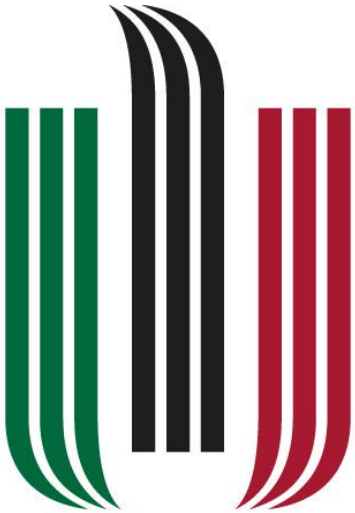


AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

PROJEKT Z PRZEDMIOTU PODSTAWY BAZ DANYCH

Krzysztof Solecki
Rafał Tekielski
Miłosz Dobosz

Rok akademicki: 2021/22
Kierunek studiów: Informatyka
3 semestr studiów stacjonarnych

Funkcje realizowane przez system:

Aktorzy:

1. **Administrator systemu -** pełny dostęp do bazy danych i jej funkcjonalności
2. **Menedżer restauracji -** dostęp do danych dot. jego restauracji oraz zarządzanie nimi
3. **Pracownik restauracji -** dostęp do funkcji niezbędnych do obsługi klientów
4. **Klient indywidualny -** dostęp do danych określających jego zamówienia,rezerwacje i rabaty, możliwość składania rezerwacji i zamówień.
5. **Firma -** dostęp do danych określających jej zamówienia ,rezerwacje i rabaty oraz możliwość składania zamówień i rezerwacji, a także zarządzania listą pracowników firmy.

Funkcje użytkowników:

1. Administrator systemu:

- Obsługa błędów
- Archiwizowanie danych
- Nadawanie uprawnień użytkownikom

2. Menedżer restauracji:

- Wybieranie menu z przygotowanej listy dań raz na 2 tygodnie z co najmniej dziennym wyprzedzeniem
- Dodawanie, usuwanie i aktualizowanie rabatów
- Rejestracja pracownika restauracji
- Dodawanie i usuwanie stolików
- Generowanie raportów i statystyk miesięcznych i tygodniowych dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia (dochód, ilość sprzedanych dań, ilość klientów, ilość rezerwacji)
- Wgląd do aktualnych rabatów, rezerwacji i zamówień klienta
- Wgląd do menu restauracji
- Wgląd do stanu dań w menu
- Aktualizacja danych dotyczących ilości dań dostępnych w restauracji

3. Pracownik restauracji:

- Anulowanie zamówienia
- Przyjęcie zamówienia i jego potwierdzenie
- Wgląd do aktualnych rabatów, rezerwacji i zamówień klienta
- Wygenerowanie listy osób na rezerwację imienną dla firmy
- Wgląd do menu restauracji
- Wgląd do stanu dań w menu

4. Klient indywidualny:

- Rejestracja klienta
- Złożenie rezerwacji
- Anulowanie rezerwacji
- Generowanie raportów dotyczących rabatów, rezerwacji i zamówień klienta
- Generowanie paragonów
- Wgląd do menu restauracji

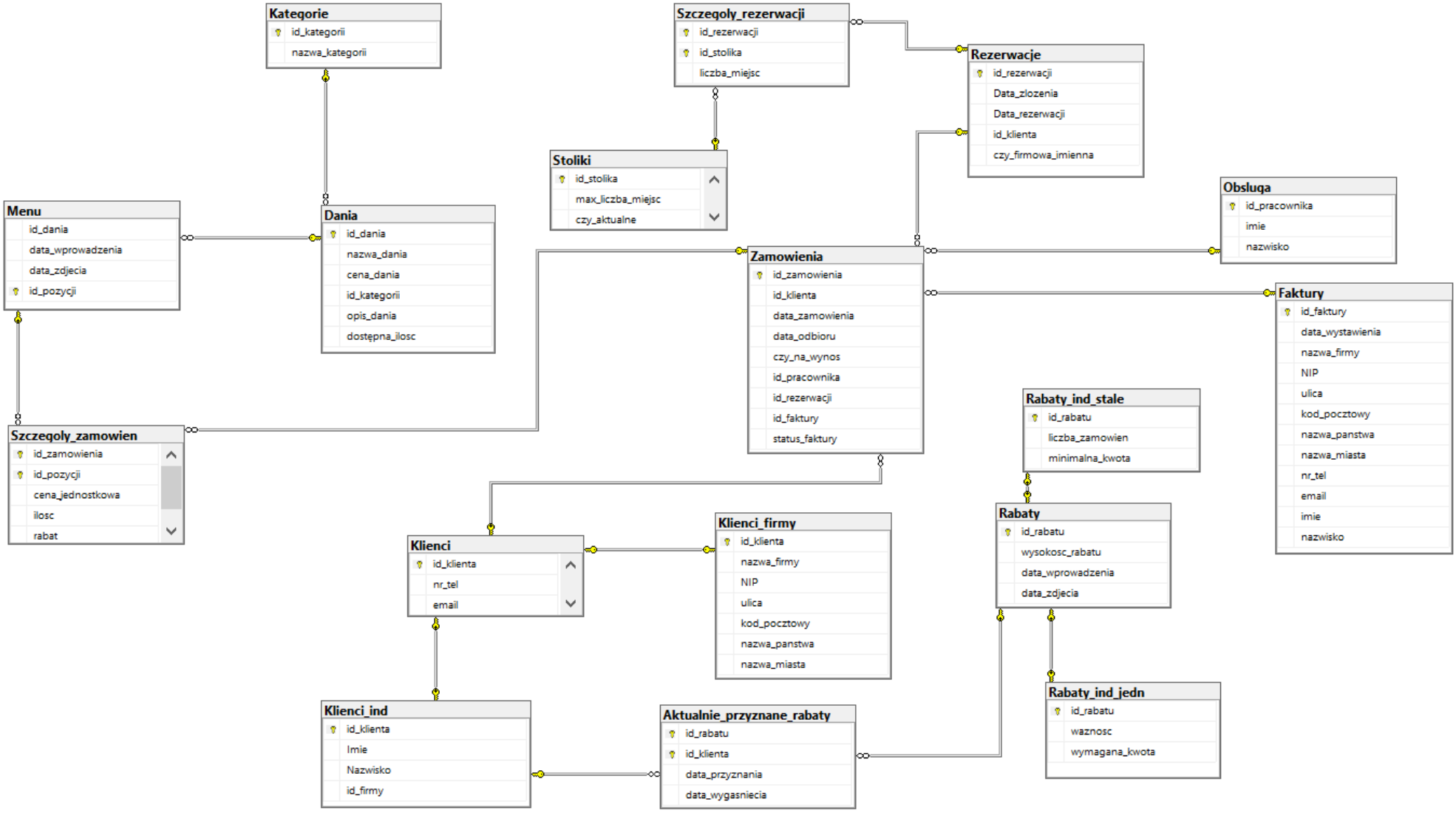
5. Firma:

- Rejestracja klienta
- Rejestracja informacji o pracowniku firmy
- Lista zarejestrowanych pracowników firmy
- Złożenie rezerwacji
- Anulowanie rezerwacji
- Generowanie raportów dotyczących rabatów, rezerwacji i zamówień klienta
- Wygenerowanie faktury za miesiąc usług
- Wygenerowanie faktury za pojedyncze zamówienie
- Wygenerowanie listy osób na rezerwację imienną dla firmy
- Wgląd do menu restauracji

Funkcje systemu:

- Automatyczne aktualizowanie dostępności poszczególnych dań
- Obliczanie rabatów dla danego klienta
- Automatyczne zarządzanie obsługą rabatów
- Kontrola nad rezerwacją przez formularz internetowy (odpowiednia minimalna wartość zamówienia oraz liczba zamówień złożonych wcześniej)
- Zarządzanie dostępnością stolików podczas rezerwacji
- Obliczanie wartości zamówienia
- Obliczanie liczby wolnych miejsc w lokalu
- Kontrola zasad związanych ze szczegółowymi wymaganiami klienta (rabaty, menu, owoce morza itp.)
- Automatyczne generowanie raportów dla firm i klientów indywidualnych na żądanie

Schemat bazy danych:



Implementacja i warunki integralności:

1. **Tabela Aktualnie_przyznane_rabaty** - tabela zawierająca wszystkie rabaty przyznane klientom, które aktualnie obowiązują

    - Klucz główny - para ID_rabatu (int) ,ID_klienta (int)
    - ID_rabatu to klucz obcy określający typ przyznanego rabatu
    - ID_klienta jest kluczem obcym oznaczającym klienta
    - Data_przyznania (date) określa dzień, w którym rabat został nadany klientowi
    - Data_wygaśnięcia (date) określa dzień, w którym rabat traci swoją ważność, o ile taki dzień znamy

Data przyznania jest chwilą obecną lub chwilą z przeszłości.
Data przyznania ma domyślną wartość getDate().
Data wygaśnięcia jest późniejsza niż data przyznania lub data wygaśnięcia może być nieznana.

```
CREATE TABLE [dbo].[Aktualnie_przyznane_rabaty](
    [id_rabatu] [INT] NOT NULL,
    [id_klienta] [INT] NOT NULL,
    [data_przyznania] [DATE] NOT NULL,
    [data_wygasniecia] [DATE] NULL,
 CONSTRAINT [PK_Aktualnie_przyznane_rabaty] PRIMARY KEY CLUSTERED
(
    [id_rabatu] ASC,
    [id_klienta] 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].[Aktualnie_przyznane_rabaty]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_przyznane_rabaty_Klienci_ind] FOREIGN KEY([id_klienta])
REFERENCES [dbo].[Klienci_ind] ([id_klienta])
GO

ALTER TABLE [dbo].[Aktualnie_przyznane_rabaty] CHECK CONSTRAINT
[FK_Aktualnie_przyznane_rabaty_Klienci_ind]
GO
```

```
ALTER TABLE [dbo].[Aktualnie_przyznane_rabaty]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_przyznane_rabaty_Rabaty] FOREIGN KEY([id_rabatu])
REFERENCES [dbo].[Rabaty] ([id_rabatu])
GO

ALTER TABLE [dbo].[Aktualnie_przyznane_rabaty] CHECK CONSTRAINT
[FK_Aktualnie_przyznane_rabaty_Rabaty]
GO


ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] ADD CONSTRAINT
[DF_Aktualnie_Przyznane_Rabaty_Data_przyznania] DEFAULT (GETDATE()) FOR
[Data_przyznania]
GO

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] WITH CHECK ADD CONSTRAINT
[CK_Aktualnie_Przyznane_Rabaty_Daty] CHECK (([Data_przyznania]<=GETDATE() AND
([Data_wygasniecia]
IS NULL OR [Data_wygasniecia]>=[Data_przyznania])))
GO
ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] CHECK CONSTRAINT
[CK_Aktualnie_Przyznane_Rabaty_Daty]
GO
```

2. **Tabela Dania** - tabela zawierająca wszystkie dania serwowane przez restaurację.

- **ID_dania (int)** to klucz główny, jednoznacznie identyfikujący każde danie
- **nazwa_dania (varchar)** określa nazwę każdego dania
- **cena_dania (money)** określa cenę danego dania
- **id_kategori (int)** określa identyfikator kategorii, do której należy danie
- **opis dania (varchar)** opcjonalne pole, zawierające opis dania
- **dostepna_ilosc (int)** określa ile razy restauracja jest w stanie zaserwować dane danie danego dnia

Cena dania musi być większa od zera.
Długość nazwy dania musi być większa niż 2.

```
CREATE TABLE [dbo].[Dania](
    [id_dania] [INT] NOT NULL,
    [nazwa_dania] [VARCHAR](50) NOT NULL,
    [cena_dania] [money] NOT NULL,
    [id_kateogorii] [INT] NOT NULL,
    [opis_dania] [VARCHAR](255) NULL,
    [dostępna_ilosc] [INT] NOT NULL,
```

```
CONSTRAINT [PK_Dania] PRIMARY KEY CLUSTERED
(
    [id_dania] 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].[Dania]  WITH CHECK ADD  CONSTRAINT [FK_Dania_Kategorie] FOREIGN
KEY([id_kateogorii])
REFERENCES [dbo].[Kategorie] ([id_kategorii])
GO

ALTER TABLE [dbo].[Dania] CHECK CONSTRAINT [FK_Dania_Kategorie]
GO

ALTER TABLE [dbo].[Dania] WITH CHECK ADD CONSTRAINT [CK_Dania_Cena] CHECK(([Cena_dania]>(0)))
GO
ALTER TABLE [dbo].[Dania] CHECK CONSTRAINT [CK_Dania_Cena]
GO
ALTER TABLE [dbo].[Dania] WITH CHECK ADD CONSTRAINT [CK_Dania_Nazwa_Dania_Min] CHECK
((len([Nazwa_dania])>(2)))
GO
ALTER TABLE [dbo].[Dania] CHECK CONSTRAINT [CK_Dania_Nazwa_Dania_Min]
GO
```

3. **Tabela Kategorie** - tabela zawierające kategorie, do których możemy przypisać każde danie.

- **ID_Kategorii (int)** to klucz główny, jednoznacznie identyfikujący każdą kategorię
- **Nazwa_Kategorii (varchar)** określa nazwę danej kategorii

Nazwa kategorii jest wartością unikalną oraz musi mieć długość co najmniej 3.

```
CREATE TABLE [dbo].[Kategorie](
    [id_kategorii] [INT] NOT NULL,
    [nazwa_kategorii] [VARCHAR](50) NOT NULL,
    CONSTRAINT [PK_Kategorie] PRIMARY KEY CLUSTERED
(
    [id_kategorii] 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].[Kategorie] WITH CHECK ADD CONSTRAINT [CK_Kategorie_Nazwa] CHECK
((len([Nazwa_kategorii])>=(3)))
GO
ALTER TABLE [dbo].[Kategorie] CHECK CONSTRAINT [CK_Kategorie_Nazwa]
GO
```

4. **Tabela Klienci -** reprezentuje zbiór wszystkich klientów w bazie danych. Zawiera informację o:

   - **ID_klienta (int),** które stanowi klucz główny jednoznacznie identyfikujący klienta
   - **nr_tel (varchar),** numer telefonu klienta
   - **email (varchar)**, będący adresem e-mail klienta

   Email postaci: ciąg_znaków@ciąg_znaków.ciąg_zanków, jest to wartość unikalna.
   Telefon składa się z 9 cyfr.

```
CREATE TABLE [dbo].[Klienci](
    [id_klienta] [INT] NOT NULL,
    [nr_tel] [VARCHAR](9) NOT NULL,
    [email] [VARCHAR](50) NOT NULL,
    CONSTRAINT [PK_Klienci_1] PRIMARY KEY CLUSTERED
(
    [id_klienta] 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].[Klienci] WITH CHECK ADD CONSTRAINT [CK_Klienci_email]
CHECK (([email] like '%@%.%'))
GO
ALTER TABLE [dbo].[Klienci] CHECK CONSTRAINT [CK_Klienci_email]
GO
ALTER TABLE [dbo].[Klienci] WITH CHECK ADD CONSTRAINT [CK_Klienci_Telefon] CHECK (([nr_tel]
like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO
ALTER TABLE [dbo].[Klienci] CHECK CONSTRAINT [CK_Klienci_Telefon]
GO
```

5. **Tabela Klienci_Firmy -** reprezentuje klientów firmowych w bazie danych. Zawiera informację o:

   - **ID_klienta (int),** stanowiące klucz główny
   - **Nazwa_firmy (varchar),** będące nazwą firmy
   - **Numer NIP (varchar)**
   - Adres, na który składa się**: ulica (varchar), kod_pocztowy (varchar), nazwa_miasta (int), nazwa_panstwa (int)**

Numer NIP jest unikalny.
Kod pocztowy jest postaci: [cyfra][cyfra]-[cyfra][cyfra][cyfra] lub [cyfra][cyfra][cyfra][cyfra][cyfra].
Numer NIP składa się z 10 cyfr.
Ulica kończy się cyfrą, co oznacza numer domu lub małą literą (w sytuacji gdy numer domu to np. 47a)

```
CREATE TABLE [dbo].[Klienci_firmy](
    [id_klienta] [INT] NOT NULL,
    [nazwa_firmy] [VARCHAR](50) NOT NULL,
    [NIP] [VARCHAR](10) NOT NULL,
    [ulica] [VARCHAR](50) NOT NULL,
    [kod_pocztowy] [VARCHAR](50) NOT NULL,
    [nazwa_panstwa] [VARCHAR](50) NULL,
    [nazwa_miasta] [VARCHAR](50) NOT NULL,
 CONSTRAINT [PK_Klienci_firmy] PRIMARY KEY CLUSTERED
(
    [id_klienta] 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].[Klienci_firmy]  WITH CHECK ADD  CONSTRAINT [FK_Klienci_firmy_Klienci1]
FOREIGN KEY([id_klienta])
```

```
REFERENCES [dbo].[Klienci] ([id_klienta])
GO
ALTER TABLE [dbo].[Klienci_firmy] CHECK CONSTRAINT [FK_Klienci_firmy_Klienci1]
GO
ALTER TABLE [dbo].[Klienci_Biz] WITH CHECK ADD CONSTRAINT [CK_Klienci_Biz_Kod_pocztowy]
CHECK (([Kod_pocztowy] like '[0-9][0-9][0-9][0-9][0-9]' OR [Kod_pocztowy] like
'[0-9][0-9]-[0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Klienci_firmy] CHECK CONSTRAINT [CK_Klienci_firmy_Kod_pocztowy]
GO
ALTER TABLE [dbo].[Klienci_firmy] WITH CHECK ADD CONSTRAINT [CK_Klienci_firmy_NIP] CHECK
(([NIP] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO
ALTER TABLE [dbo].[Klienci_firmy] CHECK CONSTRAINT [CK_Klienci_firmy_NIP]
GO
ALTER TABLE [dbo].[Klienci_firmy] WITH CHECK ADD CONSTRAINT [CK_Klienci_firmy_Ulica] CHECK
(([Ulica]
like '%[0-9][a-z]' OR [Ulica] like '%[0-9]'))
GO
ALTER TABLE [dbo].[Klienci_firmy] CHECK CONSTRAINT [CK_Klienci_firmy_Ulica]
GO
```

6. **Tabela Klienci_ind -** reprezentuje klientów indywidualnych w bazie danych. Zawiera informację o:

- **ID_klienta (int),** stanowiące klucz główny
- **imie (varchar), czyli imię klienta**
- **nazwisko (varchar), czyli nazwisko konkretnego klienta**

W imieniu i nazwisku nie mogą występować cyfry.

```
CREATE TABLE [dbo].[Klienci_ind](
    [id_klienta] [INT] NOT NULL,
    [Imie] [VARCHAR](50) NOT NULL,
    [Nazwisko] [VARCHAR](50) NOT NULL,
    [id_firmy] [INT] NULL,

 CONSTRAINT [PK_Klienci_ind] PRIMARY KEY CLUSTERED
(
    [id_klienta] 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].[Klienci_ind]  WITH CHECK ADD  CONSTRAINT [FK_Klienci_ind_Klienci1] FOREIGN
KEY([id_klienta])
```

```
REFERENCES [dbo].[Klienci] ([id_klienta])
GO

ALTER TABLE [dbo].[Klienci_ind] CHECK CONSTRAINT [FK_Klienci_ind_Klienci1]
GO


    ALTER TABLE [dbo].[Klienci_Ind] WITH CHECK ADD CONSTRAINT [CK_Klienci_Ind_Imie]
CHECK ((NOT [Imię] like '%[0-9]%'))
GO
ALTER TABLE [dbo].[Klienci_Ind] CHECK CONSTRAINT [CK_Klienci_Ind_Imie]
GO
ALTER TABLE [dbo].[Klienci_Ind] WITH CHECK ADD CONSTRAINT [CK_Klienci_Ind_Nazw]
CHECK ((NOT [Nazwisko] like '%[0-9]%'))
GO
ALTER TABLE [dbo].[Klienci_Ind] CHECK CONSTRAINT [CK_Klienci_Ind_Nazw]
GO
```

7. **Menu -** - tabela zawierająca wykaz dań, które znajdują się w menu aktualnie oraz tych obecnych w menu w przeszłości

- **ID_pozycji (int)** to klucz główny jednoznacznie określający daną pozycję, która pojawiła się w menu
- **ID_Dania (int)** określa danie, które znajduje się pod wskazaną pozycją menu
- **Data_wprowadzenia (date)** określa dzień, w którym dana pozycja została wprowadzona do menu
- **Data_zdjecia (date)** określa dzień, w którym dana pozycja została zdjęta z menu

Data wprowadzenia domyślnie ma wartość getDate()

```
CREATE TABLE [dbo].[Menu](
    [id_dania] [INT] NOT NULL,
    [data_wprowadzenia] [DATE] NOT NULL,
    [data_zdjecia] [DATE] NULL,
    [id_pozycji] [INT] NOT NULL,
 CONSTRAINT [PK_Menu] PRIMARY KEY CLUSTERED
(
    [id_pozycji] 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].[Menu]  WITH CHECK ADD  CONSTRAINT [FK_Menu_Dania] FOREIGN KEY([id_dania])
```

```
REFERENCES [dbo].[Dania] ([id_dania])
GO
ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Dania]
GO

ALTER TABLE [dbo].[Menu] ADD CONSTRAINT [DF_Menu_Data_wprowadzenia] DEFAULT (GETDATE()) FOR
[Data_wprowadzenia]
GO
```

8. **Obsługa -** tabela zawierająca dane pracowników pracujących restauracji

    - **ID_Pracownika (int)** to klucz główny jednoznacznie identyfikujący każdego pracownika restauracji
    - **Imie (varchar)** określa imię pracownika
    - **Nazwisko (varchar)** określa nazwisko pracownika

Imię i nazwisko nie może zawierać cyfr.

```
CREATE TABLE [dbo].[Obsluga](
    [id_pracownika] [INT] NOT NULL,
    [imie] [VARCHAR](50) NOT NULL,
    [nazwisko] [VARCHAR](50) NOT NULL,

 CONSTRAINT [PK_Obsluga] PRIMARY KEY CLUSTERED
```

```
(
    [id_pracownika] 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].[Obsluga] WITH CHECK ADD CONSTRAINT [CK_Obsluga_Imie] CHECK ((NOT [Imię]
LIKE
'%[0-9]%'))
GO
ALTER TABLE [dbo].[Obsluga] CHECK CONSTRAINT [CK_Obsluga_Imie]
GO
ALTER TABLE [dbo].[Obsluga] WITH CHECK ADD CONSTRAINT [CK_Obsluga_Nazwisko] CHECK ((NOT
[Nazwisko] LIKE '%[0-9]%'))
GO
ALTER TABLE [dbo].[Obsluga] CHECK CONSTRAINT [CK_Obsluga_Nazwisko]
GO
```

9. **Tabela Rabaty -** zbiera wszystkie dostępne typy rabatu wraz z warunkami, które dotyczą wszystkich typów rabatów

    - **ID_rabatu (int)** to klucz główny

- **Wysokość Rabatu (float)** to podstawowy rabat, który jest przyznawany lub naliczany przy spełnieniu warunków
- Każdy rabat posiada informację o dacie wprowadzenia **Data_wprowadzenia (date)** oraz dacie anulowania zasad, jeśli ten rabat już nie obowiązuje **Data_zdjecia (date)**. Daty te pozwalają na sprawdzenie warunków w przypadku rabatów ciągłych, jeśli w okresie ciągłości zasady naliczania rabatu uległy zmianie

Data wprowadzenia ma domyślną wartość getDate().
Jednostkowa wysokość rabatu jest liczbą rzeczywistą z przedziału [0,1].

```
CREATE TABLE [dbo].[Rabaty](
    [id_rabatu] [INT] NOT NULL,
    [wysokosc_rabatu] [money] NOT NULL,
    [data_wprowadzenia] [DATE] NOT NULL,
    [data_zdjecia] [DATE] NOT NULL,
 CONSTRAINT [PK_Rabaty] PRIMARY KEY CLUSTERED
(
    [id_rabatu] 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].[Rabaty] ADD CONSTRAINT [DF_Rabaty_Data_wprowadzenia]
```

```
DEFAULT (GETDATE()) FOR
[Data_wprowadzenia]
GO

    ALTER TABLE [dbo].[Rabaty] WITH CHECK ADD CONSTRAINT [CK_Rabaty_Wys_Jedn] CHECK
(([Wysokosc_jedn]>=(0) AND [Wysokosc_jedn]<=(1)))
GO
ALTER TABLE [dbo].[Rabaty] CHECK CONSTRAINT [CK_Rabaty_Wys_Jedn]
GO
```

10. **Tabela Rabaty_ind_jedn -** tabela zawierająca dodatkowe warunki dotyczące rabatów dla klientów indywidualnych, które są jednorazowe

    - **ID_rabatu (int)** to klucz główny i określa dodatkowo numer po którym możemy rozpoznać dany rabat
    - **Waznosc (int)** to ustalona przez zasady liczba dni, przez które dany rabat obowiązuje od chwili jego przyznania
    - **Wymagana_kwota (money)** określa minimalną łączną kwotę zamówień dzięki którym przyznana zostaje jednorazowa zniżka

    Podawana w dniach ważność rabatu musi być dodatnia.
    Wymagana kwota musi być dodatnia.

```
CREATE TABLE [dbo].[Rabaty_ind_jedn](
```

```
    [id_rabatu] [INT] NOT NULL,
    [waznosc] [INT] NOT NULL,
        [wymagana_kwota] [MONEY] NOT NULL,
 CONSTRAINT [PK_Rabaty_ind_jedn] PRIMARY KEY CLUSTERED
(
    [id_rabatu] 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].[Rabaty_ind_jedn]  WITH CHECK ADD  CONSTRAINT [FK_Rabaty_ind_jedn_Rabaty]
FOREIGN KEY([id_rabatu])
REFERENCES [dbo].[Rabaty] ([id_rabatu])
GO
ALTER TABLE [dbo].[Rabaty_ind_jedn] CHECK CONSTRAINT [FK_Rabaty_ind_jedn_Rabaty]
GO


ALTER TABLE [dbo].[Rabaty_Ind_Jedn] WITH CHECK ADD CONSTRAINT
[CK_Rabaty_Ind_Jedn_Waznosc] CHECK (([Waznosc]>(0)))
GO
ALTER TABLE [dbo].[Rabaty_Ind_Jedn] CHECK CONSTRAINT [CK_Rabaty_Ind_Jedn_Waznosc]
GO
```

```
    ALTER TABLE [dbo].[Rabaty_Ind_Jedn] WITH CHECK ADD CONSTRAINT
[CK_Rabaty_Ind_Jedn_Waznosc] CHECK (([Waznosc]>(0)))
GO
ALTER TABLE [dbo].[Rabaty_Ind_Jedn] CHECK CONSTRAINT [CK_Rabaty_Ind_Jedn_Waznosc]
GO
```

11. **Tabela Rabaty_Ind_Stale** - tabela zawierająca dodatkowe warunki, które klient indywidualny musi spełnić, aby mu przyznano rabat stały.

    - **ID_rabatu (int)** to klucz główny i określa dodatkowo numer po którym możemy rozpoznać dany rabat
    - **Liczba_zamowien (int)** określa ilość złożonych zamówień, po której przekroczeniu klientowi jest przyznawany ten typ rabatu
    - **Minimalna_kwota (money)** określa minimalną kwotę każdego zamówienia wliczanego do rabatów stałych

    Liczba zamówień oraz minimalna kwota musi być liczbą dodatnią.

```
CREATE TABLE [dbo].[Rabaty_ind_stale](
    [id_rabatu] [INT] NOT NULL,
    [liczba_zamowien] [INT] NOT NULL,
```

```
[minimalna_kwota] [MONEY] NOT NULL,
 CONSTRAINT [PK_Rabaty_ind_stale] PRIMARY KEY CLUSTERED
(
    [id_rabatu] 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].[Rabaty_ind_stale]  WITH CHECK ADD  CONSTRAINT [FK_Rabaty_ind_stale_Rabaty]
FOREIGN KEY([id_rabatu])
REFERENCES [dbo].[Rabaty] ([id_rabatu])
GO
ALTER TABLE [dbo].[Rabaty_ind_stale] CHECK CONSTRAINT [FK_Rabaty_ind_stale_Rabaty]
GO

ALTER TABLE [dbo].[Rabaty_Ind_Stale] WITH CHECK ADD CONSTRAINT [CK_Rabaty_Ind_Stale_L_Zam]
CHECK (([Liczba_zamowien]>(0)))
GO
ALTER TABLE [dbo].[Rabaty_Ind_Stale] CHECK CONSTRAINT [CK_Rabaty_Ind_Stale_L_Zam]
GO

ALTER TABLE [dbo].[Rabaty_Ind_Stale] WITH CHECK ADD CONSTRAINT
[CK_Rabaty_Ind_Stale_min_kwota] CHECK (([Minimalna_kwota]>(0)))
```

```
GO
ALTER TABLE [dbo].[Rabaty_Ind_Stale] CHECK CONSTRAINT [CK_Rabaty_Ind_Stale_min_kwota]
GO
```

12. **Tabela Rezerwacje** - tabela reprezentuje wszystkie złożone rezerwacje. Posiada informację o:

    - **ID_rezerwacji (int),** które jest kluczem głównym
    - Dacie złożenia rezerwacji przez klienta **Data_zlozenia (date)** oraz dacie, na kiedy dana rezerwacja została złożona **Data_rezerwacji (date)**
    - **id_klienta (int) -** określa klienta składającego zamówienie
    - **czy_firmowa_imienna (varchar)** - określa czy klient składa rezerwację imienną dla firmy ,w której pracuje.

Data złożenia ma domyślnie wartość getDate().
Pole czy_firmowa_imienna przyjmuje tylko wartości „T” lub „N”.

```
CREATE TABLE [dbo].[Rezerwacje](
    [id_rezerwacji] [INT] NOT NULL,
    [Data_zlozenia] [datetime] NOT NULL,
    [Data_rezerwacji] [datetime] NOT NULL,
```

```
    [id_klienta] [INT] NOT NULL,
    [czy_firmowa_imienna] [VARCHAR](1) NOT NULL,
          CONSTRAINT [PK_Rezerwacje] PRIMARY KEY CLUSTERED
(
    [id_rezerwacji] 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].[Rezerwacje] WITH CHECK ADD CONSTRAINT [CK_Rezerwacje_czy_firmowa_imienna]
CHECK (([czy_firmowa_imienna] LIKE '[TN]'))
GO
ALTER TABLE [dbo].[Rezerwacje] CHECK CONSTRAINT [CK_Rezerwacje_czy_firmowa_imienna]
GO


    ALTER TABLE [dbo].[Rezerwacje] ADD CONSTRAINT [DF_Rezerwacje_Data_złożenia]
DEFAULT (GETDATE()) FOR [Data_zlozenia] GO

ALTER TABLE [dbo].[Rezerwacje]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Klienci] FOREIGN
KEY([id_klienta])
REFERENCES [dbo].[Klienci] ([id_klienta])
GO
```

```
ALTER TABLE [dbo].[Rezerwacje] CHECK CONSTRAINT [FK_Rezerwacje_Klienci]
GO
```

13. **Tabela Stoliki** - reprezentuje zbiór informacji dotyczących stolików w restauracjach.
    - **ID_stolika (int)** stanowi klucz główny, potrzebny do identyfikacji stolików.
    - **Max_liczba_miejsc (int)** określa liczbę miejsc dostępnych przy stoliku
    - **czy_aktualne** określa czy stolik jest nadal aktualny

Maksymalna liczba miejsc przy stoliku musi być dodatnia.
Pole czy_akualne może przyjmować wartości “T” lub “N”.

```
CREATE TABLE [dbo].[Stoliki](
    [id_stolika] [INT] NOT NULL,
    [max_liczba_miejsc] [INT] NOT NULL,
[czy_aktualne] [VARCHAR](1) NOT NULL,

 CONSTRAINT [PK_Stoliki] PRIMARY KEY CLUSTERED
(
    [id_stolika] 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].[Stoliki] WITH CHECK ADD CONSTRAINT [CK_Stoliki_max_liczba_miejsc] CHECK
(([max_liczba_miejsc]>(0)))
GO
ALTER TABLE [dbo].[Stoliki] CHECK CONSTRAINT [CK_Stoliki_max_liczba_miejsc]
GO

ALTER TABLE [dbo].[Stoliki] WITH CHECK ADD CONSTRAINT [CK_Stoliki_czy_aktualne]
CHECK (([czy_aktualne] LIKE '[TN]'))
GO
ALTER TABLE [dbo].[Rezerwacje] CHECK CONSTRAINT [CK_Stoliki_czy_aktualne]
GO
```

14. **Tabela Szczegóły_Rezerwacji** - zawiera informację o stoliku, na jaki złożono rezerwację.
    - **ID_rezerwacji (int)** - klucz główny
    - **ID_stolika (int)** stanowi klucz obcy, potrzebny do identyfikacji stolików.
    - **liczba_miejsc (int)** określa liczbę miejsc dostępnych przy stoliku

Liczba miejsc przy stoliku jest dodatnia.

```
CREATE TABLE [dbo].[Szczegoly_rezerwacji](
    [id_rezerwacji] [INT] NOT NULL,
    [id_stolika] [INT] NOT NULL,
    [liczba_miejsc] [INT] NOT NULL
) ON [PRIMARY]
GO


ALTER TABLE [dbo].[Szczegoly_rezerwacji]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegoly_rezerwacji_Rezerwacje] FOREIGN KEY([id_rezerwacji])
REFERENCES [dbo].[Rezerwacje] ([id_rezerwacji])
GO

ALTER TABLE [dbo].[Szczegoly_rezerwacji] CHECK CONSTRAINT
[FK_Szczegoly_rezerwacji_Rezerwacje]
GO

ALTER TABLE [dbo].[Szczegoly_rezerwacji]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegoly_rezerwacji_Stoliki] FOREIGN KEY([id_stolika])
REFERENCES [dbo].[Stoliki] ([id_stolika])
GO
ALTER TABLE [dbo].[Szczegoly_rezerwacji] CHECK CONSTRAINT [FK_Szczegoly_rezerwacji_Stoliki]
GO
```

```
ALTER TABLE [dbo].[Szczegoly_rezerwacji]  WITH CHECK ADD CONSTRAINT
[CK_Szczegoly_rezeracji_liczba_miejsc] CHECK  (([max_liczba_miejsc]>(0)))
GO
ALTER TABLE [dbo].[Szczegoly_rezerwacji] CHECK CONSTRAINT
[CK_Szczegoly_rezerwacji_liczba_miejsc]
GO
```

15. **Tabela Szczegóły_Zamowien** - tabela zawierająca szczegóły złożonych zamówień.

- **ID_Zamówienia (int)** to klucz główny określający którego zamówienia dotyczą szczegóły
- **ID_Pozycji (int)** to drugi klucz główny, który mówi która pozycja z menu została zamówiona
- **Cena_Jednostkowa (money)** zawiera informację o jednostkowej cenie dania w chwili składania zamówienia
- **Ilość (int)** to informacja w jakiej ilości wskazane danie zostało zamówiona
- **rabat(float)** informacja o rabacie do zamówienia

Cena jednostkowa musi być większa od zera.
Ilość musi być większa od zera

```
CREATE TABLE [dbo].[Szczegoly_zamowien](
    [id_zamowienia] [INT] NOT NULL,
    [id_pozycji] [INT] NOT NULL,
```

```
    [cena_jednostkowa] [money] NOT NULL,
    [ilosc] [INT] NOT NULL,
    [rabat] [FLOAT] NULL
) ON [PRIMARY]
GO


ALTER TABLE [dbo].[Szczegoly_zamowien]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegoly_zamowien_Menu] FOREIGN KEY([id_pozycji])
REFERENCES [dbo].[Menu] ([id_pozycji])
GO

ALTER TABLE [dbo].[Szczegoly_zamowien] CHECK CONSTRAINT [FK_Szczegoly_zamowien_Menu]
GO

ALTER TABLE [dbo].[Szczegoly_zamowien]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegoly_zamowien_Zamowienia] FOREIGN KEY([id_zamowienia])
REFERENCES [dbo].[Zamowienia] ([id_zamowienia])
GO

ALTER TABLE [dbo].[Szczegoly_zamowien] CHECK CONSTRAINT [FK_Szczegoly_zamowien_Zamowienia]
GO

ALTER TABLE [dbo].[Szczegoly_Zamowien] WITH CHECK ADD CONSTRAINT [CK_Szczegóły_Zamowien_cena]
CHECK (([Cena_jednostkowa]>(0)))
```

```
GO
ALTER TABLE [dbo].[Szczegoly_Zamowien] CHECK CONSTRAINT [CK_Szczegoly_Zamowien_cena]
GO
ALTER TABLE [dbo].[Szczegoly_Zamowien] WITH CHECK ADD CONSTRAINT
[CK_Szczegoly_Zamowien_ilosc]
CHECK (([Ilość]>(0)))
GO
ALTER TABLE [dbo].[Szczegoly_Zamowien] CHECK CONSTRAINT [CK_Szczegoly_Zamowien_ilosc]
GO
```

Zamówienia - tabela zawierająca informacje o zamówieniach realizowanych w restauracji

- **ID_Zamówienia (int)** to klucz główny jednoznacznie identyfikujący każde zamówienie
- **ID_Klienta (int)** określa klienta, który złożył wskazane zamówienie
- **Data_zamowienia (datetime)** określa datę, kiedy zamówienie zostało złożone
- **Data_odbioru (datetime)** określa datę, kiedy zamówienie zostało zrealizowane, bądź na kiedy ma takie być (obsługa zamówienia z wyprzedzeniem)
- **czy_na_wynos (varchar)** określa, czy dane zamówienie ma być zrealizowane na wynos bądź nie
- **id_pracownika (int)** zawiera identyfikator pracownika, który zamówienie przyjął
- **id_rezerwacji (int)** zawiera identyfikator rezerwacji, na którą zostało złożone zamówienie
- **id_faktury (int)** zawiera odnośnik do informacji do wystawienia faktury
- **status_faktury (int)** okresla czy faktura została wystawiona i w jakiej postaci

Data zamówienia domyślnie ma wartość getDate().
Pole czy_na_wynos przyjmuje tylko wartości „T” lub „N”.
Pole status_faktury przyjmuje tylko wartości “J” - jednorazowa, “M” - miesięczna albo NULL w przypadku, gdy nie została wystawiona faktura na to zamówienie

```
CREATE TABLE [dbo].[Zamowienia](
    [id_zamowienia] [INT] NOT NULL,
    [id_klienta] [INT] NOT NULL,
    [data_zamowienia] [datetime] NOT NULL,
    [data_odbioru] [datetime] NOT NULL,
    [czy_na_wynos] [VARCHAR](1) NOT NULL,
    [id_pracownika] [INT] NOT NULL,
    [id_rezerwacji] [INT] NOT NULL,
    [id_faktury] [INT] NOT NULL,
    [status_faktury] [VARCHAR](1) NULL,
 CONSTRAINT [PK_Zamowienia] PRIMARY KEY CLUSTERED
(
    [id_zamowienia] 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].[Zamowienia]  WITH CHECK ADD  CONSTRAINT [FK_Zamowienia_Klienci1] FOREIGN
KEY([id_klienta])
REFERENCES [dbo].[Klienci] ([id_klienta])
```

```
GO

ALTER TABLE [dbo].[Zamowienia] CHECK CONSTRAINT [FK_Zamowienia_Klienci1]
GO
ALTER TABLE [dbo].[Zamowienia]  WITH CHECK ADD  CONSTRAINT [FK_Zamowienia_Obsluga] FOREIGN
KEY([id_pracownika])
REFERENCES [dbo].[Obsluga] ([id_pracownika])
GO

ALTER TABLE [dbo].[Zamowienia] CHECK CONSTRAINT [FK_Zamowienia_Obsluga]
GO

ALTER TABLE [dbo].[Zamowienia] ADD CONSTRAINT [DF_Zamowienia_Data_zamowienia] DEFAULT (GETDATE())
FOR [Data_zamowienia]
GO

ALTER TABLE [dbo].[Zamowienia] WITH CHECK ADD CONSTRAINT [CK_Zamowienia_czy_na_wynos] CHECK
(([czy_na_wynos] LIKE '[TN]'))
GO
ALTER TABLE [dbo].[Zamowienia] CHECK CONSTRAINT [CK_Zamowienia_czy_na_wynos]
GO

ALTER TABLE [dbo].[Zamowienia] WITH CHECK ADD CONSTRAINT [CK_Zamowienia_status_faktury] CHECK
(([status_faktury] LIKE '[JM]') OR [status_faktury] IS NULL)
GO
ALTER TABLE [dbo].[Zamowienia] CHECK CONSTRAINT [CK_Zamowienia_status_faktury]
```

GO

Faktury - tabela zawierająca informacje na podstawie, których można wystawić fakturę

- **ID_faktury (int)** to klucz główny jednoznacznie identyfikujący, każdy zestaw danych do konkretnego zamówienia na podstawie których zostaje wystawiona faktura
- **Data_wystawienia (datetime)** określa datę, kiedy faktura została utworzona
- **Nazwa_firmy (varchar),** będące nazwą firmy
- **Numer NIP (varchar)**
- Adres, na który składa się**: ulica (varchar), kod_pocztowy (varchar), nazwa_miasta (int), nazwa_panstwa (int)**
- **nr_tel (varchar),** numer telefonu klienta
- **email (varchar)**, będący adresem e-mail klienta
- **imie (varchar), czyli imię klienta**
- **nazwisko (varchar), czyli nazwisko konkretnego klienta**

Data wystawienia faktury domyślnie ma wartość getDate().
Pozostałe warunki integralności sprawdzane są na etapie dodawania danych klienta.

```
CREATE TABLE [dbo].[Faktury](
    [id_faktury] [int] NOT NULL,
    [data_wystawienia] [datetime] NOT NULL,
    [nazwa_firmy] [VARCHAR](50),
    [NIP] [VARCHAR](10),
    [ulica] [VARCHAR](50) NOT NULL,
    [kod_pocztowy] [VARCHAR](50),
    [nazwa_panstwa] [VARCHAR],
    [nazwa_miasta] [VARCHAR],
    [nr_tel] [VARCHAR](9),
    [email] [VARCHAR](50),
    [imie] [VARCHAR](50),
    [nazwisko] [VARCHAR](50),
 CONSTRAINT [PK_Faktury] PRIMARY KEY CLUSTERED
(
    [id_faktury] 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].[Faktury]  WITH CHECK ADD  CONSTRAINT [FK_Faktury_Zamowienia] FOREIGN
KEY([id_zamowienia])
REFERENCES [dbo].[Zamowienia] ([id_zamowienia])
GO
ALTER TABLE [dbo].[Faktury] CHECK CONSTRAINT [FK_Faktury_Zamowienia]
```

```
GO
ALTER TABLE [dbo].[Faktury] ADD CONSTRAINT [DF_Faktury_Data_wystawienia]
DEFAULT (GETDATE()) FOR
[Data_wystawienia]
GO
```

Widoki:

V_Najpopularniejsze_Dania - Lista najczęściej zamawianych potraw w ostatnim miesiącu.

```
CREATE VIEW [dbo].[V_Najpopularniejsze_Dania]
AS
SELECT TOP (20) PERCENT dbo.Dania.nazwa_dania, dbo.Dania.cena_dania,
    SUM(dbo.Szczegoly_zamowien.ilosc) AS Liczba_zamowionych_jednostek
FROM dbo.Menu
INNER JOIN dbo.Dania ON dbo.Menu.ID_dania = dbo.Dania.ID_dania
INNER JOIN dbo.Szczegoly_zamowien ON dbo.Menu.ID_pozycji = dbo.Szczegoly_zamowien.id_pozycji
INNER JOIN dbo.Zamowienia ON Zamowienia.id_zamowienia = Szczegoly_zamowien.id_zamowienia
WHERE DATEPART(m,data_zamowienia) = DATEPART(m, DATEADD(m, -1, getdate()))
```

```
    AND DATEPART(yyyy, data_zamowienia) = DATEPART(yyyy, DATEADD(m, -1, getdate()))
GROUP BY dbo.Dania.Nazwa_dania, dbo.Dania.Cena_dania, dbo.Dania.ID_dania
ORDER BY Liczba_zamowionych_jednostek DESC
GO
```

V_Owoce_morza – Lista dań z kategorii owoce morza.

```
CREATE VIEW [dbo].[V_Owoce_Morza]
AS
SELECT dbo.Dania.nazwa_dania, dbo.Dania.cena_dania
FROM dbo.Dania
INNER JOIN dbo.Kategorie ON dbo.Dania.id_kategorii = dbo.Kategorie.id_kategorii
WHERE (dbo.Kategorie.nazwa_kategorii = 'owoce morza')
GO
```

Dania_z_kategorii - lista dań z podanej kategorii

```
CREATE FUNCTION [dbo].[Dania_Z_Kategorii]
(
```

```
 @id_kategorii INT)
RETURNS TABLE
AS
RETURN
(
 SELECT Nazwa_dania,Cena_dania,nazwa_kategorii FROM Dania
 JOIN Kategorie ON Kategorie.id_kategorii = Dania.id_kategorii
 WHERE Dania.id_kategorii = @id_kategorii
)
GO
```

Rezerwacje_Klienta - lista rezerwacji dla danego klienta, (które się jeszcze nie odbyły)

```
CREATE FUNCTION [dbo].[V_Rezerwacje_klienta]
(
    @id_klienta INT
)
RETURNS TABLE
AS
RETURN
(
SELECT r.Data_zlozenia,r.Data_rezerwacji,s.id_stolika,sr.liczba_miejsc
FROM Rezerwacje r
JOIN Szczegoly_rezerwacji sr ON sr.id_rezerwacji = r.id_rezerwacji
```

```
JOIN Stoliki s ON s.id_stolika = sr.id_stolika
WHERE id_klienta = @id_klienta AND r.Data_rezerwacji > GETDATE()
)
```

V_Klienci_Wydatki – Statystyka łącznych kwot wydanych przez klientów

```
CREATE VIEW [dbo].[V_Klienci_Wydatki]
AS
SELECT
dbo.Klienci.id_klienta,SUM(dbo.Szczegoly_zamowien.ilosc*dbo.Szczegoly_zamowien.cena_jednostko
wa) AS laczna_wartosc
FROM dbo.Klienci
JOIN Zamowienia ON Zamowienia.id_klienta = Klienci.id_klienta
JOIN Szczegoly_zamowien ON Szczegoly_zamowien.id_zamowienia = Zamowienia.id_zamowienia
GROUP BY Klienci.id_klienta
GO
```

Aktualne_Rabaty_Klienta - lista wszystkich rabatów, które klient aktualnie posiada

```
CREATE FUNCTION [dbo].[Aktualne_rabaty_klienta]
(
    @id_klienta INT
```

```
)
RETURNS TABLE
AS
RETURN
(
    SELECT ar.data_przyznania,ar.data_wygasniecia,r.wysokosc_rabatu
    FROM Aktualnie_przyznane_rabaty ar
    JOIN Rabaty r ON r.id_rabatu = ar.id_rabatu
    WHERE ar.id_klienta = @id_klienta
)
```

Nalicz_Rabat_Ind_Staly – zwraca wysokość posiadanego rabatu stałego przez danego klienta indywidualnego

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Ind_Staly]
(
    @id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @id_rabatu INT = (
    SELECT TOP 1 r.ID_rabatu FROM Rabaty r
    INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
```

```
    INNER JOIN Rabaty_Ind_Stale rs ON rs.ID_rabatu=r.ID_rabatu
    WHERE (GETDATE() >= r.data_wprowadzenia AND r.Data_zdjecia IS NULL)
    OR (GETDATE() BETWEEN r.data_wprowadzenia AND r.data_zdjecia)
    ORDER BY r.wysokosc_rabatu DESC)
    IF @id_rabatu IS NULL
    BEGIN
        RETURN 0
    END
    RETURN (SELECT wysokosc_rabatu FROM Rabaty WHERE id_rabatu = @id_rabatu)
END
```

Nalicz_Rabat_Ind_Jedn – zwraca wysokość posiadanego rabatu jednorazowego przez danego klienta indywidualnego

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Ind_Jedn]
(
    @id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @id_rabatu INT = (
```

```
    SELECT TOP 1 r.ID_rabatu FROM Rabaty r
    INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
    INNER JOIN Rabaty_ind_jedn rj ON rj.ID_rabatu=r.ID_rabatu
    WHERE (GETDATE() >= a.data_przyznania AND r.Data_zdjecia IS NULL)
    OR (GETDATE() BETWEEN a.data_przyznania AND a.data_wygasniecia)
    ORDER BY r.wysokosc_rabatu DESC)
    IF @id_rabatu IS NULL
    BEGIN
        RETURN 0
    END
    RETURN (SELECT wysokosc_rabatu FROM Rabaty WHERE id_rabatu = @id_rabatu)
END
```

Ilosc_Zamowien_Powyzej_Kwoty – zwraca liczbe zamowien powyżej wskazanej kwoty dla wskazanego klienta

```
CREATE FUNCTION [dbo].[Ilosc_Zamowien_Powyzej_Kwoty]
(
    @id_klienta INT,
    @kwota MONEY
)
RETURNS INT
AS
BEGIN
```

```
RETURN (
SELECT COUNT(liczba_zam) FROM
(
SELECT COUNT(DISTINCT z.ID_zamowienia) AS liczba_zam FROM Zamowienia z
INNER JOIN Szczegoly_Zamowien sz ON sz.ID_zamowienia=z.ID_zamowienia
WHERE z.ID_klienta=@id_klienta AND z.id_pracownika IN (SELECT
ID_pracownika FROM Obsluga)
GROUP BY z.ID_zamowienia
HAVING SUM(sz.Ilosc*sz.Cena_jednostkowa)>@kwota
) AS zamowienia
)
END
GO
```

Liczba_Wolnych_Miejsc – zwraca dostępną liczbę miejsc w danej restauracji danego dnia z uwzględnieniem już zarezerwowanych

```
CREATE FUNCTION [dbo].[Liczba_wolnych_miejsc]
(
    @data_od DATETIME,
    @data_do DATETIME
```

```
)
RETURNS INT
AS
BEGIN
    DECLARE @miejsc_w_restauracji INT = (SELECT SUM(max_liczba_miejsc) FROM Stoliki)
    DECLARE @zajete_miejsca INT = (SELECT SUM(liczba_miejsc) FROM Szczegoly_rezerwacji
    JOIN Rezerwacje ON Rezerwacje.id_rezerwacji = Szczegoly_rezerwacji.id_rezerwacji
    WHERE Rezerwacje.Data_rezerwacji BETWEEN @data_od AND DATEADD(HOUR,3,@data_do))
    RETURN (@miejsc_w_restauracji-@zajete_miejsca)
END
GO
```

Pokaz_Menu - zwraca aktualną kartę dań

```
CREATE VIEW [dbo].[Pokaz_menu]
AS
SELECT nazwa_dania,cena_dania,opis_dania FROM Dania
JOIN Menu ON Menu.id_dania = Dania.id_dania
WHERE (Menu.Data_zdjecia IS NOT NULL AND GETDATE() BETWEEN Menu.data_wprowadzenia AND
Menu.data_zdjecia)
OR (Menu.Data_zdjecia IS NULL AND GETDATE() >= Menu.data_wprowadzenia)
GO
```

Pokaz_menu_dnia - zwraca kartę dań z podanego dnia w historii

```
CREATE FUNCTION [dbo].[Pokaz_menu_z_dnia]
(
    @data DATE
)
RETURNS TABLE
AS
RETURN
(
    SELECT nazwa_dania,cena_dania,opis_dania,nazwa_kategorii,data_wprowadzenia,data_zdjecia
FROM Dania
    JOIN Menu ON Menu.id_dania = Dania.id_dania
    JOIN Kategorie ON Kategorie.id_kategorii = Dania.id_kategorii
    WHERE (Data_zdjecia IS NULL AND Data_wprowadzenia<=@data)
    OR
    (Data_zdjecia IS NOT NULL and @data BETWEEN Data_wprowadzenia AND Data_zdjecia)
)
```

V_Cena_za_zamowienie - zwraca wartosc zamowienia i jego id

```
CREATE VIEW [dbo].[V_Cena_za_zamowienie]
AS
SELECT id_zamowienia, SUM(cena_jednostkowa * ilosc * (1 - rabat)) AS 'wartosc_zamowienia'
FROM Szczegoly_zamowien
GROUP BY id_zamowienia
GO
```

V_Faktury - zwraca wartosc zamowienia i dane do faktury, a także informacje czy faktura została wystawiona, a jeśli tak to czy była to faktura miesięczna czy faktura za pojedyncze zamówienie

```
CREATE VIEW [dbo].[V_Faktury]
AS
SELECT Zamowienia.id_klienta, Zamowienia.id_zamowienia, Zamowienia.id_faktury,
status_faktury, data_wystawienia, IIF(nazwa_firmy IS NULL, 'brak danych', nazwa_firmy) AS
'nazwa_firmy',
IIF(NIP IS NULL, 'brak danych', NIP) AS 'NIP', IIF(ulica IS NULL, 'brak danych', ulica) AS
'ulica',
IIF(kod_pocztowy IS NULL, 'brak danych', kod_pocztowy) AS 'kod_pocztowy', IIF(nazwa_panstwa
IS NULL, 'brak danych', nazwa_panstwa) AS 'nazwa_panstwa',
IIF(nazwa_miasta IS NULL, 'brak danych', nazwa_miasta) AS 'nazwa_miasta', IIF(nr_tel IS NULL,
'brak danych', nr_tel) AS 'nr_tel',
```

```
IIF(email IS NULL, 'brak danych', email) AS 'email', IIF(nazwisko IS NULL, 'brak danych',
nazwisko) AS 'nazwisko',
IIF(imie IS NULL, 'brak danych', imie) AS 'imie', V_Cena_za_zamowienie.wartosc_zamowienia
FROM Zamowienia
INNER JOIN Faktury ON Zamowienia.id_faktury = Faktury.id_faktury
INNER JOIN V_Cena_za_zamowienie ON V_Cena_za_zamowienie.id_zamowienia =
Zamowienia.id_zamowienia
GO
```

Faktura_za_pojedyncze_zamowienie - zwraca dane do faktury za konkretne zamowienie, jeśli została wystawiona faktura na to pojedyncze zamówienie

```
CREATE FUNCTION [dbo].[Faktura_za_pojdeyncze_zamowienie]
(
 @id_zamowienia INT
)
RETURNS TABLE
AS
RETURN
(
 SELECT * FROM V_Faktury WHERE status_faktury = 'J' AND id_zamowienia = @id_zamowienia
)
GO
```

Faktura_za_miesiac - zwraca dane do faktury za dany miesiąc, rok oraz numer klienta, jeśli taka faktura istnieje

```
CREATE FUNCTION [dbo].[Faktura_za_miesiac]
(
 @id_klienta INT,
 @miesiac INT,
 @rok INT
)
RETURNS TABLE
AS
RETURN
(
 SELECT Zamowienia.id_klienta, data_wystawienia, IIF(nazwa_firmy IS NULL, 'brak danych',
nazwa_firmy) AS 'nazwa_firmy',
    IIF(NIP IS NULL, 'brak danych', NIP) AS 'NIP', IIF(ulica IS NULL, 'brak danych', ulica)
AS 'ulica',
    IIF(kod_pocztowy IS NULL, 'brak danych', kod_pocztowy) AS 'kod_pocztowy',
IIF(nazwa_panstwa IS NULL, 'brak danych', nazwa_panstwa) AS 'nazwa_panstwa',
    IIF(nazwa_miasta IS NULL, 'brak danych', nazwa_miasta) AS 'nazwa_miasta', IIF(nr_tel IS
NULL, 'brak danych', nr_tel) AS 'nr_tel',
    IIF(email IS NULL, 'brak danych', email) AS 'email', IIF(nazwisko IS NULL, 'brak danych',
nazwisko) AS 'nazwisko',
    IIF(imie IS NULL, 'brak danych', imie) AS 'imie',
SUM(V_Cena_za_zamowienie.wartosc_zamowienia) AS 'całkowita kwota'
    FROM Zamowienia
```

```
    INNER JOIN Faktury ON Zamowienia.id_faktury = Faktury.id_faktury
    INNER JOIN V_Cena_za_zamowienie ON V_Cena_za_zamowienie.id_zamowienia =
Zamowienia.id_zamowienia
    WHERE id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND YEAR(data_odbioru)
= @rok AND status_faktury = 'N'
    GROUP BY id_klienta, data_wystawienia, nazwa_firmy, NIP, ulica, kod_pocztowy,
nazwa_panstwa, nazwa_miasta, nr_tel, email, nazwisko, imie
)
GO
```

Szczegoly_faktury_pojedyncze_zamowienie - zwraca liste dan, nazwe kategorii, ilosc oraz cene produktow do konkretnego zamowienia

```
CREATE FUNCTION [dbo].[Szczegoly_faktury_pojedyncze_zamowienie]
(
 @id_zamowienia INT
)
RETURNS TABLE
AS
RETURN
(
```

```
 SELECT nazwa_dania, nazwa_kategorii, ilosc, CONVERT(DECIMAL(8, 2), (cena_jednostkowa * (1 -
rabat) * ilosc)) AS 'cena' FROM zamowienia
INNER JOIN szczegoly_zamowien ON zamowienia.id_zamowienia = szczegoly_zamowien.id_zamowienia
INNER JOIN menu ON menu.id_pozycji = szczegoly_zamowien.id_pozycji
INNER JOIN dania ON menu.id_dania = dania.id_dania
INNER JOIN kategorie ON kategorie.id_kategorii = dania.id_kategorii
WHERE zamowienia.id_zamowienia = @id_zamowienia
)
GO
```

Szczegoly_zamowien_do_faktury_miesiecznej - zwraca listę dań, nazwę kategorii, ilość, cenę oraz numer zamówienia tych zamówień dla których w tym miesiącu została wystawiona faktura

```
CREATE FUNCTION [dbo].[Szczegoly_zamowien_do_faktury_miesiecznej]
(
 @id_klienta INT,
 @miesiac INT,
 @rok INT
)
RETURNS TABLE
AS
RETURN
```

```
(
SELECT zamowienia.id_zamowienia, nazwa_dania, nazwa_kategorii, ilosc, CONVERT(DECIMAL(8, 2),
(cena_jednostkowa * (1 - rabat) * ilosc)) AS 'cena' FROM zamowienia
INNER JOIN szczegoly_zamowien ON zamowienia.id_zamowienia = szczegoly_zamowien.id_zamowienia
INNER JOIN menu ON menu.id_pozycji = szczegoly_zamowien.id_pozycji
INNER JOIN dania ON menu.id_dania = dania.id_dania
INNER JOIN kategorie ON kategorie.id_kategorii = dania.id_kategorii
WHERE zamowienia.id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND
YEAR(data_odbioru) = @rok AND status_faktury = 'M'
)
GO
```

Procedury:

Dodaj_Klienta - procedura pomocnicza przy rejestracji wszystkich rodzajów klientów w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta](
  @email VARCHAR(50),
  @telefon VARCHAR(9)) AS
BEGIN
```

```
  SET NOCOUNT ON;
  BEGIN TRY
    IF EXISTS(
      SELECT * FROM Klienci
      WHERE email=@email
    )
  BEGIN
  ;THROW 52000, 'Email jest juz zajety',1
  END
  IF EXISTS(
    SELECT * FROM Klienci
    WHERE nr_tel=@telefon
  )
  BEGIN
  ;THROW 52000, 'Telefon jest juz zajety',1
  END
  INSERT INTO Klienci(nr_tel,email) VALUES (@telefon,@email)
  END TRY
  BEGIN CATCH
    DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta: '+ ERROR_MESSAGE () ;
    THROW 52000 , @errorMsg ,1;
  END CATCH
END
GO
```

Dodaj_Klienta_Ind - pełna rejestracja klienta indywidualnego w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta_Ind](
@imię VARCHAR(30),
@nazwisko VARCHAR(30),
@email VARCHAR(50),
@telefon VARCHAR(9)) AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
    BEGIN TRAN Dodaj_Klienta_Ind
    EXEC dbo.Dodaj_Klienta
    @email,
    @telefon
    DECLARE @id INT = @@IDENTITY
    INSERT INTO Klienci_Ind(id_klienta,imie,nazwisko) VALUES (@id,@imie,@nazwisko)
    COMMIT TRAN Dodaj_Klienta_Ind
  END TRY
  BEGIN CATCH
    ROLLBACK TRAN Dodaj_Klienta_Ind
    DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta indywidualnego: '+
```

```
    ERROR_MESSAGE () ;
    THROW 52000 , @errorMsg ,1;
  END CATCH
END
GO
```

Dodaj_Klienta_Firm - pełna rejestracja klienta firmowego w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta_Firm](
@email VARCHAR(50),
@telefon VARCHAR(9),
@nazwa_firmy VARCHAR(50),
@nip VARCHAR(10),
@ulica VARCHAR(50),
@kod VARCHAR(6),
@nazwa_miasta VARCHAR(50),
@nazwa_panstwa VARCHAR(50)) AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
    BEGIN TRAN Dodaj_Klienta_Firm
    EXEC dbo.Dodaj_Klienta
    @email,
    @telefon
```

```
    INSERT INTO Klienci_Firmy
    VALUES (@id,@nazwa_firmy,@nip,@ulica,@kod,@nazwa_panstwa,@nazwa_miasta)
    END
    COMMIT TRAN Dodaj_Klienta_Firm
  END TRY
  BEGIN CATCH
    ROLLBACK TRAN Dodaj_Klienta_Firm
    DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta biznesowego: '+
    ERROR_MESSAGE() ;
    THROW 52000 , @errorMsg ,1;
  END CATCH
END
GO
```

Dodaj_Pracownika - rejestracja pracownika restauracji w systemie

```
CREATE PROCEDURE [dbo].[Dodaj_pracownika]
    @imie VARCHAR(50),
    @nazwisko VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;
```

```
    BEGIN TRY
        INSERT INTO Obsluga VALUES(@imie,@nazwisko)
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad dodania pracownika '+ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Dodaj_Stolik - dodanie nowego stolika do systemu restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_stolik]
    @max_miejsc INT,
    @dostepny VARCHAR(1) = 'T'
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        INSERT INTO Stoliki VALUES(@max_miejsc,@dostepny);
```

```
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad dodania stolika '+ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Przyznaj_rabat_klientowi - przyznaje klientowi rabat

```
CREATE PROCEDURE [dbo].[Przyznaj_rabat_klientowi]
     @id_rabatu INT,
     @id_klienta INT,
     @data_przyznania DATE = NULL,
     @data_wygasniecia DATE = NULL
 AS
 BEGIN
     SET NOCOUNT ON;
     BEGIN TRY
         IF NOT EXISTS( SELECT * FROM Rabaty WHERE id_rabatu = @id_rabatu)
         BEGIN
             ;THROW 52000,'Nie ma takiego rabatu',1;
         END
         DECLARE @data_zdjecia DATE = (SELECT data_zdjecia FROM Rabaty WHERE id_rabatu =
```

```
@id_rabatu);
        IF NOT (@data_zdjecia IS NULL OR @data_zdjecia >= GETDATE())
        BEGIN
            ;THROW 52000,'Ten rabat utracil waznosc',1;
        END
        IF NOT EXISTS(SELECT * FROM Klienci WHERE id_klienta = @id_klienta)
        BEGIN
            ;THROW 52000,'Nie ma takiego klienta',1;
        END
        IF EXISTS (SELECT * FROM Klienci_firmy WHERE id_klienta = @id_klienta) AND
        (EXISTS (SELECT * FROM Rabaty_ind_jedn WHERE id_rabatu = @id_rabatu) OR
        EXISTS (SELECT * FROM Rabaty_ind_stale WHERE id_rabatu = @id_rabatu))
        BEGIN
            ;THROW 52000,'Proba przyznania rabatu firmie',1;
        END
        IF EXISTS(SELECT * FROM Klienci_ind WHERE id_klienta = @id_klienta) AND
        NOT EXISTS (SELECT * FROM Rabaty_ind_jedn WHERE id_rabatu = @id_rabatu) AND
        NOT EXISTS (SELECT * FROM Rabaty_ind_stale WHERE id_rabatu = @id_rabatu)
        BEGIN
            ;THROW 52000,'Proba przyznania rabatu dla klienta indywidualnego',1;
        END
        IF @data_przyznania IS NULL
        BEGIN
            SET @data_przyznania = GETDATE();
        END
```

```
        IF EXISTS(SELECT * FROM Rabaty_ind_jedn WHERE id_rabatu = @id_rabatu)
        BEGIN
            DECLARE @waznosc INT = (SELECT waznosc FROM Rabaty_ind_jedn WHERE id_rabatu =
@id_rabatu);
            SET @data_wygasniecia = DATEADD(dd,@waznosc,@data_przyznania)
        END
        INSERT INTO
Aktualnie_przyznane_rabaty(id_rabatu,id_klienta,data_przyznania,data_wygasniecia)
        VALUES(@id_rabatu,@id_klienta,@data_przyznania,@data_wygasniecia)
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad przyznania rabatu '+ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Odbierz_rabat_klientowi - odbiera dany rabat danemu klientowi

```
CREATE PROCEDURE [dbo].[Odbierz_rabat_klientowi]
(
    @id_rabatu INT,
```

```
    @id_klienta INT
)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN Odbierz_Rabat
            IF NOT EXISTS(SELECT * FROM Rabaty WHERE id_rabatu = @id_rabatu)
            BEGIN
                ;THROW 52000,'Nie ma takiego rabatu',1
            END
            IF NOT EXISTS(SELECT * FROM Klienci WHERE id_klienta = @id_klienta)
            BEGIN
                ;THROW 52000,'Nie ma takiego klienta w bazie',1
            END
            DELETE FROM Aktualnie_przyznane_rabaty WHERE id_rabatu = @id_rabatu
        COMMIT TRAN Odbierz_Rabat
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Odbierz_Rabat
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad wygaszania rabatu '+ERROR_MESSAGE();
        THROW 52000,@errorMsg,1
    END CATCH
END
GO
```

Dodaj_Danie - dodaje danie do bazy. Jeśli kategoria wcześniej się nie pojawiła to automatycznie wstawia do tabeli Kategorie typ dodawanego dania.

```
CREATE PROCEDURE [dbo].[Dodaj_danie]
    @nazwa_dania VARCHAR(50),
    @cena MONEY,
    @nazwa_kategorii VARCHAR(50),
    @opis VARCHAR(255) = NULL,
    @dostepna_ilosc INT /*Ilosc sztuk danego dania jaka restauracja jest w stanie przygotowac
w ciagu dnia*/
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN Dodaj_danie
        IF NOT EXISTS(SELECT * FROM Kategorie WHERE nazwa_kategorii = @nazwa_kategorii)
        BEGIN
            INSERT INTO Kategorie VALUES (@nazwa_kategorii)
        END
        DECLARE @id_kategorii INT = (SELECT id_kategorii FROM Kategorie WHERE nazwa_kategorii
= @nazwa_kategorii)
        IF @cena <= 0
```

```
        BEGIN
            ;THROW 52000,'Cena musi byc wartoscia dodatnia',1;
        END
        IF @dostepna_ilosc <= 0
        BEGIN
            ;THROW 52000,'Dostepna ilosc musi byc wartoscia dodatnia',1;
        END
        INSERT INTO Dania(nazwa_dania,cena_dania,id_kategorii,opis_dania,dostepna_ilosc)
        VALUES(@nazwa_dania,@cena,@id_kategorii,@opis,@dostepna_ilosc)
        COMMIT TRAN Dodaj_danie
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_danie
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad dodania dania '+ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Dodaj_do_menu - dodawanie dania do menu

```
CREATE PROCEDURE [dbo].[Dodaj_Do_Menu]
    @id_dania int,
    @data_wprowadzenia date
```

```
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    BEGIN TRAN Dodaj_Do_Menu
    IF NOT EXISTS (SELECT * FROM Dania WHERE id_dania=@id_dania)
    BEGIN
        ;THROW 52000, 'Nie ma takiego dania',1
    END
    IF NOT EXISTS (SELECT * FROM Menu WHERE @id_dania=ID_dania)
    BEGIN
        INSERT INTO Menu(ID_dania,Data_wprowadzenia)
        VALUES (@id_dania,@data_wprowadzenia)
    END
    ELSE IF (SELECT dbo.Ostatnie_Usuniecie_Z_Menu(@id_dania)) IS NULL
    BEGIN
        ;THROW 52000, 'Nie mozna dodac do menu, poniewaz jest w menu',1
    END
    ELSE IF (SELECT dbo.Ostatnie_Usuniecie_Z_Menu(@id_dania))>@data_wprowadzenia
    BEGIN
        ;THROW 52000, 'Danie nie moze zostac znow dodane przed data jego ostatniego
usuniecia',1
    END
    ELSE
    BEGIN
```

```
        INSERT INTO Menu(ID_dania,Data_wprowadzenia)
        VALUES (@id_dania,@data_wprowadzenia)
    END
    COMMIT TRAN Dodaj_Do_Menu
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_Do_Menu
        DECLARE @errorMsg nvarchar (2048) = 'Blad dodania do menu: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Usun_Z_Menu - dodaje datę zdjęcia dania w tabeli Menu

```
CREATE PROCEDURE [dbo].[Usun_Z_Menu]
     @id_dania int,
     @data_zdjecia date
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
```

```
    BEGIN TRAN Usun_Z_Menu
    IF NOT EXISTS (SELECT * FROM Menu WHERE @id_dania=ID_dania AND Data_zdjecia IS NULL)
    BEGIN
        ;THROW 52000, 'Nie ma takiej pozycji obecnie w Menu',1
    END
    DECLARE @id_pozycji int = (SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania AND
Data_zdjecia IS NULL)
    IF @data_zdjecia<GETDATE()
    BEGIN
        ;THROW 52000, 'Data zdjecia nie moze byc chwila z przeszlosci',1
    END
    IF @data_zdjecia<(SELECT Data_wprowadzenia FROM Menu WHERE ID_dania=@id_dania AND
Data_zdjecia IS NULL)
    BEGIN
        ;THROW 52000, 'Data zdjecia nie moze wczesniejsza niz dodania',1
    END
    UPDATE Menu SET Data_zdjecia=@data_zdjecia
    COMMIT TRAN Usun_Z_Menu
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Usun_Z_Menu
        DECLARE @errorMsg nvarchar (2048) = 'Blad usuniecia z menu: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

```
GO
```

Anuluj_rezerwacje - anulowanie rezerwacji klienta

```
CREATE PROCEDURE [dbo].[Anuluj_Rezerwacje]
@id_rezerwacji INT
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Szczegoly_Rezerwacji WHERE ID_rezerwacji=@id_rezerwacji
    DELETE FROM Rezerwacje WHERE ID_rezerwacji=@id_rezerwacji
END
GO
```

Dodaj_rabat - zapisuje dane w głównej tabeli rabatów - wywoływany w Dodaj_Rabat_[TYP RABATU]

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat]
    @wysokosc_rabatu float,
    @data_wprowadzenia date=null,
    @data_zdjecia date=null
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    IF NOT @wysokosc_rabatu BETWEEN 0 AND 1
    BEGIN
        ;THROW 52000, 'Rabat jest wartoscia z przedzialu od 0 do 1',1
    END
    IF @data_wprowadzenia is NULL
    BEGIN
        SET @data_wprowadzenia=GETDATE()
    END
    DECLARE @id_rabatu INT = @@IDENTITY
    INSERT INTO Rabaty(id_rabatu,wysokosc_rabatu,data_wprowadzenia,data_zdjecia)
```

```
    VALUES (@id_rabatu,@wysokosc_rabatu,@data_wprowadzenia,@data_zdjecia)
    PRINT 'Dotarlo'
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
         THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Rabat_Ind_Staly – dodanie rabatu stałego dla klientów indywidualnych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Ind_Staly]
    @wysokosc_rabatu FLOAT,
    @data_wprowadzenia DATE=NULL,
    @data_zdjecia DATE=NULL,
    @liczba_zamowien INT,
    @minimalna_kwota MONEY
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    BEGIN TRAN Dodaj_Rabat_Ind_Staly
```

```
IF @liczba_zamowien<=0
BEGIN
    ;THROW 52000, 'Minimalna ilosc zamowien musi byc liczba dodatnia calkowita',1
END
IF @minimalna_kwota < 0
BEGIN
    ;THROW 52000, 'Minimalna kwota zamowienia musi byc liczba dodatnia calkowita',1
END
DECLARE @id_poprzedniego INT = (SELECT ri.ID_rabatu FROM Rabaty_Ind_Stale ri
INNER JOIN Rabaty r ON r.ID_rabatu=ri.ID_rabatu WHERE r.Data_zdjecia IS NULL)
IF @id_poprzedniego IS NOT NULL
BEGIN
    UPDATE Rabaty SET Data_zdjecia=@data_wprowadzenia WHERE
    @id_poprzedniego=ID_rabatu
END
EXEC Dodaj_rabat
@wysokosc_rabatu,
@data_wprowadzenia,
@data_zdjecia

DECLARE @id_rabatu INT=@@IDENTITY
INSERT INTO Rabaty_Ind_Stale(ID_rabatu,minimalna_kwota,liczba_zamowien)
VALUES (@id_rabatu,@minimalna_kwota,@liczba_zamowien)
COMMIT TRAN Dodaj_Rabat_Ind_Staly
END TRY
```

```
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_Rabat_Ind_Staly
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Rabat_Ind_Jednorazowy - dodanie rabatu jednorazowego dla klientów indywidualnych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Ind_Jednorazowy]
    @wymagana_kwota MONEY,
    @wysokosc_rabatu FLOAT,
    @data_wprowadzenia DATE=NULL,
    @data_zdjecia DATE=NULL,
    @waznosc INT
AS
    BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    BEGIN TRAN Dodaj_Rabat_Ind_Jedn
    IF @waznosc<=0
    BEGIN
```

```
        ;THROW 52000, 'Rabat musi miec waznosc przynajmniej 1 dzien',1
    END
    IF @wymagana_kwota<=0
    BEGIN
        ;THROW 52000, 'Wymagana kwota rabatu musi byc dodatnia liczba calkowita',1
    END
    IF @data_wprowadzenia IS NULL
    BEGIN
        SET @data_wprowadzenia=GETDATE()
    END
    EXEC Dodaj_Rabat
        @wysokosc_rabatu,
        @data_wprowadzenia,
        @data_zdjecia
    DECLARE @id INT = @@IDENTITY
    INSERT INTO Rabaty_Ind_Jednorazowe(ID_rabatu,Waznosc,wymagan_kwota)
    VALUES (@id,@waznosc,@wymagana_kwota)
    COMMIT TRAN Dodaj_Rabat_Ind_Jedn
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_Rabat_Ind_Jedn
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

```
GO
```

Aktualizuj_Cene_Dania – ustawianie nowej ceny dla dania

```
CREATE PROCEDURE [dbo].[Aktualizuj_cene_dania]
    @id_dania INT,
    @nowa_cena MONEY
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @nazwa_dania VARCHAR(50) = (SELECT nazwa_dania FROM Dania WHERE id_dania =
@id_dania)
    IF @nazwa_dania IS NULL
    BEGIN
        ;THROW 52000,'Podane danie nie istnieje!',1;
    END
    IF @nowa_cena <= 0
    BEGIN
        ;THROW 52000,'Nowa cena musi być liczbą dodatnią!',1;
    END
    BEGIN
    UPDATE Dania SET cena_dania = @nowa_cena WHERE id_dania = @id_dania
    END
```

```
    PRINT 'Zaktualizowano pomyślnie'
END
GO
```

Wystaw_fakture_na_konkretne_zamowienie – wystawia fakture

```
CREATE PROCEDURE [dbo].[Wystaw_fakture_na_zamowienie]
    @id_zamowienia INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Zamowienia WHERE id_zamowienia = @id_zamowienia)
            THROW 50001,'Nie ma takiego zamowienia!',1
        ELSE
            UPDATE Zamowienia
            SET status_faktury = 'J'
            WHERE id_zamowienia = @id_zamowienia;

            UPDATE Faktury
            SET data_wystawienia = GETDATE()
            WHERE id_faktury = (SELECT id_faktury FROM Zamowienia WHERE id_zamowienia =
@id_zamowienia);
    END TRY
```

```
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad wystawienia faktury '+ ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Wystaw_fakture_miesieczna – wystawia fakture na konkretny miesiac i rok

```
CREATE PROCEDURE [dbo].[Wystaw_fakture_miesieczna]
    @id_klienta INT,
    @miesiac INT,
    @rok INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Zamowienia
        WHERE id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND
YEAR(data_odbioru) = @rok)
            THROW 50001,'Nie zrobiłeś żadnych zamówień w tym miesiącu!',1
        IF NOT EXISTS (SELECT * FROM Zamowienia
            WHERE id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND
```

```
YEAR(data_odbioru) = @rok AND status_faktury = 'N')
            THROW 50001,'Na wszystkie zamówienia w tym miesiącu faktura została już
wystawiona!',1
        ELSE
            UPDATE Zamowienia
            SET status_faktury = 'M'
            WHERE id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND
YEAR(data_odbioru) = @rok AND status_faktury = 'N'

            UPDATE Faktury
            SET data_wystawienia = GETDATE()
            WHERE id_faktury IN (SELECT id_faktury FROM Zamowienia
            WHERE id_klienta = @id_klienta AND MONTH(data_odbioru) = @miesiac AND
YEAR(data_odbioru) = @rok AND status_faktury = 'N');
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR(2048) = 'Blad wystawienia faktury '+ ERROR_MESSAGE();
        THROW 52000,@errorMsg,1;
    END CATCH
END
GO
```

Usun_Polowe_Pozycji_Z_Menu – usuwa połowę pozycji z tych, które są minimum od dwóch tygodni w menu

```
CREATE PROCEDURE [dbo].[Usun_Polowe_Pozycji_Z_Menu]
    @data date
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE Menu SET Data_zdjęcia=GETDATE() WHERE ID_pozycji IN(
    SELECT TOP 50 PERCENT a.ID_pozycji FROM dbo.Pokaz_Menu_Dnia(DATEADD(day,-1,@data)) a
    INNER JOIN dbo.Pokaz_Menu_Dnia(DATEADD(day,-14,@data)) b ON a.ID_pozycji =
    b.ID_pozycji
    ORDER BY a.Data_wprowadzenia ASC)
END
GO



CREATE PROCEDURE [dbo].[Usun_Z_Menu]
     @nazwa_dania varchar(50),
     @data_zdjecia date
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    BEGIN TRAN Usun_Z_Menu
    DECLARE @id_dania int = (SELECT ID_dania FROM Dania WHERE Nazwa_dania=@nazwa_dania)
```

```
    IF NOT EXISTS (SELECT * FROM Menu WHERE @id_dania=ID_dania AND Data_zdjecia IS NULL)
    BEGIN
        ;THROW 52000, 'Nie ma takiej pozycji obecnie w Menu dla tej restauracji',1
    END
    DECLARE @id_pozycji int = (SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania AND
data_zdjecia IS NULL)
    IF @data_zdjecia<GETDATE()
    BEGIN
        ;THROW 52000, 'Data zdjecia nie moze byc chwila z przeszlosci',1
    END
    IF @data_zdjecia<(SELECT Data_wprowadzenia FROM Menu WHERE ID_dania=@id_dania AND
Data_zdjecia IS NULL)
    BEGIN
        ;THROW 52000, 'Data zdjecia nie moze wczesniejsza niz dodania',1
    END
    UPDATE Menu SET Data_zdjecia=@data_zdjecia
    COMMIT TRAN Usun_Z_Menu
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Usun_Z_Menu
        DECLARE @errorMsg nvarchar (2048) = 'Blad usuniecia z menu: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Element_Do_Zamowienia – dodanie nowego elementu do istniejącego zamówienia

```
CREATE PROCEDURE [dbo].[Dodaj_Element_Do_Zamowienia]
    @id_zamowienia INT,
    @id_dania VARCHAR(50),
    @ilosc INT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    BEGIN TRAN Dodaj_Element
    IF NOT EXISTS (SELECT * FROM Zamowienia WHERE ID_zamowienia=@id_zamowienia)
    BEGIN
        ;THROW 52000, 'Nie ma takiego zamowienia',1
    END
    IF NOT EXISTS (SELECT * FROM Dania WHERE @id_dania=id_dania)
    BEGIN
        ;THROW 52000, 'Nie ma takiego dania',1
    END
    IF @ilosc<=0
```

```
BEGIN
    ;THROW 52000, 'Trzeba zamowic przynajmniej jedna sztuke',1
END
IF (SELECT dostępna_ilosc FROM Dania WHERE id_dania = @id_dania) <= 0
BEGIN
    ;THROW 52000, 'Brak produktow aby zrealizowac to danie',1;
END
IF (SELECT id_kategorii FROM Dania WHERE @id_dania=ID_dania) = (SELECT
ID_kategorii FROM Kategorie WHERE Nazwa_kategorii='owoce morza')
BEGIN
    DECLARE @data_odbioru date =(SELECT Data_odbioru FROM Zamowienia
    WHERE @id_zamowienia=ID_zamowienia)
    DECLARE @data_zamowienia date =(SELECT Data_zamowienia FROM Zamowienia
    WHERE @id_zamowienia=ID_zamowienia)
    IF(DATEPART(w,@data_odbioru)=5 AND DATEDIFF(day,@data_zamowienia,@data_odbioru)<3)
    BEGIN
        ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce morza',1
    END
    IF(DATEPART(w,@data_odbioru)=6 AND DATEDIFF(day,@data_zamowienia,@data_odbioru)<4)
    BEGIN
        ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce morza',1
    END
    IF(DATEPART(w,@data_odbioru)=7 AND DATEDIFF(DAY,@data_zamowienia,@data_odbioru)<5)
    BEGIN
        ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce    morza',1
```

```
        END
    END
    IF NOT EXISTS (SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania AND Data_zdjecia is
NULL)
    BEGIN
        ;THROW 52000, 'To danie nie jest obecnie w ofercie restauracji',1
    END
    DECLARE @id_pozycji int =(SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania AND
Data_zdjecia is NULL)
    DECLARE @rabat FLOAT=0.0
    DECLARE @id_klienta INT =(SELECT ID_klienta FROM Zamowienia WHERE
ID_zamowienia=@id_zamowienia)
    IF @id_klienta IN (SELECT ID_klienta FROM Klienci_Ind)
    BEGIN
    SET
@rabat=@rabat+dbo.Nalicz_Rabat_Ind_Staly(@id_klienta)+dbo.Nalicz_Rabat_Ind_Jednorazowy(@id_kl
ienta)
    IF @rabat>1
    BEGIN
        SET @rabat=1
    END
    DECLARE @cena MONEY = (SELECT Cena_dania FROM Dania WHERE ID_dania=@id_dania)*(1-@rabat)
    INSERT INTO Szczegoly_Zamowien(ID_zamowienia,ID_pozycji,Cena_jednostkowa,Ilosc,rabat)
    VALUES(@id_zamowienia,@id_pozycji,@cena,@ilosc,@rabat)
    COMMIT TRAN Dodaj_Element
```

```
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_Element
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad przy dodaniu dania do zamowienia: '+
ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

Aktualizuj_Rabat_Ind_Staly – aktualizowanie klientowi indywidualnemu rabatu stałego

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Ind_Staly]
    @id_klienta INT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
    INNER JOIN Rabaty_Ind_Stale ri ON ri.ID_rabatu=r.ID_rabatu
    WHERE Data_zdjecia IS NULL)
    IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
    ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
    BEGIN
    DECLARE @liczba_zamowien INT = (SELECT Liczba_zamowien FROM Rabaty_Ind_Stale WHERE
    ID_rabatu=@id_rabatu)
    DECLARE @wymagana_kwota MONEY = (SELECT ri.minimalna_kwota FROM Rabaty_ind_stale ri WHERE
    ID_rabatu=@id_rabatu)
    DECLARE @ilosc_powyzej_kwoty INT =(SELECT
    dbo.Ilosc_Zamowien_Powyzej_Kwoty(@id_klienta,@wymagana_kwota))
    IF (@ilosc_powyzej_kwoty>=@liczba_zamowien)
    BEGIN
    EXEC Przyznaj_Rabat_Klientowi
    @id_rabatu,
    @id_klienta,
    NULL,
```

```
    NULL
    END
    END
END
GO
```

Aktualizuj_Rabat_Ind_Jednorazowy – aktualizowanie klientowi indywidualnemu rabatu jednorazowego

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Ind_Jednorazowy]
    @id_klienta INT
AS
BEGIN
SET NOCOUNT ON;
DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
INNER JOIN Rabaty_Ind_Jedn rj ON rj.ID_rabatu=r.ID_rabatu WHERE Data_zdjecia IS NULL)
IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
BEGIN
DECLARE @laczna_wart_zam MONEY = (SELECT SUM(sz.Ilosc*sz.Cena_jednostkowa) FROM
Szczegoly_Zamowien sz
INNER JOIN Zamowienia z ON z.ID_zamowienia=sz.ID_zamowienia
WHERE @id_klienta=z.ID_klienta)
DECLARE @wymagana_kwota MONEY=(SELECT Wymagana_kwota FROM Rabaty_ind_jedn WHERE
ID_rabatu=@id_rabatu)
```

```
IF @laczna_wart_zam>=@wymagana_kwota
BEGIN
   EXEC Przyznaj_Rabat_Klientowi
   @id_rabatu,
   @id_klienta,
   NULL,
   NULL
END
END
END
GO
```

Dodaj_Zamowienie – dodanie nowego zamówienia do systemu

```
CREATE PROCEDURE [dbo].[Dodaj_Zamowienie]
    @id_klienta INT,
    @data_zamowienia DATETIME,
    @data_odbioru DATETIME = NULL,
    @czy_na_wynos VARCHAR(1),
    @id_pracownika INT
AS
BEGIN
    SET NOCOUNT ON;
```

```
BEGIN TRY
BEGIN TRAN Dodaj_Zamowienie
IF NOT EXISTS (SELECT * FROM Klienci WHERE @id_klienta=ID_klienta)
BEGIN
    ;THROW 52000, 'Nie ma takiego klienta',1
END
IF @data_odbioru IS NULL AND @czy_na_wynos='T'
BEGIN
    ;THROW 52000, 'Przy zamawianiu na wynos trzeba podac date odbioru',1
END
IF @data_odbioru is NULL
BEGIN
    SET @data_odbioru=DATEADD(hh,1,@data_zamowienia)
END
IF NOT EXISTS (SELECT * FROM Obsluga WHERE ID_pracownika=@id_pracownika)
BEGIN
    ;THROW 52000, 'Zamowienie obsluguje nieuprawniona osoba',1
END
DELETE FROM Aktualnie_Przyznane_Rabaty WHERE @id_klienta=ID_klienta AND
Data_wygasniecia IS NOT NULL AND Data_wygasniecia<GETDATE()
DELETE FROM Aktualnie_Przyznane_Rabaty WHERE @id_klienta=ID_klienta AND
ID_rabatu IN(
SELECT ID_rabatu FROM Rabaty WHERE Data_zdjecia IS NOT NULL)
IF @id_klienta IN (SELECT ID_klienta FROM Klienci_Ind)
BEGIN
```

```
    EXEC Aktualizuj_Rabat_Ind_Staly
        @id_klienta
    EXEC Aktualizuj_Rabat_Ind_Jednorazowy
        @id_klienta
    END
    ELSE
    BEGIN
    INSERT INTO
Zamowienia(ID_klienta,Data_zamowienia,Data_odbioru,czy_na_wynos,id_pracownika,id_rezerwacji)
    VALUES (@id_klienta,@data_zamowienia,@data_odbioru,@czy_na_wynos,@id_pracownika,NULL)
    DECLARE iter CURSOR
    FOR
    SELECT a.ID_rabatu
    FROM Aktualnie_Przyznane_Rabaty a
    INNER JOIN Rabaty r ON r.ID_rabatu=a.ID_rabatu
    WHERE @id_klienta=a.ID_klienta
    DECLARE @rabat int
    OPEN iter
    FETCH NEXT FROM iter INTO @rabat
    WHILE @@FETCH_STATUS = 0
    BEGIN
    DECLARE @data_wygasniecia date=(SELECT Data_wygasniecia FROM
    Aktualnie_Przyznane_Rabaty
    WHERE @rabat=ID_rabatu AND @id_klienta=ID_klienta)
    IF @data_wygasniecia<GETDATE()
```

```
    BEGIN
    EXEC Odbierz_Rabat_Klientowi
        @rabat,
        @id_klienta
    END
    FETCH NEXT FROM iter INTO @rabat
    END
    CLOSE iter
    DEALLOCATE iter
    END
    COMMIT TRAN Dodaj_Zamowienie
    END TRY
    BEGIN CATCH
    ROLLBACK TRAN Dodaj_Zamowienie
    DECLARE @errorMsg nvarchar (2048) = 'Blad dodania zamowienia: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

TRIGGERY:

TRIG_Blokuj_wstawianie_powtorek – zapobiega powtórzeniu w aktualnej ofercie restauracji kilkukrotnie tego samego dania

```
CREATE TRIGGER [dbo].[Blokuj_powtorki_w_menu]
ON [dbo].Menu
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @id_pozycji INT = (SELECT id_pozycji FROM inserted)
    DECLARE @id_dania INT = (SELECT id_dania FROM Menu WHERE id_pozycji = @id_pozycji)
    IF EXISTS (SELECT * FROM Menu WHERE id_dania = @id_dania AND data_zdjecia >= GETDATE()
    AND id_pozycji <> @id_pozycji)
    BEGIN
        ;THROW 52000,'To danie znajduje się obecnie w menu restauracji',1;
    END
END
GO
```

TRIG_Menu_Data_Zdjecia – w przypadku zdejmowania pozycji z menu ustawia datę zdjęcia na datę obecną.

```
CREATE TRIGGER [dbo].[TRIG_Menu_Data_Zdjecia]
ON [dbo].[Menu]
FOR UPDATE
AS
BEGIN
SET NOCOUNT ON
UPDATE Menu SET Menu.Data_zdjecia = GETDATE() FROM inserted WHERE
Menu.ID_pozycji=inserted.ID_pozycji
END
GO
```

Indeksy:

IX_Aktualnie_Przyznane_Rabaty

```
CREATE NONCLUSTERED INDEX [IX_Aktualnie_Przyznane_Rabaty] ON
[dbo].[Aktualnie_Przyznane_Rabaty]
(
[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
GO
```

IX_Aktualnie_Przyznane_Rabaty_1

```
CREATE NONCLUSTERED INDEX [IX_Aktualnie_Przyznane_Rabaty_1] ON
[dbo].[Aktualnie_Przyznane_Rabaty]
(
[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
```

```
ON
[PRIMARY]
GO
```

IX_Klienci_Firmy

```
CREATE NONCLUSTERED INDEX [IX_Klienci_Biz] ON [dbo].[Klienci_Firmy]
(
[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
GO
```

IX_Klienci_Ind

```
CREATE NONCLUSTERED INDEX [IX_Klienci_Ind] ON [dbo].[Klienci_Ind]
(
[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
```

```
GO
```

IX_Menu

```
CREATE NONCLUSTERED INDEX [IX_Menu] ON [dbo].[Menu]
(
[ID_dania] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
GO
```

IX_Rabaty_Ind_Jednorazowe

```
CREATE NONCLUSTERED INDEX [IX_Rabaty_Ind_Jednorazowe] ON [dbo].[Rabaty_Ind_Jedn]
(
[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
GO
```

IX_Rabaty_Ind_Stale

```
CREATE NONCLUSTERED INDEX [IX_Rabaty_Ind_Stale] ON [dbo].[Rabaty_Ind_Stale]
(
[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING =
OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON
[PRIMARY]
GO
```