

# Projekt bazy danych restauracji

## Spis zawartości:

1. Funkcje i uprawnienia
2. Schemat bazy danych
3. Kod bazy danych i warunki integralności
4. Złożone warunki integralności
5. Widoki i funkcje
6. Procedury

## Autorzy:

- Kamil Miśkowiec,
- Łukasz Dydek,
- Szymon Żychowicz

## **Funkcje i uprawnienia**

### **Manager:**

- generowanie statystyk i raportów tygodniowych oraz miesięcznych - z podziałem na typ klienta (rezerwacje stolików, rabaty, menu, zamówienia i czas ich składania),
- przygotowywanie menu,
- zarządzanie daniami,
- zarządzanie zniżkami,
- zarządzanie pracownikami,
- to co zwykły pracownik;

### **Pracownik:**

- akceptowanie rezerwacji,
- przyznanie stolika dla rezerwacji,
- potwierdzenie anulowania rezerwacji,
- przyjmowanie zamówienia na miejscu i na wynos,
- zmiana lub anulowanie zamówienia,
- wystawienie faktury po zamówieniu lub zbiorczej za okres miesiąca,
- przeglądanie rezerwacji,
- przeglądanie menu,
- przeglądanie zamówień,
- przeglądanie dostępności stolików,
- przeglądanie zniżek,
- generowanie raportu dla klienta indywidualnego lub firmy (rezerwacje, zamówienia, zniżki)

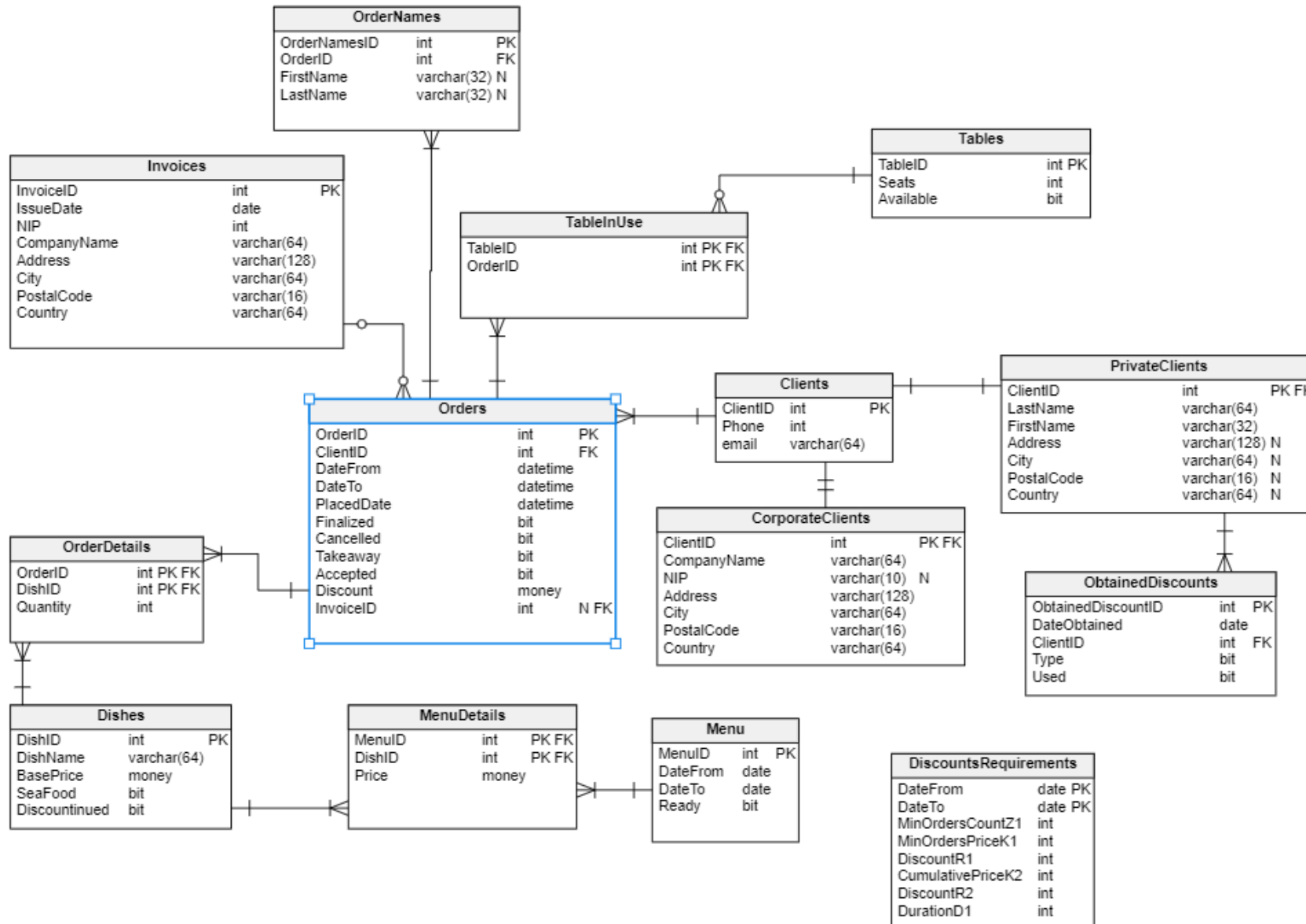
### **Klient indywidualny:**

- przeglądanie menu,
- złożenie zamówienia,
- złożenie zamówienia na owoce morza,
- rezerwacja stolika składając zamówienie,
- generowanie raportu o swoich rezerwacjach, zamówieniach, zniżkach;

### **Firma:**

- przeglądanie menu,
- złożenie zamówienia,
- złożenie zamówienia na owoce morza,
- rezerwacja stolików na firmę lub na konkretnych pracowników,
- generowanie raportu o swoich rezerwacjach, zamówieniach, zniżkach (z podziałem na pracowników);

## Schemat bazy danych



## Kod bazy danych i warunki integralności

```
-- tables
-- Table: Clients
CREATE TABLE Clients (
  ClientID int NOT NULL IDENTITY(1,1),
  Phone int NOT NULL,
  email varchar(64) NOT NULL,
  CONSTRAINT Clients_pk PRIMARY KEY (ClientID)
);

-- Table: CorporateClients
CREATE TABLE CorporateClients (
  ClientID int NOT NULL,
  CompanyName varchar(64) NOT NULL,
  NIP varchar(10) NULL,
  Address varchar(128) NOT NULL,
  City varchar(64) NOT NULL,
  PostalCode varchar(16) NOT NULL,
  Country varchar(64) NOT NULL,
  CONSTRAINT NIP UNIQUE (NIP),
  CONSTRAINT CorporateClients_pk PRIMARY KEY (ClientID)
);

-- Table: DiscountsRequirements
CREATE TABLE DiscountsRequirements (
  DateFrom date NOT NULL,
  DateTo date NOT NULL,
  MinOrdersCountZ1 int NOT NULL,
  MinOrdersPriceK1 int NOT NULL,
  DiscountR1 int NOT NULL,
  CumulativePriceK2 int NOT NULL,
  DiscountR2 int NOT NULL,
  DurationD1 int NOT NULL,
  CONSTRAINT DiscountsRequirements_pk PRIMARY KEY (DateFrom,DateTo)
);
```

```

-- Table: Dishes
CREATE TABLE Dishes (
    DishID int NOT NULL IDENTITY(1,1),
    DishName varchar(64) NOT NULL,
    BasePrice money NOT NULL, --minimalna cena, za którą opłaca się danie sprzedawać
    SeaFood bit NOT NULL,
    Discountinued bit NOT NULL,
    CONSTRAINT Dishes_pk PRIMARY KEY (DishID)
);

-- Table: Invoices
CREATE TABLE Invoices (
    InvoiceID int NOT NULL IDENTITY(1,1),
    IssueDate date NOT NULL,
    NIP int NOT NULL,
    CompanyName varchar(64) NOT NULL,
    Address varchar(128) NOT NULL,
    City varchar(64) NOT NULL,
    PostalCode varchar(16) NOT NULL,
    Country varchar(64) NOT NULL,
    CONSTRAINT Invoices_pk PRIMARY KEY (InvoiceID)
);

-- Table: Menu
CREATE TABLE Menu (
    MenuID int NOT NULL IDENTITY(1,1),
    DateFrom date NOT NULL,
    DateTo date NOT NULL,
    Ready bit NOT NULL,
    CONSTRAINT Menu_pk PRIMARY KEY (MenuID)
);

-- Table: MenuDetails
CREATE TABLE MenuDetails (
    MenuID int NOT NULL,
    DishID int NOT NULL,
    Price money NOT NULL,
    CONSTRAINT MenuDetails_pk PRIMARY KEY (MenuID,DishID)
);

```

```

-- Table: ObtainedDiscounts
CREATE TABLE ObtainedDiscounts (
    ObtainedDiscountID int NOT NULL IDENTITY(1,1),
    DateObtained date NOT NULL,
    ClientID int NOT NULL,
    Type bit NOT NULL,
    Used bit NOT NULL,
    CONSTRAINT ObtainedDiscounts_pk PRIMARY KEY (ObtainedDiscountID)
);

-- Table: OrderDetails
CREATE TABLE OrderDetails (
    OrderID int NOT NULL,
    DishID int NOT NULL,
    Quantity int NOT NULL,
    CONSTRAINT OrderDetails_pk PRIMARY KEY (OrderID,DishID)
);

-- Table: OrderNames
CREATE TABLE OrderNames (
    OrderNamesID int NOT NULL IDENTITY(1,1),
    OrderID int NOT NULL,
    FirstName varchar(32) NULL,
    LastName varchar(32) NULL,
    CONSTRAINT OrderNames_pk PRIMARY KEY (OrderNamesID)
);

-- Table: Orders
CREATE TABLE Orders (
    OrderID int NOT NULL IDENTITY(1,1),
    ClientID int NOT NULL,
    DateFrom datetime NOT NULL,
    DateTo datetime NOT NULL,
    PlacedDate datetime NOT NULL,
    Finalized bit NOT NULL,
    Cancelled bit NOT NULL,
    Takeaway bit NOT NULL,
    Accepted bit NOT NULL,
    Discount money NOT NULL,
    InvoiceID int NULL,
    CONSTRAINT Orders_pk PRIMARY KEY (OrderID)
);

```

```

-- Table: PrivateClients
CREATE TABLE PrivateClients (
  ClientID int NOT NULL,
  LastName varchar(64) NOT NULL,
  FirstName varchar(32) NOT NULL,
  Address varchar(128) NULL,
  City varchar(64) NULL,
  PostalCode varchar(16) NULL,
  Country varchar(64) NULL,
  CONSTRAINT PrivateClients_pk PRIMARY KEY (ClientID)
);

-- Table: TableInUse
CREATE TABLE TableInUse (
  TableID int NOT NULL,
  OrderID int NOT NULL,
  CONSTRAINT TableInUse_pk PRIMARY KEY (TableID,OrderID)
);

-- Table: Tables
CREATE TABLE Tables (
  TableID int NOT NULL IDENTITY(1,1),
  Seats int NOT NULL,
  Available bit NOT NULL,
  CONSTRAINT Tables_pk PRIMARY KEY (TableID)
);

-- foreign keys
-- Reference: CorporateClients_Clients (table: CorporateClients)
ALTER TABLE CorporateClients ADD CONSTRAINT CorporateClients_Clients
  FOREIGN KEY (ClientID)
  REFERENCES Clients (ClientID);

-- Reference: MenuDetails_Dishes (table: MenuDetails)
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Dishes
  FOREIGN KEY (DishID)
  REFERENCES Dishes (DishID);

-- Reference: MenuDetails_Menu (table: MenuDetails)
ALTER TABLE MenuDetails ADD CONSTRAINT MenuDetails_Menu
  FOREIGN KEY (MenuID)
  REFERENCES Menu (MenuID);

```

```

-- Reference: OrderDetails_Dishes (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Dishes
    FOREIGN KEY (DishID)
    REFERENCES Dishes (DishID);

-- Reference: OrderDetails_Orders (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);

-- Reference: OrderNames_Orders (table: OrderNames)
ALTER TABLE OrderNames ADD CONSTRAINT OrderNames_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);

-- Reference: Orders_Clients (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Clients
    FOREIGN KEY (ClientID)
    REFERENCES Clients (ClientID);

-- Reference: Orders_Invoices (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Invoices
    FOREIGN KEY (InvoiceID)
    REFERENCES Invoices (InvoiceID);

-- Reference: PrivateClients_Clients (table: PrivateClients)
ALTER TABLE PrivateClients ADD CONSTRAINT PrivateClients_Clients
    FOREIGN KEY (ClientID)
    REFERENCES Clients (ClientID);

-- Reference: PrivateClients_Discounts (table: ObtainedDiscounts)
ALTER TABLE ObtainedDiscounts ADD CONSTRAINT PrivateClients_Discounts
    FOREIGN KEY (ClientID)
    REFERENCES PrivateClients (ClientID);

-- Reference: TableInUse_Orders (table: TableInUse)
ALTER TABLE TableInUse ADD CONSTRAINT TableInUse_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);

-- Reference: Tables_TableInUse (table: TableInUse)
ALTER TABLE TableInUse ADD CONSTRAINT Tables_TableInUse
    FOREIGN KEY (TableID)
    REFERENCES Tables (TableID);

```



```

-- Constraints
ALTER TABLE Menu
ADD CONSTRAINT MenuDateCheck CHECK (
    DateTo >= DateFrom
)

ALTER TABLE OrderDetails
ADD CONSTRAINT OrderDetailsQuantityCheck CHECK (
    Quantity > 0
)

ALTER TABLE Orders
ADD CONSTRAINT OrdersDateCheck CHECK (
    DateFrom >= PlacedDate and DateTo > DateFrom
)

ALTER TABLE Tables
ADD CONSTRAINT TablesNumberOfSeatsCheck CHECK (
    Seats > 0
)

ALTER TABLE MenuDetails
ADD CONSTRAINT MenuDetailsPricePositiveCheck CHECK (
    Price > 0
)

ALTER TABLE Dishes
ADD CONSTRAINT DishesBasePricePositiveCheck CHECK (
    BasePrice > 0
)

ALTER TABLE Dishes
ADD OrdersMoneyNotNegativeCheck CHECK (
    Discounts >= 0
)

ALTER TABLE Invoices
ADD InvoicesNIPPositiveCheck CHECK (
    NIP > 0
)

ALTER TABLE Clients
ADD ClientsPhoneNumberPositiveCheck CHECK (
    Phone > 0
)

```

```

ALTER TABLE CorporateClients
ADD CorporateClientsNIPNotNegativeCheck CHECK (
    NIP > 0
)

```

### Złożone warunki integralności

```

-- complex constraint: Danie w zamówieniu występuje w menu dla daty zamówienia
ALTER TABLE OrderDetails
ADD CONSTRAINT DishInOrderInMenuCheck CHECK (
    dbo.DishInOrderInMenu(DishID, OrderID) = 0
)

CREATE FUNCTION DishInOrderInMenu (@DishID int, @OrderID int) RETURNS bit AS
BEGIN
    if exists
        (select 1
         from Orders as o
         join MenuDetails as md
         on md.DishID = @DishID
         join Menu as m
         on m.MenuID = md.MenuID
         where o.OrderID = @OrderID and (md.DateFrom between DateFrom and DateTo)
        )
        RETURN 1
    RETURN 0
END

```

```

-- complex constraint: nie możemy dokonać zamówienia do stolika
-- zajętego bądź zepsutego
ALTER TABLE Orders
ADD CONSTRAINT FreeTable
CHECK (dbo.FreeTable(OrderID) = 1)

CREATE FUNCTION FreeTableCheck (
    @OrderID int
)
RETURNS bit
AS
BEGIN
    IF exists (select * from Orders
        inner join TableInUse on Orders.OrderID = TableInUse.OrderID
        inner join Tables on Tables.TableID = TableInUse.TableID
        where Tables.Available = 0 and Orders.OrderID = @OrderID)
        RETURN 0
    RETURN 1
END

-- complex constraint: ceny dań w menu większe bądź równe do cen bazowych tych dań
ALTER TABLE MenuDetails
ADD CONSTRAINT menuPriceHigherThanBase
CHECK (dbo.BasePriceDiff(DishID) = 1)

CREATE FUNCTION BasePriceDiff(
    @DishID int
)
RETURNS bit
AS
BEGIN
    IF (select Price-BasePrice
        from Dishes join MenuDetails on Dishes.DishID=MenuDetails.DishID
        where Dishes.DishID=@DishID )>=0 --dania nie opłaca się wówczas sprzedawać
        RETURN 1
    RETURN 0
END

```

```

-- complex constraint: nie możemy dokonać zamówienia, które bierze
-- danie z niegotowego menu
ALTER TABLE Orders
ADD CONSTRAINT OrderFromReadyMenu
CHECK (dbo.OrderFromReadyMenuCheck(OrderID) = 1)

CREATE FUNCTION OrderDetailsQuantityCheck(
    @OrderID int
)
RETURNS bit
AS
BEGIN
    IF exists (select * from Orders
        inner join OrderDetails on OrderDetails.OrderID = Orders.OrderID
        inner join Dishes on Dishes.DishID = OrderDetails.DishID
        inner join MenuDetails on MenuDetails.DishID = Dishes.DishID
        inner join Menu on Menu.MenuID = MenuDetails.MenuID
        where Menu.Ready = 0 and Orders.OrderID = @OrderID)
        RETURN 0
    RETURN 1
END

--complex constraint: seafood
ALTER TABLE OrderDetails
ADD CONSTRAINT SeaFood
CHECK (dbo.SeaFoodCheck(OrderID) = 1)

CREATE FUNCTION SeaFoodCheck (
    @OrderID int
)
RETURNS bit
AS
BEGIN
    IF exists(select * from Orders o where OrderID=@OrderID and DATEPART(weekday,
DateFrom) in (5,6,7) and PlacedDate<=DATEADD(DD,-(DATEPART(WEEKDAY, DateFrom)+5)%7,
DateFrom)) or not exists (select * from Orders o join OrderDetails OD on o.OrderID =
OD.OrderID and o.OrderID=@OrderID
join Dishes D on D.DishID = OD.DishID and SeaFood=1)
        RETURN 1
    RETURN 0
END

```

```

--add order to invoice constraint
ALTER TABLE Orders
ADD CONSTRAINT SameInvoiceAndClientOrderNIP
CHECK (dbo.InvoiceAndClientOrderNIPCheck(OrderID, InvoiceID) = 1)

CREATE FUNCTION InvoiceAndClientOrderNIPCheck (
    @OrderID int,
    @InvoiceID int
)
RETURNS bit
AS
BEGIN
    IF (select NIP from Orders join Clients C on Orders.ClientID = C.ClientID and
OrderID=@OrderID join CorporateClients CC on C.ClientID = CC.ClientID )
        = (select NIP from Invoices where InvoiceID=@InvoiceID) or @InvoiceID IS NULL
        RETURN 1
    RETURN 0
END

```

```

-- complex constraint: warunek aby żadne dwa menu nie zachodziły na
-- siebie datami wystawienia
ALTER TABLE Menu
ADD CONSTRAINT MenuCheck CHECK (
    dbo.MenuCon(MenuID) = 0
)

CREATE FUNCTION MenuCon (@MenuID int)
RETURNS bit
AS
BEGIN
    DECLARE @startdate AS date=(select DateFrom from Menu where MenuId=@MenuID)
    DECLARE @enddate AS date=(select DateTo from Menu where MenuId=@MenuID)
    if exists (select 1 from Menu as m
        where m.MenuID != @MenuID and
        ((@startdate < m.DateTo and m.DateFrom < @enddate) or
        (m.DateFrom < @enddate and @startdate < m.DateTo)))
        RETURN 1
    RETURN 0
END

```

## Widoki i Funkcje

```
-- widok statystyk zamówień z zeszłego miesiąca
CREATE VIEW MonthlyDishesStats AS
select DishID, Suma, Liczba
from dbo.MonthlyDishesStatsF(getdate())

-- widok statystyk zamówień z zeszłego tygodnia
CREATE VIEW WeeklyDishesStats AS
select DishID, Suma, Liczba
from dbo.WeeklyDishesStatsF(getdate())

-- widok wolnych stolików przez następną godzinę
CREATE VIEW FreeTablesOneHour AS
select TableID, Seats
from dbo.TimePeriodFreeTablesF(
    getdate(),
    DATEADD(HOUR,1,getdate()))

-- widok dzisiejszego menu
CREATE VIEW MenuToday AS
select d.DishName, md.Price
from Menu as m
join MenuDetails as md
on md.MenuID = m.MenuID
join Dishes as d
on d.DishID = md.DishID
where GETDATE() between m.DateFrom and m.DateTo;

-- widok aktualnie wolnych stolików
CREATE VIEW FreeTablesNow AS
select distinct t.TableID, t.seats
from Tables as t
where t.TableID not in (select TableId from TableInUse as tiu
join Orders as o
on o.OrderID = tiu.OrderID
where GETDATE() between o.DateFrom and o.DateTo)
and t.Available = 1;
```

```

--funkcja: menu na wybrany dzień
CREATE FUNCTION DishesInMenuF (@date datetime)
RETURNS TABLE as
RETURN
(select D.DishID, DishName, Price
from MenuDetails
join Menu on Menu.MenuID = MenuDetails.MenuID
join Dishes D on MenuDetails.DishID = D.DishID
where @date between Menu.DateFrom and Menu.DateTo)
GO

--funkcja: oblicza sumaryczną cenę zamówienia
CREATE FUNCTION OrderPrice (@orderid INT) RETURNS INT
AS BEGIN RETURN(
select SUM((Price*Quantity)*(1-Discout))
from Orders O
join OrderDetails OD
on O.OrderID = OD.OrderID and O.OrderID = @orderid
join Dishes D
on OD.DishID = D.DishID
join MenuDetails MD
on D.DishID = MD.DishID
join Menu M
on MD.MenuID = M.MenuID and
(O.DateFrom between M.DateFrom and M.DateTo)
GROUP BY O.OrderID)
END

-- funkcja: wolne stoliki na wybrany okres
CREATE FUNCTION TimePeriodFreeTablesF(
@DateStart datetime, @DateEnd datetime)
RETURNS TABLE AS RETURN(
select t.TableID, t.Seats
from Tables as t
join TableInUse as tiu on tiu.TableID = t.TableID
join Orders as o on o.OrderID = tiu.OrderID
where t.Available = 1 and
(@DateStart not between o.DateFrom and o.DateTo) and
(@DateEnd not between o.DateFrom and o.DateTo) and
(o.DateFrom not between @DateStart and @DateEnd) and
(o.DateTo not between @DateStart and @DateEnd))

```

```

--funkcja: statystyki klientów z dowolnego miesiąca (wybieramy początkową datę)
CREATE FUNCTION MonthlyClientsStatsF (@Date datetime)
RETURNS table AS RETURN
(select * from TimePeriodClientsStatsF(@Date, DATEADD(MONTH, 1, @Date)))

--funkcja: statystyki klientów z dowolnego tygodnia (wybieramy początkową datę)
CREATE FUNCTION WeeklyClientsStatsF (@Date datetime)
RETURNS table AS RETURN
(select * from TimePeriodClientsStatsF(@Date, DATEADD(WEEK, 1, @Date)))

--funkcja: statystyki klientów z dowolnego okresu czasu
CREATE FUNCTION TimePeriodClientsStatsF
(@DateStart datetime, @DateEnd datetime)
RETURNS TABLE AS RETURN (
    select c.ClientID, sum(od.Quantity * md.Price*(1-Discout)) Suma, sum(od.Quantity) Liczba
    from Clients as c
    join Orders as o
    on c.ClientID = o.ClientID
    join OrderDetails as od
    on od.OrderID = o.OrderID
    join MenuDetails as md
    on od.DishID = md.DishID
    join Menu as m
    on md.MenuID = m.MenuID
    where (o.DateFrom between @DateStart and @DateEnd) and
    ( (m.DateFrom between CONVERT(date,@DateStart) and CONVERT(date,@DateEnd))
    or (m.DateTo between CONVERT(date,@DateStart) and CONVERT(date,@DateEnd))
    or (CONVERT(date,@DateEnd) between m.DateFrom and m.DateTo)
    or (CONVERT(date,@DateStart) between m.DateFrom and m.DateTo) )
    group by c.ClientID
)

--funkcja: statystyki dań z dowolnego miesiąca (wybieramy początkową datę)
CREATE FUNCTION MonthlyDishesStatsF (@Date datetime)
RETURNS table AS RETURN
(select * from TimePeriodStatsF(@Date, DATEADD(MONTH, 1, @Date)))

--funkcja: statystyki dań z dowolnego tygodnia (wybieramy początkową datę)
CREATE FUNCTION WeeklyDishesStatsF (@Date datetime)
RETURNS table AS RETURN

```



```

(select * from TimePeriodStatsF(@Date, DATEADD(WEEK, 1, @Date)))

-- funkcja: statystyki zamówień z dowolnego okresu
CREATE FUNCTION TimePeriodStatsF
(@DateStart datetime, @DateEnd datetime)
RETURNS TABLE AS RETURN (
    select md.DishID, sum(od.Quantity * md.Price) Suma, sum(Quantity) Liczba
    from Orders as o
    join OrderDetails as od
    on o.OrderID = od.OrderID
    join MenuDetails as md
    on md.DishID = od.DishID
    join Menu as m
    on m.MenuID = md.MenuID
    where o.Finalized = 1
    and (o.DateFrom between @DateStart and @DateEnd)
    and ( (m.DateFrom between @DateStart and @DateEnd)
    or (m.DateTo between @DateStart and @DateEnd)
    or (@DateEnd between m.DateFrom and m.DateTo)
    or (@DateStart between m.DateFrom and m.DateTo))
    group by md.DishID)

```

## Procedury

```
-- finalize
CREATE PROCEDURE FinalizeOrder (@OrderID int)
AS
BEGIN
    IF (EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID))
        UPDATE Orders
        SET Finalized = 1
        WHERE OrderID = @OrderID
    ELSE
        PRINT 'Zamówienie nie znajduje się w bazie'
END

-- cancel
CREATE PROCEDURE CancelOrder (@OrderID INT)
AS
BEGIN
    IF (EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID))
        UPDATE Orders
        SET Cancelled = 1
        WHERE OrderID = @OrderID
    ELSE
        PRINT 'Zamówienie nie znajduje się w bazie'
END

-- accept
CREATE PROCEDURE AcceptOrder (@OrderID INT)
AS
BEGIN
    IF (EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID))
        UPDATE Orders
        SET Accepted = 1
        WHERE OrderID = @OrderID
    ELSE
        PRINT 'Zamówienie nie znajduje się w bazie'
END
```

```

-- add to invoice
CREATE PROCEDURE AddOrderToInvoice (@OrderID INT, @InvoiceID INT)
AS
BEGIN
    UPDATE Orders
    SET InvoiceID = @InvoiceID
    WHERE OrderID = @OrderID
END

-- add dish to order
CREATE PROCEDURE AddDishToOrder
    @OrderID int,
    @DishID int,
    @Quantity int
AS
BEGIN
    IF(exists(select 1 from OrderDetails where OrderID=@OrderID and DishID=@DishID))
        UPDATE OrderDetails
        SET Quantity=(select Quantity from OrderDetails where @OrderID=OrderID and
DishID=@DishID)+@Quantity
        WHERE OrderID=@OrderID and DishID=@DishID
    ELSE
        INSERT INTO OrderDetails(OrderID, DishID, Quantity)
        SELECT @OrderID, @DishID, @Quantity FROM Dishes WHERE DishID = @DishID
END

```