# Krzysztof Usnarski & Wiktor Woźny & Adam Misztal

# Opis użytkowników

- klient indywidualny – klient restauracji, który korzysta z jej usług na miejscu lub składa zamówienie poprzez internetowy formularz
- firma – zbiorowy klient restauracji, korzystający z jej usług, ma możliwość otrzymania faktury dla zamówień
- pracownik restauracji – osoba zajmująca się obsługą klientów indywidualnych oraz firm i administrowaniem bazy danych

# Funkcje użytkowników

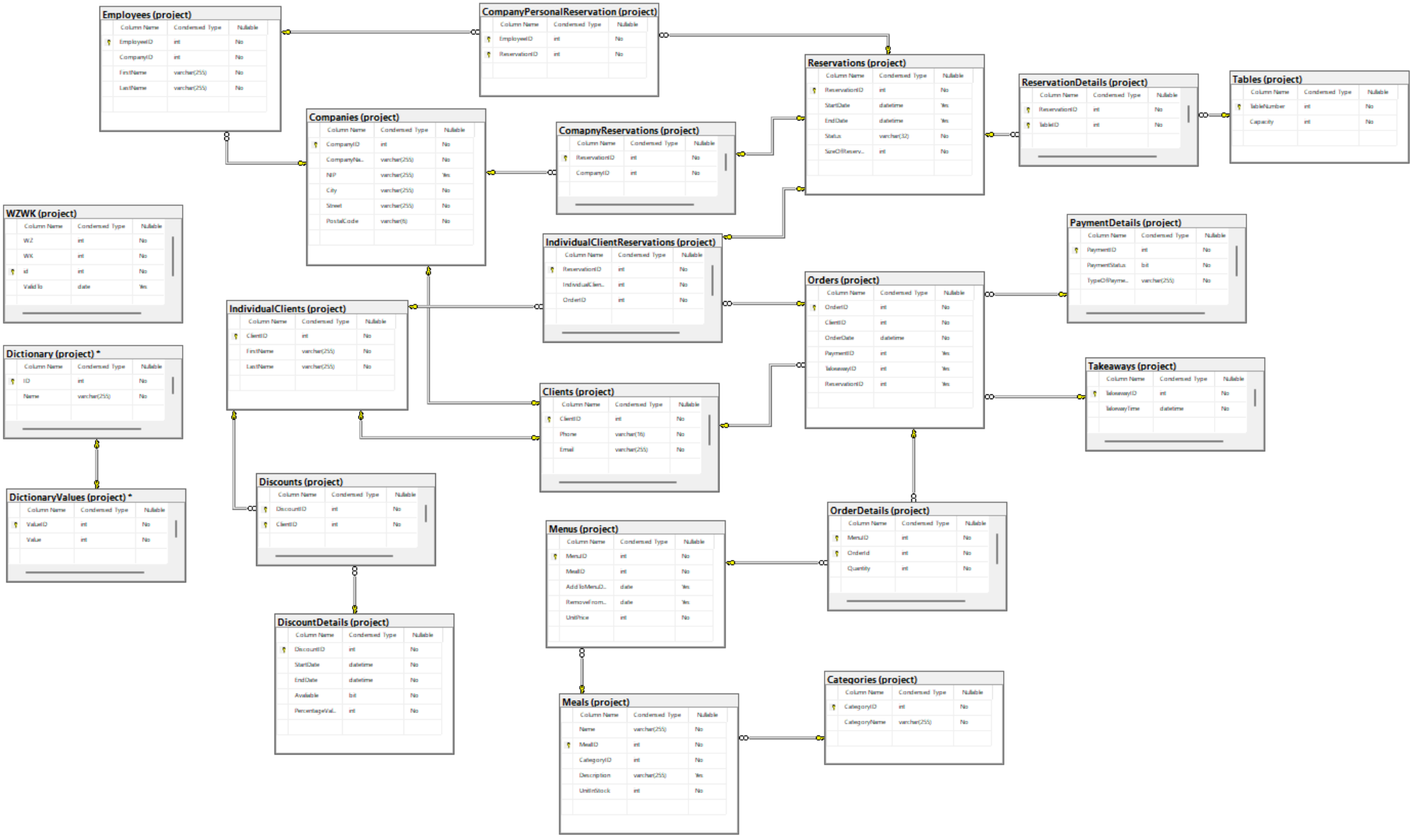
- klient indywidualny:
  - ➔ Złożenie zamówienia na miejscu
  - ➔ Złożenie zamówienia z wykorzystaniem formularza WWW
  - ➔ Rezerwacja stolika (jeśli spełni wymagania)
  - ➔ Sprawdzenie wymagań jakie trzeba spełnić, aby zarezerwować stolik
  - ➔ Generowanie raportów dotyczących zamówień
  - ➔ Generowanie statystyk dotyczących zamówień
  - ➔ Sprawdzenie swoich poprzednich zamówień
  - ➔ Sprawdzenie aktualnych rabatów
- firma
  - ➔ Złożenie zamówienia na większą ilość posiłków
  - ➔ Złożenie zamówienia jako catering (bez dostawy)
  - ➔ Rezerwacja stolik dla swoich pracowników (imiennie lub tylko stolików)
  - ➔ Wystawienie faktury dla danego zamówienia
  - ➔ Wystawienie faktury zbiorczej raz na miesiąc
- pracownik restauracji
  - ➔ Zarządzanie rezerwacjami stolików dla klientów oraz firm
  - ➔ Przyjmowanie zamówień
  - ➔ Ustalanie aktualnego menu
  - ➔ Wystawianie faktur jednorazowych
  - ➔ Wystawianie faktur zbiorczych dla firm
  - ➔ Dodawanie nowych produktów do menu
  - ➔ Zmienianie cen produktów
  - ➔ Usuwanie produktów z menu
  - ➔ Akceptacja zamówień
  - ➔ Akceptacja rezerwacji
  - ➔ Zmiana wartości Z1, K1, R1, K2, R2, D1, WK, WZ
  - ➔ Sprawdzenie dostępności stolików w danym czasie
  - ➔ Zarządzanie kategoriami produktów

- ➔ Generowanie raportów dla różnych okresów czasu (dotyczących rezerwacji, rabatów, menu)
- ➔ Generowanie statystyk
- ➔ Zarządzenie stolikami (np. Dostawienie stolika)

# Funkcje systemowe

1. Przydzielanie rabatów dla klientów indywidualnych
2. Weryfikacja możliwości zarezerwowania stolika przez klienta
3. Obliczanie wartości zamówienia
4. Informowanie o potrzebie zmiany menu
5. Obsługa wcześniejszego zamawiania dań zawierających owoce morza
6. Generowanie raportów na żądanie

# Schemat bazy danych



# Tabela Categories - kategorie

**Kolumny:**

**CategoryID:** id kategorii (klucz główny)
**CategoryName:** nazwa kategorii

```
create table project.Categories
(
    CategoryID   int          not null
        constraint Categories_pk
            primary key,
    CategoryName varchar(255) not null
)
go
```

# Tabela Cities - miasta (słownik)

**Kolumny:**

**CityID:** id miasta(klucz główny)
**CityName:** nazwa miasta
**CountryID:** id państwa (klucz obcy)

```
create table project.Cities
(
    CityID    int identity
        constraint Cities_pk
            primary key,
    CityName  varchar(255) not null,
    CountryID int
        constraint FK_Cities_Countries
            references project.Countries
)
```

# Tabela Countries - państwa (słownik)

**Kolumny:**

**CountryID:** id państwa (klucz główny)

**CountryName:** nazwa państwa

```
create table project.Countries
(
    CountryID   int identity
        constraint Countries_pk
            primary key,
    CountryName varchar(255) not null
)
go
```

# Tabela Clients - klienci

**Kolumny:**

**ClientID:** id klienta (klucz główny)

**Phone:** numer telefonu klienta - 9 cyfr

**Email:** email klienta - zawiera ‘@’ i potem ‘.’ (unikalny)

```
create table project.Clients
(
    ClientID int          not null
        constraint Clients_pk
            primary key,
    Phone    varchar(16)  not null
        constraint phone_check
            check ([Phone] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Email    varchar(255) not null
        constraint email_pk
            unique
        constraint email_check
            check ([Email] like '%@%.%')
)
go
```

# Tabela CompanyReservations

## Kolumny:

**ReservationID:** Id rezerwacji (klucz główny, klucz obcy)
**CompanyID:** Id firmy (klucz obcy)

```
create table project.ComapnyReservations
(
    ReservationID int not null
        constraint ComapnyReservations_pk
            primary key
        constraint [ReservationID:ReservationID(2)]
            references project.Reservations,
    CompanyID     int not null
        constraint FK_ComapnyReservations_Companies
            references project.Companies
)
go
```

# Tabela Companies - Firmy

## Kolumny:

**CompanyID:** id firmy (klucz główny, klucz obcy)
**CompanyName:** nazwa firmy
**NIP** - 10 cyfr (unikalny)
**CityID:** id miasta (klucz obcy)
**Street:** ulica na której znajduje się siedziba firmy
**Postal Code:** kod pocztowy - 2 liczby ‘-’ i 3 liczby

```
create table project.Companies
(
    CompanyID   int          not null
        constraint Companies_pk
            primary key
        constraint [CompanyID:ClientID]
            references project.Clients,
    CompanyName varchar(255) not null,
    NIP         varchar(255)
        constraint nip_pk
            unique
        constraint nipe_check
            check ([NIP] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    CityID      int          not null
        constraint FK_Companies_Cities
            references project.Cities,
    Street      varchar(255) not null,
    PostalCode  varchar(6)   not null
        constraint postal_code_check
            check ([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]')
)
go
```

# Tabela CompanyPersonalReservation - tabela łącząca pracowników i rezerwacje

## Kolumny:

**EmployeeID:** Id pracownika (klucz główny, klucz obcy)
**ReservationID:** Id rezerwacji (klucz główny, klucz obcy)

```
create table project.CompanyPersonalReservation
(
    EmployeeID    int not null
        constraint [EmplyeeID:EmployeeID]
            references project.Employees,
    ReservationID int not null
        constraint [ReservationID:ReservationID]
            references project.Reservations,
    constraint CompanyPersonalReservation_pk
        primary key (EmployeeID, ReservationID)
)
go
```

# Tabela Dictionary - słownik z parametrami

## Kolumny:

**ID:** id słownika (klucz główny)
**Name**: nazwa

```
create table project.Dictionary
(
    ID   int identity
        constraint Słownik_pk
            primary key
        constraint FK_Dictionary_DictionaryValues
            references project.DictionaryValues,
    Name varchar(255) not null
)
go
```

| | ID | Name |
|---|---|---|
| 1 | 1 | Z1 |
| 2 | 2 | K1 |
| 3 | 3 | R1 |
| 4 | 4 | K2 |
| 5 | 5 | R2 |
| 6 | 6 | D1 |

# Tabela DictionaryValues

## Kolumny:

**ValueID** (klucz główny)
**Value:** wartość parametru

```
create table project.DictionaryValues
(
    ValueID int identity
        constraint SłownikValues_pk
            primary key,
    Value   int not null
)
go
```

| | ValueID | Value |
|---|---|---|
| 1 | 1 | 10 |
| 2 | 2 | 30 |
| 3 | 3 | 3 |
| 4 | 4 | 1000 |
| 5 | 5 | 5 |
| 6 | 6 | 7 |

# Tabela Discounts

## Kolumny:

**DiscoutID:** id zniżki (klucz główny)
**ClientID:** id klienta (klucz obcy)

```
create table project.Discounts
(
    DiscountID int not null
        constraint FK_Discounts_DiscountDetails
            references project.DiscountDetails,
    ClientID   int not null
        constraint FK_Discounts_IndividualClients
            references project.IndividualClients,
    constraint Discounts_pk
        primary key (DiscountID, ClientID)
)
go
```

# Tabela DiscountDetails

## Kolumny:

**DiscountID:** id zniżki (klucz główny)
**StartDate:** data rozpoczęcia okresu w którym istnieje możliwość wykorzystania zniżki
**EndDate:** data zakończenia okresu j.w.
**Available:** informacja o wykorzystaniu zniżki **(**domyślnie 1)
**PercentageValue:** procentowa wartość zniżki

```
create table project.DiscountDetails
(
    DiscountID      int identity
        constraint DiscountDetails_pk
            primary key,
    StartDate       datetime      not null,
    EndDate         datetime,
    Avaliable       bit default 1 not null,
    PercentageValue int           not null
        constraint discount_check
            check ([PercentageValue] > 0 AND [PercentageValue] <= 100),
    constraint date_check
        check ([EndDate] > [StartDate])
)
go
```

# Tabela Employees

**Kolumny:**

**EmployeeID:** id pracownika (klucz główny)
**CompanyID:** id firmy (klucz obcy)
**FirstName:** imię pracownika
**LastName:** nazwisko pracownika

```
create table project.Employees
(
    EmployeeID int identity
        constraint Employees_pk
            primary key,
    CompanyID  int          not null
        constraint FK_Employees_Companies
            references project.Companies,
    FirstName  varchar(255) not null,
    LastName   varchar(255) not null
)
go
```

# Tabela IndividualClients

**Kolumny:**

**ClientID** (klucz główny, klucz obcy)
**FirstName:** imię klienta
**LastName:** nazwisko klienta

```
create table project.IndividualClients
(
    ClientID  int          not null
        constraint IndividualClients_pk
            primary key
        constraint [ClientID:ClientID]
            references project.Clients,
    FirstName varchar(255) not null,
    LastName  varchar(255) not null
)
go
```

# Tabela IndividualClientReservations

## Kolumny:

**ReservationID:** id rezerwacji (klucz główny)
**IndividualClientID:** id klienta indywidualnego (klucz obcy)
**OrderID:** id zamówienia (klucz obcy)

```
create table project.IndividualClientReservations
(
    ReservationID      int not null
        constraint IndividualClientReservations_pk
            primary key,
    IndividualClientID int not null
        constraint FK_IndividualClientReservations_IndividualClients
            references project.IndividualClients,
    OrderID            int not null
        constraint FK_IndividualClientReservations_Orders
            references project.Orders
)
go
```

# Tabela Meals

## Kolumny:

**MealID:** id posiłku (klucz główny)
**Name:** nazwa (unikalny)
**CategoryID:** id kategorii, do której należy posiłek (klucz obcy)
**Description:** opis
**UnitInStock:** liczba dostępnych jednostekk - liczba większa bądź równa 0

```
create table project.Meals
(
    Name        varchar(255) not null
        constraint name_pk
            unique,
    MealID      int          not null
        constraint Meals_pk
            primary key,
    CategoryID  int          not null
        constraint FK_Meals_Categories
            references project.Categories,
    Description varchar(255),
    UnitInStock int          not null
        constraint unit_check
            check ([UnitInStock] >= 0)
)
go
```

# Tabela Menus

## Kolumny:

**MenuID:** id menu (klucz główny)
**MealID:** id dania (klucz obcy)
**AddToMenuDate:** data dodania do menu
**RemoveFromMenuDate:** data usunięcia pozycji z menu (data usunięcia późniejsza niż data dodania)
**UnitPrice:** cena jednostkowa (liczba większa bądź równa zero)

```
create table project.Menus
(
    MenuID             int not null
        constraint Menus_pk
            primary key,
    MealID             int not null
        constraint FK_Menus_Meals
            references project.Meals,
    AddToMenuDate      date,
    RemoveFromMenuDate date,
    UnitPrice          int not null
        constraint unit_price_check
            check ([UnitPrice] >= 0),
    constraint menu_date_check
        check ([RemoveFromMenuDate] > [AddToMenuDate])
)
go
```

# Tabela OrderDetails

## Kolumny:

**MealID:** id dania (klucz główny, klucz obcy)
**OrderID:** id zamówienia (klucz główny, klucz obcy)
**Quantity:** ilość (liczba większa bądź równa zero)

```
create table project.OrderDetails
(
    MenuID   int not null
        constraint FK_OrderDetails_Menus
            references project.Menus,
    OrderId  int not null
        constraint OrderID_OrderID
            references project.Orders,
    Quantity int not null
        constraint quantity_check
            check ([Quantity] > 0),
    constraint OrderDetails_pk
        primary key (MenuID, OrderId)
)
go
```

# Tabela Orders

## Kolumny:

**OrderID:** id zamówienia (klucz główny)
**ClientID:** id klienta (klucz obcy)
**OrderDate:** data złożenia zamówienia
**PaymentID:** id płatności (klucz obcy)
**TakeawayID:** id odbioru (klucz obcy)
**ReservationID:** id rezerwacji (klucz obcy)

```
create table project.Orders
(
    OrderID       int      not null
        constraint Orders_pk
            primary key,
    ClientID      int      not null
        constraint FK_Orders_Clients
            references project.Clients,
    OrderDate     datetime not null,
    PaymentID     int      not null
        constraint FK_Orders_PaymentDetails
            references project.PaymentDetails,
    TakeawayID    int
        constraint FK_Orders_Takeaways
            references project.Takeaways,
    ReservationID int
)
go
```

# Tabela PaymentDetails

## Kolumny:

**PaymentID:** id płatności (klucz główny)
**PaymentStatus:** status płatności (domyślnie zamówienie jest nieopłacone)
**TypeOfPayment:** typ płatności (możliwość płatności tylko gotówką, kartą, blikiem, lub wzięcie na fakturę)

```
create table project.PaymentDetails
(
    PaymentID     int identity
        constraint PaymentDetails_pk
            primary key,
    PaymentStatus bit default 0 not null,
    TypeOfPayment varchar(255)  not null
        constraint payment_type_check
            check ([TypeOfPayment] = 'invoice' OR
                   [TypeOfPayment] = 'blik' OR
                   [TypeOfPayment] = 'paypal' OR
                   [TypeOfPayment] = 'card' OR
                   [TypeOfPayment] = 'cash')
)
go
```

# Tabela ReservationDetails

## Kolumny:

**ReservationID:** id rezerwacji (klucz główny, klucz obcy)
**TableID:** id stołu (klucz główny, klucz obcy)

```
create table project.ReservationDetails
(
    ReservationID int not null
        constraint ReservationDetails_Reservations_null_fk
            references project.Reservations,
    TableID       int not null
        constraint ReservationDetails_Tables_null_fk
            references project.Tables,
    constraint ReservationDetails_pk
        primary key (ReservationID, TableID)
)
go
```

# Tabela Reservations

## Kolumny:

**ReservationID:** id rezerwacji (klucz główny)
**StartDate:** data rozpoczęcia rezerwacji (musi być w przyszłości)
**EndDate**: data zakończenia rezerwacji (data rozpoczęcia rezerwacji musi być wcześniej niż data zakończenia rezerwacji)
**Status:** status rezerwacji (jedna z czterech wartości: (finished, rejected, accepted, pending))
**SizeOfReservation:** ilość osób w rezerwacji (przynajmniej dwie osoby)

```
create table project.Reservations
(
    ReservationID     int         not null
        constraint Reservations_pk
            primary key,
    StartDate         datetime    not null
        constraint reservation_date_check_2
            check ([StartDate] > getdate()),
    EndDate           datetime    not null,
    Status            varchar(32) not null
        constraint reservation_status_check
            check ([Status] = 'finished' OR
                   [Status] = 'rejected' OR
                   [Status] = 'accepted' OR
                   [Status] = 'pending'),
    SizeOfReservation int         not null
        constraint size_reservation_check
            check ([SizeOfReservation] >= 2),
    constraint reservation_date_check
        check ([EndDate] > [StartDate])
)
go
```

# Tabela Tables

## Kolumny:

**TableNumber:** numer stolika (klucz główny)
**Capacity:** ilość osób które mogą zasiąść przy stoliku (pojemność stolika przynajmniej 1)

```
create table project.Tables
(
    TableNumber int identity
        constraint Tables_pk
            primary key,
    Capacity    int not null
        constraint capacity_check
            check ([Capacity] >= 1)
)
go
```

# Tabela Takeaways

## Kolumny:

TakeawayID: id odbioru (klucz główny)
TakeawayTime: data odbioru zamówienia (data odbioru zamówienia późniejsza niż aktualna data)

```
create table project.Takeaways
(
    TakeawayID  int identity
        constraint Takeaway_pk
            primary key,
    TakewayTime datetime not null
        constraint takeway_check
            check ([TakewayTime] > getdate())
)
go
```

# Tabela WZKW

## Kolumny:

WZ: wartość WZ
WK: wartość WK
id: id (klucz główny)
ValidTo: data w której dane wartości WZ i WK przestały obowiązywać

```
create table project.WZWK
(
    WZ      int not null,
    WK      int not null,
    id      int identity
        constraint WZWK_pk
            primary key,
    ValidTo date
)
go
```

# Widoki

## ClientWithAvaliableFirstDiscount

Klienci którym przysługuje pierwsza zniżka

```
create view project.ClientsWithAvailableFirstDiscount as
SELECT ClientID
FROM project.Clients AS C
WHERE (SELECT COUNT(underQuery.orderId)
       FROM (SELECT OrderID as orderId
             FROM project.Orders
             WHERE Orders.ClientID = C.ClientID
               AND (SELECT Value
                    FROM project.OrderSummaries
                    WHERE OrderSummaries.OrderID = Orders.OrderID) >=
                   (SELECT dv.Value FROM project.DictionaryValues as dv WHERE dv.ValueID = 2)) underQuery) >
      (SELECT dv.Value FROM project.DictionaryValues as dv WHERE dv.ValueID = 1)
go
```

## ClientWithAvaliableSecondDiscount

Klienci którym przysługuje druga zniżka

```
create view project.ClientsWithAvailableSecondDiscount as
SELECT ClientID
FROM project.Clients AS C
WHERE (SELECT SUM(underQuery.value)
       FROM (SELECT (SELECT Value
                     FROM project.OrderSummaries
                     WHERE OrderSummaries.OrderID = Orders.OrderID) as value
             FROM project.Orders
             WHERE Orders.ClientID = C.ClientID) underQuery) >
      (SELECT dv.Value FROM project.DictionaryValues as dv WHERE dv.ValueID = 4)
go
```

# ClientWithAvaliableReservation

Klienci którzy mogą dokonać rezerwacji

```
CREATE view project.ClientDiscounts as
select ClientID
from project.IndividualClients IC
where ((select COUNT(*)
        from (select OrderID,
                     (select SUM(Quantity * UnitPrice) as 'Value'
                      from project.OrderDetails
                               inner join project.Menus M on M.MenuID = OrderDetails.MenuID
                      where OrderId = O.OrderID) as 'Value'
              from project.Orders O
              where ClientID = IC.ClientID) Det
        where Value > (select WZ
                       from project.WZWK
                       where ValidTo IS NULL))) > (select WK
                                                   from project.WZWK
                                                   where ValidTo IS NULL)
union select CompanyID from project.Companies
go
```

# CompaniesSummaryMonthly

Podsumowanie zamówień firm (miesięczne)

```
CREATE view project.CompaniesSummaryMonthly as
select ClientID, CompanyName, Year, Month, COUNT(Month) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as 'Year', MONTH(OrderDate) as 'Month', (
    SELECT SUM(Quantity*UnitPrice)
    FROM project.OrderDetails OD
    INNER JOIN project.Menus M on M.MenuID = OD.MenuID
    WHERE O.OrderID = OD.OrderId
    GROUP BY OD.OrderID) as 'Value'
from project.Orders O
) SAD
inner join project.Companies C ON C.CompanyID = ClientID
group by ClientID, CompanyName, Year, Month
go
```

# CompaniesSummaryWeekly

Podsumowanie zamówień firm (tygodniowe)

```
CREATE view project.CompaniesSummaryWeekly as
select ClientID, CompanyName, Year, Week, COUNT(Week) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as 'Year', DATEPART(week, OrderDate) as 'Week',
    (SELECT SUM(Quantity*UnitPrice)
    FROM project.OrderDetails OD
    INNER JOIN project.Menus M on M.MenuID = OD.MenuID
    WHERE O.OrderID = OD.OrderId
    group by OD.OrderID) as 'Value'
from project.Orders O
) SAD
inner join project.Companies C ON C.CompanyID = ClientID
group by ClientID, CompanyName, Year, Week
go
```

# CompaniesSummarOverall

Podsumowanie zamówień firm (ogólnie)

```
create view project.CompaniesSummaryOverall as

select Comp.CompanyID,
       (select count(ASD.OrderID)
        from (select Orders.OrderId
              From project.Orders
              where ClientID = Comp.CompanyID) ASD) as 'Orders',
       (select sum(DAS.Value)
        from (select Quantity * UnitPrice * (ISNULL(1 - PercentageValue, 1)) as 'Value'
              From project.Orders O
                       inner join project.OrderDetails OD on O.OrderID = OD.OrderId
                       inner join project.Menus M on M.MenuID = OD.MenuID
                       inner join project.Meals M2 on M2.MealID = M.MealID
                       left join project.Discounts D on O.ClientID = D.ClientID
                       left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
              where O.ClientID = Comp.CompanyID) DAS) as 'Value'
FROM project.Companies Comp
where (select sum(DAS.Value)
       from (select Quantity * UnitPrice * (ISNULL(1 - PercentageValue, 1)) as 'Value'
             From project.Orders O
                      inner join project.OrderDetails OD on O.OrderID = OD.OrderId
                      inner join project.Menus M on M.MenuID = OD.MenuID
                      inner join project.Meals M2 on M2.MealID = M.MealID
                      left join project.Discounts D on O.ClientID = D.ClientID
                      left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
             where O.ClientID = Comp.CompanyID) DAS) is not null
go
```

# Current Menu

Pokazuje aktualne menu

```
create view project.CurrentMenu as
select MenuID, Menus.MealID, Name, UnitPrice, UnitInStock,
       Description, CategoryName, AddToMenuDate from project.Menus
              inner join project.Meals M on M.MealID = Menus.MealID
              inner join project.Categories C on C.CategoryID = M.CategoryID
              where RemoveFromMenuDate IS null
go
```

# DiscountsMonthlyStat

Zniżki wykorzystane w danym miesiącu

```
create view project.DiscountsMonthlyStat as
select YEAR(StartDate) as 'Year', MONTH(StartDate) as 'Month', COUNT(*) as 'CountOfDiscounts'
from project.DiscountDetails
where Avaliable = 0
group by YEAR(StartDate), MONTH(StartDate)
go
```

# DiscountsWeeklyStat

Zniżki wykorzystane w danym tygodniu

```
create view project.DiscountsWeeklyStat as
select YEAR(StartDate) as 'Year', DATEPART(week, StartDate) as 'Week',
       COUNT(*) as 'CountOfDiscounts'
from project.DiscountDetails
where Avaliable = 0
group by YEAR(StartDate), DATEPART(week, StartDate)
go
```

# IndividualClientsSummaryMonthly

Podsumowanie zamówień klienta indywidualnego (miesięczne)

```
CREATE view project.IndividualClientsSummaryMonthly as
select IC.ClientID, FirstName, LastName, Year, Month, COUNT(Month) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as 'Year', MONTH(OrderDate) as 'Month', (
    SELECT SUM(Quantity*UnitPrice*ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2)) FROM project.OrderDetails OD
        INNER JOIN project.Menus M on M.MenuID = OD.MenuID
        left join project.Discounts D on O.ClientID = D.ClientID
        left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
            WHERE O.OrderID = OD.OrderId
            group by OD.OrderID
    ) as 'Value'
from project.Orders O
) SAD
inner join project.IndividualClients IC ON IC.ClientID = SAD.ClientID
group by IC.ClientID, FirstName, LastName,  Year, Month
go
```

# IndividualClientsSummaryWeekly

Podsumowanie zamówień klienta indywidualnego (tygodniowe)

```
CREATE view project.IndividualClientsSummaryWeekly as
select IC.ClientID, FirstName, LastName, Year, Week, COUNT(Week) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as 'Year', DATEPART(week, OrderDate) as 'Week', (
    SELECT SUM(Quantity*UnitPrice*ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2))
    FROM project.OrderDetails OD
        INNER JOIN project.Menus M on M.MenuID = OD.MenuID
        left join project.Discounts D on O.ClientID = D.ClientID
        left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
            WHERE O.OrderID = OD.OrderId
            group by OD.OrderID
    ) as 'Value'
from project.Orders O
) SAD
inner join project.IndividualClients IC ON IC.ClientID = SAD.ClientID
group by IC.ClientID, FirstName, LastName,  Year, Week
go
```

# IndividualClientsSummaryOverall

Podsumowanie zamówień klienta indywidualnego (ogólnie)

```
CREATE view project.IndividualClientsSummaryOverall as
select IC.ClientID,
       (select count(ASD.OrderID)
        from (select Orders.OrderId
              From project.Orders
              where ClientID = IC.ClientID) ASD) as 'Orders',
       (select sum(DAS.Value)
        from (select Quantity * UnitPrice * ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2) as 'Value'
              From project.Orders O
                       inner join project.OrderDetails OD on O.OrderID = OD.OrderId
                       inner join project.Menus M on M.MenuID = OD.MenuID
                       inner join project.Meals M2 on M2.MealID = M.MealID
                       left join project.Discounts D on O.ClientID = D.ClientID
                       left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
              where O.ClientID = IC.ClientID) DAS) as 'Value'
FROM project.IndividualClients IC
where (select sum(DAS.Value)
       from (select Quantity * UnitPrice * ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2) as 'Value'
             From project.Orders O
                      inner join project.OrderDetails OD on O.OrderID = OD.OrderId
                      inner join project.Menus M on M.MenuID = OD.MenuID
                      inner join project.Meals M2 on M2.MealID = M.MealID
                      left join project.Discounts D on O.ClientID = D.ClientID
                      left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
             where O.ClientID = IC.ClientID) DAS) is not null
go
```

# MealsIncomeStats

Przychód generowany przez poszczególne produkty

```
CREATE view project.MealsIncomeStats as
select M2.MealID, M2.Name,
       sum(Quantity * M.UnitPrice * ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID = OD.OrderId
inner join project.Menus M on M.MenuID = OD.MenuID
    inner join project.Meals M2 on M2.MealID = M.MealID
left join project.Discounts D on O.ClientID = D.ClientID
left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
group by M2.MealID, M2.Name
go
```

# MealsLongerThanTwoWeeksInMenu

Posiłki które są w menu dłużej niż 2 tygodnie

```
use u_usnarski
go

create view project.MealsLongerThanTwoWeeksInMenu as
SELECT m.MealID, DATEDIFF(DAY, m.AddToMenuDate, getdate()) as 'How long in menu (days)'
FROM project.Menus as m
WHERE( m.RemoveFromMenuDate>=GETDATE() OR  m.RemoveFromMenuDate IS NULL)
AND DATEDIFF(DAY, m.AddToMenuDate, getdate())>14
go

grant select on project.MealsLongerThanTwoWeeksInMenu to moderator
go

grant select on project.MealsLongerThanTwoWeeksInMenu to worker
go
```

# MealsSoldStatsMonthly

Pokazuje ile danych produktów zostało sprzedanych każdego miesiąca

```
create view project.MealsSoldStatsMonthly as
select year(OrderDate) as 'year',
       month(OrderDate) as 'month', M.MealID, M.Name, sum(Quantity) as 'amount'
from project.Meals M
inner join project.Menus M2 on M.MealID = M2.MealID
inner join project.OrderDetails OD on M2.MenuID = OD.MenuID
inner join project.Orders O on O.OrderID = OD.OrderId
group by M.MealID, M.Name, year(OrderDate), month(OrderDate)
go
```

# MealsSoldStatsOverall

Pokazuje ile danych produktów zostało sprzedanych ogólnie

```
create view project.MealsSoldStatsOverall as
select M.MealID, M.Name, sum(Quantity) as 'amount' from project.Meals M
inner join project.Menus M2 on M.MealID = M2.MealID
inner join project.OrderDetails OD on M2.MenuID = OD.MenuID
group by M.MealID, M.Name
go
```

# MonthlyIncome

Miesięczny dochód

```
CREATE view project.MonthlyIncome as
select year(OrderDate) as 'year', month(OrderDate) as 'month',
       sum(Quantity * M.UnitPrice * ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID = OD.OrderId
inner join project.Menus M on M.MenuID = OD.MenuID
left join project.Discounts D on O.ClientID = D.ClientID
left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
group by year(OrderDate), month(OrderDate)
go
```

# NotUsedMealsInStock

Produkty które nie są obecnie w menu ale są na stanie

```
create view project.NotUsedMealsInStock as
select Name, MealID, UnitInStock
FROM project.Meals
where MealID NOT IN (select Menus.MealID
                     from project.Menus
                              inner join project.Meals M on M.MealID = Menus.MealID
                     where RemoveFromMenuDate IS NULL)
  and UnitInStock > 0
go
```

# OrderSummaries

Podsumowanie każdego zamówienia

```
CREATE view project.OrderSummaries as
select Orders.OrderID,
       SUM(Quantity*UnitPrice*ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2)) as 'Value'
from project.Orders
inner join project.OrderDetails OD on Orders.OrderID = OD.OrderId
inner join project.Menus M on M.MenuID = OD.MenuID
left join project.Discounts D on Orders.ClientID = D.ClientID
left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
group by Orders.OrderID
go
```

# PendingReservations

Rezerwacje oczekujące na obsłużenie przez pracownika

```
create view project.CurrentReservations as
select * from project.Reservations R
where Status = 'pending'
go
```

# ReservationsInfo

Informacje o zaakceptowanych rezerwacjach

```
create view project.ReservationsInfo as
select R.ReservationID, R.SizeOfReservation, T.TableNumber, T.Capacity from project.Reservations R
inner join project.ReservationDetails RD on R.ReservationID = RD.ReservationID
                    inner join project.Tables T on T.TableNumber = RD.TableID
where Status = 'accepted'
go
```

# TablesMonthly

Miesięczne statystyki stolików

```
create view project.TablesMonthly as
select  YEAR(StartDate) as 'Year',
        MONTH(StartDate) as 'Month', TableID, Capacity,
        COUNT(TableID) as 'Count'
from project.Tables T
         inner join project.ReservationDetails RD on T.TableNumber = RD.TableID
         inner join project.Reservations R2 on R2.ReservationID = RD.ReservationID
group by TableID, YEAR(StartDate), MONTH(StartDate), Capacity
go
```

# TablesWeekly

Tygodniowe statystyki stolików

```
create view project.TablesWeekly as
select YEAR(StartDate) as 'Year',
       DATEPART(week, StartDate) as 'Week', TableID, Capacity,
       COUNT(TableID) as 'Count'
from project.Tables T
         inner join project.ReservationDetails RD on T.TableNumber = RD.TableID
         inner join project.Reservations R2 on R2.ReservationID = RD.ReservationID
group by TableID, YEAR(StartDate), DATEPART(week, StartDate), Capacity
go
```

# SeafoodMeals

Pokazuje dane potraw zawierających owoce morza

```
CREATE view project.SeafoodMeals as
SELECT MealID, Name, Description, UnitInStock
FROM project.Meals M
INNER JOIN project.Categories C ON C.CategoryID = M.CategoryID
WHERE CategoryName = 'Seafood'
go
```

# SeafoodMealsStats

Pokazuje statystyki związane z owocami morza

```
use u_usnarski
go

CREATE view project.SeafoodMealsStats as
select M.MealID, OrderDetails.OrderId, Quantity from project.OrderDetails
inner join project.Orders O on O.OrderID = OrderDetails.OrderId
inner join project.Menus M on M.MenuID = OrderDetails.MenuID
inner join project.Meals M2 on M2.MealID = M.MealID
left join project.Reservations R on R.ReservationID = O.ReservationID
where (M.MealID in (SELECT MealID FROM project.SeafoodMeals)) and GETDATE() < R.StartDate and DATEDIFF(DAY, getdate(), R.StartDate) < 7
union
select M.MealID, OrderDetails.OrderId, Quantity from project.OrderDetails
inner join project.Orders O on O.OrderID = OrderDetails.OrderId
inner join project.Menus M on M.MenuID = OrderDetails.MenuID
inner join project.Meals M2 on M2.MealID = M.MealID
left join project.Takeaways T on T.TakeawayID = O.TakeawayID
where (M.MealID in (SELECT MealID FROM project.SeafoodMeals)) and GETDATE() < T.TakewayTime and DATEDIFF(DAY, getdate(), T.TakewayTime) < 7
go

grant select on project.SeafoodMealsStats to moderator
go

grant select on project.SeafoodMealsStats to worker
go
```

# WeeklyIncome

## Tygodniowy przychód

```
CREATE view project.WeeklyIncome as
select year(OrderDate) as 'year',
       datepart(week, OrderDate) as 'week',
       sum(Quantity * M.UnitPrice * ROUND((ISNULL(100 - PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID = OD.OrderId
inner join project.Menus M on M.MenuID = OD.MenuID
left join project.Discounts D on O.ClientID = D.ClientID
left join project.DiscountDetails DD on DD.DiscountID = D.DiscountID
group by year(OrderDate), datepart(week, OrderDate)
go
```

# Procedury

## AddCategory

Dodaje kategorie do tabeli, przyjmuje w argumencie nazwę kategorii

```
CREATE PROCEDURE addCategory @CategoryName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
                SELECT *
                FROM project.Categories
                WHERE @CategoryName = CategoryName
            )
            BEGIN
                ;
                THROW 52000, N'Kategoria jest już dodana', 1
            end
        DECLARE @CategoryID INT
        SELECT @CategoryID = ISNULL(MAX(CategoryID), 0) + 1
        FROM project.Categories
        INSERT INTO project.Categories(CategoryID, CategoryName)
        VALUES (@CategoryID, @CategoryName);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048) =
            N'Błąd dodawania kategorii: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
go
```

# AddCity

Dodaje miasto do słownika miast, przyjmuje w argumencie nazwę miasta i nazwę państwa, do którego należy

```
CREATE PROCEDURE AddCity @CityName varchar(255),
                         @CountryName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
                SELECT *
                FROM project.Cities
                WHERE CityName = @CityName
            )
            BEGIN
                ;
                THROW 52000, N'Już jest takie miasto', 1
            END
        IF NOT EXISTS(
                SELECT *
                FROM project.Countries
                where CountryName = @CountryName
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma takiego państwa w bazie', 1
            END
        DECLARE @CountryID INT
        SELECT @CountryID = CountryID
        FROM project.Countries
        WHERE CountryName = @CountryName
        INSERT INTO project.Cities(CityName, CountryID)
        VALUES (@CityName, @CountryID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania nowego miasta: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
end
go
```

# AddCompany

Dodaje nową firmę, najpierw uzupełnia dane w tabeli Clients, a potem w Companies, przyjmuje w argumentach numer telefonu, e-mail, nazwę firmy, NIP, nazwę miasta, nazwę ulicy, kod pocztowy

```
CREATE PROCEDURE AddCompany @Phone varchar(16),
                            @Email varchar(255),
                            @CompanyName varchar(255),
                            @NIP varchar(255),
                            @City varchar(255),
                            @Street varchar(255),
                            @PostalCode varchar(6)
 AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRAN
        BEGIN TRY
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Phone = @Phone
                )
                BEGIN
                    ;
                    THROW 52000, N'Numer telefonu jest już w bazie', 1
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Email = @Email
                )
                BEGIN
                    ;
                    THROW 52000, N'Email jest już w bazie', 1
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Companies
                    WHERE NIP = @NIP
                )
                BEGIN
                    ;
                    THROW 52000, N'Firma o podanym NIPie jest już w bazie', 1
                END
```

```
        IF NOT EXISTS(
                SELECT *
                FROM project.Cities
                WHERE CityName = @City
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma podanego miasta w bazie', 1
            END
        DECLARE @ClientID INT
        SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
        FROM project.Clients
        INSERT INTO project.Clients(ClientID, Phone, Email)
        VALUES (@ClientID, @Phone, @Email);
        DECLARE @CityID INT
        SELECT @CityID = CityID
        FROM project.Cities
        WHERE CityName = @City
        INSERT INTO project.Companies(CompanyID, CompanyName, NIP, CityID, Street, PostalCode)
        VALUES (@ClientID, @CompanyName, @NIP, @CityID, @Street, @PostalCode)
        COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania firmy: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# AddDiscount

Dodaje zniżkę

```sql
CREATE PROCEDURE addDiscount @ClientID int
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM project.IndividualClients WHERE @ClientID = ClientID)
        BEGIN
            ;
            THROW 52000, N'Taki klient nie istnieje', 1
        end
    IF ((@ClientID in (SELECT DISTINCT ClientID FROM project.Discounts)) AND (((SELECT DD.EndDate
                                                                                FROM project.Discounts
                                                                                    INNER JOIN project.DiscountDetails DD on DD.DiscountID = Discounts.DiscountID
                                                                                WHERE @ClientID = ClientID) >=
                                                                               GETDATE()) AND ((SELECT DD.Avaliable
                                                                                                FROM project.Discounts
                                                                                                        INNER JOIN project.DiscountDetails DD on DD.DiscountID = Discounts.DiscountID
                                                                                                WHERE @ClientID = ClientID) =
                                                                                               1)))
        BEGIN
            ;
            THROW 52000, N'Ten klient ma już aktywną zniżkę', 1
        end
    DECLARE @AvailableDiscounts int = 0;
    IF (EXISTS(SELECT * FROM project.ClientsWithAvailableSecondDiscount WHERE ClientID = @ClientID) AND
            (@ClientID not in (SELECT DISTINCT ClientID FROM project.Discounts)))
        Begin
            SELECT @AvailableDiscounts = @AvailableDiscounts + 1;
            INSERT INTO project.DiscountDetails(StartDate, EndDate, Avaliable, PercentageValue)
            VALUES (GETDATE(), DATEADD(DAY, 7, GETDATE()), 1,
                    (SELECT d.Value FROM project.DictionaryValues as d WHERE d.ValueID = 5))
        end
    ELSE
        IF EXISTS(SELECT *
                  FROM project.ClientsWithAvailableFirstDiscount
                  WHERE ClientID = @ClientID)
            Begin
                SELECT @AvailableDiscounts = @AvailableDiscounts + 1;
                INSERT INTO project.DiscountDetails(StartDate, EndDate, Avaliable, PercentageValue)
                VALUES (GETDATE(), null, 1,
                        (SELECT d.Value FROM project.DictionaryValues as d WHERE d.ValueID = 3))
            end

    IF (@AvailableDiscounts = 0)
        BEGIN
            ;
            THROW 52000, N'Klientowi nie przysługuje żadna zniżka', 1
        END
    ELSE
        BEGIN
            DECLARE @DiscountID int;
            SELECT @DiscountID = MAX(DiscountDetails.DiscountID) FROM project.DiscountDetails
            INSERT INTO project.Discounts(DiscountID, ClientID)
            VALUES (@DiscountID, @ClientID)
        end
end
go
```

# AddEmployee

Dodaje pracownika do tabeli Employees łącząc go z odpowiednią firmą, przyjmuje w argumentach nazwę firmy, imię i nazwisko

```sql
CREATE OR ALTER PROCEDURE addEmployee @CompanyName varchar(255),
                                      @FirstName varchar(255),
                                      @LastName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
                SELECT *
                FROM project.Employees as E
                        INNER JOIN project.Companies C on E.CompanyID = C.CompanyID
                WHERE @FirstName = E.FirstName
                  AND @LastName = E.LastName
                  AND @CompanyName = C.CompanyName
            )
            BEGIN
                ;
                THROW 52000, N'Pracownik jest już w bazie', 1
            END
        IF NOT EXISTS(
                SELECT *
                FROM project.Companies
                WHERE @CompanyName = CompanyName
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej firmy', 1
            end
        DECLARE @CompanyID INT
        SELECT @CompanyID = CompanyID
        FROM project.Companies
        WHERE @CompanyName = CompanyName
        INSERT INTO project.Employees(CompanyID, FirstName, LastName)
        VALUES (@CompanyID, @FirstName, @LastName)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania pracownika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# AddEmployeeReservation

Dodaje rezerwację wraz z zamówieniem, jako argumenty przyjmuje listę produktów, ID płatności, typ płatności (jeżeli jest null to tworzy nową płatność), czas rozpoczęcia rezerwacji, czas zakończenia rezerwacji, listę pracowników

```
CREATE   PROCEDURE project.AddEmployeeReservation @ClientID int,
                                                  @Products project.OrderMeals readonly ,
                                                  @PaymentID int = null,
                                                  @TypeOfPayment varchar(255),
                                                  @ReservationStartDate datetime,
                                                  @ReservationEndDate datetime,
                                                  @Employees ListOfIDs readonly
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @Name varchar(20) = 'AddOrderRN'
    BEGIN TRAN @Name
        SAVE TRAN @Name
        BEGIN TRY
            DECLARE @CanAddReservation BIT
            SET @CanAddReservation = project.canClientHaveReservation( @ClientID: @ClientID,  @Products: @Products)
            IF @CanAddReservation = 0
                BEGIN
                    THROW 52000, N'Klient nie może złożyć rezerwacji', 1
                END
            DECLARE @ReservationID int;
            DECLARE @ReservationSize int;
            SELECT @ReservationSize = COUNT(*) SELECT * FROM @Employees
            EXECUTE project.AddReservation  @ReservationStartDate: @ReservationStartDate,  @ReservationEndDate: @ReservationEndDate,  @ReservationSize: @ReservationSize,
                    @ReservationID = @ReservationID
            IF NOT EXISTS(SELECT * FROM @Products)
                BEGIN
                    EXECUTE project.AddOrder  @ClientID: @ClientID,  @Products: @Products,  @PaymentID: @PaymentID,  @TypeOfPayment: @TypeOfPayment,  @IsTakeaway: 0,  @TakeawayTime: null,  @ReservationID: @ReservationID
                END
            DECLARE @EmployeeID as int
            DECLARE EmployeeCursor CURSOR FOR
                SELECT * FROM @Employees;
            OPEN EmployeeCursor;
            FETCH NEXT FROM EmployeeCursor INTO @EmployeeID;
            WHILE @@FETCH_STATUS = 0
                BEGIN
                    EXECUTE project.AddEmployeeToReservation  @ReservationID: @ReservationID,  @EmployeeID: @EmployeeID
                    FETCH NEXT FROM EmployeeCursor INTO @EmployeeID;
                END;
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go
```

# AddEmployeeToReservation

Przypisuje pracownika do rezerwacji firmy, przyjmuje w argumentach ID rezerwacji i ID pracownika

```
CREATE   PROCEDURE project.AddEmployeeToReservation @ReservationID int,
                                                    @EmployeeID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        Declare @CompanyID int

        SELECT @CompanyID = CompanyID FROM project.ComapnyReservations WHERE ReservationID = @ReservationID
        IF NOT EXISTS(SELECT * FROM project.Reservations WHERE ReservationID = @ReservationID)
            BEGIN
                THROW 52000, N'Taka rezerwacja nie isnieje', 1
            end
        IF NOT EXISTS(SELECT * FROM project.Employees WHERE EmployeeID = @EmployeeID)
            BEGIN
                THROW 52000, N'W bazie nie ma takiego pracownika', 1
            end
        INSERT INTO project.CompanyPersonalReservation(EmployeeID, ReservationID)
        VALUES (@EmployeeID, @ReservationID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania zamówienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

## AddIndividialClient

Dodaje nowego klienta indywidualnego, najpierw uzupełnia dane w tabeli Clients, a potem w IndividualClients, przyjmuje w argumentach numer telefonu, e-mail, imię i nazwisko

```
CREATE PROCEDURE AddIndividualClient @Phone varchar(16),
                                     @Email varchar(255),
                                     @FirstName varchar(255),
                                     @LastName varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRAN
        BEGIN TRY
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Phone = @Phone
                )
                BEGIN
                    ;
                    THROW 52000, N'Numer telefonu jest już w bazie', 1
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Email = @Email
                )
                BEGIN
                    ;
                    THROW 52000, N'Email jest już w bazie', 1
                END
            DECLARE @ClientID INT
            SELECT @ClientID = ISNULL(MAX(ClientID), 0) + 1
            FROM project.Clients
            INSERT INTO project.Clients(ClientID, Phone, Email)
            VALUES (@ClientID, @Phone, @Email);
            INSERT INTO project.IndividualClients(ClientID, FirstName, LastName)
            VALUES (@ClientID, @FirstName, @LastName)
            COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodania klienta indywidualnego: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
END
go
```

# AddMeal

Dodaje nowy posiłek do tabeli Meals, przyjmuje w argumentach nazwę posiłku, nazwę kategorii, opis oraz liczbę jednostek na stanie

```sql
CREATE OR ALTER PROCEDURE addMeal @Name varchar(255),
                                  @CategoryName varchar(255),
                                  @Description varchar(255),
                                  @UnitInStock int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
            )
            BEGIN
                ;
                THROW 52000, N'Potrawa jest już dodana', 1
            END
        IF NOT EXISTS(
                SELECT *
                FROM project.Categories
                WHERE CategoryName = @CategoryName
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej kategorii', 1
            END
        DECLARE @CategoryID INT
        SELECT @CategoryID = CategoryID
        FROM project.Categories
        WHERE CategoryName = @CategoryName
        DECLARE @MealID INT
        SELECT @MealID = ISNULL(MAX(MealID), 0) + 1
        FROM project.Meals
        INSERT INTO project.Meals(Name, MealID, CategoryID, Description, UnitInStock)
        VALUES (@Name, @MealID, @CategoryID, @Description, @UnitInStock);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
go
```

# AddMealToMenu

Dodaje istniejący posiłek do aktualnego menu, przyjmuje w argumentach nazwę posiłku, cenę oraz datę dodania posiłku do aktualnego menu

```
CREATE   PROCEDURE project.AddMealToMenu @Name varchar(255),
                                         @Price money,
                                         @StartDate varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej potrawy', 1
            END

        DECLARE @MealID INT
        SELECT @MealID = MealID
        FROM project.Meals
        WHERE Name = @Name

        IF EXISTS(
                SELECT *
                FROM project.Menus
                WHERE MealID = @MealID
                  AND (RemoveFromMenuDate IS NULL OR RemoveFromMenuDate > @StartDate)
            )
            BEGIN
                ;
                THROW 52000, N'W menu istnieje już taka potrawa', 1
            END

        DECLARE @MenuID INT
        SELECT @MenuID = ISNULL(MAX(MenuID), 0) + 1
        FROM project.Menus
        INSERT INTO project.Menus(MenuID, MealID, AddToMenuDate, RemoveFromMenuDate, UnitPrice)
        VALUES (@MenuID, @MealID, @StartDate, null, @Price);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy do menu: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# AddMealToOrder

Dodaje istniejący posiłek do zamówienia, przyjmuje w argumentach ID zamówienia, ilość zamówionych pozycji

```
CREATE   PROCEDURE project.AddMealToOrder @OrderID int,
                                          @Quantity int,
                                          @MenuID int
AS
BEGIN
    SET NOCOUNT ON
        BEGIN TRY
            IF project.isMenuItemAvailable( @MenuID: @MenuID) = 0
                BEGIN
                    THROW 52000, N'Ta potrawa nie jest aktualnie dostępna', 1
                END
            IF project.orderExists( @OrderID: @OrderID) = 0
                BEGIN
                    THROW 52000, N'Nie ma takiego zamówienia', 1
                end
            IF project.isMenuItemInStock( @MenuID: @MenuID,  @RequiredQuantity: @Quantity) = 0
                BEGIN
                    THROW 52000, N'Nie jest dostępna taka ilość', 1
                end
            DECLARE @MealID int
            SELECT @MealID = C.MealID
            FROM project.CurrentMenu C
            WHERE C.MenuID = @MenuID
            INSERT INTO project.OrderDetails(MenuID, OrderId, Quantity)
            VALUES (@MenuID, @OrderID, @Quantity)
            UPDATE project.Meals
            SET UnitInStock = UnitInStock - @Quantity
            WHERE MealID = @MealID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania produktu do zamówienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
END
go
```

# AddOrder

Dodaje nowe zamówienie do tablicy zamówień, jako argumenty przyjmuje ID klienta, ID płatności, wartość boolean czy zamówienie jest na wynos, czas odbioru, typ płatności (jeżeli jest null to tworzy nową płatność)

```
CREATE   PROCEDURE project.AddOrder @ClientID int,
                                    @Products project.OrderMeals readonly,
                                    @PaymentID int = null,
                                    @TypeOfPayment varchar(255) = null,
                                    @IsTakeaway bit,
                                    @TakeawayTime datetime = null,
                                    @ReservationID int
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @Name varchar(20) = 'AddOrder'
    BEGIN TRAN @Name
        SAVE TRAN @Name
        BEGIN TRY
            IF project.customerExists( @CustomerID: @ClientID) = 0
                BEGIN
                    THROW 52000, N'Taki klient nie istnieje!', 1
                END
            DECLARE @OrderDate datetime
            SET @OrderDate = GETDATE()
            DECLARE @ReservationStartDate datetime
            SET @ReservationStartDate = project.getStartDateOfReservation( @ReservationID: @ReservationID)
            IF (project.haveMealsSomeSeafood( @Meals: @Products) = 1) AND
               (project.canOrderHasSeafood( @OrderDate: @OrderDate, @ReservationDate: @ReservationStartDate, @TakeawayDate: @TakeawayTime) = 0)
                BEGIN
                    THROW 52000, N'Zamówienie zawiera owoce morza ale nie może zostać zrealizowane!', 1
                END
            DECLARE @VarPaymentID int;
            SET @VarPaymentID = @PaymentID
            IF @PaymentID IS NULL
                BEGIN
                    EXECUTE project.AddPayment @PaymentStatus: DEFAULT, @TypeOfPayment: @TypeOfPayment, @PaymentID = @VarPaymentID OUTPUT
                end
            DECLARE @TakeAwayID int
            IF @IsTakeaway = 1
                BEGIN
                    EXECUTE project.AddTakeaway @TakeawayTime: @TakeawayTime, @TakeawayID: @TakeAwayID
                END

            DECLARE @OrderID int
            SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1 FROM project.Orders
            INSERT INTO project.Orders(OrderID, ClientID, OrderDate, PaymentID, TakeawayID, ReservationID)
            VALUES (@OrderID, @ClientID, @OrderDate, @VarPaymentID, @TakeAwayID, @ReservationID)
            DECLARE @MenuID as int
            DECLARE @Quantity as int
            DECLARE ProductCursor CURSOR FOR
                SELECT * FROM @Products;
            OPEN ProductCursor;
            FETCH NEXT FROM ProductCursor INTO @MenuID, @Quantity;
            WHILE @@FETCH_STATUS = 0
                BEGIN
                    EXECUTE project.AddMealToOrder @OrderID: @OrderID, @Quantity: @Quantity, @MenuID: @MenuID
                    FETCH NEXT FROM ProductCursor INTO @MenuID, @Quantity;
                END;
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go
```

# AddOrderWithSizeReservation

Dodaje rezerwację wraz z zamówieniem, jako argumenty przyjmuje listę produktów, ID płatności, typ płatności (jeżeli jest null to tworzy nową płatność), czas rozpoczęcia rezerwacji, czas zakończenia rezerwacji, rozmiar rezerwacji

# AddPayment

Dodaje płatność, jako argumenty przyjmuje status płatności, typ, zwracając id płatności

```
CREATE   PROCEDURE project.AddPayment @PaymentStatus bit = 0,
                                      @TypeOfPayment varchar(255),
                                      @PaymentID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO project.PaymentDetails(PaymentStatus, TypeOfPayment)
        VALUES (@PaymentStatus, @TypeOfPayment)
        SELECT @PaymentID = MAX(PaymentID) FROM project.PaymentDetails
    end try
    begin catch
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania płatności: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    end catch
END
go
```

# AddReservation

Dodaje nową rezerwację do tabeli Reservations, przyjmuje jako argumenty ID klienta, datę rozpoczęcia rezerwacji, datę zakończenia rezerwacji, rozmiar rezerwacji

```
CREATE   PROCEDURE project.AddReservation @ReservationStartDate datetime,
                                          @ReservationEndDate datetime,
                                          @ReservationSize int,
                                          @ReservationID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        SELECT @ReservationID = ISNULL(MAX(ReservationID), 0) + 1 FROM project.Reservations
        INSERT INTO project.Reservations(ReservationID, StartDate, EndDate, Status, SizeOfReservation)
        VALUES (@ReservationID, @ReservationStartDate, @ReservationEndDate, 'pending', @ReservationSize)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
end
go
```

# AddTable

Dodaje nowy stolik do tabeli Tables, jako argument przyjmuje pojemność stolika

```
CREATE PROCEDURE addTable @Capacity int
 AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO project.Tables(Capacity)
        VALUES (@Capacity);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania stolika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
 go
```

# AddTableToReservation

Dodaje stolik do rezerwacji, jako argumenty przyjmuje ID rezerwacji, ID stolika

```
CREATE PROCEDURE project.AddTableToReservation @ReservationID int,
                                               @TableID int
 AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableID
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma stolika o podanym ID', 1
            END
        IF EXISTS(
                SELECT *
                FROM project.ReservationDetails
                WHERE TableID = @TableID
            )
            BEGIN
                ;
                THROW 52000, N'Ten stolik jest już przydzielony do tej rezerwacji', 1
            END

        DECLARE @StartDate DATE
        SELECT @StartDate = StartDate from project.Reservations R where R.ReservationID = @ReservationID
        DECLARE @EndDate DATE
        SELECT @EndDate = EndDate from project.Reservations R where R.ReservationID = @ReservationID

        IF @TableID not in (select * from project.getFreeTables( @startDate: @StartDate,  @endDate: @EndDate))
            BEGIN
                ;
                THROW 52000, N'Ten stolik jest zajęty w tych godzinach!', 1
            END

        BEGIN
            INSERT INTO project.ReservationDetails(ReservationID, TableID)
            VALUES (@ReservationID, @TableID)
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dopisania stolika do rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
 go
```

# AddTakeway

Dodaje odbiór do zamówienia. Przyjmuje datę odbioru i zwraca TakewayID

```
CREATE   PROCEDURE project.AddTakeaway @TakeawayTime datetime,
                                       @TakeawayID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO project.Takeaways(TakewayTime)
        VALUES (@TakeawayTime)
        SELECT @TakeawayID = MAX(TakeawayID) FROM project.Takeaways
    end try
    begin catch
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania danych odbioru zamówienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    end catch
END
go
```

# CancelReservation

```
CREATE OR ALTER PROCEDURE project.CancelReservation @ReservationID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF project.reservationExists( @ReservationID: @ReservationID) = 0
            BEGIN
                THROW 52000, N'Nie ma takiej rezerwacji', 1
            END
        IF EXISTS(SELECT * FROM project.Reservations WHERE ReservationID = @ReservationID AND Status = 'accepted')
            BEGIN
                THROW 52000, N'Rezerwacja już jest zaakceptowana', 1
            end
        IF EXISTS(SELECT *
                  FROM project.IndividualClientReservations
                  WHERE IndividualClientReservations.ReservationID = @ReservationID)
            BEGIN
                DELETE project.IndividualClientReservations WHERE ReservationID = @ReservationID
            END
        ELSE
            BEGIN
                DELETE project.ComapnyReservations WHERE ReservationID = @ReservationID
                DELETE project.CompanyPersonalReservation WHERE ReservationID = @ReservationID
            END
        DELETE project.Reservations WHERE ReservationID = @ReservationID
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd usuwania rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END catch
END
GO
```

# ChangePaymentStatus

Zmienia status płatności w tabeli PaymentDetails, jako argument przyjmuje ID zamówienia

```
CREATE OR ALTER PROCEDURE changePaymentStatus @OrderID int AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Orders
                WHERE OrderID = @OrderID
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma takiego zamówienia', 1
            END
        IF ((SELECT PaymentStatus
             FROM project.PaymentDetails
             WHERE PaymentID = (SELECT PaymentID
                                FROM project.Orders
                                WHERE OrderID = @OrderID)) = 1)
            BEGIN
                ;
                THROW 52000, N'Zamówienie jest już opłacone', 1
            END
        DECLARE @PaymentID int
        SELECT @PaymentID = PaymentID
        FROM project.Orders
        WHERE OrderID = @OrderID
        BEGIN
            UPDATE project.PaymentDetails
            SET PaymentStatus = 1
            WHERE PaymentID = @PaymentID
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd zmiany statusu platnosci: ' +
             ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# ChangeReservationStatus

Zmienia status rezerwacji na podany w argumencie, w argumentach przyjmuje ID rezerwacji, nowy status rezerwacji, ID stolika na wypadek zmienienia statusu rezerwacji z ‘pending’ na ‘accepted’

```
CREATE PROCEDURE ChangeReservationStatus @ReservationID int,
                                         @NewStatus varchar(32),
                                         @TableID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Reservations
                WHERE ReservationID = @ReservationID
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma rezerwacji o podanym ID', 1
            END
        IF (SELECT Status
            FROM project.Reservations
            WHERE ReservationID = @ReservationID) = @NewStatus
            BEGIN
                ;
                THROW 52000, N'Ta rezerwacja ma już taki status', 1
            END
        BEGIN
            UPDATE project.Reservations
            SET Status = @NewStatus
            WHERE ReservationID = @ReservationID
        END
        IF @NewStatus = 'accepted'
            BEGIN
                IF @TableID is null
                    BEGIN
                        ;
                        THROW 52000, N'Musisz podać id stolika', 1
                    END
            END
        BEGIN
```

```
            IF NOT EXISTS(
                    SELECT *
                    FROM project.Tables
                    WHERE TableNumber = @TableID
                )
                BEGIN
                    ;
                    THROW 52000, N'Nie ma stolika o podanym ID', 1
                END
        END
        BEGIN
            INSERT INTO project.ReservationDetails(ReservationID, TableID)
            VALUES (@ReservationID, @TableID)
        END
        -- CODE
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd edytowania statusu rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
GO
```

# EditTable

Zmienia ilość miejsc stolika (np jak się zepsuło krzesełko), w argumentach przyjmuje ID stolika oraz nową pojemność

```
CREATE OR ALTER PROCEDURE editTable @TableNumber int,
                                    @NewCapacity int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableNumber
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego stolika', 1
            END
        UPDATE project.Tables
        SET Capacity = @NewCapacity
        WHERE TableNumber = @TableNumber
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd usuwania stolika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# RemoveMealFromMenu

Usuwa posiłek z aktualnego menu czyli tak na prawdę updatuje kolumnę RemoveFromMenuDate w tabeli Menus, jako argumenty przyjmuje nazwę posiłku oraz datę usunięcia z menu

```
CREATE PROCEDURE removeMealFromMenu @Name varchar(255),
                                    @EndDate varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej potrawy', 1
            END

        DECLARE @MealID INT
        SELECT @MealID = MealID
        FROM project.Meals
        WHERE Name = @Name

        IF NOT EXISTS(
                SELECT *
                FROM project.Menus
                WHERE MealID = @MealID
                  AND RemoveFromMenuDate IS NULL
            )
            BEGIN
                ;
                THROW 52000, 'W menu nie ma takiej potrawy', 1
            END
        DECLARE @MenuID INT
        SELECT @MenuID = MenuID
        FROM project.Menus
        WHERE MealID = @MealID
          AND RemoveFromMenuDate IS NULL;
        UPDATE project.Menus
        SET RemoveFromMenuDate = @EndDate
        WHERE MealID = @MealID
          AND MenuID = @MenuID;
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy do menu: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# RemoveTable

Usuwa stolik z tabeli tables, jako argument przyjmuje ID stolika

```
CREATE PROCEDURE removeTable @TableNumber int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableNumber
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego stolika', 1
            END
        DELETE FROM project.Tables WHERE TableNumber = @TableNumber;
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd usuwania stolika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

# ShouldMenuBeChanged

Procedura wyrzucająca błąd, gdy menu powinno zostać zmienione (zgodnie z zasadą panującą w restauracji)

```
CREATE   PROCEDURE project.ShouldMenuBeChanged
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @MealsToChange FLOAT;
    DECLARE @MealsInMenu FLOAT;
    SELECT @MealsToChange = COUNT(*) FROM project.MealsLongerThanTwoWeeksInMenu
    SELECT @MealsInMenu = COUNT(*) FROM project.CurrentMenu
    DECLARE @Percentage FLOAT;
    SELECT @Percentage = @MealsToChange/@MealsInMenu;
    if(@Percentage>=0.5)
    BEGIN
        ;
            THROW 52000, N'Należy zmienić menu!' , 1
    end
END
go
```

# Funkcje

## canClientHaveReservation

Funkcja informuje czy klient może mieć rezerwację

```
CREATE   FUNCTION project.canClientHaveReservation(@ClientID int, @Products project.OrderMeals readonly)
    RETURNS BIT AS
BEGIN
    IF (@ClientID IN (SELECT CompanyID FROM Companies))
        BEGIN
            RETURN 1
        end
    IF (@ClientID IN (SELECT ClientID FROM IndividualClients))
        BEGIN
            DECLARE @WZ as int;
            SET @WZ = project.getActualWZ();
            DECLARE @Value as float;
            SET @Value = project.getValueOfProducts( @Products: @Products);
            DECLARE @WK as int;
            SET @WK = project.getActualWK();
            DECLARE @PreviousOrders as int;
            SELECT @PreviousOrders = COUNT(*) FROM Orders WHERE ClientID = @ClientID
            IF (@PreviousOrders >= @WK) AND (@Value >= @WZ)
                BEGIN
                    RETURN 1
                end
        end
    RETURN 0
END
go
```

# canOrderHasSeafood

Funkcja sprawdza czy można zamówić owoce morza

```
CREATE   FUNCTION project.canOrderHasSeafood(@OrderDate datetime, @ReservationDate datetime, @TakeawayDate datetime)
RETURNS BIT AS
    BEGIN
    DECLARE @Weekday int;
    DECLARE @Days int;
    IF @ReservationDate IS NOT NULL
        BEGIN
            -- Podana jest data rezerwacji
            SET @Days = DATEDIFF(DAY, @OrderDate, @ReservationDate)
            SET @Weekday = DATEPART(WEEKDAY, @ReservationDate)
        END
    IF @TakeawayDate IS NOT NULL
        BEGIN
            -- Podana jest data rezerwacji
            SET @Days = DATEDIFF(DAY, @OrderDate, @TakeawayDate)
            SET @Weekday = DATEPART(WEEKDAY, @TakeawayDate)
        END
    IF (@Weekday IN (5, 6, 7)) AND @Days > 2
        BEGIN
            RETURN 1
        END
    RETURN 0
    END
go
```

# checkIfClientHasActiveDiscount

Funkcja sprawdza czy klient ma aktywną zniżkę

```
create function project.checkIfClientHasActiveDiscount(@ClientID int)
    returns bit as
begin
    return case
               when (@ClientID in (SELECT DISTINCT ClientID
                                   FROM project.Discounts)) AND ((SELECT DD.EndDate
                                                                  FROM project.Discounts
                                                                           INNER JOIN project.DiscountDetails DD on DD.DiscountID = Discounts.DiscountID
                                                                  WHERE @ClientID = ClientID) >=
                                                                 GETDATE()) AND ((SELECT DD.Avaliable
                                                                                  FROM project.Discounts
                                                                                           INNER JOIN project.DiscountDetails DD on DD.DiscountID = Discounts.DiscountID
                                                                                  WHERE @ClientID = ClientID) =
                                                                                 1)
                   then 1
               else 0
      end
end
go
```

# customerExists

Funkcja sprawdza czy klient istnieje

```
CREATE   FUNCTION project.customerExists(@CustomerID int)
    RETURNS BIT AS
BEGIN
    IF @CustomerID IN (SELECT ClientID FROM project.Clients)
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# getActualWK

Funkcja zwraca aktualne wk

```
CREATE   FUNCTION project.getActualWK()
    RETURNS INT AS
BEGIN
    DECLARE @WK as int;
    SELECT @WK = WK FROM WZWK WHERE ValidTo IS NULL
    RETURN @WK
END
go
```

# getActualWZ

Funkcja zwraca aktualne wz

```
CREATE   FUNCTION project.getActualWZ()
    RETURNS INT AS
BEGIN
    DECLARE @WZ as int;
    SELECT @WZ = WZ FROM WZWK WHERE ValidTo IS NULL
    RETURN @WZ
END
go
```

# getFreeTables

Funkcja zwraca wolne stoliki

```
create function project.getFreeTables(@startDate datetime, @endDate datetime)
    returns table as
        return
        select TableNumber
        from project.Tables T
        where TableNumber not in (select TableID
                                  from project.ReservationDetails RD
                                           inner join project.Reservations R2 on R2.ReservationID = RD.ReservationID
                                  where (@startDate < R2.EndDate and @startDate > R2.StartDate) or (@endDate > R2.StartDate and @endDate < R2.EndDate))
go
```

# getStartDateOfReservation

Funkcja zwraca datę początkową rezerwacji

```
CREATE   FUNCTION   project.getStartDateOfReservation(@ReservationID int)
RETURNS DATETIME AS
    BEGIN
    RETURN (SELECT TOP 1 StartDate FROM
    project.Reservations R
    WHERE R.ReservationID = @ReservationID)
    END
go
```

# getValueOfProducts

Funkcja zwraca wartość produktów

```
CREATE   FUNCTION project.getValueOfProducts(@Products project.OrderMeals readonly)
    RETURNS FLOAT AS
BEGIN
    DECLARE @Value as float;
    SELECT @Value = Quantity * Menus.UnitPrice
    FROM @Products P
             INNER JOIN project.Menus Menus ON Menus.MenuID = P.MenuID
    RETURN ROUND(@Value, 2)
END
go
```

# haveMealsSomeSeafood

Funkcja zwraca czy dania zawierają jakieś owoce morza

```
CREATE   FUNCTION project.haveMealsSomeSeafood(@Meals project.OrderMeals readonly)
    RETURNS BIT AS
BEGIN
    DECLARE @MenuID as int
    DECLARE ProductCursor CURSOR FOR SELECT MenuID FROM @Meals;
    OPEN ProductCursor;
    FETCH NEXT FROM ProductCursor INTO @MenuID;
    WHILE @@FETCH_STATUS = 0
        BEGIN
            IF project.isSeaFoodMeal( @MenuID: @MenuID) = 1
                BEGIN
                    RETURN 1
                END
            FETCH NEXT FROM ProductCursor INTO @MenuID;
        END;
    RETURN 0;
END
go
```

# isMenuItemAvailable

Funkcja zwraca czy danie jest dostępne w menu

```
CREATE   FUNCTION project.isMenuItemAvailable(@MenuID int)
    RETURNS BIT AS
BEGIN
    IF @MenuID IN (SELECT MenuID FROM project.CurrentMenu)
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# isMenuItemInStock

Funkcja czy dana potrawa z menu jest jeszcze na stanie

```
CREATE   FUNCTION project.isMenuItemInStock(@MenuID int, @RequiredQuantity int)
    RETURNS BIT AS
BEGIN
    DECLARE @AvailableQuantity int
    SELECT @AvailableQuantity = UnitInStock
    FROM project.CurrentMenu
    WHERE @MenuID = MenuID
    IF @AvailableQuantity >= @RequiredQuantity
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# isSeaFoodMeal

Funkcja zwraca czy produkt jest owocem morza

```
CREATE   FUNCTION project.isSeaFoodMeal(@MenuID int)
    RETURNS BIT AS
BEGIN
    DECLARE @MealID int;
    SELECT @MealID = MealID FROM project.Menus WHERE MenuID = @MenuID
    IF (@MealID IN (SELECT MealID FROM project.SeafoodMeals))
        BEGIN
            RETURN 1
        end
    RETURN 0
END
go
```

# mealExists

Funkcja zwraca czy posiłek istnieje

```
CREATE   FUNCTION project.mealExists(@MealID int)
    RETURNS BIT AS
BEGIN
    IF @MealID IN (SELECT MealID FROM project.Meals)
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# orderExists

Funkcja zwraca czy zamówienie istnieje

```
CREATE   FUNCTION project.orderExists(@OrderID int)
    RETURNS BIT AS
BEGIN
    IF @OrderID IN (SELECT OrderID FROM project.Orders)
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# reservationExists

Funkcja zwraca czy rezerwacja istnieje

```
CREATE   FUNCTION project.reservationExists(@ReservationID int)
    RETURNS BIT AS
BEGIN
    IF @ReservationID IN (SELECT ReservationID FROM project.Reservations)
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go
```

# Triggery

## PROPERDICTIONARYVARIABLESCHECK

sprawdza, czy dodawane wartości do tabeli DictionaryValues są większe od 0

```
create or alter trigger project.PROPERDICTIONARYVARIABLESCHECK
    on project.DictionaryValues
    for insert
    as
BEGIN
    if (select Value from inserted) <= 0
        BEGIN
            RAISERROR ('Wprowadzono niepoprawny rekord', 16, 1)
            ROLLBACK TRANSACTION
        END
END
go
```

## PROPERWZWZCHECK

sprawdza, czy dodawane wartości do tabeli WZWK są większe od 0

```
create or alter trigger project.PROPERWZWKCHECK
    on project.WZWK
    for insert
    as
BEGIN
    if ((select WZ from inserted) <= 0) or ((select WK from inserted) <= 0)
        BEGIN
            RAISERROR ('Wprowadzono niepoprawny rekord', 16, 1)
            ROLLBACK TRANSACTION
        END
END
go
```

# DELETEORDERDETAILS

usuwa order details

```
create or alter trigger project.DELETEORDERDETAILS
    on project.OrderDetails
    for delete
    as
BEGIN
    DECLARE @MealID as int;
    DECLARE @MenuID as int;
    DECLARE @Quantity as int;
    DECLARE @OrderID as int;
    DECLARE CursorDetails CURSOR FOR SELECT MenuID, OrderID, Quantity FROM deleted;
    OPEN CursorDetails;
        FETCH NEXT FROM CursorDetails INTO @MenuID, @OrderID, @Quantity;
    WHILE @@FETCH_STATUS = 0
        BEGIN
            SELECT @MealID = Meals.MealID
            FROM project.Meals
                INNER JOIN Menus M on Meals.MealID = M.MealID
            WHERE M.MenuID = @MenuID
            UPDATE project.Meals
            SET UnitInStock = UnitInStock + @Quantity
            WHERE MealID = @MealID
            FETCH NEXT FROM CursorDetails INTO @MenuID, @OrderID, @Quantity;
        END;
END
go
```

# DELETERESERVATION

usuwa rezerwację

```
create or alter trigger project.DELETERESERVATION
    on project.IndividualClientReservations
    for delete
    as
BEGIN
    DECLARE @OrderID as int;
    DECLARE @PaymentID as int;
    SELECT @OrderID = OrderID FROM deleted;
    SELECT @PaymentID = PaymentID FROM project.Orders WHERE Orders.OrderID = @OrderID;
    DELETE project.OrderDetails WHERE @OrderID = OrderId
    DELETE project.Orders WHERE OrderID = @OrderID
    DELETE project.PaymentDetails WHERE PaymentID = @PaymentID;
END
go
```

# Indeksy

# Categories_pk

Ustawienie indeksu na CategoryID w tabeli Categories

```
create clustered index Categories_pk
on project.Categories(CategoryID)
```

# Clients_pk

Ustawienie indeksu na ClientID w tabeli Clients

```
create clustered index Clients_pk
on project.Clients(ClientID)
```

# CompanyReservations_pk

Ustawienie indeksu na ReservationID w tabeli CompanyReservations

```
create clustered index CompanyReservations_pk
on project.ComapnyReservations(ReservationID)
```

# Companies_pk

Ustawienie indeksu na CompanyID w tabeli Companies

```
create clustered index Companies_pk
on project.Companies(CompanyID)
```

# CompanyPersonalReservation_pk

Ustawienie indeksu na ReservationID w tabeli CompanyPersonalReservation

```
create clustered index CompanyPersonalReservation_pk
on project.CompanyPersonalReservation(ReservationID)
```

# Discounts_pk

Ustawienie indeksu na DiscountID w tabeli Discounts

```
create clustered index Discounts_pk
on project.Discounts(DiscountID)
```

# Employees_pk

Ustawienie indeksu na EmployeeID w tabeli Employees

```
CREATE CLUSTERED INDEX Employees_pk
on project.Employees(EmployeeID)
```

# IndividualClients_pk

Ustawienie indeksu na ClientID w tabeli IndividualClients

```
CREATE CLUSTERED INDEX IndividualClients_pk
on project.IndividualClients(ClientID)
```

# Meals_pk

Ustawienie indeksu na MealID w tabeli Meals

```
CREATE CLUSTERED INDEX Meals_pk
on project.Meals(MealID)
```

# Menus_pk

Ustawienie indeksu na MenuID w tabeli Menus

```
CREATE CLUSTERED INDEX Menus_pk
on project.Menus(MenuID)
```

# Orders_pk

Ustawienie indeksu na OrderID w tabeli Orders

```
CREATE CLUSTERED INDEX Orders_pk
on project.Orders(OrderID)
```

# PaymentDetails_pk

Ustawienie indeksu na PaymentID w tabeli PaymentDetails

```
CREATE CLUSTERED INDEX PaymentDetails_pk
on project.PaymentDetails(PaymentID)
```

# Reservations_pk

Ustawienie indeksu na ReservationID w tabeli Reservations

```
CREATE CLUSTERED INDEX Reservations_pk
on project.Reservations(ReservationID)
```

# Tables_pk

Ustawienie indeksu na TableNumber w tabeli Tables

```
CREATE CLUSTERED INDEX Tables_pk
on project.Tables(TableNumber)
```

# Takeways_pk

Ustawienie indeksu na TakewaysID w tabeli Takeways

```
CREATE CLUSTERED INDEX Takeaways_pk
on project.Takeaways(TakeawayID)
```

# EmployeesName

Ustawienie indeksu na FirstName i LastName w tabeli Employees

```
create nonclustered index EmployeesName
    on project.Employees (FirstName, LastName)
```

# IndividualClientsName

Ustawienie indeksu na FirstName i LastName w tabeli IndividualClients

```
create nonclustered index IndividualClientsName
    on project.IndividualClients (FirstName, LastName)
```

# CompaniesName

Ustawienie indeksu na ConpanyName w tabeli Companies

```
create nonclustered index CompaniesName
    on project.Companies (CompanyName)
```

# CompaniesNIP

Ustawienie indeksu na NIP w tabeli Companies

```
create nonclustered index CompaniesNIP
    on project.Companies (NIP)
```

# ClientPhone

Ustawienie indeksu na Phone w tabeli Clients

```
create nonclustered index ClientPhone
    on project.Clients (Phone)
```

# ClientEmail

Ustawienie indeksu na Email w tabeli Clients

```
create nonclustered index ClientEmail
    on project.Clients (Email)
```

# MealsName

Ustawienie indeksu na MealName w tabeli Meals

```
create nonclustered index MealsName
    on project.Meals (Name)
```

# OrdersReservationID

Ustawienie indeksu na ReservationID w tabeli Orders

```
create nonclustered index OrdersReservationID
    on project.Orders (ReservationID)
```

# OrdersPaymentID

Ustawienie indeksu na OrderPaymentID w tabeli Orders

```
create nonclustered index OrdersPaymentID
    on project.Orders (PaymentID)
```

# OrdersTakeawayID

Ustawienie indeksu na TakewayID w tabeli Orders

```
create nonclustered index OrdersTakeawayID
    on project.Orders (TakeawayID)
```

# CategoriesName

Ustawienie indeksu na CategoryName w tabeli Categories

```
create nonclustered index CategoriesName
    on project.Categories (CategoryName)
```

# CitiesName

Ustawienie indeksu na CityName w tabeli Cities

```
create nonclustered index CitiesName
    on project.Cities (CityName)
```

# PaymentDetailsStatus

Ustawienie indeksu na PaymentStatus w tabeli PaymentDetails

```
create nonclustered index PaymentDetailsStatus
    on project.PaymentDetails (PaymentStatus)
```

# ReservationsStatus

Ustawienie indeksu na ResrevationStatus w tabeli Resertavions

```
create nonclustered index ReservationsStatus
    on project.Reservations (Status)
```

# TablesCapacity

Ustawienie indeksu na Capacity w tabeli Tables

```
create nonclustered index TablesCapacity
    on project.Tables (Capacity)
```

# CountriesName

Ustawienie indeksu na CountryName w tabeli Countries

```
create nonclustered index CountriesName
    on project.Countries (CountryName)
```

# Uprawnienia ról w bazie danych

| Rola | Opis |
|---|---|
| **Pracownik** | przyjmuje, akceptuje, aktualizuje zamówienia (online oraz stacjonarne); dodaje produkty do menu; dodaje stoliki; obsługuje rezerwacje; sprawdza czy menu jest zmienione odpowiednio w czasie; generuje raporty |
| **Moderator** | ma dostęp do wszystkich rzeczy co pracownik; dodaje kategorie; zarządza profilami firm i użytkowników; nadaje zniżki |
| **Administrator** | ma dostęp do każdej funkcji systemu |

# Pracownik

```
create role worker

grant select on project.ClientsWithAvailableFirstDiscount to worker
grant select on project.ClientsWithAvailableSecondDiscount to worker
grant select on project.CompaniesSummaryMonthly to worker
grant select on project.CompaniesSummaryOverall to worker
grant select on project.CompaniesSummaryWeekly to worker
grant select on project.CurrentMenu to worker
grant select on project.DiscountsMonthlyStat to worker
grant select on project.DiscountsWeeklyStat to worker
grant select on project.IndividualClientsSummaryMonthly to worker
grant select on project.IndividualClientsSummaryOverall to worker
grant select on project.IndividualClientsSummaryWeekly to worker
grant select on project.MealsIncomeStats to worker
grant select on project.MealsLongerThanTwoWeeksInMenu to worker
grant select on project.MealsSoldStatsMonthly to worker
grant select on project.MealsSoldStatsOverall to worker
grant select on project.MonthlyIncome to worker
grant select on project.NotUsedMealsInStock to worker
grant select on project.OrderSummaries to worker
grant select on project.PendingReservations to worker
grant select on project.ReservationsInfo to worker
grant select on project.SeafoodMeals to worker
grant select on project.SeafoodMealsStats to worker
grant select on project.TablesMonthly to worker
grant select on project.TablesWeekly to worker
grant select on project.WeeklyIncome to worker
```

```
grant execute on project.AddDiscount to worker
grant execute on project.AddMeal to worker
grant execute on project.AddMealToMenu to worker
grant execute on project.AddORder to worker
grant execute on project.AddOrderWithSizeReservation to worker
grant execute on project.AddReservation to worker
grant execute on project.AddTable to worker
grant execute on project.AddTableToReservation to worker
grant execute on project.ChangePaymentStatus to worker
grant execute on project.ChangeReservationStatus to worker
grant execute on project.EditTable to worker
grant execute on project.RemoveMealFromMenu to worker
grant execute on project.RemoveTable to worker
grant execute on project.ShouldMenuBeChanged to worker
```

# Moderator

```
create role moderator

grant select on project.ClientsWithAvailableFirstDiscount to moderator
grant select on project.ClientsWithAvailableSecondDiscount to moderator
grant select on project.CompaniesSummaryMonthly to moderator
grant select on project.CompaniesSummaryOverall to moderator
grant select on project.CompaniesSummaryWeekly to moderator
grant select on project.CurrentMenu to moderator
grant select on project.DiscountsMonthlyStat to moderator
grant select on project.DiscountsWeeklyStat to moderator
grant select on project.IndividualClientsSummaryMonthly to moderator
grant select on project.IndividualClientsSummaryOverall to moderator
grant select on project.IndividualClientsSummaryWeekly to moderator
grant select on project.MealsIncomeStats to moderator
grant select on project.MealsLongerThanTwoWeeksInMenu to moderator
grant select on project.MealsSoldStatsMonthly to moderator
grant select on project.MealsSoldStatsOverall to moderator
grant select on project.MonthlyIncome to moderator
grant select on project.NotUsedMealsInStock to moderator
grant select on project.OrderSummaries to moderator
grant select on project.PendingReservations to moderator
grant select on project.ReservationsInfo to moderator
grant select on project.SeafoodMeals to moderator
grant select on project.SeafoodMealsStats to moderator
grant select on project.TablesMonthly to moderator
grant select on project.TablesWeekly to moderator
grant select on project.WeeklyIncome to moderator

grant execute on project.AddDiscount to moderator
grant execute on project.AddMeal to moderator
grant execute on project.AddMealToMenu to moderator
grant execute on project.AddOrder to moderator
grant execute on project.AddOrderWithSizeReservation to moderator
grant execute on project.AddReservation to moderator
grant execute on project.AddTable to moderator
grant execute on project.AddTableToReservation to moderator
grant execute on project.ChangePaymentStatus to moderator
grant execute on project.ChangeReservationStatus to moderator
grant execute on project.EditTable to moderator
grant execute on project.RemoveMealFromMenu to moderator
grant execute on project.RemoveTable to moderator
grant execute on project.ShouldMenuBeChanged to moderator

grant execute on project.addCategory to moderator
grant execute on project.AddCity to moderator
grant execute on project.AddCompany to moderator
grant execute on project.AddEmployee to moderator
grant execute on project.AddIndividualClient to moderator
```

## Administrator

```
create role administrator

grant all privileges on u_usnarski.project to administrator
```

## Wygenerowany SQL

```
create type ListOfIDs as table
(
   ID int
)
go

create type OrderMeals as table
(
   MenuID   int,
   Quantity int
)
go

create table Categories
(
   CategoryID   int           not null
       constraint Categories_pk
           primary key,
   CategoryName varchar(255) not null
)
go

create index CategoriesName
   on Categories (CategoryName)
```

```
go

create table Clients
(
   ClientID int          not null
       constraint Clients_pk
           primary key,
   Phone    varchar(16)  not null
       constraint phone_check
           check ([Phone] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
),
   Email    varchar(255) not null
       constraint email_pk
           unique
       constraint email_check
           check ([Email] like '%@%.%')
)
go

create unique index UniquePhone
   on Clients (Phone)
go

create unique index UniqueEmail
   on Clients (Email)
go

create index ClientPhone
   on Clients (Phone)
go

create index ClientEmail
   on Clients (Email)
go
```

```
create table Countries
(
   CountryID   int identity
       constraint Countries_pk
           primary key,
   CountryName varchar(255) not null
)
go

create table Cities
(
   CityID    int identity
       constraint Cities_pk
           primary key,
   CityName  varchar(255) not null,
   CountryID int
       constraint FK_Cities_Countries
           references Countries
)
go

create index CitiesName
   on Cities (CityName)
go

create table Companies
(
   CompanyID   int           not null
       constraint Companies_pk
           primary key
       constraint [CompanyID:ClientID]
           references Clients,
   CompanyName varchar(255) not null,
   NIP         varchar(255)
       constraint nip_pk
           unique
```

```
        constraint nipe_check
            check ([NIP] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][
0-9]'),
    CityID      int          not null
        constraint FK_Companies_Cities
            references Cities,
    Street      varchar(255) not null,
    PostalCode  varchar(6)   not null
        constraint postal_code_check
            check ([PostalCode] like
'[0-9][0-9]-[0-9][0-9][0-9]')
)
go

create unique index UniqueNIP
    on Companies (NIP)
go

create index CompaniesName
    on Companies (CompanyName)
go

create index CompaniesNIP
    on Companies (NIP)
go

create index CountriesName
    on Countries (CountryName)
go

create table DictionaryValues
(
    ValueID int identity
        constraint SłownikValues_pk
            primary key,
```

```
    Value   int not null
)
go

create table Dictionary
(
    ID   int identity
        constraint Słownik_pk
            primary key
        constraint
FK_Dictionary_DictionaryValues
            references DictionaryValues,
    Name varchar(255) not null
)
go

create   trigger
project.PROPERDICTIONARYVARIABLESCHECK
    on project.DictionaryValues
    for insert
    as
BEGIN
    if (select Value from inserted) <= 0
        BEGIN
            RAISERROR ('Wprowadzono niepoprawny
rekord', 16, 1)
            ROLLBACK TRANSACTION
        END
END
go

create table DiscountDetails
(
    DiscountID      int identity
        constraint DiscountDetails_pk
            primary key,
```

```
    StartDate       datetime      not null,
    EndDate         datetime,
    Avaliable       bit default 1 not null,
    PercentageValue int           not null
        constraint discount_check
            check ([PercentageValue] > 0 AND
[PercentageValue] <= 100),
    constraint date_check
        check ([EndDate] > [StartDate])
)
go

create table Employees
(
    EmployeeID int identity
        constraint Employees_pk
            primary key,
    CompanyID  int          not null
        constraint FK_Employees_Companies
            references Companies,
    FirstName  varchar(255) not null,
    LastName   varchar(255) not null
)
go

create index EmployeesName
    on Employees (FirstName, LastName)
go

create table IndividualClients
(
    ClientID  int          not null
        constraint IndividualClients_pk
            primary key
        constraint [ClientID:ClientID]
            references Clients,
```

```
    FirstName varchar(255) not null,
    LastName  varchar(255) not null
)
go

create table Discounts
(
    DiscountID int not null
        constraint FK_Discounts_DiscountDetails
            references DiscountDetails,
    ClientID   int not null
        constraint
FK_Discounts_IndividualClients
            references IndividualClients,
    constraint Discounts_pk
        primary key (DiscountID, ClientID)
)
go

create index IndividualClientsName
    on IndividualClients (FirstName, LastName)
go

create table Meals
(
    Name        varchar(255) not null
        constraint name_pk
            unique,
    MealID      int          not null
        constraint Meals_pk
            primary key,
    CategoryID  int          not null
        constraint FK_Meals_Categories
            references Categories,
    Description varchar(255),
    UnitInStock int          not null
```

```
        constraint unit_check
            check ([UnitInStock] >= 0)
)
go

create index MealsName
   on Meals (Name)
go

create table Menus
(
   MenuID              int not null
        constraint Menus_pk
            primary key,
   MealID              int not null
        constraint FK_Menus_Meals
            references Meals,
   AddToMenuDate       date,
   RemoveFromMenuDate date,
   UnitPrice           int not null
        constraint unit_price_check
            check ([UnitPrice] >= 0),
   constraint menu_date_check
        check ([RemoveFromMenuDate] >
[AddToMenuDate])
)
go

create table PaymentDetails
(
   PaymentID     int identity
        constraint PaymentDetails_pk
            primary key,
   PaymentStatus bit default 0 not null,
   TypeOfPayment varchar(255)  not null
        constraint payment_type_check
```

```
            check ([TypeOfPayment] = 'invoice'
OR [TypeOfPayment] = 'blik' OR [TypeOfPayment]
= 'paypal' OR
                   [TypeOfPayment] = 'card' OR
[TypeOfPayment] = 'cash')
)
go

create index PaymentDetailsStatus
   on PaymentDetails (PaymentStatus)
go

create table Reservations
(
   ReservationID     int          not null
       constraint Reservations_pk
           primary key,
   StartDate         datetime     not null
       constraint reservation_date_check_2
           check ([StartDate] > getdate()),
   EndDate           datetime     not null,
   Status            varchar(32) not null
       constraint reservation_status_check
           check ([Status] = 'finished' OR
[Status] = 'rejected' OR [Status] = 'accepted'
OR [Status] = 'pending'),
   SizeOfReservation int          not null
       constraint size_reservation_check
           check ([SizeOfReservation] >= 2),
   constraint reservation_date_check
       check ([EndDate] > [StartDate])
)
go

create table ComapnyReservations
(
```

```
   ReservationID int not null
       constraint ComapnyReservations_pk
           primary key
       constraint
[ReservationID:ReservationID(2)]
           references Reservations,
   CompanyID     int not null
       constraint
FK_ComapnyReservations_Companies
           references Companies
)
go

create table CompanyPersonalReservation
(
   EmployeeID    int not null
       constraint [EmplyeeID:EmployeeID]
           references Employees,
   ReservationID int not null
       constraint [ReservationID:ReservationID]
           references Reservations,
   constraint CompanyPersonalReservation_pk
       primary key (EmployeeID, ReservationID)
)
go

create index ReservationsStatus
   on Reservations (Status)
go

create table Tables
(
   TableNumber int identity
       constraint Tables_pk
           primary key,
   Capacity    int not null
```

```
        constraint capacity_check
            check ([Capacity] >= 1)
)
go

create table ReservationDetails
(
   ReservationID int not null
        constraint
ReservationDetails_Reservations_null_fk
            references Reservations,
   TableID       int not null
        constraint
ReservationDetails_Tables_null_fk
            references Tables,
   constraint ReservationDetails_pk
        primary key (ReservationID, TableID)
)
go

create index TablesCapacity
   on Tables (Capacity)
go

create table Takeaways
(
   TakeawayID  int identity
        constraint Takeaway_pk
            primary key,
   TakewayTime datetime not null
        constraint takeway_check
            check ([TakewayTime] > getdate())
)
go

create table Orders
```

```
(
   OrderID       int      not null
       constraint Orders_pk
           primary key,
   ClientID      int      not null
       constraint FK_Orders_Clients
           references Clients,
   OrderDate     datetime not null,
   PaymentID     int      not null
       constraint FK_Orders_PaymentDetails
           references PaymentDetails,
   TakeawayID    int
       constraint FK_Orders_Takeaways
           references Takeaways,
   ReservationID int
)
go

create table IndividualClientReservations
(
   ReservationID      int not null
       constraint
IndividualClientReservations_pk
           primary key,
   IndividualClientID int not null
       constraint
FK_IndividualClientReservations_IndividualClien
ts
           references IndividualClients,
   OrderID            int not null
       constraint
FK_IndividualClientReservations_Orders
           references Orders
)
go
```

```
create table OrderDetails
(
   MenuID   int not null
       constraint FK_OrderDetails_Menus
           references Menus,
   OrderId  int not null
       constraint OrderID_OrderID
           references Orders,
   Quantity int not null
       constraint quantity_check
           check ([Quantity] > 0),
   constraint OrderDetails_pk
       primary key (MenuID, OrderId)
)
go

create index OrdersReservationID
   on Orders (ReservationID)
go

create index OrdersPaymentID
   on Orders (PaymentID)
go

create index OrdersTakeawayID
   on Orders (TakeawayID)
go

create table WZWK
(
   WZ       int not null,
   WK       int not null,
   id       int identity
       constraint WZWK_pk
           primary key,
   ValidTo date
```

```
)
go

create   trigger project.PROPERWZWKCHECK
   on project.WZWK
   for insert
   as
BEGIN
   if ((select WZ from inserted) <= 0) or
((select WK from inserted) <= 0)
       BEGIN
           RAISERROR ('Wprowadzono niepoprawny
rekord', 16, 1)
           ROLLBACK TRANSACTION
       END
END
go

create view
project.ClientsWithAvailableFirstDiscount as
SELECT ClientID
FROM project.Clients AS C
WHERE (SELECT COUNT(underQuery.orderId)
      FROM (SELECT OrderID as orderId
            FROM project.Orders
            WHERE Orders.ClientID = C.ClientID
              AND (SELECT Value
                   FROM project.OrderSummaries
                   WHERE OrderSummaries.OrderID
= Orders.OrderID) >=
                  (SELECT dv.Value FROM
project.DictionaryValues as dv WHERE dv.ValueID
= 2)) underQuery) >
     (SELECT dv.Value FROM
project.DictionaryValues as dv WHERE dv.ValueID
= 1)
```

```
go

grant select on
ClientsWithAvailableFirstDiscount to moderator
go

grant select on
ClientsWithAvailableFirstDiscount to worker
go

create view
project.ClientsWithAvailableSecondDiscount as
SELECT ClientID
FROM project.Clients AS C
WHERE (SELECT SUM(underQuery.value)
      FROM (SELECT (SELECT Value
                    FROM project.OrderSummaries
                    WHERE
OrderSummaries.OrderID = Orders.OrderID) as
value
            FROM project.Orders
            WHERE Orders.ClientID = C.ClientID)
underQuery) >
     (SELECT dv.Value FROM
project.DictionaryValues as dv WHERE dv.ValueID
= 4)
go

grant select on
ClientsWithAvailableSecondDiscount to moderator
go

grant select on
ClientsWithAvailableSecondDiscount to worker
go
```

```
CREATE view project.CompaniesSummaryMonthly as
select ClientID, CompanyName, Year, Month,
COUNT(Month) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as
'Year', MONTH(OrderDate) as 'Month', (
   SELECT SUM(Quantity*UnitPrice)
   FROM project.OrderDetails OD
   INNER JOIN project.Menus M on M.MenuID =
OD.MenuID
   WHERE O.OrderID = OD.OrderId
   GROUP BY OD.OrderID) as 'Value'
from project.Orders O
) SAD
inner join project.Companies C ON C.CompanyID =
ClientID
group by ClientID, CompanyName, Year, Month
go

grant select on CompaniesSummaryMonthly to
moderator
go

grant select on CompaniesSummaryMonthly to
worker
go

create view project.CompaniesSummaryOverall as

select Comp.CompanyID,
      (select count(ASD.OrderID)
       from (select Orders.OrderId
             From project.Orders
             where ClientID = Comp.CompanyID)
ASD) as 'Orders',
      (select sum(DAS.Value)
```

```
      from (select Quantity * UnitPrice *
(ISNULL(1 - PercentageValue, 1)) as 'Value'
            From project.Orders O
                        inner join
project.OrderDetails OD on O.OrderID =
OD.OrderId
                        inner join project.Menus
M on M.MenuID = OD.MenuID
                        inner join project.Meals
M2 on M2.MealID = M.MealID
                        left join
project.Discounts D on O.ClientID = D.ClientID
                        left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
             where O.ClientID = Comp.CompanyID)
DAS) as 'Value'
FROM project.Companies Comp
where (select sum(DAS.Value)
      from (select Quantity * UnitPrice *
(ISNULL(1 - PercentageValue, 1)) as 'Value'
            From project.Orders O
                        inner join
project.OrderDetails OD on O.OrderID =
OD.OrderId
                        inner join project.Menus M
on M.MenuID = OD.MenuID
                        inner join project.Meals
M2 on M2.MealID = M.MealID
                        left join
project.Discounts D on O.ClientID = D.ClientID
                        left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
            where O.ClientID = Comp.CompanyID)
DAS) is not null
```

```
go

grant select on CompaniesSummaryOverall to
moderator
go

grant select on CompaniesSummaryOverall to
worker
go

CREATE view project.CompaniesSummaryWeekly as
select ClientID, CompanyName, Year, Week,
COUNT(Week) as 'Orders', SUM(Value) as 'Value'
FROM (select ClientID, YEAR(OrderDate) as
'Year', DATEPART(week, OrderDate) as 'Week', (
   SELECT SUM(Quantity*UnitPrice)
   FROM project.OrderDetails OD
   INNER JOIN project.Menus M on M.MenuID =
OD.MenuID
   WHERE O.OrderID = OD.OrderId
   group by OD.OrderID) as 'Value'
from project.Orders O
) SAD
inner join project.Companies C ON C.CompanyID =
ClientID
group by ClientID, CompanyName, Year, Week
go

grant select on CompaniesSummaryWeekly to
moderator
go

grant select on CompaniesSummaryWeekly to
worker
go
```

```
create view project.CurrentMenu as
select MenuID, Menus.MealID, Name, UnitPrice,
UnitInStock, Description, CategoryName,
AddToMenuDate from project.Menus
             inner join project.Meals M on
M.MealID = Menus.MealID
             inner join project.Categories C on
C.CategoryID = M.CategoryID
             where RemoveFromMenuDate IS null
go

grant select on CurrentMenu to moderator
go

grant select on CurrentMenu to worker
go

CREATE view project.DiscountsMonthlyStat as
select YEAR(StartDate) as 'Year',
MONTH(StartDate) as 'Month', COUNT(*) as
'CountOfDiscounts'
from project.DiscountDetails
group by YEAR(StartDate), MONTH(StartDate)
go

grant select on DiscountsMonthlyStat to
moderator
go

grant select on DiscountsMonthlyStat to worker
go

CREATE view project.DiscountsWeeklyStat as
select YEAR(StartDate) as 'Year',
DATEPART(week, StartDate) as 'Week',
      COUNT(*) as 'CountOfDiscounts'
```

```
from project.DiscountDetails
group by YEAR(StartDate), DATEPART(week,
StartDate)
go

grant select on DiscountsWeeklyStat to
moderator
go

grant select on DiscountsWeeklyStat to worker
go

CREATE view
project.IndividualClientsSummaryMonthly as
select IC.ClientID, FirstName, LastName, Year,
Month, COUNT(Month) as 'Orders',
ROUND(SUM(Value), 2) as 'Value'
FROM (select ClientID,
            YEAR(OrderDate)       as 'Year',
            MONTH(OrderDate)      as 'Month',
             (SELECT ROUND(SUM(Quantity *
UnitPrice *

ROUND((CAST(ISNULL(100 - PercentageValue, 100)
AS FLOAT) / CAST(100 AS FLOAT)), 2)), 2)
             FROM project.OrderDetails OD
                        INNER JOIN project.Menus
M on M.MenuID = OD.MenuID
                        left join
project.Discounts D on O.ClientID = D.ClientID
                        left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
             WHERE O.OrderID = OD.OrderId
             group by OD.OrderID) as 'Value'
     from project.Orders O) SAD
```

```
        inner join project.IndividualClients IC
ON IC.ClientID = SAD.ClientID
group by IC.ClientID, FirstName, LastName,
Year, Month
go

grant select on IndividualClientsSummaryMonthly
to moderator
go

grant select on IndividualClientsSummaryMonthly
to worker
go

CREATE view
project.IndividualClientsSummaryOverall as
select IC.ClientID,
      (select count(ASD.OrderID)
       from (select Orders.OrderId
             From project.Orders
             where ClientID = IC.ClientID) ASD)
as 'Orders',
      (select ROUND(sum(DAS.Value), 2)
       from (select Quantity * UnitPrice *
                    ROUND((CAST(ISNULL(100 -
PercentageValue, 100) AS FLOAT) / CAST(100 AS
FLOAT)), 2)
                         as 'Value'
             From project.Orders O
                        inner join
project.OrderDetails OD on O.OrderID =
OD.OrderId
                        inner join project.Menus
M on M.MenuID = OD.MenuID
                        inner join project.Meals
M2 on M2.MealID = M.MealID
```

```
                        left join
project.Discounts D on O.ClientID = D.ClientID
                        left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
             where O.ClientID = IC.ClientID)
DAS) as 'Value'
FROM project.IndividualClients IC
where (select ROUND(sum(DAS.Value), 2)
      from (select Quantity * UnitPrice *
                   ROUND((CAST(ISNULL(100 -
PercentageValue, 100) AS FLOAT) / CAST(100 AS
FLOAT)), 2)
                         as 'Value'
            From project.Orders O
                        inner join
project.OrderDetails OD on O.OrderID =
OD.OrderId
                        inner join project.Menus M
on M.MenuID = OD.MenuID
                        inner join project.Meals
M2 on M2.MealID = M.MealID
                        left join
project.Discounts D on O.ClientID = D.ClientID
                        left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
            where O.ClientID = IC.ClientID)
DAS) is not null
go

grant select on IndividualClientsSummaryOverall
to moderator
go
```

```
grant select on IndividualClientsSummaryOverall
to worker
go

CREATE view
project.IndividualClientsSummaryWeekly as
select IC.ClientID, FirstName, LastName, Year,
Week, COUNT(Week) as 'Orders', SUM(Value) as
'Value'
FROM (select ClientID,
             YEAR(OrderDate)           as
'Year',
             DATEPART(week, OrderDate) as
'Week',
             (SELECT ROUND(SUM(Quantity *
UnitPrice *

ROUND((CAST(ISNULL(100 - PercentageValue, 100)
AS FLOAT) / CAST(100 AS FLOAT)), 2)), 2)
              FROM project.OrderDetails OD
                       INNER JOIN project.Menus
M on M.MenuID = OD.MenuID
                       left join
project.Discounts D on O.ClientID = D.ClientID
                       left join
project.DiscountDetails DD on DD.DiscountID =
D.DiscountID
              WHERE O.OrderID = OD.OrderId
              group by OD.OrderID)     as
'Value'
     from project.Orders O) SAD
        inner join project.IndividualClients IC
ON IC.ClientID = SAD.ClientID
group by IC.ClientID, FirstName, LastName,
Year, Week
go
```

```
grant select on IndividualClientsSummaryWeekly
to moderator
go

grant select on IndividualClientsSummaryWeekly
to worker
go

CREATE view project.MealsIncomeStats as
select M2.MealID, M2.Name, sum(Quantity *
M.UnitPrice * ROUND((ISNULL(100 -
PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID
= OD.OrderId
inner join project.Menus M on M.MenuID =
OD.MenuID
   inner join project.Meals M2 on M2.MealID =
M.MealID
left join project.Discounts D on O.ClientID =
D.ClientID
left join project.DiscountDetails DD on
DD.DiscountID = D.DiscountID
group by M2.MealID, M2.Name
go

grant select on MealsIncomeStats to moderator
go

grant select on MealsIncomeStats to worker
go

create view
project.MealsLongerThanTwoWeeksInMenu as
```

```
SELECT m.MealID, DATEDIFF(DAY, m.AddToMenuDate,
getdate()) as 'How long in menu (days)'
FROM project.Menus as m
WHERE( m.RemoveFromMenuDate>=GETDATE() OR
m.RemoveFromMenuDate IS NULL)
AND DATEDIFF(DAY, m.AddToMenuDate,
getdate())>14
go

grant select on MealsLongerThanTwoWeeksInMenu
to moderator
go

grant select on MealsLongerThanTwoWeeksInMenu
to worker
go

create view project.MealsSoldStatsMonthly as
select year(OrderDate) as 'year',
month(OrderDate) as 'month', M.MealID, M.Name,
sum(Quantity) as 'amount' from project.Meals M
inner join project.Menus M2 on M.MealID =
M2.MealID
inner join project.OrderDetails OD on M2.MenuID
= OD.MenuID

inner join project.Orders O on O.OrderID =
OD.OrderId
group by M.MealID, M.Name, year(OrderDate),
month(OrderDate)
go

grant select on MealsSoldStatsMonthly to
moderator
go
```

```
grant select on MealsSoldStatsMonthly to worker
go

create view project.MealsSoldStatsOverall as
select M.MealID, M.Name, sum(Quantity) as
'amount' from project.Meals M
inner join project.Menus M2 on M.MealID =
M2.MealID
inner join project.OrderDetails OD on M2.MenuID
= OD.MenuID
group by M.MealID, M.Name
go

grant select on MealsSoldStatsOverall to
moderator
go

grant select on MealsSoldStatsOverall to worker
go

CREATE view project.MonthlyIncome as
select year(OrderDate) as 'year',
month(OrderDate) as 'month', sum(Quantity *
M.UnitPrice * ROUND((ISNULL(100 -
PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID
= OD.OrderId
inner join project.Menus M on M.MenuID =
OD.MenuID
left join project.Discounts D on O.ClientID =
D.ClientID
left join project.DiscountDetails DD on
DD.DiscountID = D.DiscountID
group by year(OrderDate), month(OrderDate)
go
```

```
grant select on MonthlyIncome to moderator
go

grant select on MonthlyIncome to worker
go

create view project.NotUsedMealsInStock as
select Name, MealID, UnitInStock
FROM project.Meals
where MealID NOT IN (select Menus.MealID
                     from project.Menus
                              inner join
project.Meals M on M.MealID = Menus.MealID
                     where RemoveFromMenuDate IS
NULL)
 and UnitInStock > 0
go

exec sp_addextendedproperty 'MS_Description',
'Dania które sa na magazynie ale nie w
aktualnym menu', 'SCHEMA',
    'project', 'VIEW', 'NotUsedMealsInStock'
go

grant select on NotUsedMealsInStock to
moderator
go

grant select on NotUsedMealsInStock to worker
go

CREATE view project.OrderSummaries as
select Orders.OrderID,
      ROUND(SUM(Quantity * UnitPrice *
```

```
                ROUND((CAST(ISNULL(100 -
PercentageValue, 100) AS FLOAT) / CAST(100 AS
FLOAT)), 2)), 2) as 'Value'
from project.Orders
        inner join project.OrderDetails OD on
Orders.OrderID = OD.OrderId
        inner join project.Menus M on M.MenuID
= OD.MenuID
        left join project.Discounts D on
Orders.ClientID = D.ClientID
        left join project.DiscountDetails DD on
DD.DiscountID = D.DiscountID
group by Orders.OrderID
go

grant select on OrderSummaries to moderator
go

grant select on OrderSummaries to worker
go

create view project.CurrentReservations as
select * from project.Reservations R
where Status = 'pending'
go

grant select on PendingReservations to
moderator
go

grant select on PendingReservations to worker
go

create view project.ReservationsInfo as
```

```
select R.ReservationID, R.SizeOfReservation,
T.TableNumber, T.Capacity from
project.Reservations R
inner join project.ReservationDetails RD on
R.ReservationID = RD.ReservationID
                        inner join project.Tables
T on T.TableNumber = RD.TableID
where Status = 'accepted'
go

grant select on ReservationsInfo to moderator
go

grant select on ReservationsInfo to worker
go

CREATE view project.SeafoodMeals as
SELECT MealID, Name, Description, UnitInStock
FROM project.Meals M
INNER JOIN project.Categories C ON C.CategoryID
= M.CategoryID
WHERE CategoryName = 'Seafood'
go

grant select on SeafoodMeals to moderator
go

grant select on SeafoodMeals to worker
go

CREATE view project.SeafoodMealsStats as
select M.MealID, OrderDetails.OrderId, Quantity
from project.OrderDetails
inner join project.Orders O on O.OrderID =
OrderDetails.OrderId
```

```
inner join project.Menus M on M.MenuID =
OrderDetails.MenuID
inner join project.Meals M2 on M2.MealID =
M.MealID
left join project.Reservations R on
R.ReservationID = O.ReservationID
where (M.MealID in (SELECT MealID FROM
project.SeafoodMeals)) and GETDATE() <
R.StartDate and DATEDIFF(DAY, getdate(),
R.StartDate) < 7
union
select M.MealID, OrderDetails.OrderId, Quantity
from project.OrderDetails
inner join project.Orders O on O.OrderID =
OrderDetails.OrderId
inner join project.Menus M on M.MenuID =
OrderDetails.MenuID
inner join project.Meals M2 on M2.MealID =
M.MealID
left join project.Takeaways T on T.TakeawayID =
O.TakeawayID
where (M.MealID in (SELECT MealID FROM
project.SeafoodMeals)) and GETDATE() <
T.TakewayTime and DATEDIFF(DAY, getdate(),
T.TakewayTime) < 7
go

grant select on SeafoodMealsStats to moderator
go

grant select on SeafoodMealsStats to worker
go

create view project.TablesMonthly as
```

```
select  YEAR(StartDate) as 'Year',
MONTH(StartDate) as 'Month', TableID, Capacity,
COUNT(TableID) as 'Count'
from project.Tables T
        inner join project.ReservationDetails
RD on T.TableNumber = RD.TableID
        inner join project.Reservations R2 on
R2.ReservationID = RD.ReservationID
group by TableID, YEAR(StartDate),
MONTH(StartDate), Capacity
go

grant select on TablesMonthly to moderator
go

grant select on TablesMonthly to worker
go

create view project.TablesWeekly as
select YEAR(StartDate) as 'Year',
DATEPART(week, StartDate) as 'Week', TableID,
Capacity, COUNT(TableID) as 'Count'
from project.Tables T
        inner join project.ReservationDetails
RD on T.TableNumber = RD.TableID
        inner join project.Reservations R2 on
R2.ReservationID = RD.ReservationID
group by TableID, YEAR(StartDate),
DATEPART(week, StartDate), Capacity
go

grant select on TablesWeekly to moderator
go

grant select on TablesWeekly to worker
go
```

```
CREATE view project.WeeklyIncome as
select year(OrderDate) as 'year',
datepart(week, OrderDate) as 'week',
sum(Quantity * M.UnitPrice * ROUND((ISNULL(100
- PercentageValue, 100) / 100), 2)) as 'amount'
from project.Orders O
inner join project.OrderDetails OD on O.OrderID
= OD.OrderId
inner join project.Menus M on M.MenuID =
OD.MenuID
left join project.Discounts D on O.ClientID =
D.ClientID
left join project.DiscountDetails DD on
DD.DiscountID = D.DiscountID
group by year(OrderDate), datepart(week,
OrderDate)
go

grant select on WeeklyIncome to moderator
go

grant select on WeeklyIncome to worker
go

CREATE PROCEDURE project.AddCity @CityName
varchar(255),
                          @CountryName
varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
               SELECT *
               FROM project.Cities
```

```
                WHERE CityName = @CityName
            )
            BEGIN
                ;
                THROW 52000, N'Już jest takie
miasto', 1
            END
        IF NOT EXISTS(
                SELECT *
                FROM project.Countries
                where CountryName = @CountryName
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma takiego
państwa w bazie', 1
            END
        DECLARE @CountryID INT
        SELECT @CountryID = CountryID
        FROM project.Countries
        WHERE CountryName = @CountryName
        INSERT INTO project.Cities(CityName,
CountryID)
        VALUES (@CityName, @CountryID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania nowego miasta: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
end
go

grant execute on AddCity to moderator
go
```

```
CREATE PROCEDURE project.AddCompany @Phone
varchar(16),
                                @Email varchar(255),
                                @CompanyName
varchar(255),
                                @NIP varchar(255),
                                @City varchar(255),
                                @Street
varchar(255),
                                @PostalCode
varchar(6)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRAN
        BEGIN TRY
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Phone = @Phone
                )
                BEGIN
                    ;
                    THROW 52000, N'Numer
telefonu jest już w bazie', 1
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Email = @Email
                )
                BEGIN
                    ;
                    THROW 52000, N'Email jest
już w bazie', 1
```

```
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Companies
                    WHERE NIP = @NIP
                )
                BEGIN
                    ;
                    THROW 52000, N'Firma o
podanym NIPie jest już w bazie', 1
                END
            IF NOT EXISTS(
                    SELECT *
                    FROM project.Cities
                    WHERE CityName = @City
                )
                BEGIN
                    ;
                    THROW 52000, N'Nie ma
podanego miasta w bazie', 1
                END
            DECLARE @ClientID INT
            SELECT @ClientID =
ISNULL(MAX(ClientID), 0) + 1
            FROM project.Clients
            INSERT INTO
project.Clients(ClientID, Phone, Email)
            VALUES (@ClientID, @Phone, @Email);
            DECLARE @CityID INT
            SELECT @CityID = CityID
            FROM project.Cities
            WHERE CityName = @City
            INSERT INTO
project.Companies(CompanyID, CompanyName, NIP,
CityID, Street, PostalCode)
```

```
            VALUES (@ClientID, @CompanyName,
@NIP, @CityID, @Street, @PostalCode)
            COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodania firmy: ' +
ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
END
go

grant execute on AddCompany to moderator
go

CREATE PROCEDURE project.AddDiscount @ClientID
int
AS
BEGIN
   IF NOT EXISTS(SELECT * FROM
project.IndividualClients WHERE @ClientID =
ClientID)
        BEGIN
            ;
            THROW 52000, N'Taki klient nie
istnieje', 1
        end
   IF
(project.checkIfClientHasActiveDiscount(@Client
ID) = 1)
        BEGIN
            ;
            THROW 52000, N'Ten klient ma już
aktywną zniżkę', 1
```

```
        end
    DECLARE @AvailableDiscounts int = 0;
    IF (EXISTS(SELECT * FROM
project.ClientsWithAvailableSecondDiscount
WHERE ClientID = @ClientID) AND
        (@ClientID not in (SELECT DISTINCT
ClientID FROM project.Discounts)))
        Begin
            SELECT @AvailableDiscounts =
@AvailableDiscounts + 1;
            INSERT INTO
project.DiscountDetails(StartDate, EndDate,
Avaliable, PercentageValue)
            VALUES (GETDATE(), DATEADD(DAY, 7,
GETDATE()), 1,
                    (SELECT d.Value FROM
project.DictionaryValues as d WHERE d.ValueID =
5))
        end
    ELSE
        IF EXISTS(SELECT *
                  FROM
project.ClientsWithAvailableFirstDiscount
                  WHERE ClientID = @ClientID)
            Begin
                SELECT @AvailableDiscounts =
@AvailableDiscounts + 1;
                INSERT INTO
project.DiscountDetails(StartDate, EndDate,
Avaliable, PercentageValue)
                VALUES (GETDATE(), null, 1,
                        (SELECT d.Value FROM
project.DictionaryValues as d WHERE d.ValueID =
3))
            end
```

```
    IF (@AvailableDiscounts = 0)
        BEGIN
            ;
            THROW 52000, N'Klientowi nie
przysługuje żadna zniżka', 1
        END
    ELSE
        BEGIN
            DECLARE @DiscountID int;
            SELECT @DiscountID =
MAX(DiscountDetails.DiscountID) FROM
project.DiscountDetails
            INSERT INTO
project.Discounts(DiscountID, ClientID)
            VALUES (@DiscountID, @ClientID)
        end
end
go

grant execute on AddDiscount to moderator
go

grant execute on AddDiscount to worker
go

CREATE   PROCEDURE project.AddEmployee
@CompanyName varchar(255),
                                        @FirstName
varchar(255),
                                        @LastName
varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF EXISTS(
```

```
                SELECT *
                FROM project.Employees as E
                        INNER JOIN
project.Companies C on E.CompanyID =
C.CompanyID
                WHERE @FirstName = E.FirstName
                  AND @LastName = E.LastName
                  AND @CompanyName =
C.CompanyName
            )
            BEGIN
                ;
                THROW 52000, N'Pracownik jest
już w bazie', 1
            END
        IF NOT EXISTS(
                SELECT *
                FROM project.Companies
                WHERE @CompanyName = CompanyName
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej
firmy', 1
            end
        DECLARE @CompanyID INT
        SELECT @CompanyID = CompanyID
        FROM project.Companies
        WHERE @CompanyName = CompanyName
        INSERT INTO project.Employees(CompanyID,
FirstName, LastName)
        VALUES (@CompanyID, @FirstName,
@LastName)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
```

```
            =N'Błąd dodania pracownika: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

grant execute on AddEmployee to moderator
go

CREATE   PROCEDURE
project.AddEmployeeReservation @ClientID int,

@Products project.OrderMeals readonly ,

@PaymentID int = null,

@TypeOfPayment varchar(255),

@ReservationStartDate datetime,

@ReservationEndDate datetime,

@Employees ListOfIDs readonly
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @Name varchar(20) = 'AddOrderRN'
    BEGIN TRAN @Name
        SAVE TRAN @Name
        BEGIN TRY
            IF
project.canClientHaveReservation(@ClientID,
@Products) = 0
                BEGIN
```

```
                    THROW 52000, N'Klient nie
może złożyć rezerwacji', 1
                END
            DECLARE @ResID int;
            DECLARE @ReservationSize int;
            SELECT @ReservationSize = COUNT(*)
FROM @Employees
            EXECUTE project.AddReservation
@ReservationStartDate, @ReservationEndDate, 2,
                    @ReservationID = @ResID
OUTPUT
            IF EXISTS(SELECT * FROM @Products)
                BEGIN
                    EXECUTE project.AddOrder
@ClientID, @Products, @PaymentID,
@TypeOfPayment, 0, null, @ResID
                END
            Insert Into
project.ComapnyReservations(ReservationID,
CompanyID)
            VALUES (@ResID, @ClientID)
            DECLARE @EmployeeID as int
            DECLARE EmployeeCursor CURSOR FOR
                SELECT * FROM @Employees;
            OPEN EmployeeCursor;
            FETCH NEXT FROM EmployeeCursor INTO
@EmployeeID;
            WHILE @@FETCH_STATUS = 0
                BEGIN
                    EXECUTE
project.AddEmployeeToReservation @ResID,
@EmployeeID
                    FETCH NEXT FROM
EmployeeCursor INTO @EmployeeID;
                END;
            COMMIT TRAN @Name
```

```
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia: '
+ ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go

CREATE   PROCEDURE
project.AddEmployeeToReservation @ReservationID
int,

@EmployeeID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
        Declare @CompanyID int

        SELECT @CompanyID = CompanyID FROM
project.ComapnyReservations WHERE ReservationID
= @ReservationID
        IF NOT EXISTS(SELECT * FROM
project.Reservations WHERE ReservationID =
@ReservationID)
            BEGIN
                THROW 52000, N'Taka rezerwacja
nie isnieje', 1
            end
        IF NOT EXISTS(SELECT * FROM
project.Employees WHERE EmployeeID =
@EmployeeID)
            BEGIN
```

```
                THROW 52000, N'W bazie nie ma
takiego pracownika', 1
            end
        INSERT INTO
project.CompanyPersonalReservation(EmployeeID,
ReservationID)
        VALUES (@EmployeeID, @ReservationID)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania zamówienia: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

CREATE PROCEDURE project.AddIndividualClient
@Phone varchar(16),
                                        @Email
varchar(255),
                                        @FirstName
varchar(255),
                                        @LastName
varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRAN
        BEGIN TRY
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Phone = @Phone
                )
                BEGIN
```

```
                    ;
                    THROW 52000, N'Numer
telefonu jest już w bazie', 1
                END
            IF EXISTS(
                    SELECT *
                    FROM project.Clients
                    WHERE Email = @Email
                )
                BEGIN
                    ;
                    THROW 52000, N'Email jest
już w bazie', 1
                END
            DECLARE @ClientID INT
            SELECT @ClientID =
ISNULL(MAX(ClientID), 0) + 1
            FROM project.Clients
            INSERT INTO
project.Clients(ClientID, Phone, Email)
            VALUES (@ClientID, @Phone, @Email);
            INSERT INTO
project.IndividualClients(ClientID, FirstName,
LastName)
            VALUES (@ClientID, @FirstName,
@LastName)
            COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodania klienta
indywidualnego: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
END
```

```
go

grant execute on AddIndividualClient to
moderator
go

CREATE   PROCEDURE project.AddMeal @Name
varchar(255),
                                   @CategoryName
varchar(255),
                                   @Description
varchar(255),
                                   @UnitInStock
int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
           )
           BEGIN
                ;
                THROW 52000, N'Potrawa jest już
dodana', 1
           END
       IF NOT EXISTS(
                SELECT *
                FROM project.Categories
                WHERE CategoryName =
@CategoryName
           )
           BEGIN
                ;
```

```
                THROW 52000, 'Nie ma takiej
kategorii', 1
            END
        DECLARE @CategoryID INT
        SELECT @CategoryID = CategoryID
        FROM project.Categories
        WHERE CategoryName = @CategoryName
        DECLARE @MealID INT
        SELECT @MealID = ISNULL(MAX(MealID), 0)
+ 1
        FROM project.Meals
        INSERT INTO project.Meals(Name, MealID,
CategoryID, Description, UnitInStock)
        VALUES (@Name, @MealID, @CategoryID,
@Description, @UnitInStock);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
go

grant execute on AddMeal to moderator
go

grant execute on AddMeal to worker
go

CREATE   PROCEDURE project.AddMealToMenu @Name
varchar(255),
                                        @Price
money,
```

```
@StartDate varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej
potrawy', 1
            END

        DECLARE @MealID INT
        SELECT @MealID = MealID
        FROM project.Meals
        WHERE Name = @Name

        IF EXISTS(
                SELECT *
                FROM project.Menus
                WHERE MealID = @MealID
                  AND (RemoveFromMenuDate IS
NULL OR RemoveFromMenuDate > @StartDate)
            )
            BEGIN
                ;
                THROW 52000, N'W menu istnieje
już taka potrawa', 1
            END

        DECLARE @MenuID INT
```

```
        SELECT @MenuID = ISNULL(MAX(MenuID), 0)
+ 1
        FROM project.Menus
        INSERT INTO project.Menus(MenuID,
MealID, AddToMenuDate, RemoveFromMenuDate,
UnitPrice)
        VALUES (@MenuID, @MealID, @StartDate,
null, @Price);
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy do menu: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

grant execute on AddMealToMenu to moderator
go

grant execute on AddMealToMenu to worker
go

CREATE   PROCEDURE project.AddMealToOrder
@OrderID int,

@Quantity int,

@MenuID int
AS
BEGIN
    SET NOCOUNT ON
        BEGIN TRY
            IF
project.isMenuItemAvailable(@MenuID) = 0
```

```
                BEGIN
                    THROW 52000, N'Ta potrawa
nie jest aktualnie dostępna', 1
                END
            IF project.orderExists(@OrderID) = 0
                BEGIN
                    THROW 52000, N'Nie ma
takiego zamówienia', 1
                end
            IF
project.isMenuItemInStock(@MenuID, @Quantity) =
0
                BEGIN
                    THROW 52000, N'Nie jest
dostępna taka ilość', 1
                end
            DECLARE @MealID int
            SELECT @MealID = C.MealID
            FROM project.CurrentMenu C
            WHERE C.MenuID = @MenuID
            INSERT INTO
project.OrderDetails(MenuID, OrderId, Quantity)
            VALUES (@MenuID, @OrderID,
@Quantity)
            UPDATE project.Meals
            SET UnitInStock = UnitInStock -
@Quantity
            WHERE MealID = @MealID
        END TRY
        BEGIN CATCH
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania produktu do
zamówienia: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END CATCH
END
```

```
go

CREATE   PROCEDURE project.AddOrder @ClientID
int,

@Products project.OrderMeals readonly,

@PaymentID int = null,

@TypeOfPayment varchar(255) = null,

@IsTakeaway bit,

@TakeawayTime datetime = null,

@ReservationID int
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Name varchar(20) = 'AddOrder'
   BEGIN TRAN @Name
       SAVE TRAN @Name
       BEGIN TRY
           IF project.customerExists(@ClientID)
= 0
               BEGIN
                   THROW 52000, N'Taki klient
nie istnieje!', 1
               END
           DECLARE @OrderDate datetime
           SET @OrderDate = GETDATE()
           DECLARE @ReservationStartDate
datetime
           SET @ReservationStartDate =
project.getStartDateOfReservation(@ReservationI
D)
```

```
            IF
(project.haveMealsSomeSeafood(@Products) = 1)
AND

(project.canOrderHasSeafood(@OrderDate,
@ReservationStartDate, @TakeawayTime) = 0)
                BEGIN
                    THROW 52000, N'Zamówienie
zawiera owoce morza ale nie może zostać
zrealizowane!', 1
                END
            DECLARE @VarPaymentID int;
            SET @VarPaymentID = @PaymentID
            IF @PaymentID IS NULL
                BEGIN
                    EXECUTE project.AddPayment
DEFAULT, @TypeOfPayment, @PaymentID =
@VarPaymentID OUTPUT
                end
            DECLARE @TakeAwayID int
            IF @IsTakeaway = 1
                BEGIN
                    EXECUTE project.AddTakeaway
@TakeawayTime, @TakeAwayID
                END
            DECLARE @OrderID int
            SELECT @OrderID =
ISNULL(MAX(OrderID), 0) + 1 FROM project.Orders
            INSERT INTO project.Orders(OrderID,
ClientID, OrderDate, PaymentID, TakeawayID,
ReservationID)
            VALUES (@OrderID, @ClientID,
@OrderDate, @VarPaymentID, @TakeAwayID,
@ReservationID)
            DECLARE @MenuID as int
            DECLARE @Quantity as int
```

```
            DECLARE ProductCursor CURSOR FOR
                SELECT * FROM @Products;
            OPEN ProductCursor;
            FETCH NEXT FROM ProductCursor INTO
@MenuID, @Quantity;
            WHILE @@FETCH_STATUS = 0
                BEGIN
                    EXECUTE
project.AddMealToOrder @OrderID, @Quantity,
@MenuID
                    FETCH NEXT FROM
ProductCursor INTO @MenuID, @Quantity;
                END;
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia: '
+ ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go

grant execute on AddOrder to moderator
go

grant execute on AddOrder to worker
go

CREATE   PROCEDURE
project.AddOrderWithEmployeeReservation
@ClientID int,

@Products project.OrderMeals readonly,
```

```
@PaymentID int = null,

@TypeOfPayment varchar(255),

@ReservationStartDate datetime,

@ReservationEndDate datetime,

@Employees ListOfIDs readonly
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @Name varchar(20) = 'AddOrderRN'
    BEGIN TRAN @Name
        SAVE TRAN @Name
        BEGIN TRY
            DECLARE @CanAddReservation BIT
            SET @CanAddReservation =
project.canClientHaveReservation(@ClientID,
@Products)
            IF @CanAddReservation = 0
                BEGIN
                    THROW 52000, N'Klient nie
może złożyć rezerwacji', 1
                END
            DECLARE @ReservationID int;
            DECLARE @ReservationSize int;
            SELECT @ReservationSize = COUNT(*)
SELECT * FROM @Employees
            EXECUTE project.AddReservation
@ReservationStartDate, @ReservationEndDate,
@ReservationSize,
                    @ReservationID =
@ReservationID
```

```
            EXECUTE project.AddOrder @ClientID,
@Products, @PaymentID, @TypeOfPayment, 0, null,
@ReservationID
            DECLARE @EmployeeID as int
            DECLARE EmployeeCursor CURSOR FOR
                SELECT * FROM @Employees;
            OPEN EmployeeCursor;
            FETCH NEXT FROM EmployeeCursor INTO
@EmployeeID;
            WHILE @@FETCH_STATUS = 0
                BEGIN
                    EXECUTE
project.AddEmployeeToReservation
@ReservationID, @EmployeeID
                    FETCH NEXT FROM
EmployeeCursor INTO @EmployeeID;
                END;
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia: '
+ ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go

CREATE   PROCEDURE
project.AddOrderWithSizeReservation @ClientID
int,

@Products project.OrderMeals readonly,

@PaymentID int = null,
```

```
@TypeOfPayment varchar(255),

@ReservationStartDate datetime,

@ReservationEndDate datetime,

@ReservationSize int
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @Name varchar(20) = 'AddOrderRS'
    BEGIN TRAN @Name
        SAVE TRAN @Name
        BEGIN TRY
            DECLARE @CanAddReservation BIT
            SET @CanAddReservation =
project.canClientHaveReservation(@ClientID,
@Products)
            IF @CanAddReservation = 0
                BEGIN
                    THROW 52000, N'Klient nie
może złożyć rezerwacji', 1
                END
            DECLARE @ResID int;
            EXECUTE project.AddReservation
@ReservationStartDate, @ReservationEndDate,
@ReservationSize,
                    @ReservationID = @ResID
OUTPUT
            EXECUTE project.AddOrder @ClientID,
@Products, @PaymentID, @TypeOfPayment, 0, null,
@ResID
            DECLARE @OrderID as int;
            SELECT @OrderID =
ISNULL(MAX(OrderID), 0) FROM project.Orders
```

```
            IF EXISTS(SELECT * FROM
project.IndividualClients WHERE ClientID =
@ClientID)
                BEGIN
                    INSERT INTO
project.IndividualClientReservations(Reservatio
nID, IndividualClientID, OrderID)
                    VALUES (@ResID, @ClientID,
@OrderID)
                end
            IF EXISTS(SELECT * FROM
project.Companies WHERE CompanyID = @ClientID)
                BEGIN
                    Insert Into
project.ComapnyReservations(ReservationID,
CompanyID)
                    VALUES (@ResID, @ClientID)
                end
            COMMIT TRAN
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN @Name
            DECLARE @msg nvarchar(2048)
                =N'Błąd dodawania zamówienia z
rezerwacją: ' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
        END catch
END
go

grant execute on AddOrderWithSizeReservation to
moderator
go

grant execute on AddOrderWithSizeReservation to
worker
```

```
go

CREATE   PROCEDURE project.AddPayment
@PaymentStatus bit = 0,

@TypeOfPayment varchar(255),

@PaymentID int OUTPUT
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       INSERT INTO
project.PaymentDetails(PaymentStatus,
TypeOfPayment)
       VALUES (@PaymentStatus, @TypeOfPayment)
       SELECT @PaymentID = MAX(PaymentID) FROM
project.PaymentDetails
   end try
   begin catch
       DECLARE @msg nvarchar(2048)
           =N'Błąd dodawania płatności: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1
   end catch
END
go

CREATE   PROCEDURE project.AddReservation
@ReservationStartDate datetime,

@ReservationEndDate datetime,

@ReservationSize int,

@ReservationID int OUTPUT
```

```
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        SELECT @ReservationID =
ISNULL(MAX(ReservationID), 0) + 1 FROM
project.Reservations
        INSERT INTO
project.Reservations(ReservationID, StartDate,
EndDate, Status, SizeOfReservation)
        VALUES (@ReservationID,
@ReservationStartDate, @ReservationEndDate,
'pending', @ReservationSize)
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania rezerwacji: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
end
go

grant execute on AddReservation to moderator
go

grant execute on AddReservation to worker
go

CREATE PROCEDURE project.AddTable @Capacity int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO project.Tables(Capacity)
        VALUES (@Capacity);
```

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

grant execute on AddTable to moderator
go

grant execute on AddTable to worker
go

CREATE PROCEDURE project.AddTableToReservation
@ReservationID int,

@TableID int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableID
            )
            BEGIN
                ;
                THROW 52000, N'Nie ma stolika o
podanym ID', 1
            END
        IF EXISTS(
                SELECT *
```

```
                FROM project.ReservationDetails
                WHERE TableID = @TableID
            )
            BEGIN
                ;
                THROW 52000, N'Ten stolik jest
już przydzielony do tej rezerwacji', 1
            END

        DECLARE @StartDate DATE
        SELECT @StartDate = StartDate from
project.Reservations R where R.ReservationID =
@ReservationID
        DECLARE @EndDate DATE
        SELECT @EndDate = EndDate from
project.Reservations R where R.ReservationID =
@ReservationID

        IF @TableID not in (select * from
project.getFreeTables(@StartDate, @EndDate))
            BEGIN
                ;
                THROW 52000, N'Ten stolik jest
zajęty w tych godzinach!', 1
            END

        BEGIN
            INSERT INTO
project.ReservationDetails(ReservationID,
TableID)
            VALUES (@ReservationID, @TableID)
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
```

```
            =N'Błąd dopisania stolika do
rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

grant execute on AddTableToReservation to
moderator
go

grant execute on AddTableToReservation to
worker
go

CREATE   PROCEDURE project.AddTakeaway
@TakeawayTime datetime,

@TakeawayID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        INSERT INTO
project.Takeaways(TakewayTime)
        VALUES (@TakeawayTime)
        SELECT @TakeawayID = MAX(TakeawayID)
FROM project.Takeaways
    end try
    begin catch
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodawania danych odbioru
zamówienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    end catch
END
```

```
go

CREATE   PROCEDURE project.CancelReservation
@ReservationID int
AS
BEGIN
   SET NOCOUNT ON
       BEGIN TRY
           IF
project.reservationExists(@ReservationID) = 0
               BEGIN
                   THROW 52000, N'Nie ma takiej
rezerwacji', 1
               END
           DELETE project.Reservations WHERE
ReservationID = @ReservationID
       END TRY
       BEGIN CATCH
           DECLARE @msg nvarchar(2048)
               =N'Błąd usuwania rezerwacji: ' +
ERROR_MESSAGE();
           THROW 52000, @msg, 1
       END catch
END
go

CREATE   PROCEDURE project.ChangePaymentStatus
@OrderID int AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
               SELECT *
               FROM project.Orders
               WHERE OrderID = @OrderID
           )
```

```
            BEGIN
                ;
                THROW 52000, N'Nie ma takiego
zamówienia', 1
            END
        IF ((SELECT PaymentStatus
             FROM project.PaymentDetails
             WHERE PaymentID = (SELECT PaymentID
                                FROM
project.Orders
                                WHERE OrderID =
@OrderID)) = 1)
            BEGIN
                ;
                THROW 52000, N'Zamówienie jest
już opłacone', 1
            END
        DECLARE @PaymentID int
        SELECT @PaymentID = PaymentID
        FROM project.Orders
        WHERE OrderID = @OrderID
        BEGIN
            UPDATE project.PaymentDetails
            SET PaymentStatus = 1
            WHERE PaymentID = @PaymentID
        END
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd zmiany statusu platnosci: '
+
             ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

```
grant execute on ChangePaymentStatus to
moderator
go

grant execute on ChangePaymentStatus to worker
go

CREATE PROCEDURE
project.ChangeReservationStatus @ReservationID
int,

@NewStatus varchar(32),

@TableID int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM project.Reservations
                WHERE ReservationID =
@ReservationID
           )
           BEGIN
                ;
                THROW 52000, N'Nie ma rezerwacji
o podanym ID', 1
           END
       IF (SELECT Status
           FROM project.Reservations
           WHERE ReservationID =
@ReservationID) = @NewStatus
           BEGIN
                ;
```

```
                THROW 52000, N'Ta rezerwacja ma
już taki status', 1
            END
        BEGIN
            UPDATE project.Reservations
            SET Status = @NewStatus
            WHERE ReservationID = @ReservationID
        END
        IF @NewStatus = 'accepted'
            BEGIN
                IF @TableID is null
                    BEGIN
                        ;
                        THROW 52000, N'Musisz
podać id stolika', 1
                    END
            END
        BEGIN
            IF NOT EXISTS(
                    SELECT *
                    FROM project.Tables
                    WHERE TableNumber = @TableID
                )
                BEGIN
                    ;
                    THROW 52000, N'Nie ma
stolika o podanym ID', 1
                END
        END
        BEGIN
            INSERT INTO
project.ReservationDetails(ReservationID,
TableID)
            VALUES (@ReservationID, @TableID)
        END
        -- CODE
```

```
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd edytowania statusu
rezerwacji: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

grant execute on ChangeReservationStatus to
moderator
go

grant execute on ChangeReservationStatus to
worker
go

CREATE   PROCEDURE project.EditTable
@TableNumber int,
                                    @NewCapacity
int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableNumber
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego
stolika', 1
            END
```

```
        UPDATE project.Tables
        SET Capacity = @NewCapacity
        WHERE TableNumber = @TableNumber
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd usuwania stolika: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go

grant execute on EditTable to moderator
go

grant execute on EditTable to worker
go

CREATE PROCEDURE project.RemoveMealFromMenu
@Name varchar(255),
                                    @EndDate
varchar(255)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Meals
                WHERE Name = @Name
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiej
potrawy', 1
```

```
            END

        DECLARE @MealID INT
        SELECT @MealID = MealID
        FROM project.Meals
        WHERE Name = @Name

        IF NOT EXISTS(
                SELECT *
                FROM project.Menus
                WHERE MealID = @MealID
                  AND RemoveFromMenuDate IS NULL
            )
            BEGIN
                ;
                THROW 52000, 'W menu nie ma
takiej potrawy', 1
            END
        DECLARE @MenuID INT
        SELECT @MenuID = MenuID
        FROM project.Menus
        WHERE MealID = @MealID
          AND RemoveFromMenuDate IS NULL;
        UPDATE project.Menus
        SET RemoveFromMenuDate = @EndDate
        WHERE MealID = @MealID
          AND MenuID = @MenuID;
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd dodania potrawy do menu: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

```
grant execute on RemoveMealFromMenu to
moderator
go

grant execute on RemoveMealFromMenu to worker
go

CREATE PROCEDURE project.RemoveTable
@TableNumber int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        IF NOT EXISTS(
                SELECT *
                FROM project.Tables
                WHERE TableNumber = @TableNumber
            )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego
stolika', 1
            END
        DELETE FROM project.Tables WHERE
TableNumber = @TableNumber;
    END TRY
    BEGIN CATCH
        DECLARE @msg nvarchar(2048)
            =N'Błąd usuwania stolika: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1
    END CATCH
END
go
```

```
grant execute on RemoveTable to moderator
go

grant execute on RemoveTable to worker
go

CREATE   PROCEDURE project.ShouldMenuBeChanged
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @MealsToChange FLOAT;
   DECLARE @MealsInMenu FLOAT;
   SELECT @MealsToChange = COUNT(*) FROM
project.MealsLongerThanTwoWeeksInMenu
   SELECT @MealsInMenu = COUNT(*) FROM
project.CurrentMenu
   DECLARE @Percentage FLOAT;
   SELECT @Percentage =
@MealsToChange/@MealsInMenu;
   if(@Percentage>=0.5)
   BEGIN
       ;
           THROW 52000, N'Należy zmienić menu!'
, 1
   end
--     ELSE
--     BEGIN
--         ;
--             THROW 52000, @Percentage, 1
--     end
END
go

grant execute on ShouldMenuBeChanged to
moderator
go
```

```
grant execute on ShouldMenuBeChanged to worker
go

CREATE   PROCEDURE project.AddCategory
@CategoryName varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF EXISTS(
                SELECT *
                FROM project.Categories
                WHERE @CategoryName =
CategoryName
            )
            BEGIN
                ;
                THROW 52000, N'Kategoria jest
już dodana', 1
            end
       DECLARE @CategoryID INT
       SELECT @CategoryID =
ISNULL(MAX(CategoryID), 0) + 1
       FROM project.Categories
       INSERT INTO
project.Categories(CategoryID, CategoryName)
       VALUES (@CategoryID, @CategoryName);
   END TRY
   BEGIN CATCH
       DECLARE @msg nvarchar(2048) =
           N'Błąd dodawania kategorii: ' +
ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
```

```
go

grant execute on addCategory to moderator
go

CREATE   FUNCTION
project.canClientHaveReservation(@ClientID int,
@Products project.OrderMeals readonly)
   RETURNS BIT AS
BEGIN
   IF (@ClientID IN (SELECT CompanyID FROM
Companies))
       BEGIN
           RETURN 1
       end
   IF (@ClientID IN (SELECT ClientID FROM
IndividualClients))
       BEGIN
           DECLARE @WZ as int;
           SET @WZ = project.getActualWZ();
           DECLARE @Value as float;
           SET @Value =
project.getValueOfProducts(@Products);
           DECLARE @WK as int;
           SET @WK = project.getActualWK();
           DECLARE @PreviousOrders as int;
           SELECT @PreviousOrders = COUNT(*)
FROM Orders WHERE ClientID = @ClientID
           IF (@PreviousOrders >= @WK) AND
(@Value >= @WZ)
               BEGIN
                   RETURN 1
               end
       end
   RETURN 0
END
```

```
go

CREATE   FUNCTION
project.canOrderHasSeafood(@OrderDate datetime,
@ReservationDate datetime, @TakeawayDate
datetime)
RETURNS BIT AS
   BEGIN
   DECLARE @Weekday int;
   DECLARE @Days int;
   IF @ReservationDate IS NOT NULL
       BEGIN
           -- Podana jest data rezerwacji
           SET @Days = DATEDIFF(DAY,
@OrderDate, @ReservationDate)
           SET @Weekday = DATEPART(WEEKDAY,
@ReservationDate)
       END
   IF @TakeawayDate IS NOT NULL
       BEGIN
           -- Podana jest data rezerwacji
           SET @Days = DATEDIFF(DAY,
@OrderDate, @TakeawayDate)
           SET @Weekday = DATEPART(WEEKDAY,
@TakeawayDate)
       END
   IF (@Weekday IN (5, 6, 7)) AND @Days > 2
       BEGIN
           RETURN 1
       END
   RETURN 0
   END
go
```

```
create function
project.checkIfClientHasActiveDiscount(@ClientI
D int)
   returns bit as
begin
   return case
               when (@ClientID in (SELECT
DISTINCT ClientID
                                     FROM
project.Discounts)) AND ((SELECT DD.EndDate

FROM project.Discounts

INNER JOIN project.DiscountDetails DD on
DD.DiscountID = Discounts.DiscountID

WHERE @ClientID = ClientID) >=

GETDATE()) AND ((SELECT DD.Avaliable

FROM project.Discounts

INNER JOIN project.DiscountDetails DD on
DD.DiscountID = Discounts.DiscountID

WHERE @ClientID = ClientID) =

1)
                   then 1
               else 0
       end
end
go

CREATE   FUNCTION
project.customerExists(@CustomerID int)
```

```
   RETURNS BIT AS
BEGIN
   IF @CustomerID IN (SELECT ClientID FROM
project.Clients)
       BEGIN
           RETURN 1
       END
   RETURN 0
END
go

CREATE   FUNCTION project.getActualWK()
   RETURNS INT AS
BEGIN
   DECLARE @WK as int;
   SELECT @WK = WK FROM WZWK WHERE ValidTo IS
NULL
   RETURN @WK
END
go

CREATE   FUNCTION project.getActualWZ()
   RETURNS INT AS
BEGIN
   DECLARE @WZ as int;
   SELECT @WZ = WZ FROM WZWK WHERE ValidTo IS
NULL
   RETURN @WZ
END
go

create function
project.getFreeTables(@startDate datetime,
@endDate datetime)
   returns table as
       return
```

```
        select TableNumber
        from project.Tables T
        where TableNumber not in (select TableID
                                  from
project.ReservationDetails RD
                                                inner
join project.Reservations R2 on
R2.ReservationID = RD.ReservationID
                                      where
(@startDate < R2.EndDate and @startDate >
R2.StartDate) or (@endDate > R2.StartDate and
@endDate < R2.EndDate))
go

CREATE   FUNCTION
project.getStartDateOfReservation(@ReservationI
D int)
RETURNS DATETIME AS
   BEGIN
   RETURN (SELECT TOP 1 StartDate FROM
   project.Reservations R
   WHERE R.ReservationID = @ReservationID)
   END
go

CREATE   FUNCTION
project.getValueOfProducts(@Products
project.OrderMeals readonly)
   RETURNS FLOAT AS
BEGIN
   DECLARE @Value as float;
   SELECT @Value = Quantity * Menus.UnitPrice
   FROM @Products P
            INNER JOIN project.Menus Menus ON
Menus.MenuID = P.MenuID
   RETURN ROUND(@Value, 2)
```

```
END
go

CREATE   FUNCTION
project.haveMealsSomeSeafood(@Meals
project.OrderMeals readonly)
   RETURNS BIT AS
BEGIN
   DECLARE @MenuID as int
   DECLARE ProductCursor CURSOR FOR SELECT
MenuID FROM @Meals;
   OPEN ProductCursor;
   FETCH NEXT FROM ProductCursor INTO @MenuID;
   WHILE @@FETCH_STATUS = 0
       BEGIN
           IF project.isSeaFoodMeal(@MenuID) =
1
               BEGIN
                  RETURN 1
               END
           FETCH NEXT FROM ProductCursor INTO
@MenuID;
       END;
   RETURN 0;
END
go

CREATE   FUNCTION
project.isMenuItemAvailable(@MenuID int)
   RETURNS BIT AS
BEGIN
   IF @MenuID IN (SELECT MenuID FROM
project.CurrentMenu)
       BEGIN
           RETURN 1
       END
```

```
    RETURN 0
END
go

CREATE    FUNCTION
project.isMenuItemInStock(@MenuID int,
@RequiredQuantity int)
    RETURNS BIT AS
BEGIN
    DECLARE @AvailableQuantity int
    SELECT @AvailableQuantity = UnitInStock
    FROM project.CurrentMenu
    WHERE @MenuID = MenuID
    IF @AvailableQuantity >= @RequiredQuantity
        BEGIN
            RETURN 1
        END
    RETURN 0
END
go

CREATE    FUNCTION project.isSeaFoodMeal(@MenuID
int)
    RETURNS BIT AS
BEGIN
    DECLARE @MealID int;
    SELECT @MealID = MealID FROM project.Menus
WHERE MenuID = @MenuID
    IF (@MealID IN (SELECT MealID FROM
project.SeafoodMeals))
        BEGIN
            RETURN 1
        end
    RETURN 0
END
go
```

```
CREATE   FUNCTION project.mealExists(@MealID
int)
   RETURNS BIT AS
BEGIN
   IF @MealID IN (SELECT MealID FROM
project.Meals)
       BEGIN
           RETURN 1
       END
   RETURN 0
END
go

CREATE   FUNCTION project.orderExists(@OrderID
int)
   RETURNS BIT AS
BEGIN
   IF @OrderID IN (SELECT OrderID FROM
project.Orders)
       BEGIN
           RETURN 1
       END
   RETURN 0
END
go

CREATE   FUNCTION
project.reservationExists(@ReservationID int)
   RETURNS BIT AS
BEGIN
   IF @ReservationID IN (SELECT ReservationID
FROM project.Reservations)
       BEGIN
           RETURN 1
       END
```

```
    RETURN 0
END
go
```