Projekt Bazy Danych

Barbara Doncer, Nikodem Oleniacz, Wojciech Siwek

SPIS TREŚCI

1.Funkcje systemu

1.Administrator systemu
- dodawanie managera

2.Manager restauracji
- zarządzanie menu
    - Tworzenie menu z wyprzedzeniem (przynajmniej jedno do przodu)
    - Zmiana cennika
- dodawanie i usuwanie pracowników
- generowanie raportów miesięcznych i tygodniowych
    - statystyki zajętych stolików
    - statystyki rabatów (ilość i łączna oszczędność dla klienta)
    - statystyki zamówień (Ilość zamówień i łączna kwota)
- generowanie statystyk zamówień i rabatów klientów

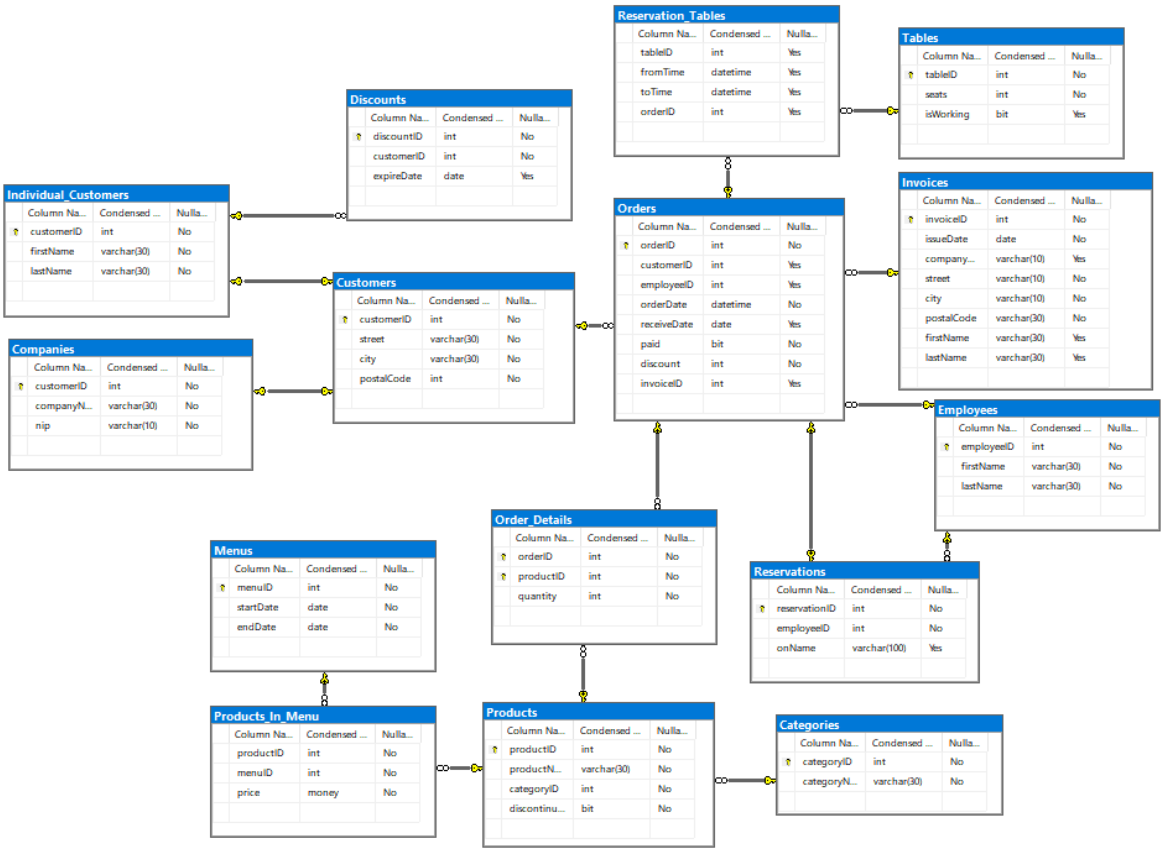
3.Kelner
- rozdzielanie stolików
- przyjmowanie zamówień (w tym zamówień specjalnych)
- akceptacja wcześniejszej rezerwacji zamówienia/stolika
- wydawanie zamówienia
- zmiana statusu płatności
- wystawianie faktur

4.Klient indywidualny
- możliwość zobaczenia aktualnego menu
- złożenie zamówienia w restauracji (na wynos lub na miejscu)
- złożenie zamówienia na wynos przez formularz WWW
- rezerwacja stolika (przy równoczesnym złożeniu zamówienia przez formularz WWW)
- tworzenie konta
- generowanie historii własnych zamówień
- użycie specjalnego rabatu

5.Firma
- tworzenie konta
- możliwość zobaczenia aktualnego menu
- złożenie zamówienia w restauracji (na wynos lub na miejscu)
- rezerwacja stolika (na firmę/dla konkretnych pracowników) przez formularz WWW
- zamówienie na wynos przez formularz WWW
- generowanie historii własnych zamówień

6. System
   - udzielanie rabatu (po spełnieniu konkretnych warunków)
   - proponowanie składu nowego menu
   - sprawdzanie czy użytkownik (klient indywidualny) może złożyć zamówienie online
   - sprawdzanie czy istnieje wolny stolik w wybranym przez klienta terminie

2.Schemat bazy danych



3. Opis tabel z warunkami integralnościowymi

1. **Customers** - tabela zawierająca wszystkich klientów
   *customerID
   *street
   *city
   *postalCode

```
CREATE TABLE [dbo].[Customers](
          [customerID] [int] IDENTITY(1,1) NOT NULL,
          [street] [varchar](30) NOT NULL,
          [city] [varchar](30) NOT NULL,
          [postalCode] [int] NOT NULL,
 CONSTRAINT [Customer_pk] PRIMARY KEY CLUSTERED
(
          [customerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Customers]  WITH CHECK ADD  CONSTRAINT [CK_Customer] CHECK  ((len([postalCode])=(5)))
GO

ALTER TABLE [dbo].[Customers] CHECK CONSTRAINT [CK_Customer]
GO
```

2. **Individual_Customers** - tabela zawierająca wszystkich klientów indywidualnych
   *customerID
   *firstName
   *lastName

```
CREATE TABLE [dbo].[Individual_Customers](
          [customerID] [int] NOT NULL,
          [firstName] [varchar](30) NOT NULL,
          [lastName] [varchar](30) NOT NULL,
 CONSTRAINT [Invidual_Customer_pk] PRIMARY KEY CLUSTERED
(
          [customerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Individual_Customers]  WITH CHECK ADD  CONSTRAINT [FK_Invidual_Customer_Customer] FOREIGN KEY([customerID])
REFERENCES [dbo].[Customers] ([customerID])
GO

ALTER TABLE [dbo].[Individual_Customers] CHECK CONSTRAINT [FK_Invidual_Customer_Customer]
GO
```

3. **Companies** - tabela zawierająca wszystkich klientów firmowych
   *customerID
   *companyName
   *nip

```
CREATE TABLE [dbo].[Companies](
          [customerID] [int] NOT NULL,
          [companyName] [varchar](30) NOT NULL,
          [nip] [varchar](10) NOT NULL,
 CONSTRAINT [Company_pk] PRIMARY KEY CLUSTERED
(
          [customerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [U_nip] UNIQUE NONCLUSTERED
(
          [nip] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [FK_Company_Customer] FOREIGN KEY([customerID])
REFERENCES [dbo].[Customers] ([customerID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Company_Customer]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [CK_Company] CHECK  ((len([nip])=(10)))
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [CK_Company]
GO
```

4. **Discounts** - tabela zawierająca zniżki 2 typu (pojawiające się po zrealizowaniu zamówień na łączną daną kwotę)
   *discountID
   *customerID
   *expireDate - data, w której zniżka przestaje działać

```
CREATE TABLE [dbo].[Discounts](
          [discountID] [int] NOT NULL,
          [customerID] [int] NOT NULL,
          [expireDate] [date] NULL,
 CONSTRAINT [Discounts_pk] PRIMARY KEY CLUSTERED
(
          [discountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discounts]  WITH CHECK ADD  CONSTRAINT [FK_Discounts_Invidual_Customer] FOREIGN KEY([customerID])
REFERENCES [dbo].[Individual_Customers] ([customerID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [FK_Discounts_Invidual_Customer]
GO

ALTER TABLE [dbo].[Discounts]  WITH CHECK ADD  CONSTRAINT [CK_Discounts] CHECK  (([expireDate]>=getdate()))
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [CK_Discounts]
GO
```

5. **Orders** - tabela zawierająca wszystkie zamówienia w restauracji (z rezerwacją, bez rezerwacji, na miejscu, na wynos)
   *orderID
   *customerID - null, jeśli ktoś przychodzi “z ulicy”
   *employeeID
   *orderDate - data złożenia zamówienia
   *receiveDate - data otrzymania zamówienia
   *paid - czy zamówienie zostało opłacone
   *discount - zniżka (podana jako procent)
   *invoiceID

```
CREATE TABLE [dbo].[Orders](
	[orderID] [int] IDENTITY(1,1) NOT NULL,
	[customerID] [int] NULL,
	[employeeID] [int] NULL,
	[orderDate] [datetime] NOT NULL,
	[receiveDate] [date] NULL,
	[paid] [bit] NOT NULL,
	[discount] [int] NOT NULL,
	[invoiceID] [int] NULL,
 CONSTRAINT [Orders_pk] PRIMARY KEY CLUSTERED
(
	[orderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_paid]  DEFAULT ((0)) FOR [paid]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_discount]  DEFAULT ((0)) FOR [discount]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_issuedInvoice]  DEFAULT ((0)) FOR [invoiceID]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_Orders_Customer1] FOREIGN KEY([customerID])
REFERENCES [dbo].[Customers] ([customerID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customer1]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_Orders_Employees] FOREIGN KEY([employeeID])
REFERENCES [dbo].[Employees] ([employeeID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Employees]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_Orders_Invoices] FOREIGN KEY([invoiceID])
REFERENCES [dbo].[Invoices] ([invoiceID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Invoices]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders] CHECK  (([receiveDate]=CONVERT([date],getdate())))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders_1] CHECK  (([discount]>=(0) AND [discount]<=(100)))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders_1]
GO
```

6. **Reservation_Tables** - tabela zawierająca informacje o wszystkich zajętych stołach dla danego zamówienia (jeśli danego orderID nie ma w tej tabeli, to znaczy, że zamówienie jest na wynos)
   *tableID
   *fromTime - od kiedy stolik jest zajęty
   *toTime - do kiedy stolik jest zajęty
   *orderID

```
CREATE TABLE [dbo].[Reservation_Tables](
          [tableID] [int] NULL,
          [fromTime] [datetime] NULL,
          [toTime] [datetime] NULL,
          [orderID] [int] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservation_Tables]  WITH CHECK ADD  CONSTRAINT [FK_Reservation_Tables_Orders] FOREIGN KEY([orderID])
REFERENCES [dbo].[Orders] ([orderID])
GO

ALTER TABLE [dbo].[Reservation_Tables] CHECK CONSTRAINT [FK_Reservation_Tables_Orders]
GO

ALTER TABLE [dbo].[Reservation_Tables]  WITH CHECK ADD  CONSTRAINT [FK_Reservation_Tables_Table] FOREIGN KEY([tableID])
REFERENCES [dbo].[Tables] ([tableID])
GO

ALTER TABLE [dbo].[Reservation_Tables] CHECK CONSTRAINT [FK_Reservation_Tables_Table]
GO

ALTER TABLE [dbo].[Reservation_Tables]  WITH CHECK ADD  CONSTRAINT [CK_Reservation_Tables] CHECK  ((datepart(hour,[fromTime])>=(8)
AND datepart(hour,[fromTime])<=(23)))
GO

ALTER TABLE [dbo].[Reservation_Tables] CHECK CONSTRAINT [CK_Reservation_Tables]
GO

ALTER TABLE [dbo].[Reservation_Tables]  WITH CHECK ADD  CONSTRAINT [CK_Reservation_Tables_1] CHECK  ((datepart(hour,[toTime])>=(8)
AND datepart(hour,[toTime])<=(23)))
GO

ALTER TABLE [dbo].[Reservation_Tables] CHECK CONSTRAINT [CK_Reservation_Tables_1]
GO

ALTER TABLE [dbo].[Reservation_Tables]  WITH CHECK ADD  CONSTRAINT [CK_Reservation_Tables_2] CHECK  (([toTime]>[fromTime]))
GO

ALTER TABLE [dbo].[Reservation_Tables] CHECK CONSTRAINT [CK_Reservation_Tables_2]
GO
```

7. **Tables** - tabela zawierająca wszystkie stoliki w restauracji
   *tableID
   *seats - ilość dostępnych siedzeń przy stoliku
   *isWorking - czy stół działa (nie ma złamanej nogi itp.)

```
CREATE TABLE [dbo].[Tables](
          [tableID] [int] IDENTITY(1,1) NOT NULL,
          [seats] [int] NOT NULL,
          [isWorking] [bit] NULL,
 CONSTRAINT [Table_pk] PRIMARY KEY CLUSTERED
(
          [tableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [CK_Table] CHECK  (([seats]>(0)))
GO

ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [CK_Table]
GO
```

8. **Employees** - tabela zawierająca wszystkich pracowników restauracji
   *employeeID
   *firstName
   *lastName

```
CREATE TABLE [dbo].[Employees](
          [employeeID] [int] IDENTITY(1,1) NOT NULL,
          [firstName] [varchar](30) NOT NULL,
          [lastName] [varchar](30) NOT NULL,
 CONSTRAINT [Employees_pk] PRIMARY KEY CLUSTERED
(
          [employeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

9. **Reservations** - tabela zawierająca informacje o wszystkich złożonych rezerwacjach (dla rezerwacji zawsze od razu uzupełniamy pola w tabeli Orders i Reservation_Tables)
   *reservationID
   *employeeID - pracownik, który zaakceptował rezerwację
   *onName - na kogo jest zamówienie (w przypadku klienta firmowego może to być lista imion i nazwisk)

```
CREATE TABLE [dbo].[Reservations](
          [reservationID] [int] NOT NULL,
          [employeeID] [int] NOT NULL,
          [onName] [varchar](100) NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
          [reservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_Employees] FOREIGN KEY([employeeID])
REFERENCES [dbo].[Employees] ([employeeID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_Employees]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT [FK_Reservations_Orders] FOREIGN KEY([reservationID])
REFERENCES [dbo].[Orders] ([orderID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_Orders]
GO
```

10. **Order_Details**
    *orderID
    *productID
    *quantity - ilość zamówionego produktu

```
CREATE TABLE [dbo].[Order_Details](
          [orderID] [int] NOT NULL,
          [productID] [int] NOT NULL,
          [quantity] [int] NOT NULL,
 CONSTRAINT [Order_Details_pk] PRIMARY KEY CLUSTERED
(
          [orderID] ASC,
          [productID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Order_Details] ADD  CONSTRAINT [DF_Order_Details_quantity]  DEFAULT ((1)) FOR [quantity]
GO

ALTER TABLE [dbo].[Order_Details]  WITH CHECK ADD  CONSTRAINT [FK_Order_Details_Orders] FOREIGN KEY([orderID])
REFERENCES [dbo].[Orders] ([orderID])
GO

ALTER TABLE [dbo].[Order_Details] CHECK CONSTRAINT [FK_Order_Details_Orders]
GO

ALTER TABLE [dbo].[Order_Details]  WITH CHECK ADD  CONSTRAINT [FK_Order_Details_Products] FOREIGN KEY([orderID])
REFERENCES [dbo].[Products] ([productID])
GO

ALTER TABLE [dbo].[Order_Details] CHECK CONSTRAINT [FK_Order_Details_Products]
GO

ALTER TABLE [dbo].[Order_Details]  WITH CHECK ADD  CONSTRAINT [CK_Order_Details] CHECK  (([quantity]>(0)))
GO

ALTER TABLE [dbo].[Order_Details] CHECK CONSTRAINT [CK_Order_Details]
GO
```

11. **Products** - tabela zawierająca spis wszystkich produktów (dań) jakie można było kiedykolwiek zamówić (lub będzie można zamówić) w restauracji
    *productID
    *productName
    *categoryName
    *discontinued - czy produkt można aktualnie dodać do jakiegoś menu

```
CREATE TABLE [dbo].[Products](
          [productID] [int] IDENTITY(1,1) NOT NULL,
          [productName] [varchar](30) NOT NULL,
          [categoryID] [int] NOT NULL,
          [discontinued] [bit] NOT NULL,
 CONSTRAINT [Products_pk] PRIMARY KEY CLUSTERED
(
          [productID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Products] ADD  CONSTRAINT [DF_Products_isAvailable]  DEFAULT ((1)) FOR [discontinued]
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_Category] FOREIGN KEY([categoryID])
REFERENCES [dbo].[Categories] ([categoryID])
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_Category]
GO
```

12. **Products_In_Menu** - tabela zawierająca każdy produkt z dopasowanym do niego menu, w którym się pojawił / pojawi
    *productID
    *menuID
    *price - cena produktu w konkretnym menu

```
CREATE TABLE [dbo].[Products_In_Menu](
          [productID] [int] NOT NULL,
          [menuID] [int] NOT NULL,
          [price] [money] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Products_In_Menu]  WITH CHECK ADD  CONSTRAINT [FK_Products_In_Menu_MenuID1] FOREIGN KEY([menuID])
REFERENCES [dbo].[Menus] ([menuID])
GO

ALTER TABLE [dbo].[Products_In_Menu] CHECK CONSTRAINT [FK_Products_In_Menu_MenuID1]
GO

ALTER TABLE [dbo].[Products_In_Menu]  WITH CHECK ADD  CONSTRAINT [FK_Products_In_Menu_Products1] FOREIGN KEY([productID])
REFERENCES [dbo].[Products] ([productID])
GO

ALTER TABLE [dbo].[Products_In_Menu] CHECK CONSTRAINT [FK_Products_In_Menu_Products1]
GO

ALTER TABLE [dbo].[Products_In_Menu]  WITH CHECK ADD  CONSTRAINT [CK_Products_In_Menu] CHECK  (([price]>(0)))
GO

ALTER TABLE [dbo].[Products_In_Menu] CHECK CONSTRAINT [CK_Products_In_Menu]
GO
```

13. **Menus** - tabela ze wszystkimi menu (także z tymi zaplanowanymi w przyszłości)
    *menuID
    *startDate
    *endDate

```
CREATE TABLE [dbo].[Menus](
          [menuID] [int] IDENTITY(1,1) NOT NULL,
          [startDate] [date] NOT NULL,
          [endDate] [date] NOT NULL,
 CONSTRAINT [PK_MenuID] PRIMARY KEY CLUSTERED
(
          [menuID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Menus]  WITH CHECK ADD  CONSTRAINT [CK_MenuID] CHECK  (([endDate]>getdate() AND [endDate]>[startDate]))
GO

ALTER TABLE [dbo].[Menus] CHECK CONSTRAINT [CK_MenuID]
GO

ALTER TABLE [dbo].[Menus]  WITH CHECK ADD  CONSTRAINT [CK_MenuID_1] CHECK  (([startDate]>=getdate()))
GO

ALTER TABLE [dbo].[Menus] CHECK CONSTRAINT [CK_MenuID_1]
GO
```

14. **Invoices** - tabela z wystawionymi fakturami
    *invoiceID
    *issueDate - data wydania
    *companyName
    *street
    *city
    *postalCode
    *firstName
    *lastName

```
CREATE TABLE [dbo].[Invoices](
          [invoiceID] [int] IDENTITY(1,1) NOT NULL,
          [issueDate] [date] NOT NULL,
          [companyName] [varchar](10) NULL,
          [street] [varchar](10) NOT NULL,
          [city] [varchar](10) NOT NULL,
          [postalCode] [varchar](30) NOT NULL,
          [firstName] [varchar](30) NULL,
          [lastName] [varchar](30) NULL,
 CONSTRAINT [PK_Invoices] PRIMARY KEY CLUSTERED
(
          [invoiceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

15. **Categories** - tabela z kategoriami dań
    *categoryID
    *categoryName

```
CREATE TABLE [dbo].[Categories](
          [categoryID] [int] NOT NULL,
          [categoryName] [varchar](30) NOT NULL,
 CONSTRAINT [PK_Category] PRIMARY KEY CLUSTERED
(
          [categoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

4. Procedury

1. **AddEmployee** - dodawanie pracownika

```
CREATE PROCEDURE [dbo].[AddEmployee]
          @f_name varchar(30),
          @l_name varchar(30)
AS
BEGIN
          SET NOCOUNT ON;
          INSERT INTO [dbo].[Employees]
          (
           firstName                    ,
           lastName
          )
     VALUES
          (
           @f_name,
           @l_name
          )

END
```

2. **AddTable** - dodawanie stolika

```
CREATW PROCEDURE [dbo].[AddTable]
          @seats int
AS
BEGIN
          SET NOCOUNT ON;
          INSERT INTO [dbo].[Tables]
          (
           seats,
           isWorking
          )
     VALUES
          (
           @seats,
           1
          )
END
```

3. **AddIndividualCustomer** - dodawanie klienta indywidualnego (do tabel Individual_Customer i Customer naraz)

```
CREATE PROCEDURE [dbo].[AddIndividualCustomer]
          @f_name varchar(30),
          @l_name varchar(30),
          @street varchar(30),
          @city varchar(30),
          @postal_code int
AS
BEGIN
          SET NOCOUNT ON;
          INSERT INTO [dbo].[Customers]
          (
           street,
           city,
           postalCode
          )
     VALUES
          (
           @street,
           @city,
           @postal_code
          )
          INSERT INTO [dbo].[Individual_Customers]
          (
           customerID,
           firstName,
           lastName
          )
     VALUES
          (
           SCOPE_IDENTITY(),
           @f_name,
           @l_name
          )

END
```

4. **AddCompany** - dodawanie klienta firmowego (do tabel Company i Customer naraz)

```
CREATE PROCEDURE [dbo].[AddCompany]
          @company_name varchar(30),
          @nip varchar(10),
          @street varchar(30),
          @city varchar(30),
          @postal_code int
AS
BEGIN
          SET NOCOUNT ON;
          INSERT INTO [dbo].[Customers]
          (
           street,
           city,
           postalCode
          )
     VALUES
          (
           @street,
           @city,
           @postal_code
          )
           INSERT INTO [dbo].[Companies]
          (
           customerID,
           companyName,
           nip
          )
     VALUES
          (
           SCOPE_IDENTITY(),
           @company_name,
           @nip
          )

END
```

5. **AddOrder** - dodawanie zamówienia do tabeli Orders (dużo pól może być nullami ze względu na to, że tworzymy order przy każdej rezerwacji)
   Procedura AddOrder:
   *uruchamia procedurę AddDiscount, która sprawdza, czy użytkownikowi należy się zniżka typu 2
   *sprawdza poprawność wprowadzonych danych (czy klient, pracownik istnieje)
   *oblicza zniżkę dla klienta (przy uruchamianiu procedury wpisywany jest typ zniżki (0 dla typu 1 i 1 dla typu 2) - jeśli klient nie ma dostępnej takiej zniżki (jest to sprawdzane), to nie otrzymuje żadnej zniżki)
   *wprowadza dane do tabeli Orders
   *rozdziela stoliki potrzebne do obsłużenia tego kienta (dodaje je do Reservation_Tables)
   *dodaje zamówione produkty (do Order_Details)

```
ALTER PROCEDURE [dbo].[AddOrder]
          @seats varchar(500)='',
          @reservationFrom datetime = NULL,
          @productsID varchar(500) = '',
          @customer_id int = NULL,
          @employee_id int = NULL,
          @receiveDate date = NULL,
          @paid bit = 0,
          --discount percentage on the order
          @invoice_id int = NULL,
          --does client uses his one-time discount
          @choosesDiscount bit = 0,
          @output int output
AS
BEGIN
          SET NOCOUNT ON;
          DECLARE @orderDate AS date ;
          DECLARE @discount as int;
          SET @discount = 0;
          SET @orderDate = GETDATE();
```

```
	IF ((SELECT COUNT(*) FROM Customers where customerID = @customer_id) < 1 AND @customer_id is not NULL)
	BEGIN
		DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customer_id) + ' does not exist' ;
		;THROW 52000, @msg,1
		RETURN
	END
	IF ((SELECT COUNT(*) FROM Employees where @employee_id = @employee_id) < 1 AND @employee_id is not NULL)
	BEGIN
		DECLARE @msg2 NVARCHAR(2048) = 'Employee with ID:' + CONVERT(varchar, @employee_id) + ' does not exist' ;
		;THROW 52000, @msg2,1
		RETURN
	END

	BEGIN TRANSACTION
	--jesli wydane pieniadze przkroczą próg, dodaj zniżke do profilu klienta
	EXEC AddDiscount @customer_id

	DECLARE @K1 as int = 30;
	DECLARE @Z1 as int = 3;
	DECLARE @R1 as int = 5;
	DECLARE @R2 as int = 10;

	IF(@customer_id) is not null
	BEGIN
		--add passive discount
		IF(SELECT COUNT(*) FROM Orders WHERE customerID = customerID AND dbo.GetOrderValue(orderID,discount) >= @K1) >= @Z1
		BEGIN
			SET @discount = @R1;
		END
		IF(@choosesDiscount) = 1
		BEGIN
			--add one-time discount if there is any, and update the expireDate
			IF(SELECT [expireDate] FROM Discounts WHERE customerID = @customer_id AND [expireDate] >= @orderDate) is not null
				BEGIN
					SET @discount = @discount+@R2;
					SET ROWCOUNT 1
					UPDATE Discounts
					SET [expireDate] = @orderDate
					WHERE customerID = @customer_id
					SET ROWCOUNT 0
				END
			END
		 END

INSERT INTO [dbo].[Orders]
	(
		customerID,
		employeeID,
		orderDate,
		receiveDate,
		paid,
		discount,
		invoiceID
	)
 VALUES
	(
		@customer_id,
		@employee_id,
		@orderDate,
		@receiveDate,
		@paid,
		@discount,
		@invoice_id
	)

 DECLARE @ctr AS int ;
 DECLARE @Tctr AS int ;
 DECLARE @singleTable AS int ;
 DECLARE @identity AS int ;
 DECLARE @reservationTO AS datetime ;

 SET @identity = SCOPE_IDENTITY()
 SET @output = @identity;
 DECLARE @tableSeats AS int ;
 SET @ctr = (SELECT COUNT(*) FROM STRING_SPLIT(@seats,','));
 IF(@reservationFrom is NULL)
 BEGIN
	SET @reservationFrom = @orderDate;
 END
 WHILE @ctr >= 1
 BEGIN
	SET @tableSeats = CAST(((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@seats,',') ) as p) EXCEPT (SELECT TOP (@ctr-1) * FROM (SELECT value FROM
	STRING_SPLIT(@seats,','))as d))as INT);
	--wszystkie stoli z dobra iloscia miejsc, sprawdz czy sa zarezerwowanie czy nie
	SET @Tctr = (SELECT COUNT(*) FROM [Tables] WHERE seats >= @tableSeats AND isWorking = 1)
```

```
        WHILE @Tctr >=1
        BEGIN
                SET @singleTable = ((SELECT TOP (@Tctr) * FROM (SELECT tableID FROM [Tables] WHERE seats >= @tableSeats AND isWorking = 1) as p) EXCEPT (SELECT TOP
                (@Tctr-1) * FROM (SELECT tableID FROM [Tables] WHERE seats >= @tableSeats AND isWorking = 1)as d));
                IF(SELECT tableID FROM Reservation_Tables WHERE @reservationFrom <= toTime AND DATEADD(HOUR, 3, @reservationFrom) >= fromTime AND tableID =
                @singleTable) is NULL
                        BEGIN
                                --ten stolik jest dobry, wyjdz z petli
                                SET @reservationTO =DATEADD(HOUR, 3, @reservationFrom);
                                EXEC AddTableToOrder @identity,@reservationFrom,@reservationTO,@singleTable; -- rollback inside if failed
                                BREAK
                        END
                        SET @Tctr= @Tctr-1 ;
                END
                IF(@Tctr = 0)
                BEGIN
                        ROLLBACK
                        ;THROW 52000, 'Could not find a valid table for reservation',1
                        RETURN
                END

                SET @ctr = @ctr-1
        END

        DECLARE @prID as int;
        SET @ctr = (SELECT COUNT(*) FROM STRING_SPLIT(@productsID,','));
        WHILE @ctr >= 1
        BEGIN
                SET @prID = CAST(((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@productsID,',')) as p) EXCEPT (SELECT TOP (@ctr-1) * FROM (SELECT value FROM
                STRING_SPLIT(@productsID,','))as d))as int);
                EXEC AddProductToOrder @identity,@prID; -- rollback inside if failed
                SET @ctr = @ctr-1;
        END
        COMMIT
END
```

6. **AddProduct** - dodawanie produktu do tabeli Products

```
CREATE PROCEDURE [dbo].[AddProduct]
          @productName varchar(30),
          @categoryID varchar(30),
          @discontinued bit = 1
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Categories where categoryID = @categoryID) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Category with ID:' + CONVERT(varchar, @categoryID) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
     INSERT INTO [dbo].[Products]
          (
                              productName,
                              categoryID,
                              discontinued
          )
     VALUES
          (
                              @productName,
                              @categoryID,
                              @discontinued
          )
END
```

7. **AddProductToOrder** - dodawanie produktu do zamówienia (tabela Order_Details)

Procedura AddProductToOrder:
*sprawdza czy dany produkt istnieje i jest dostępny w menu w tym czasie
*dodaje produkt do zamówienia (lub zwiększa o 1 liczbę w polu quantity, jeśli produkt jest już zamówiony)

```
CREATE PROCEDURE [dbo].[AddProductToOrder]
          @orderID int,
          @productID int
AS
BEGIN
          BEGIN TRANSACTION
          SET NOCOUNT ON;
          DECLARE @productGaveDate date
          DECLARE @fromTime date = (SELECT top 1 fromTime FROM Reservation_Tables where orderID = @orderID)
          IF @fromTime is null
          BEGIN
                SET @productGaveDate = (SELECT orderDate FROM Orders where orderID = @orderID)
          END
          ELSE
          BEGIN
                SET @productGaveDate = @fromTime
          END
          IF (SELECT COUNT(*) FROM Products_In_Menu
          inner join Menus on Products_In_Menu.menuID = Menus.menuID
          where Products_In_Menu.productID = @productID and Menus.startDate <= @productGaveDate and @productGaveDate <=
          Menus.endDate) = 0
          BEGIN
                    ROLLBACK
                    DECLARE @msg NVARCHAR(2048) = 'Product with ID:' + CONVERT(varchar, @productID) + ' is not avaiable in the menu at this time' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          IF ((SELECT COUNT(*) FROM Products WHERE productID = @productID)= 0)
          BEGIN
                    ROLLBACK
                    DECLARE @msg2 NVARCHAR(2048) = 'Product with ID:' + CONVERT(varchar, @productID) + ' does not exist' ;
                    ;THROW 52000, @msg2,1
                    RETURN
          END
          IF(SELECT quantity FROM Order_Details WHERE orderID = @orderID AND productID = @productID) is not null
          BEGIN
                    UPDATE [Order_Details]
                    SET quantity = quantity+1
                    WHERE orderID = @orderID AND productID = @productID
                    COMMIT
                    RETURN
          END
          INSERT INTO [dbo].[Order_Details]
         (
           orderID,
           productID,
           quantity
         )
     VALUES
         (
           @orderID,
           @productID,
           1
         )
          COMMIT
END
```

8. **AddProductToMenu** - dodawanie produktu do menu (tabela Products_in_menu)

Procedura AddProductToMenu:
*sprawdza czy istnieje takie menu
*sprawdza czy dany produkt nadal można dodawać do menu
*dodaje produkt do menu

```
CREATE PROCEDURE [dbo].[AddProductToMenu]
          @product_id int,
          @menu_id int,
          @price money
AS
BEGIN
          BEGIN TRANSACTION
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Menus where menuID = @menu_id) < 1)
          BEGIN
                    ROLLBACK
                    DECLARE @msg NVARCHAR(2048) = 'Menu with ID:' + CONVERT(varchar, @menu_id) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          IF ((SELECT COUNT(*) FROM Products WHERE productID = @product_id AND discontinued=1) > 0)
          BEGIN
                    ROLLBACK
                    DECLARE @msg2 NVARCHAR(2048) = 'You cannot add product with ID:' + CONVERT(varchar, @product_id) + ' to the menu' ;
                    ;THROW 52000, @msg2,1
                    RETURN
          END
          INSERT INTO [dbo].[Products_In_Menu]
          (
                    productID,
                    menuID,
                    price
          )
           VALUES
          (
                    @product_id,
                    @menu_id,
                    @price
          )
           COMMIT
END
```

9. **AddMenu** - dodawanie Menu

Procedura AddMenu:
*sprawdza czy w danym czasie nie istnieje już inne menu
*dodaje produkty wraz z cenami do menu
*sprawdza czy zmieniła się przynajmniej połowa dań

```
CREATE PROCEDURE [dbo].[AddMenu]
    @startDate date,
    @endDate date,
    @productsID varchar(500),
    @productsPrice varchar(500)
AS
BEGIN
    IF (SELECT COUNT(*) FROM Menus WHERE  @startDate <= endDate AND @endDate >= startDate) != 0
    BEGIN
          ROLLBACK
          ;THROW 52000, 'There cannot be two menus at the same time',1
          RETURN
    END

    BEGIN TRANSACTION
    --polowa rzeczy musi sie zmienic
    INSERT INTO [dbo].[Menus]
         (
           startDate,
           endDate
         )
    VALUES
         (
           @startDate,
           @endDate
         )
    SET NOCOUNT ON;
    DECLARE @ctr AS int ;
    DECLARE @prID AS varchar(10);
```

```
    DECLARE @prPrice AS varchar(10);
    DECLARE @productID AS INT;
    DECLARE @productPrice AS INT;
    DECLARE @ident AS INT = SCOPE_IDENTITY();
    DECLARE @lastMenuID AS INT;
    DECLARE @howManyChanged AS INT = 0;
    SET @ctr = (SELECT COUNT(*) FROM STRING_SPLIT(@productsID,','));
    SET @ident = SCOPE_IDENTITY();
    SET @lastMenuID = (SELECT MenuID FROM Menus WHERE DATEDIFF(DAY,endDate,@startDate) = 1);
    IF (@ctr != (SELECT COUNT(*) FROM STRING_SPLIT(@productsPrice,',')))
    BEGIN
           ROLLBACK
           ;THROW 52000, 'Length of productID list is different that length of prices list',1
           RETURN
    END
    WHILE @ctr >= 1
    BEGIN
        SET @prID = ((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@productsID,',')) as p) EXCEPT (SELECT TOP (@ctr-1) * FROM (SELECT value FROM
        STRING_SPLIT(@productsID,','))as d));
        SET @prPrice =  ((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@productsPrice,',')) as p) EXCEPT (SELECT TOP (@ctr-1) * FROM (SELECT value FROM
        STRING_SPLIT(@productsPrice,','))as d));
        SET @productID  = CAST(@prID AS INT);
        SET @productPrice = CAST(@prPrice AS INT);
        IF ((SELECT COUNT(*) FROM Products WHERE productID = @productID) < 1)
        BEGIN
                    ROLLBACK
                    DECLARE @msg NVARCHAR(2048) = 'Product with ID:' + CONVERT(varchar, @productID) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
        END
           IF ((SELECT COUNT(*) FROM Products_In_Menu WHERE MenuID = @lastMenuID AND productID = @productID) = 0)
           BEGIN
                    SET @howManyChanged = @howManyChanged + 1;
           END
           EXEC AddProductToMenu @productID ,@ident,@productPrice -- rollback inside if failed
           SET @ctr = @ctr-1
    END
    IF (@howManyChanged*2 < @ctr)
    BEGIN
           ROLLBACK
           ;THROW 52000, 'At least half of the products must be new',1
           RETURN
    END
    COMMIT
END
```

10. **AddReservation** - dodawanie rezerwacji (dla każdej rezerwacji od razu dodajemy zamówienie w Orders)

Procedura AddReservation:
*sprawdza czy można zamówić owoce morza na wybrany termin (czy jest już menu na ten czas i czy rezerwacja jest z odpowiednim wyprzedzeniem)
*tworzy Order
*sprawdza czy klient indywidualny ma prawo zrobić rezerwację
*dodaje dane do Reservations

```
CREATE PROCEDURE [dbo].[AddReservation]
          -- Add the parameters for the stored procedure here
          @seats varchar(500),
          @reservationFrom datetime,
          @productsID varchar(500) = NULL,
          @employee_id int,
          @onName varchar(100) = NULL,
          @customer_id int = NULL,
          @o_employee_id int = NULL,
          @receiveDate date = NULL,
          @paid bit = 0,
          @invoice_id int = NULL
AS
BEGIN
          SET NOCOUNT ON;

          IF (SELECT COUNT(*) FROM Employees WHERE @employee_id = employeeID) = 0
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Employee with ID:' + CONVERT(varchar, @employee_id) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END

          --minimalna wartość zamówienia wymagana do złożenia rezerwacji
          DECLARE @WZ as int = 50;
```

```
          --ilość poprzednich zamówień wymagana do złożenia rezerwacji
          DECLARE @WK as int = 5;
          DECLARE @prID as int;
          DECLARE @ctr as int;
          SET @ctr = (SELECT COUNT(*) FROM STRING_SPLIT(@productsID,','));
          WHILE @ctr >= 1
          BEGIN
                    SET @prID = CAST(((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@productsID,',')) as p) EXCEPT (SELECT TOP (@ctr-1) * FROM (SELECT value FROM
                    STRING_SPLIT(@productsID,','))as d))as int);
                    IF(SELECT categoryID FROM Products WHERE productID = @prID)=6
                    BEGIN
                              IF(NOT((DATEPART(WEEKDAY,CONVERT(DATE, @reservationFrom)) = 3 OR DATEPART(WEEKDAY,CONVERT(DATE,@reservationFrom)) = 4 OR
                              DATEPART(WEEKDAY,CONVERT(DATE,@reservationFrom)) = 5) OR
                              DATEDIFF(DAY,CONVERT(DATE,GETDATE()),CONVERT(DATE,@reservationFrom))>=DATEPART(WEEKDAY,CONVERT(DATE,@reservationFrom))))
                              BEGIN
                                        ;THROW 52000, 'Seafood needs to be ordered in advance and only at set days!',1
                                        RETURN
                              END
                     END
                     SET @ctr = @ctr-1;
          END

          BEGIN TRANSACTION

          DECLARE @getIdentity int;
          EXEC AddOrder @seats, @reservationFrom,@productsID,@customer_id, @o_employee_id, @receiveDate,@paid,@invoice_id,0, @output = @getIdentity output -- rollback inside
if failed
          IF(SELECT COUNT(*) FROM Individual_Customers WHERE @customer_id = customerID) = 1
          BEGIN
              DECLARE @productGaveDate date
              DECLARE @fromTime date = (SELECT top 1 fromTime FROM Reservation_Tables where orderID = @getIdentity)
              IF @fromTime is null
              BEGIN
                   SET @productGaveDate = (SELECT orderDate FROM Orders where orderID = @getIdentity)
              END
              ELSE
              BEGIN
                   SET @productGaveDate = @fromTime
              END
              IF(SELECT SUM(price)
              FROM Products_In_Menu
              JOIN Products ON Products_in_menu.productID = Products.productID
              JOIN Menus ON Products_In_Menu.menuID = Menus.menuID
              WHERE Menus.startDate <= @productGaveDate AND Menus.endDate >= @productGaveDate AND (Products_in_menu.productID IN (SELECT value FROM
              STRING_SPLIT(@productsID,','))) <@WZ
              BEGIN
                     ROLLBACK
                     ;THROW 52000, 'The reservation value is too low',1
                     RETURN
              END
              IF(SELECT COUNT(*) FROM Orders WHERE @customer_id = customerID) < @WK
              BEGIN
                     ROLLBACK
                     ;THROW 52000, 'Client does not have required amount of orders',1
                     RETURN
              END
          END

          INSERT INTO [dbo].[Reservations]
          (
           reservationID,
           employeeID,
           onName
          )
     VALUES
          (
           @getIdentity,
           @employee_id,
           @onName
          )

          COMMIT
          RETURN;
END
```

11. **AddTableToOrder** - dodawanie stolika do zamówienia

Procedura AddTableToOrder:
*sprawdza czy podany stolik jest dostępny w wybranym czasie
*dodaje stolik do zamówienia

```
CREATE PROCEDURE [dbo].[AddTableToOrder]
          @orderID int,
          @fromTime datetime,
          @toTime datetime,
          @tableID int

AS
BEGIN
          BEGIN TRANSACTION
          SET NOCOUNT ON;
          --czy jest stolik zarezerwowany w tym czasie
          IF(SELECT tableID FROM Reservation_Tables WHERE @fromTime <= toTime AND @toTime >= fromTime AND tableID = @tableID) is not NULL
          BEGIN
                    ROLLBACK
                    DECLARE @msg NVARCHAR(2048) = 'Table with ID:' + CONVERT(varchar, @tableID) + ' is already reserved at this time' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          BEGIN
                    INSERT INTO [dbo].[Reservation_Tables]
                              (
                                        orderID,
                                        fromTime,
                                        toTime,
                                        tableID
                              )
                     VALUES
                              (
                                        @orderID,
                                        @fromTime,
                                        @toTime,
                                        @tableID
                              )
          END
          COMMIT
END
```

12. **AddDiscount** - dodawanie zniżki typu 2 (jeśli klient wydał określoną kwotę)

```
CREATE PROCEDURE [dbo].[AddDiscount]
    @customerID int,
    @K2 int = 1000,
    @R2 int = 5,
    @D1 int = 7
AS
BEGIN
          SET NOCOUNT ON;
          DECLARE @from_date date = NULL;

          SELECT @from_date = MAX(expireDate) FROM Discounts
          WHERE expireDate< CONVERT(DATE,GETDATE())

          IF(@from_date) is not null
          BEGIN
                    IF(SELECT sum(quantity*price)*(1-discount/100)
                    FROM ORDERS as O
                    INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
                    INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
                    INNER JOIN Menus as M ON M.menuID = PM.menuID
                    WHERE O.customerID = @customerID AND CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate
                    GROUP BY O.orderID, discount) >= @K2
                    BEGIN
                              INSERT INTO [dbo].[Discounts]
                              (
                                   customerID,
                                   expireDate
                              )
                              VALUES
                              (
                                    @customerID,
                                    DATEADD(DAY, @D1,GETDATE())
                              )
                    END
          END
END
```

13. **AddInvoice** - dodawanie faktury z zamówienia z określonego przedziału czasowego

```
CREATE PROCEDURE [dbo].[AddInvoice]
          @customerID int,
          @fromDate date,
          @toDate date,
          @companyName varchar(30),
          @street varchar(30),
          @city varchar(30),
          @postalCode varchar(30),
          @firstName varchar(30),
          @lastName varchar(30)
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Customers where customerID = @customerID) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'The customer with ID:' + CONVERT(varchar, @customerID) + 'does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          INSERT INTO [dbo].[Invoices]
          (
           issueDate,
           companyName,
           street,
           city,
           postalCode,
           firstName,
           lastName
          )
           VALUES
          (
           CONVERT(DATE,GETDATE()),
           @companyName,
           @street,
           @city,
           @postalCode,
           @firstName,
           @lastName
          )
          UPDATE Orders
          SET invoiceID = SCOPE_IDENTITY()
          WHERE receiveDate >= @fromDate AND receiveDate <= @toDate AND @customerID = customerID
END
```

14. **TakenTables Week** - obliczanie ile było zajętych stolików w wybranym tygodniu

```
CREATE PROCEDURE [dbo].[TakenTablesWeek]
          @from_date date
AS
BEGIN
          SET NOCOUNT ON;

          DECLARE @total int;

          SET @total = 0;

          SELECT COUNT(*) FROM Reservation_Tables
          WHERE CONVERT(DATE, fromTime) BETWEEN @from_date AND DATEADD(DAY, 7, @from_date)
END
```

15. **TakenTablesMonth** - obliczanie ile było zajętych stolików w wybranym miesiącu

```
CREATE PROCEDURE [dbo].[TakenTablesMonth]
          @from_date date
AS
BEGIN
          SET NOCOUNT ON;

          DECLARE @total int;

          SET @total = 0;

          SELECT COUNT(*) FROM Reservation_Tables
          WHERE CONVERT(DATE, fromTime) BETWEEN @from_date AND DATEADD(MONTH, 1, @from_date)
END
```

16. **GenerateSavedMoney** - obliczanie ile zaoszczędził na rabatach dany klient w danym tygodniu

```
CREATE PROCEDURE [dbo].[GenerateSavedMoney]
          @customerID int,
          @from_date datetime
AS
BEGIN
          SET NOCOUNT ON;

          IF ((SELECT COUNT(*) FROM Customers where customerID = @customerID) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customerID) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          SELECT sum(quantity*price)*discount/100
          FROM ORDERS as O
          INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
          INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
          INNER JOIN Menus as M ON M.menuID = PM.menuID
          WHERE @customerID = customerID AND CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate AND CONVERT(DATE,
          receiveDate) BETWEEN @from_date AND DATEADD(DAY, 7, @from_date)
          GROUP BY O.orderID, discount

END
```

17. **GenerateSavedMoneyMonthly** - obliczanie ile oszczędził na rabatach dany klient w danym miesiącu

```
CREATE PROCEDURE [dbo].[GenerateSavedMoneyMonthly]
          @customerID int,
          @from_date datetime
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Customers where customerID = @customerID) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customerID) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END

          SELECT sum(quantity*price)*discount/100
          FROM ORDERS as O
          INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
          INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
          INNER JOIN Menus as M ON M.menuID = PM.menuID
          WHERE @customerID = customerID AND CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate AND CONVERT(DATE,
          receiveDate) BETWEEN @from_date AND DATEADD(MONTH, 1, @from_date)
          GROUP BY O.orderID, discount

END
```

18. **OrdersInfoWeek** - obliczanie ile zamówień i na jaką kwotę złożył dany klient w danym tygodniu

```
CREATE PROCEDURE [dbo].[OrdersInfoWeek]
          @from_date date,
          @customer_id int
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Customers where customerID = @customer_id) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customer_id) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          DECLARE @count int;

          SET @count = 0;

          SELECT @count=@count+1 FROM Orders
          WHERE customerID = @customer_id AND CONVERT(DATE, orderDate) BETWEEN @from_date AND DATEADD(MONTH, 1, @from_date)

          SELECT sum(quantity*price)*(1-discount/100)
          FROM ORDERS as O
          INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
          INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
          INNER JOIN Menus as M ON M.menuID = PM.menuID
          WHERE CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate AND CONVERT(DATE, receiveDate) BETWEEN @from_date AND
          DATEADD(MONTH, 1, @from_date) AND o.customerID = @customer_id
          GROUP BY O.orderID, discount

END
```

19. **OrdersInfoMonth** - obliczanie ile zamówień i na jaką kwotę złożył dany klient w danym miesiącu

```
CREATE PROCEDURE [dbo].[OrdersInfoMonth]
          @customer_id int,
          @from_date date
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Customers where customerID = @customer_id) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customer_id) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          DECLARE @count int;

          SET @count = 0;

          SELECT @count=@count+1 FROM Orders
          WHERE customerID = @customer_id AND CONVERT(DATE, orderDate) BETWEEN @from_date AND DATEADD(MONTH, 1, @from_date)

          SELECT sum(quantity*price)*(1-discount/100)
          FROM ORDERS as O
          INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
          INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
          INNER JOIN Menus as M ON M.menuID = PM.menuID
          WHERE CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate AND CONVERT(DATE, receiveDate) BETWEEN @from_date AND
          DATEADD(MONTH, 1, @from_date) AND o.customerID = @customer_id
          GROUP BY O.orderID, discount
END
```

20. **ModifyCustomerAdressData** - zmiana danych klienta (np. po przeprowadzce)

```
CREATE PROCEDURE [dbo].[ModifyCustomerAdressData]
          @customerID int,
          @city varchar(30),
          @street varchar(30),
          @postalCode int
AS
BEGIN
          SET NOCOUNT ON;
          IF ((SELECT COUNT(*) FROM Customers where customerID = @customerID) < 1)
          BEGIN
                    DECLARE @msg NVARCHAR(2048) = 'Customer with ID:' + CONVERT(varchar, @customerID) + ' does not exist' ;
                    ;THROW 52000, @msg,1
                    RETURN
          END
          UPDATE Customers
          SET city = @city, street = @street, postalCode = @postalCode
          WHERE customerID = @customerID
END
```

21. **MenuInDate -** pokazuje menu w wybranej dacie

```
CREATE PROCEDURE [dbo].[MenuInDate]
          @date date = NULL
AS
BEGIN
          IF (@date = NULL)
          BEGIN
                    ;THROW 52000, 'No date',1
                    RETURN
          END
          SET NOCOUNT ON;

          SELECT Products_In_Menu.menuID,Products_In_Menu.productID,productName, price
          FROM Products_In_Menu
          JOIN Products ON Products_in_menu.productID = Products.productID
          JOIN Menus ON Products_In_Menu.menuID = Menus.menuID
          WHERE Menus.startDate <= @date AND Menus.endDate >= @date
END
```

22. **SeaFoodInDate** - pokazuje owoce morza z menu w wybranej dacie

```
ALTER PROCEDURE [dbo].[SeaFoodInDate]
          @date date = NULL
AS
BEGIN
          IF (@date = NULL)
          BEGIN
                    ;THROW 52000, 'No date',1
                    RETURN
          END
          SET NOCOUNT ON;

          SELECT productName, price
          FROM Products_In_Menu
          JOIN Products ON Products_in_menu.productID = Products.productID
          JOIN Menus ON Products_In_Menu.menuID = Menus.menuID
          JOIN Categories ON Products.categoryID = Categories.categoryID
          WHERE Menus.startDate <= @date AND Menus.endDate >= @date AND Categories.categoryID = 6
END
```

23. ModifyOrder - umożliwia edycję pól w zamówieniu (np zmiana statusu)

Procedura ModifyOrder:

*jest potrzebna, bo zawsze dla Reservation tworzymy Order

*uzupełnia dane, których do tej pory nie było w Order

```
CREATE PROCEDURE [dbo].[ModifyOrder]
          @orderID int,
          @productsID varchar(500) = NULL,
          @employee_id int = NULL,
          @receiveDate date = NULL,
          @paid bit = NULL,
          @discount int = NULL,
          @invoice_id int = NULL
AS
BEGIN
          SET NOCOUNT ON;
          IF (@employee_id is not null)
          BEGIN
                    IF ((SELECT COUNT(*) FROM Employees where employeeID = @employee_id) < 1)
                    BEGIN
                              DECLARE @msg NVARCHAR(2048) = 'Employee with ID:' + CONVERT(varchar, @employee_id) + ' does not exist' ;
                              ;THROW 52000, @msg,1
                              RETURN
                    END
                    UPDATE Orders
                    SET employeeID = @employee_id
                    WHERE orderID = @orderID
          END
          IF (@receiveDate is not null)
          BEGIN
                    UPDATE Orders
                    SET receiveDate = @receiveDate
                    WHERE orderID = @orderID
          END
          IF (@paid is not null)
          BEGIN
                    UPDATE Orders
                    SET paid = @paid
                    WHERE orderID = @orderID
          END
          IF (@invoice_id is not null)
          BEGIN
                    UPDATE Orders
                    SET invoiceID = @invoice_id
                    WHERE orderID = @orderID
          END
          IF (@discount is not null)
          BEGIN
                    UPDATE Orders
                    SET discount = @discount
                    WHERE orderID = @orderID
          END
          IF (@productsID is not null)
          BEGIN
                    BEGIN TRANSACTION
                    DECLARE @prID as int;
                    DECLARE @ctr as int;
                    SET @ctr = (SELECT COUNT(*) FROM STRING_SPLIT(@productsID,','));
                    WHILE @ctr >= 1
                    BEGIN
                              SET @prID = CAST(((SELECT TOP (@ctr) * FROM (SELECT value FROM STRING_SPLIT(@productsID,',')) as p) EXCEPT (SELECT TOP
                              (@ctr-1) * FROM (SELECT value FROM STRING_SPLIT(@productsID,','))as d))as int);
                              EXEC AddProductToOrder @orderID, @prID; -- rollback inside if failed
                              SET @ctr = @ctr-1;
                    END
                    COMMIT
          END
END
```

5.Widoki

1. **OrdersCount** - wyświetl spis klientów z ilością złożonych zamówień

```
CREATE VIEW [dbo].[OrdersCount]
AS SELECT Customers.customerID, (IC.firstName+' '+IC.lastName) as customer_name, C.companyName, COUNT(*) as cnt
FROM Customers
JOIN Orders ON Customers.customerID = Orders.customerID
LEFT OUTER JOIN Individual_Customers as IC ON IC.customerID = Customers.customerID
LEFT OUTER JOIN Companies as C ON C.customerID = Customers.customerID
GROUP BY Customers.customerID, IC.firstName, IC.lastName, C.companyName
GO
```

2. **MoneySpend** - wyświetl spis klientów z łączną ceną złożonych zamówień

```
CREATE VIEW [dbo].[MoneySpend]
AS SELECT Customers.customerID,(IC.firstName+' '+IC.lastName) as customer_name, C.companyName,sum(quantity*price)*(1-discount/100) as
sum
    FROM Orders as O
          JOIN Customers ON Customers.customerID = O.customerID
    JOIN Order_Details as OD ON O.orderID = OD.orderID
    JOIN Products_In_Menu as PM ON PM.productID = OD.productID
    JOIN Menus as M ON M.menuID = PM.menuID
          LEFT OUTER JOIN Individual_Customers as IC ON IC.customerID = Customers.customerID
          LEFT OUTER JOIN Companies as C ON C.customerID = Customers.customerID
    GROUP BY O.orderID, discount,Customers.customerID,IC.firstName,IC.lastName, C.companyName

GO
```

3. **LatePayments** - wyświetl spis klientów, którzy nie uregulowali jeszcze wszystkich płatności

```
CREATE VIEW [dbo].[LatePayments]
AS SELECT Customers.customerID, (IC.firstName+' '+IC.lastName) as customer_name, C.companyName, SUM(dbo.GetOrderValue(Orders.orderID,
discount)) as to_pay
FROM Customers
JOIN Orders ON Customers.customerID = Orders.customerID
LEFT OUTER JOIN Individual_Customers as IC ON IC.customerID = Customers.customerID
LEFT OUTER JOIN Companies as C ON C.customerID = Customers.customerID
WHERE paid = 0
GROUP BY Customers.customerID, IC.firstName, IC.lastName, C.companyName
GO
```

4. **AllCompanies** - wyświetla spis wszystkich klientów firmowych

```
CREATE VIEW [dbo].[AllCompanies]
AS
SELECT dbo.Companies.customerID, dbo.Companies.companyName, dbo.Companies.nip, dbo.Customers.street, dbo.Customers.city,
dbo.Customers.postalCode
FROM     dbo.Customers
INNER JOIN dbo.Companies ON dbo.Customers.customerID = dbo.Companies.customerID
GO
```

5. **AllIndividualCustomers** - wyświetla spis wszystkich klientów indywidualnych

```
CREATE VIEW [dbo].[AllIndividualCustomers]
AS
SELECT dbo.Individual_Customers.customerID, dbo.Individual_Customers.firstName, dbo.Individual_Customers.lastName,
dbo.Customers.street, dbo.Customers.city, dbo.Customers.postalCode
FROM     dbo.Individual_Customers
INNER JOIN dbo.Customers ON dbo.Individual_Customers.customerID = dbo.Customers.customerID
GO
```

6. **AllEmployees** - wyświetla spis wszystkich pracowników

```
CREATE VIEW [dbo].[AllEmployees]
AS
SELECT employeeID, firstName, lastName
FROM     dbo.Employees
GO
```

7. **CustomerDiscounts** - wyświetla spis zniżek

```
CREATE VIEW [dbo].[CustomerDiscounts]
AS
SELECT dbo.Customers.customerID, dbo.Discounts.discountID
FROM     dbo.Customers
LEFT OUTER JOIN  dbo.Discounts ON dbo.Customers.customerID = dbo.Discounts.customerID
GO
```

8. **SpareTables** - wyświetla wolne stoliki w danym momencie

```
CREATE VIEW [dbo].[SpareTables]
AS
SELECT dbo.[Tables].tableID, dbo.[Tables].seats
FROM     dbo.[Tables]
INNER JOIN dbo.Reservation_Tables ON dbo.[Tables].tableID = dbo.Reservation_Tables.tableID
WHERE  (GETDATE() NOT BETWEEN dbo.Reservation_Tables.fromTime AND dbo.Reservation_Tables.toTime)
GO
```

6.Funkcje

1.**GetOrderValue** - zwraca wartość zamówienia o danym orderID bez zniżek

```
CREATE FUNCTION [dbo].[GetOrderValue]
(
          @orderID int,
          @discount int  = 0
)
RETURNS int
AS
BEGIN
          IF(@discount is null)
          BEGIN
                    set @discount = 0
          END
          DECLARE @value int
          SET @value = (
          SELECT sum(quantity*price *(1-(@discount/100)))
          FROM ORDERS as O
          INNER JOIN Order_Details as OD ON O.orderID = OD.orderID
          INNER JOIN Products_In_Menu as PM ON PM.productID = OD.productID
          INNER JOIN Menus as M ON M.menuID = PM.menuID
          WHERE @orderID = O.orderID AND CONVERT(DATE, O.orderDate) BETWEEN M.startDate AND M.endDate
          GROUP BY O.orderID
          )
          RETURN @value
END
```

7. Indeksy

```
CREATE INDEX ix_discounts_expireDate ON Discounts(expireDate);
CREATE INDEX ix_tables_seats ON [Tables](seats);
CREATE INDEX ix_res_tables_time ON [Reservation_Tables](toTime,fromTime);
CREATE INDEX ix_menu_date ON Menus(startDate,endDate);
```

8.Uprawnienia

1.Administrator systemu
- dostęp i modyfikacja wszystkich tabel
- przyznawanie i modyfikacja ról użytkownikom

2.Manager restauracji
- Wszystkie uprawnienia które posiada Kelner
- Dostęp do widoków:
    - AllCompanies
    - AllEmployees
    - AllIndividualCustomers
    - MoneySpend
    - OrdersCount
- Dostęp do procedur:
    - AddEmployee
    - TakenTablesWeek/Month
    - OrdersInfoWeek/Month
    - GenerateSavedMoney
    - AddProductToMenu
    - AddProduct
    - AddMenu

3.Kelner
- Dostęp do widoków:
    - SpareTables
    - CustomerDiscounts
    - LatePayments
- Dostęp do procedur:
    - AddCompany
    - AddIndividualCustomer
    - AddInvoice
    - AddOrder
    - AddReservation
    - ModifyOrder
    - AddProductToOrder
    - ModifyCustomerAdressData

4.Klient
- Dostęp do procedur:
    - ModifyCustomerAdressData
    - GenerateSavedMoney
    - OrdersInfoWeek/Month