Projekt bazy danych firmy świadczącej usługi gastronomiczne

Podstawy Baz Danych

Arnold Kokot
Daniel Sopel
Grzegorz Czernecki

Kraków, 2021

Spis treści

Kraków, 2021

Kraków, 2021

1. Wstęp

Projekt bazy danych firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm który przechowuje dane na temat gości, umożliwia rezerwację stolików, generuje raporty finansowe restauracji, automatyzuje zmiany w oferowanym menu. Projekt został zaimplementowany przy użyciu MS SQL Server oraz portalu Azure.

2. Opis funkcji systemu

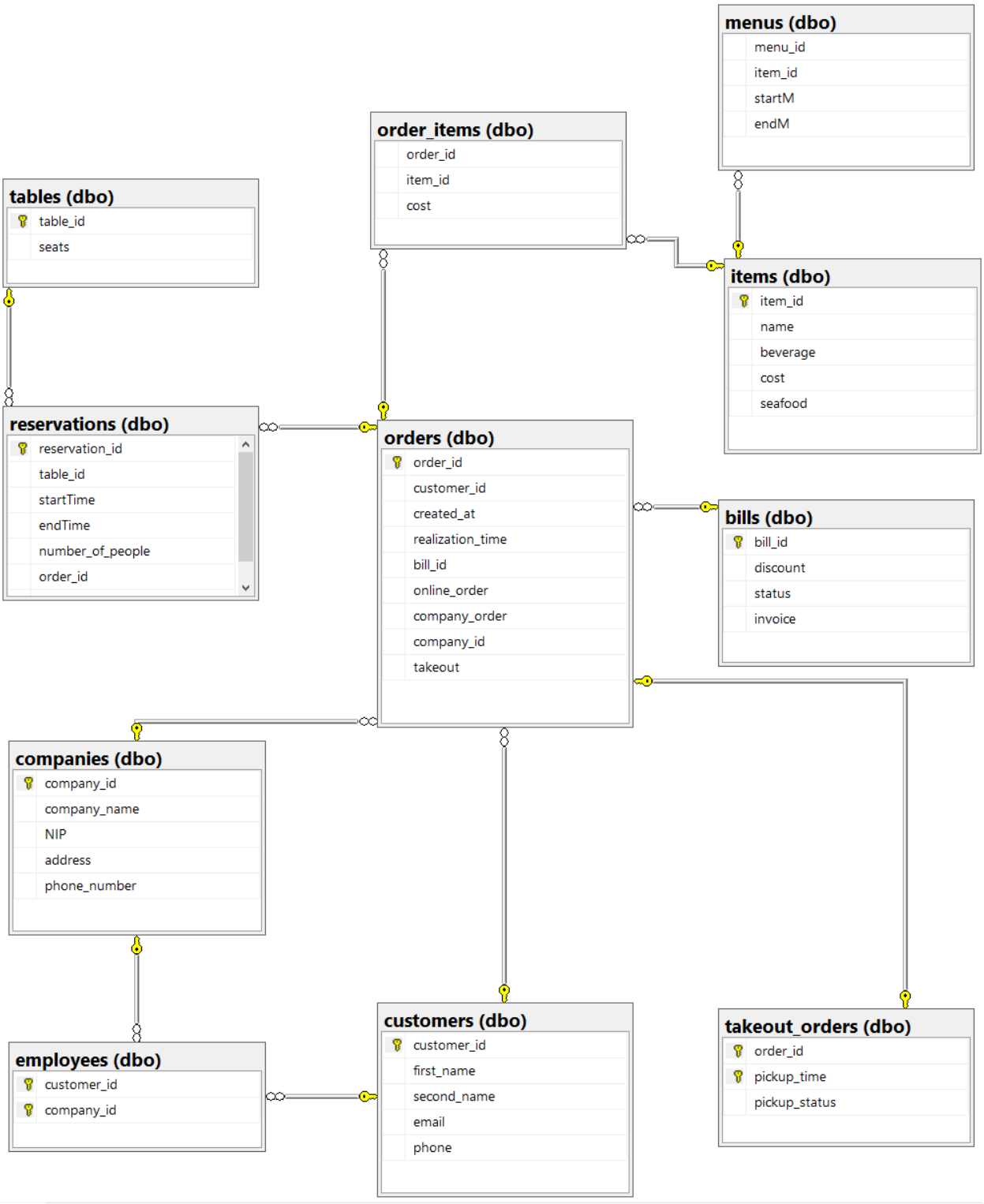
Główną funkcją systemu jest wspieranie codziennej działalności restauracji poprzez udostępnianie im możliwości wykonywania operacji na danych w tabeli, zależnie od pozycji w firmie użytkownik może:

- Klient
    - Tworzyć i edytować wybrane elementy zamówień, rezerwacji oraz konta klienta
    - Sprawdzać swoje zamówienia oraz rabaty
- Pracownik
    - Generować oraz edytować nowe menu
    - Zmieniać ilość dostępnych miejsc
    - Wyświetlać status zamówień oraz wystawione faktury
    - Generować rachunek klienta
    - Edytować i wprowadzać rabaty klienta
- Menadżer
    - Generować raporty miesięczne
    - Wyświetlać statystyki prezentujące działalność restauracji
    - Dodawać, edytować, usuwać pracowników
    - Wprowadzać zmiany w strukturze bazy danych

Istnieje także użytkownik który posiada rolę “Administrator” i posiada pełny dostęp do manipulowania bazą danych i jej strukturą.

Kraków, 2021

3. Schemat bazy danych



Kraków, 2021

4. Opis tabel

4.1. Tabela “orders”

Krotki w tabeli reprezentują pojedyncze zamówienia składane przez klientów

Nazwa kolumny	**Opis**	**Typ**	**Warunki Integralności**
order_id	Identyfikator zamówienia	int	PK, NOT NULL IDENTITY(1,1)
customer_id	Identyfikator klienta zamawiającego	int	FK , NOT NULL
created_at	Data utworzenia	datetime	default: now() NOT NULL
realization_time	Data planowanego wykonania zamówienia	datetime	default: NULL == ASAP
bill_id	Identyfikator rachunku powiązanego z zamówieniem	int	FK , NOT NULL
online_order	Zamówienie online	bit	default: 0
company_order	Zamówienie firmowe	bit	default: 0
company_id	Identyfikator firmy na którą jest zamówienie	int	FK Default: NULL == not a Company order,
takeout	Zamówienie na wynos	bit	default: 0

```sql
CREATE TABLE [dbo].[orders](
    [order_id] [int] IDENTITY(1,1) NOT NULL,
    [customer_id] [int] NOT NULL,
    [created_at] [smalldatetime] NOT NULL,
    [realization_time] [smalldatetime] NULL,
    [bill_id] [int] NOT NULL,
    [online_order] [bit] NOT NULL,
    [company_order] [bit] NOT NULL,
    [company_id] [int] NULL,
    [takeout] [bit] NOT NULL,
 CONSTRAINT [PK_orders] PRIMARY KEY CLUSTERED
(
    [order_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[orders] ADD  CONSTRAINT [DF_orders_online_order]  DEFAULT ((0)) FOR [online_order]
GO

ALTER TABLE [dbo].[orders] ADD  CONSTRAINT [DF_orders_company_order]  DEFAULT ((0)) FOR [company_order]
GO

ALTER TABLE [dbo].[orders] ADD  CONSTRAINT [DF_orders_takeout]  DEFAULT ((0)) FOR [takeout]
GO

ALTER TABLE [dbo].[orders]  WITH CHECK ADD  CONSTRAINT [FK_orders_bills] FOREIGN KEY([bill_id])
REFERENCES [dbo].[bills] ([bill_id])
GO

ALTER TABLE [dbo].[orders] CHECK CONSTRAINT [FK_orders_bills]
GO

ALTER TABLE [dbo].[orders]  WITH CHECK ADD  CONSTRAINT [FK_orders_companies] FOREIGN KEY([company_id])
REFERENCES [dbo].[companies] ([company_id])
GO

ALTER TABLE [dbo].[orders] CHECK CONSTRAINT [FK_orders_companies]
GO

ALTER TABLE [dbo].[orders]  WITH CHECK ADD  CONSTRAINT [FK_orders_customers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[customers] ([customer_id])
GO

ALTER TABLE [dbo].[orders] CHECK CONSTRAINT [FK_orders_customers]
GO
```

4.2. Tabela “takeout_orders”

Krotki w tabeli reprezentują zamówienia złożone online

Nazwa kolumny	Opis	Typ	Warunki Integralności
order_id	Identyfikator zamówienia	int	PK , FK NOT NULL
pickup_time	Czas odbioru	datetime	PK default: now()+0,5h
pickup_status	Status odbioru	bit	default : 0

Kraków, 2021

```
CREATE TABLE [dbo].[takeout_orders](
    [order_id] [int] NOT NULL,
    [pickup_time] [datetime] NOT NULL,
    [pickup_status] [bit] NOT NULL,
 CONSTRAINT [PK_takeout_orders] PRIMARY KEY CLUSTERED
(
    [order_id] ASC,
    [pickup_time] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_takeout_orders] UNIQUE NONCLUSTERED
(
    [order_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[takeout_orders] ADD  CONSTRAINT [DF_takeout_orders_pickup_status]  DEFAULT ((0)) FOR [pickup_status]
GO

ALTER TABLE [dbo].[takeout_orders]  WITH CHECK ADD  CONSTRAINT [FK_takeout_orders_orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[orders] ([order_id])
GO

ALTER TABLE [dbo].[takeout_orders] CHECK CONSTRAINT [FK_takeout_orders_orders]
GO
```

4.3. Tabela “bills”

Krotki w tabeli reprezentują rachunki wystawiane klientom

Nazwa kolumny	Opis	Typ	Warunki Integralności
bill_id	Identyfikator rachunku	int	PK, NOT NULL IDENTITY(1,1)
discount	Przyznany rabat	int	NOT NULL default: 0
status	Status opłacenia zamówienia, 0 == Nieopłacone	bit	NOT NULL default: 0
invoice	Rachunek na fakturę	bit	NOT NULL default: 0

```
CREATE TABLE [dbo].[bills](
    [bill_id] [int] IDENTITY(1,1) NOT NULL,
    [discount] [int] NOT NULL,
    [status] [bit] NOT NULL,
    [invoice] [bit] NOT NULL,
 CONSTRAINT [PK_bills] PRIMARY KEY CLUSTERED
(
    [bill_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[bills] ADD  CONSTRAINT [DF_bills_discount]  DEFAULT ((0)) FOR [discount]
GO

ALTER TABLE [dbo].[bills] ADD  CONSTRAINT [DF_bills_status]  DEFAULT ((0)) FOR [status]
GO

ALTER TABLE [dbo].[bills] ADD  CONSTRAINT [DF_bills_invoice]  DEFAULT ((0)) FOR [invoice]
GO
```

4.4. Tabela “order_items”

Krotki w tabeli reprezentują zamówione pozycje

Nazwa kolumny	Opis	Typ	Warunki Integralności
order_id	Identyfikator zamówienia	int	PK, FK NOT NULL
item_id	Identyfikator pozycji z menu	int	FK NOT NULL
cost	Koszt pozycji w momencie składania zamówienia	int	cost >= 0, default: 0

```
CREATE TABLE [dbo].[order_items](
    [order_id] [int] NOT NULL,
    [item_id] [int] NOT NULL,
    [cost] [int] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[order_items] ADD  CONSTRAINT [DF_order_items_cost]  DEFAULT ((0)) FOR [cost]
GO

ALTER TABLE [dbo].[order_items]  WITH CHECK ADD  CONSTRAINT [FK_order_items_items] FOREIGN KEY([item_id])
REFERENCES [dbo].[items] ([item_id])
GO

ALTER TABLE [dbo].[order_items] CHECK CONSTRAINT [FK_order_items_items]
GO

ALTER TABLE [dbo].[order_items]  WITH CHECK ADD  CONSTRAINT [FK_order_items_orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[orders] ([order_id])
GO

ALTER TABLE [dbo].[order_items] CHECK CONSTRAINT [FK_order_items_orders]
GO
```

4.5. Tabela “items”

Krotki w tabeli wszystkie pozycje w ofercie restauracji

Nazwa kolumny	Opis	Typ	Warunki Integralności
item_id	Identyfikator pozycji	int	PK,IDENTITY(1,1) NOT NULL
name	Nazwa pozycji	varchar(50)	NOT NULL UNIQUE
beverage	Czy pozycja jest napojem	bit	default: 0 NOT NULL
cost	Koszt pozycji	float	cost >= 0, default: 0 NOT NULL
seafood	Pozycja z owoców morza	bit	default: 0 NOT NULL

```
CREATE TABLE [dbo].[items](
    [item_id] [int] IDENTITY(1,1) NOT NULL,
    [name] [varchar](50) NOT NULL,
    [beverage] [bit] NOT NULL,
    [cost] [float] NOT NULL,
    [seafood] [bit] NOT NULL,
 CONSTRAINT [PK_items] PRIMARY KEY CLUSTERED
(
    [item_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_items] UNIQUE NONCLUSTERED
(
    [name] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[items] ADD  CONSTRAINT [DF_items_beverage]  DEFAULT ((0)) FOR [beverage]
GO

ALTER TABLE [dbo].[items] ADD  CONSTRAINT [DF_items_cost]  DEFAULT ((0)) FOR [cost]
GO

ALTER TABLE [dbo].[items] ADD  CONSTRAINT [DF_items_seafood]  DEFAULT ((0)) FOR [seafood]
GO
```

4.6. Tabela “menus”

Krotki w tabeli reprezentują dostępność w czasie pozycji w menu

Nazwa kolumny	Opis	Typ	Warunki Integralności
menu_id	Identyfikator menu	int	PK, NOT NULL IDENTITY(1,1)

Kraków, 2021

item_id	Identyfikator pozycji	int	FK, NOT NULL
start	Czas dołączenia pozycji do oferty	datetime	NOT NULL start >= now()
end	Czas odłączenia pozycji z oferty	datetime	end >= start

```
CREATE TABLE [dbo].[menus](
    [menu_id] [int] NOT NULL,
    [item_id] [int] NOT NULL,
    [startM] [smalldatetime] NOT NULL,
    [endM] [smalldatetime] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[menus]  WITH CHECK ADD  CONSTRAINT [FK_menus_items] FOREIGN KEY([item_id])
REFERENCES [dbo].[items] ([item_id])
GO

ALTER TABLE [dbo].[menus] CHECK CONSTRAINT [FK_menus_items]
GO
```

4.7. Tabela “customers”

Krotki w tabeli reprezentują gości restauracji

Nazwa kolumny	**Opis**	**Typ**	**Warunki Integralności**
customer_id	Unikalny klucz główny	int	PK NOT NULL IDENTITY(1,1)
first_name	Imię	varchar(50)	NOT NULL Do 50 znaków
second_name	Nazwisko	varchar(50)	Do 50 znaków
email	Adres mailowy	varchar(50)	UNIQUE Do 50 znaków
phone	Numer kontaktowy	varchar(16)	UNIQUE Do16 znaków NOT NULL

```
CREATE TABLE [dbo].[customers](
    [customer_id] [int] IDENTITY(1,1) NOT NULL,
    [first_name] [varchar](50) NOT NULL,
    [second_name] [varchar](50) NULL,
    [email] [varchar](50) NOT NULL,
    [phone] [varchar](16) NOT NULL,
 CONSTRAINT [PK_customers] PRIMARY KEY CLUSTERED
(
    [customer_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_customers] UNIQUE NONCLUSTERED
(
    [email] ASC,
    [phone] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

4.8. Tabela “employees”

Krotki w tabeli reprezentują połączenia pracowników z firmami

Nazwa kolumny	Opis	Typ	Warunki Integralności
customer_id	Identyfikator gościa	int	PK, FK NOT NULL
company_id	Identyfikator firmy	int	PK,FK NOT NULL

```
CREATE TABLE [dbo].[employees](
    [customer_id] [int] NOT NULL,
    [company_id] [int] NOT NULL,
 CONSTRAINT [PK_employees] PRIMARY KEY CLUSTERED
(
    [customer_id] ASC,
    [company_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[employees]  WITH NOCHECK ADD  CONSTRAINT [FK_employees_companies] FOREIGN KEY([company_id])
REFERENCES [dbo].[companies] ([company_id])
GO

ALTER TABLE [dbo].[employees] CHECK CONSTRAINT [FK_employees_companies]
GO

ALTER TABLE [dbo].[employees]  WITH NOCHECK ADD  CONSTRAINT [FK_employees_customers] FOREIGN KEY([customer_id])
REFERENCES [dbo].[customers] ([customer_id])
GO

ALTER TABLE [dbo].[employees] CHECK CONSTRAINT [FK_employees_customers]
GO
```

4.9. Tabela “companies”

Krotki w tabeli reprezentują firmy

Nazwa kolumny	Opis	Typ	Warunki Integralności
company_id	Unikalny klucz główny	int	PK NOT NULL IDENTITY(1,1)
company_name	Nazwa firmy	Varchar (100)	UNIQUE NOT NULL
NIP	NIP do faktury	Varchar(15)	UNIQUE NOT NULL
address	Adres firmy	Varchar (200)	NOT NULL
phone_number	Telefon kontaktowy firmy	Varchar(16)	UNIQUE NOT NULL

```
CREATE TABLE [dbo].[companies](
    [company_id] [int] IDENTITY(1,1) NOT NULL,
    [company_name] [varchar](100) NOT NULL,
    [NIP] [varchar](15) NOT NULL,
    [address] [varchar](200) NOT NULL,
    [phone_number] [varchar](50) NOT NULL,
 CONSTRAINT [PK_companies] PRIMARY KEY CLUSTERED
(
    [company_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_companies] UNIQUE NONCLUSTERED
(
    [NIP] ASC,
    [phone_number] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

4.10. Tabela “reservations”

Krotki w tabeli reprezentują rezerwacje dokonane przez klientów

Nazwa kolumny	Opis	Typ	Warunki Integralności
reservation_id	Identyfikator rezerwacji	int	PK IDENTITY(1,1) NOT NULL
table_id	Identyfikator stolika	smallint	FK NOT NULL
startTime	Czas rozpoczęcia rezerwacji	smalldatetime	time >= now() NOT NULL
endTime	Czas zakończenia rezerwacji	smalldatetime	

Kraków, 2021

number_of_people	Ilość zarezerwowanych miejsc	smallint	number_of_people > 0, default: 1 NOT NULL
order_id	Identyfikator zamówienia	int	FK NOT NULL Order_id.cost > 50 or ….
acceptanceStatus	Status zaakceptowania przez personel rezerwacji	bit	NOT NULL
~~customer_id~~	~~Identyfikator klienta tworzącego rezerwację~~	~~int~~	~~FK~~ ~~NOT NULL~~

```
CREATE TABLE [dbo].[reservations](
    [reservation_id] [int] IDENTITY(1,1) NOT NULL,
    [table_id] [smallint] NOT NULL,
    [startTime] [smalldatetime] NOT NULL,
    [endTime] [smalldatetime] NULL,
    [number_of_people] [smallint] NOT NULL,
    [order_id] [int] NOT NULL,
    [acceptanceStatus] [bit] NOT NULL,
 CONSTRAINT [PK_reservations222222222] PRIMARY KEY CLUSTERED
(
    [reservation_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[reservations] ADD  CONSTRAINT [DF_reservations_acceptanceStatus]  DEFAULT ((0)) FOR [acceptanceStatus]
GO

ALTER TABLE [dbo].[reservations]  WITH CHECK ADD  CONSTRAINT [FK_reservations_orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[orders] ([order_id])
GO

ALTER TABLE [dbo].[reservations] CHECK CONSTRAINT [FK_reservations_orders]
GO

ALTER TABLE [dbo].[reservations]  WITH CHECK ADD  CONSTRAINT [FK_reservations_tables] FOREIGN KEY([table_id])
REFERENCES [dbo].[tables] ([table_id])
GO

ALTER TABLE [dbo].[reservations] CHECK CONSTRAINT [FK_reservations_tables]
GO
```

4.11. Tabela “tables”

Krotki w tabeli reprezentują stoliki w restauracji wraz z dostępnymi przy nich miejscami.

Nazwa kolumny	**Opis**	**Typ**	**Warunki Integralności**
table_id	Identyfikator stolika	smallint	PK NOT NULL
seats	Ilość miejsc przy stoliku	smallint	NOT NULL default : 1

```
CREATE TABLE [dbo].[tables](
    [table_id] [smallint] IDENTITY(1,1) NOT NULL,
    [seats] [smallint] NOT NULL,
 CONSTRAINT [PK_tables] PRIMARY KEY CLUSTERED
(
    [table_id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

5. Procedury

5.1. Procedury wprowadzające dane

5.1.1. dbo.addBill

Procedura używana podczas wystawiania przez kelnera rachunku klientowi.

```
CREATE PROCEDURE [dbo].[addBill](
    @bill_id [int] = NULL OUT,
    @discount [int],
    @status [bit],
    @invoice [bit]
)
AS
BEGIN
    SET NOCOUNT ON

    IF @bill_id IS NULL THROW 51000, 'Id cant be null', 1
    IF @discount IS NULL THROW 51000, 'Discount cant be null', 1
    IF @status IS NULL THROW 51000, 'Status cant be null', 1
    IF @invoice IS NULL THROW 51000, 'Invoice cant be null', 1

    INSERT INTO dbo.bills(
        --bill_id,
        discount,
        status,
        invoice
    )
    VALUES (
        --@bill_id,
        @discount,
        @status,
        @invoice
    )
    SET @bill_id = SCOPE_IDENTITY()
END
GO
```

5.1.2. addCompany

Procedura używana do zapisania danych nowej firmy.

```
CREATE PROCEDURE [dbo].[addCompany](
    @company_name [varchar](100),
    @NIP [varchar](15),
    @address [varchar](200),
    @phone_number [varchar](50)
)
AS
BEGIN
    SET NOCOUNT ON

    IF @company_name IS NULL THROW 51000, 'Pusta nazwa firmy', 1
    IF @NIP IS NULL THROW 51000, 'Niepoprawny NIP', 1
    IF @address IS NULL THROW 51000, 'Pusty adres', 1
    IF @phone_number IS NULL THROW 51000, 'Pusty numer telefonu', 1

    INSERT INTO dbo.companies(
        company_name,
        NIP,
        address,
        phone_number
    )
    VALUES (
        @company_name,
        @NIP,
        @address,
        @phone_number
    )

END
GO
```

5.1.3. addReservation

Procedura używana zarejestrowania rezerwacji na danego klienta

```
CREATE PROCEDURE [dbo].[addReservation](
    @reservation_id int = NULL OUT,
    @table_id smallint,
    @customer_id smallint,
    @startTime datetime,
    @number_of_people smallint,
    @order_id int,
    @acceptanceStatus bit
)
AS
BEGIN
    IF @table_id IS NULL THROW 51000, 'Podaj table_id', 1
    IF @startTime IS NULL THROW 51000, 'Podaj date rezerwacji', 1
    IF @number_of_people IS NULL OR @number_of_people < 2 THROW 51000, 'Niepoprawna ilość osób', 1 --minimum 2 osoby
    IF @order_id IS NULL THROW 51000, 'Przed utworzeniem rezerwacji stwórz zamówienie', 1

    IF(dbo.callculateOrderCost(@order_id) < 200)
    BEGIN
        IF(dbo.callculateOrderCost(@order_id) < 50 AND dbo.calculateAmountOfIndividualCustomerOrders(@customer_id) < 5)
        THROW 51000, 'Nie spełniasz wymagań na utworzenie rezerwacji', 1
    END

    INSERT INTO [dbo].[reservations]
           ([table_id]
           ,[startTime]
           ,[endTime]
           ,[number_of_people]
           ,[order_id]
           ,[acceptanceStatus])
     VALUES
           (@table_id,
            @startTime,
            DATEADD(HOUR, 2, @startTime), -- czas trwania rezerwacji
            @number_of_people,
            @order_id,
            @acceptanceStatus)
    SET @reservation_id = SCOPE_IDENTITY();

END
GO
```

5.1.4. addOrder

Procedura używana do zarejestrowania nowego zamówienia w systemie oraz zweryfikowania poprawności zamówienia potraw morskich.

```
CREATE PROCEDURE [dbo].[sp_addOrder]
(
    @order_id int ,
    @customer_id int,
    @created_at smalldatetime,
    @realization_time smalldatetime,
    @online_order bit,
    @company_order bit,
    @bill_id int,
    @company_id int,
    @takeout bit,
    @status bit,
    @invoice bit
)
AS
BEGIN

    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @amountOfOrdersForDiscount  INT
    Set @amountOfOrdersForDiscount  = 10

    DECLARE @repetitiveDiscount INT
    Set @repetitiveDiscount  = 3

    IF @customer_id IS NULL
    THROW 51000, '@No such customer', 1
        -- Or exec addCustomer

    IF @created_at IS NULL
    THROW 51000, '@created_at is null', 1

    IF @bill_id IS NULL
    THROW 51000, 'Could not register oreder because there is no matching bill', 1
        -- Or exec addBill

    IF @online_order IS NULL
    THROW 51000, '@online_order is null', 1

    IF @company_order IS NULL
    THROW 51000, '@company_order is null', 1

    IF @company_order = 1

        IF @company_id IS NULL
        THROW 51000, '@company_id is null', 1
        -- Or exec addCompany

    IF @takeout IS NULL
    THROW 51000, '@takeout is null', 1

    IF ([dbo].[checkIfSeafoodAreAvailable](@order_id) = 0 )
    THROW 51000, '@the order has seafood which is unavailable today', 1
```

```
    IF ([dbo].[calculateAmountOfIndividualCustomerOrders](@customer_id) >= @amountOfOrdersForDiscount)

        INSERT INTO bills
            (bill_id,
            discount,
            [status],
            invoice)
        VALUES
            (@bill_id
            ,@repetitiveDiscount
            ,@status
            ,@invoice)


    INSERT INTO orders
        (customer_id
        ,created_at
        ,realization_time
        ,online_order
        ,company_order
        ,company_id
        ,takeout
        ,bill_id)
    VALUES
        (@customer_id
        ,@created_at
        ,@realization_time
        ,@online_order
        ,@company_order
        ,@company_id
        ,@takeout
        ,@bill_id)

    SET @order_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.1.5. addCustomer

Procedura służąca do rejestrowania oraz walidacji poprawności danych klienta

```
CREATE PROCEDURE [dbo].[sp_addCustomer]
(
    @customer_id int = NULL OUT,
    @first_name varchar(50),
    @second_name varchar(50),
    @email varchar(50),
    @phone varchar(16)
)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

      IF @first_name IS NULL
        THROW 51000, 'first name is null', 1

      IF @second_name IS NULL
        THROW 51000, 'secondname is null', 1

      IF @email IS NULL
        THROW 51000, 'email is null', 1

      IF @phone IS NULL
        THROW 51000, 'phone is null', 1

    INSERT INTO customers
           (first_name
           ,second_name
           ,email
           ,phone)
     VALUES
           (@first_name
           ,@second_name
           ,@email
           ,@phone)
    SET @customer_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.1.6. addItems

```sql
CREATE PROCEDURE [dbo].[sp_addItems]
(
    @item_id [int],
    @name [varchar](50),
    @beverage [bit],
    @cost [float],
    @seafood [bit]
)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

      IF @name IS NULL
        THROW 51000, 'first name is null', 1

      IF @beverage IS NULL OR @beverage < 0 OR @beverage > 1
        THROW 51000, 'invalid logical value', 1

      IF @cost IS NULL OR @cost<=0
        THROW 51000, 'cost is null or <=0', 1

      IF @seafood IS NULL OR @seafood < 0 OR @seafood > 1
        THROW 51000, 'invalid logical value', 1

    INSERT INTO items
           (name
           ,beverage
           ,cost
           ,seafood)
     VALUES
           (@name
           ,@beverage
           ,@cost
           ,@seafood)
    SET @item_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.1.7. addTakeoutOrder

Procedura używana do skojarzenia istniejącego zamówienia jako zamówienia na wynos.

```
CREATE PROCEDURE [dbo].[addTakeoutOrder](
    @order_id [int],
    @pickup_time [datetime]
)
AS
BEGIN
    SET NOCOUNT ON

    IF @order_id IS NULL THROW 51000, 'Wpisz ID istniejącego zamówienia', 1
    IF @pickup_time IS NULL THROW 51000, 'Wpisz czas odbioru zamówienia', 1

    INSERT INTO dbo.takeout_orders(
        order_id,
        pickup_time,
        pickup_status
    )
    VALUES (
        @order_id,
        @pickup_time,
        0
    )

END
GO
```

5.2. Procedury modyfikujące dane

5.2.1. addMenu

Procedura używana do zapisania potrawy/napoju do aktualnego lub przyszłego menu restauracji.

```
CREATE PROCEDURE [dbo].[sp_addMenu]
(
    @menu_id [int],
    @item_id [int],
    @startM [smalldatetime],
    @endM [smalldatetime]
)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

      IF @item_id IS NULL
        THROW 51000, 'item_id is null', 1

      IF @startM IS NULL OR @endM IS NULL
        THROW 51000, 'invalid logical value', 1


    INSERT INTO menus
           (menu_id
           ,item_id
           ,startM
           ,endM)
     VALUES
           (@menu_id
           ,@item_id
           ,@startM
           ,@endM)


    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.2.2. cancelReservation

Procedura używana do anulowania istniejących rezerwacji.

```
CREATE PROCEDURE [dbo].[cancelReservation](
    @reservation_id [int]
)
AS
BEGIN
    IF @reservation_id IS NULL THROW 51000, 'Podaj id rezerwacji', 1
    DELETE FROM dbo.reservations WHERE reservation_id = @reservation_id;
END
GO
```

5.2.3. updateTable

Procedura używana zmiany ilości miejsc przy stoliku oraz wyłączenia stolika z dostępnych

```
CREATE PROCEDURE [dbo].[sp_updateTable]
(
    @table_id INT,
    @newCapacity INT
)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

      IF @table_id IS NULL
        THROW 51000, 'Nr stolika jest niepoprawny', 1

      IF @newCapacity IS NULL
        THROW 51000, 'Ilosc miejsc jest niepoprawna', 1

      IF  0 >= @newCapacity
        THROW 51000, 'Ilosc miejsc jest niepoprawna', 1

    UPDATE tables
    SET seats = @newCapacity
    WHERE tables.table_id = @table_id

    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.2.4. addCompany

Procedura używana do zapisania danych nowej firmy

```
CREATE PROCEDURE [dbo].[addCompany](
    @company_name [varchar](100),
    @NIP [varchar](15),
    @address [varchar](200),
    @phone_number [varchar](50)
)
AS
BEGIN
    SET NOCOUNT ON

    IF @company_name IS NULL THROW 51000, 'Pusta nazwa firmy', 1
    IF @NIP IS NULL THROW 51000, 'Niepoprawny NIP', 1
    IF @address IS NULL THROW 51000, 'Pusty adres', 1
    IF @phone_number IS NULL THROW 51000, 'Pusty numer telefonu', 1

    INSERT INTO dbo.companies(
        company_name,
        NIP,
        address,
        phone_number
    )
    VALUES (
        @company_name,
        @NIP,
        @address,
        @phone_number
    )

END
GO
```

5.2.5. editItem

Procedura używana do zmiany danych potrawy lub napoju w rejestrze na podstawie id.

```
CREATE PROCEDURE [dbo].[editItem](
    @item_id [int],
    @name [varchar](50),
    @beverage [bit],
    @cost [float],
    @seafood [bit]
)
AS
BEGIN
    SET NOCOUNT ON

    IF @item_id IS NULL THROW 51000, 'Wpisz ID', 1
    UPDATE dbo.items SET
        name = @name,
        beverage = @beverage,
        cost = @cost,
        seafood = @seafood
    WHERE item_id = @item_id

END
GO
```

5.2.6. addEmployee

Procedura używana do powiązania klienta z tabeli “customers” z firmą (w celu późniejszych zamówień firmowych itp.)

```
CREATE PROCEDURE [dbo].[sp_addEmployees]
(
    @customer_id int = NULL OUT,
    @company_id int = NULL OUT

)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
    BEGIN TRANSACTION;

      IF @customer_id IS NULL
        THROW 51000, 'Customer_id is null', 1

      IF @company_id IS NULL
        THROW 51000, 'Company_id is null', 1


    INSERT INTO employees
           (customer_id
           ,company_id
           )
     VALUES
           (@customer_id
           ,@company_id
           )


    COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW
    END CATCH
END
GO
```

5.3. Procedury zwracające dane

5.3.1. checkTableAvalibility

Procedura do sprawdzania obecnej dostępności stolika

```
CREATE PROCEDURE [dbo].[checkTableAvalibility](
    @table_id [int]
)
AS
BEGIN
    DECLARE @tables INT
    SET @tables = 0
    SELECT @tables = COUNT(*) FROM dbo.availability WHERE table_id = @table_id
    IF @tables > 0 RETURN 1
    ELSE RETURN 0
END
GO
```

6. Triggery

6.1. orders_INSERT

Trigger sprawdza możliwość zaaplikowania rabatu czasowego dla klienta (dokładniejszy opis tej implementacji jest w punkcie 10)

```
CREATE TRIGGER [dbo].[orders_INSERT] ON [dbo].[orders]
    AFTER INSERT,UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @insertedCustomer_id int
    -- Deklaracja czasu trwania discount  w dniach
    DECLARE @durationOfDiscount int
    SET @durationOfDiscount = 5

    -- Deklaracja wymaganej kwoty po ktorej przysluguje znizka
    DECLARE @discountCodition int
    SET @discountCodition = 1000

    -- Pobieranie czasu rozpoczecia znizki - czyli momentu zreazlizowania
    DECLARE @startOfDiscount smalldatetime
    SET @startOfDiscount = CONVERT(smalldatetime,GETDATE())

    SELECT @insertedCustomer_id  = INSERTED.customer_id
    FROM INSERTED
    LEFT JOIN dbo.discounts ON @insertedCustomer_id = dbo.discounts.customer_id

    WHERE
        -- Sprawdzanie czy klient  ma nieoplacone zamowienia
        --dbo.calculateCustomerBalance(INSERTED.customer_id) = NULL

        -- Sprawdzenie czy klient sie kwalifikuje na przyznanie znizki
        dbo.calculateCustomerExpenses(INSERTED.customer_id) >= @discountCodition
        AND
        -- Sprawdzenie czy klient nie otrzymal juz znizki
        dbo.discounts.start_of_discount = NULL

    INSERT INTO discounts
    VALUES(
        @insertedCustomer_id,
        CONVERT(smalldatetime,GETDATE()),
        DATEADD(DAY,@durationOfDiscount, @startOfDiscount))
END
GO

ALTER TABLE [dbo].[orders] ENABLE TRIGGER [orders_INSERT]
GO
```

6.2. tables_INSERT

Trigger sprawdza czy ilość dostępnych stolików nie przekracza zdefiniowanego limitu.

```
CREATE TRIGGER [dbo].[tables_INSERT] ON [dbo].[tables]
    AFTER INSERT,UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @updatedTable_id int

    -- Deklaracja maksymalnej ilosci miejsc w tej restauracji

    DECLARE @currentAmountOfSeats int
    SET @currentAmountOfSeats = [dbo].[calculateCompanyBalance](@updatedTable_id)

    DECLARE @MaxAmountOfSeats smallint
    SET @MaxAmountOfSeats = 79

    SELECT @updatedTable_id  = INSERTED.table_id
    FROM INSERTED


    -- Informowanie uzytkownika o braku spelnionych wymagan
    IF  @currentAmountOfSeats >= @MaxAmountOfSeats
         PRINT N'Ilosc miejsc nie spelnia rygorow sanitarnych ! Maksymalna ilosc siedzen ma wynosic : ' +
         @MaxAmountOfSeats +
         ' a wynosi : ' +
         @currentAmountOfSeats;



END
GO

ALTER TABLE [dbo].[tables] ENABLE TRIGGER [tables_INSERT]
GO
```

7. Funkcje

7.1. avaliableTables (T)

Funkcja zwraca tabelę zawierającą zajęte stoliki w podanym czasie

```
CREATE FUNCTION [dbo].[avaliableTables](
    @date DATETIME
)
RETURNS TABLE
AS
RETURN
    SELECT
        *
    FROM dbo.reservations
    WHERE reservations.startTime <= @date AND @date <=reservations.endTime

GO
```

7.2. getCompanyUnpaidOrders (T)

Funkcja zwraca tabelę zawierającą nieopłacone zamówienia powiązane z daną firmą.

```
CREATE FUNCTION [dbo].[getCompanyUnpaidOrders] (
    @company_id INT
)
RETURNS TABLE AS

    RETURN SELECT order_id
    FROM dbo.orders JOIN dbo.bills ON bills.bill_id = orders.bill_id
    WHERE company_id = @company_id AND status = 0

GO
```

7.3. getCustomerPaidOrders (T)

Funkcja zwraca tabelę zawierającą opłacone zamówienia danego indywidualnego klienta.

```
CREATE FUNCTION [dbo].[getCustomerPaidOrders] (
    @customer_id INT
)
RETURNS TABLE AS

    RETURN SELECT order_id
    FROM dbo.orders JOIN dbo.bills ON bills.bill_id = orders.bill_id
    WHERE
        customer_id = @customer_id
        AND
        status = 1
        AND
        orders.company_order = 0

GO
```

7.4. getCustomerUnpaidOrders (T)

Funkcja zwraca tabelę zawierającą nieopłacone zamówienia danego indywidualnego klienta.

```
CREATE FUNCTION [dbo].[getCustomerUnpaidOrders] (
    @customer_id INT
)
RETURNS TABLE AS

    RETURN SELECT order_id
    FROM dbo.orders JOIN dbo.bills ON bills.bill_id = orders.bill_id
    WHERE customer_id = @customer_id AND status = 0

GO
```

7.5. calculateAmountOfIndividualCustomerOrders

Funkcja zwraca liczbę opłaconych zamówień przez indywidualnego klienta.

```
CREATE FUNCTION [dbo].[calculateAmountOfIndividualCustomerOrders] (
    @customer_id INT

)
RETURNS INT AS
BEGIN
    DECLARE @minimalAmount INT
    SET @minimalAmount = 30

    DECLARE @amountOfOrders INT
    SET @amountOfOrders =
        (
        SELECT COUNT(order_id) FROM [dbo].[getCustomerPaidOrders](@customer_id)
        WHERE [dbo].[callculateOrderCost]([dbo].[getCustomerPaidOrders](@customer_id).order_id) > @minimalAmount
        )

    RETURN @amountOfOrders

END;
GO
```

7.6. calculateCompanyBalance

Funkcja zwraca sumę nieopłaconych zamówień złożonych przez daną firmę.

```
CREATE FUNCTION [dbo].[calculateCompanyBalance] (
    @company_id INT
)
RETURNS INT AS
BEGIN

    DECLARE @unpaidAmount INT
    SET @unpaidAmount = (
        SELECT SUM(cost) AS do_opłacenia
        FROM [dbo].[getCompanyUnpaidOrders](@company_id)
        JOIN order_items ON order_items.order_id = [dbo].[getCompanyUnpaidOrders].order_id)
    RETURN @unpaidAmount

END;
GO
```

7.7. calculateCustomerBalance

Funkcja zwraca sumę nieopłaconych zamówień klienta indywidualnego.

```
CREATE FUNCTION [dbo].[calculateCustomerBalance] (
    @customer_id INT
)
RETURNS INT AS
BEGIN

    DECLARE @amount INT
    SET @amount = (
        SELECT SUM(cost) AS do_opłacenia
        FROM [dbo].[getCustomerUnpaidOrders](@customer_id)
        JOIN order_items ON order_items.order_id = [dbo].[getCustomerUnpaidOrders].order_id)
    RETURN @amount

END;
GO
```

7.8. calculateCustomerExpenses

Funkcja zwraca sumę opłaconych zamówień klienta indywidualnego.

```
CREATE FUNCTION [dbo].[calculateCustomerExpenses] (
    @customer_id INT
)
RETURNS INT AS
BEGIN

    DECLARE @amount INT
    SET @amount = (
        SELECT SUM(cost) AS do_opłacenia
        FROM [dbo].[getCustomerPaidOrders](@customer_id)
        JOIN order_items ON order_items.order_id = [dbo].[getCustomerPaidOrders].order_id)
    RETURN @amount

END;
GO
```

7.9. calculateMonthlyBill

Funkcja zwraca sumę kosztów zamówień danej firmy z danego miesiąca roku.

```
CREATE FUNCTION [dbo].[calculateMonthlyBill] (
    @company_id INT,
    @selectedMonth INT,
    @selectedYear INT
)
RETURNS INT AS

BEGIN

    DECLARE @monthlyBill INT

    SET @monthlyBill = (
        SELECT SUM(cost)
        FROM dbo.order_items
        WHERE order_id = (
            SELECT order_id
            FROM dbo.orders JOIN dbo.bills ON bills.bill_id = orders.bill_id
            WHERE
                MONTH(realization_time) = @selectedMonth
                AND YEAR(realization_time) = @selectedYear
                AND company_id = @company_id
                AND company_order = 1
                AND status = 0
            )
        )

    RETURN @monthlyBill
END;
GO
```

7.10. calculateAmountOfSeats

Funkcja zwraca sumę wszystkich dostępnych miejsc w restauracji.

```
CREATE FUNCTION [dbo].[callculateAmountOfSeats](
    @table_id INT
)
RETURNS INT
AS
BEGIN
    DECLARE @amountOfSeats INT
    SET @amountOfSeats = (SELECT SUM(seats) FROM dbo.tables)


    RETURN @amountOfSeats
END
GO
```

7.11. calculateOrderCost

Funkcja zwraca koszt zamówienia.

```
CREATE FUNCTION [dbo].[callculateOrderCost](
    @order_id SMALLINT
)
RETURNS INT
AS
BEGIN
    DECLARE @amount INT
    SET @amount = (SELECT SUM(cost) FROM dbo.order_items WHERE order_id = @order_id)
    RETURN @amount
END
GO
```

7.12. checkIfSeafoodsAreAvaliable

Funkcja sprawdza czy zamówienie może posiadać potrawy morskie. Zwraca 1 jeżeli może, 0 jeżeli nie.

```
CREATE FUNCTION [dbo].[checkIfSeafoodAreAvailable] (@order_id INT)

RETURNS BIT
AS

BEGIN

    DECLARE @created datetime
    DECLARE @isSeafoodTime BIT

    SET @created = (SELECT created_at FROM dbo.orders WHERE order_id = @order_id)

    IF DATEPART(dw, @created) = 5 OR DATEPART(dw, @created) = 6 OR DATEPART(dw, @created) = 7
        SET @isSeafoodTime = 1
    ELSE
        SET @isSeafoodTime = 0

    RETURN @isSeafoodTime
END;
GO
```

8. Widoki

8.1. companiesBalance

Wyświetla bilans wszystkich firm które są klientami restauracji.

```
CREATE VIEW [dbo].[companiesBilance]
AS
    SELECT
        companies.company_name,
        dbo.calculateCompanyBalance(companies.company_id) AS Bilans
    FROM companies
GO
```

8.2. currentMenu

Wyświetla obecnie dostępne potrawy oraz napoje.

```
CREATE VIEW [dbo].[currentMenu]
AS
    SELECT
    items.name,
    items.cost,
    items.beverage

    FROM menus
    INNER JOIN dbo.items
        ON menus.item_id = items.item_id
    WHERE  menus.endM >= GETDATE() AND GETDATE() >= menus.startM
GO
```

8.3. currentReservations

Wyświetla wszystkie aktualne rezerwacje.

```
CREATE VIEW [dbo].[currentReservations]
AS
    SELECT
    customers.first_name,
    customers.second_name,
    orders.order_id,
    reservations.table_id,
    reservations.number_of_people
    FROM dbo.orders
    JOIN dbo.reservations
        ON orders.order_id = reservations.order_id
    JOIN dbo.customers
        ON customers.customer_id = orders.customer_id
    WHERE reservations.startTime >= GETDATE()
GO
```

8.4. customersBilance

Wyświetla bilans wszystkich klientów indywidualnych rezerwacji.

```
CREATE VIEW [dbo].[customersBilance]
AS
    SELECT
        customers.customer_id,
        dbo.calculateCustomerBalance(customers.customer_id) AS Zadluzenie
    FROM dbo.customers
    LEFT JOIN dbo.employees ON dbo.customers.customer_id = dbo.employees.customer_id
    WHERE dbo.employees.company_id IS NULL
GO
```

8.5. customersExpenses

Wyświetla sumę wydatków wszystkich klientów restauracji.

```
CREATE VIEW [dbo].[customersExpenses]
AS
    SELECT
        customers.customer_id,
        dbo.calculateCustomerExpenses(customers.customer_id) AS OplaconeKwotyZamowien
    FROM dbo.customers
    LEFT JOIN dbo.employees ON dbo.customers.customer_id = dbo.employees.customer_id
    WHERE dbo.employees.company_id IS NULL
GO
```

8.6. showBusinessCustomersBills

Wyświetla bilans rachunków klientów firmowych.

```
CREATE VIEW [dbo].[showBuisnessCustomersBills]
AS
    SELECT
    companies.company_name AS Nazwa_firmy,
    customers.first_name AS Imie_Pracownika,
    customers.second_name AS Nazwisko_pracownika,
    orders.bill_id AS Numer_Rachunku,

    -- Wykorzystanie funkcji do obliczenia kwoty zamowienia
    dbo.callculateOrderCost(orders.order_id) AS Kwota_Zamowienia,

    dbo.bills.status AS StatusPlatnosci


    -- Oblicz rachunek z zamowienia i wstaw do tabeli
    FROM customers
    JOIN dbo.orders
        ON orders.customer_id = customers.customer_id
    JOIN companies
        ON companies.company_id = orders.company_id
    JOIN dbo.bills
        ON bills.bill_id = orders.bill_id

    WHERE orders.company_order = 1

GO
```

8.7. showBusinessCustomers

Wyświetla wszystkich klientów biznesowych oraz ich firmy

```
CREATE VIEW [dbo].[showBusinessCustomers]
AS
    SELECT
    customers.first_name,
    customers.second_name,
    customers.email,
    customers.phone,
    companies.company_name,
    companies.NIP,
    companies.address,
    companies.phone_number
    FROM dbo.employees
    JOIN dbo.customers
        ON employees.customer_id = customers.customer_id
    JOIN dbo.companies
        ON employees.company_id = companies.company_id

GO
```

8.8. showIndividualCustomers

Wyświetla wszystkich klientów indywidualnych.

```
CREATE VIEW [dbo].[showIndividualCustomers]
AS
    SELECT
        dbo.customers.first_name AS Imie,
        dbo.customers.second_name AS Nazwisko,
        dbo.customers.phone AS Numer_telefonu,
        dbo.customers.email AS Email
    FROM dbo.customers
    LEFT JOIN dbo.employees ON dbo.customers.customer_id = dbo.employees.customer_id
    WHERE dbo.employees.company_id IS NULL
GO
```

8.9. showIndividualCustomersBills

Wyświetla bilans wszystkich indywidualnych klientów restauracji..

```
CREATE VIEW [dbo].[showIndividualCustomersBills]
AS
    SELECT
    customers.first_name AS Imie,
    customers.second_name AS Nazwisko,
    orders.bill_id AS Numer_Rachunku,
    dbo.callculateOrderCost(orders.order_id) AS Kwota_Zamowienia,
    dbo.bills.status AS StatusPlatnosci


    FROM customers
    JOIN dbo.orders
        ON orders.customer_id = customers.customer_id


    JOIN dbo.bills
        ON bills.bill_id = orders.bill_id

    WHERE orders.company_order = 0

GO
```

8.10. showUnrealizedOrders

Wyświetla wszystkie jeszcze nie zrealizowane zamówienia.

```
CREATE VIEW [dbo].[showUnrealizedOrders]
AS
    SELECT
    customers.first_name AS Imie,
    customers.second_name AS Nazwisko,
    companies.company_name AS Nazwa_Firmy,
    customers.phone AS Numer_Telefonu,
    dbo.callculateCost(orders.order_id) AS Kwota_Zamowienia,
    orders.realization_time AS Data_realizacji


    FROM dbo.bills
    JOIN dbo.orders
        ON bills.bill_id = orders.bill_id
    JOIN dbo.customers
        ON customers.customer_id = orders.order_id
    JOIN dbo.companies
        ON companies.company_id = orders.bill_id
    WHERE dbo.orders.realization_time = NULL

GO
```

8.11. thisMonthOrders

Wyświetla wszystkie zamówienia złożone w tym miesiącu.

```
CREATE VIEW [dbo].[thisMonthOrders]
AS
    SELECT *
    FROM dbo.orders
    WHERE MONTH(created_at) = MONTH(getdate())
GO
```

8.12. unpaidBills

Wyświetla wszystkie nieuregulowane płatności.

```
CREATE VIEW [dbo].[unpaidBills]
AS
    SELECT
    customers.first_name AS Imie,
    customers.second_name AS Nazwisko,
    companies.company_name AS Nazwa_Firmy,
    customers.phone AS Numer_Telefonu,
    dbo.callculateOrderCost(orders.order_id) AS Kwota_Zamowienia,
    orders.realization_time AS Data_realizacji


    FROM dbo.bills
    JOIN dbo.orders
        ON bills.bill_id = orders.bill_id
    JOIN dbo.customers
        ON customers.customer_id = orders.order_id
    JOIN dbo.companies
        ON companies.company_id = orders.bill_id
    WHERE [status] = 0

GO
```

9. Generator danych

Dane zostały wygenerowane przy użyciu języka python. Do “połaczenia” z baza danych oraz z jej “komunikacją” wykorzystano biblioteke pymssql. Do generowania danych imitujących realne odwzorowanie danych wykorzystano bibliotekę faker.

Poniżej został przedstawiony fragment kodu funkcji main:

```python
def main():
    local_params = ["pbd-projekt.database.windows.net", "admin1234", "qwerty123!", "pdb-projekt"]
    with pymssql.connect(*local_params) as conn:  # type: pymssql.Connection
        conn.autocommit(True)

        steps = [
            czysc,
            klienci_indywidualni,
            menu,
            company,
            rachunek,
            table,
            order,
            reservation,
            takeoutorder,
            orderitems,
            employees
        ]

        with conn.cursor() as cursor:  # type: pymssql.Cursor
            for i, fn in enumerate(steps, start=1):
                print("Etap {:>2}/{}:".format(i, len(steps)), fn.__name__)
                fn(cursor, conn)
            print("Koniec")
```

Poniżej zaprezentowano dwie wybrane funkcje:

- Pierwsza funkcja generuje dane klientów indywidualnych

```python
def klienci_indywidualni(cursor, connection):
    # language=SQL Server

    for _ in range(100):
        cursor.callproc("sp_addCustomer", (
            pymssql.output(int),
            fake.first_name(),
            fake.last_name(),
            random_email(),
            fake.phone_number()
        ))
```

Kraków, 2021

	customer_id	first_name	second_name	email	phone
1	1	Julita	Dorynek	sadowyprzemyslaw@o2.pl	32 679 20 45
2	2	Przemyslaw	Mical	ewasuchta@grupa.com	737 968 637
3	3	Aleksander	Klinkosz	ymik@fundacja.com	+48 32 133 34 56
4	4	Karol	Grotek	mwujek@yahoo.com	512 934 269
5	5	Kamila	Miros	marciniszynkonrad@yahoo.com	738 475 504
6	6	Tadeusz	Pachla	natan73@hotmail.com	518 135 225
7	7	Anastazja	Sierota	rmuniak@gmail.com	+48 785 741 244
8	8	Sylwia	Wandas	witoldrzaca@interia.pl	+48 517 509 706
9	9	Anna Maria	Parafiniuk	gsiwka@yahoo.com	+48 32 577 60 75
10	10	Radoslaw	Bajko	kfras@stowarzyszenie.com	507 704 790
11	11	Antoni	Wawak	barejagabriel@grupa.com	+48 32 774 31 22
12	12	Damian	Dwomiczak	clapamateusz@stowarzyszenie.pl	+48 737 864 748
13	13	Nikodem	Wyderka	tguzdek@lesik.pl	+48 32 614 41 01
14	14	Marianna	Jaszczur	kubekalan@kunysz-matacz.org	+48 578 445 697
15	15	Pawel	Starzak	lpleskot@o2.pl	+48 32 263 12 08
16	16	Marek	Masiak	ksawery22@truchel-jaskowiec.pl	+48 739 031 579
17	17	Tadeusz	Garczyk	danielukjedrzej@hotmail.com	609 526 320
18	18	Oliwier	Kusik	rbochnia@stowarzyszenie.org	+48 609 389 067
19	19	Leon	Golda	anielaarmatys@spoldzielnia.pl	+48 575 735 308
20	20	Róza	Kenig	albert25@fpuh.com	880 703 594
21	21	Marianna	Powalka	engelangelika@yahoo.com	+48 538 526 138
22	22	Lidia	Kubek	koziejanikodem@gabinety.pl	737 953 756
23	23	Julita	Kucharz	szymonwojtczuk@yahoo.com	+48 720 360 870
24	24	Oliwier	Konsek	sara75@onet.pl	+48 530 491 150
25	25	Juliusz	Mucka	pasciakangelika@kurpisz-matu...	+48 32 870 19 21
26	26	Stefan	Machowicz	filipperczak@yahoo.com	500 423 727
27	27	Maksymili...	Beska	norbert22@spoldzielnia.com	+48 729 154 250
28	28	Agnieszka	Nedzi	iwodyszkiewicz@interia.pl	518 705 374
29	29	Marianna	Jaszczyszyn	vpietrusiewicz@yahoo.com	+48 22 742 42 20
30	30	Marianna	Pancerz	przemyslawsteuer@mlynczyk.org	+48 504 312 722
31	31	Maurycy	Miodek	nataszamieszala@hotmail.com	+48 888 537 049
32	32	Damian	Cyron	liwia35@yahoo.com	+48 888 374 569
33	33	Ada	Dzwonnik	ryszardgoss@spoldzielnia.org	600 880 771

- Natomiast druga generuje dane dla rezerwacji

Kraków, 2021

```python
def reservation(cursor, connection):
    # language=SQL Server
    cursor.execute("SELECT table_id, seats FROM tables")
    all_tables = cursor.fetchall()
    cursor.execute("SELECT order_id, realization_time, takeout  FROM orders")
    all_orders = cursor.fetchall()

    for order_id, realization_time, takeout in all_orders:

        if takeout != 1:
            start_date = realization_time
            end_date = start_date + timedelta.Timedelta(hours=fake.random_int(min=1, max=2))
            Tablenumber = fake.random_sample(elements=all_tables, length=fake.random_int(min=1, max=len(all_tables)))

            i = 0
            while i < len(Tablenumber):
                id = int(Tablenumber[i][0])
                minseat = int((Tablenumber[i][1])/2)
                maxseat = int(Tablenumber[i][1])

                cursor.callproc("addReservation", (
                    pymssql.output(int),
                    id,
                    start_date,
                    end_date,
                    fake.random_int(min=minseat, max=maxseat),
                    order_id
                ))
                i += 1
```

	reservation_id	table_id	startTime	endTime	number_of_people	order_id
1	1	9	2021-07-09 11:16:00	2021-07-09 12:16:00	5	1
2	2	4	2021-07-09 11:16:00	2021-07-09 12:16:00	5	1
3	3	5	2021-07-09 11:16:00	2021-07-09 12:16:00	7	1
4	4	0	2021-07-09 11:16:00	2021-07-09 12:16:00	4	1
5	5	2	2021-07-09 11:16:00	2021-07-09 12:16:00	5	1
6	6	5	2020-10-12 04:11:00	2020-10-12 05:11:00	5	2
7	7	0	2020-10-12 04:11:00	2020-10-12 05:11:00	5	2
8	8	2	2020-10-12 04:11:00	2020-10-12 05:11:00	4	2
9	9	4	2020-10-12 04:11:00	2020-10-12 05:11:00	6	2
10	10	3	2020-10-12 04:11:00	2020-10-12 05:11:00	3	2
11	11	9	2020-10-12 04:11:00	2020-10-12 05:11:00	3	2
12	12	3	2021-06-25 07:57:00	2021-06-25 09:57:00	7	3
13	13	0	2021-06-25 07:57:00	2021-06-25 09:57:00	3	3
14	14	6	2021-06-25 07:57:00	2021-06-25 09:57:00	6	3
15	15	7	2021-06-25 07:57:00	2021-06-25 09:57:00	4	3
16	16	4	2021-06-25 07:57:00	2021-06-25 09:57:00	8	3
17	17	1	2021-06-25 07:57:00	2021-06-25 09:57:00	5	3
18	18	2	2021-06-25 07:57:00	2021-06-25 09:57:00	6	3
19	19	3	2021-03-18 16:06:00	2021-03-18 18:06:00	3	4
20	20	6	2021-03-18 16:06:00	2021-03-18 18:06:00	9	4
21	21	8	2021-03-18 16:06:00	2021-03-18 18:06:00	7	4
22	22	0	2021-03-18 16:06:00	2021-03-18 18:06:00	2	4
23	23	2	2021-03-18 16:06:00	2021-03-18 18:06:00	6	4
24	24	7	2021-03-18 16:06:00	2021-03-18 18:06:00	6	4
25	25	9	2021-03-18 16:06:00	2021-03-18 18:06:00	7	4
26	26	1	2021-03-18 16:06:00	2021-03-18 18:06:00	5	4
27	27	4	2021-03-18 16:06:00	2021-03-18 18:06:00	9	4
28	28	5	2021-03-18 16:06:00	2021-03-18 18:06:00	5	4
29	29	9	2021-10-01 12:56:00	2021-10-01 14:56:00	4	5
30	30	3	2021-10-01 12:56:00	2021-10-01 14:56:00	3	5
31	31	6	2021-10-01 12:56:00	2021-10-01 14:56:00	10	5

Kraków, 2021

10. Implementacja systemu rabatów i wyliczanie ostatecznej kwoty do zapłaty

Rabat permanentny - przyznawany jest po realizacji określonej ilości zamówień.

Tworząc zamówienie sprawdzamy czy klient spełnia warunki, jeśli tak to do bills.discount - rachunku podpiętego do tworzonego zamówienia przypisujemy wartość rabatu.

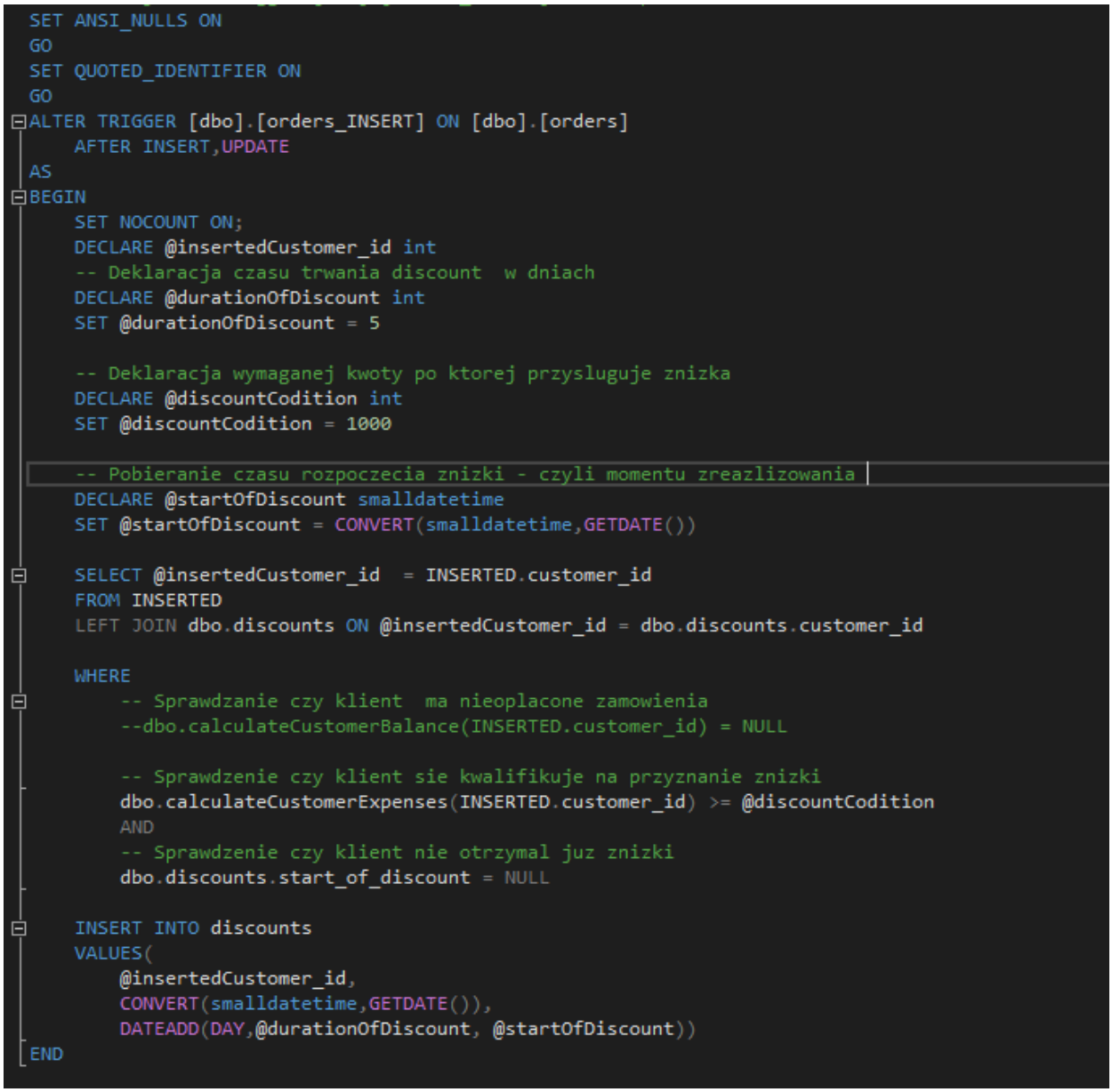
```
IF ([dbo].[calculateAmountOfIndividualCustomerOrders](@customer_id) >= @amountOfOrdersForDiscount)

    INSERT INTO bills
        (bill_id,
        discount,
        [status],
        invoice)
    VALUES
        (@bill_id
        ,@repetitiveDiscount
        ,@status
        ,@invoice)
```

Rabat jednorazowy - przyznawany po realizacji zamówień za określoną kwotę.

Po utworzeniu albo modyfikacji zamówienia uruchamia się trigger orders_INSERT. Odpowiedzialny jest on za sprawdzenie czy klient spełnia warunki, jeśli tak to do tabeli discounts dodajemy nowy rabat.

Ostateczna kwota - Kwota jaką klient ma zapłacić po uwzględnieniu wszystkich rabatów.

Po zatwierdzeniu realizacji zamówienia przez pracownika, wyliczany jest finalny rachunek wg formuły sprawdzającej wysokość rabatu permanentnego z tabeli bill.discount, oraz tego czy klientowi przysługuje rabat jednorazowy na podstawie tabeli discounts(Czy zamówienie zostało zrealizowane w okresie między discounts.start_of_discount a discounts.end_of_discount)



```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[orders_INSERT] ON [dbo].[orders]
    AFTER INSERT,UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @insertedCustomer_id int
    -- Deklaracja czasu trwania discount  w dniach
    DECLARE @durationOfDiscount int
    SET @durationOfDiscount = 5

    -- Deklaracja wymaganej kwoty po ktorej przysluguje znizka
    DECLARE @discountCodition int
    SET @discountCodition = 1000

    -- Pobieranie czasu rozpoczecia znizki - czyli momentu zreazlizowania
    DECLARE @startOfDiscount smalldatetime
    SET @startOfDiscount = CONVERT(smalldatetime,GETDATE())

    SELECT @insertedCustomer_id  = INSERTED.customer_id
    FROM INSERTED
    LEFT JOIN dbo.discounts ON @insertedCustomer_id = dbo.discounts.customer_id

    WHERE
        -- Sprawdzanie czy klient  ma nieoplacone zamowienia
        --dbo.calculateCustomerBalance(INSERTED.customer_id) = NULL

        -- Sprawdzenie czy klient sie kwalifikuje na przyznanie znizki
        dbo.calculateCustomerExpenses(INSERTED.customer_id) >= @discountCodition
        AND
        -- Sprawdzenie czy klient nie otrzymal juz znizki
        dbo.discounts.start_of_discount = NULL

    INSERT INTO discounts
    VALUES(
        @insertedCustomer_id,
        CONVERT(smalldatetime,GETDATE()),
        DATEADD(DAY,@durationOfDiscount, @startOfDiscount))
END
```

Implementacja order_INSERT - tworzenie rabatu jednorazowego

Kraków, 2021

11. Uprawnienia użytkowników

- Klient
    - addReservation,
    - addOrder,
    - addCustomer,
    - addTakeoutOrder,
    - cancelReservation,
    - checkTableAvalibility,
    - currentMenu

- Pracownik (posiada również uprawnienia klienta)
    - showUnrealizedOrders,
    - unpaidBills,
    - currentReservations,
    - customerExpenses,
    - thisMonthOrders

- Menedżer (posiada również uprawnienia pracownika)
    - addMenu,
    - updateTable,
    - editItem,
    - addEmployee,
    - companiesBalance,
    - customersBilance,
    - showBuisnessCustomers,
    - showIndividualCustomers

Administrator posiada całkowity dostęp do wszystkich elementów bazy danych.

Kraków, 2021