Dokumentacja Systemu Bazy Danych

Mateusz Cyganek, Kacper Korta

Cel projektu

Celem projektu jest w wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm.

Użytkownicy i ich uprawnienia

Poniższe opisy rozpoczęliśmy od użytkowników z najniższymi uprawnieniami, które dziedziczą i rozbudowują grupy wyższe.

Uprawnienia pracowników:

Pracownik restauracji

- Dodawanie zamówień
- Dodawanie rezerwacji
- Tworzenie kont klienckich
- Podgląd aktualnego Menu

Menadżer restauracji

- Dziedziczy wszystkie permisje pracownika
- Zarządzanie dodawanie i usuwanie pozycji menu, stolików i produktów w odrębnie restauracji, którą zarządza
- Podgląd i modyfikacja danych restauracji

Uprawnienia klientów

Klient (osoba prywatna)

- Utworzenie konta
- Składanie zamówień i rezerwacji (poprzez formularze)

Firma (osoba kontaktowa firmy)

- Utworzenie konta firmowego
- Dodanie i usuwanie pracowników firmy
- Składanie zamówień i rezerwacji (poprzez formularze)

Pracownik firmy (pracownik klienta)

- Składanie zamówień i rezerwacji (poprzez formularze)
- Edycja konta pracownika klienta

Funkcje systemu

Udostępnianie menu

Na żądanie klienta lub jego żądanie przez pracownika.

Każdy klient i pracownik ma dostęp do menu danej restauracji, które jest sumą obecnego rotacyjnego i stałego menu.

Dodawanie zamówienia

Na żądanie klienta lub jego żądanie przez pracownika.

Dodawanie zamówienia odbywa się poprzez wypełnienie formularza. Formularz umożliwia zarówno złożenie zamówienia na miejscu przez pracownika restauracji na życzenie klienta, jak i na wynos, zdalnie przez klienta.

Jeżeli zamówienie zawiera owoce morza, System uruchomi dodatkową walidacje, czy takie zamówienie można złożyć.

Rezerwacja stolika

Na żądanie klienta lub jego żądanie przez pracownika.

Przy zamawianiu będzie opcja zarezerwowania stolika poprzez wypełnienie odpowiedniego formularza.

System uruchomi walidację czy istnieje możliwość zajęcia stolika.

Przedawnienie produktu w menu

Automatyczna funkcja systemu.

System automatycznie znajduje i oznacza produkty, które można dodać do aktualnego menu oraz które muszą zostać z niego usunięta.

Wystawienie faktury

Na żądanie klienta lub jego żądanie przez pracownika.

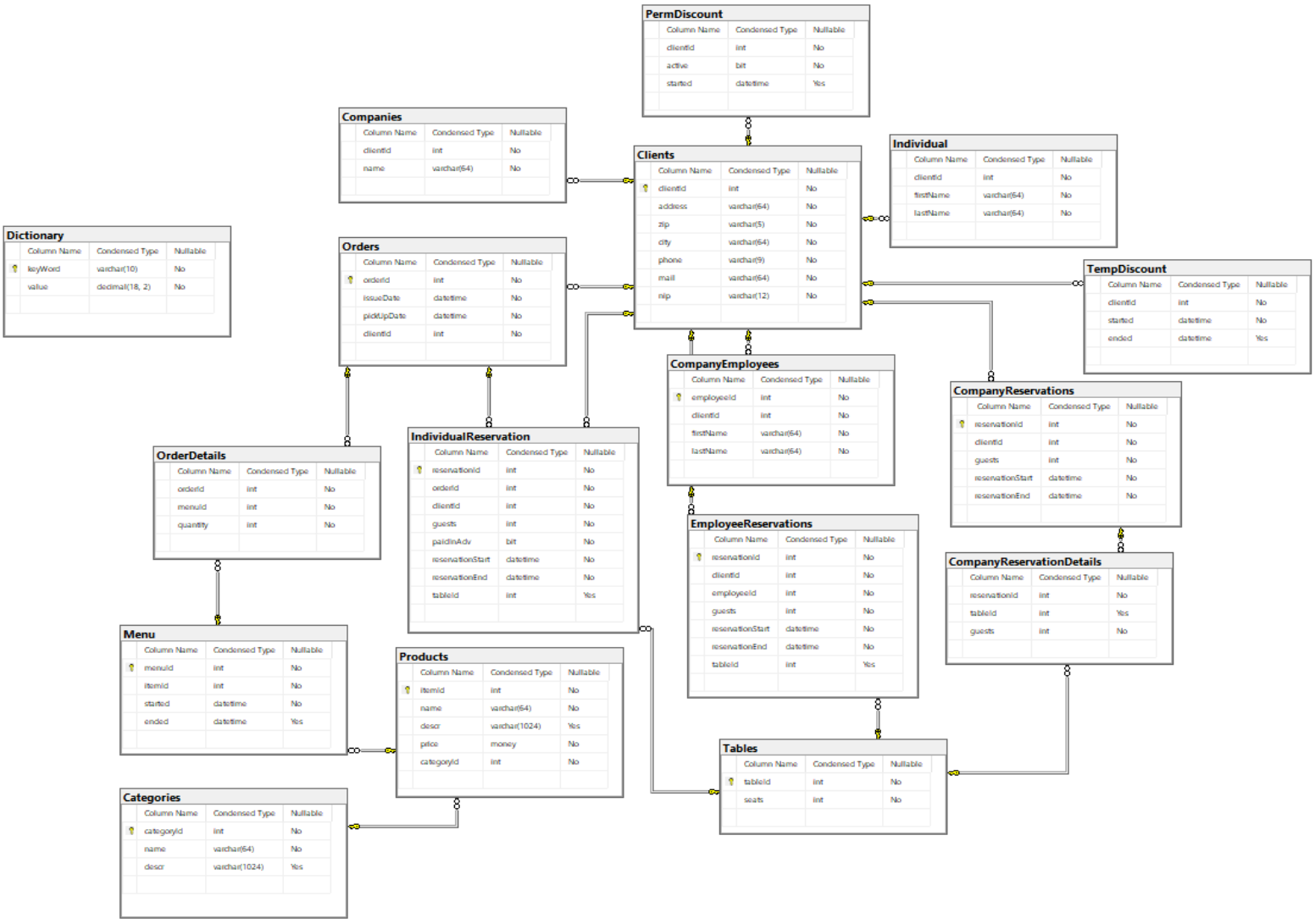
Wystawianie faktury, również tej zbiorczej opiera się na dostarczeniu klientowi informacji o zamówieniu lub zamówieniach jakie wykonał w zażądanym przez siebie czasie

Obsługa zniżek

Automatyczna funkcja systemu.

Przy wykonaniu zamówienia System winien automatycznie zliczać wydatki klienta oraz naliczać stosowne zniżki według polityki restauracji.

Schemat Bazy Danych



Opis i implementacja tabel

Poniżej opisaliśmy wszystkie tabele jakie znajdują się w bazie danych oraz ich implementacje.

Tabela Dictionary

Tabela jest słownikiem zmiennych bazy

1. **keyWord** - Nazwa zmiennej **(Primary key, varchar(10), nie null)**
2. **value** - wartość zmiennej **(decimal(18, 2), nie null)**

Implementacja tabeli.

```
create table Dictionary
(
   keyWord varchar not null
      constraint Dictionary_pk
         primary key nonclustered,
   value   decimal(18, 2) not null
)
go
```

Tabela Categories

Tabela zawiera nazwy kategorii produktów.

1. **categoryID** - ID kategorii **(Primary key, int, nie null)**
2. **name** - nazwa kategorii **(varchar(64), nie null)**
3. **descr** - opis kategorii **(varchar(1024), null)**

Implementacja tabeli.

```
create table Categories
(
	categoryId int identity
		constraint Categories_pk
			primary key nonclustered,
	name varchar(64) not null,
	descr varchar(1024)
)
go
```

Tabela Products

Tabela zawiera dane wszystkich produktów jakie serwuje firma.

1. **itemId** - ID produktu **(Primary key, int, nie null)**
2. **name** - nazwa produktu **(varchar(64), nie null)**
3. **descr** - opis produktu **(varchar(1024), null)**
4. **price** - cena (za jednostkę) **(money, nie null), (check price > 0)**
5. **categoryid** - ID kategorii **(int, nie null, klucz obcy z Categories)**

Implementacja tabeli.

```
use u_mcyganek
go

create table Products
(
   itemId     int identity
      constraint Products_pk
         primary key nonclustered,
   name       varchar(64) not null,
   descr      varchar(1024),
   price      money       not null
      constraint CHK_Products_price
         check ([price] > 0),
   categoryId int         not null
      constraint Products_Categories_categoryId_fk
         references Categories
)
go
```

Tabela Menu

Tabela zawiera historie dostępności produktów.

1. **menuId** - ID pozycji z menu **(Primary Key, int, nie null)**
2. **itemId** - ID produktu **(int, nie null, klucz obcy z Products)**
3. **started** - data dodania produktu do menu **(datetime, nie null, domyślnie getdate())**
4. **ended** - data usunięcia produktu menu **(datetime, null), (check ended > started)**

Implementacja tabeli.

```
create table Menu
(
   menuId  int identity
      constraint Menu_pk
         primary key nonclustered,
   itemId  int      not null
      constraint FK_Menu_Products
         references Products,
   started datetime not null,
   ended   datetime default NULL,
   constraint CHK_Menu_ended
      check ([ended] > [Menu].[started])
)
go

grant alter, delete, select on Menu to []
go
1 2 28 3
```

Tabela Tables

Tabela zawiera dane o stolikach w restauracjach.

1. **tableId** - ID stolika **(Primary Key, int, nie null)**
2. **seats** - maksymalna liczba siedzeń przy stoliku **(int, nie null)**

Implementacja tabeli.

```
create table Tables
(
   tableId int identity
      constraint Tables_pk
         primary key nonclustered,
   seats   int not null
)
go
```

Tabela Orders

Tabela zawiera zamówienia zbiorcze i ich dane.

1. **orderId** - ID zamówienia **(Primary Key, int, nie null)**
2. **issueDate** - data utworzenia zamówienia **(datetime, nie null, domyślnie getdate())**
3. **pickUpDate** - umówiona data odbioru zamówienia **(datetime, nie null, domyślnie getdate()), (check pickupdate >= issuedate)**
4. **clientId** - ID klienta wystawiającego zamówienia **(int, nie null, klucz obcy z Clients)**
5. **isTakeway** – flaga czy na wynos (bit, nie null, default 0)

Implementacja tabeli.

```
use u_mcyganek
go

create table Orders
(
	orderId int identity
		constraint Orders_pk
			primary key nonclustered,
	issueDate datetime default getdate() not null,
	pickUpDate datetime default getdate() not null,
	clientId int not null
		constraint Orders_Clients_clientId_fk
			references Clients,
	isTakeaway bit default 0 not null,
	constraint CHK_Orders_pickUpDate
		check ([pickUpDate]>=[Orders].[issueDate])
)
go
```

Tabela OrderDetails

Tabela zawiera produkty zamówienia zbiorczego

1. **orderId** - ID zamówienia zbiorczego **(int, nie null, klucz obcy z orders, uniklany)**
2. **menuId** - ID pozycji z menu **(int, nie null, klucz obcy z menu, uniklany)**
3. **quantity** - ilość sztuk produktu **(int, nie null), (check quantity > 0)**

Implementacja tabeli.

```
use u_mcyganek
go

create table OrderDetails
(
	orderId int not null
		constraint OrderDetails_Orders_orderId_fk
			references Orders,
	menuId int not null
		constraint OrderDetails_Menu_menuId_fk
			references Menu,
	quantity int not null
		constraint CHK_OrderDetails
			check ([quantity]>0)
)
go
```

Tabela IndividualReservation

Tabela zawiera rezerwacje stolików. Umożliwia nadzorowanie i rezerwowanie stolików

1. **reservationId** – ID rezerwacji **(Primary Key, int, nie null)**
2. **orderId** - ID zamówienia **(int, nie null, klucz obcy z orders)**
3. **clientId** - ID klienta który wykonał rezerwacje **(int, nie null, klucz obcy z clients)**
4. **guests** – ilość osób przy stoliku **(int, nie null), (check guests > 0)**
5. **paidInAdv** - flaga informująca czy zapłacono z góry **(bit, nie null)**
6. **ReservationStart** - data złożenia rezerwacji **(datettime, nie null)**
7. **ReservationEnd** - data na kiedy rezerwacja została złożona **(datettime, nie null), (check reservationend > reservationstart)**
8. **TableId** - nr stolika **(int, null, klucz obcy z Tables)**

Implementacja tabeli.

```
use u_mcyganek
go

create table IndividualReservation
(
   reservationId    int     not null
      constraint IndividualReservation_pk
         primary key nonclustered,
   orderId         int     not null
      constraint IndividualReservation_Orders_orderId_fk
         references Orders,
   clientId        int     not null
      constraint IndividualReservation_Clients_clientId_fk
         references Clients,
   guests          int     not null,
   paidInAdv       bit     not null,
   reservationStart datetime not null,
   reservationEnd   datetime not null,
   tableId         int
      constraint FK_IndividualReservation_Tables
         references Tables,
   constraint CHK_IndividualReservation_guests_dates
      check ([guests] > 0 AND [reservationEnd] > [IndividualReservation].[reservationStart])
)
go
```

Tabela CompanyReservations

Tabela zawiera rezerwacje firm.

1. **reservationId** - ID rezerwacji **(Primary Key, int, nie null)**
2. **clientId** – ID firmy pracownika **(int, nie null, klucz obcy z clients)**
3. **guests** – ilość osób przy stoliku **(int, nie null), (check guests > 0)**
4. **ReservationStart** - data złożenia rezerwacji **(datettime, nie null)**
5. **ReservationEnd** - data na kiedy rezerwacja została złożona **(datettime, nie null),**
6. **(check reservationend > reservationstart)**

Implementacja tabeli.

```
use u_mcyganek
go

create table CompanyReservations
(
   reservationId    int     not null
      constraint PK_CompanyReservations
         primary key,
   clientId        int     not null
      constraint CompanyReservations_Clients_clientId_fk
         references Clients,
   guests          int     not null,
   reservationStart datetime not null,
   reservationEnd   datetime not null,
   constraint CHK_CompanyReservations_guests_dates
      check ([guests] > 0 AND [reservationEnd] > [CompanyReservations].[reservationStart])
)
go
```

Tabela CompanyReservationDetails

Tabela zawiera zarezerwowane stoliki i przypisaną im liczbę osób dla rezerwacji firmowej

1. **reservationId** – ID rezerwacji (int, nie null, klucz obcy z CompanyReservations, uniklany)
2. **tableId** – ID stolika (int, null, klucz obcy z Tables, uniklany)
3. **guests** – liczba osób przy stoliku (int, nie null), **(check guests > 0)**

Implementacja tabeli.

```
use u_mcyganek
go

create table CompanyReservationDetails
(
	reservationId int not null
		constraint FK_CompanyReservationDetails_CompanyReservations
			references CompanyReservations,
	tableId int
		constraint FK_CompanyReservationDetails_Tables
			references Tables,
	guests int not null
		constraint CHK_CompanyReservationDetails_guests
			check ([guests]>0),
	unique (tableId, reservationId)
)
go
```

Tabela EmployeeReservation

Tabela zawiera informacje o osobach na które został zamówiony stolk przez firmę.

1. **reservationId** - ID rezerwacji **(Primary Key, int, nie null)**
2. **clientId** - ID firmy pracownika **(int, nie null, klucz obcy z clients)**
3. **employeeId** - ID pracownika firmy **(int, nie null, klucz obcy z companyEmployees)**
4. **guests** - ilość osób przy stoliku **(int, nie null), (check guests > 0)**
5. **ReservationStart** - data złożenia rezerwacji **(datettime, nie null)**
6. **ReservationEnd** - data na kiedy rezerwacja została złożona **(datettime, nie null), (check reservationend > reservationstart)**
7. **TableId** - nr stolika **(int, null, klucz obcy z tables)**

Implementacja tabeli.

```
use u_mcyganek
go

create table EmployeeReservations
(
   reservationId    int     not null
      constraint EmployeeReservations_pk
         primary key nonclustered,
   clientId        int     not null
      constraint EmployeeReservations_Clients_clientId_fk
         references Clients,
   employeeId      int     not null
      constraint EmployeeReservations_CompanyEmployees_employeeId_fk
         references CompanyEmployees,
   guests          int     not null,
   reservationStart datetime not null,
   reservationEnd   datetime not null,
   tableId         int
      constraint FK_EmployeeReservations_Tables
         references Tables,
   constraint CHK_EmployeeReservations_guests_dates
      check ([guests] > 0 AND [reservationEnd] >
[EmployeeReservations].[reservationStart])
)
go
```

Tabela Clients

Tabela zawiera dane klientów i dane osoby kontaktowej firmy.

1. **clientId** - ID klienta lub firmy **(Primary Key, int, nie null)**
2. **address** - adres klienta lub osoby kontaktowej firmy **(varchar(64), nie** null)
3. **zip** - kod pocztowy klienta lub osoby kontaktowej firmy **(varchar(5), nie null), (check len(zip) = 5 and isnumeric(zip) = 1)**
4. **city** - miasto klienta lub osoby kontaktowej firmy **(varchar(64), nie null)**
5. **phone** - telefon do lub osoby kontaktowej firmy **(varchar(9), nie null) (check len(phone) = 9) and isnumeric(phone) = 1)**
6. **mail** - mail do klienta lub osoby kontaktowej firmy **(varchar(64), nie null)**
7. **nip** - nip klienta lub firmy **(varchar(12), nie null,), check len(nip) = 10**

```
use u_mcyganek
go

create table Clients
(
   clientId int identity
      constraint Clients_pk
         primary key nonclustered,
   address  varchar(64) not null,
   zip      varchar(5)  not null,
   city     varchar(64) not null,
   phone    varchar(9)  not null,
   mail     varchar(64) not null,
   nip      varchar(12) not null,
   constraint CHK_Clients_all
      check (len([zip]) = 5 AND isnumeric([zip]) = 1 AND len([phone]) = 9 AND
isnumeric([phone]) = 1 AND
           [mail] like '%@%.%' AND len([nip]) = 10 AND isnumeric([nip]) = 1)
)
go
```

Implementacja tabeli

Tabela Companies

Tabela zawiera nazwy firm i nip.

1. **clientId** - ID klienta **(int, nie null, klucz obcy z clients)**
2. **name** - nazwa firmy **(varchar(64), nie null)**

Implementacja tabeli.

```
create table Companies
(
		clientId int not null
		  constraint Companies_Clients_clientId_fk
			references Clients,
		name varchar(64) not null
)
go
```

Tabela CompanyEmployees

Tabela zawiera dane pracowników firmy.

1. **employeeId** - ID pracownika firmy **(Primary Key, int, nie null)**
2. **clientId** - ID firmy pracownika **(int, nie null, klucz obcy z clients)**
3. **firstName** - imię pracownika **(varchar(64), nie null)**
4. **lastName** - nazwisko pracownika **(varchar(64), nie null)**

Implementacja Tabeli.

```
create table CompanyEmployees
(
	employeeId int identity
		constraint CompanyEmployees_pk
			primary key nonclustered,
	clientId int not null
		constraint CompanyEmployees_Clients_clientId_fk
			references Clients,
	firstName varchar(64) not null,
	lastName varchar(64) not null
)
go
```

Tabela Individual

Tabela zawiera imię i nazwisko klienta indywidualnego.

1. **cleintId** - ID klienta **(int, nie null, klucz obcy z clients)**
2. **firstName** - imię klienta **(varchar(64), nie null)**
3. **lastName** - nazwisko klienta **(varchar(64), nie null)**

Implementacja tabeli.

```
create table Individual
(
	clientId int not null
		constraint Individual_Clients_clientId_fk
			references Clients,
	firstName varchar(64) not null,
	lastName varchar(64) not null
)
go
```

Tabela PermDiscount

Tabela zawiera informacje o tym czy dany klient ma aktywną permanentną zniżkę

1. **clientId** - ID klienta **(int, nie null, klucz obcy z clients)**
2. **active** - flaga aktywności zniżki **(bit, nie null)**
3. **started –** dzień w którym rozpoczęła się zniżka (datetime, null, domyślnie null)

Implementacja tabeli.

```
use u_mcyganek
go

create table PermDiscount
(
	clientId int not null
		constraint PermDiscount_Clients_clientId_fk
			references Clients,
	active bit default 0 not null,
	started datetime default NULL
)
go
```

Tabela TempDiscount

Tabela zawiera zniżki klientów i licznik kwoty zamówień.

1. **clientId** - ID klienta **(int, nie null, klucz obcy z clients)**
2. **start** - data startu zniżki chwilowej **(datetime, nie null, domyślnie getdate())**
3. **expire** - data końca zniżki chwilowej **(datetime, nie null, domyślnie** dateadd(week, 1, getdate()**)**
4. **orderId –** ID zamówienia, na które zostało zużyte (int, null, domyślnie null)

Implementacja tabeli.

```
create table TempDiscount
(
   clientId int                                not null
      constraint TempDiscount_Clients_clientId_fk
         references Clients,
   started  datetime default getdate()              not null,
   ended    datetime default dateadd(week, 1, getdate()) not null,
   orderId  int      default NULL
)
Go
```

Opis i implementacja widoków

TablesReservedLastWeek

Widok podaje stoliki jakie były rezerwowane w zeszłym tygodniu, kiedy i przez kogo

Implementacja widoku.

```
CREATE VIEW TablesReservedLastWeek
AS
select IR.tableId, IR.reservationId, IR.clientId, IR.reservationStart, IR.reservationEnd
from IndividualReservation IR
where (DATEDIFF(day, IR.reservationEnd, GETDATE()) <= 7)
union
select ER.tableId, ER.reservationId, ER.clientId, ER.reservationStart, ER.reservationEnd
from EmployeeReservations ER
where (DATEDIFF(day, ER.reservationEnd, GETDATE()) <= 7)
union
select CRD.tableId, CR.reservationId, CR.clientId, CR.reservationStart, CR.reservationEnd
from CompanyReservations CR
join CompanyReservationDetails CRD on CR.reservationId = CRD.reservationId
where (DATEDIFF(day, CR.reservationEnd, GETDATE()) <= 7)
go
```

TablesReservedLastMonth

Widok podaje stoliki jakie były rezerwowane w zeszłym miesiącu, kiedy i przez kogo

Implementacja widoku.

```
CREATE VIEW TablesReservedLastMonth
AS
select IR.tableId, IR.reservationId, IR.clientId, IR.reservationStart, IR.reservationEnd
from IndividualReservation IR
where (DATEDIFF(day, IR.reservationEnd, GETDATE()) <= day(eomonth(getdate())))
union
select ER.tableId, ER.reservationId, ER.clientId, ER.reservationStart, ER.reservationEnd
from EmployeeReservations ER
where (DATEDIFF(day, ER.reservationEnd, GETDATE()) <= day(eomonth(getdate())))
union
select CRD.tableId, CR.reservationId, CR.clientId, CR.reservationStart, CR.reservationEnd
from CompanyReservations CR
join CompanyReservationDetails CRD on CR.reservationId = CRD.reservationId
where (DATEDIFF(day, CR.reservationEnd, GETDATE()) <= day(eomonth(getdate())))
go
```

DiscountsGrantedLastWeek

Widok podaje jakie zniżki zostały przyznane w zeszłym tygodniu

Implementacja widoku.

```
CREATE VIEW DiscountsGrantedLastWeek
AS
select clientId, started, ended ended, 'temp' type
from TempDiscount
where started > dateadd(week,(-1),getdate())
union
select clientId, started, null ended, 'perm' type
from PermDiscount
where started > dateadd(week,(-1),getdate())
go
```

DiscountsGrantedLastMonth

Widok podaje jakie zniżki zostały przyznane w zeszłym miesiącu

Implementacja widoku.

```
CREATE VIEW DiscountsGrantedLastMonth
AS
select clientId, started, ended ended, 'temp' type
from TempDiscount
where started > dateadd(month,(-1),getdate())
union
select clientId, started, null ended, 'perm' type
from PermDiscount
where started > dateadd(month,(-1),getdate())
go
```

ActiveDiscounts

Widok przedstawia obecnie aktywne zniżki oraz informacje o nich

Implementacja widoku.

```
CREATE view dbo.ActiveDiscounts as
   select
        clientId,
        dbo.CurrentClientDiscountValue(clientId) value,
        'temp' type,
        (select top 1 TD.started from TempDiscount TD where TD.clientId = C.clientId order
by TD.started desc) started
   from Clients C
   where dbo.CurrentClientDiscountValue(clientId) = 0.05
   union
   select
        clientId,
        dbo.CurrentClientDiscountValue(clientId) value,
        'perm' type,
        (select started from PermDiscount PD where PD.clientId = C.clientId) started
   from Clients C
   where dbo.CurrentClientDiscountValue(clientId) = 0.03
go

grant select on ActiveDiscounts to Manager
go
```

MostPopularProductsLastWeek

Widok przedstawia najczęściej kupowane potrawy w ostatnim tygodniu.

Implementacja widoku.

```
use u_mcyganek
go

create view dbo.MostPopularProductsLastWeek as
	select P.name, sum(OD.quantity) sum from Products P
join Menu M on P.itemId = M.itemId
join OrderDetails OD on M.menuId = OD.menuId
join Orders O on OD.orderId = O.orderId and datediff(day, O.issueDate, getdate()) <= 7
group by P.name
go
```

MostPopularProductsLastMonth

Widok przedstawia najczęściej kupowane potrawy w ostatnim miesiącu.

Implementacja widoku.

```
use u_mcyganek
go

create view dbo.MostPopularProductsLastMonth as
	select P.name, sum(OD.quantity) sum from Products P
join Menu M on P.itemId = M.itemId
join OrderDetails OD on M.menuId = OD.menuId
join Orders O on OD.orderId = O.orderId and datediff(day, O.issueDate, getdate()) <=
day(eomonth(getdate()))
group by P.name
go
```

CurrentMenu

Widok przedstawia produkty obecnie znajdujące się w menu

Implementacja widoku.

```
CREATE view CurrentMenu
as
   select M.menuId, M.itemId, P.name ProductName, P.price, C.name CategoryName, M.started,
M.ended from Menu M
join Products P on M.itemId = P.itemId
join Categories C on P.categoryId = C.categoryId
where (ended is null and getdate() > started ) or (ended is not null and getdate() between started
and ended)
go
```

MonthlyOrders

Widok przedstawia Id klienta i nr zamówienia oraz jego łączną kwotę według roku oraz miesiąca.

Implementacja widoku.

```
CREATE view MonthlyOrders
as
   select year(issueDate) as year, DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)) as
month, clientId, Orders.orderid, sum(quantity * price * (1 -
dbo.CurrentClientDiscountValue(clientId))) as 'Price' from Orders
   inner join OrderDetails OD on Orders.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
   group by year(issueDate), DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)),
clientId, Orders.orderid
go
```

MonthlyOrderDetails

Widok przedstawia szczegóły nr zamówienia oraz całkowity koszt według miesiąca oraz roku.

Implementacja widoku.

```
CREATE view MonthlyOrderDetails
as
   select year(issueDate) as year, DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)) as
month, clients.clientId, o.orderid, p.name as ProductName, c.name as CategoryName,
        quantity, sum(od.quantity * p.price * (1 - dbo.OrderDiscountValue(o.orderId))) AS
TotalPrice from Clients
   inner join Orders O on Clients.clientId = O.clientId
   inner join OrderDetails OD on O.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
   inner join Categories C on C.categoryId = P.categoryId
   group by year(issueDate), DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)),
clients.clientId, o.orderid, p.name, c.name, quantity
go
```

WeeklyOrders

Widok przedstawia Id klienta i nr zamówienia oraz jego łączną kwotę według roku oraz tygodnia.

Implementacja widoku.

```
CREATE view WeeklyOrders
as
      select year(issueDate) as year,  DATENAME(MONTH, DATEADD(MONTH, 0, issueDate))
as month, DATEPART(week, issueDate) as week, clientId, Orders.orderid, sum(quantity *
price * (1 - dbo.CurrentClientDiscountValue(clientId))) as 'Price' from Orders
   inner join OrderDetails OD on Orders.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
   group by year(issueDate), DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)),
DATEPART(week, issueDate), clientId, Orders.orderid
go
```

WeeklyOrderDetails

Widok przedstawia szczegóły nr zamówienia oraz całkowity koszt według tygodnia oraz roku.

Implementacja widoku.

```
CREATE view WeeklyOrderDetails
as
   select year(issueDate) as year,  DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)) as month,  DATEPART(week, issueDate) as week, clients.clientId, o.orderid, p.name as ProductName, c.name as CategoryName,
        quantity, sum(od.quantity * p.price * (1 - dbo.OrderDiscountValue(o.orderId))) AS TotalPrice from Clients
   inner join Orders O on Clients.clientId = O.clientId
   inner join OrderDetails OD on O.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
   inner join Categories C on C.categoryId = P.categoryId
   group by year(issueDate), DATENAME(MONTH, DATEADD(MONTH, 0, issueDate)), DATEPART(week, issueDate), clients.clientId, o.orderid, p.name, c.name, quantity
go
```

WeekIndividualReservations

Widok przedstawia informacje dotyczące rezerwacji klientów indywidualnych którzy je wykonali z ostatniego tygodnia.

Implementacja widoku.

```
CREATE VIEW dbo.WeekIndividualReservations
AS
SELECT       year(reservationStart) as year,  DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month, DATEPART(week, reservationStart) as
week,dbo.IndividualReservation.clientId, i.firstName, i.lastName,
dbo.IndividualReservation.reservationId, dbo.IndividualReservation.guests,
dbo.Tables.tableId,  reservationStart, reservationEnd
FROM          dbo.IndividualReservation INNER JOIN
                   dbo.Tables ON dbo.IndividualReservation.tableId = dbo.Tables.tableId
join Individual i on i.clientId = dbo.IndividualReservation.clientId
go
```

MonthIndividualReservations

Widok przedstawia informacje dotyczące rezerwacji klientów indywidualnych którzy je wykonali ostatniego miesiąca.

Implementacja widoku.

```
CREATE VIEW dbo.MonthIndividualReservations
AS
SELECT       year(reservationstart) as year, DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month,dbo.IndividualReservation.clientId, i.firstName, i.lastName,
dbo.IndividualReservation.reservationId, dbo.IndividualReservation.guests,
dbo.Tables.tableId, reservationStart, reservationEnd
FROM          dbo.IndividualReservation INNER JOIN
                  dbo.Tables ON dbo.IndividualReservation.tableId = dbo.Tables.tableId
join Individual i on i.clientId = dbo.IndividualReservation.clientId
go
```

WeekCompanyReservations

Widok przedstawia informacje dotyczące rezerwacji klientów firmowych którzy je wykonali ostatniego tygodnia. Implementacja widoku.

```
CREATE VIEW dbo.WeekCompanyReservations
AS
SELECT       year(reservationStart) as year,  DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month, DATEPART(week, reservationStart) as
week,dbo.Companies.name, dbo.CompanyReservations.reservationId,
dbo.CompanyReservations.guests, dbo.Tables.tableId
FROM          dbo.CompanyReservations INNER JOIN
                  dbo.Clients ON dbo.CompanyReservations.clientId = dbo.Clients.clientId
INNER JOIN
                  dbo.Companies ON dbo.Clients.clientId = dbo.Companies.clientId INNER JOIN
                  dbo.CompanyReservationDetails ON dbo.CompanyReservations.reservationId
= dbo.CompanyReservationDetails.reservationId INNER JOIN
                  dbo.Tables ON dbo.CompanyReservationDetails.tableId = dbo.Tables.tableId
go
```

MonthCompanyReservations

Widok przedstawia informacje dotyczące rezerwacji klientów firmowych którzy je wykonali ostatniego miesiąca.
Implementacja widoku.

```
CREATE VIEW dbo.MonthCompanyReservations
AS
SELECT      year(reservationstart) as year, DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month, dbo.Companies.name,dbo.companies.clientid,
dbo.CompanyReservations.reservationId, dbo.CompanyReservations.guests,
dbo.Tables.tableId, dbo.CompanyReservations.reservationStart,
dbo.CompanyReservations.reservationEnd
FROM        dbo.Companies INNER JOIN
                 dbo.Clients ON dbo.Companies.clientId = dbo.Clients.clientId INNER JOIN
                 dbo.CompanyReservations ON dbo.Clients.clientId =
dbo.CompanyReservations.clientId INNER JOIN
                 dbo.CompanyReservationDetails ON dbo.CompanyReservations.reservationId
= dbo.CompanyReservationDetails.reservationId INNER JOIN
                 dbo.Tables ON dbo.CompanyReservationDetails.tableId = dbo.Tables.tableId
go
```

WeekEmployeesReservations

Widok przedstawia informacje dotyczące rezerwacji pracowników którzy je wykonali ostatniego tygodnia.

Implementacja widoku.

```
CREATE VIEW dbo.WeekEmployeesReservations
AS
SELECT       year(reservationStart) as year,  DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month, DATEPART(week, reservationStart) as
week,dbo.EmployeeReservations.employeeId, dbo.EmployeeReservations.clientId,
ce.firstName, ce.lastName, reservationId, dbo.EmployeeReservations.guests,
dbo.Tables.tableId, reservationStart, reservationEnd
FROM          dbo.EmployeeReservations INNER JOIN
                  dbo.Tables ON dbo.EmployeeReservations.tableId = dbo.Tables.tableId
inner join CompanyEmployees ce on ce.employeeId = EmployeeReservations.employeeId
go
```

MonthEmployeesReservations

Widok przedstawia informacje dotyczące rezerwacji pracowników którzy je wykonali ostatniego

miesiąca.

Implementacja widoku

```
CREATE VIEW dbo.MonthEmployeeReservation
AS
SELECT       year(reservationstart) as year, DATENAME(MONTH, DATEADD(MONTH, 0,
reservationStart)) as month, dbo.EmployeeReservations.employeeId,
dbo.EmployeeReservations.clientId, ce.firstName, ce.lastName,
dbo.EmployeeReservations.reservationId, dbo.EmployeeReservations.guests,
dbo.Tables.tableId, reservationStart, reservationEnd
FROM          dbo.EmployeeReservations INNER JOIN
                  dbo.Tables ON dbo.EmployeeReservations.tableId = dbo.Tables.tableId
inner join CompanyEmployees ce on ce.employeeId = EmployeeReservations.employeeId
go
```

OldMenuProducts

Widok wyświetla identyfikatory produktów które są w menu dłużej niż 2 tygodnie

Implementacja widoku.

```
CREATE VIEW OldMenuProducts
AS
   select menuId from Menu where ended is null and datediff(day, started, getdate()) > 14
go
```

Opis i implementacja procedur

Procedura MenuAlertJob

Procedura sprawdza czy w menu łamie zasadę, że ponad połowa produktów nie może być starsza niż dwa tygodnie.

Wymaga się od menadżera by codziennie upewniał się, za pomocą tej procedury, że menu jest w poprawnym stanie

Implementacja procedury.

```
create procedure MenuAlertJob
as
begin
   if (
      (select count(menuId) from Menu where ended is null and datediff(day, started,
getdate()) > 14 )
      >
      ((select count(menuId) from Menu where ended is null ) / 2)
   )
   begin
      print ('Ponad połowa pozycji jest w menu od więcej niż dwóch tygodni')
      return
   end
   else
   begin
      print ('OK')
      return
   end
end
go
```

Procedura AddCategory

Umożliwia dodanie kategorii.

Implementacja procedury.

```
create procedure AddCategory
   @name varchar(64)
as
begin
   insert into Categories(name, descr)
   values (@name, null)
end
go
```

Procedura AddProduct

Umożliwia dodanie Produktu.

```
create procedure AddProduct
   @name varchar(64), @price money, @category varchar(64)
as
   declare @categoryId int
   select @categoryId = categoryId from Categories where Categories.name = @category
begin
   insert into Products(name, descr, price, categoryId)
   values (@name, null, @price, @categoryId)
end
go
```

Procedura AddProductToMenu

Umożliwia dodanie produktu do menu.

```
create procedure AddProductToMenu
   @name varchar(64), @started date
as
begin
   declare @itemid int
   select @itemid = itemId from Products
   where lower(@name) = lower(name)
   insert into Menu(ItemId, started)
   values (@itemId, @started)
end
go
```

Procedura SetPermDiscount

Procedura umożliwia uaktywnienie aktywnej zniżki klientowi

Implementacja procedury.

```
create procedure SetPermDiscount
   @clientId int
as
begin
   update PermDiscount
   set active = 1, started = getdate()
   where clientId = @clientId
end
go
```

Procedura AddTempDiscount

Umożliwia dodanie zniżki klientowi. (DODANIE DZIAŁA ALE NIE DA SIĘ USUNĄĆ?)

```
create procedure AddTempDiscount
   @clientId int
as
begin
   insert into TempDiscount(clientId)
   values (@clientId)
end
go
```

Procedura AddOrder

Procedura umożliwia dodanie zamówienia.

Implementacja procedury.

```
CREATE procedure AddOrder
   @pickUpDate datetime, @clientId int, @isTakeaway bit
as
begin
   insert into Orders(issueDate, pickUpDate, clientId, isTakeaway)
   values (getdate(), @pickUpDate, @clientId, @isTakeaway)
end
go
```

Procedura AddOrderDetail

Procedura umożliwia dodanie pozycji zamówienia zbiorczego.

Implementacja procedury.

```
create procedure AddOrderDetail
   @orderId int, @menuId int, @quantity int
as
begin
   insert into OrderDetails(orderId, menuId, quantity)
   values (@orderId, @menuId, @quantity)
end
go
```

Procedura EditAccount

Procedura umożliwia edytowanie danych klienta

Implementacja procedury.

```
create procedure EditAccount
@clientid  int, @address varchar(64), @zip varchar(5), @city varchar(64), @phone
varchar(9), @mail varchar(64), @nip varchar(10), @firstname varchar(64), @lastname
varchar(64)
as
   begin
      if ((select count(*) from CompanyEmployees where clientId = @clientid) > 0)
      begin
         update Clients set address = @address where clientId = @clientid
         update Clients set zip = @zip where clientId = @clientid
         update Clients set city = @city where clientId = @clientid
         update Clients set phone = @phone where clientId = @clientid
         update Clients set mail = @mail where clientId = @clientid
         update Clients set nip = @nip where clientId = @clientid
         update CompanyEmployees set firstName = @firstname where clientId = @clientid
         update CompanyEmployees set lastName = @lastname where clientId = @clientid
      end
      else
      begin
         update Clients set address = @address where clientId = @clientid
         update Clients set zip = @zip where clientId = @clientid
         update Clients set city = @city where clientId = @clientid
         update Clients set phone = @phone where clientId = @clientid
         update Clients set mail = @mail where clientId = @clientid
         update Clients set nip = @nip where clientId = @clientid
         update Individual set firstName = @firstname where clientId = @clientid
         update Individual set lastName = @lastname where clientId = @clientid

      end
   end
go
```

Procedura DeleteAccount

Procedura umożliwia usunięcie konta klienta

Implementacja procedury.

```
create procedure DeleteAccount
@clientId int
as
begin
   delete from Clients where clientId = @clientId
   delete from Individual where clientId = @clientId
   delete from CompanyEmployees where clientId = @clientId
   delete from PermDiscount where clientId = @clientId
   delete from TempDiscount where clientId = @clientId
end
go
```

Procedura AddIndividualReservation

Procedura umożliwia dodanie rezerwacji dla klienta indywidualnego.

Implementacja procedury.

```
create procedure AddIndividualReservation
   @orderId int, @clientId int, @guests int, @paidInAdv bit, @reservationStart datetime,
@reservationEnd datetime
as
   declare @reservationId int
   select top 1 @reservationId = reservationId + 1 from IndividualReservation order by
reservationId desc
begin
   insert into IndividualReservation(reservationId, orderId, clientId, guests, paidInAdv,
reservationStart, reservationEnd, tableId)
   values (@reservationId, @orderId, @clientId, @guests, @paidInAdv, @reservationStart,
@reservationEnd, null)
end
go
```

Procedura SetIndividualReservationTableId

Procedura umożliwia ustawienie stolika rezerwacji klienta indywidulanego.

Implementacja procedury.

```
create procedure SetIndividualReservationTableId
   @reservationId int, @tableId int
   as
begin
   update IndividualReservation set tableId = @tableId where reservationId =
@reservationId
end
go
```

Procedura AddCompanyReservation

Procedura umożliwia dodanie rezerwacji dla firmy.

Implementacja procedury.

```
create procedure AddEmployeeReservation
   @employeeId int, @guests int, @reservationStart datetime, @reservationEnd datetime
as
   declare @reservationId int
   select top 1 @reservationId = reservationId from EmployeeReservations order by
reservationId desc

   declare @clientId int
   select @clientId = clientId from CompanyEmployees where employeeId = @employeeId
begin
   insert into EmployeeReservations(reservationId, clientId, employeeId, guests,
reservationStart, reservationEnd, tableId)
   values (@reservationId, @clientId, @employeeId, @guests, @reservationStart,
@reservationEnd, null)
end
go
```

Procedura AddCompanyReservationDetail

Procedura umożliwia dodanie szczegółów rezerwacji firmy.

Implementacja procedury.

```
create procedure AddCompanyReservationDetail
   @reservationId int, @tableId int, @guests int
as
begin
   insert into CompanyReservationDetails(reservationId, tableId, guests)
   values (@reservationId, @tableId, @guests)
end
go
```

Procedura AddEmployeeReservation

Procedura umożliwia dodanie rezerwacji dla klienta indywidualnego.

Implementacja procedury.

```
create procedure AddEmployeeReservation
   @employeeId int, @guests int, @reservationStart datetime, @reservationEnd datetime
as
   declare @reservationId int
   select @reservationId = count(reservationId) + 1 from EmployeeReservations

   declare @clientId int
   select @clientId = clientId from CompanyEmployees where employeeId = @employeeId
begin
   insert into EmployeeReservations(reservationId, clientId, employeeId, guests,
reservationStart, reservationEnd, tableId)
   values (@reservationId, @clientId, @employeeId, @guests, @reservationStart,
@reservationEnd, null)
end
go
```

Procedura SetEmployeeReservationTable

Procedura umożliwia ustawienie stolika rezerwacji pracownika firmy.

Implementacja procedury.

```
create procedure SetEmployeeReservationTableId
   @reservationId int, @tableId int
   as
begin
   update EmployeeReservations set tableId = @tableId where reservationId =
@reservationId
end
go
```

Procedura AddIndividual

Procedura umożliwia dodanie klienta indywidualnego.

```
CREATE procedure AddIndividual
    @address varchar(64), @zip varchar(5), @city varchar(64), @phone varchar(9), @mail
varchar(64), @nip varchar(64), @firstName varchar(64), @lastName varchar(64)
as
begin
   insert into Clients(address, zip, city, phone, mail, nip)
   values (@address, @zip, @city, @phone, @mail, @nip)
end
   declare @clientId int
   select top 1 @clientId = clientId from Clients order by clientId desc
begin
   insert into Individual(clientId, firstName, lastName)
   values (@clientId, @firstName, @lastName)
end
begin
   insert into PermDiscount(clientid)
   values (@clientId)
end
go
```

Procedura AddCompany

Procedura umożliwia dodanie nowej Firmy.

```
CREATE procedure AddCompany
    @address varchar(64), @zip varchar(5), @city varchar(64), @phone varchar(9), @mail
varchar(64), @nip varchar(64), @name varchar(64)
as
begin
   insert into Clients(address, zip, city, phone, mail, nip)
   values (@address, @zip, @city, @phone, @mail, @nip)
end
   declare @clientId int
   select top 1 @clientId = clientId from Clients order by clientId desc
begin
   insert into Companies(clientId, name)
   values (@clientId, @name)
end
begin
   insert into PermDiscount(clientId)
   values (@clientId)
end
go
```

Procedura AddCompanyEmployee

Procedura umożliwia dodanie pracownika danej firmy.

```
CREATE procedure AddCompanyEmployee
   @companyName varchar(64), @firstName varchar(64), @lastName varchar(64)
as
   declare @clientId int
   select @clientId = clientId from Companies
   where lower(@companyName) = lower(name)
begin
   insert into CompanyEmployees(clientId, firstName, lastName)
   values (@clientId, @firstName, @lastName)
end
go
```

Opis i implementacja funkcji

Funkcja CurrentClientDioscountValue

Funkcja pozwala zobaczyć jaką ma obecnie klient.

Implementacja funkcji

```
CREATE function CurrentClientDiscountValue(@clientId int)
returns decimal(18,2)

begin
   declare @discount decimal(18,2)

   if ((select count(clientId) from TempDiscount where clientId = @clientId and orderId is
null and getdate() between started and dateadd(week, 1, started)) > 0)
      begin
         select @discount = value from Dictionary where keyWord = 'R2'
      end
   else if ((select count(clientId) from PermDiscount where clientId = @clientId and active =
1) > 0)
      begin
         select @discount = value from Dictionary where keyWord = 'R1'
      end
   else
      begin
         select @discount = 0.00
      end

   return @discount
end
go
```

Funkcja OrderDiscountValue

Funkcja pozwala wyliczyć zniżkę dla danego zamówienia

Implementacja funkcji.

```
CREATE function CurrentClientDiscountValue(@clientId int)
returns decimal(18,2)

begin
   declare @discount decimal(18,2)

   if ((select count(clientId) from TempDiscount where clientId = @clientId and orderId is
null and getdate() between started and dateadd(week, 1, started)) > 0)
      begin
         select @discount = value from Dictionary where keyWord = 'R2'
      end
   else if ((select count(clientId) from PermDiscount where clientId = @clientId and active =
1) > 0)
      begin
         select @discount = value from Dictionary where keyWord = 'R1'
      end
   else
      begin
         select @discount = 0.00
      end

   return @discount
end
go
```

Funkcja ClientOrdersLastWeek

Funkcja pozwala zobaczyć jakie zamówienia dokonał dany klient w ostatnim tygodniu.

Implementacja funkcji.

```
CREATE function ClientOrdersLastWeek(@clientId int)
returns table
as
return
   select clients.clientId, O.orderId, o.issueDate, o.pickUpDate,p.name as ProductName,
c.name as Categoryname, od.quantity from Clients
   inner join Orders O on Clients.clientId = O.clientId
   inner join OrderDetails OD on O.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
```

```
    inner join Categories C on C.categoryId = P.categoryId
    where Clients.clientId = @clientId and (DATEDIFF(day, O.issueDate, GETDATE()) <= 7)
go
```

Funckja ClientOrdersLastMonth

Funkcja pozwala zobaczyć jakie zamówienia dokonał dany klient w ostatnim miesiącu.

Implementacja funkcji.

```
create function ClientOrdersLastMonth(@clientId int)
returns table
as
return
   select clients.clientId, O.orderId, o.issueDate, o.pickUpDate,p.name as ProductName,
c.name as Categoryname, od.quantity from Clients
   inner join Orders O on Clients.clientId = O.clientId
   inner join OrderDetails OD on O.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on P.itemId = M.itemId
   inner join Categories C on C.categoryId = P.categoryId
   where Clients.clientId = @clientId and datediff(day, O.issueDate, getdate()) <=
day(eomonth(getdate()))
go
```

Funkcja ClientExpensesLastWeek

Funkcja pozwala zobaczyć ile pieniędzy wydał dany klient w restauracji w przeciągu tygodnia.

Implementacja funkcji.

```
CREATE function ClientExpensesLastWeek(@clientid int)
returns money
begin
   declare @cost money

   select @cost = isnull(sum(dbo.OrderCost(O.orderId)), 0) from Clients
   inner join Orders O on Clients.clientId = O.clientId
   where Clients.clientId = @clientid and (DATEDIFF(day, O.issueDate, GETDATE()) <= 7)
return @cost
end
go
```

Funkcja ClientExpensesLastMonth

Funkcja pozwala zobaczyć ile pieniędzy wydał dany klient w restauracji w przeciągu tygodnia.

Implementacja funkcji.

```
CREATE function ClientExpensesLastMonth(@clientid int)
returns money
begin

   declare @cost money
   select @cost = isnull(sum(dbo.OrderCost(O.orderId)), 0) from Clients
   inner join Orders O on Clients.clientId = O.clientId
   where Clients.clientId = @clientid and datediff(day, O.issueDate, getdate()) <=
day(eomonth(getdate()))
return @cost
end
go
```

Funkcja ClientMonthlyOrders

Funckja po podaniu Id klienta oraz miesiąca pozwala zobaczyć nr zamówienia oraz jego koszt w różnych latach.

Implementacja funkcji.

```
create function ClientMonthlyOrders(@clientid int, @month int)
returns table
as return
   select * from MonthlyOrders where @clientid = clientId and month = (SELECT
DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja ClientMonthlyOrderDetails

Funkcja po podaniu Id klienta oraz miesiąca pozwala zobaczyć szczegółowe informacje zamówień, które w tamtym czasie złożyli.

Implementacja funkcji.

```
create function ClientMonthlyOrderDetails(@clientid int, @month int)
returns table
as return
   select * from MonthlyOrderDetails where @clientid = clientId and month = (SELECT
DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja ClientWeekOrders

Funkcja po podaniu tygodnia oraz id klienta zwraca zamówienia jakich dokonał klient w danym roku.

Implementacja funkcji.

```
create function ClientWeekOrders(@clientid int, @week int)
returns table
as return
   select * from WeeklyOrders where week = @week and clientId = @clientid
go
```

Funkcja ClientWeekOrderDetails

Funkcja po podaniu tygodnia oraz id klienta zwraca szczegóły zamówień jakich dokonał klient w danym roku.

Implementacja funkcji.

```
create function ClientWeekOrderDetails(@clientid int, @week int)
returns table
as return
   select * from weeklyOrderDetails where week = @week and clientId = @clientid
go
```

Funkcja WeekOrders

Funkcja po podaniu tygodnia zwraca zamówienia jakie zostały dokonane w danym okresie.

Implementacja Funkcji.

```
create function WeekOrders(@week int)
returns table
as return
   select * from WeeklyOrders where week = @week
go
```

Funkcja WeekOrderDetails

Funkcja po podaniu nr. Tygodnia zwraca szczegóły zamówień złożonych w tamtym okresie.

Implementacja funkcji

```
create function WeekOrderDetails(@week int)
returns table
as return
   select * from WeeklyOrderDetails where week = @week
go
```

Funckja MonthOrders

Funkcja po podaniu nr. Miesiąca zwraca zamówienia które zostały zamówione danego miesiąca.

Implementacja funkcji.

```
create function MonthOrders(@month int)
returns table
as return
   select * from MonthlyOrders where month = (SELECT DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja MonthOrderDetails

Funckja po podaniu nr. Miesiąca zwraca szczegóły zamówień które zostały zamówione danego miesiąca.

Implementacja funkcji.

```
create function MonthOrderDetails(@month int)
returns table
as return
   select * from MonthlyOrderDetails where month = (SELECT DATENAME( MONTH,
DATEADD( MONTH, @month, -1)))
go
```

Funkcja ClientOrdersForPermDiscount

Funkcja pozwala zobaczyć liczbę zamówień klienta powyżej kwoty kwalifikującej zamówienie do permanentnej zniżki

Implementacja funkcji.

```
CREATE function ClientOrdersForPermDiscount(@clientId int)
returns int
begin
   declare @count int
   select @count = count(OD.orderId) from Orders O
   join OrderDetails OD on O.orderId = OD.orderId
   join Menu M on OD.menuId = M.menuId
   join Products P on M.itemId = P.itemId
   where O.clientId = @clientId and dbo.OrderCost(O.orderId) >= (select value from
Dictionary where keyWord = 'K1')

   return @count
end
go
```

Funkcja AvailabeTables

Funkcja pozwala zobaczyć jakie dostępne stoliki są w danym przedziale czasowym dla danej liczby gosci

```
CREATE function AvailableTables(@guests int, @start datetime, @end datetime)
returns table
as
   return
   select tables.tableid from Tables where tables.tableId not in (select tableId from
CompanyReservationDetails inner join CompanyReservations CR on CR.reservationId =
CompanyReservationDetails.reservationId
      where cr.reservationStart < @end or cr.reservationEnd > @start) and tables.tableId not
in (select tableId from IndividualReservation ir where ir.reservationStart < @end or
ir.reservationEnd > @start)
   and tables.tableId not in (select tableId from EmployeeReservations er where
er.reservationStart < @end or er.reservationEnd > @start)
   and seats >= @guests
go
```

Funkcja ClientSpendingsSinceLastTempDiscount

Funkcja pozwala zobaczyć wydatki klienta jakich dokonał od zakończenia ostatniej zniżki tymczasowej.

Implementacja funkcji.

```
CREATE function ClientSpendingsSinceLastTempDiscount(@clientId int)
returns money
begin
   declare @spendings money

   if (dbo.CurrentClientDiscountValue(@clientId) = (select value from Dictionary where
keyWord = 'R2'))
   begin
      select @spendings = 0.0
   end
   else if ((select count(clientId) from TempDiscount where clientId = @clientId) > 0)
   begin
      declare @startDate datetime
      select top 1 @startDate = ended from TempDiscount where clientId = @clientId order
by ended desc

      select @spendings = isNull(sum(dbo.OrderCost(O.orderId)), 0.00) from Orders O
      where O.issueDate > @startDate and O.clientId = @clientId
   end
   else
   begin
      select @spendings = isNull(sum(dbo.OrderCost(O.orderId)), 0.00) from Orders O
      where O.clientId = @clientId
   end
   return @spendings
end
go
```

Funkcja OrderCost

Funkcja pozwala zobaczyć koszt danego nr. Zamówienia z odliczoną obniżką jeżeli klient takową posiada.

```
CREATE function OrderCost(@orderId INT)
returns money
begin
   declare @cost money
   declare @clientId int
   select @clientId = clientId from Orders
   where orderId = @orderId

   select @cost = sum(quantity * p.price * (1 - dbo.OrderDiscountValue(O.orderId)))
   from Orders O
   inner join OrderDetails OD on O.orderId = OD.orderId
   inner join Menu M on M.menuId = OD.menuId
   inner join Products P on M.itemId = P.itemId
   where OD.orderId = @orderId
   return @cost
end
go
```

Funkcja EmployeeWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji klienta firmowego według nr. Tygodnia.

Implementacja funkcji.

```
create function EmployeeWeekReservation(@week int)
   returns table as return
      select * from WeekEmployeesReservations WHERE week = @week
go
```

Funkcja IndividualWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji klienta indywidualnego według nr. Tygodnia.

Implementacja funkcji.

```
create function IndividualWeekReservation(@week int)
   returns table as return
      select * from weekIndividualReservations WHERE week = @week
go
```

Funkcja CompanyWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji frimowej według nr. Tygodnia.

Implementacja funkcji.

```
CREATE function CompanyWeekReservation(@week int)
   returns table as return
      select * from weekCompanyReservations WHERE week = @week
go
```

Funkcja EmployeeClientWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danego klienta firmowego według nr. Tygodnia.

Implementacja funkcji.

```
create function EmployeeClientWeekReservation(@employeeid int, @week int)
   returns table as return
      select * from WeekEmployeesReservations WHERE week = @week and employeeId =
@employeeid
go
```

Funkcja IndividualClientWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danego klienta indywidualnego według nr. Tygodnia.

Implementacja funkcji.

```
create function IndividualClientWeekReservation(@clientid int, @week int)
   returns table as return
      select * from WeekIndividualReservations WHERE week = @week and clientId =
@clientid
go
```

Funkcja CompanyClientWeekReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danej frimy według nr. Tygodnia.

Implementacja funkcji.

```
create function CompanyClientWeekReservation(@clientid int, @week int)
   returns table as return
      select * from WeekCompanyReservations WHERE week = @week and clientId =
@clientid
go
```

Funkcja EmployeeClientMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danego klienta firmowego według miesiąca.

Implementacja funkcji.

```
create function EmployeeClientMonthReservation(@employeeid int, @month int)
returns table as return
   select * from MonthEmployeeReservation where employeeId = @employeeid and month
= (SELECT DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja IndividualClientMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danego klienta indywidualnego według miesiąca.

Implementacja funkcji.

```
create function IndividualClientMonthReservation(@clientid int, @month int)
returns table as return
   select * from MonthIndividualReservations where clientId = @clientid and month =
(SELECT DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja CompanyClientMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji danej firmy według miesiąca.

Implementacja funkcji.

```
create function CompanyClientMonthReservation(@clientid int, @month int)
returns table as return
   select * from MonthCompanyReservations where clientId = @clientid and month =
(SELECT DATENAME( MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja EmployeeMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji klienta firmowego według miesiąca.

Implementacja funkcji.

```
create function EmploeeMonthReservation(@month int)
   returns table as return
      select * from MonthEmployeeReservation WHERE month = (SELECT DATENAME(
MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja IndividualMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji klienta indywidualnego według miesiąca.

Implementacja funkcji.

```
create function IndividualMonthReservation(@month int)
   returns table as return
      select * from MonthIndividualReservations WHERE month = (SELECT DATENAME(
MONTH, DATEADD( MONTH, @month, -1)))
go
```

Funkcja CompanyMonthReservation

Funkcja pozwala zobaczyć szczegóły rezerwacji frimowej według miesiąca.

Implementacja funkcji.

```
create function CompanyMonthReservation(@month int)
   returns table as return
      select * from MonthCompanyReservations WHERE month = (SELECT DATENAME(
MONTH, DATEADD( MONTH, @month, -1)))
go
```

Opis i implementacja triggerów

Trigger DiscountAfterOrder

Trigger dodaj zniżki klientowi po spełnieniu przez niego warunków po zamówieniu

Implementacja triggera.

```
create trigger DiscountAfterOrder
   on dbo.OrderDetails
   after insert
as
   begin
      declare @clinetId int
      select @clinetId = O.clientId from Orders O where O.orderId = (select i.orderId from
inserted i)

      if (dbo.ClientOrdersForPermDiscount (@clinetId) >= (select value from Dictionary
where keyWord = 'Z1'))
      begin
         exec dbo.SetPermDiscount @clinetId
      end
      if (dbo.ClientSpendingsSinceLastTempDiscount (@clinetId) >= (select value from
Dictionary where keyWord = 'K2'))
      begin
         exec dbo.AddTempDiscount @clinetId
      end
   end
go
```

Trigger Fish

Trigger sprawdza przed dodaniem zamowienia jezeli jest to produkt rybny to czy został zamówiony w piątek sobotę lub niedzielę oraz czy zamowienie ma zsotac zrealizowane przed poniedzialkiem.

```
CREATE trigger Fish
   on dbo.OrderDetails
   after insert
   as
   begin
   declare @menuid int
   select @menuid = i.menuId from inserted i
   declare @categoryid int
   select @categoryid = categoryId from Categories where name = 'rybne'
   declare @orderid int
   select @orderid = o.orderid from inserted o


   if @categoryid = (select categoryid from products inner join Menu m on Products.itemId
= m.itemId where m.menuId = @menuid)
      begin
        if ((SELECT DATEPART(DW, issueDate) from OrderDetails inner join Orders O on
O.orderId = OrderDetails.orderId where OrderDetails.orderId = @orderid ) not in (5, 6, 7)) -
-and
          --(select count(*) from OrderDetails inner join Menu M on M.menuId =
OrderDetails.menuId inner join Products P on P.itemId = M.itemId and categoryId =
@categoryid where @menuid = OrderDetails.menuId and orderId = @orderid) > 0
        begin
          raiserror ('Produkt rybny musi być zamówiony w czwartek, piątek lub sobotę', 1, 1)
          rollback transaction
          return
        end
        else if ((SELECT DATEPART(DW, pickUpDate) from OrderDetails inner join Orders O
on O.orderId = OrderDetails.orderId where OrderDetails.orderId = @orderid) in (2, 3, 4) or
(select datediff(day , issueDate, pickUpDate) from OrderDetails inner join Orders O on
O.orderId = OrderDetails.orderId where OrderDetails.orderId = @orderid) > 4)
        begin
          raiserror ('Produkt rbyny musi być odberany przed poniedziałkiem', 1, 1)
          rollback transaction
          return
        end
      end
   end
go
```

Trigger CheckIfInMenu

Trigger sprawdza przed dodaniem zamowienia czy produkt obcenie znajduje się w menu

```
CREATE trigger CheckIfInMenu
   on dbo.orderdetails
   instead of insert

   as
   begin
      declare @menuid int
      select @menuid = I.menuid from inserted I

      if ((select count(*) from Menu M where M.menuId = @menuid and getdate() >
M.started and M.ended is null) = 0)
      begin
         raiserror ('Produkt jest obecnie niedostępny w menu', 1, 1)
         rollback transaction
      end
      else
      begin
         insert into OrderDetails(orderId, menuId, quantity)
         values ((select i.orderid from inserted i), (select i.menuid from inserted i), (select
i.quantity from inserted i))
      end
   end
go
```

Trigger MenuItemAddedInAdvance

Trigger sprawdza czy pozycje menu dodano z odpowiednim wyprzedzeniem o czym informuje użytkownika w przypadku niepowodzenia

Implementacja triggera.

```
create trigger MenuItemAddedInAdvance
   on dbo.Menu
   instead of insert
as
   begin
      if (datediff(day, (select I.started from inserted I), getdate()) < 1)
      begin
         raiserror ('Pozycja menu musi zostać dodana na conajmniej dzień wcześniej', 1, 1)
         rollback transaction
         return
      end
      else
      begin
         insert into Menu (itemId, started)
         values ((select I.itemId from inserted I), (select I.started from inserted I))
      end
   end
go
```

Trigger IsClientIsAllowedToMakeReservation

Trigger sprawdza czy klient indywidualny może dokonać rezerwacji oraz czy podał poprawne godziny rozpoczęcia i zakończenia rezerwacji.

Implementacja triggera.

```
CREATE trigger IsClientIsAllowedToMakeReservation
   on dbo.IndividualReservation
   instead of insert
as
   begin
      declare @clientId int
      select @clientId =  I.clientId from inserted I
      declare @orderId int
      select @orderId =  I.orderId from inserted I
      declare @started datetime
      select @started = I.reservationStart from inserted I
      declare @ended datetime
      select @ended = I.reservationEnd from inserted I
      declare @pickup datetime
      select @pickup = pickUpDate from Orders where orderId = @orderId


      if (dbo.OrderCost(@orderId) < (select value from Dictionary where keyWord = 'WZ'))
      begin
         raiserror ('Nie można wstawić rezerwacji! Zbyt nisko kosz zamówienia', 1, 1)
         return
      end
      else if ((select count(orderId) from Orders where clientId = @clientId) < (select value
from Dictionary where keyWord = 'WK'))
      begin
         raiserror ('Nie można wstawić rezerwacji! Posiadasz zbyt mało zamówień', 1, 1)
         rollback transaction
         return
      end
      else if (datediff(hour, @pickup, @started) > 2 or datediff(hour, @started, @ended) > 8)
      begin
         raiserror ('Niepoprawna data, godzina lub czas trwania rezerwacji', 1, 1)
         rollback transaction
         return
      end
      else
      begin
         insert into IndividualReservation (reservationId, orderId, clientId, guests, paidInAdv,
reservationStart, reservationEnd, tableId)
         values ((select I.reservationId from inserted I), (select I.orderId from inserted I),
(select I.clientId from inserted I), (select I.guests from inserted I), (select I.paidInAdv from
```

```
inserted I), (select I.reservationStart from inserted I), (select I.reservationEnd from
inserted I), (select I.tableId from inserted I))
      end
   end
go
```

Trigger Checkif2

Trigger sprawdzający przed dodaniem rezerwacji indywidualnej czy jest na co najmniej 2 gości.

Implementacja Triggera

```
CREATE trigger CheckIf2 on dbo.IndividualReservation
   Instead of insert
   as
   declare @orderid int
   select @orderid = i.orderid from inserted i
   declare @guests int
   select @guests = O.guests from inserted o
   if @guests < 2
      begin
         raiserror ('Rezerwacja musi być na co najmniej 2 osoby', 1, 1)
         rollback
      end
   else
   begin
      insert into IndividualReservation(reservationId, orderId, clientId, guests, paidInAdv,
reservationStart, reservationEnd, tableId)
      values ((select I.reservationId from inserted I), (select I.orderId from inserted I), (select
I.clientId from inserted I), (select I.guests from inserted I), (select I.paidInAdv from inserted
I), (select I.reservationStart from inserted I), (select I.reservationEnd from inserted I),
(select I.tableId from inserted I))
   end
go
```

Trigger Checkif2e

Trigger sprawdzający przed dodaniem rezerwacji przez pracownika czy jest na co najmniej 2 gości.

Implemetavcja Triggera.

```
CREATE trigger CheckIf2e on dbo.EmployeeReservations
   instead of insert
   as
   declare @guests int
   select @guests = O.guests from inserted o
   if @guests < 2
      begin
         raiserror ('Rezerwacja musi być na co najmniej 2 osoby', 1, 1)
         rollback
      end
   else
      begin
      insert into EmployeeReservations(reservationId, clientId, employeeId, guests,
reservationStart, reservationEnd, tableId)
      values ((select I.reservationId from inserted I), (select I.clientId from inserted I), (select
i.employeeid from inserted i), (select I.guests from inserted I), (select I.reservationStart
from inserted I), (select I.reservationEnd from inserted I), (select I.tableId from inserted I))
   end
go
```

Trigger Checkif2c

Trigger sprawdzający przed dodaniem rezerwacji firmowej czy jest na co najmniej 2 gości.

Implementacja triggera.

```
CREATE trigger CheckIf2c on dbo.CompanyReservations
   instead of insert
   as
   declare @guests int
   select @guests = O.guests from inserted o
   if @guests < 2
      begin
         raiserror ('Rezerwacja musi być na co najmniej 2 osoby', 1, 1)
         rollback
      end
   else
   begin
      insert into CompanyReservations(reservationId, clientId, guests, reservationStart,
reservationEnd)
      values((select i.reservationid from inserted i), (select i.clientid from inserted i), (select
i.guests from inserted i), (select i.reservationstart from inserted i), (select i.reservationend
from inserted i))
   end
go
```

Trigger CheckIfTableIsAvaliableI

Trigger sprawdza dostępność stolika podanego przez pracownika dla rezerwacji

Implementacja triggera.

```
create trigger CheckIftablesAvailableI on dbo.IndividualReservation
   after insert
   as

   begin
      if (select count(*) from
       dbo.AvailableTables((select i.guests from inserted i), (select i.reservationstart from
inserted i), (select i.reservationend from inserted i)) where tableId = (select i.tableid from
inserted i)) = 0
      begin
         raiserror('Stolik jest już zarezerwowany', 1, 1)
         rollback transaction
      end
   end
```

Trigger CheckIfTableIsAvaliableE

Trigger sprawdza dostępność stolika podanego przez pracownika dla rezerwacji firmy

Implementacja triggera.

```
create trigger CheckIftablesAvailableE on dbo.EmployeeReservations
   after insert
   as

   begin
      if (select count(*) from
       dbo.AvailableTables((select i.guests from inserted i), (select i.reservationstart from
inserted i), (select i.reservationend from inserted i)) where tableId = (select i.tableid from
inserted i)) = 0
      begin
         raiserror('Stolik jest już zarezerwowany', 1, 1)
         rollback transaction
      end
   end
go
```

Trigger UseTempDiscount

Trigger zamyka tymczasową zniżkę klientowi po wykonaniu zamówienia, jeżeli taka istnieje, oraz przypisuje ją do wykonanego zamówienia.

Implementacja triggera.

```
CREATE trigger UseTempDiscount
   on Orders
   after insert
   as
   begin
      update TempDiscount
      set ended = (select issueDate from inserted), orderId = (select orderId from inserted)
      where clientId = (select clientId from inserted) and orderId is null
   end
go
```

Opis i implementacja indeksów

By przyśpieszyć wyszukiwanie danych zaimplementowano indeksy dla następujących kolumn

1. Orders.issueDate
2. OrderDetails.quantity
3. Products.Price
4. TempDiscount.started i .ended
5. PermDiscount.started
6. CompanyReservations.started i .edned
7. EmployeeReservations.started i .edned
8. IndividualReservation.started i .ended

Implementacja indeksów.

```
create index Orders_issueDate_index on Orders (issueDate)
go

create index OrderDetails_quantity_index on OrderDetails (quantity)
go

create index Products_price_index on Products (price)
go

create index TempDiscount_started_ended_index
        on TempDiscount (started, ended)
go

create index PermDiscount_started_index
        on PermDiscount (started)
go

create index CompanyReservations_reservationStart_reservationEnd_index
        on CompanyReservations (reservationStart, reservationEnd)
go

create index EmployeeReservations_reservationStart_reservationEnd_index
        on EmployeeReservations (reservationStart, reservationEnd)
go

create index IndividualReservation_reservationStart_reservationEnd_index
        on IndividualReservation (reservationStart, reservationEnd)
go
```

Opis i implementacja roli w bazie danych

Rola Employee

Rola pracownika restauracji. Pracownik ma dostęp do widoków związanymi z rezerwacjami i stolikami, obecnego menu, ma dostęp do podglądu tabel z stolikami, produktami, klientami i zamówieniami, oraz ma dostęp do funkcji i procedur dodających klientów, dodających zamówienia i rezerwacje, obliczania kosztu zamówienia i przypisywania stolików do rezerwacji.

Implementacja roli.

```
create role Employee

grant select on CurrentMenu to Employee
grant select on MonthCompanyReservations to Employee
grant select on WeekEmployeesReservations to Employee
grant select on MonthEmployeeReservation to Employee
grant select on WeekEmployeesReservations to Employee
grant select on MonthIndividualReservations to Employee
grant select on WeekIndividualReservations to Employee
grant select on TablesReservedLastMonth to Employee
grant select on TablesReservedLastWeek to Employee
grant select on ClientMonthWeekOrderDetails to Employee
grant select on ClientMonthWeekOrderDetails to Employee
grant select on ClientMonthWeekOrders to Employee
grant select on ClientMonthlyOrderDetails to Employee
grant select on ClientMonthlyOrders to Employee
GRANT select on ClientWeekOrderDetails to Employee
grant select on ClientOrdersLastWeek to Employee
grant select on ClientOrdersLastMonth to Employee
grant select on ClientWeekOrders to Employee
grant select on CompanyClientMonthReservation to Employee
grant select on CompanyClientWeekReservation  to Employee
grant select on CompanyMonthReservation to Employee
grant select on CompanyWeekReservation to Employee
grant select on EmployeeClientMonthReservation to Employee
grant select on EmploeeMonthReservation to Employee
grant select on EmployeeClientWeekReservation to Employee
grant select on EmployeeWeekReservation to Employee
grant select on IndividualClientMonthReservation  to Employee
grant select on IndividualClientWeekReservation  to Employee
grant select on IndividualMonthReservation to Employee
grant select on IndividualWeekReservation to Employee
```

```
grant select on Categories to Employee
grant select on Clients to Employee
grant select on Companies to Employee
grant select on CompanyEmployees to Employee
grant select on Individual to Employee
grant select on Orders to Employee
grant select on OrderDetails to Employee
grant select on Products to Employee
grant select on Tables to Employee
grant select on AvailableTables to Employee

grant execute on AddCompany to Employee
grant execute on AddCompanyEmployee to Employee
grant execute on AddCompanyReservation to Employee
grant execute on AddCompanyReservationDetail to Employee
grant execute on AddEmployeeReservation to Employee
grant execute on AddIndividual to Employee
grant execute on AddIndividualReservation to Employee
grant execute on AddOrder to Employee
grant execute on AddOrderDetail to Employee
grant execute on OrderCost to Employee
grant execute on SetEmployeeReservationTableId to Employee
grant execute on SetIndividualReservationTableId to Employee

go
```

Rola Manager

Rola menadżera restauracji. Menadżer może przeglądać i modyfikować tabele opisujące restauracje takie jak produkty, kategorie produktów, stoliki i menu. Ma dostęp do wszystkich tabel i widoków poza słownikiem bazy. Może wywoływać wszystkie funkcje i procedury dodające klientów, dodających zamówienia i rezerwacje, obliczania kosztu zamówienia i przypisywania stolików do rezerwacji, oraz w odróżnieniu od pracownika, może ściągać oferty z menu.

Implementacja roli.

```
create role Manager

grant alter, delete, select on Categories to Manager
go

grant select on Clients to Manager
go

grant select on Companies to Manager
go
```

```
grant select on CompanyEmployees to Manager
go

grant select on CompanyReservationDetails to Manager
go

grant select on CompanyReservations to Manager
go

grant select on EmployeeReservations to Manager
go

grant select on Individual to Manager
go

grant select on IndividualReservation to Manager
go

grant alter, delete, select on Menu to Manager
go

grant select on OrderDetails to Manager
go

grant select on Orders to Manager
go

grant select on PermDiscount to Manager
go

grant alter, delete, select on Products to Manager
go

grant alter, delete, select on Tables to Manager
go

grant select on TempDiscount to Manager
go

grant select on ActiveDiscounts to Manager
go

grant select on CurrentMenu to Manager
go

grant select on DiscountsGrantedLastMonth to Manager
```

```
go

grant select on DiscountsGrantedLastWeek to Manager
go

grant select on MonthCompanyReservations to Manager
go

grant select on MonthEmployeeReservation to Manager
go

grant select on MonthIndividualReservations to Manager
go

grant select on OldMenuProducts to Manager
Go
grant select on ClientMonthlyOrders to Manager
grant select on ClientMonthlyOrderDetails to Manager
grant select on ClientMonthWeekOrderDetails to Manager
grant select on ClientMonthWeekOrderDetails to Manager
grant select on ClientMonthWeekOrders to Manager
grant select on ClientOrdersLastMonth to Manager
grant select on ClientOrdersLastWeek to Manager
GRANT select on ClientWeekOrderDetails to Manager
GRANT select on ClientWeekOrders to Manager
grant select on ClientWeekOrders to Manager
grant select on CompanyClientMonthReservation to Manager
grant select on CompanyClientWeekReservation  to Manager
grant select on CompanyMonthReservation to Manager
grant select on CompanyWeekReservation to Manager
grant select on EmployeeClientMonthReservation to Manager
grant select on EmploeeMonthReservation to Manager
grant select on EmployeeClientWeekReservation to manager
grant select on EmployeeWeekReservation to Manager
grant select on IndividualClientMonthReservation  to Manager
grant select on IndividualClientWeekReservation  to Manager
grant select on IndividualMonthReservation to Manager
grant select on IndividualWeekReservation to Manager



grant execute on MenuAlertJob to Manager
go

grant select on WeekEmployeesReservations to Manager
go
```

```
grant execute on AddCategory to Manager
go

grant execute on AddCompany to Manager
go

grant execute on AddCompanyEmployee to Manager
go

grant execute on AddCompanyReservation to Manager
go

grant execute on AddCompanyReservationDetail to Manager
go

grant execute on AddIndividual to Manager
go

grant execute on AddIndividualReservation to Manager
go

grant execute on AddOrder to Manager
go

grant execute on AddOrderDetail to Manager
go

grant execute on AddProduct to Manager
go

grant execute on AddProductToMenu to Manager
go

grant select on AvailableTables to Manager
go

grant execute on SetEmployeeReservationTableId to Manager
go

grant execute on SetIndividualReservationTableId to Manager
go

grant execute on SetItemEnd to Manager
go
```

Rola Individual

Rola Klient Indywidualny. Klient może edytować bądź usunąć swoje konto. Ma on również możliwość do zaglądania do widoków takich jak obcene menu, oraz zobaczenia swoich zamówień ostatniego miesiąca I tygodnia. Również ma dostęp do funckji sprawdzających jego wydatki miesięczne oraz na pojedyńcze zamówienie. Ma on możliwość zobaczenia wielkości swojej zniżki

Implementacja roli.

```
create role Individual

grant select on ClientOrdersLastMonth to Individual
grant select on ClientOrdersLastWeek to Individual
grant select on CurrentMenu to Individual
GRANT execute on ClientWeekOrders to Individual
GRANT select on ClientWeekOrderDetails to Individual
grant select on ClientOrdersLastWeek to Individual
grant select on ClientOrdersLastMonth to Individual
grant select on ClientMonthWeekOrders to Individual
grant select on ClientMonthWeekOrderDetails to Individual
grant select on ClientMonthlyOrderDetails to Individual
grant select on ClientMonthlyOrders  to Individual
grant select on ClientWeekOrders to Individual
grant select on IndividualClientMonthReservation  to Individual
grant select on IndividualClientWeekReservation  to Individual
grant select on IndividualMonthReservation to Individual
grant select on IndividualWeekReservation to Individual


grant execute on ClientExpensesLastMonth to Individual
grant execute on ClientExpensesLastWeek to Individual
grant execute on OrderCost to Individual
grant execute on ClientDiscountValue to Individual
```

Rola Company

Rola Firma. Firma może edytować bądź usunąć swoje konto. Ma ona również możliwość do zaglądania do widoków takich jak obcene menu, oraz zobaczenia swoich zamówień ostatniego miesiąca I tygodnia. Również ma dostęp do funckji sprawdzających ich wydatki miesięczne oraz na pojedyńcze zamówienie. Ma ona możliwość zobaczenia wielkości swojej zniżki

Implementacja roli.

```
create role Company

grant select on CurrentMenu to Company
grant select on ClientOrdersLastMonth to Company
grant select on ClientOrdersLastWeek to Company
grant select on ClientMonthlyOrders to Company
grant select on ClientMonthlyOrderDetails to Company
grant select on ClientMonthWeekOrderDetails to Company
grant select on ClientMonthWeekOrderDetails to Company
grant select on ClientMonthWeekOrders to Company
grant select on ClientOrdersLastMonth to Company
grant select on ClientOrdersLastWeek to Company
GRANT select on ClientWeekOrderDetails to Company
GRANT execute on ClientWeekOrders to Company
grant select on ClientMonthlyOrderDetails to Company
grant select on ClientWeekOrders to Company
grant select on CompanyClientMonthReservation to Company
grant select on CompanyClientWeekReservation  to Company
grant select on CompanyMonthReservation to Company
grant select on CompanyWeekReservation to Company


grant execute on ClientExpensesLastMonth to Company
grant execute on ClientExpensesLastWeek to Company
grant execute on OrderCost to Company
grant execute on ClientDiscountValue to Company
```

Rola CompanyEmployee

Rola Klient firmowy. Klient firmowy może edytować bądź usunąć swoje konto. Ma on również możliwość do zaglądania do widoków takich jak obcene menu, oraz zobaczenia swoich zamówień ostatniego miesiąca I tygodnia. Również ma dostęp do funckji sprawdzających jego wydatki miesięczne oraz na pojedyńcze zamówienie. Ma on możliwość zobaczenia wielkości swojej zniżki

Implementacja roli.

```
create role CompanyEmployee

grant select on CurrentMenu to CompanyEmployee
grant select on ClientOrdersLastMonth to CompanyEmployee
grant select on ClientOrdersLastWeek to CompanyEmployee
GRANT execute on ClientWeekOrders to CompanyEmployee
GRANT select on ClientWeekOrderDetails to CompanyEmployee
grant select on ClientOrdersLastWeek to CompanyEmployee
grant select on ClientOrdersLastMonth to CompanyEmployee
grant select on ClientMonthWeekOrders to CompanyEmployee
grant select on ClientMonthWeekOrderDetails to CompanyEmployee
grant select on ClientMonthWeekOrderDetails to CompanyEmployee
grant select on ClientMonthlyOrderDetails to CompanyEmployee
grant select on ClientMonthlyOrders to CompanyEmployee
grant select on ClientWeekOrders to CompanyEmployee
grant select on EmployeeClientMonthReservation to CompanyEmployee
grant select on EmploeeMonthReservation to CompanyEmployee
grant select on EmployeeClientWeekReservation to CompanyEmployee
grant select on EmployeeWeekReservation to CompanyEmployee

grant execute on ClientExpensesLastMonth to CompanyEmployee
grant execute on ClientExpensesLastWeek to CompanyEmployee
grant execute on OrderCost to CompanyEmployee
grant execute on ClientDiscountValue to CompanyEmployee
```