Semestr zimowy 2020/21

# Podstawy Baz Danych

## Projekt: System wspomagania działalności firmy świadczącej usługi gastronomiczne

## Dokumentacja

Autorzy: Paweł Kruczkiewicz, Filip Piwosz

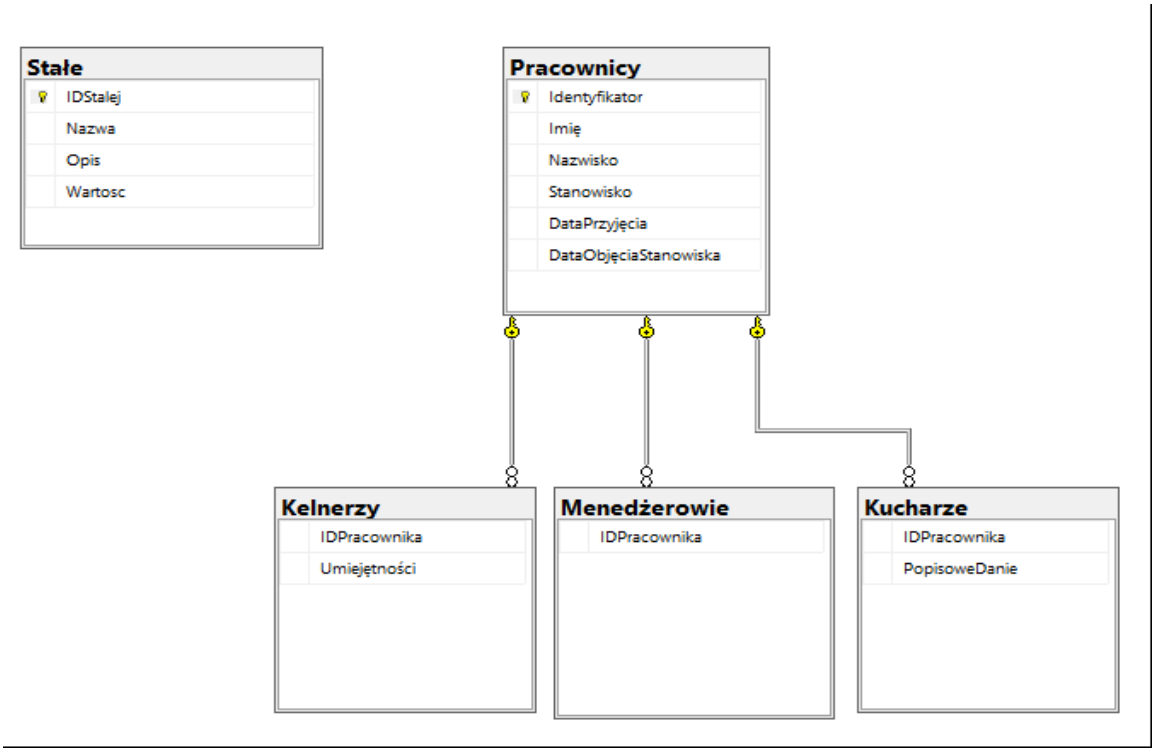
## 1. Informacje wstępne

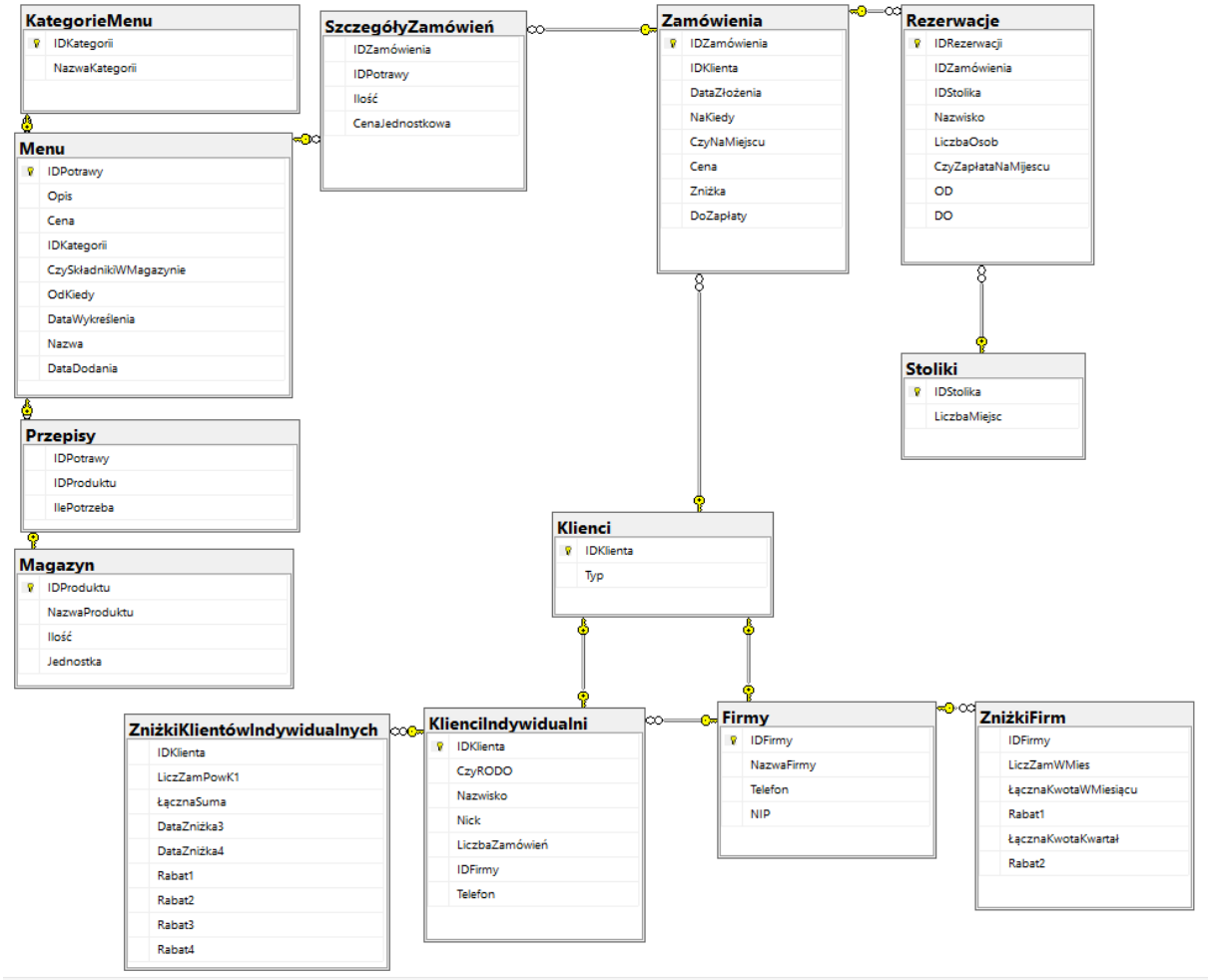
Projekt dotyczy wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. Może być ona użytkowana przez kilka tego typu firm.

System został stworzony z myślą o firmie, w której ofercie znajduje się żywność i napoje bezalkoholowe. Klient może zamawiać produkty na miejscu i na wynos, również z wyprzedzeniem, pod warunkiem uprzedniego wypełnienia formularza na stronie WWW i wyboru daty i godziny. Pod uwagę brane jest również ograniczenie miejsc związane z pandemią COVID.

System sprawuje również pieczę nad różnymi typami rabatów dla klientów indywidualnych oraz firm. Posiada również szereg funkcjonalności dla pracowników firmy, które pomogą m. in. w przygotowaniu menu, archiwizacji dokonanych zamówień czy zarządzaniu personelem.

## 2. Schemat bazy






## 3. Opis tabel

**Menu –** tabela przechowująca dane na temat wszystkich dań i napojów oferowanych przez firmę. Zawiera wszystkie najważniejsze informacje o potrawie zawarte w karcie dań (Nazwa, Cena, Opis), a także informacje o tym, czy składnik potrzebne do zrobienia dania znajdują się w magazynie, datę dodania do bazy danych, daty od kiedy danie dostępne jest w karcie oraz kiedy zostanie z niej wykreślone (kolumny OdKiedy i DataWykreślenia).

**Warunki integralnościowe:**

- IDKategorii to FK odwołujący się do tabeli Kategorie. (modyfikacja kaskadowo, usuwanie „set default")
- Wartość domyślna IDKategorii to 0 (oznacza kategorię „Pozostałe")
- Cena > 0

```
CREATE TABLE [dbo].[Menu](
	[IDPotrawy] [int] IDENTITY(1,1) NOT NULL,
	[Opis] [varchar](255) NULL,
	[Cena] [smallmoney] NOT NULL,
	[IDKategorii] [int] NOT NULL,
	[CzySkładnikiWMagazynie] [bit] NOT NULL,
	[OdKiedy] [date] NOT NULL,
```

```
	[DataWykreślenia] [date] NULL,
	[Nazwa] [varchar](255) NOT NULL,
	[DataDodania] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[IDPotrawy] 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],
UNIQUE NONCLUSTERED
(
	[Nazwa] 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] ADD  DEFAULT ((0)) FOR [IDKategorii]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [fk_IDKategorii] FOREIGN
KEY([IDKategorii])
REFERENCES [dbo].[KategorieMenu] ([IDKategorii])
ON UPDATE CASCADE
ON DELETE SET DEFAULT
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [fk_IDKategorii]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD CHECK  (([Cena]>(0)))
	GO
```

**KategorieMenu –** tabela przechowująca Klucz główny oraz Nazwę danej kategorii

```
CREATE TABLE [dbo].[KategorieMenu](
	[IDKategorii] [int] IDENTITY(1,1) NOT NULL,
	[NazwaKategorii] [varchar](255) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[IDKategorii] 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
```

**Magazyn –** tabela zawierająca informacje o półproduktach potrzebnych znajdujących się w magazynie. Zawiera informacje takie jak nazwa półproduktu, jego ilość oraz jednostkę. **Warunki integralnościowe**:

- Ilość musi być większa lub równa niż 0

```
CREATE TABLE [dbo].[Magazyn](
	[IDProduktu] [int] IDENTITY(1,1) NOT NULL,
```

```
	[NazwaProduktu] [varchar](255) NOT NULL,
	[Ilość] [float] NOT NULL,
	[Jednostka] [varchar](64) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[IDProduktu] 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].[Magazyn]  WITH CHECK ADD  CONSTRAINT [CK__Magazyn__Ilość__114A936A]
CHECK  (([Ilość]>=(0)))
GO

ALTER TABLE [dbo].[Magazyn] CHECK CONSTRAINT [CK__Magazyn__Ilość__114A936A]
GO
```

**Przepisy –** tabela łącząca tabele Magazyn oraz Menu. Zawiera klucz obcy produktu potrzebnego do stworzenia potrawy o danym kluczu. Dla każdej pary IDProduktu i IDPotrawy tabela zapisuje również informację, ile produktu potrzeba, do zrobienia danego dania.

**Warunki integralnościowe:**

- IDProduktu to FK odwołujący się do tabeli Magazyn (usuwanie i aktualizacja – kaskadowe)
- IDPotrawy to FK odwołujący się do tabeli Menu (usuwanie i aktualizacja – kaskadowe)
- IlePotrzeba > 0

```
CREATE TABLE [dbo].[Przepisy](
	[IDPotrawy] [int] NOT NULL,
	[IDProduktu] [int] NOT NULL,
	[IlePotrzeba] [float] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD  CONSTRAINT [fk_IDPotrawy] FOREIGN
KEY([IDPotrawy])
REFERENCES [dbo].[Menu] ([IDPotrawy])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Przepisy] CHECK CONSTRAINT [fk_IDPotrawy]
GO

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD  CONSTRAINT [fk_IDProduktu] FOREIGN
KEY([IDProduktu])
REFERENCES [dbo].[Magazyn] ([IDProduktu])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Przepisy] CHECK CONSTRAINT [fk_IDProduktu]
GO

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD CHECK  (([IlePotrzeba]>(0)))
GO
```

**Zamówienia –** tabela zawierająca wszystkie przeszłe, obecne i przyszłe zamówienia. Przechowuje najpotrzebniejsze informacje o danym zamówieniu (DataZłożenia, NaKiedy, CzyNaMiejscu, Cena, Zniżka). Kolumna DoZapłaty zawiera wartość zamówienia pomniejszoną o wartość zniżki. IDKlienta wskazuje na klienta w bazie, który złożył dane zamówienie. Pole może być NULLem w przypadku, gdy klient złożył zamówienie na miejscu i nie został/ nie chciał zostać zapisany w bazie.

**Warunki integralnościowe**:

- IDKlienta to FK odwołujący się do tabeli Klienci (aktualizacja kaskadowo, usuwanie SET NULL)
- Zniżka musi być większa lub równa 0

```
CREATE TABLE [dbo].[Zamówienia](
	[IDZamówienia] [int] IDENTITY(1,1) NOT NULL,
	[IDKlienta] [int] NULL,
	[DataZłożenia] [datetime] NOT NULL,
	[NaKiedy] [datetime] NOT NULL,
	[CzyNaMiejscu] [bit] NOT NULL,
	[Cena] [float] NOT NULL,
	[Zniżka] [float] NOT NULL,
	[DoZapłaty] [float] NULL,
PRIMARY KEY CLUSTERED
(
	[IDZamówienia] 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].[Zamówienia]  WITH CHECK ADD  CONSTRAINT [fk_IDKlienta2] FOREIGN
KEY([IDKlienta])
REFERENCES [dbo].[Klienci] ([IDKlienta])
ON UPDATE CASCADE
ON DELETE SET NULL
GO

ALTER TABLE [dbo].[Zamówienia] CHECK CONSTRAINT [fk_IDKlienta2]
GO

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD CHECK  (([Zniżka]>=(0)))
GO
```

**Szczegóły zamówień –** tabela zawierająca szczegóły dotyczące zamówienia, tj. potrawy jakie się w nim znajdują wraz z ich ilością oraz ceną jednostkową (dzięki czemu można łatwiej wyliczyć wartość zamówienia).

**Warunki integralnościowe:**

- IDPotrawy to FK odwołujący się do tabeli Menu
- IDZamówienia to FK odwołujący się do tabeli Zamówienia
- Cena jednostkowa oraz Ilość muszą być większe od 0

```
CREATE TABLE [dbo].[Zamówienia](
	[IDZamówienia] [int] IDENTITY(1,1) NOT NULL,
	[IDKlienta] [int] NULL,
	[DataZłożenia] [datetime] NOT NULL,
	[NaKiedy] [datetime] NOT NULL,
```

```
	[CzyNaMiejscu] [bit] NOT NULL,
	[Cena] [float] NOT NULL,
	[Zniżka] [float] NOT NULL,
	[DoZapłaty] [float] NULL,
PRIMARY KEY CLUSTERED
(
	[IDZamówienia] 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].[Zamówienia]  WITH CHECK ADD  CONSTRAINT [fk_IDKlienta2] FOREIGN
KEY([IDKlienta])
REFERENCES [dbo].[Klienci] ([IDKlienta])
ON UPDATE CASCADE
ON DELETE SET NULL
GO

ALTER TABLE [dbo].[Zamówienia] CHECK CONSTRAINT [fk_IDKlienta2]
GO

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD CHECK  (([Zniżka]>=(0)))
GO

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD CHECK  (([Zniżka]>=(0) AND
[Zniżka]<=(1)))
GO
```

**Rezerwacje –** tabela przechowująca informacje dotyczące przeszłych, obecnych i przyszłych rezerwacji. Zawiera wszystkie potrzebne informacje odnośnie danej rezerwacji, tj. IDZamówienia, IDStolika, LiczbęOsób, Nazwisko, CzyZapłataNaMiejscu oraz OD i DO, czyli daty rozpoczęcia i zakończenia rezerwacji. Dla klientów indywidualnych muszą być spełnione specjalne warunki, by złożyć zamówienie.

**Warunki integralnościowe**:

- IDStolika to FK odwołujący się do tablicy Stoliki (usuwanie i modyfikacja – kaskadowo)
- IDZamówienia to FK odwołujący się do tablicy Zamówienia
- LiczbaOsób > 0
- Wartość OD danego rekordu musi być mniejsza od wartości DO

```
CREATE TABLE [dbo].[Rezerwacje](
	[IDRezerwacji] [int] IDENTITY(1,1) NOT NULL,
	[IDZamówienia] [int] NULL,
	[IDStolika] [int] NOT NULL,
	[Nazwisko] [varchar](255) NOT NULL,
	[LiczbaOsob] [int] NOT NULL,
	[CzyZapłataNaMijescu] [bit] NOT NULL,
	[OD] [datetime] NOT NULL,
	[DO] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[IDRezerwacji] 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 FOREIGN KEY([IDZamówienia])
REFERENCES [dbo].[Zamówienia] ([IDZamówienia])
GO

ALTER TABLE [dbo].[Rezerwacje]  WITH CHECK ADD  CONSTRAINT [fk_Stoliki] FOREIGN
KEY([IDStolika])
REFERENCES [dbo].[Stoliki] ([IDStolika])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Rezerwacje] CHECK CONSTRAINT [fk_Stoliki]
GO

ALTER TABLE [dbo].[Rezerwacje]  WITH CHECK ADD CHECK  (([LiczbaOsob]>(0)))
GO

ALTER TABLE [dbo].[Rezerwacje]  WITH CHECK ADD CHECK  (([DO]>[OD]))
   GO
```

**Stoliki –** tabela przechowująca informacje o stolikach. Są to – IDStolika oraz liczba miejsc danego stolika

**Warunki integralnościowe:**

- Liczba miejsc musi być większa niż 0

```
CREATE TABLE [dbo].[Stoliki](
       [IDStolika] [int] IDENTITY(1,1) NOT NULL,
       [LiczbaMiejsc] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
       [IDStolika] 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 CHECK  (([LiczbaMiejsc]>(0)))
   GO
```

**Klienci –** tabela zawierająca najbardziej podstawowe informacje o klientach, czyli ich ID oraz typ (klient indywidualny bądź firma).

**Warunki integralnościowe:**

- Pole typ może zawierać jedynie wartości ‘Firma’ lub ‘Klient Indywidualny’)

```
CREATE TABLE [dbo].[Klienci](
       [IDKlienta] [int] IDENTITY(1,1) NOT NULL,
       [Typ] [varchar](255) NOT NULL,
PRIMARY KEY CLUSTERED
(
       [IDKlienta] 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 CHECK  (([Typ]='Firma' OR [Typ]='Klient
Indywidualny'))
GO
```

**KlienciIndywidualni –** tabela zawierająca informacje o danym kliencie indywidualnym, takie jak Nazwisko, LiczbaZamówień, Telefon. Klient może nie zgodzić się na używanie swoich danych, co zapisywane jest jako 0 w polu CzyRODO. Wtedy użytkownik rozpoznawany jest po unikalnym Nicku. Ponadto istnieje możliwość połączenia tzw. „Modelu biznesowego", czyli połączenia konta klienta indywidualnego do firmy. Zapisuje się wówczas identyfikator danej firmy w polu IDFirmy.

**Warunki integralnościowe:**

- IDKlienta to FK odwołujący się do tabeli Klienci (usuwanie i modyfikacja – kaskadowe)
- IDFirmy to FK odwołujący się do tabeli Firmy
- Wartość domyślna IDFirmy to 0 (oznaczająca brak modelu biznesowego)
- Telefon musi być 9-cyfrowym łańcuchem znaków
- LiczbaZamówień musi być większa lub równa 0
- Nick jest unikatowy

```
CREATE TABLE [dbo].[KlienciIndywidualni](
	[IDKlienta] [int] NOT NULL,
	[CzyRODO] [bit] NOT NULL,
	[Nazwisko] [varchar](255) NULL,
	[Nick] [varchar](255) NULL,
	[LiczbaZamówień] [int] NOT NULL,
	[IDFirmy] [int] NULL,
	[Telefon] [varchar](9) NULL,
PRIMARY KEY CLUSTERED
(
	[IDKlienta] 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],
UNIQUE NONCLUSTERED
(
	[Nick] 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].[KlienciIndywidualni] ADD  DEFAULT ((0)) FOR [IDFirmy]
GO

ALTER TABLE [dbo].[KlienciIndywidualni]  WITH CHECK ADD  CONSTRAINT [fk_czlonekFirmy]
FOREIGN KEY([IDFirmy])
REFERENCES [dbo].[Firmy] ([IDFirmy])
GO

ALTER TABLE [dbo].[KlienciIndywidualni] CHECK CONSTRAINT [fk_czlonekFirmy]
```

```
GO

ALTER TABLE [dbo].[KlienciIndywidualni]  WITH CHECK ADD  CONSTRAINT [fk_IDKlienta]
FOREIGN KEY([IDKlienta])
REFERENCES [dbo].[Klienci] ([IDKlienta])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[KlienciIndywidualni] CHECK CONSTRAINT [fk_IDKlienta]
GO

ALTER TABLE [dbo].[KlienciIndywidualni]  WITH CHECK ADD  CONSTRAINT [chk_telefon]
CHECK  (([Telefon] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[KlienciIndywidualni] CHECK CONSTRAINT [chk_telefon]
GO

ALTER TABLE [dbo].[KlienciIndywidualni]  WITH CHECK ADD CHECK
(([LiczbaZamówień]>=(0)))
GO

ALTER TABLE [dbo].[KlienciIndywidualni]  WITH CHECK ADD CHECK
(([LiczbaZamówień]>=(0)))
GO
```

**ZniżkiKlientówIndywidualnych –** tabela zawierająca informacje o zniżkach danego klienta. Ze względu na złożoność systemu rezerwacji tabela została wydzielona z tabeli KlienciIndywidualni i stanowi niejako jej „przedłużenie”. Zwiera informacje takie jak:
LiczbaZamPowK1 – Liczba zamówień powyżej kwoty K1 określonej w tabeli Stałe.
ŁącznaSuma – kwota wydana przez klienta do tej pory
DataZniżka3 i DataZniżka4 – daty przyznania zniżek 3 i 4.
Rabat1 do Rabat 4 – kwota poszczególnego rabatu przysługująca obecnie danemu klientowi

**Warunki integralnościowe:**

- IDKlienta to FK odwołujący się do tabeli KlienciIndywidualni (usuwanie i modyfikacja – kaskadowe)
- Wartość domyślna LiczZamPowK1 i ŁącznaSuma to 0
- LiczZamPowK1 i ŁącznaSuma muszą mieć wartości większe lub równe 0

```
CREATE TABLE [dbo].[ZniżkiKlientówIndywidualnych](
	[IDKlienta] [int] NOT NULL,
	[LiczZamPowK1] [int] NOT NULL,
	[ŁącznaSuma] [float] NOT NULL,
	[DataZniżka3] [date] NULL,
	[DataZniżka4] [date] NULL,
	[Rabat1] [float] NULL,
	[Rabat2] [float] NULL,
	[Rabat3] [float] NULL,
	[Rabat4] [float] NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych] ADD  DEFAULT ((0)) FOR [LiczZamPowK1]
GO

ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych] ADD  DEFAULT ((0)) FOR [ŁącznaSuma]
GO

ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych]  WITH CHECK ADD  CONSTRAINT
[fk_IDZniżkiKlInd] FOREIGN KEY([IDKlienta])
REFERENCES [dbo].[KlienciIndywidualni] ([IDKlienta])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych] CHECK CONSTRAINT [fk_IDZniżkiKlInd]
GO

ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych]  WITH CHECK ADD CHECK
(([ŁącznaSuma]>=(0)))
GO

ALTER TABLE [dbo].[ZniżkiKlientówIndywidualnych]  WITH CHECK ADD CHECK
(([LiczZamPowK1]>=(0)))
GO
```

**Firmy–** tabela zawierająca informacje o danej firmie, takie jak Nazwa, LiczbaZamówień, Telefon.

**Warunki integralnościowe:**

- IDKlienta to FK odwołujący się do tabeli Klienci (usuwanie i modyfikacja – kaskadowe)
- Telefon musi być 9-cyfrowym łańcuchem znaków
- NIP musi być 10-cyfrowym łańcuchem znaków
- Telefon musi być unikatowy

```
CREATE TABLE [dbo].[Firmy](
	[IDFirmy] [int] NOT NULL,
	[NazwaFirmy] [varchar](255) NOT NULL,
	[Telefon] [varchar](9) NULL,
	[NIP] [varchar](10) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[IDFirmy] 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],
UNIQUE NONCLUSTERED
(
	[Telefon] 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],
UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Firmy]  WITH CHECK ADD  CONSTRAINT [fk_IDFirmy] FOREIGN
KEY([IDFirmy])
REFERENCES [dbo].[Klienci] ([IDKlienta])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Firmy] CHECK CONSTRAINT [fk_IDFirmy]
GO

ALTER TABLE [dbo].[Firmy]  WITH CHECK ADD  CONSTRAINT [chk_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].[Firmy] CHECK CONSTRAINT [chk_NIP]
GO

ALTER TABLE [dbo].[Firmy]  WITH CHECK ADD  CONSTRAINT [chk_telefonFirm] CHECK
(([Telefon] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Firmy] CHECK CONSTRAINT [chk_telefonFirm]
GO
```

**ZniżkiFirm –** tabela zawierająca informacje o zniżkach danego klienta. Ze względu na złożoność systemu rezerwacji tabela została wydzielona z tabeli Firmy i stanowi niejako jej „przedłużenie”. Zwiera informacje takie jak:
LiczbaZamWMiesiącu – Liczba zamówień w tym miesiącu.
ŁącznaKwotaWMiesiącu, ŁącznaKwatoKwartał – Suma pieniędzy wydanych w restauracji odpowiednio w tym miesiącu i w kwartale.
Rabat1 i Rabat2 – wartość rabatu1 i rabatu2. Ważne: rabat1 jest rabatem procentowym, a rabat2 – kwotowym.

**Warunki integralnościowe:**

- IDFirmy to FK odwołujący się do tabeli Firmy (usuwanie i modyfikacja – kaskadowe)
- Wartość domyślna Rabat1 i Rabat2 to 0
- LiczZamWMies, ŁącznaKwotaWMiesiącu, ŁącznaKwotaKwartał, Rabat1 oraz Rabat2 muszą być większe lub równe 0

```
CREATE TABLE [dbo].[ZniżkiFirm](
	[IDFirmy] [int] NOT NULL,
	[LiczZamWMies] [int] NOT NULL,
	[ŁącznaKwotaWMiesiącu] [float] NOT NULL,
	[Rabat1] [float] NOT NULL,
	[ŁącznaKwotaKwartał] [float] NOT NULL,
	[Rabat2] [float] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ZniżkiFirm] ADD  DEFAULT ((0)) FOR [Rabat1]
GO

ALTER TABLE [dbo].[ZniżkiFirm] ADD  DEFAULT ((0)) FOR [Rabat2]
GO
```

```
ALTER TABLE [dbo].[ZniżkiFirm]  WITH CHECK ADD  CONSTRAINT [fk_IDZniżkiFirm] FOREIGN
KEY([IDFirmy])
REFERENCES [dbo].[Firmy] ([IDFirmy])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[ZniżkiFirm] CHECK CONSTRAINT [fk_IDZniżkiFirm]
GO

ALTER TABLE [dbo].[ZniżkiFirm]  WITH CHECK ADD CHECK  (([ŁącznaKwotaWMiesiącu]>=(0)))
GO

ALTER TABLE [dbo].[ZniżkiFirm]  WITH CHECK ADD CHECK  (([LiczZamWMies]>=(0)))
GO

ALTER TABLE [dbo].[ZniżkiFirm]  WITH CHECK ADD CHECK  (([Rabat2]>=(0)))
GO

ALTER TABLE [dbo].[ZniżkiFirm]  WITH CHECK ADD CHECK  (([Rabat1]>=(0) AND
[ŁącznaKwotaWMiesiącu]>=(0) AND [ŁącznaKwotaKwartał]>=(0)))
GO
```

**Stałe** – tabela zawierająca stałe definiowane przez użytkowników bazy. Zawierają przede wszystkim informacje o stałych ważnych przy wyliczaniu rabatu a także maksymalną liczbę miejsc dozwolonych z powodu epidemii COVID.

```
CREATE TABLE [dbo].[Stałe](
	[IDStalej] [int] IDENTITY(1,1) NOT NULL,
	[Nazwa] [varchar](255) NULL,
	[Opis] [varchar](255) NULL,
	[Wartosc] [float] NULL,
PRIMARY KEY CLUSTERED
(
	[IDStalej] 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
```

**Pracownicy** – tabela zawierająca podstawowe informacje o pracownikach, tj. Imię, Nazwisko, Stanowisko, DatePrzyjecia oraz DataObjęciaStanowiska.

**Warunki integralnościowe**:

- Stanowisko przyjmuje wartości ‘Kucharz’, ‘Kelner’, ‘Menedzer’)

```
CREATE TABLE [dbo].[Pracownicy](
	[Identyfikator] [int] IDENTITY(1,1) NOT NULL,
	[Imię] [varchar](255) NOT NULL,
	[Nazwisko] [varchar](255) NOT NULL,
	[Stanowisko] [varchar](255) NOT NULL,
	[DataPrzyjęcia] [date] NOT NULL,
	[DataObjęciaStanowiska] [date] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[Identyfikator] 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].[Pracownicy]  WITH CHECK ADD CHECK  (([Stanowisko]='Kucharz' OR
[Stanowisko]='Kelner' OR [Stanowisko]='Menedzer'))
GO
```

**Kucharze, Kelnerzy, Menadzerowie** – tabele zawierające szczegółowe informacje o pracownikach, którzy wykonują specyficzne zadania i z tego powodu posiadający dodatkowe kolumny oznaczający ich dodatkowe umiejętności. Każda tabela składa się klucza obcego IDPracownika oraz dodatkowych pól (np. PopisoweDanie w tabeli Kucharze)

**WarunkiIntegralnościowe:**

- IDPracownika to FK odwołujący się do tabeli Pracownicy (usuwanie i aktualizacja kaskadowo)

```
CREATE TABLE [dbo].[Kelnerzy](
	[IDPracownika] [int] NOT NULL,
	[Umiejętności] [varchar](255) NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Kelnerzy]  WITH CHECK ADD  CONSTRAINT [fk_IDKelnera] FOREIGN
KEY([IDPracownika])
REFERENCES [dbo].[Pracownicy] ([Identyfikator])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Kelnerzy] CHECK CONSTRAINT [fk_IDKelnera]
GO



CREATE TABLE [dbo].[Kucharze](
	[IDPracownika] [int] NOT NULL,
	[PopisoweDanie] [varchar](255) NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Kucharze]  WITH CHECK ADD  CONSTRAINT [fk_IDKucharza] FOREIGN
KEY([IDPracownika])
REFERENCES [dbo].[Pracownicy] ([Identyfikator])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Kucharze] CHECK CONSTRAINT [fk_IDKucharza]
GO



CREATE TABLE [dbo].[Menedżerowie](
	[IDPracownika] [int] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Menedżerowie]  WITH CHECK ADD  CONSTRAINT [fk_IDMenedżera] FOREIGN
KEY([IDPracownika])
REFERENCES [dbo].[Pracownicy] ([Identyfikator])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Menedżerowie] CHECK CONSTRAINT [fk_IDMenedżera]
GO
```

## Indeksy

SQLServer automatycznie dodaje indeksy klastrowe na kluczach głównych. W celu szybszego wyszukiwania, dodano również indeksy nieklastrowe na niektórych kluczach obcych. Kod do nich przedstawiono poniżej:

```
CREATE NONCLUSTERED INDEX [ix_FKzamówienia]
ON [dbo].[Rezerwacje]([IDZamówienia])

CREATE NONCLUSTERED INDEX [ix_FKklienta]
ON [dbo].[Zamówienia]([IDKlienta])

CREATE NONCLUSTERED INDEX [ix_FKZamówienia]
ON [dbo].[SzczegółyZamówień]([IDZamówienia])

CREATE NONCLUSTERED INDEX [ix_FKPotrawy]
ON [dbo].[SzczegółyZamówień]([IDPotrawy])

CREATE NONCLUSTERED INDEX [ix_FKKategorii]
ON [dbo].[Menu]([IDKategorii])

CREATE NONCLUSTERED INDEX [ix_FKKPotraw]
ON [dbo].[Przepisy]([IDPotrawy])

CREATE NONCLUSTERED INDEX [ix_FKProduktu]
ON [dbo].[Przepisy]([IDProduktu])

CREATE NONCLUSTERED INDEX [ix_FKStolika]
ON [dbo].[Rezerwacje]([IDStolika])

CREATE NONCLUSTERED INDEX [ix_FKFirmy]
ON [dbo].[KlienciIndywidualni]([IDFirmy])
```

## FUNKCJE ZWRACAJĄCE WARTOŚĆ SKALARNĄ

**Czy_danie_dostepne**(
@IDPotrawy int,
@naKiedy DATE,
@ilość INT) – sprawdza, czy można zamówić dane danie

```
CREATE FUNCTION
[dbo].[Czy_danie_dostepne]
(
@IDPotrawy int,
```

```
@naKiedy DATE,
@ilość INT
)
RETURNS bit
AS
BEGIN
	IF(
	(([dbo].[Czy_danie_wykreslone] (@IDPotrawy,@naKiedy) )=0)
	AND
	([dbo].[Czy_sa_skladniki] (@IDPotrawy,@ilość))=1
	)
	RETURN 1
	RETURN 0
END
```

**Czy_danie_wykreslone**

(

@IDPotrawy int,

@naKiedy DATE

) – sprawdza, czy danie zostało wykreślone

```
CREATE FUNCTION
[dbo].[Czy_danie_wykreslone]
(
@IDPotrawy int,
@naKiedy DATE
)
RETURNS bit
AS
BEGIN
	DECLARE @DataWykr DATE=(SELECT DataWykreślenia FROM Menu
		WHERE IDPotrawy=@IDPotrawy
		)
	IF(@DataWykr IS NULL) RETURN 0
	ELSE IF (@DataWykr<=@naKiedy) RETURN 1

	RETURN 0
END
```

**Czy_mozna_dodac_wczesniej_skreslona_potrawe**(

@IDPotrawy int

) – sprawdzenie, czy potrawa nie została wykreślona mniej niż miesiąc wcześniej

```
CREATE FUNCTION [dbo].[Czy_mozna_dodac_wczesniej_skreslona_potrawe](
	@IDPotrawy int
)
RETURNS bit
AS
BEGIN
	DECLARE @DeletionDate datetime = (SELECT DataWykreślenia FROM Menu WHERE
IDPotrawy=@IDPotrawy);
	DECLARE @Result bit = 0

	IF @DeletionDate IS NULL OR	--Danie nie było wykreslone
		DATEADD(MONTH, 1, @DeletionDate) > GETDATE()	--Danie wykreslono ponad
miesiac temu
		SET @Result = 1
```

```
	RETURN @Result
END
```

**Czy_rezerwacja_mozliwa_dla_zamowienia**(
@IDZamówienia int
) – sprawdzenie warunków dokonania rezerwacji

```
CREATE FUNCTION [dbo].[Czy_rezerwacja_mozliwa_dla_zamowienia]
(
	@IDZamowienia int
)
RETURNS bit
AS
BEGIN
	DECLARE @Result bit;
	DECLARE @KlientID int = (SELECT IDKlienta
						FROM Zamówienia
						WHERE IDZamówienia = @IDZamowienia);
	DECLARE @WartoscZamowienia int = [dbo].Wartosc_zamowienia(@IDZamowienia);
	DECLARE @LiczbaZamowien int = 0;

	IF @KlientID IS NOT NULL
		SET @LiczbaZamowien = (SELECT LiczbaZamówień
							FROM KlienciIndywidualni
							WHERE IDKlienta = @KlientID)

	IF (@WartoscZamowienia >= 50 AND @LiczbaZamowien >= 5) OR (@WartoscZamowienia >=
200)
		SET @Result = 1;
	ELSE
		SET @Result = 0;

	RETURN @Result
END
```

**Czy_sa_skladniki**(
@IDPotrawy int,
@ilośćPotrawy int
) – sprawdzenie, czy składniki danej potrawy są dostępne w magazynie

```
CREATE FUNCTION
[dbo].[Czy_sa_skladniki]
(
@IDPotrawy int,
@ilośćPotrawy int
)
RETURNS bit
AS
BEGIN
	IF EXISTS
	(
	SELECT Przepisy.IDProduktu,Przepisy.IlePotrzeba, Magazyn.Ilość
	FROM Przepisy
```

```
        JOIN Magazyn ON Magazyn.IDProduktu=Przepisy.IDProduktu
        WHERE IDPotrawy=@IDPotrawy AND Magazyn.Ilość*@ilośćPotrawy<Przepisy.IlePotrzeba
        )
        RETURN 0

        RETURN 1
END
```

**Firmowa_znizka_kwotowa_na_nowy_kwartal** (
@IDFirmy int
) – obliczenie wartości kwotowego rabatu firm

```
CREATE FUNCTION
[dbo].[Firmowa_znizka_kwotowa_na_nowy_kwartal]
(
@IDFirmy int
--zakladajac ze GETDATE() działa jak chce
--,@poczatekPoprzedniegoKwartalu DATE
)
RETURNS FLOAT
AS
BEGIN
--zakładam że automatycznie to się będzie wykonywać co kwartał
        DECLARE @koniecKwartalu DATE=CONVERT(DATE,GETDATE())
        DECLARE @poczatekKwartalu DATE=DATEADD(month,-3,@koniecKwartalu)
        DECLARE @lacznaKwota FLOAT=
                (SELECT SUM(Cena)
                FROM Zamówienia
                WHERE (IDKlienta=@IDFirmy) AND (DataZłożenia BETWEEN @poczatekKwartalu
AND @koniecKwartalu )
                )
        IF @lacznaKwota>=(SELECT Wartosc FROM Stałe WHERE Nazwa LIKE 'FK2')
                BEGIN
                DECLARE @rabatKwotowy FLOAT=ROUND((SELECT Wartosc FROM Stałe WHERE Nazwa
LIKE 'FR2')*@lacznakwota,2)
                RETURN @rabatKwotowy
                END
        RETURN 0
END
```

**Ilość_osób_w_lokalu**(
@dataOd datetime,
@dataDo datetime,
) – liczba osób w lokalu od danej do danej chwili

```
CREATE FUNCTION
[dbo].[Ilość_osób_w_lokalu]
(@dataOd DATETIME,
@dataDo DATETIME
)
RETURNS INT
AS
BEGIN
        DECLARE @result INT=
                (SELECT SUM(LiczbaOsob)
                FROM Rezerwacje WHERE
                (OD BETWEEN @dataOd AND @dataDo) OR (DO BETWEEN @dataOd AND @dataDo)
                )
```

```
	RETURN @result
END
```

**Następny_poniedziałek_owoce_morza**(
@data datetime
) – Funkcja pomocnicza, do wyznaczania najwcześniejszej daty na którą można zamówić owoce morza

```
CREATE FUNCTION
[dbo].[Następny_poniedziałek_na_owoce_morza]
(@data DATETIME
)
RETURNS DATETIME
AS
BEGIN
	DECLARE @result DATETIME=(SELECT
		CASE
		WHEN DATEPART(WEEKDAY,@data)=1 THEN DATEADD(day,8,@data)--niedziela
		WHEN DATEPART(WEEKDAY,@data)=2 THEN DATEADD(day,7,@data)--pon
		WHEN DATEPART(WEEKDAY,@data)=3 THEN DATEADD(day,6,@data)--wt
		WHEN DATEPART(WEEKDAY,@data)=4 THEN DATEADD(day,5,@data)--śr
		WHEN DATEPART(WEEKDAY,@data)=5 THEN DATEADD(day,4,@data)--czw
		WHEN DATEPART(WEEKDAY,@data)=6 THEN DATEADD(day,3,@data)--pt
		WHEN DATEPART(WEEKDAY,@data)=7 THEN DATEADD(day,2,@data)--sobota
		END
		)
	RETURN @result
END
```

**Oblicz_rabat_miesieczny**(
@IDFirmy int
) – obliczenie wartości rabatu miesięcznego firmy

```
CREATE FUNCTION [dbo].[Oblicz_rabat_miesieczny](
	@IDFirmy int
)
RETURNS float
AS
BEGIN
	DECLARE @LiczbaZamWMies int =
		(SELECT LiczZamWMies
		FROM ZniżkiFirm
		WHERE IDFirmy=@IDFirmy);
	DECLARE @ŁącznaKwotaWMiesiącu float =
		(SELECT ŁącznaKwotaWMiesiącu
		FROM ZniżkiFirm
		WHERE IDFirmy=@IDFirmy);
	DECLARE @Rabat float=
		(SELECT Rabat1
		FROM ZniżkiFirm
		WHERE IDFirmy=@IDFirmy);

	DECLARE @FZ float = dbo.Wartosc_stalej('FZ');
	DECLARE @FK1 float = dbo.Wartosc_stalej('FK1');
	DECLARE @FM float = dbo.Wartosc_stalej('FM');
	DECLARE @FR1 float = dbo.Wartosc_stalej('FR1');
```

```
	IF @LiczbaZamWMies >= @FZ AND
	   @ŁącznaKwotaWMiesiącu >= @FK1
		IF(@Rabat + @FR1 < @FM)
			SET @Rabat = @Rabat + @FR1;
		ELSE
			SET @Rabat = @FM;
	ELSE
			SET @Rabat = 0;

	RETURN @Rabat
END
```

**Policz_cene_z_rabatem**(

@IDKlienta int,

@cenaBezRabatu float

) – policzenie ceny z uwzględnieniem rabatu

```
CREATE FUNCTION
[dbo].[Policz_cene_z_rabatem]
(
@IDKlienta int,
@cenaBezRabatu float
)
RETURNS FLOAT
AS
BEGIN
	--zamowienie dla losowego klienta nieebędącego w naszej bazie
	IF (@IDKlienta IS NULL)
			BEGIN
			RETURN @cenaBezRabatu
			END
	--zamowienie dla firmy
	IF ((SELECT Typ FROM Klienci WHERE IDKlienta=@IDKlienta) LIKE 'Firma')
			BEGIN
				DECLARE @cenaOstateczna float=@cenaBezRabatu
				DECLARE @znizkaKwotowa float=(SELECT Rabat2 FROM ZniżkiFirm WHERE
IDFirmy=@IDKlienta)
				IF (@znizkaKwotowa>0)
					BEGIN
						IF (@cenaBezRabatu-@znizkaKwotowa)>=0
							SET @cenaOstateczna= (@cenaBezRabatu-
@znizkaKwotowa)
						ELSE
							RETURN 0
					END
				DECLARE @znizkaProcentowa float=(SELECT Rabat1 FROM ZniżkiFirm
WHERE IDFirmy=@IDKlienta)
				RETURN ROUND((1.0-@znizkaProcentowa)*(@cenaOstateczna),2)
			END
	--klient to nie firma więc mamy do czynienia z klientem indywidualnym
	DECLARE @znizkaProcentowaKlientaIndyw float=((SELECT
(Rabat1+Rabat2+Rabat3+Rabat4) FROM ZniżkiKlientówIndywidualnych WHERE
IDKlienta=@IDKlienta))
	RETURN ROUND((1.0-@znizkaProcentowaKlientaIndyw)*(@cenaBezRabatu),2)
END
```

**Wartosc_stalej**(
@NazwaStałej varchar(255)
) – zwraca wartość stałej o danej nazwie

```
CREATE FUNCTION [dbo].[Wartosc_stalej](
	@NazwaStalej varchar(255)
)
RETURNS float
AS
BEGIN
	RETURN(
	SELECT Wartosc
	FROM Stałe
	WHERE Nazwa=@NazwaStalej
)
END
```

**Wartosc_zamowienia**(
@IDZamowienia int
) – zwraca wartość zamówienia (czyli cenę bez uwzględnienia rabatu)

```
CREATE FUNCTION [dbo].[Wartosc_zamowienia]
(
	@IDZamowienia int
)
RETURNS float
AS
BEGIN
	RETURN (SELECT SUM(CenaJednostkowa*Ilość)
			FROM SzczegółyZamówień
			WHERE IDZamówienia = @IDZamowienia)
END
```

**Wartosc_Znizka_1**(
@IDKlienta int
) – wartość zniżki nr 1 dla klienta indywidualnego

```
CREATE FUNCTION [dbo].[Wartosc_Znizka_1]
(
	@IDKlienta int
)
RETURNS float
AS
BEGIN
	DECLARE @PowK1 int = (SELECT LiczZamPowK1 FROM ZniżkiKlientówIndywidualnych
WHERE IDKlienta = @IDKlienta)
	DECLARE @Z1 int = (SELECT Wartosc FROM Stałe WHERE Nazwa = 'Z1')
	DECLARE @Result int = 0;

	IF @PowK1 >= @Z1
		SET @Result = (SELECT Wartosc FROM Stałe WHERE Nazwa = 'R1')
	RETURN @Result
END
```

**Wartosc_Znizka_2**(
@IDKlienta int

) – wartość zniżki nr 2 dla klienta indywidualnego

```
CREATE FUNCTION [dbo].[Wartosc_Znizka_2]
(
	@IDKlienta int
)
RETURNS float
AS
BEGIN
	DECLARE @PowK1 int = (SELECT LiczZamPowK1 FROM ZniżkiKlientówIndywidualnych
WHERE IDKlienta = @IDKlienta)
	DECLARE @Z1 float = 2*(SELECT Wartosc FROM Stałe WHERE Nazwa = 'Z1')
	DECLARE @Result float = 0;

	IF @PowK1 >= @Z1
		SET @Result = (SELECT Wartosc FROM Stałe WHERE Nazwa = 'R1')
	RETURN @Result
END
```

**Wartosc_Znizka_3**(
@IDKlienta int
) – wartość zniżki nr 3 dla klienta indywidualnego

```
CREATE FUNCTION [dbo].[Wartosc_Znizka_3]
(
	@IDKlienta int
)
RETURNS float
AS
BEGIN
	DECLARE @LacznaKwota int = (SELECT łącznaSuma FROM ZniżkiKlientówIndywidualnych
WHERE IDKlienta=@IDKlienta)
	DECLARE @K2 float = dbo.Wartosc_stalej('K2');
	DECLARE @DataOtrzymaniaZnizki date = (SELECT DataZniżka3 FROM
ZniżkiKlientówIndywidualnych WHERE IDKlienta=@IDKlienta);
	DECLARE @D1 float = dbo.Wartosc_stalej('D1');
	DECLARE @Result float = 0;

	IF @LacznaKwota >= @K2 AND
	DATEADD(day, @D1, @DataOtrzymaniaZnizki) <= CONVERT(DATE, GETDATE()) AND
	@DataOtrzymaniaZnizki != CONVERT(DATE, GETDATE())
		SET @Result =  dbo.Wartosc_stalej('R2');
	RETURN @Result
END
```

**Wartosc_Znizka_4**(
@IDKlienta int
) – wartość zniżki nr 4 dla klienta indywidualnego

```
CREATE FUNCTION [dbo].[Wartosc_Znizka_4]
(
	@IDKlienta int
)
RETURNS float
AS
BEGIN
	DECLARE @LacznaKwota int = (SELECT łącznaSuma FROM ZniżkiKlientówIndywidualnych
WHERE IDKlienta=@IDKlienta)
	DECLARE @K3 float = dbo.Wartosc_stalej('K3');
```

```
	DECLARE @DataOtrzymaniaZnizki date = (SELECT DataZniżka4 FROM
ZniżkiKlientówIndywidualnych WHERE IDKlienta=@IDKlienta);
	DECLARE @D2 float = dbo.Wartosc_stalej('D2');
	DECLARE @Result float = 0;

	IF @LacznaKwota >= @K3 AND
	DATEADD(day, @D2, @DataOtrzymaniaZnizki) <= CONVERT(DATE, GETDATE()) AND
	@DataOtrzymaniaZnizki != CONVERT(DATE, GETDATE())
		SET @Result =  dbo.Wartosc_stalej('R3');
	RETURN @Result
END
```

**Zwroc_wolny_stolik**(
@StolikOd datetime,
@StolikDo datetime,
@ileMiejsc datetime
) – Funkcja pomocnicza zwracająca ID najmniejszego stolika, który może zarezerwować dana liczba osób

```
CREATE FUNCTION [dbo].[Zwroc_wolny_stolik]
(
	@StolikOd datetime,
	@StolikDo datetime,
	@IleMiejsc int
)
RETURNS int
AS
BEGIN
	RETURN(SELECT TOP 1 IDStolika
		FROM Stoliki
		WHERE LiczbaMiejsc>=@IleMiejsc
		AND IDStolika NOT IN
			(SELECT IDStolika
			From REZERWACJE
			WHERE ((@StolikOd BETWEEN OD and [DO])
			OR (@StolikDo BETWEEN OD AND [DO]))
			)
		ORDER BY LiczbaMiejsc ASC
		)
END
```

**Tygodniowe_dochody**(
	@dataOd datetime
) – funkcja zwracająca dochód firmy z 7 dni przed podaną datą

```
CREATE FUNCTION
Tygodniowe_dochody
(
@dataOd DATETIME
)
RETURNS TABLE
AS
RETURN
(
	SELECT SUM(DoZapłaty) AS Suma FROM Zamówienia
	WHERE DATEDIFF(DAY,@dataOd,NaKiedy)<=7 AND DATEDIFF(DAY,@dataOd,NaKiedy)>=0
```

```
)
```

**Miesieczne_dochody**(
	@dataOd Datetime
) - funkcja zwracająca dochód firmy z 7 dni przed podaną datą

```
CREATE FUNCTION
Miesieczne_dochody
(
@dataOd DATETIME
)
RETURNS TABLE
AS
RETURN
(
	SELECT SUM(DoZapłaty) AS Suma FROM Zamówienia
	WHERE DATEDIFF(MONTH,@dataOd,NaKiedy)=0
)
```

## PROCEDURY

**Dodaj_danie -** dodaje danie do bazy danych

```
CREATE PROCEDURE [dbo].[Dodaj_danie]
(
@nazwa VARCHAR(255),
@opis VARCHAR(255),
@cena FLOAT,
@kategoria VARCHAR(255),
@odKiedy DATE
)
AS
BEGIN
	DECLARE @IDKategorii int = (SELECT IDKategorii FROM KategorieMenu WHERE
NazwaKategorii LIKE @kategoria)
	INSERT INTO Menu VALUES
(@opis,@cena,@IDKategorii,1,@odKiedy,null,@nazwa,GETDATE())
END
```

**Dodaj_do_przepisu -** procedura w domyśle używana wielokrotnie, do danego przepisu dodajemy składnik i jego ilość

```
CREATE PROCEDURE [dbo].[Dodaj_do_przepisu]
(
@nazwaPotrawy varchar(255),
@nazwaProduktu varchar(255),
@ilosc int
)
AS
BEGIN
	DECLARE @IDPotrawy int = (SELECT IDPotrawy FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	DECLARE @IDProduktu int =(SELECT IDProduktu FROM Magazyn WHERE NazwaProduktu
LIKE @nazwaProduktu)
	IF (@IDPotrawy IS NOT NULL AND @IDProduktu IS NOT NULL AND @ilosc>=0)
		INSERT INTO Przepisy VALUES(@IDPotrawy,@IDProduktu,@ilosc)
END
```

**Dodaj_do_zamówienia** - procedura działająca podobnie do dodaj_do_przepisu, po wcześniejszym rozpoczęciu zamówienia, możemy w kilku wywołaniach je skompletować
w procedurze sprawdzane jest czy danie jest dostępne, czy jeśli są to owoce morza to czy data zamówienia jest odpowiednia oraz jeśli klient jest firmą, to odejmowana jest zniżka kwotowa z tablicy ZniżkiFirm

```
CREATE PROCEDURE
[dbo].[Dodaj_do_zamówienia]
(@IDZamówienia INT,
@nazwaPotrawy VARCHAR(255),
@ilość INT
)
AS
BEGIN
	--sprawdzanie poprawnosci
	IF(@ilość<=0) RETURN
	IF NOT EXISTS(SELECT Nazwa FROM Menu WHERE Nazwa LIKE @nazwaPotrawy) RETURN

	DECLARE @IDKategorii INT= (SELECT IDKategorii FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	DECLARE @kategoriaPotrawy VARCHAR=(SELECT NazwaKategorii FROM KategorieMenu
WHERE IDKategorii=@IDKategorii)
	DECLARE @naKiedy DATETIME=(SELECT NaKiedy FROM Zamówienia WHERE
IDZamówienia=@IDZamówienia)
	DECLARE @dataZłożenia DATETIME=(SELECT DataZłożenia FROM Zamówienia WHERE
IDZamówienia=@IDZamówienia)
	DECLARE @IDKlienta INT=(SELECT IDKlienta FROM Zamówienia WHERE
IDZamówienia=@IDzamówienia)

	--owoce morza
	IF(@kategoriaPotrawy LIKE 'owoce morza')
		BEGIN

	IF([dbo].[Następny_poniedziałek_na_owoce_morza](@dataZłożenia)>@naKiedy)
				BEGIN
					PRINT('NIE MOŻNA DODAĆ OWOCÓW MORZA, zamówienie jest
za wcześnie')
					RETURN
				END
		END
	DECLARE @IDPotrawy INT=(SELECT IDPotrawy FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	DECLARE @cenaPotrawy FLOAT=(SELECT Cena FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	--czy danie jest dostepne
	IF(([dbo].[Czy_danie_dostepne](@IDPotrawy,@naKiedy,@ilość))=0)
	BEGIN
		PRINT('Danie niedostepne')
		RETURN
	END

	--wstaw do szczegolow
	INSERT INTO SzczegółyZamówień
VALUES(@IDZamówienia,@IDPotrawy,@ilość,@cenaPotrawy)
	--po wstawieniu aktualizacja lacznej ceny za zamowienie
	DECLARE @cenaBezRabatu FLOAT=@ilość*@cenaPotrawy
	DECLARE @cenaZRabatem
FLOAT=[dbo].[Policz_cene_z_rabatem](@IDKlienta,@cenaBezRabatu)
	BEGIN
		UPDATE Zamówienia
		SET Cena=Cena+@cenaBezRabatu,Zniżka=Zniżka+(@cenaBezRabatu-
@cenaZRabatem),DoZapłaty=DoZapłaty+@cenaZRabatem
```

```
		WHERE IDZamówienia=@IDZamówienia
	END
	--jesli klient to firma to odejmujemy znizke kwotowa
	IF((SELECT Typ FROM Klienci WHERE IDKlienta=@IDKlienta) LIKE 'Firma')
	BEGIN
		DECLARE @rabatKwotowyFirmy FLOAT=(SELECT Rabat2 FROM ZniżkiFirm WHERE
IDFirmy=@IDKlienta)
		IF(@rabatKwotowyFirmy>0)
		BEGIN
			DECLARE @doUsuniecia FLOAT=@cenaPotrawy*@ilość
			IF(@cenaPotrawy*@ilość>=@rabatKwotowyFirmy)
				--czyli skonczyla sie znizka kwotowa
				SET @doUsuniecia=@rabatKwotowyFirmy
			BEGIN
				UPDATE ZniżkiFirm
				SET Rabat2=Rabat2-@doUsuniecia
				WHERE IDFirmy=@IDKlienta
			END
		END
	END
	--odejmujemy skladniki z magazynu
	EXEC Zaktualizuj_skladniki @IDPotrawy=@IDPotrawy,@ilośćPotrawy=@ilość
	RETURN
END
```

**Dodaj_firme -** dodaje firmę do bazy danych

```
CREATE PROCEDURE [dbo].[Dodaj_firme]
	@NazwaFirmy varchar(255),
	@Telefon varchar(9),
	@NIP varchar(10)
AS
BEGIN
SET NOCOUNT ON;
	BEGIN TRY
		INSERT INTO Klienci(Typ)
		VALUES('Firma');
		DECLARE @IDFirmy int = SCOPE_IDENTITY();

		INSERT INTO Firmy(
			IDFirmy, NazwaFirmy,
			Telefon, NIP)
		VALUES (
		@IDFirmy,
		@NazwaFirmy,
		@Telefon,
		@NIP
		)
		INSERT INTO ZniżkiFirm(
			IDFirmy, LiczZamWMies, łącznaKwotaWMiesiącu,
			Rabat1, łącznaKwotaKwartał, Rabat2)
		VALUES(@IDFirmy, 0, 0.0, 0.0, 0.0, 0.0);
	END TRY
	BEGIN CATCH
		DELETE FROM Klienci WHERE IDKlienta=@IDFirmy

		SELECT ERROR_MESSAGE() AS ErrorMessage;
	END CATCH
END;
```

**Dodaj_klienta_indywidualnego -** dodaje klienta indywidualnego do bazy danych

```
CREATE PROCEDURE [dbo].[Dodaj_klienta_indywidualnego]
	@CzyRODO bit,
	@NazwiskoLubNick varchar(255),
```

```
	@NazwaFirmy varchar(255),
	@Telefon varchar(9)
AS
BEGIN
SET NOCOUNT ON;
	BEGIN TRY
		DECLARE @IDKlienta int;

		INSERT INTO Klienci(Typ)
		VALUES('Klient Indywidualny');
		SET @IDKlienta = SCOPE_IDENTITY();

		DECLARE @IDFirmy int = (SELECT IDFirmy FROM Firmy WHERE NazwaFirmy =
@NazwaFirmy)

		IF @CzyRODO=1
			BEGIN
				DECLARE @GeneratedNick varchar(10) = LEFT(@NazwiskoLubNick,
5);
				DECLARE @Acc int = 1;
				WHILE (@GeneratedNick IN (SELECT Nick FROM
KlienciIndywidualni))
				BEGIN
					SET @GeneratedNick = @GeneratedNick +
CONVERT(varchar(6), @Acc);
					SET @Acc = @Acc + 1;
				END

			INSERT INTO KlienciIndywidualni(
				IDKlienta, CzyRODO, Nazwisko, Nick,
				LiczbaZamówień, IDFirmy, Telefon)
			VALUES (
			@IDKlienta,
			@CzyRODO,
			@NazwiskoLubNick,
			@GeneratedNick,
			0,
			@IDFirmy,
			@Telefon
			);
			END
		ELSE
			INSERT INTO KlienciIndywidualni(
				IDKlienta, CzyRODO, Nick,
				LiczbaZamówień, IDFirmy, Telefon)
			VALUES (
			@IDKlienta,
			@CzyRODO,
			@NazwiskoLubNick,
			0,
			@IDFirmy,
			@Telefon
			);

		INSERT INTO ZniżkiKlientówIndywidualnych(
			IDKlienta, LiczZamPowK1, ŁącznaSuma,
			DataZniżka3, DataZniżka4,
			Rabat1, Rabat2, Rabat3, Rabat4)
		VALUES(
		@IDKlienta, 0, 0.0,
		NULL, NULL,                         --Data przyznania rabatu
		0.0, 0.0, 0.0, 0.0); --Rabat
```

```
	END TRY
	BEGIN CATCH
		IF ERROR_NUMBER() = 2627	-- złamany Unique_value constraint => trzeba
wyrzucić śmieci z tabeli Klienci
			DELETE FROM Klienci WHERE IDKlienta=@IDKlienta

		SELECT ERROR_MESSAGE() AS ErrorMessage;
	END CATCH
END;
```

**Dodaj_pracownika** - dodaje pracownika do bazy danych

```
CREATE PROCEDURE [dbo].[Dodaj_pracownika] @Imie varchar(255),
	@Nazwisko varchar(255),
	@Stanowisko varchar(255),
	@DataPrzyjecia date,
	@DataObjeciaStanowiska date
AS
BEGIN
	BEGIN TRY
		INSERT INTO Pracownicy (Imię, Nazwisko, Stanowisko, DataPrzyjęcia,
DataObjęciaStanowiska)
		VALUES (@Imie, @Nazwisko, @Stanowisko, @DataPrzyjecia,
@DataObjeciaStanowiska);

		DECLARE @Key int;
		SET @Key = SCOPE_IDENTITY() --zwraca ostatnio inkrementowaną wartość (w
naszym przypadku klucz)

		IF @Stanowisko = 'Kelner'
		BEGIN
			INSERT INTO Kelnerzy (IDPracownika)
			VALUES (@Key);
		END
		IF @Stanowisko = 'Kucharz'
		BEGIN
			INSERT INTO Kucharze (IDPracownika)
			VALUES (@Key);
		END
		IF @Stanowisko = 'Menedzer'
		BEGIN
			INSERT INTO Menedżerowie(IDPracownika)
			VALUES (@Key);
		END
	END TRY
	BEGIN CATCH
		SELECT ERROR_MESSAGE() AS ErroMessage;
	END CATCH
END;
```

**Dodaj_produkt** - dodajemy produkt który będzie w magazynie, musimy podać jednostkę w jakiej trzymamy dany produkt

```
CREATE PROCEDURE [dbo].[Dodaj_produkt]
(
@nazwaProduktu varchar(255),
@ilosc float,
@jednostka varchar(255)
)
AS
BEGIN
	IF @ilosc<0
	RETURN
```

```
	IF NOT EXISTS (SELECT * FROM Magazyn WHERE NazwaProduktu LIKE @nazwaProduktu)
		BEGIN
			INSERT INTO Magazyn Values(@nazwaProduktu,@ilosc,@jednostka)
		END

END
```

**Dodaj_rezerwacje_dla_firmy** - dodaje rezerwację na daną firmę, parametrem wejściowym moze być null co oznacza wszystkich pracowników firmy

```
CREATE PROCEDURE [dbo].[Dodaj_rezerwacje_dla_firmy]
--@NazwaFirmy = 'Los Pallos Hormonos', @LiczbaOsób = 4, @OD = '2021-01-22 17:00', @DO
= '2021-01-22 19:00'
	@NazwaFirmy varchar(255),
	@LiczbaOsób int,
	@OD datetime,
	@DO datetime
AS
BEGIN
	BEGIN TRY
		SET NOCOUNT ON

		IF (@LiczbaOsób IS NULL )   -- jeżeli liczba osób nie została
uspecyfikowana, to przyjmujemy, że wszyscy pracownicy mają rezerwację
		BEGIN
			IF (@NazwaFirmy IN (SELECT TOP 1 NazwaFirmy FROM Firmy))
			BEGIN
				SET @LiczbaOsób = (SELECT COUNT(*) FROM KlienciIndywidualni
									WHERE IDFirmy=
										(SELECT IDFirmy FROM
Firmy
										WHERE
NazwaFirmy=@NazwaFirmy))
			END

			ELSE
			BEGIN
				SET @LiczbaOsób=0
			END
		END

		WHILE (@LiczbaOsób > 0)
		BEGIN
			DECLARE @Tmp int = @LiczbaOsób
			DECLARE @IDStolika int = NULL
			WHILE (@IDStolika IS NULL AND @Tmp > 0)
			BEGIN
				SET @IDStolika = [dbo].Zwroc_wolny_stolik(@OD, @DO, @Tmp)
				SET @Tmp = @Tmp - 1
			END
			IF (@IDStolika IS NOT NULL)
			BEGIN
				DECLARE @LiczbaMiejsc int = (SELECT LiczbaMiejsc FROM
Stoliki WHERE IDStolika=@IDStolika)

				INSERT INTO Rezerwacje(IDZamówienia, IDStolika, Nazwisko,
LiczbaOsob, CzyZapłataNaMijescu, OD, DO)
				VALUES (NULL, @IDStolika, @NazwaFirmy, @LiczbaOsób, 1, @OD,
@DO)

				SET @LiczbaOsób = @LiczbaOsób - @LiczbaMiejsc
			END
			ELSE
				PRINT 'Brak miejsc'
```

```
                        BREAK
                END
        END TRY

        BEGIN CATCH
                SELECT ERROR_MESSAGE() AS ErrorMessage;
        END CATCH
END
```

**Dodaj_skladniki_z_powrotem** - procedura wywoływana podczas usuwania zamówienia, składniki przestają być "zarezerwowane" dla danego dania

```
CREATE PROCEDURE
[dbo].[Dodaj_skladniki_z_powrotem]
(@IDPotrawy INT,
@ilośćPotrawy INT
)
AS
BEGIN
        DECLARE @skladnikiPotrzebne TABLE(ID INT, IDProduktu INT,IlePotrzeba INT)
        INSERT INTO @skladnikiPotrzebne (ID,IDProduktu,IlePotrzeba)
        (SELECT ROW_NUMBER() OVER(ORDER BY IDProduktu), IDProduktu, IlePotrzeba
        FROM Przepisy
        WHERE IDPotrawy=@IDPotrawy)

        DECLARE @wskaznik INT =(SELECT MIN(ID) FROM @skladnikiPotrzebne)
        DECLARE @maksInt INT=(SELECT MAX(id) FROM @skladnikiPotrzebne)

        WHILE(@wskaznik<=@maksInt)
        BEGIN
                DECLARE @ilePotrzeba INT=(SELECT IlePotrzeba FROM @skladnikiPotrzebne
WHERE ID=@wskaznik)
                DECLARE @idProduktu INT=(SELECT IDProduktu FROM @skladnikiPotrzebne WHERE
ID=@wskaznik)
                BEGIN
                        UPDATE Magazyn
                        SET Ilość=Ilość+@ilePotrzeba*@ilośćPotrawy
                        WHERE IDProduktu=@idProduktu
                END
                SET @wskaznik=@wskaznik+1
        END

END
```

**Dodaj_stala** - dodaje stałą do bazy danych

```
CREATE PROCEDURE [dbo].[Dodaj_stala]
        @Nazwa varchar(255),
        @Opis varchar(255),
        @Wartosc float
AS
BEGIN
        BEGIN TRY
                INSERT INTO Stałe (Nazwa, Opis, Wartosc)
                VALUES (@Nazwa, @Opis, @Wartosc)
        END TRY
        BEGIN CATCH
                SELECT ERROR_MESSAGE() AS ErrorMessage;
        END CATCH
END
```

**Dodaj_stolik** - dodaje stolik

```
CREATE PROCEDURE [dbo].[Dodaj_stolik] @LiczbaMiejsc int
AS
BEGIN
        SET NOCOUNT ON;
```

```
	INSERT INTO Stoliki (LiczbaMiejsc)
	VALUES (@LiczbaMiejsc);
END;
```

**Dokonaj_rezerwacji -** dodaje rezerwację na danego klienta, podczas procedury sprawdzane jest czy nie przekroczymy norm związanych z COVID i czy klient się kwalifikuje na rezerwację

```
CREATE PROCEDURE
[dbo].[Dokonaj_rezerwacji]
(
@IDZamówienia INT,
@nazwisko VARCHAR(255),
@liczbaOsob INT,
@czyZaplataNaMiejscu BIT,
@rezerwacjaOd DATETIME,
@rezerwacjaDo DATETIME

)
AS
BEGIN
	SET NOCOUNT ON
	--czy Klient kwalifikuje się na rezerwacje
	IF(([dbo].[Czy_rezerwacja_mozliwa_dla_zamowienia](@IDZamówienia))=0)
		BEGIN
			PRINT 'Zamówienie nie kwalifikuje się na rezerwację'
			RETURN
		END
	--czy jest wolny stolik
	DECLARE @IDKlienta INT=(SELECT IDKlienta FROM Zamówienia WHERE
IDZamówienia=@IDZamówienia)
	DECLARE @IDStolika
INT=([dbo].[Zwroc_wolny_stolik](@rezerwacjaOd,@rezerwacjaDo,@liczbaOsob))
	IF (@IDStolika IS NULL)
		BEGIN
			Print('Brak wolnych miejsc')
			RETURN
		END
	--ograniczenia covid
	DECLARE @rezerwacjeWTymCzasie TABLE(dataTab DATETIME)
	INSERT INTO @rezerwacjeWTymCzasie (dataTab)
		(SELECT OD FROM Rezerwacje
		UNION
		SELECT DO FROM Rezerwacje)

	DECLARE @maksOsob INT=(SELECT Wartosc FROM Stałe WHERE Nazwa Like
'Liczba_osób_w_lokalu')

	DECLARE @rezerwacjeWTymCzasie2 TABLE (id INT,datarez DATETIME)
	INSERT INTO @rezerwacjeWTymCzasie2 (id,datarez)
		(SELECT ROW_NUMBER() OVER(ORDER BY datatab), datatab FROM
@rezerwacjeWTymCzasie
		WHERE dataTab BETWEEN @rezerwacjaOd AND @rezerwacjaDo)

	DECLARE @wskaznik INT =(SELECT MIN(id) FROM @rezerwacjeWTymCzasie2)
	DECLARE @maksInt INT=(SELECT MAX(id) FROM @rezerwacjeWTymCzasie2)

	WHILE (@wskaznik<=@maksInt)
	BEGIN
		DECLARE @godzinaDanejRezerwacji DATETIME=(SELECT datarez FROM
@rezerwacjeWTymCzasie2 WHERE id=@wskaznik)

	IF(([dbo].Ilość_osób_w_lokalu(@godzinaDanejRezerwacji,@godzinaDanejRezerwacji))>
@maksOsob)
			BEGIN
```

```
                        PRINT 'Nie można dokonać rezerwacji, ograniczenia COVID'
                        RETURN
                END
            SET @wskaznik=@wskaznik+1
        END


        INSERT INTO Rezerwacje(IDZamówienia, IDStolika, Nazwisko, LiczbaOsob,
CzyZapłataNaMijescu, OD, DO)
        VALUES(@IDZamówienia,@IDStolika,@nazwisko,@liczbaOsob,@czyZaplataNaMiejscu,@reze
rwacjaOd,@rezerwacjaDo)
END
```

**Modyfikuj_firme_klienta** - procedura zmieniająca firmę klienta

```
CREATE PROCEDURE [dbo].[Modyfikuj_firme_klienta]
        @IDKlienta int,
        @IDFirmy int
AS
BEGIN
SET NOCOUNT ON;
        BEGIN TRY
                UPDATE KlienciIndywidualni
                SET IDFirmy=@IDFirmy
                WHERE IDKlienta=@IDKlienta
        END TRY
        BEGIN CATCH
                SELECT ERROR_MESSAGE() AS ErrorMessage;
        END CATCH
END;
```

**Modyfikuj_rabat_firmy** - liczy i ustawia zniżki dla firmy

```
CREATE PROCEDURE [dbo].[Modyfikuj_rabat_firmy]  --procedura odpalana pierwszego dnia
nowego miesiąca
        @IDFirmy int
AS
BEGIN
        DECLARE @Dzisiaj date = CONVERT(date, GETDATE());
        DECLARE @WartoscRabat1 float = [dbo].[Oblicz_rabat_miesieczny](@IDFirmy);
        UPDATE ZniżkiFirm SET LiczZamWMies = 0 WHERE IDFirmy=@IDFirmy
        UPDATE ZniżkiFirm SET ŁącznaKwotaWMiesiącu = 0 WHERE IDFirmy=@IDFirmy
        UPDATE ZniżkiFirm SET Rabat1 = @WartoscRabat1 WHERE IDFirmy=@IDFirmy

        IF MONTH(@Dzisiaj) IN (1,4,7,10)  --czyli mamy nowy kwartał (taki zapis działa,
sprawdzałem)
        BEGIN
                DECLARE @WartoscRabat2 float =
[dbo].[Firmowa_znizka_kwotowa_na_nowy_kwartal](@IDFirmy)
                UPDATE ZniżkiFirm SET ŁącznaKwotaKwartał = 0 WHERE IDFirmy=@IDFirmy
                UPDATE ZniżkiFirm SET Rabat2 = @WartoscRabat2 WHERE IDFirmy=@IDFirmy
        END
END
```

**Modyfikuj_rabat_wszystkich_firm** - procedura wywoływana co miesiąc, wywołuje „Modyfikuj_rabat_firmy” dla każdej firmy w bazie

```
CREATE PROCEDURE [dbo].[Modyfikuj_rabat_wszystkich_firm]  --opakowanie dla
[modyfikuj_rabat_firmy]
AS
BEGIN
        DECLARE @IDFirmy int = 0

        WHILE (1=1)
        BEGIN
```

```
                SELECT TOP 1 @IDFirmy = IDFirmy
                FROM ZniżkiFirm
                WHERE IDFirmy > @IDFirmy
                ORDER BY IDFirmy

        IF @@ROWCOUNT = 0 BREAK;

        EXEC dbo.Modyfikuj_rabat_firmy @IDFirmy = @IDFirmy
        END
END
```

**UWAGA** – powyższa procedura korzysta z tzw. *scheduled job* i jest uruchamiana na początku każdego nowego miesiąca. Kod znajduje się poniżej:

```
USE msdb ;
GO

EXEC sp_add_schedule
    @schedule_name = 'ObliczanieNowegoRabatu' ,
    @freq_type = 16,                    --uruchamiane co miesiąc
    @freq_interval = 1,                         --pierwszy dzień
    @active_start_time = 000100 ; --uruchamiamy minutę po północy
GO

EXEC sp_attach_schedule
   @job_name = 'dbo.Modyfikuj_rabat_wszystkich_firm',
   @schedule_name = 'ObliczanieNowegoRabatu';
GO
```

**Modyfikuj_stala –** modyfikuje wartość stałej

```
CREATE PROCEDURE [dbo].[Modyfikuj_stala]
        @Nazwa varchar(255),
        @Wartosc float
AS
BEGIN
        BEGIN TRY
                UPDATE Stałe
                SET Wartosc=@Wartosc
                WHERE Nazwa=@Nazwa
        END TRY
        BEGIN CATCH
                SELECT ERROR_MESSAGE() AS ErrorMessage;
        END CATCH
END
```

**Powiadom_o_zmianie_karty -** procedura wywoływana przez

```
CREATE PROCEDURE [dbo].[Powiadom_o_zmianie_karty]
AS
BEGIN
        DECLARE @LiczbaDniWKarcie int = 14;
        DECLARE @ZaIleDni int = 7;

        DECLARE @LiczbaProduktówWKarcie int = (SELECT COUNT(*) FROM Dzisiejsze_menu)
        DECLARE @LiczbaStarychProduktów int = (SELECT COUNT(*) FROM
[dbo].Dania_starsze_niz_x_dni(@LiczbaDniWKarcie - @ZaIleDni))

        IF @LiczbaProduktówWKarcie < 2*@LiczbaStarychProduktów
                PRINT 'UWAGA Liczba produktów znajdujacych sie w menu od 14 dni
przekroczy połowę wszystkich obecnych produktów za 7 dni'

END
```

**Uwaga.** Powyższa funkcja korzysta z tzw. „scheduled jobs”. Oznacza to, że powinna się wykonywać regularnie (w tym wypadku – codziennie o godzinie 8:00). Kod znajduje się poniżej

```
USE msdb ;
GO

EXEC sp_add_schedule
    @schedule_name = 'PowiadomienieOKoniecznosciZmianyMenu' ,
    @freq_type = 4,                                  --uruchamiane codziennie
    @active_start_time = 080000 ;        --uruchamiamy minutę po północy
GO

EXEC sp_attach_schedule
   @job_name = 'dbo.Powiadom_o_zmianie_karty',
   @schedule_name = 'PowiadomienieOKoniecznosciZmianyMenu';
GO
```

**Przyjmij_dostawe-** procedura przyjmująca nazwę produktu i jego ilość

```
CREATE PROCEDURE [dbo].[Przyjmij_dostawe]
(
@nazwaProduktu varchar(255),
@ilosc float
)
AS
BEGIN
       IF @ilosc<0
       RETURN
       --check na istnienie nie zaszkodzi
       IF EXISTS (SELECT * FROM Magazyn WHERE NazwaProduktu LIKE @nazwaProduktu)
              BEGIN
                     DECLARE @staraIlosc float=(SELECT Ilość FROM Magazyn WHERE
NazwaProduktu LIKE @nazwaProduktu)
                     UPDATE Magazyn SET Ilość=@ilosc+@staraIlosc WHERE NazwaProduktu
LIKE @nazwaProduktu
              END

END
```

**Przywroc_danie-**procedura sprawiająca że danie nie jest już wykreślone

```
CREATE PROCEDURE [dbo].[Przywroc_Danie]
       @nazwaPotrawy varchar(255),
       @odKiedy DATE
AS
BEGIN
       UPDATE Menu
       SET OdKiedy=@odKiedy
       WHERE Nazwa LIKE @nazwaPotrawy
END
```

**Rozpocznij_zamówienie -** procedura rozpoczynająca puste zamówienie, w założeniu powinno się po niej wykonać procedurę **Dodaj_do_zamówienia**

```
CREATE PROCEDURE
[dbo].[Rozpocznij_zamówienie]
(
@IDKlienta INT,
@naKiedy DATETIME,
@czyNaMiejscu BIT,
@dataZłożenia DATETIME
)
AS
BEGIN
```

```
	IF(@dataZłożenia IS NULL)
	BEGIN
		INSERT INTO Zamówienia
VALUES(@IDKlienta,CONVERT(SMALLDATETIME,GETDATE()),@naKiedy,@czyNaMiejscu,0,0,0)
	END
	INSERT INTO Zamówienia
VALUES(@IDKlienta,@dataZłożenia,@naKiedy,@czyNaMiejscu,0,0,0)

	DECLARE @najnowszeID VARCHAR(255)=CONVERT(VARCHAR(255),(SELECT TOP 1
IDZamówienia FROM Zamówienia ORDER BY IDZamówienia DESC))
	PRINT 'Rozpoczęto zamówienie ID:'+@najnowszeID
END
```

**Usun_dania_starsze_niz** - wykreślamy z karty menu dania starsze niż określona ilość dni

```
CREATE PROCEDURE [dbo].[Usun_dania_starsze_niz]
(
@liczbaDni int
)
AS
BEGIN
	SET NOCOUNT ON
	DECLARE @dzis DATE=CONVERT(DATE,GETDATE())
	DELETE FROM Menu
	WHERE (DATEDIFF(day,OdKiedy,@dzis))>@liczbaDni
END
```

**Usun_danie** - usuwamy danie z bazy danych

```
CREATE PROCEDURE [dbo].[Usun_danie]
(
@nazwaPotrawy varchar(255)
--przepis powinien też się usunąć poprzez usuwanie kaskadowe
)
AS
BEGIN
	DELETE FROM Menu
	WHERE Nazwa LIKE @nazwaPotrawy
END
```

**Usun_klienta** - usuwamy klienta z bazy danych

```
CREATE PROCEDURE [dbo].[Usun_klienta]
	@IDKlienta int
AS
BEGIN
	SET NOCOUNT ON

	BEGIN TRY
		DELETE FROM Klienci WHERE IDKlienta=@IDKlienta
	END TRY
	BEGIN CATCH
		SELECT ERROR_MESSAGE() AS ErrorMessage;
	END CATCH
END
```

**Usun_nadmiarowe_rezerwacje** - procedura wywoływana przez trigger, który sprawdza czy stała Liczba_osob_w_lokalu nie zostala zmieniona, procedura filtruje wszystkie przyszłe rezerwacje i usuwa te nadmiarowe w kolejności od największego ID (zasada kto złożył wcześniej rezerwację ten

```
CREATE PROCEDURE
[dbo].[Usuń_nadmiarowe_rezerwacje]
AS
BEGIN

	DECLARE @przyszleRezrwacje TABLE(dataTab DATETIME)
```

```
	INSERT INTO @przyszleRezrwacje (dataTab)
		(SELECT OD FROM Rezerwacje
		UNION
		SELECT Do FROM Rezerwacje)

	DECLARE @przyszleRezrwacje2 TABLE (id INT,datarez DATETIME)
	INSERT INTO @przyszleRezrwacje2 (id,datarez)
	(SELECT ROW_NUMBER() OVER(ORDER BY datatab), datatab FROM @przyszleRezrwacje)

	DECLARE @wskaznik INT =(SELECT MIN(id) FROM @przyszleRezrwacje2)
	DECLARE @maksInt INT=(SELECT MAX(id) FROM @przyszleRezrwacje2)

	WHILE (@wskaznik<=@maksInt)
	BEGIN
		DECLARE @dataRezerwacji DATETIME=(SELECT datarez FROM @przyszleRezrwacje2
WHERE id=@wskaznik)
		EXEC Usuń_nadmiarowe_rezerwacje_z_danej_godziny @data=@dataRezerwacji
		SET @wskaznik=@wskaznik+1
	END

END
```

**Usun_nadmiarowe_rezerwacje_z_danej_godziny** - procedura wywoływana wielokrotnie przez procedurę **Usun_nadmiarowe_rezerwacje**

```
CREATE PROCEDURE
[dbo].[Usuń_nadmiarowe_rezerwacje_z_danej_godziny]
(@data DATETIME)
AS
BEGIN
	DECLARE @maksOsob INT = (SELECT Wartosc FROM Stałe WHERE Nazwa LIKE
'Liczba_osób_w_lokalu')
	--DECLARE @rezerwacje_z_danej_godziny TABLE(IDRezerwacji INT)
		--(SELECT IDRezerwacji FROM Rezerwacje WHERE @data BETWEEN OD AND DO )
	WHILE ([dbo].[Ilość_osób_w_lokalu] (@data,@data)>@maksOsob)
	BEGIN
		DECLARE @rezerwacjaDoUsuniecia INT = (SELECT MAX(IDRezerwacji) FROM
Rezerwacje WHERE @data BETWEEN OD AND DO)
		IF @rezerwacjaDoUsuniecia IS NULL
			RETURN
		DELETE FROM Rezerwacje WHERE IDRezerwacji=@rezerwacjaDoUsuniecia
	END

END
```

**Usun_przepis** - usuwa całkowicie przepis z bazy

```
CREATE PROCEDURE [dbo].[Usun_przepis]
(
@nazwaPotrawy varchar(255)
)
AS
BEGIN
	DECLARE @IDPotrawy int = (SELECT IDPotrawy FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	DELETE FROM Przepisy WHERE IDPotrawy=@IDPotrawy
END
```

**Usun_stala** - usuwa stałą

```
CREATE PROCEDURE [dbo].[Usun_stala]
	@Nazwa varchar(255)
AS
BEGIN
	BEGIN TRY
		DELETE FROM Stałe WHERE Nazwa=@Nazwa
	END TRY
```

```
	BEGIN CATCH
		SELECT ERROR_MESSAGE() AS ErrorMessage;
	END CATCH
END
```

**Usun_stolik** - usuwa stolik

```
CREATE PROCEDURE [dbo].[Usun_stolik]
(
	@idstolika int
)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN
	IF (@idstolika IN (SELECT @idstolika FROM STOLIKI))
	BEGIN
		DELETE FROM Stoliki WHERE IDStolika=@idstolika
	END
	END
END
```

**Usun_zamowienie** - usuwa zamówienie i jednocześnie zwraca do magazynu składniki, które miały zostać użyte na to zamówienie

```
CREATE PROCEDURE [dbo].[Usun_zamowienie]
(
@IDZamowienia int
)
AS
BEGIN
	SET NOCOUNT ON
	DECLARE @daniaZamowione TABLE (Nr INT,IDPotrawy INT,Ilosc INT)
	INSERT INTO @daniaZamowione (Nr,IDPotrawy,Ilosc)
	(SELECT ROW_NUMBER() OVER(ORDER BY IDPotrawy),IDPotrawy, Ilość FROM
SzczegółyZamówień WHERE IDZamówienia=@IDZamowienia)

	DECLARE @wskaznik1 INT=(SELECT MIN(Nr) FROM @daniaZamowione)
	DECLARE @maksInt1 INT=(SELECT MAX(Nr) FROM @daniaZamowione)

	WHILE(@wskaznik1<=@maksInt1)
	BEGIN
		DECLARE @IDPotrawy INT=(SELECT IDPotrawy FROM @daniaZamowione WHERE
Nr=@wskaznik1)
		DECLARE @iloscPotrawy INT=(SELECT Ilosc FROM @daniaZamowione WHERE
Nr=@wskaznik1)

		EXEC Dodaj_skladniki_z_powrotem
@IDPotrawy=@IDPotrawy,@ilośćPotrawy=@iloscPotrawy

		SET @wskaznik1=@wskaznik1+1
	END

	DELETE FROM Zamówienia WHERE IDZamówienia=@IDZamowienia
END
```

**Wykreśl_danie** - wykreśla danie z karty menu

```
CREATE PROCEDURE [dbo].[Wykreśl_danie]
(
@nazwaPotrawy varchar(255),
@dataWykreślenia DATE
)
AS
BEGIN
```

```
	UPDATE Menu
	SET DataWykreślenia=@dataWykreślenia
	WHERE Nazwa LIKE @nazwaPotrawy
END
```

**Zaktualizuj_skladniki** - procedura wywoływana przez procedurę Dodaj_do_zamówienia, odejmuje składniki na daną potrawę z magazynu

```
CREATE PROCEDURE
[dbo].[Zaktualizuj_skladniki]
(@IDPotrawy INT,
@ilośćPotrawy INT
)
AS
BEGIN
	DECLARE @skladnikiPotrzebne TABLE(ID INT, IDProduktu INT,IlePotrzeba INT)
	INSERT INTO @skladnikiPotrzebne (ID,IDProduktu,IlePotrzeba)
	(SELECT ROW_NUMBER() OVER(ORDER BY IDProduktu), IDProduktu, IlePotrzeba
	FROM Przepisy
	WHERE IDPotrawy=@IDPotrawy)
	DECLARE @wskaznik INT =(SELECT MIN(ID) FROM @skladnikiPotrzebne)
	DECLARE @maksInt INT=(SELECT MAX(id) FROM @skladnikiPotrzebne)
	WHILE(@wskaznik<=@maksInt)
	BEGIN
		DECLARE @ilePotrzeba INT=(SELECT IlePotrzeba FROM @skladnikiPotrzebne
WHERE ID=@wskaznik)
		DECLARE @idProduktu INT=(SELECT IDProduktu FROM @skladnikiPotrzebne WHERE
ID=@wskaznik)
		BEGIN
			UPDATE Magazyn
			SET Ilość=Ilość-@ilePotrzeba*@ilośćPotrawy
			WHERE IDProduktu=@idProduktu
		END
		SET @wskaznik=@wskaznik+1
	END

END
```

**Zaktualizuj_znizki_klientow_indywidualnych** - procedura wywoływana codziennie, liczy łączną sumę wydaną przez klienta oraz ilość jego zamówień w dniu poprzednim

```
CREATE PROCEDURE
[dbo].[Zaktualizuj_znizki_klientow_indywidualnych]
AS
BEGIN

	DECLARE @wczoraj DATETIME=DATEADD(day,-1,GETDATE())
	DECLARE @zamowieniaZWczoraj TABLE(Nr INT,IDKlienta INT, IDZamówienia INT)
	INSERT INTO @zamowieniaZWczoraj (Nr,IDKlienta,IDZamówienia)
	(SELECT ROW_NUMBER() OVER(ORDER BY IDKlienta) , IDKlienta,IDZamówienia FROM
	Zamówienia WHERE DAY(NaKiedy)=DAY(@wczoraj))

	DECLARE @wskaznik INT =(SELECT MIN(Nr) FROM @zamowieniaZWczoraj)
	DECLARE @maksInt INT=(SELECT MAX(Nr) FROM @zamowieniaZWczoraj)

	WHILE(@wskaznik<=@maksInt)
	BEGIN
		DECLARE @IDKlienta INT= (SELECT IDKlienta FROM @zamowieniaZWczoraj WHERE
Nr=@wskaznik)
		DECLARE @IDZamówienia INT=(SELECT IDZamówienia FROM @zamowieniaZWczoraj
WHERE Nr=@wskaznik)
		IF((SELECT Typ FROM Klienci WHERE IDKlienta=@IDKlienta) LIKE 'Klient
Indywidualny')
			BEGIN
```

```
                        DECLARE @wydanaKwota FLOAT=(SELECT DoZapłaty FROM
Zamówienia WHERE IDZamówienia=@IDZamówienia)
                        BEGIN
                              UPDATE ZniżkiKlientówIndywidualnych
                              SET ŁącznaSuma=ŁącznaSuma+@wydanaKwota
                              WHERE IDKlienta=@IDKlienta
                        END
                        IF(@wydanaKwota>(SELECT Wartosc FROM STAŁE WHERE Nazwa LIKE
'K1'))
                              BEGIN
                              UPDATE ZniżkiKlientówIndywidualnych
                              SET LiczZamPowK1=LiczZamPowK1+1
                              WHERE IDKlienta=@IDKlienta
                              END
                  END
            SET @wskaznik=@wskaznik+1
      END
      --druga petla
      --aktualizacja znizek
      SET @wskaznik=(SELECT MIN(Nr) FROM @zamowieniaZWczoraj)
      WHILE (@wskaznik<=@maksInt)
      BEGIN
            DECLARE @IDKlienta2 INT= (SELECT IDKlienta FROM @zamowieniaZWczoraj WHERE
Nr=@wskaznik)
            IF((SELECT Typ FROM Klienci WHERE IDKlienta=@IDKlienta) LIKE 'Klient
Indywidualny')
                  BEGIN
                        UPDATE ZniżkiKlientówIndywidualnych
                        SET Rabat1=([dbo].[Wartosc_Znizka_1](@IDKlienta2)),
                        Rabat2=([dbo].[Wartosc_Znizka_2](@IDKlienta2)),
                        Rabat3=([dbo].[Wartosc_Znizka_3](@IDKlienta2)),
                        Rabat4=([dbo].[Wartosc_Znizka_4](@IDKlienta2))
                        WHERE IDKlienta=@IDKlienta2
                  END
            SET @wskaznik=@wskaznik+1
      END
END
```

**UWAGA** – procedura ta powinna być uruchamiana codziennie za pomocą tzw. *Scheduled Jobs*.

**Zmien_ilosc_produktu -** zmienia ilość produktu w magazynie

```
CREATE PROCEDURE [dbo].[Zmien_ilosc_produktu]
(
@nazwaProduktu varchar(255),
@nowaIlosc float

)
AS
BEGIN
      IF @nowaIlosc<0
      RETURN
      UPDATE Magazyn SET Ilość=@nowaIlosc WHERE NazwaProduktu LIKE @nazwaProduktu

END
```

**Zmien_przepis -** zmienia ilość składnika w danej potrawie, jeśli ilość=0 składnik jest usuwany

```
CREATE PROCEDURE [dbo].[Zmien_przepis]
(
@nazwaPotrawy varchar(255),
@nazwaProduktu varchar(255),
@ilosc int
```

```
)
AS
BEGIN
	IF @ilosc<0
	RETURN
	DECLARE @IDPotrawy int = (SELECT IDPotrawy FROM Menu WHERE Nazwa LIKE
@nazwaPotrawy)
	DECLARE @IDProduktu int =(SELECT IDProduktu FROM Magazyn WHERE NazwaProduktu
LIKE @nazwaProduktu)
	IF EXISTS (SELECT * FROM Przepisy WHERE IDPotrawy=@IDPotrawy AND
IDProduktu=@IDProduktu)
		BEGIN
			IF @Ilosc=0
			BEGIN
				DELETE FROM Przepisy WHERE IDPotrawy=@IDPotrawy AND
IDProduktu=@IDProduktu
			END
			ELSE
			BEGIN
				UPDATE Przepisy SET IlePotrzeba=@ilosc WHERE
IDPotrawy=@IDPotrawy AND IDProduktu=@IDProduktu
			END
		END
END
```

## TRIGGERY

**Produkt_wyczerpal_sie –** Powiadamia o produktach których ilość=0

```
ALTER TRIGGER [dbo].[Produkt_wyczerpal_sie]
ON [dbo].[Magazyn]
AFTER UPDATE
AS
BEGIN
	IF EXISTS (SELECT NazwaProduktu FROM Magazyn WHERE Ilość=0)
	BEGIN
	Print 'Skończyły się następujące produkty'
	(SELECT NazwaProduktu FROM Magazyn WHERE Ilość=0)
	END
END
```

**Dodano_potrawe_niedawno_skreslona** – uruchamia się gdy do menu zostało dodane, bądź zmieniła się jego data wykreślenia, danie niedawno wykreślone

```
ALTER TRIGGER [dbo].[Sprawdz_czy_danie_nie_zostalo_dodane_dzis]
	ON [dbo].[Menu]
	AFTER INSERT, UPDATE
	AS
BEGIN
	DECLARE @OdKiedy date = (SELECT OdKiedy FROM inserted)
	DECLARE @DataDodania date = CONVERT(date, (SELECT DataDodania FROM inserted))
	DECLARE @Dzis date = CONVERT(date, GETDATE())

	IF EXISTS (SELECT * FROM Menu
				WHERE @OdKiedy = @Dzis AND @DataDodania = @Dzis)
	PRINT 'UWAGA. Dodano produkt do dzisiejszego menu'

END
```

**Sprawdz_czy_danie_nie_zostalo_dodane_dzis –** trigger sprawdzający czy danie zostało dodane na datę dzisiejszą

```
ALTER TRIGGER [dbo].[Dodano_potrawe_niedawno_skreslona]
```

```
	ON [dbo].[Menu]
	AFTER UPDATE
	AS
BEGIN
	DECLARE @OdstepCzasowy int = 30   --liczba dni, po których danie moze znów być w
menu
	DECLARE @DataWykreslenia date = (SELECT DataWykreślenia FROM inserted)
	DECLARE @OdKiedy date = (SELECT OdKiedy FROM inserted)

	IF (DATEADD(day, @OdstepCzasowy, @DataWykreslenia) < @OdKiedy)
		PRINT 'UWAGA. Dodano produkt, który znajdował się w menu mniej niż
miesiąc temu'

END
```

**Usuwanie_rezerwacji_po_zmianie_stalej_COVID –** trigger wywołujący się po zmianie stałej Liczba_osob_w_lokalu, trigger wywoluje procedurę filtrującą i usuwającą nadmiarowe rezerwacje

```
ALTER TRIGGER [dbo].[Usuwanie_rezerwacji_po_zmianie_stalej_COVID]
	ON [dbo].[Stałe]
	AFTER UPDATE
	AS
BEGIN
	DECLARE @NazwaZmienionejStalej varchar(255) = (SELECT Nazwa FROM inserted)
	IF (@NazwaZmienionejStalej LIKE 'Liczba_osob_w_lokalu')
	BEGIN
		DECLARE @StaraWartosc float = (SELECT Wartosc FROM deleted)
		DECLARE @NowaWartosc float = (SELECT Wartosc FROM inserted)
		IF (@NowaWartosc < @StaraWartosc)
			EXEC dbo.Usuń_nadmiarowe_rezerwacje
	END
END
```

## FUNKCJE ZWRACAJĄCE TABLICE

**Dania_starsze_niz_x_dni(**
@Dni int
) – zwraca wszystkie dania, które znajdują się w karcie dań od dłużej niż x dni

```
CREATE FUNCTION [dbo].[Dania_starsze_niz_x_dni](
	@Dni int
)
RETURNS TABLE
AS
RETURN
(
	SELECT Nazwa
	FROM Menu
	WHERE (DataWykreślenia IS NULL OR  CONVERT(date, GETDATE()) < DataWykreślenia)
AND  --danie jest nieskreslone
		DATEADD(dd, @Dni, DataDodania) < CONVERT(date, GETDATE())          --
data dodania + dni < dzisiaj
)
```

**Klienci_z_danej_firmy**(
@IDFirmy int
) – zwraca wszystkich klientów podanej firmy

```
CREATE FUNCTION
[dbo].[Klienci_z_danej_firmy]
(
@IDFirmy int

)
RETURNS TABLE
AS
RETURN
	(SELECT * FROM KlienciIndywidualni WHERE IDFirmy=@IDFirmy)
```

**Zwroc_zamowienie_po_ID_rezerwacji**(
@IDRezerwacji int
) – zwraca zamówienie po IDRezerwacji

```
CREATE FUNCTION [dbo].[Zwroc_zamowienie_po_ID_rezerwacji](
	@IDRezerwacji int
)
RETURNS TABLE
AS
RETURN(
	SELECT * FROM Zamówienia
	WHERE IDZamówienia=
		(SELECT IDZamówienia FROM Rezerwacje
		WHERE IDRezerwacji=@IDRezerwacji)
	)
```

## WIDOKI

**Dzisiejsze_menu** – pokazuje wszystkie dania, które znajdują się w dzisiejszej ofercie. Zawiera ona informacje zazwyczaj zawierane w karcie dań, tj. nazwa, opis dania, jego cena oraz kategoria, do jakiej należy

```
CREATE VIEW [dbo].[Dzisiejsze_menu] AS
(
SELECT Menu.Nazwa,Menu.Opis,Menu.Cena,KategorieMenu.NazwaKategorii
FROM Menu
JOIN KategorieMenu ON Menu.IDKategorii=KategorieMenu.IDKategorii
WHERE CzySkładnikiWMagazynie=1 AND (DataWykreślenia>GETDATE() OR DataWykreślenia IS
NULL)
)
```

**Dania_do_zrobienia_dzisiaj** – zawiera informacje o daniach zamówionych na dzisiaj. Widok został stworzony z myślą przede wszystkim dla kucharzy, którzy muszą dynamicznie rozplanowywać, które dania powinni teraz przygotować. Zawiera informacje takie jak nazwa produktu, jego opis, zamówioną ilość, czas , na kiedy trzeba zrobić danie oraz, czy danie jest na miejscu

```
CREATE VIEW [dbo].[Dania_do_zrobienia_dzisiaj] AS
	(SELECT Menu.Nazwa, Menu.Opis, SzczegółyZamówień.Ilość, Zamówienia.NaKiedy,
Zamówienia.CzyNaMiejscu
		FROM SzczegółyZamówień JOIN Menu ON
Menu.IDPotrawy=SzczegółyZamówień.IDPotrawy
			JOIN Zamówienia ON
SzczegółyZamówień.IDZamówienia=Zamówienia.IDZamówienia
			WHERE Zamówienia.IDZamówienia IN
					(SELECT IDZamówienia
					FROM
[dbo].Wszystkie_zamowienia_danego_dnia(CONVERT(date, GETDATE())) )
				AND Zamówienia.NaKiedy > GETDATE()
	)
```

**Dania_do_zrobienia_dzisiaj_na_miejscu** – widok analogiczny do powyżej opisanego, jednak zawiera jedynie dania na miejscu

```
CREATE VIEW [dbo].[Dania_do_zrobienia_dzisiaj_na_miejscu] AS
	(SELECT * FROM Dania_do_Zrobienia_dzisiaj
	WHERE CzyNaMiejscu = 1
	)
```

**Dania_do_zrobienia_dzisiaj_na_wynos** – widok analogiczny do powyższego, jednak zawiera jedynie dania na wynos

```
CREATE VIEW [dbo].[Dania_do_zrobienia_dzisiaj_na_wynos] AS
	(SELECT * FROM Dania_do_Zrobienia_dzisiaj
	WHERE CzyNaMiejscu = 0
	)
```

**Klienci_z_rezerwacją_dzisiaj –** widok zawierający informacje o klientach mających rezerwację na dany dzień. Zawiera podstawowe informacje o kliencie tj. Nazwisko (lub Nick, gdy nie zgodził się na RODO) oraz telefon, a także informacje o rezerwacji – daty i godziny rozpoczęcia i zakończenia rezerwacji, liczbę osób danej rezerwacji oraz czy klient zapłaci za swoje danie na miejscu

```
CREATE VIEW [dbo].[Klienci_z_rezerwacja_dzisiaj] AS
(
SELECT KlienciIndywidualni.Nazwisko, KlienciIndywidualni.Nick,
Rezerwacje.OD, Rezerwacje.DO, Rezerwacje.LiczbaOsob,
Rezerwacje.CzyZapłataNaMijescu, KlienciIndywidualni.Telefon
FROM Rezerwacje JOIN Zamówienia ON Rezerwacje.IDZamówienia = Zamówienia.IDZamówienia
JOIN Klienci ON Zamówienia.IDKlienta = Klienci.IDKlienta
JOIN KlienciIndywidualni ON Klienci.IDKlienta = KlienciIndywidualni.IDKlienta
WHERE CONVERT(date, Rezerwacje.OD) = CONVERT(date, GETDATE())
)
```

**Przyszle_dzisiejsze_rezerwacje** – widok zawierący informacje o wszystkich jeszcze nie rozpoczętych rezerwacjach, które są zaplanowane na dzisiaj. Zawiera informacje takie jak nazwisko, na które jest rezerwacja, numer (identyfikacyjny) stolika, liczbę osób oraz przedział czasowy (kolumny OD i DO) rezerwacji.

```
CREATE VIEW [dbo].[Przyszle_dzisiejsze_rezerwacje] AS
```

```
	(SELECT Nazwisko, IDStolika, LiczbaOsob, OD, DO, CzyZapłataNaMijescu FROM
[dbo].Wszystkie_rezerwacje_danego_dnia(CONVERT(date, GETDATE()))
	 WHERE DO < GETDATE())
GO
```

**Wszystkie_obecne_rezerwacje** – widok zawierający informacje o wszystkich rezerwacjach, które powinny odbywać się teraz. Widok został stworzony z myślą o kelnerach obsługujących klientów przy wejściu. Zawiera kolumny identyczne do kolumn widoku Przyszle_dzisiejsze_rezerwacje

```
CREATE VIEW [dbo].[Wszystkie_obecne_rezerwacje] AS
	(SELECT Nazwisko, IDStolika, LiczbaOsob, CzyZapłataNaMijescu, OD, DO FROM
Rezerwacje
	 WHERE OD <= GETDATE() AND GETDATE() <= DO)
GO
```

**Klienci_z_zamowieniami_dzisiaj** – widok zawierający informacje o klientach indywidualnych, którzy mają zamówienie na dzisiaj. Zawiera kolumny z informacjami takimi jak data złożenia zamówienia, data odebrania zamówienia, kwotę do zapłaty oraz czy dane zamówienie jest na miejscu czy na wynos. Ponadto zawiera podstawowe informacje o kliencie, tj. nazwisko (bądź nick w przypadku nie zgodzenia się na RODO) oraz telefon

```
CREATE VIEW [dbo].[Klienci_z_zamowieniami_dzisiaj] AS
	(SELECT * FROM [dbo].Klienci_z_zamowieniami_danego_dnia(CONVERT(date,
GETDATE()))
	WHERE NaKiedy > GETDATE())
```

# RAPORTY

## Dla firm

**Miesieczne_zamowienia_firmy(**
@nazwaFirmy varchar(255),
@dateOd date
) – zwraca informacje na temat wszystkich zamówień danej firmy na 30 dni przed podaną datą włącznie

```
CREATE FUNCTION
[dbo].[Miesieczne_zamowienia_firmy]
(@nazwaFirmy VARCHAR(255),
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT
Menu.Nazwa,SzczegółyZamówień.Ilość,SzczegółyZamówień.Ilość*SzczegółyZamówień.CenaJedno
stkowa AS Cena
FROM SzczegółyZamówień
JOIN Menu ON SzczegółyZamówień.IDPotrawy=Menu.IDPotrawy
JOIN Zamówienia ON SzczegółyZamówień.IDZamówienia=Zamówienia.IDZamówienia
```

```
WHERE Zamówienia.IDKlienta=(SELECT IDFirmy FROM Firmy WHERE NazwaFirmy LIKE
@nazwaFirmy)
AND DATEDIFF(MONTH,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Miesieczne_znizki_firmy**(
@nazwaFirmy varchar(255),
@dateOd date
) – zwraca informacje na temat wszystkich zniżek danej firmy na 30 dni przed podaną datą włącznie

```
CREATE FUNCTION
[dbo].[Miesieczne_zniżki_firmy]
(@nazwaFirmy VARCHAR(255),
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT SUM(Zamówienia.Cena) AS 'Cena bez rabatu',SUM(Zamówienia.Zniżka) AS 'Zniżka',
SUM(Zamówienia.DoZapłaty) AS 'Łączna wydana kwota'
FROM Zamówienia
WHERE Zamówienia.IDKlienta=(SELECT IDFirmy FROM Firmy WHERE NazwaFirmy LIKE
@nazwaFirmy)
AND DATEDIFF(MONTH,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Tygodniowe_znizki_firmy**(
@nazwaFirmy varchar(255),
@dateOd date
) – zwraca informacje na temat wszystkich zamówień danej firmy na 7 dni przed podaną datą włącznie

```
CREATE FUNCTION
[dbo].[Tygodniowe_zniżki_klienta]
(@IDKlienta INT,
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT SUM(Zamówienia.Cena) AS 'Cena bez rabatu',SUM(Zamówienia.Zniżka) AS 'Zniżka',
SUM(Zamówienia.DoZapłaty) AS 'Łączna wydana kwota'
FROM Zamówienia
WHERE Zamówienia.IDKlienta=@IDKlienta
AND DATEDIFF(WEEK,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Tygodniowe_zamowienia_firmy(**
@nazwaFirmy varchar(255),
@dateOd date
) – zwraca informacje na temat wszystkich zniżek danej firmy na 7 dni przed podaną datą włącznie

```
CREATE FUNCTION
[dbo].[Tygodniowe_zamowienia_firmy]
(@nazwaFirmy VARCHAR(255),
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT
Menu.Nazwa,SzczegółyZamówień.Ilość,SzczegółyZamówień.Ilość*SzczegółyZamówień.CenaJedno
stkowa AS Cena
FROM SzczegółyZamówień
JOIN Menu ON SzczegółyZamówień.IDPotrawy=Menu.IDPotrawy
JOIN Zamówienia ON SzczegółyZamówień.IDZamówienia=Zamówienia.IDZamówienia

WHERE Zamówienia.IDKlienta=(SELECT IDFirmy FROM Firmy WHERE NazwaFirmy LIKE
@nazwaFirmy)
AND DATEDIFF(WEEK,@dataOd,Zamówienia.NaKiedy)=0
)
```

## Dla klientów indywidualnych

**Miesieczne_zamowienia_klienta**(

@IDKlienta int,

@dataOd date

) – zwraca najważniejsze informacje na temat zamówień klienta indywidualnego zrealizowanych w ciągu 30 dni od podanej daty

```
CREATE FUNCTION
[dbo].[Miesieczne_zamowienia_klienta]
(@IDKlienta INT,
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT
Menu.Nazwa,SzczegółyZamówień.Ilość,SzczegółyZamówień.Ilość*SzczegółyZamówień.CenaJedno
stkowa AS Cena
FROM SzczegółyZamówień
JOIN Menu ON SzczegółyZamówień.IDPotrawy=Menu.IDPotrawy
JOIN Zamówienia ON SzczegółyZamówień.IDZamówienia=Zamówienia.IDZamówienia
WHERE Zamówienia.IDKlienta=@IDKlienta
AND DATEDIFF(MONTH,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Miesieczne_znizki_klienta(**

@IDKlienta int,

@dataOd date

) – zwraca najważniejsze informacje na temat zniżek klienta indywidualnego zrealizowanych w ciągu 30 dni od podanej daty

```
CREATE FUNCTION
[dbo].[Miesieczne_zniżki_klienta]
(@IDKlienta INT,
@dataOd DATE
)
```

```
RETURNS TABLE
RETURN
(
SELECT SUM(Zamówienia.Cena) AS 'Cena bez rabatu',SUM(Zamówienia.Zniżka) AS 'Zniżka',
SUM(Zamówienia.DoZapłaty) AS 'Łączna wydana kwota'
FROM Zamówienia
WHERE Zamówienia.IDKlienta=@IDKlienta
AND DATEDIFF(MONTH,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Tygodniowe_zamowienia_klienta(**

@IDKlienta int,

@dataOd date

) – zwraca najważniejsze informacje na temat zamówień klienta indywidualnego zrealizowanych w ciągu 7 dni od podanej daty

```
CREATE FUNCTION
[dbo].[Tygodniowe_zamowienia_klienta]
(@IDKlienta INT,
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT
Menu.Nazwa,SzczegółyZamówień.Ilość,SzczegółyZamówień.Ilość*SzczegółyZamówień.CenaJedno
stkowa AS Cena
FROM SzczegółyZamówień
JOIN Menu ON SzczegółyZamówień.IDPotrawy=Menu.IDPotrawy
JOIN Zamówienia ON SzczegółyZamówień.IDZamówienia=Zamówienia.IDZamówienia
WHERE Zamówienia.IDKlienta=@IDKlienta
AND DATEDIFF(WEEK,@dataOd,Zamówienia.NaKiedy)=0
)
```

**Tygodniowe_znizki_klienta**(

@IDKlienta int,

@dataOd date

) – zwraca najważniejsze informacje na temat zniżek klienta indywidualnego zrealizowanych w ciągu 7 dni od podanej daty

```
CREATE FUNCTION
[dbo].[Tygodniowe_zniżki_klienta]
(@IDKlienta INT,
@dataOd DATE
)
RETURNS TABLE
RETURN
(
SELECT SUM(Zamówienia.Cena) AS 'Cena bez rabatu',SUM(Zamówienia.Zniżka) AS 'Zniżka',
SUM(Zamówienia.DoZapłaty) AS 'Łączna wydana kwota'
FROM Zamówienia
WHERE Zamówienia.IDKlienta=@IDKlienta
AND DATEDIFF(WEEK,@dataOd,Zamówienia.NaKiedy)=0
)
```

## Dla pracowników

**Produkty_do_potrawy**(
@IDPotrawy int
) – zwraca tablicę z nazwami i ilością produktów, które potrzebne są do danej potrawy.

```
CREATE FUNCTION [dbo].[Produkty_do_potrawy]
(
	@IDPotrawy int
)
RETURNS TABLE
AS
RETURN
(
	SELECT Magazyn.NazwaProduktu, Przepisy.IlePotrzeba
	FROM Przepisy JOIN Magazyn ON Przepisy.IDProduktu = Magazyn.IDProduktu
	WHERE Przepisy.IDPotrawy = @IDPotrawy
)
```

**Rezerwacje_o_danej_porze(**
@DataICzas datetime
) – zwraca wszystkie rezerwacje, które mają miejsce o wskazanej dacie i godzinie

```
CREATE FUNCTION [dbo].[Rezerwacje_o_danej_porze](
	@DataICzas datetime
)RETURNS TABLE
AS
RETURN (SELECT * FROM Rezerwacje
		WHERE OD <= @DataICzas AND @DataICzas <= DO)
```

**Wszysktie_rezerwacje_danego_dnia**(
	@Dzien date
) – pokazuje informacje na temat rezerwacji danego dnia

```
CREATE FUNCTION [dbo].[Wszystkie_rezerwacje_danego_dnia](
	@Dzien date
)RETURNS TABLE
AS
RETURN (SELECT * FROM Rezerwacje
		WHERE CONVERT(date, OD) = @Dzien)
```

**Wszystkie_zamówienia_danego_dnia**(
	@Dzien date
) – pokazuje informacje na temat zamówień danego dnia

```
CREATE FUNCTION [dbo].[Wszystkie_zamowienia_danego_dnia](
	@Dzien date
)RETURNS TABLE
AS
RETURN (SELECT * FROM Zamówienia
WHERE CONVERT(date, NaKiedy) = @Dzien)
```

### Dla menedżera

Tygodniowe_dochody(
	@dataOd datetime
) – funkcja zwracająca dochód

```
CREATE FUNCTION
Tygodniowe_dochody
(
@dataOd DATETIME
)
RETURNS TABLE
AS
RETURN
(
	SELECT SUM(DoZapłaty) AS Suma FROM Zamówienia
	WHERE DATEDIFF(DAY,@dataOd,NaKiedy)<=7 AND DATEDIFF(DAY,@dataOd,NaKiedy)>=0
)
```

```
CREATE FUNCTION
Miesieczne_dochody
(
@dataOd DATETIME
)
RETURNS TABLE
AS
RETURN
(
	SELECT SUM(DoZapłaty) AS Suma FROM Zamówienia
	WHERE DATEDIFF(MONTH,@dataOd,NaKiedy)=0
)
```

## OPIS UŻYTKOWNIKÓW

W naszym systemie przyjęto 3 główne role, każde z odpowiednimi możliwościami w systemie:

1. Menedżer – jako główna osoba w firmie przejmuje rolę administratora bazy danych. Ma on dostęp do wszystkich procedur, funkcji i widoków w bazie, a w szczególności do:
    a. Dodawania/usuwania pracowników w bazie danych
    b. Planowanie i modyfikacja menu
    c. Zmiany stałych zniżek i ograniczeń COVIDowych
    d. Sprawdzanie podsumowań tygodniowych i miesięcznych własnej firmy, a także klientów indywidualnych i firm
2. Pracownik – (np. Kelner, Kucharz) – do jego możliwości w systemie należą między innymi:
    a. Dodawanie i anulowanie rezerwacji/zamówień
    b. Widoki dzisiejszych rezerwacji/zamówień/klientów
    c. Przyjęcie dostawy do magazynu

3. Klient (indywidualny i firma) mogą
    a. Złożenie zamówienia
    b. Dokonanie rezerwacji
    c. Zobaczyć widoki „Dzisiejsze menu”
    d. Sprawdzić własne podsumowania tygodniowe i miesięczne

## GENERATOR DANYCH

Aby sprawdzić poprawność większości funkcji, procedur i wyzwalaczy, stworzono generator danych, który „zapełnił” bazę danych przykładowymi rekordami. Miały one symulować ok. roku prężnej pracy restauracji.
Zapełnianie bazy przebiegało trzyetapowo:

1. **Wygenerowanie, dostosowanie i pobranie plików** typu *json* za pomocą strony Mockaroo. Ze względu na jej ograniczenia strony odnośnie tworzenia nazw potraw i przepisów, zdecydowano się na pobranie informacji (tzw. Web Scraping) ze strony jednej z krakowskich pizzernii (strona źródłowa: http://www.czesiopizza.pl/). Kod programu do pobrania informacji i zapisu ich do plików json, zamieszczono poniżej:

```
import bs4
import json
import random
import re
import requests

# stałe
opis = ["Niebiańska uczta", "Palce lizać", "Ulubione danie szefa
kuchni", "Dobre, bo polskie", "Syte, a tanie",
        "Najlepsze w mieście"]


def getBSObjHTML(url):
    res = requests.get(url)
    soup = bs4.BeautifulSoup(res.text, 'html.parser')
    return soup


def clearProductsDescriptionText(description):
    description = description.lstrip("(")  # removing parenthesis
    description = description.rstrip(")")
    productsList = description.split(",")
    return [product.strip() for product in productsList]  # clearing
whitespaces


def getDatesFromJson(jsonFileDest):
    with open(jsonFileDest, 'r') as jsonFile:
        datesList = json.loads(jsonFile.read())

    return [elem['id'] for elem in datesList]


def toJson(data, destination):
    with open(destination, 'w', encoding='utf8') as file:
        file.write(json.dumps(data, ensure_ascii=False))
```

```
def getPrice(rawStringWithPriceTag):
    priceRegex = re.compile(r'\d+,\d\d')
    mo = priceRegex.search(rawStringWithPriceTag)
    priceStr = mo.group()
    priceSrtComma = priceStr.replace(",", ".")
    return float(priceSrtComma)


def getProductsAndRecipes(soup):
    products = set()
    productsToSkip = {"kieÅ□basa peperoni", "plus 6 dowolnych
skÅ□adnikÃ³w"}
    recipes = []

    productListObjects = soup.select(".opis")
    rawDescriptions = [elem.text for elem in productListObjects]
    for rawDesc in rawDescriptions:
        recipe = clearProductsDescriptionText(rawDesc)
        for product in recipe:
            if product not in productsToSkip:
                products.add(product)
            else:
                recipe.remove(product)
        recipes.append(recipe)


    productsJsonFormat = [{"NazwaProduktu": productName, "Ilość":
99999, "Jednostka": "kg"} for productName in products]

    return productsJsonFormat, recipes


def getPizzas(soup, recipes, addDates):
    def clearPizzaName(rawName):
        prefixRegex = re.compile(r'\d+.\s')
        mo = prefixRegex.search(rawName)
        prefix = mo.group()
        return rawName.lstrip(prefix)

    pizzaNamesListObjects = soup.select(".title1")
    rawPizzaNames = [elem.text for elem in pizzaNamesListObjects]
    pizzaNames = [clearPizzaName(rawName) for rawName in
rawPizzaNames]

    pizzaPricesListObject = soup.select(".post > table")
    pizzaPrices = [getPrice(pizzaPriceLabel.text) for pizzaPriceLabel
in pizzaPricesListObject[1:43]]

    mockDescriptions = [random.choice(opis) for _ in
range(len(pizzaNames))]

    pizzaJsonFormat = [{"nazwa": pizzaNames[i],
                        "opis": mockDescriptions[i],
                        "cena": pizzaPrices[i],
                        "kategoria": "pizza",
                        "odKiedy": addDates[i],
                        "przepis": recipes[i]}
                       for i in range(len(pizzaNames))]
```

```
    return pizzaJsonFormat


if __name__ == "__main__":
    url = "http://www.czesiopizza.pl/"
    jsonDest = "Daty_dodania_pizz.json"

    productJsonFileDest = "Produkty.json"
    pizzasJsonFileDest = "Menu.json"
    bs = getBSObjHTML(url)

    dates = getDatesFromJson(jsonDest)
    products, recipes = getProductsAndRecipes(bs)
    pizzas = getPizzas(bs, recipes, dates)

    toJson(products, productJsonFileDest)
    toJson(pizzas, pizzasJsonFileDest)
```

2. **Stworzenie zapytań** do bazy danych, zawierający dane z wcześniej przygotowanych plików *json*. W tym celu stworzono program w języku Python 3.8 Pobiera on dane z plików, tworzy kwerendy i zapisuje je w pliku tekstowym. Jego kod znajduje się poniżej:

```
import json
import os
from random import choice, randint


def readJSON(jsonPath):
    with open(jsonPath, 'r', encoding='utf-8') as jsonFile:
        jsonList = json.loads(jsonFile.read())
    return jsonList


def concatenateJsons(folderName):
    cwd = os.getcwd()
    os.chdir(folderName)
    jsonPaths = os.listdir()
    jsonFullList = []
    for jsonPath in jsonPaths:
        list = readJSON(jsonPath)
        jsonFullList.extend(list)

    os.chdir(cwd)
    return jsonFullList


def writeToFile(queriesList, dest):
    with open(dest, 'w', encoding='utf-8') as destFile:
        destFile.writelines(queriesList)


def usefulDeclarations(declarations):  # deklaracje zmiennych na
początek pliku
    return [f'DECLARE {declaration} int\n' for declaration in
declarations]


def makeIndividualClientsQuery(jsonPath, procedureName):  # dodanie
klientow indywidualnych do bazy
```

```
    clientsList = readJSON(jsonPath)
    result = []

    for client in clientsList:
        nazwiskoLubNick = client['Nazwisko'] if client['CzyRODO'] ==
1 else client['Nick']

        queryLine = f'EXEC {procedureName} '
        queryLine += f'@CzyRODO = {client["CzyRODO"]}, '
        queryLine += f'@NazwiskoLubNick = \'{nazwiskoLubNick}\', '
        queryLine += f'@NazwaFirmy = {client["NazwaFirmy"]}, '
        queryLine += f'@Telefon = \'{client["Telefon"]}\'\n'
        result.append(queryLine)

    return result


def makeCompanyQuery(jsonPath, procedureName):  # dodanie firm do
bazy
    companiesList = readJSON(jsonPath)
    result = []
    for companyInfo in companiesList:
        queryLine = f'EXEC {procedureName} '
        queryLine += f'@NazwaFirmy = \'{companyInfo["NazwaFirmy"]}\',
'
        queryLine += f'@Telefon = \'{companyInfo["Telefon"]}\', '
        queryLine += f'@NIP = \'{companyInfo["NIP"]}\'\n'
        result.append(queryLine)
    return result


# dodanie losowo firm do klientów
def attachCompaniesToClients(clientsJsonPath, companiesJsonPath,
procedure_name, n):
    result = []
    clients = readJSON(clientsJsonPath)
    companies = readJSON(companiesJsonPath)

    for _ in range(n):
        randClient = choice(clients)
        clientVarDeclaration = 'SET @IDKlienta = (SELECT TOP 1
IDKlienta FROM KlienciIndywidualni WHERE '
        if randClient['CzyRODO'] == 1:
            clientVarDeclaration +=
f'Nazwisko=\'{randClient["Nazwisko"]}\')\n'
        else:
            clientVarDeclaration +=
f'Nick=\'{randClient["Nick"]}\')\n'

        randCompany = choice(companies)
        companyVarDeclaration = f'SET @IDFirmy =' \
                                f' (SELECT TOP 1 IDFirmy FROM Firmy
WHERE NazwaFirmy = \'{randCompany["NazwaFirmy"]}\')\n'

        queryLine = f'EXEC {procedure_name} @IDKlienta = @IDKlienta,
@IDFirmy = @IDFirmy\n'
        result.extend([clientVarDeclaration, companyVarDeclaration,
queryLine])

    return result
```

```
def makeStorageQueries(jsonPath, procedure_name):  # dodanie
produktów w magazynie do bazy
    productsInStorage = readJSON(jsonPath)
    result = []
    for prod in productsInStorage:
        queryLine = f'EXEC {procedure_name} '
        queryLine += f'@NazwaProduktu = \'{prod["NazwaProduktu"]}\',
'
        queryLine += f'@ilosc = {prod["Ilość"]}, '
        queryLine += f'@Jednostka = {prod["Jednostka"]}\n'
        result.append(queryLine)
    return result


def makeMenuQueries(jsonPath, dishAdd_procedure_name,
ingredients_procedure_name):
    dishes = readJSON(jsonPath)
    result = []
    for dish in dishes:
        singleDishQuery = []
        addDishQueryLine = f'EXEC {dishAdd_procedure_name}
@nazwa=\'{dish["nazwa"]}\', ' \
                          f'@opis=\'{dish["opis"]}\',
@cena={dish["cena"]}, @odKiedy=\'{dish["odKiedy"]}\',
@kategoria=\'{dish["kategoria"]}\'\n'
        singleDishQuery.append(addDishQueryLine)
        recipe = dish["przepis"]
        for ingredient in recipe:
            addIngredientQueryLine = f'EXEC
{ingredients_procedure_name}' \
                                     f'
@nazwaPotrawy=\'{dish["nazwa"]}\', @nazwaProduktu=\'{ingredient}\',
@ilosc=1\n'
            singleDishQuery.append(addIngredientQueryLine)
        result.extend(singleDishQuery)

    return result


def makeOrdersQueries(dirPath, menuPath, createOrder_procedure_name,
addToOrder_procedure_name):
    orders = concatenateJsons(dirPath)
    orders = list(filter(lambda x: x["DataZłozenia"] < x["NaKiedy"],
orders))
    dishes = readJSON(menuPath)
    dishNames = [dish["nazwa"] for dish in dishes]

    result = []
    for order in orders:
        singleOrderQuery = []
        ifStatement = f"IF ({order['IDKlienta']} IN (SELECT IDKlienta
FROM Klienci) OR {order['IDKlienta']} IS NULL)\nBEGIN\n"
        createOrderQueryLine = f"EXEC {createOrder_procedure_name}
@IDKlienta={order['IDKlienta']}, " \
                               f"@naKiedy=\'{order['NaKiedy']}\',
@dataZłożenia=\'{order['DataZłozenia']}\', " \

f"@czyNaMiejscu={order['CzyNaMiejscu']}\n"
        setIDQuery = f'SET @IDZamówienia =' \
                     f'(SELECT TOP 1 IDZamówienia FROM Zamówienia
```

```
WHERE dataZłożenia = \'{order["DataZłozenia"]}\')\n'
        singleOrderQuery.extend([ifStatement, createOrderQueryLine,
setIDQuery])

        randomDishes = [choice(dishNames) for _ in range(3)]
        for randDish in randomDishes:
            addRandomDishQueryLine = f'EXEC
{addToOrder_procedure_name} @IDZamówienia=@IDZamówienia, ' \
                                     f'@nazwaPotrawy=\'{randDish}\',
@ilość={randint(1, 4)}\n'
            singleOrderQuery.append(addRandomDishQueryLine)
        singleOrderQuery.append('END\n')

        result.extend(singleOrderQuery)

    return result


def makeTableQuaries(n, a, b):
    return [f"INSERT INTO Stoliki(LiczbaMiejsc) VALUES ({randint(a,
b)})\n" for _ in range(n)]


def makeCompanyReservationQueries(jsonPath, procedure_name):
    reservations = readJSON(jsonPath)
    for res in reservations:
        if len(res["OD"]) == 4:  res["OD"] = '0' + res["OD"]
        if len(res["DO"]) == 4:  res["DO"] = '0' + res["DO"]

        res["OD"] = res["dzien"] + ' ' + res["OD"]
        res["DO"] = res["dzien"] + ' ' + res["DO"]
    reservations = list(filter(lambda x: x["OD"] < x["DO"],
reservations))

    result = []
    for res in reservations:
        queryLine = f'EXEC {procedure_name} @NazwaFirmy=
\'{res["NazwaFirmy"]}\',' \
                    f'@LiczbaOsób={res["LiczbaOsób"]},
@OD=\'{res["OD"]}\', @DO=\'{res["DO"]}\'\n'
        result.append(queryLine)
    return result


def makeClientReservationQueries(jsonPath, procedure_name):
    reservations = readJSON(jsonPath)
    for res in reservations:
        if len(res["OD"]) == 4:  res["OD"] = '0' + res["OD"]
        if len(res["DO"]) == 4:  res["DO"] = '0' + res["DO"]
        res["OD"] = res["dzien"] + ' ' + res["OD"]
        res["DO"] = res["dzien"] + ' ' + res["DO"]
    reservations = list(filter(lambda x: x["OD"] < x["DO"],
reservations))

    result = []
    for res in reservations:
        singleReservationQuery = []
        ifStatement = f"IF ({res['IDZamówienia']} IN (SELECT
IDZamówienia FROM Zamówienia))\nBEGIN\n"
        createReservationQueryLine = f'EXEC {procedure_name}
@IDZamówienia={res["IDZamówienia"]},' \
```

```
f'@nazwisko=\'{res["Nazwisko"]}\', @liczbaOsob={res["liczbaOsób"]}, '
\

f'@czyZaplataNaMiejscu={res["czyZaplataNaMiejscu"]},
@rezerwacjaOD=\'{res["OD"]}\',' \

f'@rezerwacjaDO=DATEADD(MINUTE,45, @wygenerowanaData)\n'
        singleReservationQuery.extend([ifStatement,
createReservationQueryLine, "END\n"])
        result.extend(singleReservationQuery)
    return result


if __name__ == "__main__":
    # Dodawanie podstawowych deklaracji zmiennych
    declarations = usefulDeclarations(["@IDKlienta", "@IDFirmy",
"@IDPotrawy",
                                       "@IDZamówienia",
"@IDProduktu", "@IDKategorii"])

    # Dodanie klientów, firm oraz przyporządkowanie klientom firm
    individualClientsQueries =
makeIndividualClientsQuery('KlienciIndywidualni.json',
'Dodaj_klienta_indywidualnego')
    companyQueries = makeCompanyQuery('Firmy.json', 'Dodaj_firme')
    addingCompToClientsQueries = attachCompaniesToClients \
        ('KlienciIndywidualni.json', 'Firmy.json',
'Modyfikuj_firme_klienta', 250)
    clientsQueries = individualClientsQueries + companyQueries +
addingCompToClientsQueries

    # Dodanie produktów, dań i przepisów
    storageQueries = makeStorageQueries('Produkty.json',
'Dodaj_produkt')
    menuQueries = makeMenuQueries('Menu.json', 'Dodaj_danie',
'Dodaj_do_przepisu')
    foodQueries = storageQueries + menuQueries

    # Dodawanie zamówień i rezerwacji
    ordersQueries = makeOrdersQueries("Zamówienia", "Menu.json",
"Rozpocznij_zamówienie", "Dodaj_do_zamówienia")
    tablesQueries = makeTableQuaries(15, 1, 10)
    companyReservations =
makeCompanyReservationQueries("RezerwacjeFirm.json",
"Dodaj_rezerwacje_dla_firmy")
    clientsReservations =
makeClientReservationQueries("Rezerwacje_klientow_indywidualnych.json
", "Dokonaj_rezerwacji")
    functionalityQueries = ordersQueries + tablesQueries +
companyReservations + clientsReservations

    result = declarations + clientsQueries + foodQueries +
functionalityQueries
    writeToFile(result, "komendy.txt")    #zapisanie do pliku
```

3. **Zrealizowanie komend** w SQL-Server. Zdecydowano się na „przeklejenie” zawartości pliku wygenerowanego przez powyższy program i uruchomienie kwerendy w oknie. O wiele bardziej eleganckim rozwiązaniem byłoby użycie specjalnie przygotowanych

pod tego typu zadania komend, tj. OPENROWS czy BULK. Niestety, nie posiadano zgody na użycie komendy BULK, więc zdecydowano się na „ręczne” wprowadzenie zapytań.

Fragment stworzonego przez generator kodu zapytań zamieszczono poniżej:

```
--deklaracja zmiennych
DECLARE @IDKlienta int
DECLARE @IDFirmy int
DECLARE @IDPotrawy int
DECLARE @IDZamówienia int
DECLARE @IDProduktu int
DECLARE @IDKategorii int

--dodanie klientów indywidualnych
EXEC Dodaj_klienta_indywidualnego @CzyRODO = 0, @NazwiskoLubNick =
'Zoolab128', @NazwaFirmy = NULL, @Telefon = '757632531'
EXEC Dodaj_klienta_indywidualnego @CzyRODO = 0, @NazwiskoLubNick =
'Stronghold91', @NazwaFirmy = NULL, @Telefon = '021693834'
EXEC Dodaj_klienta_indywidualnego @CzyRODO = 0, @NazwiskoLubNick =
'Zoolab810', @NazwaFirmy = NULL, @Telefon = '697088349'
EXEC Dodaj_klienta_indywidualnego @CzyRODO = 0, @NazwiskoLubNick =
'Alpha442', @NazwaFirmy = NULL, @Telefon = '600129398'
EXEC Dodaj_klienta_indywidualnego @CzyRODO = 1, @NazwiskoLubNick =
'Doloritas Stopforth', @NazwaFirmy = NULL, @Telefon = '650509510'

(...)

EXEC Dodaj_klienta_indywidualnego @CzyRODO = 0, @NazwiskoLubNick =
'Stringtough532', @NazwaFirmy = NULL, @Telefon = '522716989'

--dodanie firm
EXEC Dodaj_firme @NazwaFirmy = 'Brainsphere', @Telefon = '145082677', @NIP
= '1569707251'
EXEC Dodaj_firme @NazwaFirmy = 'Bubblemix', @Telefon = '833080945', @NIP
= '8905182380'
EXEC Dodaj_firme @NazwaFirmy = 'Photospace', @Telefon = '893757289', @NIP
= '3549932863'
EXEC Dodaj_firme @NazwaFirmy = 'Trilia', @Telefon = '677268798', @NIP =
'9597368927'
EXEC Dodaj_firme @NazwaFirmy = 'Dabvine', @Telefon = '796215950', @NIP =
'9202799261'
EXEC Dodaj_firme @NazwaFirmy = 'Babbleblab', @Telefon = '257168132', @NIP
= '0348867395'
EXEC Dodaj_firme @NazwaFirmy = 'Yadel', @Telefon = '861296045', @NIP =
'0095808035'
EXEC Dodaj_firme @NazwaFirmy = 'Buzzbean', @Telefon = '678881229', @NIP =
'3862861295'

(...)

EXEC Dodaj_firme @NazwaFirmy = 'Skidoo', @Telefon = '425712580', @NIP =
'6956670267'
EXEC Dodaj_firme @NazwaFirmy = 'Reallinks', @Telefon = '726113293', @NIP =
'5964926886'
```

**--dodanie "modeli biznesowych" dla klientów indywidualnych**

```
SET @IDKlienta = (SELECT TOP 1 IDKlienta FROM KlienciIndywidualni WHERE
Nazwisko='Barnard Heasman')
SET @IDFirmy = (SELECT TOP 1 IDFirmy FROM Firmy WHERE NazwaFirmy =
'Blogpad')
EXEC Modyfikuj_firme_klienta @IDKlienta = @IDKlienta, @IDFirmy = @IDFirmy
SET @IDKlienta = (SELECT TOP 1 IDKlienta FROM KlienciIndywidualni WHERE
Nazwisko='Wendel Elijahu')
SET @IDFirmy = (SELECT TOP 1 IDFirmy FROM Firmy WHERE NazwaFirmy = 'Tazz')
EXEC Modyfikuj_firme_klienta @IDKlienta = @IDKlienta, @IDFirmy = @IDFirmy
SET @IDKlienta = (SELECT TOP 1 IDKlienta FROM KlienciIndywidualni WHERE
Nick='Cardify496')
SET @IDFirmy = (SELECT TOP 1 IDFirmy FROM Firmy WHERE NazwaFirmy =
'Cogilith')
EXEC Modyfikuj_firme_klienta @IDKlienta = @IDKlienta, @IDFirmy = @IDFirmy
SET @IDKlienta = (SELECT TOP 1 IDKlienta FROM KlienciIndywidualni WHERE
Nick='Fixflex870')
SET @IDFirmy = (SELECT TOP 1 IDFirmy FROM Firmy WHERE NazwaFirmy = 'Vitz')

(...)

EXEC Modyfikuj_firme_klienta @IDKlienta = @IDKlienta, @IDFirmy = @IDFirmy
SET @IDKlienta = (SELECT TOP 1 IDKlienta FROM KlienciIndywidualni WHERE
Nazwisko='Bernardina Maccree')
SET @IDFirmy = (SELECT TOP 1 IDFirmy FROM Firmy WHERE NazwaFirmy = 'Tavu')
EXEC Modyfikuj_firme_klienta @IDKlienta = @IDKlienta, @IDFirmy = @IDFirmy
```

**--dodawanie produktów do magazynu**

```
EXEC Dodaj_produkt @NazwaProduktu = 'kukurydza', @ilosc = 99999, @Jednostka
= kg
EXEC Dodaj_produkt @NazwaProduktu = 'oregano', @ilosc = 99999, @Jednostka =
kg
EXEC Dodaj_produkt @NazwaProduktu = 'czerwona cebula', @ilosc = 99999,
@Jednostka = kg
EXEC Dodaj_produkt @NazwaProduktu = 'boczek', @ilosc = 99999, @Jednostka =
kg

(...)

EXEC Dodaj_produkt @NazwaProduktu = 'kurczak', @ilosc = 99999, @Jednostka =
kg
```

**--dodanie dań i przepisów**

```
EXEC Dodaj_danie @nazwa='Margharita ', @opis='Niebiańska uczta', @cena=17.0,
@odKiedy='2020-08-06', @kategoria='pizza'
EXEC  Dodaj_do_przepisu  @nazwaPotrawy='Margharita  ',  @nazwaProduktu='sos
pomidorowy', @ilosc=1
EXEC  Dodaj_do_przepisu  @nazwaPotrawy='Margharita  ',  @nazwaProduktu='ser',
@ilosc=1
EXEC         Dodaj_do_przepisu         @nazwaPotrawy='Margharita         ',
@nazwaProduktu='oregano', @ilosc=1
EXEC  Dodaj_danie  @nazwa='Dino',  @opis='Najlepsze  w  mieście',  @cena=18.0,
@odKiedy='2020-11-26', @kategoria='pizza'
EXEC     Dodaj_do_przepisu     @nazwaPotrawy='Dino',     @nazwaProduktu='sos
pomidorowy', @ilosc=1
EXEC Dodaj_do_przepisu @nazwaPotrawy='Dino', @nazwaProduktu='ser', @ilosc=1
EXEC   Dodaj_do_przepisu   @nazwaPotrawy='Dino',   @nazwaProduktu='oregano',
@ilosc=1
```

```
(...)

EXEC Dodaj_do_przepisu @nazwaPotrawy='Uczta smakosza',
@nazwaProduktu='kieÅ□basa', @ilosc=1
EXEC Dodaj_do_przepisu @nazwaPotrawy='Uczta smakosza',
@nazwaProduktu='pomidor', @ilosc=1
EXEC Dodaj_do_przepisu @nazwaPotrawy='Uczta smakosza',
@nazwaProduktu='cebula', @ilosc=1

(...)
--dodanie zamówień (każde zawiera 3 rodzaje pizzy w ilości od 1 do 4 każda)
IF (312 IN (SELECT IDKlienta FROM Klienci) OR 312 IS NULL)
BEGIN
EXEC Rozpocznij_zamówienie @IDKlienta=312, @naKiedy='2020-10-28 01:45:40',
@dataZłożenia='2020-10-20 23:35:18', @czyNaMiejscu=1
SET @IDZamówienia =(SELECT TOP 1 IDZamówienia FROM Zamówienia WHERE
dataZłożenia = '2020-10-20 23:35:18')
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia, @nazwaPotrawy='Czesio
Pizza', @ilość=2
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia,
@nazwaPotrawy='Imperio', @ilość=2
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia,
@nazwaPotrawy='Margharita ', @ilość=2
END
IF (394 IN (SELECT IDKlienta FROM Klienci) OR 394 IS NULL)
BEGIN
EXEC Rozpocznij_zamówienie @IDKlienta=394, @naKiedy='2020-10-25 01:53:57',
@dataZłożenia='2020-07-04 04:29:51', @czyNaMiejscu=0
SET @IDZamówienia =(SELECT TOP 1 IDZamówienia FROM Zamówienia WHERE
dataZłożenia = '2020-07-04 04:29:51')
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia,
@nazwaPotrawy='Bussola', @ilość=1
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia,
@nazwaPotrawy='Kiler', @ilość=4
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia, @nazwaPotrawy='Szefa
Kuchni', @ilość=4
END

(...)

EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia,
@nazwaPotrawy='Diabelska', @ilość=1
EXEC Dodaj_do_zamówienia @IDZamówienia=@IDZamówienia, @nazwaPotrawy='Uczta
smakosza', @ilość=3
END

--Dodanie stolików
INSERT INTO Stoliki(LiczbaMiejsc) VALUES (3)
INSERT INTO Stoliki(LiczbaMiejsc) VALUES (9)

(...)

INSERT INTO Stoliki(LiczbaMiejsc) VALUES (10)
INSERT INTO Stoliki(LiczbaMiejsc) VALUES (2)

--dodanie rezerwacji firmowych
```

```
EXEC Dodaj_rezerwacje_dla_firmy @NazwaFirmy= 'Flashspan',@LiczbaOsób=NULL,
@OD='2020-04-23 00:37', @DO='2020-04-23 18:19'
EXEC Dodaj_rezerwacje_dla_firmy @NazwaFirmy= 'Devify',@LiczbaOsób=12,
@OD='2021-01-25 12:44', @DO='2021-01-25 23:42'
EXEC Dodaj_rezerwacje_dla_firmy @NazwaFirmy= 'Trudoo',@LiczbaOsób=6,
@OD='2020-08-28 03:52', @DO='2020-08-28 15:20'
EXEC Dodaj_rezerwacje_dla_firmy @NazwaFirmy= 'Zoombeat',@LiczbaOsób=NULL,
@OD='2020-06-11 21:21', @DO='2020-06-11 23:07'

(...)

EXEC Dodaj_rezerwacje_dla_firmy @NazwaFirmy= 'Oyondu',@LiczbaOsób=NULL,
@OD='2020-12-05 02:28', @DO='2020-12-05 10:27'

--dodanie rezerwacji klientow indywidualnych
IF (505 IN (SELECT IDZamówienia FROM Zamówienia))
BEGIN
EXEC Dokonaj_rezerwacji @IDZamówienia=505,@nazwisko='Isadora Tenney',
@liczbaOsob=4, @czyZaplataNaMiejscu=0, @rezerwacjaOD='2020-10-08
08:15',@rezerwacjaDO=DATEADD(MINUTE,45, @wygenerowanaData)
END
IF (2819 IN (SELECT IDZamówienia FROM Zamówienia))
BEGIN
EXEC Dokonaj_rezerwacji @IDZamówienia=2819,@nazwisko='Riannon Crispin',
@liczbaOsob=8, @czyZaplataNaMiejscu=1, @rezerwacjaOD='2020-04-25
20:12',@rezerwacjaDO=DATEADD(MINUTE,45, @wygenerowanaData)
END

(...)

IF (3251 IN (SELECT IDZamówienia FROM Zamówienia))
BEGIN
EXEC Dokonaj_rezerwacji @IDZamówienia=3251,@nazwisko='Ruthann Bedborough',
@liczbaOsob=8, @czyZaplataNaMiejscu=0, @rezerwacjaOD='2020-10-17
11:49',@rezerwacjaDO=DATEADD(MINUTE,45, @wygenerowanaData)
END
```

Wypełniając bazę wygenerowanymi danymi, sprawdziliśmy fundamentalne dla działania projektu procedury, funkcje i wyzwalacze.