Projekt Bazy Danych – Dokumentacja

Zespół 12:

Szymon Twardosz

Filip Piskorski

Dominik Puz

Spis treści

Użytkownicy systemu: 5

Funkcje użytkowników: 5

Administrator: 5

Klient Indywidualny: 5

Klient – Firma 5

Kucharz: 6

Kelner: 6

System: 6

DDL: 8

Bills 8

Categories 8

Clients 9

CompanyClients 9

IndividualClients 10

MenuDetails 11

Menus 11

OneTimeDiscounts 12

OrderDetails 13

Orders 14

Reservations 15

Payments 16

Products 17

ReservationDetails 17

ReservationGuestList 18

Tables 18

Constants 19
Schemat bazy danych 20
Widoki 21
ActualMenu 21
ActualReservation 21
BillReportCompany 21
BillReportIndividual 21
CategoryAndProducts 22
ClientToMeal 22
CompanyGuestList 22
DiscountInfo 22
FutureMenus 23
NonAcceptedReservation 23
OrderReportMon 23
OrderReportWeek 24
OrdersPayment 24
OrdersCounter 24
ProductInfoMon 25
ProductInfoWeek 25
ReservationReportMon 25
ReservationReportWeek 25
TablesReport 26
TakeAwayOrders 26
TableOccupancy 26
Procedury 27
AcceptReservation 27
AddBill 27
AddOrderToBill 28
AddProduct 28
AddToMenu 29
ChangeDiscountCoefficients 29
ChangePriceInMenu 30
ChangePriceInProducts 30
ChangeReservationEndtime 31
RemoveFromMenu 31
AddReservation 32

AddOrder....34
AddOrderToReservation....36
AddTableToReservation....37
AddCompanyClient....38
AddIndividualClient....38
AddProductToOrder....39
AddMenu....40
AddPayment....40
AddReservationGuest....41
CheckReservationCorrect....42
ChangeDiscountStatus....42
DeleteIncorrectReservation....43
Funkcje....44
ClientOrderReportMon....44
ClientOrderReportWeek....44
ClientOrdersHistory....45
ClientReservationReportMon....45
ClientReservationReportWeek....46
ViewClientDiscounts....46
ViewClientUnusedDiscounts....47
ViewBillsOfAClient....47
ViewOrdersOfABill....47
ReturnOrderWithoutBill....48
CheckProductInMenu....48
ChangeMenuStatus....49
FindFreeTable....49
GetOrdersCnt....50
GetMenu....50
GetOrderValue....50
GetOrderID....50
GetMenuID....51
GetReservationID....51
IsTableFree....51
OrderPaySum....52
IsOrderPaid....52
ViewUnpaidOrdersOfAClient....53

ViewOrdersOfPayment....53
Triggery....54
DeleteDiscount....54
CheckRequirementsForOneTimeDiscount....55
CheckRequirementsForPermanentDiscount....56
CheckRequirementsForPermanentDiscountOnDelete....57
Indeksy....58
IX_OneTimeDiscount....58
IX_OneTimeDiscount....58
IX_Orders....58
IX_Orders_1....58
IX_Orders_2....58
IX_Payments....59
IX_Products....59
IX_ReservationGuestList....59
IX_Reservations....59
Indeksy dla kluczy głównych....59

Użytkownicy systemu:

- Szef (administrator)
- Klient indywidualny
- Klient (firma)
- Kucharz
- Kelner

Funkcje użytkowników:

Administrator:

1. **Dodawanie produktów do bazy danych –** AddProduct, ChangePriceInProducts
2. **Tworzenie Menu –** AddMenu, AddToMenu, ChangePriceInMenu, RemoveFromMenu, ChangeMenuStatus
3. **Edytowanie współczynników dotyczących rabatów –** ChangeDiscountCoefficients
4. **Dodanie klienta (indywidualnego lub firmy) –** AddCompanyClient, AddIndividualClient

Klient Indywidualny:

1. **Generowanie rachunków/faktur** – AddBill
2. **Dodanie nowej rezerwacji** – AddReservation
3. **Dodanie zamówienia (na wynos lub na miejscu) –** AddOrder, AddOrderToReservation, AddProductToOrder
4. **Sprawdzenie dostępnych stolików -** FindFreeTable
5. **Rezerwacja wybranego stolika –** AddTableToReservation
6. **Sprawdzenie Menu -** GetMenu
7. **Złożenie płatności –** AddPayment
8. **Generowanie raportów tygodniowych\miesięcznych dotyczących:**
    a) Zamówień – ClientOrderReportMon, ClientOrderReportWeek, ClientOrderHistory, GetOrderCnt
    b) Rezerwacji – ClientReservationReportMon, ClientReservationReportWeek
    c) Zniżek – ViewClientDiscount, ViewClientUnusedDiscounts

Klient – Firma

1. **Generowanie rachunków/faktur –** AddBill
2. **Dodanie nowej rezerwacji –** AddReservation
3. **Dodanie zamówienia (na wynos lub na miejscu) –** AddOrder, AddOrderToReservation, AddProductToOrder

4. **Sprawdzenie dostępnych stolików –** FindFreeTable
5. **Sprawdzenie Menu -** GetMenu
6. **Rezerwacja wybranego stolika –** AddTableToReservation
7. **Rezerwacja stolików dla pracowników -** AddReservationGuest
8. **Złożenie płatności –** AddPayment
9. **Generowanie raportów tygodniowych\miesięcznych dotyczących:**
    a) Zamówień – ClientOrderReportMon, ClientOrderReportWeek, ClientOrderHistory, GetOrderCnt
    b) Rezerwacji – ClientReservationReportMon, ClientReservationReportWeek
    c) Zniżek – ViewClientDiscount, ViewClientUnusedDiscounts

Kucharz:

1. **Tworzenie Menu** – AddMenu,AddToMenu, ChangePriceInMene, RemoveFromMenu, ChangeMenuStatus
2. **Edytowanie dostępnych produktów –** AddProduct, ChangePriceInProducts

Kelner:

1. **Akceptacja rezerwacji –** AcceptReservation
2. **Zmiana atrybutów rezerwacji (czas zakończenia)** – ChangeReservationEndtime
3. **Dodanie nowej rezerwacji** – AddReservation
4. **Dodanie zamówienia (na wynos lub na miejscu) –** AddOrder, AddOrderToReservation, AddProductToOrder
5. **Sprawdzenie dostępnych stolikow –** FindFreeTable
6. **Sprawdzenie Menu -** GetMenu
7. **Rezerwacja wybranego stolika –** AddTableToReservation
8. **Dodanie klienta (indywidualnego lub firmy) –** AddCompanyClient, AddIndividualClient
9. **Złożenie płatności –** AddPayment
10. **Generowanie raportów tygodniowych\miesięcznych dotyczących:**
    a) Zamówień – ClientOrderReportMon, ClientOrderReportWeek, ClientOrderHistory
    b) Rezerwacji – ClientReservationReportMon, ClientReservationReportWeek
    c) Zniżek – ViewClientDiscount, ViewClientUnusedDiscounts

System:

1. **Sprawdzanie wymagań rezerwacji –** CheckReservationCorrect
2. **Usuwanie nielegalnych rezerwacji –** DeleteIncorrectReservation
3. **Sprawdzanie dostępność produktu w Menu –** CheckProductInMenu

4. **Sprawdza ile klient wykonał zamówień w przeszłości –** GetOrderCnt
5. **Oblicza wartość zamówienia –** GetOrderValue
6. **Sprawdza czy stolik jest wolny –** IsTableFree
7. **Obsługa zniżek (dodanie, usuwanie, zmiana statusu) –** ChangeDiscountStatus, DeleteDiscount, CheckRequirementsForOneTimeDiscount, CheckRequirementsForPermamentDiscount, CheckRequirementsForPermamentDiscountOnDelete,

DDL:

Bills

```
create table Bills
(
    BillID       int      not null
        constraint PK_Bills
            primary key,
    IssueDate    datetime not null,
    IsCollective bit      not null
)
```

BillID - klucz główny
IssueDate - data wystawienia rachunku
IsCollective - flaga określająca czy rachunek jest zbiorowy

Categories

```
create table Categories
(
    CategoryID          int          not null
        constraint PK_Categories
            primary key,
    CategoryName        varchar(64) not null,
    CategoryDescription varchar(255)
)
```

CategoryID - klucz główny

CategoryName - nazwa kategorii

CategoryDescription - opis kategorii

Clients

```
create table Clients
(
    ClientID int not null
        constraint PK_Clients
            primary key
)
```

ClientID - klucz główny

CompanyClients

```
create table CompanyClients
(
    ClientID    int not null
        constraint PK_CompanyClients
            primary key
        constraint FK_CompanyClients_Clients
            references Clients,
    CompanyName varchar(64),
    NIP         varchar(16),
    City        varchar(64),
    Street      varchar(64),
    PostalCode  varchar(16),
    PhoneNumber varchar(16),
    Email       varchar(64)
)
```

ClientID - klucz główny i obcy
CompanyName - nazwa firmy
NIP - numer NIP
City - miasto klienta
Street - ulica
PostalCode - kod pocztowy
PhoneNumber - numer telefonu
Email - adres email

IndividualClients

```
create table IndividualClients
(
    ClientID              int         not null
        constraint PK_IndividualClients
            primary key
        constraint FK_IndividualClients_Clients
            references Clients,
    FirstName             varchar(64) not null,
    LastName              varchar(64) not null,
    PhoneNumber           varchar(16),
    Email                 varchar(64),
    PermamentDiscountLVL int          not null
)
```

ClientID - klucz główny i obcy
FirstName - imię klienta
LastName - nazwisko klienta
PhoneNumber - numer telefonu
Email - adres email
PermanentDiscountLVL - poziom stałej zniżki (0 - brak zniżki 1 - zniżka zdefiniowana w wymaganiach) - pozwala na implementację wielu poziomów zniżek

MenuDetails

```
create table MenuDetails
(
    MenuID     int             not null
        constraint FK_MenuDetails_Menus
            references Menus,
    ProductID int              not null
        constraint FK_MenuDetails_Products
            references Products,
    UnitPrice decimal(18, 2) not null
        constraint CK_MenuDetails
            check ([UnitPrice] >= 0),
    constraint PK_MenuDetails
        primary key (MenuID, ProductID)
)
```

MenuID - klucz główny i obcy

ProductID - klucz główny i obcy

UnitPrice - cena produktu

Warunki integralności:

- Cena nie może być ujemna

Menus

```
create table Menus
(
    MenuID     int identity
        constraint PK_Menus
            primary key,
    StartTime date not null,
    EndTime   date not null,
    IsLegal   bit,
    constraint CK_Menus
        check (datediff(day, [StartTime], [EndTime]) < 14)
)
```

MenuID - klucz główny
StartTime - początkowa data obowiązywania menu
EndTime - końcowa data obowiązywania menu
Warunki integralności:
- różnica między StartTime i EndTime nie może być większa niż 14 dni

OneTimeDiscounts

```
create table OneTimeDiscounts
(
    OTDiscountID int identity
        constraint PK_OneTimeDiscounts
            primary key,
    ClientID     int      not null
        constraint FK_OneTimeDiscounts_IndividualClients
            references IndividualClients,
    used         bit      not null,
    ApplyDate    datetime not null,
    ExpireDate   datetime not null,
    Value        float
        constraint CK_OneTimeDiscounts
            check ([Value] > 0),
    OrderID      int      not null
)
```

OTDiscountID - klucz główny
ClientID - klucz obcy
used - flaga określająca czy rabat został już zużyty
ApplyDate - data wystawienia rabatu
ExpireDate - data upłynięcia ważności
Value - wartość procentowa rabatu
OrderID - id zamówienia po którym został przyznany rabat
Warunki integralności:
- wartość musi być większa od 0

OrderDetails

```
create table OrderDetails
(
    OrderID          int             not null
        constraint FK_OrderDetails_Orders
            references Orders,
    ProductID        int             not null
        constraint FK_OrderDetails_Products
            references Products,
    PlannedServeTime datetime        not null,
    Served           bit             not null,
    Quantity         float           not null
        constraint CK_OrderDetails_1
            check ([Quantity] > 0),
    UnitPrice        decimal(18, 2) not null
        constraint CK_OrderDetails
            check ([UnitPrice] >= 0),
    constraint PK_OrderDetails
        primary key (OrderID, ProductID)
)
```

OrderID - klucz główny i obcy
ProductID - klucz główny i obcy
PlannedServeTime - planowany czas podania zamówienia
Served - flaga określająca czy zamówienie zostało już podane
Quantity - ilość produktu
UnitPrice - cena za 1 sztukę produktu
Warunki integralności:
- Cena nie może być ujemna

Orders

```
create table Orders
(
    OrderID          int not null
        constraint PK_Orders
            primary key,
    ClientID         int
        constraint FK_Orders_Clients
            references Clients,
    ReservationID    int
        constraint FK_Orders_Reservations
            references Reservations,
    BillID           int
        constraint FK_Orders_Bills
            references Bills,
    OrderTime        datetime,
    TakeawayTime     datetime,
    AppliedDiscount  float,
    OTDiscountID     int
        constraint FK_Orders_OneTimeDiscounts
            references OneTimeDiscounts
)
```

OrderID - klucz główny
ClientID - klucz obcy
ReservationID - klucz obcy, jeżeli zamówienie jest na wynos to wartość jest null
BillID - klucz obcy, jeżeli nie został jeszcze wystawiony rachunek t o wartość jest null
OrderTime - czas złożenia zamówienia
TakeawayTime - czas odbioru zamówienia na wynos (jeżeli zamówienie nie jest na wynos to wartość jest null)
AppliedDiscount - wartość rabatu zastosowanego do zamówienia
OTDDiscountID - klucz obcy, jeżeli rabat nie został użyty to wartość jest null

Reservations

```
create table Reservations
(
    ReservationID        int identity
        constraint PK_Reservations
            primary key,
    ClientID             int      not null
        constraint FK_Reservations_Clients
            references Clients,
    StartTime            datetime not null,
    EndTime              datetime not null,
    IsCompanyReservation bit      not null,
    NumberOfPeople       int      not null,
    Accepted             bit
)
go
```

Relacja Reservations – Orders jest 1 do wielu, ponieważ przyjmujemy, że do jednej rezerwacji będzie można złożyć wiele zamówień.

ReservationID - klucz główny
ClientID - klucz obcy
StartTime - data i godzina rezerwacji
EndTime - przewidywany czas końca rezerwacji
IsCompanyReservation - flaga określająca czy rezerwacja jest na firmę
NumberOfPeople - ilość osób przypisanych do rezerwacji
Accepted - flaga określająca czy rezerwacja jest zaakceptowana

Payments

```
create table Payments
(
    PaymentID   int             not null
        constraint PK_Payments
            primary key,
    OrderID     int             not null
        constraint FK_Payments_Orders
            references Orders,
    Amount      decimal(18, 2) not null,
    Description varchar(64),
    PayDate     datetime        not null
)
```

PaymentID - klucz główny
OrderID - klucz obcy
Amount - wartość wpłaty
Description - opis wpłaty
PayDate - data wpłaty

Products

```
create table Products
(
   ProductID          int          not null
       constraint PK_Products
           primary key,
   CategoryID         int
       constraint Category
           references Categories,
   ProductName        varchar(64) not null,
   IsImportedFlag     bit,
   UnitPrice          decimal(18, 2)
       constraint CK_Products
           check ([UnitPrice] >= 0),
   ProductDescription varchar(255),
   Active             bit
)
```

ProductID - klucz główny
CategoryID - klucz obcy
ProductName - nazwa produktu
IsImportedFlag - flaga określająca czy produkt jest importowany
UnitPrice - cena produktu za sztukę
ProductDescription - opis produktu
Active - flaga określająca czy produkt jest dostępny
Warunki integralności:
- cena nie może być ujemna

ReservationDetails

```
create table ReservationDetails
(
   ReservationID int not null
       constraint FK_ReservationDetails_Reservations
           references Reservations,
   TableID       int not null
       constraint FK_ReservationDetails_Tables
           references Tables,
   constraint PK_ReservationDetails
       primary key (ReservationID, TableID)
)
```

ReservationID - klucz główny i obcy
TableID - klucz główny i obcy

ReservationGuestList

```
create table ReservationGuestList
(
    ReservationGuestID int          not null
        constraint PK_ReservationGuestList
            primary key,
    ReservationID      int          not null
        constraint FK_ReservationGuestList_Reservations
            references Reservations,
    FirstName          varchar(64)  not null,
    LastName           varchar(64)  not null
)
```

ReservationGuestID - klucz głównu
ReservationOD - klucz obcy
FirstName - imie klienta
LastName - nazwisko klienta

Tables

```
create table Tables
(
    TableID int not null
        constraint PK_Tables
            primary key,
    Size    int not null
        constraint CK_Tables
            check ([Size] >= 2),
    Active  bit
)
```

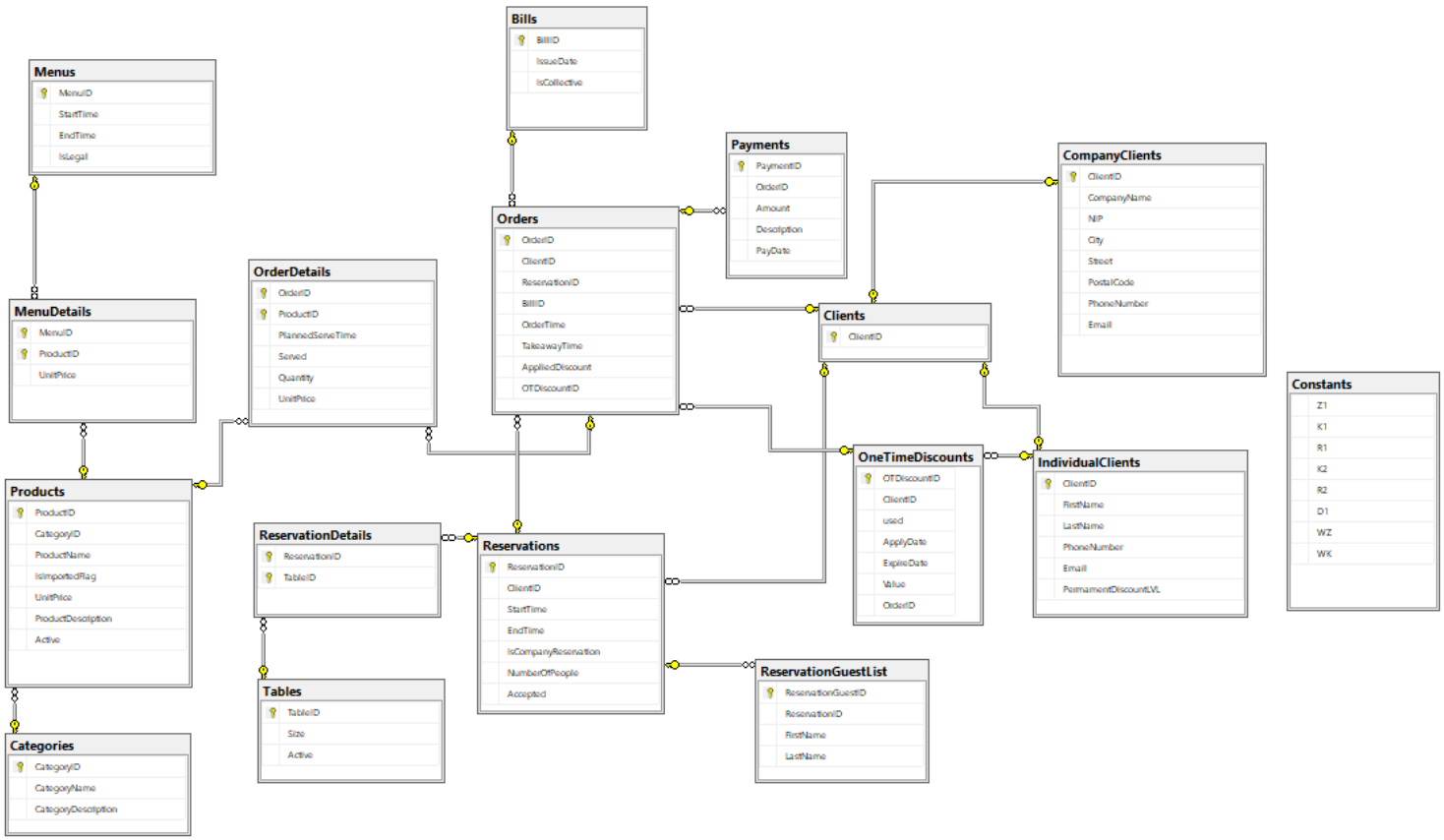
TableID - klucz główny
Size - rozmiar stołu
Active - flaga określająca czy stół jest dostępny
Warunki integralności:
- minimalny rozmiar stołu to 2

Constants

```
create table Constants
(
    Z1 int,
    K1 decimal(18, 2),
    R1 float,
    K2 decimal(18, 2),
    R2 float,
    D1 int,
    WZ decimal(18, 2),
    WK int
)
```

Z1 - minimalne liczba zamówień do rabatu permanentnego
K1 - minimalna wartość zamówień do rabatu permanentnego
R1 - wartość rabatu permanentnego
K2 - wartość sumy zamówień do rabatu jednorazowego
R2 - wartość rabatu jednorazowego
D1 - liczba dni przez, które rabat jednorazowy jest ważny
WZ - minimalna wartość zamówienia przy rezerwacji
WK - minimalna liczba wcześniejszych zamówień przy rezerwacji

Schemat bazy danych



Widoki

ActualMenu

Zwraca Menu, które obowiązuje w danym

```
create view ActualMenu as
    select P.ProductID, P.ProductName, P.ProductDescription,
P.UnitPrice
    from Menus M
    join MenuDetails MD on M.MenuID = MD.MenuID
    join Products P on P.ProductID = MD.ProductID
    where getdate() between M.StartTime and M.EndTime
```

ActualReservation

Zwraca rezerwację obowiązujące na dzisiejszy dzień

```
 create view ActualReservation as
    select *
    from Reservations
    where DATEPART(day, StartTime) = DATEPART(day, getdate())
```

BillReportCompany

Zwraca informację na temat faktur dla firm

```
create view BillReportCompany as
    select B.BillID, B.IssueDate, B.PayAmount,
           CC.ClientID, CC.CompanyName, CC.NIP, CC.Email,
CC.PhoneNumber,
           CC.PostalCode, CC.City, CC.Street
    from Bills B
    join Orders O on B.BillID = O.BillID
    join Clients C on O.ClientID = C.ClientID
    join CompanyClients CC on C.ClientID = CC.ClientID
```

BillReportIndividual

Zwraca informację na temat faktur dla klientów indywidualnych

```
create view BillReportIndividual as
    select B.BillID, B.IssueDate, B.PayAmount, IC.ClientID,
IC.FirstName, IC.LastName, IC.Email, IC.PhoneNumber
    from Bills B
    join Orders O on B.BillID = O.BillID
    join Clients C on O.ClientID = C.ClientID
    join IndividualClients IC on C.ClientID = IC.ClientID
```

CategoryAndProducts

Zwraca listę produktów dla każdej kategorii

```
create view CategoryAndProducts as
    select C.CategoryID,C.CategoryName, P.ProductName from Categories C
    join Products P on C.CategoryID = P.CategoryID
```

ClientToMeal

Dla każdego klienta zwraca produkty, które zamówił wraz z ich ilością

```
create view ClientToMeal as
    select IC.ClientID, IC.FirstName, IC.LastName, P.ProductID,
P.ProductName,
           isnull(sum(od.Quantity),0) 'OrdersSum'
    from IndividualClients IC
    join Clients C on C.ClientID = IC.ClientID
    join Orders O on C.ClientID = O.ClientID
    join OrderDetails OD on O.OrderID = OD.OrderID
    join Products P on P.ProductID = OD.ProductID
    group by IC.ClientID, IC.FirstName, IC.LastName, P.ProductID,
P.ProductName
```

CompanyGuestList

Dla każdej firmy zwraca listę gości przypisanych do każdego zamówienia

```
create view CompanyGuestList as
    select CC.ClientID,CC.CompanyName, R2.StartTime ,RGL.FirstName,
RGL.LastName
    from ReservationGuestList RGL
    join Reservations R2 on R2.ReservationID = RGL.ReservationID
    join Clients C on C.ClientID = R2.ClientID
    join CompanyClients CC on C.ClientID = CC.ClientID
```

DiscountInfo

Dla każdego klienta indywidualnego zwraca informację na temat jego zniżek

```
alter view DiscountInfo as
    select IC.ClientID, IC.FirstName, IC.LastName,
OTD.OTDiscountID,OTD.ApplyDate, OTD.Value, OTD.used
    from IndividualClients IC
    join OneTimeDiscounts OTD on IC.ClientID = OTD.ClientID
```

FutureMenus

Wyświetla wszystkie przyszłe Menu wraz z aktualnym

```
alter view FutureMenus as
    select M.MenuID, M.StartTime, M.EndTime,P.ProductID, P.ProductName from
Menus M
    join MenuDetails MD on M.MenuID = MD.MenuID
    join Products P on P.ProductID = MD.ProductID
    where getdate() <= M.StartTime or getdate() between M.StartTime and
M.EndTime
```

NonAcceptedReservation

Zwraca wszystkie rezerwację, które nie zostały jeszcze zaakceptowane

```
create view NotAcceptedReservation as
    select ReservationID, ClientID, StartTime, EndTime,
IsCompanyReservation, NumberOfPeople
    from Reservations
    where Accepted = 0
```

OrderReportMon

Dla każdego klienta zwraca wartość produktów które zamówił, z podziałem na lata i miesiące

```
create view OrderReportMon as
    select C.ClientID, 0 'isCompanyClient',year(O.OrderTime) 'Year',
           month(O.OrderTime) 'Month', isnull(sum(O.OrderID),0) as
'OrdersSum',
           isnull(sum(OD.UnitPrice * OD.Quantity) * (1 -
max(O.AppliedDiscount)),0) 'TotalMoneySpend'
    from Clients C
    join IndividualClients IC on C.ClientID = IC.ClientID
    join Orders O on C.ClientID = O.ClientID
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by C.ClientID,year(O.OrderTime), month(O.OrderTime)
    union
    select C.ClientID, 1 'isCompanyClient',year(O.OrderTime) 'Year',
           month(O.OrderTime) 'Month', isnull(sum(O.OrderID),0) as
'OrdersSum',
           isnull(sum(OD.UnitPrice * OD.Quantity) * (1 -
max(O.AppliedDiscount)),0) 'TotalMoneySpend'
    from Clients C
    join CompanyClients CC on C.ClientID = CC.ClientID
    join Orders O on C.ClientID = O.ClientID
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by C.ClientID, year(O.OrderTime), month(O.OrderTime)
```

OrderReportWeek

Dla każdego klienta zwraca wartość produktów które zamówił, z podziałem na lata i miesiące i tygodnie.

```
create view OrderReportWeek as
    select C.ClientID, 0 'isCompanyClient',year(O.OrderTime) as 'Year',
           DATEPART(week,O.OrderTime) 'Week', isnull(sum(O.OrderID),0) as
'OrdersSum',
           isnull(sum(OD.UnitPrice * OD.Quantity) * (1 -
max(O.AppliedDiscount)),0) 'TotalMoneySpend'
    from Clients C
    join IndividualClients IC on C.ClientID = IC.ClientID
    join Orders O on C.ClientID = O.ClientID
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by C.ClientID, year(O.OrderTime) ,DATEPART(week, O.OrderTime)
    union
    select C.ClientID, 1 'isCompanyClient',year(O.OrderTime) as 'Year',
DATEPART(week,O.OrderTime) 'Month', isnull(sum(O.OrderID),0) as
'OrdersSum',
           isnull(sum(OD.UnitPrice * OD.Quantity) * (1 -
max(O.AppliedDiscount)),0) 'TotalMoneySpend'
    from Clients C
    join CompanyClients CC on C.ClientID = CC.ClientID
    join Orders O on C.ClientID = O.ClientID
    join OrderDetails OD on O.OrderID = OD.OrderID
    group by C.ClientID,year(O.OrderTime), DATEPART(week, O.OrderTime)
```

OrdersPayment

Zwraca te zamówienia, które nie zostały jeszcze zapłacone wraz z liczbą pieniędzy do zapłaty

```
create view OrdersPayment as
    select O.OrderID,O.ClientID,O.OrderTime, isnull(P.Amount, 0) as
'PaidMoney',
           cast(isnull((select sum(od.Quantity * od.UnitPrice) from
OrderDetails od
                               where od.OrderID = O.OrderID),0) * (1 -
O.AppliedDiscount) - isnull(P.Amount,0) as DECIMAL(10, 2)) as 'MoneyToPay'
    from Orders O
    join Payments P on O.OrderID = P.OrderID
    where P.Amount < isnull((select sum(od.Quantity * od.UnitPrice) from
OrderDetails od
                               where od.OrderID = O.OrderID),0) * (1 -
O.AppliedDiscount)
```

OrdersCounter

Dla każdego klienta sprawdza ile razy złożył zamówienie w przeszłości.

```
alter view OrdersCounter
as
    select O.ClientID, isnull(count(O.OrderID),0) as 'OrderCounter' from
Clients C
    left join Orders O on C.ClientID = O.ClientID
    group by O.ClientID
```

ProductInfoMon

Dla każdego produktu ilość zamówionych sztuk, wraz sumą ich cen z podziałem na lata i miesiące

```
create view ProductInfoMon as
    select P.ProductID, P.ProductName, year(O.OrderTime)
'Year',month(O.OrderTime) as 'Month',
           sum(OD.Quantity) as 'OrdersSum',
           sum(OD.UnitPrice * od.Quantity) * (1 - max(O.AppliedDiscount))
as 'Total income'
    from Products P
    left join OrderDetails OD on P.ProductID = OD.ProductID
    left join Orders O on O.OrderID = OD.OrderID
    group by P.ProductID, P.ProductName, year(O.OrderTime),
month(O.OrderTime)
```

ProductInfoWeek

Dla każdego produktu ilość zamówionych sztuk, wraz sumą ich cen z podziałem na lata i miesiące i tygodnie.

```
create view ProductInfoWeek as
    select P.ProductID, P.ProductName,year(O.OrderTime) 'Year',
DATEPART(week,O.OrderTime) 'Week',
           sum(OD.Quantity) as 'OrdersSum',
           sum(OD.UnitPrice * od.Quantity) * (1 - max(O.AppliedDiscount))
as 'Total income'
    from Products P
    left join OrderDetails OD on P.ProductID = OD.ProductID
    left join Orders O on O.OrderID = OD.OrderID
    group by P.ProductID, P.ProductName,year(O.OrderTime), DATEPART(week,
O.OrderTime)
```

ReservationReportMon

Zwraca liczbę rezerwacji dla każdego klienta z podziałem na lata i miesiące

```
create view ReservationReportMon as
    select C.ClientID, year(R.StartTime) 'Year', month(R.StartTime) as
'Month', count(R.ReservationID) as 'NumberOfReservations'
    from Reservations R
    join Clients C on C.ClientID = R.ClientID
    group by C.ClientID, year(R.StartTime), month(R.StartTime)
```

ReservationReportWeek

Zwraca liczbę rezerwacji dla każdego klienta z podziałem na lata i miesiące i tygodnie

```
alter view ReservationReportWeek as
    select C.ClientID, year(R.StartTime) 'Year', datepart(week,
R.StartTime) as 'Week', count(R.ReservationID) as 'NumberOfReservations'
    from Reservations R
    join Clients C on C.ClientID = R.ClientID
    group by C.ClientID, year(R.StartTime), datepart(week, R.StartTime)
```

TablesReport

Dla każdego stolika zwraca liczbę ile razy był on zajęty przez klientów

```
create view TablesReport as
    select T.TableID, T.Size, isnull(count(RD.ReservationID),0)
'TableOccupancy'
    from Tables T
    left join ReservationDetails RD on T.TableID = RD.TableID
    group by T.TableID, T.Size
```

TakeAwayOrders

Zwraca informację na temat zamówień na wynos, które nie zostały jeszcze wydane

```
create view TakeAwayOrders as
    select O.OrderID, ClientID, OrderTime, TakeawayTime, AppliedDiscount
from Orders O
    where O.TakeawayTime is not null and getdate() < O.TakeawayTime
```

TableOccupancy

Zwraca informację na temat stolików oraz dat kiedy są zarezerwowane

```
CREATE view TableOccupancy as
    select T.TableID, T.Size, isnull(R2.StartTime, '1900-01-12 10:30')
'StartTime', isnull(R2.EndTime, '1900-01-12 10:30') 'EndTime'
    from Tables T
    left join ReservationDetails RD on T.TableID = RD.TableID
    left join Reservations R2 on R2.ReservationID = RD.ReservationID
go
```

Procedury

AcceptReservation

Procedura ustawia flagę Accepted danej rezerwacji na true.

```
CREATE PROCEDURE [dbo].[AcceptReservation]
   -- Add the parameters for the stored procedure here
   @ReservationID int
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE Reservations
   SET Accepted = 1
   where ReservationID = @ReservationID
END
go
```

AddBill

Procedura tworzy nowy rekord w tabeli Bills.

```
CREATE PROCEDURE AddBill
   -- Add the parameters for the stored procedure here
   @IsCollective bit
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   insert into Bills (IssueDate, IsCollective)
   values (getdate(), @IsCollective)
END
go
```

AddOrderToBill

Procedura ta ustawia BillID w danym rekordzie tabeli Orders na również podane w argumencie BillID.

```
CREATE PROCEDURE AddOrderToBill
   -- Add the parameters for the stored procedure here
   @OrderID int,
   @BillID int
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE Orders
   set BillID = @BillID
   where OrderID = @OrderID
END
go
```

AddProduct

Procedura dodaje nowy rekord do tabeli Products na podstawie argumentów.

```
CREATE PROCEDURE [dbo].[AddProduct]
   -- Add the parameters for the stored procedure here
   @CategoryID int,
   @ProductName varchar(64),
   @IsImportedFlag bit,
   @UnitPrice decimal(18,2),
   @ProductDescription varchar(255),
   @Active bit
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   INSERT INTO Products
   values (
      @CategoryID,
      @ProductName,
      @IsImportedFlag,
      @UnitPrice,
      @ProductDescription,
      @Active
   )
END
go
```

AddToMenu

Procedura dodaje produkt do menu zgodnie z podanymi argumentami. Jeśli dany produkt jest nieaktywny, to procedura zwraca wyjątek.

```
CREATE procedure AddToMenu @MenuID int, @ProductID int, @UnitPrice int
as
begin
begin try
    declare @active bit
    SELECT @active = active FROM Products where ProductID = @ProductID
    if (@active = 0)
        begin
            throw 50001, 'Produkt is not active',1
        end
    insert into MenuDetails (MenuID, ProductID, UnitPrice)
    values (@MenuID, @ProductID, @UnitPrice)
end try
begin catch

end catch
end
go
```

ChangeDiscountCoefficients

Procedura zmienia współczynniki rabatów zgodnie z podanymi argumentami.

```
CREATE PROCEDURE [dbo].[ChangeDiscountCoefficients]
   -- Add the parameters for the stored procedure here
   @Z1 int,
   @K1 decimal(18,2),
   @R1 float,
   @K2 decimal(18,2),
   @R2 float,
   @D1 int,
   @WZ decimal(18,2),
   @WK int
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE Constants
   SET Z1 = @Z1,
      K1 = @K1,
      R1 = @R1,
      K2 = @K2,
      R2 = @R2,
      D1 = @D1,
      WZ = @WZ,
      WK = @WK
END
go
```

ChangePriceInMenu

Procedura ustawia podaną cenę do podanej w argumentach pozycji w menu.

```
CREATE PROCEDURE [dbo].[ChangePriceInMenu]
   -- Add the parameters for the stored procedure here
   @MenuIDArg int,
   @ProductIDArg int,
   @NewPrice decimal(18,2)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE MenuDetails
   SET UnitPrice = @NewPrice
   WHERE MenuID = @MenuIDArg and ProductID = @ProductIDArg;
END
go
```

ChangePriceInProducts

Procedura ustawia podaną cenę do podanego w argumentach produktu.

```
CREATE PROCEDURE [dbo].[ChangePriceInProducts]
   -- Add the parameters for the stored procedure here
   @ProductID int,
   @NewPrice decimal(18,2)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE Products
   set UnitPrice = @NewPrice
   where ProductID = @ProductID
END
go
```

ChangeReservationEndtime

Procedura zmienia czas zakończenia rezerwacji.

```
CREATE PROCEDURE [dbo].[ModifyReservationEndtime]
   -- Add the parameters for the stored procedure here
   @ReservationID int,
   @NewEndtime datetime
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

    -- Insert statements for procedure here
   UPDATE Reservations
   set EndTime = @NewEndtime
   where ReservationID = @ReservationID
END
go
```

RemoveFromMenu

Procedura usuwa pozycję z menu na podstawie argumentów. Funkcja zwraca wyjątek jeśli podanej pary MenuID i ProductID nie ma w tabeli MenuDetails

```
CREATE procedure RemoveFromMenu @MenuID int, @ProductID int
as
begin try
    if not EXISTS(SELECT MenuID FROM MenuDetails WHERE MenuID = @MenuID and
ProductID = @ProductID)
        BEGIN
            throw 50001, 'Menu or products does not exists in database', 1

        END
    delete from MenuDetails where MenuID = @MenuID and ProductID =
@ProductID;
end try
begin catch

end catch
go
```

AddReservation

Procedura umożliwia dodanie rezerwacji. Dodaje ona krotkę do tabeli Reservations z odpowiednimi argumentami przekazanymi przez użytkownika. Następnie ustawia flagę Accepted na Null. Nie wiemy bowiem na razie nic o stolikach jakie będą zajmować goście oraz ich zamówieniach, przez co baza danych nie jest w stanie stwierdzić czy rezerwacja jest legalna według zasad panujących w restauracji. Wskaźnik ten zostanie zaktualizowany po uzupełnieniu zamówienia przez osobę składającą zamówienie.

```
alter procedure AddReservation(@ReservationID as int,
                               @ClientID as int,
                               @StartTime as datetime,
                               @EstimatedEndTime as datetime,
                               @NumberOfPeople as int)
as
    begin
        set nocount on
        begin try
            if ((select ClientID from Clients where ClientID = @ClientID)
is null)
                begin
                    ;throw 50001, N'The client with this ID does not exist
in the database',1
                end

            if (@EstimatedEndTime < @StartTime)
            begin
                ;throw 50001, N'Reservation ends earlier than it starts',1
            end

            if (@NumberOfPeople < 2)
            begin
                ;throw 50001, N'The number of guests should be bigger than
1', 1
            end

--          Check if Client is individual or company
        DECLARE @IsCompanyFlag bit;
        if exists(select * from CompanyClients where ClientID = @ClientID)
set @IsCompanyFlag = 1;
        else set @IsCompanyFlag = 0;

--         EstimaedEndTime elongate by 1 hour
        if (@ReservationID is not null)
            begin
        insert into Reservations(ReservationID, ClientID,StartTime,
EndTime, IsCompanyReservation, NumberOfPeople, Accepted)
        values (@ReservationID, @ClientID, @StartTime,
DATEADD(hour,1,@EstimatedEndTime),
                                        @IsCompanyFlag, @NumberOfPeople,
null)
            end

        end try

        begin catch
            Declare @msg NVARCHAR(2048) = N'Procedure error: '+
Error_Message();
            ;throw 50001, @msg, 1
```

```
        end catch
    end
```

AddOrder

Procedura dodaję zamówienie. Obsługuje ona wydarzenie, gdy klient składa zamówienie **bez wcześniejszej rezerwacji**. Może on zamówić na wynos, bądź zjeść na miejscu. Jeżeli składa zamówienie na wynos, wówczas nie jest rezerwowany dla niego żaden stolik. Jeżeli decyduje się na spożycie na miejscu wówczas potrzebuje dodatkowo informacji ile osób ,,składa się" na dane zamówienie, aby móc złożyć dla nich rezerwację. Dalsze dodawanie produktów oraz stolików do zamówienia/rezerwacji obsługuje już inna metoda.

*Na podstawie tego typu zamówienia możliwe jest uzyskanie zniżki permanentnej poprzez zamówienie Z1 zamówień za minimum K1 złotych

```
CREATE procedure AddOrder(@OrderID as int,
                          @ClientID as int = null,
                          @GuestNumber as int = null,
                          @TakeAwayTime as datetime = null,
                          @OTDiscount as int = null )
as
    begin
        Declare @ActualDiscount as float = 0;
    begin try
        if (@ClientID is not null and (select ClientID from Clients where
ClientID = @ClientID) is null)
                begin
                    ;throw 50001, N'The client with this ID does not exist
in the database',1
                end

        if (@TakeAwayTime is not null and @TakeAwayTime < getdate())
            begin
                ;throw 50001, N'the customer cannot set the pick-up date to
a date in the past', 1
            end

        if (@GuestNumber is not null and @GuestNumber <= 0)
            begin
                ;throw 50001, N'The number of guests should be bigger than
0', 1
            end

        if (@OTDiscount is not null)
            begin
                if not exists(select * from OneTimeDiscounts where
OTDiscountID = @OTDiscount

and ClientID = @ClientID

and used = 0)
                begin
                   ;throw 50001, N'Client does not have discount with this
ID', 1
                end

                if (getdate() > (select ExpireDate from OneTimeDiscounts
where OTDiscountID = @OTDiscount))
                begin
                    ;throw 50001, N'Client discount expire',1
                end
```

```
            end


        else
            begin
                set @ActualDiscount = isnull((select Value from
DiscountInfo where OTDiscountID = @OTDiscount),0)
                Exec ChangeDiscountStatus @OTDiscountID = @OTDiscount
            end

         if (select PermamentDiscountLVL from IndividualClients where
ClientID = @ClientID) = 1
        begin
            if (@ActualDiscount < (select max(R1) from Constants))
                set @ActualDiscount = (select max(R1) from Constants)
        end

    --     TakeAwayTime is not null means that client will not eat in
restaurant. Therefore it does not need a free table
    if (@TakeAwayTime is not null)
        begin
            insert into Orders(OrderID,ClientID, ReservationID, BillID,
OrderTime, TakeawayTime, AppliedDiscount, OTDiscountID)
            values (@OrderID, @ClientID, null, null, getdate(),
@TakeAwayTime,@ActualDiscount, @OTDiscount)
        end

--     DataBase have to reservate table for the guest
    else
        begin

            Declare @StartTime datetime = getdate()
            Declare @EndTime datetime = dateadd(hour, 1, @StartTime)
            Declare @ReservationID int = (select max(ReservationID) + 1
from Reservations) -- Can be done better

            Exec AddReservation @ClientID = @ClientID, @StartTime =
@StartTime, @NumberOfPeople = @GuestNumber,
                @ReservationID = @ReservationID, @EstimatedEndTime =
@EndTime

            Update Reservations
            set Accepted = 1
            where ReservationID = @ReservationID

            insert into Orders(OrderID,ClientID, ReservationID, BillID,
OrderTime, TakeawayTime, AppliedDiscount, OTDiscountID)
            values (@OrderID, @ClientID, @ReservationID, null, getdate(),
@TakeAwayTime,@ActualDiscount, @OTDiscount)

        end
    end try
    begin catch
         Declare @msg NVARCHAR(2048) = N'Procedure error: '+
Error_Message();
            ;throw 50001, @msg, 1
    end catch
    end
go
```

AddOrderToReservation

Dodaje zamówienie do rezerwacji. Rezerwacja niekoniecznie musi spełniać odpowiednie wymagania. Sprawdzane jest to dopiero po uzyskaniu szczegółowych informacji dotyczących całego zamówienia.

```
CREATE procedure  AddOrderToReservation(@OrderID int,
                                        @ClientID int,
                                        @ReservationID int,
                                        @BillID int = null,
                                        @OTDiscount int = null)
as
    begin
        set nocount on
        begin try

            Declare @AppliedDiscount float = 0;

            if (@ClientID is not null and (select ClientID from Clients
where ClientID = @ClientID) is null)
                begin
                    ;throw 50001, N'The client with this ID does not exist
in the database',1
                end

            if (not exists(select * from Reservations where ReservationID =
@ReservationID))
                begin
                    ;throw 50001, N'There is no reservation with this ID in
the database', 1
                end

            if (@OTDiscount is not null)
            begin
                if not exists(select * from OneTimeDiscounts where
OTDiscountID = @OTDiscount

and ClientID = @ClientID

and used = 0)
                begin
                   ;throw 50001, N'Client does not have discount with this
ID', 1
                end

                if (getdate() > (select ExpireDate from OneTimeDiscounts
where OTDiscountID = @OTDiscount))
                begin
                    ;throw 50001, N'Client discount expire',1
                end

            end

            else
                set @AppliedDiscount = isnull((select max(Value) from
OneTimeDiscounts where OTDiscountID = @OTDiscount

and ClientID = @ClientID

and used = 0),0)
                Exec ChangeDiscountStatus @OTDiscount
```

```
            if (select PermamentDiscountLVL from IndividualClients where
ClientID = @ClientID) = 1
                begin
                    if (@AppliedDiscount < (select max(R1) from Constants))
                        set @AppliedDiscount = (select max(R1) from
Constants)
                end

            insert into Orders(OrderID,ClientID, ReservationID, BillID,
OrderTime, TakeawayTime, AppliedDiscount, OTDiscountID)
            values (@OrderID, @ClientID, @ReservationID, @BillID,
getdate(), null, @AppliedDiscount, @OTDiscount)

        end try
        begin catch
            Declare @msg NVARCHAR(2048) = N'Procedure AddOrderToReservation
error: '+ Error_Message();
            ;throw 50001, @msg, 1
        end catch
    end
go
```

AddTableToReservation

Procedura dodaje stolik do danej rezerwacji.

```
alter procedure AddTableToReservation(@ReservationID as int,
                                      @TableID as int)
as
    begin
        set nocount on
        begin try
            if (not exists(select * from Reservations where ReservationID =
@ReservationID))
                begin
                    ;throw 50001, N'There is no reservation with this ID in
the database',1
                end

            if (not exists(select * from Tables where TableID = @TableID))
                begin
                    ;throw 50001, N'There is no table with this ID in the
database',1
                end

        insert into ReservationDetails(ReservationID, TableID)
        values (@ReservationID, @TableID)

        end try
        begin catch
             Declare @msg NVARCHAR(2048) = N'Procedure error: '+
Error_Message();
            ;throw 50001, @msg, 1
        end catch

    end
```

AddCompanyClient

Procedura dodaje klienta (firmę) do bazy danych

```
create procedure AddCompanyClient(@CompanyName varchar(64),
                                  @NIP varchar(16),
                                  @City varchar(64),
                                  @Street varchar(64),
                                  @PostalCode varchar(16),
                                  @PhoneNumber varchar(16),
                                  @Email varchar(64))
as
    begin
        set nocount on

        insert into Clients default values
        Declare @ClientID int = SCOPE_IDENTITY()

        insert into CompanyClients(ClientID, CompanyName, NIP, City,
Street, PostalCode, PhoneNumber, Email)
        values (@ClientID, @CompanyName, @NIP, @City, @Street, @PostalCode,
@PhoneNumber, @Email)

    end
```

AddIndividualClient

Procedura dodaje klienta (indywidualnego) do bazy danych

```
alter procedure AddIndividualClient(@Firstname varchar(64),
                                    @Lastname varchar(64),
                                    @PhoneNumber varchar(16),
                                    @Email varchar(64))
as
    begin
        set nocount on
        insert into Clients default values

        DECLARE @ClientId int = SCOPE_IDENTITY();
        insert into IndividualClients(ClientID, FirstName, LastName,
PhoneNumber, Email, PermamentDiscountLVL)
        values (@ClientID, @Firstname, @Lastname,@PhoneNumber,@Email, 0)

    end
```

AddProductToOrder

Procedura dodaje produkt do istniejącego zamówienia

```
alter procedure AddProductToOrder(@Order_ID int,
                                  @Product_ID int,
                                  @Quantity float)
as
    begin
        set nocount on
        begin try

            Declare @MenuDate datetime;
            if not exists(select * from Orders where OrderID = @Order_ID)
                begin
                    ;throw 50001, N'There is no order with this ID in the
database', 1
                end

            if not exists(select * from Products where ProductID =
@Product_ID)
                begin
                    ;throw 50001, N'There is no product with this ID in the
database', 1
                end

            Declare @TakeAwayTime datetime = (select TakeawayTime from
Orders where OrderID = @Order_ID)
            if (@TakeAwayTime is not null) set @MenuDate = @TakeAwayTime;


            else
                set @MenuDate = (select max(R2.StartTime) from Orders join
Reservations R2 on R2.ReservationID = Orders.ReservationID where OrderID =
@Order_ID)

            if  (dbo.CheckProductInMenu(@Product_ID, @MenuDate)) = 0
                begin
                    ;throw 50001, N'There is no selected item in the
current menu (for this day)',1
                end

            Declare @Price decimal(18,2) = (select UnitPrice from Products
where ProductID = @Product_ID)
            if exists(select * from OrderDetails where OrderID = @Order_ID
and ProductID = @Product_ID)
                begin
                    update OrderDetails
                    set Quantity = (select max(Quantity) from OrderDetails
where OrderID = @Order_ID and ProductID = @Product_ID) +
                                    @Quantity
                    where OrderID = @Order_ID and ProductID = @Product_ID
                end
            else
                begin
                    insert into OrderDetails(OrderID, ProductID,
PlannedServeTime, Served, Quantity, UnitPrice)
                    values(@Order_ID, @Product_ID, @MenuDate, 0, @Quantity,
@Price)
                end
        end try
```

```
        begin catch
             Declare @msg NVARCHAR(2048) = N'Procedure AddProductToOrder
error: '+ Error_Message();
            ;throw 50001, @msg, 1
        end catch
    end
```

AddMenu

Procedura dodaje nowe Menu do tabeli Menus

```
CREATE procedure AddMenu(@StartTime as datetime,
                         @EndTime as datetime)
as
    begin try
        if @StartTime > @EndTime
            begin
                ;throw 50001, N'Menu ends before it start', 1
            end

        insert into Menus(starttime, endtime, IsLegal)
        values (@StartTime, @EndTime, 0)

    end try
    begin catch
         Declare @msg NVARCHAR(2048) = N'Procedure AddMenu error: '+
Error_Message();
            ;throw 50001, @msg, 1
    end catch
go
```

AddPayment

Procedura umożliwia dodanie płatności za spożyty posiłek (Parametr Percentage określa jaki procent z całkowitej wartości zamówienia klient chce TERAZ zapłacić)

```
create procedure AddPayment(@Order_ID int,
                            @PayDate datetime,
                            @Percentage float)
as
    begin
        Declare @amount float = (select sum(Od.UnitPrice*od.Quantity)
                                 from Orders O join OrderDetails OD on
O.OrderID = OD.OrderID
                                 where O.OrderID = @Order_ID) * @Percentage
        DECLARE @PaymentID int = (select count(*) from Payments) + 1
        insert into Payments values (@PaymentID, @Order_ID, @amount, null,
@PayDate)
    end
```

AddReservationGuest

Procedura dodaje gości firmowych do złożonej rezerwacji (firmowej)

```
create procedure AddReservationGuest(@Reservation_Guest_ID as int,
                                     @Reservation_ID as int,
                                     @First_Name as varchar(20),
                                     @Last_Name as varchar(20)
)
as
    begin
--  Checking if the guest is not already registered
        if EXISTS(select * from ReservationGuestList
                      where ReservationGuestID = @Reservation_Guest_ID
                         and ReservationID = @Reservation_ID
                         and @First_Name like FirstName
                         and @Last_Name like LastName)
            begin;
                throw 50001, N'Miejsce dla tej osoby jest juz
zarezerwowane',1
            end

-- Inserting Guest

        insert into ReservationGuestList (ReservationGuestID,
ReservationID, FirstName, LastName)
                values (@Reservation_Guest_ID, @Reservation_ID, @First_Name
, @Last_Name);
    end
go;
```

CheckReservationCorrect

Procedura sprawdza czy rezerwacja, której dokonał klient (wraz z zamówieniem produktów) spełnia regulamin restauracji. Jeżeli nie – usuwa informację o nielegalnej rezerwacji(rezerwacje, zamówienie, produkty, stoliki)

```
create procedure CheckReservationCorrect(@ReservationID int)
as
    begin


        Declare @MainOrderID int = (select O.OrderID from Reservations R
join Orders O on R.ReservationID = O.ReservationID
                                            where R.ReservationID =
@ReservationID and R.ClientID = O.ClientID)

        Declare @TotalValue float =
dbo.CalculateOrderValueWithDiscounts(@MainOrderID)
        DECLARE @WZ decimal(18,2) = (select max(WZ) from Constants)

        --Check if Products Amount bigger than WZ
        if (@TotalValue < @WZ)
            begin
                Exec DeleteIncorrectReservation @ReservationID
            end

        Declare @ClientID int = (select ClientID from Reservations where
ReservationID = @ReservationID)
        Declare @OrderCnt int = dbo.GetOrdersCnt (@ClientID) - 1 --Without
this order
        Declare @WK int = (select max(WK) from Constants)

--          Check if number of previous Orders bigger than WK
        if (@OrderCnt < @WK)
        begin
            Exec DeleteIncorrectReservation @ReservationID
        end


        update Reservations
        set Accepted = 0
        where ReservationID = @ReservationID

    end
```

ChangeDiscountStatus

Procedura zmienia status zniżki na użytą, po wykorzystaniu przez klienta

```
create procedure ChangeDiscountStatus(@OTDiscountID as int)
as
    begin
        set nocount on
        update OneTimeDiscounts
        set used = 1
        where OTDiscountID = @OTDiscountID
    end
```

DeleteIncorrectReservation

Procedura usuwa wszystkie krotki z tablic Reservation, Orders, Orders Details, Reservation Details, które dotyczą danej rezerwacji.

```
alter procedure DeleteIncorrectReservation(@ReservationID int)
as
begin

    set nocount on

    delete from OrderDetails
    where OrderID in (select Orders.OrderID from Orders where ReservationID
= @ReservationID)

    delete from Orders
    where ReservationID = @ReservationID

    delete from ReservationDetails
    where ReservationID = @ReservationID

    delete from Reservations
    where ReservationID = @ReservationID


end
```

Funkcje

ClientOrderReportMon

Funkcja zwraca listę miesięcy w których klient składał zamówienia wraz z ich statystykami.

```
CREATE FUNCTION ClientOrderReportMon
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT [ClientID]
      ,[isCompanyClient]
      ,[Year]
      ,[Month]
      ,[OrdersSum]
      ,[TotalMoneySpend]
   FROM [u_twardosz].[dbo].[OrderReportMon]
   where ClientID = @ClientID
)
go
```

ClientOrderReportWeek

Funkcja zwraca listę tygodni w których klient coś zamawiał wraz ze statystykami zamówień.

```
CREATE FUNCTION ClientOrderReportWeek
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT [ClientID]
      ,[isCompanyClient]
      ,[Year]
      ,[Week]
      ,[OrdersSum]
      ,[TotalMoneySpend]
   FROM [u_twardosz].[dbo].[OrderReportWeek]
   where ClientID = @ClientID
)
go
```

ClientOrdersHistory

Funkcja zwraca listę przeszłych zamówień.

```
CREATE FUNCTION ClientOrdersHistory
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   select * FROM ClientToMeal WHERE ClientID = @ClientID
)
go
```

ClientReservationReportMon

Funkcja zwraca listę miesięcy w których klient miał rezerwację i statystyki ich dotyczące.

```
CREATE FUNCTION ClientReservationReportMon
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT [ClientID]
      ,[isCompanyClient]
      ,[Year]
      ,[Month]
      ,[OrdersSum]
      ,[TotalMoneySpend]
  FROM [u_twardosz].[dbo].[OrderReportMon]
  where ClientID = @ClientID
)
go
```

ClientReservationReportWeek

Funkcja zwraca listę tygodni w których klient miał rezerwację i statystyki ich dotyczące.

```
CREATE FUNCTION ClientReservationReportWeek
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT [ClientID]
      ,[isCompanyClient]
      ,[Year]
      ,[Week]
      ,[OrdersSum]
      ,[TotalMoneySpend]
  FROM [u_twardosz].[dbo].[OrderReportWeek]
  where ClientID = @ClientID
)
go
```

ViewClientDiscounts

Funkcja przyjmuje w argumencie ClientID i dla danego klienta wyświetla wszystkie jego przeszłe rabaty.

```
CREATE FUNCTION ViewClientDicounts
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT *
   from OneTimeDiscounts
   where ClientID = @ClientID
)
go
```

ViewClientUnusedDiscounts

Funkcja przyjmuje w argumencie ClientID i dla danego klienta wyświetla wszystkie jego niewykorzystane rabaty.

```
CREATE FUNCTION ViewClientUnusedDicounts
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT *
   from OneTimeDiscounts
   where ClientID = @ClientID and used = 0
)
go
```

ViewBillsOfAClient

Funkcja zwraca rachunki/faktury wybranego klienta

```
CREATE FUNCTION ViewBillsOfAClient
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT B.BillID, B.IsCollective, B.IssueDate from Bills B join Orders O
on O.BillID = B.BillID  where O.ClientID = @ClientID group by B.BillID,
B.IsCollective, B.IssueDate
)
```

ViewOrdersOfABill

Funkcja zwraca wszystkie informacje na temat zamówień przypisanych do danego rachunku/faktury

```
CREATE FUNCTION ViewOrdersOfABill
(
   -- Add the parameters for the function here
   @BillID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   select * from orders where BillID = @BillID
)
```

ReturnOrderWithoutBill

Funkcja zwraca wszystkie zamówienia, które nie mają przypisanego żadnego rachunku/faktury

```
CREATE FUNCTION [dbo].[ReturnOrdersWithoutBill]
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS
@OrdersWithNoBill TABLE
(
   -- Add the column definitions for the TABLE variable here
   OrderID int
)
AS
BEGIN
   -- Fill the table variable with the rows for your result set

   insert into @OrdersWithNoBill
   select OrderID
   from Orders
   where ClientID = @ClientID and BillID = null

   return
END
```

CheckProductInMenu

Funkcja sprawdza czy w menu na dany okres czasu znajduje się zamawiany produkt

```
create function CheckProductInMenu(@ProductId as int,
                                   @Date as date)
returns bit
as begin
    if @ProductId in (select FM.ProductID from FutureMenus FM where @Date
between FM.StartTime and FM.EndTime) return 1
    return 0
end
```

ChangeMenuStatus

Zmienia status Menu na aktywny

```
create procedure ChangeMenuStatus(@MenuID as int,
                                  @ActualStatus as bit)
as
    begin try
        if @MenuID not in (select MenuID from Menus)
        begin
            ;throw 50001, N'There is not Menu with this ID in database', 1
        end

        if @ActualStatus = 1 and dbo.CheckMenuRequirements(@MenuID) = 0
        begin
            print N'Warning ! Changing the status of the illegal Menu to
the correct one'
        end

    update Menus
    set IsLegal = @ActualStatus
    where MenuID = @MenuID

    end try
    begin catch
        Declare @msg NVARCHAR(2048) = N'Procedure ChangeMenuStatus error:
'+ Error_Message();
            ;throw 50001, @msg, 1
    end catch
go
```

FindFreeTable

Funkcja zwraca tablicę wszystkich wolnych stolików, które pomieszczą przynajmniej @MinSize klientów oraz nie są zajęte w okresie czasu [@StartTime, @EndTime] .

```
CREATE function FindFreeTable(@MinSize as int,
                              @StartTime as datetime,
                              @EndTime as datetime)
returns table
    return (
        select TableID, size from Tables
        where @MinSize <= Tables.Size
        except
        select distinct TableID, size from TableOccupancy
    where @StartTime  between StartTime and EndTime and
          @EndTime  between StartTime and EndTime and
          @MinSize <= Size)
go
```

GetOrdersCnt

Funkcja zwraca ilość zamówień jakie wykonał w przeszłości klient o zadanym ID

```
create function GetOrdersCnt(@ClientID as int)
returns int
as
begin
    return (select max(OrderCounter) from OrdersCounter where ClientID =
@ClientID)
end
```

GetMenu

Funkcja zwraca Menu dla podanej daty

```
create function GetMenu(@Date as datetime)
returns table
as
    return select * from FutureMenus where @Date between StartTime and
EndTime
go
```

GetOrderValue

Funkcja zwraca wartość danego zamówienia (na podstawie jego ID)

```
create function GetOrderValue(@OrderID as int)
returns decimal(18,2)
as
    begin
    Declare @sum decimal(18,2) = (select sum(UnitPrice *
OrderDetails.Quantity)*(1-max(O.AppliedDiscount)) from OrderDetails
                                                                join
Orders O on O.OrderID = OrderDetails.OrderID
                                                                where
OrderDetails.OrderID = @OrderID)
    return isnull(@sum, 0)
    end
go
```

GetOrderID

Funkcja zwraca największy klucz główny z tabeli Orders

```
CREATE function GetOrderID()
returns int
as
    begin
        Declare @MaxOrderID int = (select max(OrderID) from Orders)
        return @MaxOrderID
    end
go
```

GetMenuID

Funkcja zwraca największy klucz główny z tabeli Menus

```
create function GetMenuID()
returns int
as
    begin
        Declare @MaxMenuID int = (select max(MenuID) from Menus)
        return @MaxMenuID + 1
    end
go
```

GetReservationID

Funkcja zwraca największy klucz główny z tabeli Reservations

```
CREATE function GetReservationID()
returns int
as
    begin
        Declare @MaxReservationID int = (select max(ReservationID) from
Reservations)
        return @MaxReservationID
    end
go
```

IsTableFree

Funkcja sprawdza czy wybrany stolik nie jest zarezerwowany w danym czasie

```
CREATE function IsTableFree(@TableID as int,
                            @StarDate as datetime,
                            @EndDate as datetime)
returns bit
as
begin
    if exists(select * from TableOccupancy where TableID = @TableID and
((@StarDate  between StartTime and EndTime) or
                                                (@EndDate  between
StartTime and EndTime))) return 0
    return 1
end
go
```

OrderPaySum

Funkcja sprawdza ile pieniędzy zostało zapłacone dla każdego zamówienia

```
CREATE FUNCTION OrderPaySum
(
   -- Add the parameters for the function here
   @OrderID int
)
RETURNS decimal(18,2)
AS
BEGIN
   RETURN (select sum(P.Amount) from Payments P where OrderID = @OrderID
group by OrderID)

END
```

IsOrderPaid

Funkcja zwraca informację czy zamówienie zostało opłacone

```
CREATE FUNCTION [dbo].[IsOrderPaid]
(
   -- Add the parameters for the function here
   @OrderID int
)
RETURNS bit
AS
BEGIN
   -- Declare the return variable here
   DECLARE @PaySum decimal(18,2) = (select sum(P.Amount) from Payments P
where OrderID = @OrderID group by OrderID)
   DECLARE @OrderValue decimal(18,2) = (select
sum(OD.Quantity*OD.UnitPrice)*(1-O.AppliedDiscount) from Orders O join
OrderDetails OD on O.OrderID = OD.OrderID where O.OrderID = @OrderID group
by O.OrderID, O.AppliedDiscount)

   -- Add the T-SQL statements to compute the return value here


   -- Return the result of the function
   RETURN convert(Bit, Case When @OrderValue = @PaySum Then 1 Else 0 End)

END
```

ViewUnpaidOrdersOfAClient

Funkcja zwraca tablice nieopłaconych zam

```
CREATE FUNCTION [dbo].[ViewUnpaidOrdersOfAClient]
(
   -- Add the parameters for the function here
   @ClientID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT * from Orders O where ([dbo].[IsOrderPaid](O.OrderID)) = 0 and
ClientID = @ClientID
)
```

ViewOrdersOfPayment

Funkcja zwraca zamówienia dla których została dokonana już wpłata

```
CREATE FUNCTION ViewOrderOfPayment
(
   -- Add the parameters for the function here
   @PaymentID int
)
RETURNS TABLE
AS
RETURN
(
   -- Add the SELECT statement with parameter references here
   SELECT * from Orders where OrderID = (select OrderID from Payments where
PaymentID = @PaymentID)
)
```

Triggery

DeleteDiscount

W przypadku usunięcia zamówienia usuwa wszystkie rabaty dodane w tym zamówieniu

```
CREATE TRIGGER DeleteDiscounts
    ON Orders
    AFTER DELETE
    AS
BEGIN
    DECLARE @Discounts int
    DECLARE @OrderID int
    SELECT @OrderID = OrderID FROM deleted
    SELECT @Discounts = count(*) FROM OneTimeDiscounts WHERE OrderID =
@OrderID
    IF (@Discounts > 0)
    BEGIN
        DELETE FROM OneTimeDiscounts WHERE OrderID=@OrderID
    END
END
go
```

CheckRequirementsForOneTimeDiscount

Sprawdza wymagania do jednorazowego rabatu i w przypadku ich spełnienia dodaje rabat

```
CREATE TRIGGER CheckRequirementsForOneTimeDiscount
    ON OrderDetails
    AFTER INSERT, UPDATE
    AS
BEGIN
    DECLARE @ClientID INT
    DECLARE @OrderDate DATE
    DECLARE @OrderID INT
    SELECT @OrderID = OrderID FROM inserted
    SELECT @ClientID = o.ClientID, @OrderDate = o.TakeawayTime FROM Orders
o WHERE o.OrderID = @OrderID
    IF (@OrderDate IS NULL)
    BEGIN
        SELECT @OrderDate = StartTime
        FROM Reservations r
        JOIN Orders O2 on r.ReservationID = O2.ReservationID
        WHERE O2.OrderID = @OrderID
    END
    IF EXISTS(SELECT * FROM IndividualClients WHERE ClientID = @ClientID)
    BEGIN
        DECLARE @LastDiscount date
        DECLARE @K2 DECIMAL(18,2)
        DECLARE @D1 INT
        DECLARE @R2 DECIMAL(18,2)
        DECLARE @CurrentSum DECIMAL(18,2)
        SELECT @K2 = K2, @D1 = D1, @R2 = R2 FROM Constants
        SELECT TOP 1 @LastDiscount = ApplyDate FROM OneTimeDiscounts WHERE
ClientID = @ClientID ORDER BY ApplyDate DESC
        IF (@LastDiscount IS NOT NULL)
        BEGIN
            SELECT @CurrentSum = SUM(OD.UnitPrice*OD.Quantity)
            FROM Orders o
            JOIN OrderDetails OD on o.OrderID = OD.OrderID
            WHERE o.ClientID = @ClientID AND o.OrderTime > @LastDiscount
        END
        ELSE
        BEGIN
            SELECT @CurrentSum = SUM(OD.UnitPrice*OD.Quantity)
            FROM Orders o
            JOIN OrderDetails OD on o.OrderID = OD.OrderID
            WHERE o.ClientID = @ClientID
        END
        IF (@CurrentSum >= @K2)
        BEGIN
            INSERT INTO OneTimeDiscounts (ClientID, used, ApplyDate,
ExpireDate, Value, OrderID)
            VALUES (@ClientID, 0, @OrderDate, DATEADD(day, @D1,@OrderDate),
@R2, @OrderID)
        END
    END
END
go
```

CheckRequirementsForPermanentDiscount

Sprawdza wymagania do permanentnego rabatu i w przypadku ich spełnienia dodaje rabat klientowi

```
CREATE TRIGGER CheckRequirementsForPermanentDiscount
    ON OrderDetails
    AFTER INSERT, UPDATE
    AS
BEGIN
    declare @ClientID int
    SELECT @ClientID = o.ClientID FROM inserted JOIN Orders o ON o.OrderID
= inserted.OrderID
    if EXISTS(SELECT * FROM IndividualClients ID WHERE ID.ClientID =
@ClientID)
        begin
            declare @CurrentLevel int
            SELECT @CurrentLevel = PermamentDiscountLVL FROM
IndividualClients WHERE ClientID = @ClientID
            if (@CurrentLevel = 0)
                begin
                    declare @Z1 int
                    declare @K1 decimal(18, 2)
                    declare @Count int
                    SELECT @Z1 = Z1, @K1 = K1 FROM Constants
                    SELECT @Count = count(distinct o.OrderID)
                    FROM Orders o
                             JOIN OrderDetails OD on o.OrderID = OD.OrderID
                    WHERE ClientID = @ClientID
                    HAVING SUM(OD.UnitPrice * OD.Quantity) >= @K1
                    if (@Count >= @Z1)
                        begin
                            UPDATE IndividualClients
                            SET PermamentDiscountLVL = 1
                            WHERE ClientID = @ClientID
                        end
                end
        end
END
go
```

CheckRequirementsForPermanentDiscountOnDelete

W przypadku usunięcia zamówienia lub jego elementów sprawdza czy klient dalej spełnia warunki na permanentny rabat i jeżeli ich nie spełnia to odbiera ten rabat

```
CREATE TRIGGER CheckRequirementsForPermanentDiscountOnDelete
    ON OrderDetails
    AFTER DELETE
    AS
BEGIN
    declare @ClientID int
    SELECT @ClientID = o.ClientID FROM deleted JOIN Orders o ON o.OrderID =
deleted.OrderID
    if EXISTS(SELECT * FROM IndividualClients ID WHERE ID.ClientID =
@ClientID)
        begin
            declare @CurrentLevel int
            SELECT @CurrentLevel = PermamentDiscountLVL FROM
IndividualClients WHERE ClientID = @ClientID
            if (@CurrentLevel = 1)
                begin
                    declare @Z1 int
                    declare @K1 decimal(18, 2)
                    declare @Count int
                    SELECT @Z1 = Z1, @K1 = K1 FROM Constants
                    SELECT @Count = count(distinct o.OrderID)
                    FROM Orders o
                             JOIN OrderDetails OD on o.OrderID = OD.OrderID
                    WHERE ClientID = @ClientID
                    HAVING SUM(OD.UnitPrice * OD.Quantity) >= @K1
                    if (@Count < @Z1)
                        begin
                            UPDATE IndividualClients
                            SET PermamentDiscountLVL = 0
                            WHERE ClientID = @ClientID
                        end
                end
        end
END
go
```

Indeksy

IX_OneTimeDiscount

Indeks dla tabeli OneTimeDiscount i kolumny ClientID

```
create index IX_OneTimeDiscounts
    on OneTimeDiscounts (ClientID)
go
```

IX_OneTimeDiscount

Indeks dla tabeli OneTimeDiscount i kolumny OrderID

```
create index IX_OneTimeDiscounts_1
    on OneTimeDiscounts (OrderID)
go
```

IX_Orders

Indeks dla tabeli Orders i kolumny ClientID

```
create index IX_Orders
    on Orders (ClientID)
go
```

IX_Orders_1

Indeks dla tabeli Orders i kolumny ReservationID

```
create index IX_Orders_1
    on Orders (ReservationID)
go
```

IX_Orders_2

Indeks dla tabeli Orders i kolumny BillID

```
create index IX_Orders_2
    on Orders (BillID)
go
```

IX_Payments

Indeks dla tabeli Payments i kolumy OrderID

```
create index IX_Payments
    on Payments (OrderID)
go
```

IX_Products

Indeks dla tabeli Products i kolumny CategoryID

```
create index IX_Products
    on Products (CategoryID)
go
```

IX_ReservationGuestList

Indeks dla tabeli ReservationGuestList i kolumny ReservationID

```
create index IX_ReservationGuestList
    on ReservationGuestList (ReservationID)
go
```

IX_Reservations

Indeks dla tabeli Reservations i kolumny ClientID

```
create index IX_Reservations
    on Reservations (ClientID)
go
```

Indeksy dla kluczy głównych

Do każdej kolumny która jest kluczem głównym w danej relacji, indeks został wygenerowany automatycznie.