7 + 1 AS week,
(isnull((select count(r.resid) as resNum from tables t
inner join CompanyResDetails crd on crd.tableid = t.TableID
inner join CompanyReservation cr on cr.resid = crd.resid
inner join Reservation r on cr.resid = r.resid
where year(r1.requireddate) = year(r.requireddate) and month(r1.requireddate) =
month(r.requireddate)
and (DATEPART(day, r1.RequiredDate)-1)/7 + 1 = (DATEPART(day,
r.RequiredDate)-1)/7 + 1 and t.TableID = t1.TableID), 0)
+
isnull((select count(r.resid) as resNum from tables t
inner join IndividualCustomerResDetails icrd on icrd.tableid = t.tableid
inner join IndividualCustomersReservation icr on icr.resid = icrd.resid
inner join reservation r on icr.resid = r.resid
```

```
where year(r1.requireddate) = year(r.requireddate) and month(r1.requireddate) =
month(r.requireddate)
and (DATEPART(day, r1.RequiredDate)-1)/7 + 1 = (DATEPART(day,
r.RequiredDate)-1)/7 + 1 and t.TableID = t1.TableID), 0)) AS number
from reservation r1
left outer join IndividualCustomersReservation icr1 on icr1.ResID = r1.ResID
left outer join IndividualCustomerResDetails icrd1 on icrd1.ResID = icr1.ResID
left outer join Tables t1 on t1.TableID = icrd1.TableID
where r1.IsConfirmed = 1 and t1.TableID is not null
```

16. Widok CostsAndNumberOfDiscounts

Widok, który wyświetla kwotę, która firma straciła przez rabaty oraz ich wykorzystaną ilość z podziałem na lata i miesiące

```
create view CostsAndNumberOfDiscounts as
select year(o.OrderDate) as year, month(o.OrderDate) as month, (DATEPART(day,
o.OrderDate)-1)/7 + 1 AS week,
(sum(od.Quantity * m.Price) - sum(od.Quantity * m.Price * (1 - convert(float, dd.Value)
/ 100))) AS costs, count(o.DiscountID) AS number from Orders o
inner join OrderDetails od on od.OrderID = o.OrderID
inner join Menu m on m.MenuPositionID = od.MenuPositionID
inner join Discounts d on d.DiscountID = o.DiscountID
inner join DiscountDetails dd on dd.DiscountID = d.DiscountID
group by year(o.OrderDate), month(o.OrderDate), (DATEPART(day, o.OrderDate) -
1)/7 + 1
```

17. Widok TodaysOrders

Widok, który pokazuje dzisiejsze zamówienia (MenuPositionID, PositionName, Price)

```
create view TodaysOrders as
select m.MenuPositionID, p.PositionName, Quantity from orders o
inner join orderdetails od on od.orderid = o.orderid
inner join Menu m on m.MenuPositionID = od.MenuPositionID
inner join Positions p on p.PositionID = m.PositionID
where year(RequiredDate) = year(GETDATE()) and month(RequiredDate) =
month(GETDATE()) and day(RequiredDate) = day(GETDATE())
```

18. Widok ShowTablesDateAndTime

Widok, który pokazuje nr POTWIERDZONYCH rezerwacji, nr stolika oraz informacje dotyczące czasu rezerwacji stolików (data + godziny, na które jest zarezerwowany stolik)

```
create view ShowTablesDateAndTime as(
select crd.ResID, t.tableid, PeopleNum, RequiredDate, StartTime, EndTime from
CompanyResDetails crd
inner join Tables t on t.TableID = crd.TableID
inner join CompanyReservation cr on cr.ResID = crd.ResID
inner join Reservation r on r.ResID = cr.ResID
union
select icrd.ResID, t.tableid, PeopleNum, RequiredDate, StartTime, EndTime from
IndividualCustomerResDetails icrd
inner join Tables t on t.TableID = icrd.TableID
inner join IndividualCustomersReservation icr on icr.ResID = icrd.ResID
inner join Reservation r on r.ResID = icr.ResID)
```

PROCEDURY

1. AddCategory

Dodaje nowa kategorię.

```
create procedure AddCategory @Categoryname varchar(50), @Description
varchar(50) as
begin
insert into Categories(Categoryname, Description) values(@Categoryname,
@description)
end
go
```

2. AddPosition

Dodaje nową pozycję w tabeli Positions.

```
create procedure AddPosition @CategoryID int, @PositionName varchar(50) as
```

```
begin
insert into Positions(CategoryID, PositionName) values(@CategoryID,
@PositionName)
end
go
```

3.GetRevenue

Procedura wyświetla zysk z konkretnego roku i miesiąca lub roku, miesiąca i tygodnia.

```
create procedure GetRevenue @Year int, @Month int, @Week int = NULL as
if(@Week is null)
begin
	SELECT SUM(R.VALUE) AS VALUE FROM Revenue R
	WHERE R.YEAR = @Year AND R.MONTH = @Month
end
else
begin
	SELECT R.VALUE FROM Revenue R
	WHERE R.YEAR = @Year AND R.MONTH = @Month AND R.WEEK =
@Week
end
```

4. GetOrderValue

Procedura, która wyświetla wartość zamówienia dla konkretnego orderid (rabat uwzględniony)

```
create procedure GetOrderValue
@orderid int
as
begin
select * from OrdersValue where orderid = @orderid
end
go
```

5. GetCustomerTotalOrderValue

Procedura, która wyświetla sumę wartości wszystkich zamówień konkretnego klienta (rabat uwzględniony)

```
create procedure GetCustomerTotalOrderValue @customerid int
as
begin
select customerid, sum(orderValue) from OrdersValue
where customerid = @customerid
group by customerid
end
go
```

6. GetCustomersOrderValueGreaterThan

Procedura, która wyświetla id klientów oraz sumę wartości ich zamówień, którzy dokonali zamówień większych od wartości ‘value’

```
create procedure GetCustomers @value int
as
begin
select customerid, sum(orderValue) as orderValue from OrdersValue
group by customerid
having sum(ordervalue) > @value
end
go
```

7. GetQuantityOfTablesReservationMonth

Procedura, która wyświetla ilość rezerwacji konkretnego stolika w określonym roku i miesiącu lub roku, miesiącu i tygodniu

```
create procedure GetQuantityOfTablesReservation @TableID int, @Year int, @Month
int, @Week int = NULL as
if(@Week is null)
begin
        select sum(qtr.number) AS resNum from quantityOftablesreservation qtr
```

```
	where qtr.year = @Year and qtr.month = @Month and qtr.tableid = @TableID
end
else
begin
	select sum(qtr.number) as resNum from quantityOftablesreservation qtr
	where qtr.year = @Year and qtr.month = @Month and qtr.week = @Week and
qtr.tableid = @TableID
end
```

8. GetCostsAndNumberOfDiscounts

Procedura, która wyświetla informację ile firma straciła na zniżkach w konkretnym roku, miesiącu lub roku, miesiącu i tygodniu

```
create procedure GetCostsAndNumberOfDiscounts @Year int, @Month int, @Week
int = NULL as
if(@Week is null)
begin
	select sum(cnod.costs) as costs, sum(cnod.number) as number from
CostsAndNumberOfDiscounts cnod
	where cnod.year = @Year and cnod.month = @Month
end
else
begin
	select sum(cnod.costs) as costs, sum(cnod.number) as number from
CostsAndNumberOfDiscounts cnod
	where cnod.year = @Year and cnod.month = @Month and cnod.week =
@Week
end
```

9. GetMenuPositionStatistics

Procedura, która wyświetla nazwy pozycji, wartości za jakie zostały sprzedane (z uwzględnieniem rabatów) oraz ilość w konkretnym roku i miesiącu lub roku, miesiącu i tygodniu

```
create procedure GetMenuPositionStatistics @Year int, @Month int, @Week int =
NULL as
if(@Week is null)
begin
	select PositionName, sum(Value) as VALUE, sum(Number) AS NUMBER from
CategoryAndPositionStatistics
	where year = @Year and month = @Month group by PositionName
end
else
begin
	select PositionName, Value, Number from CategoryAndPositionStatistics
	where year = @Year and month = @Month and week = @Week group by
PositionName, Value, Number
end
```

10. GetCategoryStatistics

Procedura, która wyświetla kategorie, wartość sprzedaży produktów z kategorii (z uwzględnieniem rabatów) oraz ilość sprzedanych produktów z danej kategorii w konkretnym roku i miesiącu lub roku, miesiącu i tygodniu

```
create procedure GetCategoryStatistics @Year int, @Month int, @Week int = NULL
as
if(@Week is null)
begin
	select CategoryName, sum(Value) as VALUE, sum(Number) AS NUMBER
from CategoryAndPositionStatistics
	where year = @Year and month = @Month group by CategoryName
end
else
begin
	select CategoryName, sum(Value) as VALUE, sum(Number) AS NUMBER
from CategoryAndPositionStatistics
	where year = @Year and month = @Month and week = @Week group by
CategoryName
end
```

11. AddTable

Procedura dodająca do tabeli Tables nowy stolik z określoną ilością miejsc.

```
create procedure AddTable @SeatsNum int as
begin
insert into Tables(SeatsNum) values(@SeatsNum)
end
go
```

12. AddState

Procedura dodająca do tabeli States nowy stan

```
create procedure AddState
@Statename varchar(50)
as
begin
insert into States(StateName)
values(@Statename)
end
go
```

13. AddCity

Procedura dodająca do tabeli Cities nowe miasto

```
create procedure AddCity @CityName varchar(50), @StateID int as
begin
insert into Cities(CityName, StateID)
values(@CityName, @StateID)
end
go
```

14. AddReservation

Procedura dodająca do tabeli Reservation nową rezerwację

```
create procedure AddReservation @CustomerID int, @PeopleNum int,
@RequiredDate date, @StartTime time, @Clients varchar(100) = NULL, @Order
varchar(100) = NULL, @Paid bit = NULL, @DiscountID int = NULL as
begin
declare @IDVar table (ID int)
```

```
insert into Reservation(CustomerID, IsConfirmed, PeopleNum, RequiredDate,
StartTime, EndTime) output inserted.ResID into @IDVar(ID) values (@CustomerID, 0,
@PeopleNum, @RequiredDate, @StartTime, DATEADD(HOUR, 2, @StartTime))

if exists (select CompanyID from Companies where CompanyID = @CustomerID)
begin
	insert into CompanyReservation values ((select ID from @IDVar),
@CustomerID)
	if (@Clients is not null)
	begin
		insert into CompanyReservationCustomers select ce.EmployeeID,
ce.CompanyID, (select ID from @IDVar) from string_split(replace(@Clients, ' ', ''), ',') c
		inner join CompanyEmployees ce on ce.FirstName + ce.LastName =
c.value;
	end
end

else
begin
	if (@Paid = 0 AND (select sum(numOfOrders) from
QuantityOfCustomersOrders where CustomerID = @CustomerID) >
		(select rdv.Value from RequiredDiscountValues rdv
		inner join Constants c on c.ConstantValueID = rdv.ConstantValueID
		where GETDATE() BETWEEN rdv.StartDate and rdv.ExpiryDate and
C.ConstantValueName = 'WK') AND (select sum(cm.price * substring(value,
PATINDEX('%[0-9]%', value), LEN(value))) from string_split(replace(@Order, ' ', ''), ',')
as o
		inner join CurrentMenu cm on substring(value, 0, PATINDEX('%[0-9]%',
value)) = replace(cm.PositionName, ' ', '')) > (select rdv.Value from
RequiredDiscountValues rdv
		inner join Constants c on c.ConstantValueID = rdv.ConstantValueID
		where GETDATE() BETWEEN rdv.StartDate and rdv.ExpiryDate and
C.ConstantValueName = 'WZ'))
	begin
		insert into IndividualCustomersReservation values ((select ID from
@IDVar), @CustomerID, NULL, @Paid, @DiscountID)
	end
	else
	begin
		insert into IndividualCustomersReservation values ((select ID from
@IDVar), @CustomerID, NULL, 1, @DiscountID)
	end
	insert into UnconfirmedOrders select (select ID from @IDVar),
m.MenuPositionID, substring(value, PATINDEX('%[0-9]%', value), LEN(value)) from
string_split(replace(@Order, ' ', ''), ',')
```

```
                                                    inner join Positions p on
replace(p.PositionName, ' ', '') = substring(value, 0, PATINDEX('%[0-9]%', value))
                                                    inner join Menu m on
m.PositionID = p.positionID
                                                    where GETDATE() >
M.StartServingDate AND (M.ExpiryDate IS NULL OR GETDATE() <= M.ExpiryDate)

end
end
```

15. AddIndividualCustomer

Procedura, która dodaje nowego klienta indywidualnego

```
create procedure AddIndividualCustomer @FirstName varchar(50), @LastName
varchar(50), @Street varchar(50), @CityName varchar(50), @PhoneNum
varchar(12), @Email varchar(50) as
begin
declare @IDvar table (ID int)
insert into Customers(Email) output inserted.CustomerID into @IDvar(ID) values
(@Email)
insert into IndividualCustomers values ((select ID from @IDvar), @FirstName,
@LastName, @Street, (select CityID from cities where CityName = @CityName),
@PhoneNum)
end
```

16. AddCompany

Procedura, która dodaje do tabeli Companies nową firmę

```
create procedure AddCompany @CompanyName varchar(50), @Street varchar(50),
@CityName varchar(50), @PhoneNum varchar(12), @Email varchar(50) as
begin
declare @IDvar table (ID int)
insert into Customers(Email) output inserted.CustomerID into @IDvar(ID) values
(@Email)
insert into Companies values ((select ID from @IDvar), @CompanyName, @Street,
(select CityID from cities where CityName = @CityName), @PhoneNum)
end
```

17. AddToMenu

Procedura, która dodaje nową pozycję do menu

```
 create procedure AddToMenu @PositionID int, @ExpiryDate date = NULL, @Price
money
 as
 begin
 insert into Menu(PositionID, ExpiryDate, Price)
 values(@PositionID, @ExpiryDate, @Price)
 end
```

18. AddOrder

Procedura, która dodaje nowe zamówienie

```
 create procedure AddOrder @CustomerID int = NULL, @RequiredDate date,
@DiscountID int = NULL, @TakeAway bit = 0, @Order varchar(100) as
begin
declare @IDVar table (ID int)
insert into Orders(CustomerID, RequiredDate, DiscountID, TakeAway) output
inserted.OrderID into @IDVar(ID) values (@CustomerID, @RequiredDate,
@DiscountID, @TakeAway)
insert into OrderDetails select (select ID from @IDVar), m.MenuPositionID,
substring(value, PATINDEX('%[0-9]%', value), LEN(value)) from
string_split(replace(@Order, ' ', ''), ',')
                                                    inner join Positions p on
replace(p.PositionName, ' ', '') = substring(value, 0, PATINDEX('%[0-9]%', value))
                                                    inner join Menu m on
m.PositionID = p.positionID
                                                    where GETDATE() >
M.StartServingDate AND (M.ExpiryDate IS NULL OR GETDATE() <= M.ExpiryDate)

end
```

19. RemoveFromMenu

Procedura, która usuwa z aktualnego menu pozycję

```
create procedure RemoveFromMenu @MenuPositionID int as
begin
update Menu
set ExpiryDate = getdate()
where @MenuPositionID = MenuPositionID
end
```

20. showUnchangedPositions

Procedura, która pokazuje pozycje, które nie zostały zmienione w przeciągu dwóch tygodni

```
create procedure showUnchangedPositions as
begin
	select m.MenuPositionID, p.PositionName, DATEDIFF(DAY,
m.StartServingDate, GETDATE()) AS DAYS from Menu m
	inner join CurrentMenu cm on cm.MenuPositionID = m.MenuPositionID
	inner join positions p on p.PositionID = m.PositionID
	where DATEDIFF(DAY, m.StartServingDate, GETDATE()) > 14
end
```

21. ConfirmReservation

Procedura, która potwierdza rezerwację

```
create procedure ConfirmReservation @ResID int as
begin

if exists (select r.CustomerID from reservation r where r.ResID = @ResID and
r.CustomerID in (select CompanyID from Companies))
begin
	if (select ISNULL(sum(ft.seatsNum), 0) from ShowFreeTables((select
RequiredDate from Reservation where ResID = @ResID), (select StartTime from
Reservation where ResID = @ResID)) as ft)
		>= (select PeopleNum from Reservation where ResID = @ResID)
	begin
		declare @left int = (select PeopleNum from Reservation where ResID =
@ResID)
		declare @tablesReserved table (TableID int)
		declare @insertedTable table (ID int)
		while @left > 0
		begin
```

```
                        if exists (select ft.tableID from ShowFreeTables((select
RequiredDate from Reservation where ResID = @ResID), (select StartTime from
Reservation where ResID = @ResID)) as ft where @left - ft.seatsNum >= 0 and
ft.tableid not in (select ISNULL(TableID, -1) from @tablesReserved))
                        begin
                                insert into CompanyResDetails output inserted.TableID
into @insertedTable values (@ResID, (select top 1 ft.tableID from
ShowFreeTables((select RequiredDate from Reservation where ResID = @ResID),
(select StartTime from Reservation where ResID = @ResID)) as ft where @left -
ft.seatsNum >= 0 and ft.tableid not in (select ISNULL(TableID, -1) from
@tablesReserved) order by ft.seatsNum desc))
                                set @left = @left - (select max(ft.seatsNum) from
ShowFreeTables((select RequiredDate from Reservation where ResID = @ResID),
(select StartTime from Reservation where ResID = @ResID)) as ft where @left -
ft.seatsNum >= 0 and ft.tableid not in (select ISNULL(TableID, -1) from
@tablesReserved))
                                insert into @tablesReserved values ((select * from
@insertedTable))
                        end
                        else
                        begin
                                insert into CompanyResDetails output inserted.TableID
into @insertedTable values (@ResID, (select top 1 ft.tableID from
ShowFreeTables((select RequiredDate from Reservation where ResID = @ResID),
(select StartTime from Reservation where ResID = @ResID)) as ft where @left -
ft.seatsNum < 0 and ft.tableid not in (select ISNULL(TableID, -1) from
@tablesReserved) order by ft.seatsNum))
                                set @left = @left - (select min(ft.seatsNum) from
ShowFreeTables((select RequiredDate from Reservation where ResID = @ResID),
(select StartTime from Reservation where ResID = @ResID)) as ft where @left -
ft.seatsNum < 0 and ft.tableid not in (select ISNULL(TableID, -1) from
@tablesReserved))
                                insert into @tablesReserved values ((select * from
@insertedTable))
                        end
                        delete from @insertedTable
                end
                select TableID from @tablesReserved
                update Reservation set IsConfirmed = 1 where ResID = @ResID
        end
end

else
begin
```

```
	if exists (select * from ShowFreeTables((select RequiredDate from Reservation
where ResID = @ResID), (select StartTime from Reservation where ResID =
@ResID)) as ft
				where ft.seatsNum >= (select PeopleNum from Reservation
where ResID = @ResID))
	begin
		update Reservation set IsConfirmed = 1 where ResID = @ResID
		declare @IDVar table (ID int)
		insert into Orders (CustomerID, OrderDate, RequiredDate, DiscountID)
output inserted.OrderID into @IDVar(ID) values ((select CustomerID from Reservation
where ResID = @ResID), GETDATE(), (select RequiredDate from Reservation where
ResID = @ResID), (select DiscountID from IndividualCustomersReservation where
ResID = @ResID))
		insert into OrderDetails select (select ID from @IDVar), MenuPositionID,
Quantity from UnconfirmedOrders where ResID = @ResID
		delete from UnconfirmedOrders where ResID = @ResID
		update IndividualCustomersReservation set OrderID = (select ID from
@IDVar) where ResID = @ResID
		insert into IndividualCustomerResDetails values (@ResID, (select top 1
ft.tableid from ShowFreeTables((select RequiredDate from Reservation where ResID
= @ResID), (select StartTime from Reservation where ResID = @ResID)) as ft

					where ft.seatsNum >= (select PeopleNum from
Reservation where ResID = @ResID) order by ft.seatsNum))
		select icrd.TableID from Orders o inner join
IndividualCustomersReservation icr on icr.OrderID = o.OrderID
		inner join IndividualCustomerResDetails icrd on icrd.ResID = icr.ResID
where icr.OrderID = (select * from @IDVar)
	end
end

end
```

22. CancelReservation

Procedura, która odwołuje rezerwację

```
create procedure CancelReservation @ResID int as
begin
	if DATEDIFF(DAY, GETDATE(), (select RequiredDate from Reservation where
ResID = @ResID)) >= 1
	begin
		delete from UnconfirmedOrders where ResID = @ResID
```

```
		if exists (select CustomerID From Reservation where ResID = @ResID
and CustomerID not in (select CompanyID from Companies))
		begin
			declare @IDVar table (ID int)
			insert into @IDVar values ((select OrderID from
IndividualCustomersReservation where ResID = @ResID))
			delete from IndividualCustomerResDetails where ResID =
@ResID
			delete from IndividualCustomersReservation where ResID =
@ResID
			if (select * from @IDVar) IS NOT NULL
			begin
				delete from OrderDetails where OrderID = (select * from
@IDVar)
				delete from Orders where OrderID = (select * from
@IDVar)
			end
		end

		delete from CompanyResDetails where ResID = @ResID
		delete from CompanyReservation where ResID = @ResID
		delete from CompanyReservationCustomers where ResID = @ResID
		delete from Reservation where ResID = @ResID
	end
end
```

23. ChangeConstant

Procedura, która zmienia wartość stałych rabatowych (zniżki)

```
create procedure ChangeConstant @ConstantValueName varchar(2), @Value int,
@ExpiryDate date as
begin
	declare @OldConstantValueID int
```

```
	set @OldConstantValueID = (select ConstantValueID from CurrentConstants
where ConstantValueName = @ConstantValueName)
	update RequiredDiscountValues
	set ExpiryDate = getdate() where ConstantValueID = @OldConstantValueID
	insert into Constants(ConstantValueName) values(@ConstantValueName)
	declare @StartDate date
	set @StartDate = getdate()
	insert into RequiredDiscountValues(StartDate, ExpiryDate, Value)
values(@StartDate, @ExpiryDate, @Value)
end
```

24. AddDiscount

Procedura, która dodaje zniżkę dla klienta

```
create procedure AddDiscount @CustomerID int, @StartDate date, @ExpiryDate date
= NULL as
begin
	insert into Discounts(CustomerID, StartDate, ExpiryDate) values
(@CustomerID, @StartDate, @ExpiryDate)
end
```

25. RemoveTable

Procedura, która usuwa stolik

```
create procedure RemoveTable @TableID int as
begin
delete from Tables where TableID = @TableID
end
```

FUNKCJE

1. HasMenuChanged

Funkcja zwracająca wartość typu bit, czy co najmniej połowa pozycji została zmieniona w przeciągu dwóch tygodni

```
create function HasMenuChanged () returns bit as
```

```
begin
	if (select count(*) from Menu m
		inner join CurrentMenu cm on cm.MenuPositionID = m.MenuPositionID
		where DATEDIFF(DAY, m.StartServingDate, GETDATE()) > 14) * 2 >
(select count(*) from CurrentMenu)
	begin
		return 0;
	end
	return 1;
end
```

2. ShowFreeTables

Funkcja, która pokazuje wolne stoliki w konkretnym dniu, o konkretnej godzinie

```
create function ShowFreeTables (@Date date, @StartTime time)
returns table as
return
	select tableID, SeatsNum from Tables where TableID not in
			(select ISNULL(ISNULL(t.TableID, t1.TableID), -1) from
reservation r
			left outer join CompanyReservation cr on r.ResID = cr.ResID
			left outer join CompanyResDetails crd on crd.ResID = cr.ResID
			left outer join IndividualCustomersReservation icr on r.ResID =
icr.ResID
			left outer join IndividualCustomerResDetails icrd on icrd.ResID =
icr.ResID
			left outer join tables t on t.TableID = icrd.TableID
			left outer join tables t1 on t1.TableID = crd.TableID
			where R.RequiredDate = @Date AND (R.StartTime <=
@StartTime AND R.EndTime > @StartTime OR R.StartTime < DATEADD(HOUR, 2,
@StartTime) AND R.EndTime >= DATEADD(HOUR, 2, @StartTime)) AND
R.IsConfirmed = 1)
```

3. GetOrdersValueAfterDate

Funkcja która zwraca wartość zamówień po określonej dacie

```
create function GetOrdersValueAfterDate (@CustomerID int, @Date date) returns
table
as
```

```
return
	select sum(od.Quantity * m.Price * (1 - ISNULL(convert(float, dd.Value)/100,
0))) AS res from Orders o inner join OrderDetails od on od.OrderID = o.OrderID
	inner join menu m on m.MenuPositionID = od.MenuPositionID
	left outer join Discounts d on d.DiscountID = o.DiscountID
	left outer join DiscountDetails dd on dd.DiscountID = d.DiscountID
	where O.CustomerID = @CustomerID AND O.OrderDate >= @Date
```

TRIGGERY

1. SeaFoodCheck

Trigger, który anuluje zamówienie na owoce morza, jeśli zostało złożone w niewłaściwej porze

```
create trigger SeaFoodCheck ON UnconfirmedOrders
after insert as
begin
	if exists (select * from inserted i inner join menu m on m.MenuPositionID =
i.MenuPositionID
		inner join Positions p on p.PositionID = m.PositionID
		inner join Categories c on c.CategoryID = p.CategoryID
		where CategoryName = 'Owoce morza') AND (DATEPART(WEEKDAY,
(select r.RequiredDate from inserted i inner join Reservation r on r.ResID = i.ResID))
NOT BETWEEN 5 AND 7
		OR DATEDIFF(DAY, (select r.ResDate from inserted i inner join
Reservation r on r.ResID = i.ResID), (select r.RequiredDate from inserted i inner join
Reservation r on r.ResID = i.ResID)) < 5
		AND NOT (DATEPART(WEEKDAY, (select r.RequiredDate from inserted
i inner join Reservation r on r.ResID = i.ResID)) >= 5 AND
		(DATEPART(WEEKDAY, (select r.ResDate from inserted i inner join
Reservation r on r.ResID = i.ResID)) = 1 OR
		 DATEPART(WEEKDAY, (select r.RequiredDate from inserted i inner
join Reservation r on r.ResID = i.ResID)) = 2)))
	begin
		declare @IDVar int = (select ResID from inserted)
		exec CancelReservation @ResID = @IDVar
	end
end
```

2. CheckMenuInsert

Trigger, który sprawdza czy nie dodajemy do menu czegoś co już w nim jest (np. żeby uniknąć dwóch takich samych pozycji w aktualnym menu, które różnią się tylko ceną)

```
create trigger CheckMenuInsert ON Menu
after insert as
begin
if exists (select m.PositionID from Menu m where m.StartServingDate <= GETDATE()
AND (m.ExpiryDate IS NULL OR m.ExpiryDate >= GETDATE()) AND m.PositionID =
(select PositionID from inserted))
begin
	ROLLBACK;
	RAISERROR('The dish is already in menu', 17, -1);
end
end
```

3. GrantDiscounts

Trigger, który przyznaje automatycznie zniżki dla klientów

```
create trigger GrantDiscounts ON Orders
after insert as
begin

declare @ID int = (select CustomerID from inserted)
declare @currDate date = GETDATE()
if @ID IS NOT NULL
begin
	if not exists (select * from Discounts where CustomerID = @ID AND
ExpiryDate IS NULL)
	AND (select count(res.OrderID) from (select O.OrderID from Orders o inner join
OrderDetails od on od.OrderID = o.OrderID
		inner join menu m on m.MenuPositionID = od.MenuPositionID
		left outer join Discounts d on d.DiscountID = o.DiscountID
		left outer join DiscountDetails dd on dd.DiscountID = d.DiscountID
		where O.CustomerID = @ID group by o.OrderID having
sum(od.Quantity * m.Price * (1 - ISNULL(convert(float, dd.Value)/100, 0))) >=
		(select Value from Constants c inner join RequiredDiscountValues rdv
on rdv.ConstantValueID = c.ConstantValueID where ConstantValueName = 'K1' AND
GETDATE() BETWEEN StartDate AND ExpiryDate)) res) >=
```

```
                (select Value from Constants c inner join RequiredDiscountValues rdv
on rdv.ConstantValueID = c.ConstantValueID where ConstantValueName = 'Z1' AND
GETDATE() BETWEEN StartDate AND ExpiryDate)
        begin
                exec AddDiscount @CustomerID = @ID, @StartDate = @currDate,
@ExpiryDate = NULL
        end

        declare @tempDate date = (select MAX(StartDate) from Discounts where
CustomerID = @ID AND ExpiryDate IS NOT NULL)
        declare @expiry date = DATEADD(DAY, (select Value from Constants c inner
join RequiredDiscountValues rdv on rdv.ConstantValueID = c.ConstantValueID where
ConstantValueName = 'D1' AND GETDATE() BETWEEN StartDate AND ExpiryDate),
GETDATE())
        if @tempDate IS NOT NULL
        begin
                if (select res from GetOrdersValueAfterDate(@ID, @tempDate)) >=
(select Value from Constants c inner join RequiredDiscountValues rdv on
rdv.ConstantValueID = c.ConstantValueID where ConstantValueName = 'K2' AND
GETDATE() BETWEEN StartDate AND ExpiryDate)
                begin
                        exec AddDiscount @CustomerID = @ID, @StartDate =
@currDate, @ExpiryDate = @expiry
                end
        end
        else
        begin
                if (select res from GetOrdersValueAfterDate(@ID, '1900-01-01')) >=
(select Value from Constants c inner join RequiredDiscountValues rdv on
rdv.ConstantValueID = c.ConstantValueID where ConstantValueName = 'K2' AND
GETDATE() BETWEEN StartDate AND ExpiryDate)
                begin
                        exec AddDiscount @CustomerID = @ID, @StartDate =
@currDate, @ExpiryDate = @expiry
                end
        end
end

end
```

INDEKSY

1. idx_cityName

Indeksuj miasto po nazwie

```
CREATE NONCLUSTERED INDEX [idx_cityName] ON Cities(CityName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

2. idx_stateName

Indeksuj stan po nazwie

```
CREATE NONCLUSTERED INDEX [idx_stateName] ON States(StateName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

3. idx_categoryName

Indeksuj kategorię po nazwie

```
CREATE NONCLUSTERED INDEX [idx_categoryName] ON Categories(CategoryName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

4. idx_positionName

Indeksuj pozycję po nazwie

```
CREATE NONCLUSTERED INDEX [idx_positionName] ON Positions(PositionName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

5. idx_individual_client

Indeksuj klienta indywidualnego po imieniu i nazwisku

```
CREATE NONCLUSTERED INDEX [idx_individual_client] ON
IndividualCustomers(FirstName, LastName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

6. idx_companyEmployee

Indeksuj klienta firmowego po imieniu i nazwisku

```
CREATE NONCLUSTERED INDEX [idx_companyEmployee] ON
CompanyEmployees(FirstName, LastName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

7. idx_companyName

Indeksuj firmę po nazwie

```
CREATE NONCLUSTERED INDEX [idx_companyName] ON Companies(CompanyName)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

8. idx_seatsNum

Indeksuj stolik po liczbie miejsc przy stoliku

```
CREATE NONCLUSTERED INDEX [idx_seatsNum] ON Tables(SeatsNum)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

9. idx_dish_serving_date

Indeksuj danie po dacie rozpoczęcia oraz zakończenia serwowania

```
CREATE NONCLUSTERED INDEX [idx_dish_serving_date] ON Menu(StartServingDate,
ExpiryDate)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

10. idx_validity_date

Indeksuj zniżkę po dacie rozpoczęcia i zakończenia ważności

```
CREATE NONCLUSTERED INDEX [idx_validity_date] ON
RequiredDiscountValues(StartDate, ExpiryDate)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

Uprawnienia

1. Rola IndividualCustomer

Rola przyznawana indywidualnemu klientowi

```
CREATE ROLE IndividualCustomer
```

Uprawnienia:

A. Dostęp do widoku CurrentMenu (aktualne Menu)

```
GRANT SELECT ON CurrentMenu TO IndividualCustomer
```

B. Dostęp do aktualnie panujących wartości rabatowych

```
GRANT SELECT ON CurrentConstants TO IndividualCustomer
```

C. Składanie rezerwacji

```
GRANT EXECUTE ON AddReservation TO IndividualCustomer
```

D. Odwołanie rezerwacji

```
GRANT EXECUTE ON CancelReservation TO IndividualCustomer
```

2. Rola CompanyCustomer

Rola przyznawana klientowi firmowemu

```
CREATE ROLE CompanyCustomer
```

Uprawnienia:

A. Dostęp do widoku CurrentMenu (aktualne Menu)

```
GRANT SELECT ON CurrentMenu TO CompanyCustomer
```

B. Dostęp do aktualnie panujących wartości rabatowych

```
GRANT SELECT ON CurrentConstants TO CompanyCustomer
```

C. Składanie rezerwacji

```
GRANT EXECUTE ON AddReservation TO CompanyCustomer
```

D. Odwołanie rezerwacji

```
GRANT EXECUTE ON CancelReservation TO CompanyCustomer
```

3. Rola KitchenService

Rola przyznawana Personelowi Kuchni

```
CREATE ROLE KitchenService
```

Uprawnienia:

A. Dostęp do widoku CurrentMenu (aktualne Menu)

```
GRANT SELECT ON CurrentMenu TO KitchenService
```

B. Widok na dzisiejsze zamówienia

```
GRANT SELECT ON TodaysOrders TO KitchenService
```

C. Dodawanie nowych dań do Menu

```
GRANT EXECUTE ON AddToMenu TO KitchenService
```

D. Dodawanie nowej Pozycji do Menu

```
GRANT EXECUTE ON AddPosition TO KitchenService
```

E. Podgląd na pozycje, które nie zostały zmienione w przeciągu dwóch tygodni

```
GRANT EXECUTE ON showUnchangedPositions TO KitchenService
```

4. Rola Admin

Rola przyznawana administratorowi bazy

```
CREATE ROLE Admin
```

Uprawnienia:

A. Dodawanie nowej kategorii

```
GRANT EXECUTE ON AddCategory TO Admin
```

B. Dodawanie nowej pozycji

```
GRANT EXECUTE ON AddPosition TO Admin
```

C. Dodawanie nowego miasta

```
GRANT EXECUTE ON AddCity TO Admin
```

D. Dodawanie nowego stanu

```
GRANT EXECUTE ON AddState TO Admin
```

E. Dodawanie stolików

```
GRANT EXECUTE ON AddTable TO Admin
```

F. Sprawdzanie dochodów firmy

```
GRANT EXECUTE ON GetRevenue TO Admin
```

5. Rola RestaurantService

Rola przyznawana obsłudze restauracji (kelnerzy)

```
CREATE ROLE RestaurantService
```

Uprawnienia:

A. Składanie zamówień

```
GRANT EXECUTE ON AddOrder TO RestaurantService
```

B. Akceptacja rezerwacji

```
GRANT EXECUTE ON ConfirmReservation TO RestaurantService
```

C. Dodanie stolika

```
GRANT EXECUTE ON AddTable TO RestaurantService
```

D. Usunięcie stolika

```
GRANT EXECUTE ON RemoveTable TO RestaurantService
```

6. Rola RestaurantManager

Rola przyznawana menedżerowi restauracji

```
CREATE ROLE RestaurantManager
```

Uprawnienia:

A. Dodawanie nowego klienta indywidualnego

```
GRANT EXECUTE ON AddIndividualCustomer TO RestaurantManager
```

B. Dodawanie nowego klienta firmowego

```
GRANT EXECUTE ON AddCompany TO RestaurantManager
```

C. Wgląd do statystyk kategorii z określonego roku i miesiąca

```
GRANT EXECUTE ON GetCategoryStatistics TO RestaurantManager
```

D. Sprawdzanie dochodów firmy

```
GRANT EXECUTE ON GetRevenue TO RestaurantManager
```

E. Dodawanie stolików

```
GRANT EXECUTE ON AddTable TO RestaurantManager
```

F. Modyfikacja stałych rabatowych

```
GRANT EXECUTE ON ChangeConstant TO RestaurantManager
```

G. Wgląd do ilości przyznanych zniżek w danym roku i miesiącu

```
GRANT EXECUTE ON GetCostsAndNumberOfDiscounts TO RestaurantManager
```

H. Sprawdzenie wartości wszystkich zamówień konkretnego klienta

```
GRANT EXECUTE ON GetCustomerTotalOrderValue TO RestaurantManager
```

I. Sprawdzenie wartości konkretnego zamówienia

```
GRANT EXECUTE ON GetOrderValue TO RestaurantManager
```

J. Widok na dzisiejsze rezerwacje

```
GRANT SELECT ON TodaysReservation TO RestaurantManager
```

K. Widok na niepotwierdzone rezerwacje

```
GRANT SELECT ON UnconfirmedReservations TO RestaurantManager
```

L. Usunięcie stolika

```
GRANT EXECUTE ON RemoveTable TO RestaurantManager
```