PROJEKT BAZY DANYCH

Podstawy Baz Danych 2022/2023

Mateusz Śmigała, Grzegorz Liana, Martyna Sokołowska

SPIS TREŚCI

OPIS BAZY DANYCH

Opis użytkowników

- **pracownik**
  - **zwykły pracownik**

Wystawia faktury (dla klienta indywidualnego oraz zbiorcze raz na miesiąc), odbiera zamówienia dotyczące owoców morza, oraz je realizuje w odpowiednim czasie (importuje), odpowiada za potwierdzenie zamówienia rezerwacji stolika, ma dostęp do informacji rabatowej klienta, jest odpowiedzialny za generowanie raportów (miesięcznych i tygodniowych).

  - **menadżer**

Wybiera aktualne (w dany terminie) MENU, ustala progi rabatowe, może generować statystyki zamówień, posiada wszystkie uprawnienia zwykłego pracownika

  - **admin**

Nadzoruje bazę danych, oraz system

- **klient niezarejestrowany w systemie**

  Ma możliwość złożenia zamówienia

- **klient indywidualny**

  Ma możliwość złożenia zamówienia (na miejscu i na wynos), podgląd historii własnych zamówień, możliwość rezerwacji stolika, naliczają mu się rabaty, możliwość wygenerowania raportów

- **klient firmowy**

  Ma możliwość złożenia zamówienia (na miejscu i na wynos), może zażądać wystawienia faktury, możliwość rezerwacji stolika (firmowo), możliwość wygenerowania raportów

Funkcje użytkowników

- **pracownik**
  - **zwykły pracownik**

a) Odpowiada za wystawianie faktury, na życzenie klienta firmowego
b) Importuje owoce morza wchodzące w skład zamówień złożonych przed poniedziałkiem danego tygodnia
c) Nadzoruje system rezerwacji stolików, może wyświetlić listę wolnych stolików
d) Generuje raporty na życzenie klientów, oraz firm dotyczących kwot oraz czasu składania zamówień

e) Możliwość sprawdzenia historii zamówień klientów

❖ **menadżer**

f) Z listy dań wybiera, te które aktualnie znajdować się w MENU (wymienia co najmniej połowę z dań co najmniej raz na dwa tygodnie)
g) Możliwość dodawania i usuwania nowych dań do listy dań, oraz MENU
h) Odpowiada za wystawianie faktury, na życzenie klienta firmowego
i) Importuje owoce morza wchodzące w skład zamówień złożonych przed poniedziałkiem danego tygodnia
j) Nadzoruje system rezerwacji stolików, może wyświetlić listę wolnych stolików, modyfikacja kryteriów zamówienia stolika (WK,WZ)
k) Modyfikuje i nadzoruje rabaty dla klientów indywidualnych (Z1,K1,R1,K2,R2,D1)
l) Generuje raporty tygodniowe oraz miesięczne, dotyczących rezerwacji stolików, rabatów, menu
m) Generuje raporty na życzenie klientów, oraz firm dotyczących kwot oraz czasu składania zamówień
n) Możliwość sprawdzenia historii zamówień klientów
o) Zmienia ceny dań

● **klient niezarejestrowany w systemie**

a) Możliwość złożenia zamówienia w restauracji (na miejscu)

● **klient indywidualny**

a) Możliwość złożenia zamówienia w restauracji, oraz na wynos
b) Możliwość podglądu historii własnych zamówień
c) Możliwość rezerwacji stolika dla co najmniej dwóch osób, po spełnieniu określonych warunków (złożenie co najmniej WK zamówień), oraz złożeniu zamówienia na co najmniej WZ
d) Automatyczne naliczanie się rabatów przez system
e) Możliwość zapytania pracownika o wygenerowanie raportów dotyczących aktualnych rabatów, rezerwacji stolików, oraz historii zamówień

● **klient firmowy**

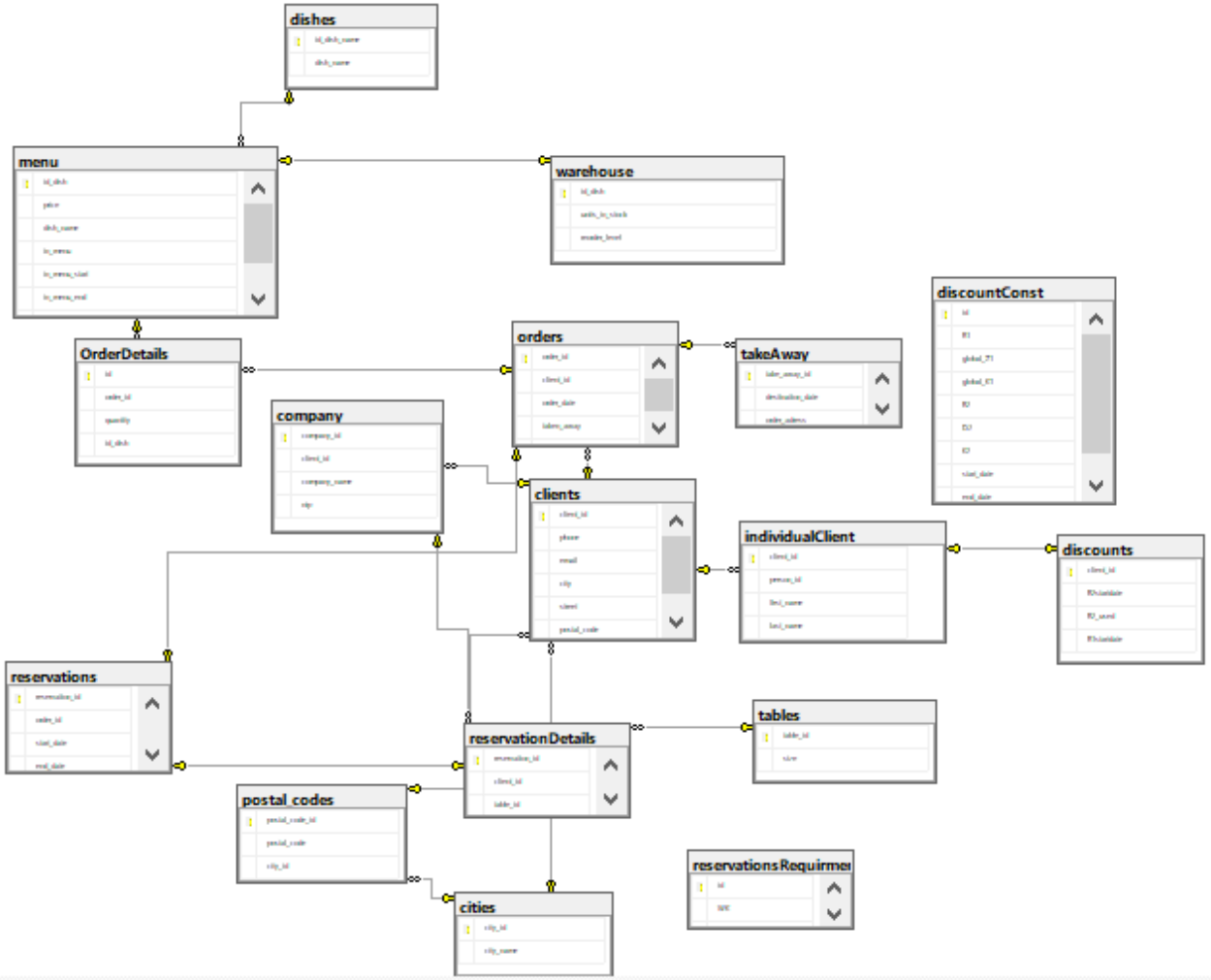
a) Możliwość złożenia zamówienia w restauracji, oraz na wynos
b) Możliwość wystawienia faktury (miesięcznej oraz pojedynczej) na życzenie u pracownika
c) Możliwość rezerwacji stolika, indywidualnie, oraz grupowo (dla firmy)
d) Możliwość zapytania pracownika o wygenerowanie raportów dotyczących rezerwacji stolików, oraz historii zamówień

Funkcje Systemowe

a) Obsługa internetowego formularza
b) Sprawdzenie czy klient spełnia odpowiednie warunki do rezerwacji stolika
c) Możliwość rezerwacji stolika
d) Wyliczanie oraz naliczanie rabatów podczas realizacji zamówień
e) Generowanie zdefiniowane raportów

f) Obliczanie sumarycznego kosztu zamówienia

SCHEMAT BAZY DANYCH



TABELE

Tabela clients

Przechowuje dane klientów.

Klucz główny: client_id
Numer telefonu: phone
Adres email: email
Miasto: city
Ulica: street
Kod pocztowy: postal_code

```
CREATE TABLE [dbo].[clients](
    [client_id] [int] NOT NULL,
    [phone] [varchar](12) NOT NULL,
    [email] [varchar](100) NOT NULL,
    [city] [varchar](100) NULL,
    [street] [varchar](100) NULL,
    [postal_code] [varchar](6) NULL,
 CONSTRAINT [PK_clients] PRIMARY KEY CLUSTERED
(
    [client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [unique_email] UNIQUE NONCLUSTERED
(
    [email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [unique_phone] UNIQUE NONCLUSTERED
(
    [phone] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Warunki integralności

- **[email] like '%@%'**
- **[phone] like '[0-9]{9}|[+][0-9]{11}'**
- **[postal_code] like '[0-9][0-9]-[0-9][0-9][0-9]'**
- **UNIQUE ([phone])**
- **UNIQUE ([email])**

Tabela cities

Słownikuje miasta

Klucz główny: city_id
Nazwa miasta: city_name

```
/****** Object:  Table [dbo].[cities]    Script Date: 17.12.2022 23:15:53 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[cities](
    [city_id] [int] NOT NULL,
    [city_name] [varchar](100) NULL,
 CONSTRAINT [PK_cities] PRIMARY KEY CLUSTERED
(
    [city_id] 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
```

Tabela company

Przechowuje dane firm.

Klucz główny: company_id
Numer klienta: client_id
Nazwa firmy: company_name
Nip: nip

```
/****** Object:  Table [dbo].[company]    Script Date: 17.12.2022 23:16:37 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[company](
    [company_id] [int] NOT NULL,
    [client_id] [int] NOT NULL,
    [company_name] [varchar](100) NOT NULL,
    [nip] [varchar](10) NOT NULL,
 CONSTRAINT [PK_company] PRIMARY KEY CLUSTERED
(
    [company_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [unique_nip] UNIQUE NONCLUSTERED
(
    [nip] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[company]  WITH CHECK ADD  CONSTRAINT [FK_company_clients] FOREIGN KEY([client_id])
REFERENCES [dbo].[clients] ([client_id])
GO

ALTER TABLE [dbo].[company] CHECK CONSTRAINT [FK_company_clients]
GO

ALTER TABLE [dbo].[company]  WITH CHECK ADD  CONSTRAINT [nip] CHECK  (([nip] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0
```

Warunki integralności

- **[nip] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'**
- **UNIQUE ([nip])**

Tabela discountConst

Przechowuje informacje o wymaganiach potrzebnych do zdobycia rabatów.

Klucz główny: id
Zmienna R1: R1
Zmienna Z1: global_Z1
Zmienna K1: global_K1
Zmienna R2: R2
Zmienna D2: D2
Zmienna K2: K2
Początek obowiązywania zniżek: start_date
Koniec obowiązywania zniżek: end_date

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[discountConst](
    [id] [int] NOT NULL,
    [R1] [int] NOT NULL,
    [global_Z1] [int] NOT NULL,
    [global_K1] [int] NOT NULL,
    [R2] [int] NOT NULL,
    [D2] [int] NOT NULL,
    [K2] [int] NOT NULL,
    [start_date] [datetime] NOT NULL,
    [end_date] [datetime] NULL,
 CONSTRAINT [PK_discountConst] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[discountConst]  WITH CHECK ADD  CONSTRAINT [correct_dates] CHECK  (([start_date]<[end_date]))
GO

ALTER TABLE [dbo].[discountConst] CHECK CONSTRAINT [correct_dates]
GO
```

Warunki integralności

- **[start_date] < [end_date]**
- **[D2] > 0**
- **[global_K1] > 0**
- **[global_Z1] > 0**
- **[K2] > 0**
- **[global_K1] > 0**
- **[R1] > 0**
- **[R2] > 0**

Tabela dishes

Słownikuje informacje o daniach

Klucz główny: id_dish_name
Nazwa dania: dish_name

```
/****** Object:  Table [dbo].[dishes]    Script Date: 17.12.2022 23:19:16 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[dishes](
    [id_dish_name] [int] NOT NULL,
    [dish_name] [nchar](20) NULL,
 CONSTRAINT [PK_dishes] PRIMARY KEY CLUSTERED
(
    [id_dish_name] 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
```

Tabela individualClient

Przechowuje dane klientów indywidualnych.

Klucz główny: client_id
Numer osoby: person_id
Imię klienta: first_name
Nazwisko klienta: last_name

```
/****** Object:  Table [dbo].[individualClient]    Script Date: 17.12.2022 23:24:32 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[individualClient](
    [client_id] [int] NOT NULL,
    [person_id] [int] NOT NULL,
    [first_name] [varchar](100) NOT NULL,
    [last_name] [varchar](100) NOT NULL,
 CONSTRAINT [PK_individualClient_1] PRIMARY KEY CLUSTERED
(
    [client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SE
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[individualClient]  WITH CHECK ADD  CONSTRAINT [FK_individualClient_clients1] FOREIGN KEY([person_id])
REFERENCES [dbo].[clients] ([client_id])
GO

ALTER TABLE [dbo].[individualClient] CHECK CONSTRAINT [FK_individualClient_clients1]
GO

ALTER TABLE [dbo].[individualClient]  WITH CHECK ADD  CONSTRAINT [FK_individualClient_discounts] FOREIGN KEY([client_id])
REFERENCES [dbo].[discounts] ([client_id])
GO
```

Tabela menu

Przechowuje informacje o daniach.

Klucz główny: id_dish
Cena: price
Nazwa pozycji: dish_name
Czy znajduję się aktualnie w menu: in_menu
Od kiedy pozycja znajduje się w menu: in_menu_start
Kiedy pozycja została usunięta z menu: in_menu_end
Opis: description
Czy danie jest z kategorii owoce morza: is_seafood

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[menu](
    [id_dish] [int] NOT NULL,
    [price] [money] NOT NULL,
    [dish_name] [int] NOT NULL,
    [in_menu] [bit] NOT NULL,
    [in_menu_start] [datetime] NOT NULL,
    [in_menu_end] [datetime] NOT NULL,
    [description] [varchar](100) NOT NULL,
    [is_seafood] [bit] NULL,
 CONSTRAINT [PK_menu] PRIMARY KEY CLUSTERED
(
    [id_dish] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[menu]  WITH CHECK ADD  CONSTRAINT [FK_menu_dishes] FOREIGN KEY([dish_name])
REFERENCES [dbo].[dishes] ([id_dish_name])
GO

ALTER TABLE [dbo].[menu] CHECK CONSTRAINT [FK_menu_dishes]
GO
```

Warunki integralności

- **[price] > 0**

Tabela OrderDetails

Przechowuje szczegóły zamówień.

Klucz główny: id
Ilość: quantity
Klucz obcy: id_dish
Klucz obcy 2: order_id

```
/****** Object:  Table [dbo].[OrderDetails]    Script Date: 17.12.2022 23:28:36 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[OrderDetails](
    [id] [int] NOT NULL,
    [order_id] [int] NOT NULL,
    [quantity] [int] NOT NULL,
    [id_dish] [int] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [FK_OrderDetails_menu1] FOREIGN KEY([id_dish])
REFERENCES [dbo].[menu] ([id_dish])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_OrderDetails_menu1]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [FK_OrderDetails_orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[orders] ([order_id])
GO
```

Warunki integralności

- **[quantity] >= 0**

Tabela orders

Przechowuje dane zamówień.

Klucz główny: order_id
Klucz obcy 1: client_id
Data zamówienia: order_date
Czy zamówienia zostało wydane na wynos: taken_away
Jeżeli zamawiamy owoce morza to na jaką datę: pref_date_of_seafood

```
CREATE TABLE [dbo].[orders](
    [order_id] [int] NOT NULL,
    [client_id] [int] NULL,
    [order_date] [datetime] NOT NULL,
    [taken_away] [bit] NOT NULL,
    [pref_date_of_seafood] [datetime] NULL,
 CONSTRAINT [PK_orders_1] PRIMARY KEY CLUSTERED
(
    [order_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SE
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[orders]  WITH CHECK ADD  CONSTRAINT [FK_orders_clients] FOREIGN KEY([client_id])
REFERENCES [dbo].[clients] ([client_id])
GO

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

ALTER TABLE [dbo].[orders]  WITH CHECK ADD  CONSTRAINT [dates] CHECK  (([order_date]>=getdate()))
GO
```

Warunki integralności

- **[order_date] > getdate()**

Tabela postal_codes

Słownikuje adresy pocztowa

Klucz główny: psotal_code_id
Klucz obcy 1: city_id
Kod pocztowy: postal_code

```
/****** Object:  Table [dbo].[postal_codes]    Script Date: 17.12.2022 23:30:52 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[postal_codes](
    [postal_code_id] [int] NOT NULL,
    [postal_code] [varchar](6) NOT NULL,
    [city_id] [int] NOT NULL,
 CONSTRAINT [PK_postal_codes] PRIMARY KEY CLUSTERED
(
    [postal_code_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZ
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[postal_codes]  WITH CHECK ADD  CONSTRAINT [FK_postal_codes_cities] FOREIGN KEY([city_id])
REFERENCES [dbo].[cities] ([city_id])
GO

ALTER TABLE [dbo].[postal_codes] CHECK CONSTRAINT [FK_postal_codes_cities]
GO

ALTER TABLE [dbo].[postal_codes]  WITH CHECK ADD  CONSTRAINT [CK_postal_codes] CHECK  (([postal_code] like '[0-9][0-9]-[0-9][0-9
GO

ALTER TABLE [dbo].[postal_codes] CHECK CONSTRAINT [CK_postal_codes]
GO
```

Warunki integralności

- **[postal_code] like '[0-9][0-9]-[0-9][0-9][0-9]'**

Tabela reservationDetails

Przechowuje szczegóły rezerwacji.

Klucz główny: reservation_id
Klucz obcy 1: client_id
Klucz obcy 2: table_id

```
/****** Object:  Table [dbo].[reservationDetails]    Script Date: 17.12.2022 23:33:19 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[reservationDetails](
    [reservation_id] [int] NOT NULL,
    [client_id] [int] NOT NULL,
    [table_id] [int] NOT NULL,
 CONSTRAINT [PK_reservationDetails] PRIMARY KEY CLUSTERED
(
    [reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[reservationDetails]  WITH CHECK ADD  CONSTRAINT [FK_reservationDetails_company] FOREIGN KEY([client_id])
REFERENCES [dbo].[company] ([company_id])
GO

ALTER TABLE [dbo].[reservationDetails] CHECK CONSTRAINT [FK_reservationDetails_company]
GO

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

ALTER TABLE [dbo].[reservationDetails] CHECK CONSTRAINT [FK_reservationDetails_tables]
```

Tabela reservations

Przechowuje dane rezerwacji.

Klucz główny: reservation_id
Klucz obcy: order_id
Data początku rezerwacji: start_date
Data końca rezerwacji: end_date
Zatwierdzenie rezerwacji: approved
Pojemność stolika: amount_people

```
/****** Object:  Table [dbo].[reservations]    Script Date: 17.12.2022 23:34:02 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[reservations](
    [reservation_id] [int] NOT NULL,
    [order_id] [int] NOT NULL,
    [start_date] [datetime] NOT NULL,
    [end_date] [datetime] NOT NULL,
    [approved] [bit] NOT NULL,
    [amount_people] [int] NOT NULL,
 CONSTRAINT [PK_reservations] PRIMARY KEY CLUSTERED
(
    [reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE
) ON [PRIMARY]
GO

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

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

ALTER TABLE [dbo].[reservations]  WITH CHECK ADD  CONSTRAINT [FK_reservations_reservationDetails] FOREIGN KEY([reservation_id])
REFERENCES [dbo].[reservationDetails] ([reservation_id])
```

Warunki integralności

- **[amount_people] > 1**
- **[start_date] < [end_date]**

Tabela reservationsRequirments

Przechowuje informacje o wymaganiach potrzebnych do możliwości wykonania rezerwacji przez klienta indywidualnego.

Klucz główny: id
Zmienna WK: WK
Zmienna WZ: WZ

```
/****** Object:  Table [dbo].[reservationsRequirments]    Script Date: 17.12.2022 23:35:44 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[reservationsRequirments](
    [id] [int] NOT NULL,
    [WK] [int] NOT NULL,
    [WZ] [int] NOT NULL,
 CONSTRAINT [PK_reservationsRequirments] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FO
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[reservationsRequirments]  WITH CHECK ADD  CONSTRAINT [WK] CHECK  (([WK]>(0)))
GO

ALTER TABLE [dbo].[reservationsRequirments] CHECK CONSTRAINT [WK]
GO

ALTER TABLE [dbo].[reservationsRequirments]  WITH CHECK ADD  CONSTRAINT [WZ] CHECK  (([WZ]>(0)))
GO

ALTER TABLE [dbo].[reservationsRequirments] CHECK CONSTRAINT [WZ]
GO
```

Warunki integralności

- **[WK] > 0**
- **[WZ] > 0**

Tabela tables

Przechowuje informacje o stolikach.

Klucz główny: table_id
Pojemność stolika: size

```
CREATE TABLE [dbo].[tables](
    [table_id] [int] NOT NULL,
    [size] [int] NOT NULL,
 CONSTRAINT [PK_tables] PRIMARY KEY CLUSTERED
(
    [table_id] 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
```

Warunki integralności

- **[size] > 1**

Tabela takeAway

Przechowuje informacje o zamówieniach na wynos.

Klucz główny: take_away_id
Docelowa data zamówienia: destination_date
Docelowy adres zamówienia: order_adress
Klucz obcy: order_id

```
CREATE TABLE [dbo].[takeAway](
    [take_away_id] [int] NOT NULL,
    [destination_date] [datetime] NOT NULL,
    [order_adress] [varchar](50) NOT NULL,
    [order_id] [int] NULL,
 CONSTRAINT [PK_takeAway] PRIMARY KEY CLUSTERED
(
    [take_away_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[takeAway]  WITH CHECK ADD  CONSTRAINT [FK_takeAway_orders] FOREIGN KEY([order_id])
REFERENCES [dbo].[orders] ([order_id])
GO
```

Warunki integralności

- **[destination_date] > getdate()**

Tabela warehouse

Przechowuje informacje o ilości produktów w magazynie.

Klucz główny: id_dish
Stan danego towaru w magazynie: units_in_stock
Stan poniżej, którego należy złożyć u dostawców: reorder_level

```
CREATE TABLE [dbo].[warehouse](
    [id_dish] [int] NOT NULL,
    [units_in_stock] [int] NOT NULL,
    [reorder_level] [int] NOT NULL,
 CONSTRAINT [PK_warehouse] PRIMARY KEY CLUSTERED
(
    [id_dish] 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
```

Warunki integralności

- **[reorder_level] >= 0**
- **[units_in_stock] >= 0**

WIDOKI

Widok AllDishes

Wyświetla nazwy wszystkich dań.

```
CREATE VIEW [dbo].[AllDishes]
AS
SELECT        dbo.dishes.dish_name, dbo.menu.description, dbo.menu.price
FROM            dbo.dishes INNER JOIN
                         dbo.menu ON dbo.dishes.id_dish_name = dbo.menu.dish_name
GO
```

Widok ClientOrderedXTimes

Listuje klientów oraz ilość złożonych przez nich zamówień

```
CREATE VIEW [dbo].[ClientOrderedXTimes]
AS
SELECT        client_id, COUNT(*) AS Expr1
FROM            dbo.orders
GROUP BY client_id
GO
```

Widok ClientStats

Wyświetla statystyki klienta. Łączną wartość jego zamówień.

```
CREATE VIEW [dbo].[ClientStats]
AS
SELECT        dbo.clients.client_id, ISNULL((SUM(dbo.OrderDetails.quantity * dbo.menu.price) *
(100 - ISNULL(dbo.ClientDiscountInfoR1.R1, 0)) / 100) * (100 - ISNULL(dbo.ClientDiscountInfo.R2, 0)) / 100, 0) AS Total,
                         dbo.ClientDiscountInfoR1.R1, dbo.ClientDiscountInfo.R2
FROM            dbo.clients LEFT OUTER JOIN
                         dbo.ClientDiscountInfo ON dbo.clients.client_id = dbo.ClientDiscountInfo.client_id LEFT OUTER JOIN
                         dbo.ClientDiscountInfoR1 ON dbo.clients.client_id = dbo.ClientDiscountInfoR1.client_id LEFT OUTER JOI
                         dbo.orders ON dbo.clients.client_id = dbo.orders.client_id LEFT OUTER JOIN
                         dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id LEFT OUTER JOIN
                         dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish
GROUP BY dbo.clients.client_id, dbo.ClientDiscountInfoR1.R1, dbo.ClientDiscountInfo.R2
GO
```

Widok CurrentMenu

Wyświetla aktualne menu.

```
CREATE VIEW [dbo].[CurrentMenu]
AS
SELECT        dbo.dishes.dish_name, dbo.menu.price
FROM            dbo.menu INNER JOIN
                         dbo.dishes ON dbo.menu.dish_name = dbo.dishes.id_dish_name
WHERE        (dbo.menu.in_menu = 1)
GO
```

Widok OrdersInfo

Wyświetla informacje dotyczące zamówienia. Klienta, który je złożył oraz jego wartość.

```
CREATE VIEW [dbo].[OrdersInfo]
AS
SELECT        TOP (100) PERCENT dbo.orders.order_id, dbo.orders.client_id, (SUM(dbo.menu.price * dbo.OrderDetails.quantity)
* (100 - ISNULL(dbo.ClientDiscountInfoR1.R1, 0)) / 100) * (100 - ISNULL(dbo.ClientDiscountInfo.R2, 0))
                         / 100 AS TotalPrice, dbo.ClientDiscountInfoR1.R1, dbo.ClientDiscountInfo.R2
FROM            dbo.orders INNER JOIN
                         dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                         dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish INNER JOIN
                         dbo.clients ON dbo.orders.client_id = dbo.clients.client_id LEFT OUTER JOIN
                         dbo.ClientDiscountInfo ON dbo.clients.client_id = dbo.ClientDiscountInfo.client_id LEFT OUTER JOIN
                         dbo.ClientDiscountInfoR1 ON dbo.clients.client_id = dbo.ClientDiscountInfoR1.client_id
GROUP BY dbo.orders.order_id, dbo.orders.client_id, dbo.ClientDiscountInfoR1.R1, dbo.ClientDiscountInfo.R2
ORDER BY dbo.orders.order_id
GO
```

Widok Seafood

Widok pokazuje dania które są zaliczane jako ‘owoce morza’

```
CREATE VIEW [dbo].[Seafood]
AS
SELECT        id_dish, price
FROM            dbo.menu AS w
WHERE        (is_seafood = 1)
GO
```

Widok SoldDishes

Wyświetla informacje ile jednostek dania zostało sprzedane.

```
CREATE VIEW [dbo].[SoldDishes]
AS
SELECT        dbo.dishes.dish_name, SUM(dbo.OrderDetails.quantity) AS TotalQuantity
FROM            dbo.OrderDetails INNER JOIN
                         dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish INNER JOIN
                         dbo.dishes ON dbo.menu.dish_name = dbo.dishes.id_dish_name
GROUP BY dbo.dishes.dish_name
GO
```

Widok ClientDiscountInfo

Informacja o jednorazowej zniżce klienta. Data rozpoczęcia czasu na wykorzystanie, informacja o jej wykorzystaniu/bądź nie oraz imię i nazwisko klienta.

```
CREATE VIEW [dbo].[ClientDiscountInfo]
AS
SELECT dbo.individualClient.client_id, dbo.discounts.R2startdate, DATEADD(day, dbo.discountConst.D2,
dbo.discounts.R2startdate) AS R2enddate, dbo.discounts.R2_used, dbo.individualClient.first_name,
dbo.individualClient.last_name,
                  dbo.discountConst.R2
FROM     dbo.individualClient INNER JOIN
                  dbo.discounts ON dbo.individualClient.client_id = dbo.discounts.client_id CROSS JOIN
                  dbo.discountConst
WHERE  (dbo.discounts.R2_used = 1)
GO
```

Widok ClientDiscountInfoR1

Informacje na temat bezterminowych zniżek klientów indywidualnych.

```
CREATE VIEW [dbo].[ClientDiscountInfoR1]
AS
SELECT dbo.individualClient.client_id, dbo.individualClient.first_name, dbo.individualClient.last_name,
dbo.discounts.R1startdate, dbo.discountConst.R1
FROM     dbo.discounts INNER JOIN
                  dbo.individualClient ON dbo.discounts.client_id = dbo.individualClient.client_id CROSS JOIN
                  dbo.discountConst
WHERE  (GETDATE() > dbo.discounts.R1startdate)
GO
```

Widok MonthOrderStatistics

Wyświetla ilość zamówień oraz ich wartość ze względu na rok i miesiące.

```
CREATE VIEW [dbo].[MonthOrderStatistic]
AS
SELECT        YEAR(dbo.orders.order_date) AS year, MONTH(dbo.orders.order_date) AS month,
COUNT(*) AS order_count, SUM(dbo.menu.price * dbo.OrderDetails.quantity) AS order_value
FROM            dbo.orders INNER JOIN
                         dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                         dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish
GROUP BY YEAR(dbo.orders.order_date), MONTH(dbo.orders.order_date)
GO
```

Widok NotApprovedReservations

Wyświetla jeszcze nie potwierdzone rezerwacje.

```
CREATE VIEW [dbo].[NotApprovedReservations]
AS
SELECT        reservation_id, order_id, start_date, end_date, approved, amount_people
FROM            dbo.reservations
WHERE        (approved = 0)
GO
```

Widok TablesReservationStats

Wyświetla statystyki rezerwacji stolików.

```
CREATE VIEW [dbo].[TablesReservationStats]
AS
SELECT        YEAR(dbo.reservations.start_date) AS year, MONTH(dbo.reservations.start_date) AS month,
dbo.reservationDetails.table_id, COUNT(dbo.reservationDetails.table_id) AS ReservationCounter
FROM            dbo.reservationDetails INNER JOIN
                         dbo.reservations ON dbo.reservationDetails.reservation_id = dbo.reservations.reservation_id
GROUP BY YEAR(dbo.reservations.start_date), MONTH(dbo.reservations.start_date), dbo.reservationDetails.table_id
GO
```

Widok NotYetDeliveredTakeaways

Wyświetla informacje odnośnie nie dostarczonych jeszcze zamówień.

```
CREATE VIEW [dbo].[NotYetDeliveredTakeaways]
AS
SELECT        dbo.orders.order_id, dbo.orders.client_id
FROM            dbo.takeAway INNER JOIN
                         dbo.orders ON dbo.takeAway.order_id = dbo.orders.order_id
WHERE        (dbo.takeAway.destination_date > GETDATE())
GO
```

Widok clientsInfo

Wyświetla informacje o zamówieniach klientów liczbę zamówionych produktów i ich cenę.

```
CREATE VIEW [dbo].[clientsInfo]
AS
SELECT DISTINCT
                dbo.companyClientsInfo.client_id, dbo.companyClientsInfo.quantity, dbo.companyClientsInfo.price
FROM     dbo.companyClientsInfo CROSS JOIN
                dbo.individualClientInfo
union
SELECT DISTINCT
                dbo.companyClientsInfo.client_id, dbo.individualClientInfo.quantity,
                dbo.individualClientInfo.price
FROM     dbo.companyClientsInfo CROSS JOIN
                dbo.individualClientInfo
GO
```

Widok monthlyInvoice

Wyświetla informacje dla miesięcznych oraz rocznych (dotyczących zamówień z każdego miesiąca/roku) faktur dla firm.

```
CREATE VIEW [dbo].[monthlyInvoice]
AS
SELECT        dbo.company.nip, dbo.company.company_name, SUM(dbo.menu.price * dbo.OrderDetails.quantity) AS [total pri
MONTH(dbo.orders.order_date) AS month, YEAR(dbo.orders.order_date) AS year, dbo.OrderDetails.quantity,
                         dbo.dishes.dish_name
FROM            dbo.clients INNER JOIN
                         dbo.company ON dbo.clients.client_id = dbo.company.client_id INNER JOIN
                         dbo.orders ON dbo.clients.client_id = dbo.orders.client_id INNER JOIN
                         dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                         dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish INNER JOIN
                         dbo.dishes ON dbo.menu.dish_name = dbo.dishes.id_dish_name LEFT OUTER JOIN
                         dbo.individualClient ON dbo.clients.client_id = dbo.individualClient.person_id
GROUP BY MONTH(dbo.orders.order_date), YEAR(dbo.orders.order_date), dbo.company.nip, dbo.company.company_name,
dbo.OrderDetails.quantity, dbo.dishes.dish_name
GO
```

Widok companyClientInfo

Informacje na temat klientów firmowych.

```
CREATE VIEW [dbo].[companyClientsInfo]
AS
SELECT DISTINCT dbo.clients.client_id, dbo.OrderDetails.quantity, dbo.menu.price
FROM     dbo.clients INNER JOIN
                  dbo.orders ON dbo.clients.client_id = dbo.orders.client_id INNER JOIN
                  dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                  dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish INNER JOIN
                  dbo.company ON dbo.clients.client_id = dbo.company.client_id
GO
```

Widok individualClientInfo

Informacje na temat klientów indywidualnych.

```
CREATE VIEW [dbo].[individualClientInfo]
AS
SELECT DISTINCT dbo.clients.client_id, dbo.OrderDetails.quantity, dbo.menu.price
FROM     dbo.clients INNER JOIN
                  dbo.orders ON dbo.clients.client_id = dbo.orders.client_id INNER JOIN
                  dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                  dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish INNER JOIN
                  dbo.individualClient ON dbo.clients.client_id = dbo.individualClient.person_id
GO
```

Widok individualClientStats

Wyświetla statystyki dla klientów indywidualnych, ich całkowitą sumę i zniżki.

```
CREATE VIEW [dbo].[individualClientStats]
AS
SELECT dbo.clients.client_id, ISNULL(SUM(dbo.OrderDetails.quantity * dbo.menu.price) * (100 - ISNULL(dbo.ClientDiscountInfoR1.R1, 0)) / 100, 0)
AS Total, dbo.ClientDiscountInfoR1.R1
FROM     dbo.clients LEFT OUTER JOIN
                  dbo.ClientDiscountInfoR1 ON dbo.clients.client_id = dbo.ClientDiscountInfoR1.client_id LEFT OUTER JOIN
                  dbo.orders ON dbo.clients.client_id = dbo.orders.client_id LEFT OUTER JOIN
                  dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id LEFT OUTER JOIN
                  dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish
GROUP BY dbo.clients.client_id, dbo.ClientDiscountInfoR1.R1
GO
```

Widok menuDates

Wyświetla menu, które było aktualne konkretnego dnia.

```
CREATE VIEW [dbo].[menuDates]
AS
SELECT dish_name, price, in_menu_start, in_menu_end
FROM     dbo.menu
WHERE  (in_menu_start < '2023/10/12') AND ('2023/10/12' < in_menu_end)
GO
```

Widok discountUsed

Wyświetla zniżki które zostały już użyte.

```
CREATE VIEW [dbo].[discountUsed]
AS
SELECT dbo.orders.order_id, dbo.orders.client_id, dbo.orders.order_date, dbo.ClientDiscountInfoR1.R1, dbo.ClientDiscountInfo.R
FROM     dbo.orders INNER JOIN
                  dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                  dbo.clients ON dbo.orders.client_id = dbo.clients.client_id INNER JOIN
                  dbo.individualClient ON dbo.clients.client_id = dbo.individualClient.person_id LEFT OUTER JOIN
                  dbo.ClientDiscountInfo ON dbo.clients.client_id = dbo.ClientDiscountInfo.client_id LEFT OUTER JOIN
                  dbo.ClientDiscountInfoR1 ON dbo.clients.client_id = dbo.ClientDiscountInfoR1.client_id
GO
```

Widok singleInvoice

Wyświetla informacje dla pojedynczych (dotyczących jednego zamówienia) faktur dla firm.

```
--
CREATE VIEW [dbo].[singleInvoice]
AS
SELECT dbo.company.nip, dbo.company.company_name, dbo.OrderDetails.quantity, dbo.menu.price, dbo.menu.price AS [total price]
FROM     dbo.clients INNER JOIN
                  dbo.company ON dbo.clients.client_id = dbo.company.client_id INNER JOIN
                  dbo.orders ON dbo.clients.client_id = dbo.orders.client_id INNER JOIN
                  dbo.OrderDetails ON dbo.orders.order_id = dbo.OrderDetails.order_id INNER JOIN
                  dbo.menu ON dbo.OrderDetails.id_dish = dbo.menu.id_dish LEFT OUTER JOIN
                  dbo.individualClient ON dbo.clients.client_id = dbo.individualClient.person_id
GO
```

PROCEDURY

Procedura addDish

Dodaje nowe danie do tabeli Dishes.

```
CREATE PROCEDURE [dbo].[addDish] @dish_name nchar(20)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;


    -- Insert statements for procedure here
    BEGIN TRY
        IF EXISTS(
        SELECT *
        FROM dishes
        WHERE dish_name = @dish_name
        )
        BEGIN
        ;
        THROW 52000, 'Danie jest już w tabeli', 1
        END

        DECLARE @id_dish_name INT
        SELECT @id_dish_name = ISNULL(MAX(id_dish_name), 0) + 1
        FROM dishes
        INSERT INTO dishes(id_dish_name,dish_name)
        VALUES (@id_dish_name,@dish_name);
        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        ='Error: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
        END CATCH
END
GO
```

Procedura addProductToMenu

Procedura przyjmuje informację o daniu i dodaje je do aktualnego menu.

```
ALTER PROCEDURE [dbo].[addProductToMenu] @price money,@in_menu bit,@in_menu_start datetime,
@in_menu_end datetime,@desc varchar(100),@name_of_dish nchar(20),@reorder_level int,@units_in_stock int,@is_seafood Bit
AS
BEGIN
    SET NOCOUNT ON;
                BEGIN TRY
            IF NOT EXISTS(
            SELECT *
            FROM dishes
            WHERE @name_of_dish = dish_name
            )
            BEGIN
            ;
            THROW 52000, 'Nie ma takiego dania', 1
            END
            DECLARE @dish_name INT
            SELECT @dish_name = id_dish_name
            FROM dishes
            WHERE dish_name = @name_of_dish

            Declare @id_dish INT
            Select  @id_dish=ISNULL(MAX(id_dish), 0) + 1
            From menu
            INSERT INTO warehouse(id_dish,units_in_stock,reorder_level)
            VALUES (@id_dish,@units_in_stock,@reorder_level);
            INSERT INTO menu(id_dish,price,dish_name,in_menu,in_menu_start,in_menu_end,description,is_seafood)
            VALUES (@id_dish, @price, @dish_name, @in_menu, @in_menu_start,@in_menu_end,@desc,@is_seafood);
            END TRY
            BEGIN CATCH
            DECLARE @msg nvarchar(2048)
            =N'Error:' + ERROR_MESSAGE();
            THROW 52000, @msg, 1
            END CATCH


END
```

Procedura addCompanyClient

Procedura przyjmuje informacje o kliencie firmowym(nazwa, nip, telefon, mail oraz adres) i dodaje je do tabeli clients oraz company.

```
CREATE PROCEDURE [dbo].[addCompanyClient] @company_name varchar(100),@nip varchar(10),@phone varchar(12),@email varchar(255),
@city_name varchar(100),@street varchar(255),@postal_code_ varchar(6)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    BEGIN TRY

            IF EXISTS(
            SELECT *
            FROM company
            WHERE nip=@nip
            )
            BEGIN;
            THROW 52000, 'Już jest taka firma', 1
            END

                IF Not EXISTS(
            SELECT *
            FROM cities
            WHERE city_name=@city_name
            )
            BEGIN;
            Declare @id_city INT
            Select  @id_city=ISNULL(MAX(city_id), 0) + 1
            From cities
            INSERT INTO cities(city_id,city_name)
            VALUES (@id_city,@city_name);
            Declare @id_postal_code INT
            Select  @id_postal_code=ISNULL(MAX(postal_code_id), 0) + 1
            From postal_codes
            INSERT INTO postal_codes(postal_code_id,postal_code,city_id)
            VALUES (@id_postal_code,@postal_code_,@id_city);
            END

            DECLARE @city_id INT
            SELECT @city_id = city_id
            FROM cities
            WHERE city_name = @city_name

            DECLARE @postal_code_id INT
            SELECT @postal_code_id = postal_code_id
            FROM postal_codes
            WHERE @postal_code_=postal_code and city_id=@city_id

            DECLARE @company_id INT
            Select  @company_id=ISNULL(MAX(company_id), 0) + 1
            FROM company

        DECLARE @client_id INT
        Select  @client_id=ISNULL(MAX(client_id), 0) + 1
        FROM clients

        INSERT INTO company(company_id,client_id,company_name,nip)
        VALUES (@company_id,@client_id,@company_name,@nip);

        INSERT INTO clients(client_id,phone,email,city,street,postal_code)
        VALUES (@client_id,@phone,@email,@city_id,@street,@postal_code_id);

        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

    -- Insert statements for procedure here

END
GO
```

Procedura addIndividualClient

Procedura dodaje do tabel clients i individualClient dane informacje na temat nowego klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[addIndividualClient] @firstname varchar(100),@lastname varchar(100),@phone varchar(12),@email varchar(255),
@city_name varchar(100),@street varchar(255),@postal_code_ varchar(6)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    BEGIN TRY

            IF EXISTS(
            SELECT *
            FROM clients
            WHERE @phone=phone
            )
            BEGIN;
            THROW 52000, 'Już jest taki klient', 1
            END
                IF Not EXISTS(
            SELECT *
            FROM cities
            WHERE city_name=@city_name
            )
            BEGIN;
            Declare @id_city INT
            Select  @id_city=ISNULL(MAX(city_id), 0) + 1
            From cities
            INSERT INTO cities(city_id,city_name)
            VALUES (@id_city,@city_name);
            Declare @id_postal_code INT
            Select  @id_postal_code=ISNULL(MAX(postal_code_id), 0) + 1
            From postal_codes
            INSERT INTO postal_codes(postal_code_id,postal_code,city_id)
            VALUES (@id_postal_code,@postal_code_,@id_city);
            END

            DECLARE @city_id INT
            SELECT @city_id = city_id
            FROM cities
            WHERE city_name = @city_name

            DECLARE @postal_code_id INT
            SELECT @postal_code_id = postal_code_id
            FROM postal_codes
            WHERE @postal_code_=postal_code and city_id=@city_id

            DECLARE @person_id INT
            Select  @person_id=ISNULL(MAX(client_id), 0) + 1
            FROM clients
DECLARE @client_id INT
Select  @client_id=ISNULL(MAX(client_id), 0) + 1
FROM individualClient

INSERT INTO discounts(client_id,R2startdate,R2_used,R1startdate)
VALUES (@client_id,NULL,NULL,NULL);

INSERT INTO clients(client_id,phone,email,city,street,postal_code)
VALUES (@person_id,@phone,@email,@city_id,@street,@postal_code_id);

INSERT INTO individualClient(client_id,person_id,first_name,last_name)
VALUES (@client_id,@person_id,@firstname,@lastname);



END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error:' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
GO
```

Procedura addOrder

Procedura otrzymuje dane dotyczące zamówienia i dodaje zamówienie do tabeli orders.

```
ALTER PROCEDURE  [dbo].[addOrder] @client_id int,@taken_away bit,@destination_date datetime,@order_adress varchar(50),@quantity int,@dish_name nchar(20)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

            BEGIN TRY
                DECLARE @OrderID INT
                SELECT @OrderID = ISNULL(MAX(order_id), 0) + 1
                FROM Orders
                INSERT INTO Orders(order_id, client_id, order_date, taken_away)
                VALUES (@OrderID, @client_id,GETDATE(),@taken_away)
             execute addProductToOrder @OrderId , @quantity , @dish_name

                if(@taken_away=1)
                    Begin
                        DECLARE @takeaway_id INT
                        SELECT @takeaway_id = ISNULL(MAX(take_away_id), 0) + 1
                        FROM takeAway
                        INSERT INTO takeAway(take_away_id,destination_date,order_adress,order_id)
                        VALUES (@takeaway_id,@destination_date,@order_adress,@OrderID)
                    END


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


END
```

Procedura ApproveReservation

Procedura potwierdzająca rezerwację o podanym id.

```
CREATE PROCEDURE [dbo].[ApproveReservation]

    @ReservationId int

AS
BEGIN

    SET NOCOUNT ON;
            BEGIN TRY
        IF NOT EXISTS(
        SELECT * FROM reservations
        WHERE reservation_id = @ReservationId)
        BEGIN;
        THROW 52000, 'Nie ma takiej rezerwacji',1
        END

        UPDATE reservations
        SET approved = 1
        WHERE reservation_id = @ReservationId;
        END TRY

        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH


END
GO
```

Procedura addProductToOrder

Procedura dodaje produkt daną liczbę produktów, po id produktu do danego zamówienia.

```
-----------------------------------------
CREATE PROCEDURE [dbo].[addProductToOrder] @orderId int,@quantity int,@dish_name nchar(20)

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
          SET NOCOUNT ON;
          BEGIN TRY
        IF NOT EXISTS(
        SELECT *
        FROM menu
        inner join dishes on dishes.id_dish_name=menu.dish_name
        WHERE dishes.dish_name=@dish_name
        )
        BEGIN
        ;
        THROW 52000, 'Nie ma takiego dania', 1
        END
        IF NOT EXISTS(
        SELECT *
        FROM Orders
        WHERE order_id = @OrderID
        )
        BEGIN
        ;
        THROW 52000, 'Nie ma takiego zamowienia', 1
        END

        Declare @id INT
        SELECT @id = ISNULL(MAX(id), 0) + 1
        FROM OrderDetails

        Declare @id_dish INT
        SELECT @id_dish=menu.id_dish
        FROM menu
        inner join dishes on dishes.id_dish_name=menu.dish_name
        WHERE dishes.dish_name=@dish_name

        INSERT INTO OrderDetails(id,order_id,quantity,id_dish)
        VALUES (@id,@orderId,@quantity,@id_dish)


        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Błąd dodania produktu do zamowienia: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

    -- Insert statements for procedure here
END
GO
```

Procedura addSeaFoodOrder

Procedura dodaje zamówienia na owoce morza.

```
ALTER PROCEDURE [dbo].[addSeafoodOrder] @client_id int,@taken_away bit,@destination_date datetime,@order_adress varchar(50),
@quantity int,@dish_name nchar(20),@pref_date_seafood datetime
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

       BEGIN TRY
                IF NOT EXISTS(
                SELECT *
                FROM menu
                Inner join dishes as d on d.id_dish_name=menu.dish_name
                WHERE @dish_name=d.dish_name and is_seafood=1
                )
                BEGIN
                ;
                THROW 52000, 'To nie są owoce morza', 1
                end

                    DECLARE @OrderID INT
                    SELECT @OrderID = ISNULL(MAX(order_id), 0) + 1
                    FROM Orders
                    INSERT INTO Orders(order_id, client_id, order_date, taken_away,pref_date_of_seafood)
                    VALUES (@OrderID, @client_id,GETDATE(),@taken_away,@pref_date_seafood)
                    execute addProductToOrder @OrderId , @quantity , @dish_name

                    if(@taken_away=1)
                        Begin
                            DECLARE @takeaway_id INT
                            SELECT @takeaway_id = ISNULL(MAX(take_away_id), 0) + 1
                            FROM takeAway
                            INSERT INTO takeAway(take_away_id,destination_date,order_adress,order_id)
                            VALUES (@takeaway_id,@destination_date,@order_adress,@OrderID)
                        END


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

Procedura addReservation_CompanyClient

Procedura dodaje rezerwacje otrzymując dane: id klienta, ilość osób, datę rozpoczęcia i zakończenia oraz informację o potwierdzeniu rezerwacji,nazwie dania i jego ilości .

```
ALTER PROCEDURE [dbo].[addReservation_CompanyClient] @client_id int, @amount_people INT,
@start_date datetime,@end_date datetime,@approved bit,@quantity int,@dish_name nchar(20)

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

   BEGIN TRY
                IF NOT EXISTS(
                SELECT *
                FROM Clients
                WHERE client_id = @client_id
                )
                BEGIN
                ;
                THROW 52000, 'Nie ma takiego klienta', 1
                end

                if @start_date is null or @end_date is null
                begin;
                throw 52000, 'trzeba podac obie daty',2
                end

                IF @start_date<GETDATE()
                Begin;
                throw 52000, 'rezerwacja na przeszłość', 3
                end

                DECLARE @ReservationID INT
                SELECT @ReservationID = ISNULL(MAX(reservation_id), 0) + 1
                FROM reservations

                DECLARE @orderID INT
                SELECT @orderID = ISNULL(MAX(order_id), 0) + 1
                FROM orders
```

```
Select top 1 @table_id=table_id
From tables
Where size>=@amount_people

Declare @is_seafood bit
Select @is_seafood=is_seafood from menu as m
inner join dishes as d on d.id_dish_name=m.dish_name
Where @dish_name=d.dish_name

IF @is_seafood=0
Begin
execute addOrder @client_id,0,null,null,@quantity,@dish_name
End

IF @is_seafood=1
Begin
execute addSeafoodOrder @client_id,0,null,null,@quantity,@dish_name,@start_date
End


INSERT INTO reservationDetails(reservation_id, client_id,table_id)
VALUES(@ReservationID,@client_id,@table_id);
INSERT INTO reservations(reservation_id, order_id,start_date,end_date,approved,amount_people)
VALUES(@ReservationID,@orderID,@start_date,@end_date,@approved,@amount_people);



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

Procedura addReservation_IndividualClient

Procedura dodaje rezerwacje otrzymując dane: id klienta, ilość osób, datę rozpoczęcia i zakończenia oraz informację o potwierdzeniu rezerwacji,nazwie dania i jego ilości .

```
ALTER PROCEDURE [dbo].[addReservation_IndividualClient] @client_id int, @amount_people INT,
@start_date datetime,@end_date datetime,@approved bit,@quantity int,@dish_name nchar(20)

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

   BEGIN TRY
                IF NOT EXISTS(
                SELECT *
                FROM Clients
                WHERE client_id = @client_id
                )
                BEGIN
                ;
                THROW 52000, 'Nie ma takiego klienta', 1
                end

                DECLARE @ReservationID INT
                SELECT @ReservationID = ISNULL(MAX(reservation_id), 0) + 1
                FROM reservations

                DECLARE @orderID INT
                SELECT @orderID = ISNULL(MAX(order_id), 0) + 1
                FROM orders

                Declare @table_id INT
                Select top 1 @table_id=table_id
                From tables
                Where size>=@amount_people

                if @start_date is null or @end_date is null
                begin;
                throw 52000, 'trzeba podac obie daty',2
                end
```

```
IF @start_date<GETDATE()
Begin;
throw 52000, 'rezerwacja na przeszłość', 3
end

IF (select * from dbo.getWKClient (@client_id) )< (select top 1 WK from reservationsRequirments order by id desc)
BEGIN
;
THROW 52000, 'Zbyt mało wcześniejszych zamówień', 4
end

if @quantity * (select price from AllDishes where dish_name=@dish_name) < (select top 1 WZ from reservationsRequirments order by id desc)
begin;
throw 52000, 'Zbyt niska cena zamówienia', 5
end

Declare @is_seafood bit
Select @is_seafood=is_seafood from menu as m
inner join dishes as d on d.id_dish_name=m.dish_name
Where @dish_name=d.dish_name

IF @is_seafood=0
Begin
execute addOrder @client_id,0,null,null,@quantity,@dish_name
End

IF @is_seafood=1
Begin
execute addSeafoodOrder @client_id,0,null,null,@quantity,@dish_name,@start_date
End


INSERT INTO reservationDetails(reservation_id, client_id,table_id)
VALUES(@ReservationID,@client_id,@table_id);
INSERT INTO reservations(reservation_id, order_id,start_date,end_date,approved,amount_people)
VALUES(@ReservationID,@orderID,@start_date,@end_date,@approved,@amount_people);


END TRY
```

Procedura SetAsActiveInMenu

Procedura ustawia wybranym pozycją w menu przekazanym po przecinku lub innym znaku rozdzielającym pole in_menu na True

```
ALTER PROCEDURE [dbo].[SetAsActiveInMenu] @List VARCHAR(MAX), @Delimiter  CHAR(1)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    -- Insert statements for procedure here
    Begin Try


        Update menu
        Set in_menu=1
        from menu inner JOin SplitInts(@List,@Delimiter) as t1 on t1.Item=menu.id_dish
        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

END
```

Procedura ChangeMenu

Procedura wymienia dania z menu które były dłużej niż dwa tygodnie i wstawia na ich miejsce nowe pozycje które były nieaktywne wcześniej.

```
ALTER PROCEDURE [dbo].[ChangeMenu]  @end_date datetime
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    --Zmienia Rzeczy przedawnione z menu
    SET NOCOUNT ON;

    Declare @how_much_to_add int

    Declare @table as Table (id_dish int)
    Insert into @table
    select t1.id_dish from getOutDatedMenu() as t1

    Select @how_much_to_add=Count(*) from @table

    Begin Try
    update menu
    set in_menu=0
    from menu inner Join (select * from menu where id_dish in (Select id_dish from @table) ) as t on t.id_dish=menu.id_dish

    END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

    Begin Try
    update menu
    set in_menu=1,in_menu_start=GETDATE(),in_menu_end=@end_date
    from menu inner Join (select top (@how_much_to_add) * from menu where in_menu=0 and id_dish not in(Select id_dish from @table)) as t on t.id_dish=menu.id_dish

    END TRY
        BEGIN CATCH
        Declare @msg1 nvarchar(2048) =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH
```

Procedura ChangeHalfMenu

Procedura wymienia połowę rzeczy z aktualnego menu na nowe

```
ALTER PROCEDURE [dbo].[ChangeHalfMenu]
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    Declare  @how_much_to_swap int=dbo.getNumberOfHalfActiveMenu()

    Declare @table as Table (id_dish int,price money,dish_name int,in_menu bit,in_menu_start datetime,in_menu_end datetime,description varchar(100),is_seafood bit)

    Insert into @table
    select top (@how_much_to_swap) * from menu where in_menu=0
    Begin Try
    update menu
    set in_menu=1
    from menu inner Join (select top (@how_much_to_swap) * from menu where in_menu=0 ) as t on t.id_dish=menu.id_dish

    END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

    Begin Try
    update menu
    set in_menu=0
    from menu inner Join (select top (@how_much_to_swap) * from menu where in_menu=1 and id_dish not in(Select id_dish from @table)) as t on t.id_dish=menu.id_dish

    END TRY
        BEGIN CATCH
        Declare @msg1 nvarchar(2048) =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

    -- Insert statements for procedure here

END
```

Procedura SetAsNonActiveInMenu

Procedura ustawia wybranym pozycją w menu przekazanym po przecinku lub innym znaku rozdzielającym pole in_menu na False

```
ALTER PROCEDURE [dbo].[SetAsNonActiveInMenu] @List VARCHAR(MAX), @Delimiter  CHAR(1)
 AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    -- Insert statements for procedure here
    Begin Try


        Update menu
        Set in_menu=0
        from menu inner JOin SplitInts(@List,@Delimiter) as t1 on t1.Item=menu.id_dish
        END TRY
        BEGIN CATCH
        DECLARE @msg nvarchar(2048)
        =N'Error:' + ERROR_MESSAGE();
        THROW 52000, @msg, 1
        END CATCH

END
```

FUNKCJE

Funkcja priceOverGiven

Funkcja zwracająca tabelę z zamówieniami, za które zapłacono więcej niż podana w funkcji cena.

```
CREATE FUNCTION [dbo].[priceOverGiven](@input int)
RETURNS table
AS
    return
    select order_id, TotalPrice from OrdersInfo where TotalPrice > @input
GO
```

Funkcja getOutDatedMenu

Funkcja zwraca przedawnione menu

```
CREATE FUNCTION [dbo].[getOutDatedMenu]()
RETURNS table
AS

RETURN (SELECT id_dish,dish_name, in_menu_start, in_menu_end from CurrentMenu where DATEDIFF(DAY, GETDATE(),
in_menu_start) > 14 or DATEDIFF(DAY, in_menu_start, in_menu_end) > 14)

GO
```

Funkcja getWKClient

Zwraca ilość złożonych zamówień przez danego klienta

```
CREATE FUNCTION [dbo].[getWKClient](@input int)
RETURNS table
AS

RETURN (SELECT c.Expr1 from ClientOrderedXTimes as c where c.client_id = @input)

GO
```

Funkcja clientsOrdersPriceOverGiven

Funkcja zwraca listę klientów, którzy zapłacili w sumie za zamówienia więcej niż podana w funkcji wartość.

```
CREATE FUNCTION [dbo].[clientsOrdersPriceOverGiven](@input int)
RETURNS table
AS
    return
    select ClientStats.client_id from ClientStats where @input < ClientStats.Total
GO
```

Funkcja soldMoreThanGiven

Funkcja zwracająca zamówienia, w których liczba zamówionych dań jest większa niż liczba podana w funkcji.

```
CREATE FUNCTION [dbo].[soldMoreThamGiven](@input int)
RETURNS table
AS
    return
    select menu.dish_name, menu.price, OrderDetails.quantity from menu inner join OrderDetails
    on menu.id_dish = OrderDetails.id_dish where @input < OrderDetails.quantity
GO
```

Funkcja getMenuByDate

Funkcja zwracająca dania które były w menu w podanym dniu.

```
CREATE FUNCTION [dbo].[getMemuByDate](@date date)
RETURNS table
AS
    return
    select dish_name, price, in_menu_start, in_menu_end from menu where in_menu_start < @date and @date < in_menu_end
GO
```

Funkcja getCorrectMenu

Funkcja wyświetla wszystkie dania, które zostały prawidłowo dodane

```
CREATE FUNCTION [dbo].[getCorrectMenu]()
RETURNS TABLE
AS
    RETURN(SELECT dishes.dish_name FROM menu inner join dishes on menu.id_dish = dishes.id_dis
    where DATEDIFF(day, in_menu_end, in_menu_start) < 15)
GO
```

Funkcja getOrderByDate

Funkcja zwraca zamówienia z danego dnia.

```
CREATE FUNCTION [dbo].[getOrderByDate](@date date)
RETURNS TABLE
AS
    return
    (select dishes.dish_name from orders inner join OrderDetails on orders.order_id = OrderDetails.order_id
    inner join menu on menu.id_dish = OrderDetails.id_dish inner join dishes on dishes.id_dish_name = menu.dish_name
    where day(orders.order_date) = day(@date) and month(orders.order_date) = month(@date) and year(orders.order_date) = year(@date)
GO
```

Funkcja getOrderValue

Funkcja zwracająca całkowitą wartość dla wybranego zamówienia.

```
CREATE FUNCTION [dbo].[getOrderValue](@id int)
RETURNS money
AS
BEGIN
RETURN (SELECT TotalPrice FROM OrdersInfo
WHERE order_id = @id)
END
GO
```

Funkcja clientsBoughtXTimes

Zwraca danie które zostały kupione przez klientów podaną w funkcji liczbę razy lub więcej.

```
CREATE FUNCTION [dbo].[clientsBoughtXTimes](@input int)
RETURNS table
AS
    return
    select clients.client_id, COUNT(*) as ilosc from clients inner join orders on orders.client_id = clients.client_id inner join individualClient
    individualClient.person_id=clients.client_id inner join company on clients.client_id=company.company_id
    group by clients.client_id having @input <= COUNT(*)
GO
```

Funkcja GetAvgPriceOfCurrentMenu

Zwraca średnią cenę dania w aktualnym menu.

```
CREATE FUNCTION [dbo].[GetAvgPriceOfCurrentMenu]()
RETURNS money
AS
BEGIN
RETURN (SELECT AVG(price) FROM CurrentMenu)
END
GO
```

Funkcja GetMaxPriceFromCurrentMenu

Zwraca najwyższą cenę dania w aktualnym menu.

```
CREATE FUNCTION [dbo].[GetMaxPriceFromCurrentMenu]()
RETURNS money
AS
BEGIN
RETURN (SELECT TOP 1 MAX(Price) FROM CurrentMenu)
END
GO
```

Funkcja GetMinPriceFromCurrentMenu

Funkcja zwracająca najniższą cenę dania w aktualnym menu.

```
CREATE FUNCTION [dbo].[GetMinPriceFromCurrentMenu]()
RETURNS money
AS
BEGIN
RETURN (SELECT TOP 1 MIN(price) FROM CurrentMenu)
END
GO
```

Funkcja GetMostQuantitySoldDish

Funkcja zwraca danie, które zostało kupione największą liczbę razy.

```
CREATE FUNCTION [dbo].[GetMostQuantitySoldDish]()
RETURNS table AS
RETURN
Select Top(1) * From SoldDishes Order By TotalQuantity Desc
GO
```

Funkcja isClientGotDiscount

Funkcja zwraca informację czy podany klient (po numerze id) posiada zniżkę/zniżki.

```
SET QUOTED_IDENTIFIER ON
GO

CREATE FUNCTION [dbo].[isClientGotDiscount](@input int)
RETURNS table
AS
    return
    (select client_id from ClientDiscountInfo where R2startdate < GETDATE() and
    getdate() < R2enddate and @input = client_id
    union
    select client_id from ClientDiscountInfoR1 where @input = client_id)
GO
```

Funkcja SplitInts

Funkcja zwraca tabele z rekordami, które zawierają wartości podane po przecinku lub innym znaku rozdzielającym

```
ALTER FUNCTION [dbo].[SplitInts]
(
    @List      VARCHAR(MAX),
    @Delimiter  CHAR(1)
)

RETURNS TABLE
AS
RETURN
(
    SELECT Item = CONVERT(INT, Item)
       FROM
       (
           SELECT Item = x.i.value('(./text())[1]', 'INT')
           FROM
           (
               SELECT [XML] = CONVERT(XML, '<i>'
                    + REPLACE(@List, @Delimiter, '</i><i>')
                    + '</i>').query('.')
           ) AS a
           CROSS APPLY
           [XML].nodes('i') AS x(i)
       ) AS y
       WHERE Item IS NOT NULL
)
```

Funkcja areDatesCorrect

Funkcja sprawdza czy pozycje w menu nie są dłużej niż dwa tygodnie

```
ALTER FUNCTION [dbo].[areDatesCorrect](@input1 date, @input2 date)
RETURNS bit
AS
    begin
        DECLARE @val bit;
        IF DATEDIFF(day, @input1, @input2) < 15
            begin
                SET @val = 1
            end
        IF DATEDIFF(day, @input1, @input2) >= 15
            begin
                SET @val = 0
            end
        return @val
    end
```

TRIGGERY

Trigger ProperDataForDiscount

Trigger blokuje dodawanie zniżek, których wartości nie są z przedziału 0-1 (czyli 0%-100%)

```
CREATE TRIGGER [dbo].[ProperDataForDiscount]
   ON  [dbo].[discountConst]
   AFTER insert
AS
BEGIN
    if (select count(*) from inserted) > 1
    begin
        RAISERROR ('za dużo danych, wprowadź wartość jednej zniżki',16,1)
        ROLLBACK TRANSACTION
    end
    else if (select  R1 from inserted) not between 0 and 1
    begin
        RAISERROR('Niepoprawny rabat R1, podaj liczbę z przedzialu 0 do 1',16,1)
        ROLLBACK TRANSACTION
    END
    else if (select  R2 from inserted) not between 0 and 1
    begin
        RAISERROR('Niepoprawny rabat R2, podaj liczbę z przedzialu 0 do 1',16,1)
        ROLLBACK TRANSACTION
    END
END
GO

ALTER TABLE [dbo].[discountConst] ENABLE TRIGGER [ProperDataForDiscount]
GO
```

Trigger ProperPrice

Trigger sprawdza czy cena dodawanego do menu dania jest prawidłowa

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER trigger [dbo].[TR_ProperPrice] on [dbo].[menu]
for insert
as
BEGIN
if (select price from inserted) <0
begin
RAISERROR('Cena nie może być ujemna ', 16, 1)
ROLLBACK TRANSACTION
End
END
```

Trigger SeaFoodCheck

Sprawdza czy zamówienie zostało prawidłowo złożone

```
ALTER TRIGGER [dbo].[TR_SeaFoodCheck]
ON [dbo].[OrderDetails]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF not  EXISTS(
SElect inserted.order_id from inserted inner join  OrderDetails AS OD on od.order_id=inserted.order_id
    INNER JOIN Orders AS O ON OD.order_id = O.order_id
    INNER JOIN menu AS M ON M.id_dish=OD.id_dish and m.is_seafood=1
    WHERE ((DATENAME(WEEKDAY, O.pref_date_of_seafood) like 'Thursday' and DATEDIFF(DAY,o.order_date, O.pref_date_of_seafood) > 3)
    OR (DATENAME(WEEKDAY, O.pref_date_of_seafood) like 'Friday' and DATEDIFF(DAY,o.order_date, O.pref_date_of_seafood) > 4)
    OR (DATENAME(WEEKDAY, O.pref_date_of_seafood) like 'Saturday' and DATEDIFF(DAY,o.order_date, O.pref_date_of_seafood) > 5)
    )) AND Exists(SElect inserted.order_id from inserted inner join  OrderDetails AS OD on od.order_id=inserted.order_id
    INNER JOIN Orders AS O ON OD.order_id = O.order_id
    INNER JOIN menu AS M ON M.id_dish=OD.id_dish and m.is_seafood=1)
BEGIN;
    THROW 50001, 'Nie możemy przyjąć takiego zamówienia. ', 1
END
END
```

Trigger DeleteCanceledReservations

Usuwa rezerwacje stolików, które zostały odrzucone.

```
CREATE TRIGGER [dbo].[DeleteCanceledReservations]
   ON [dbo].[reservations]
   FOR DELETE
AS
BEGIN

    SET NOCOUNT ON;
    DELETE FROM reservations
    where reservation_id in (
    select reservation_id from reservations where approved = 0
    )

END
GO
```

INDEKSY

Indeks PK_cities

Klucz główny tabeli cities

```
ALTER TABLE [dbo].[cities] ADD  CONSTRAINT [PK_cities] PRIMARY KEY CLUSTERED
(
    [city_id] ASC
)WITH
ON [PRIMARY]
GO
```

Indeks PK_clients

Klucz główny tabeli clients

```
ALTER TABLE [dbo].[clients] ADD  CONSTRAINT [PK_clients] PRIMARY KEY CLUSTERED
(
    [client_id] ASC
) ON [PRIMARY]
GO
```

Indeks unique_email

Indeks dla emila w tabeli clients

```
ALTER TABLE [dbo].[clients] ADD  CONSTRAINT [unique_email] UNIQUE NONCLUSTERED
(
    [email] ASC
) ON [PRIMARY]
GO
```

Indeks PK_discount

Klucz główny w tabeli discounts

```
ALTER TABLE [dbo].[discounts] ADD  CONSTRAINT [PK_discounts] PRIMARY KEY CLUSTERED
(
    [client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_dishes

Klucz główny w tabeli dishes

```
ALTER TABLE [dbo].[dishes] ADD  CONSTRAINT [PK_dishes] PRIMARY KEY CLUSTERED
(
    [id_dish_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_individualClient

Klucz główny w tabeli individualClient

```
ALTER TABLE [dbo].[individualClient] ADD  CONSTRAINT [PK_individualClient] PRIMARY KEY CLUSTERED
(
    [client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_menu

Klucz główny w tabeli menu

```
ALTER TABLE [dbo].[menu] ADD  CONSTRAINT [PK_menu] PRIMARY KEY CLUSTERED
(
    [id_dish] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks IX_OrderDetails

Klucz w tabeli OrderDetails łączący ją z tabelą orders

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

Indeks PK_OrderDetails

Klucz główny w tabeli OrderDetails

```
ALTER TABLE [dbo].[OrderDetails] ADD  CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_orders_1

Klucz główny w tabeli orders

```
ALTER TABLE [dbo].[orders] ADD  CONSTRAINT [PK_orders_1] PRIMARY KEY CLUSTERED
(
    [order_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_postal_codes

Klucz główny w tabeli postal_codes

```
ALTER TABLE [dbo].[postal_codes] ADD  CONSTRAINT [PK_postal_codes] PRIMARY KEY CLUSTERED
(
    [postal_code_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_reservationsRequirments

Klucz główny w tabeli reservationsRequirments

```
ALTER TABLE [dbo].[reservationsRequirments] ADD  CONSTRAINT [PK_reservationsRequirments] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_reservationDetails

Klucz główny w tabeli reservationDetails

```
ALTER TABLE [dbo].[reservationDetails] ADD  CONSTRAINT [PK_reservationDetails] PRIMARY KEY CLUSTERED
(
    [reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_reservations

Klucz główny w tabeli reservations

```
ALTER TABLE [dbo].[reservations] ADD  CONSTRAINT [PK_reservations] PRIMARY KEY CLUSTERED
(
    [reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks orderid_reservations

Klucz w tabeli reservations łączący ją z tabelą orders

```
CREATE UNIQUE NONCLUSTERED INDEX [orderid_reservations] ON [dbo].[reservations]
(
    [order_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_tables

Klucz główny w tabeli tables

```
ALTER TABLE [dbo].[tables] ADD  CONSTRAINT [PK_tables] PRIMARY KEY CLUSTERED
(
    [table_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_takeAway

Klucz główny w tabeli takeAway

```
ALTER TABLE [dbo].[takeAway] ADD  CONSTRAINT [PK_takeAway] PRIMARY KEY CLUSTERED
(
    [take_away_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks PK_warehouse

Klucz główny w tabeli warehouse

```
ALTER TABLE [dbo].[warehouse] ADD  CONSTRAINT [PK_warehouse] PRIMARY KEY CLUSTERED
(
    [id_dish] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON,ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
GO
```

Indeks unique_phone

Indeks dla numeru telefonu w tabeli clients

```
ALTER TABLE [dbo].[clients] ADD  CONSTRAINT [unique_phone] UNIQUE NONCLUSTERED
(
    [phone] ASC
)ON [PRIMARY]
GO
```

Indeks PK_company

Klucz główny w tabeli company

```
ALTER TABLE [dbo].[company] ADD  CONSTRAINT [PK_company] PRIMARY KEY CLUSTERED
(
    [company_id] ASC
) ON [PRIMARY]
GO
```

Indeks unique_nip

Indeks dla numeru NIP w tabeli company

```
ALTER TABLE [dbo].[company] ADD  CONSTRAINT [unique_nip] UNIQUE NONCLUSTERED
(
    [nip] ASC
) ON [PRIMARY]
GO
```

Indeks PK_discountConst

Klucz główny w tabeli discountConst

```
ALTER TABLE [dbo].[discountConst] ADD  CONSTRAINT [PK_discountConst] PRIMARY KEY CLUSTERED
(
    [id] ASC
) ON [PRIMARY]
GO
```

UPRAWNIENIA

Uprawnienia pracownika

Wystawia faktury (dla klienta indywidualnego oraz zbiorcze raz na miesiąc), odbiera zamówienia dotyczące owoców morza, oraz je realizuje w odpowiednim czasie (importuje), odpowiada za potwierdzenie zamówienia rezerwacji stolika, ma dostęp do informacji rabatowej klienta oraz danych zamówienia, jest odpowiedzialny za generowanie raportów (miesięcznych i tygodniowych). Ma dostęp do menu i może tworzyć nowe zamówienia.

```
CREATE ROLE worker

GRANT SELECT ON AllDishes to worker
GRANT SELECT ON ClientDiscountInfo to worker
GRANT SELECT ON ClientDiscountInfoR1 to worker
GRANT SELECT ON clientsInfo to worker
GRANT SELECT ON ClientStats to worker
GRANT SELECT ON companyClientsInfo to worker
GRANT SELECT ON CurrentMenu to worker
GRANT SELECT ON discountUsed to worker
GRANT SELECT ON individualClientInfo to worker
GRANT SELECT ON individualClientStats to worker
GRANT SELECT ON menuDates to worker
GRANT SELECT ON monthlyInvoice to worker
GRANT SELECT ON MonthOrderStatistic to worker
GRANT SELECT ON NotApprovedReservations to worker
GRANT SELECT ON NotYetDeliveredTakeaways to worker
GRANT SELECT ON OrdersInfo to worker
GRANT SELECT ON Seafood to worker
GRANT SELECT ON singleInvoice to worker
GRANT SELECT ON SoldDishes to worker
GRANT SELECT ON TablesReservationStats to worker

GRANT SELECT ON isClientGotDiscount to worker
GRANT EXECUTE ON getOrderValue to worker

GRANT EXECUTE ON addCompanyClient to worker
GRANT EXECUTE ON addIndividualClient to worker
GRANT EXECUTE ON addOrder to worker
GRANT EXECUTE ON addProductToOrder to worker
GRANT EXECUTE ON addReservation to worker
GRANT EXECUTE ON ApproveReservation to worker
```

Uprawnienia menadżera

Wybiera aktualne (w dany terminie) MENU może dodawać nowe produkty do menu, ustala progi rabatowe, może generować statystyki zamówień, posiada wszystkie uprawnienia zwykłego pracownika. Ma możliwość usunięcia dodania lub zmiany dla danych klienta, firmy i indywidualnego klienta.

```
CREATE ROLE manager

GRANT SELECT ON AllDishes to manager
GRANT SELECT ON ClientDiscountInfo to manager
GRANT SELECT ON ClientDiscountInfoR1 to manager
GRANT SELECT ON clientsInfo to manager
GRANT SELECT ON ClientStats to manager
GRANT SELECT ON companyClientsInfo to manager
GRANT SELECT ON CurrentMenu to manager
GRANT SELECT ON discountUsed to manager
GRANT SELECT ON individualClientInfo to manager
GRANT SELECT ON individualClientStats to manager
GRANT SELECT ON menuDates to manager
GRANT SELECT ON monthlyInvoice to manager
GRANT SELECT ON MonthOrderStatistic to manager
GRANT SELECT ON NotApprovedReservations to manager
GRANT SELECT ON NotYetDeliveredTakeaways to manager
GRANT SELECT ON OrdersInfo to manager
GRANT SELECT ON Seafood to manager
GRANT SELECT ON singleInvoice to manager
GRANT SELECT ON SoldDishes to manager
GRANT SELECT ON TablesReservationStats to manager

GRANT SELECT ON isClientGotDiscount to manager
GRANT EXECUTE ON getOrderValue to manager
```

```
GRANT EXECUTE ON addCompanyClient to manager
GRANT EXECUTE ON addIndividualClient to manager
GRANT EXECUTE ON addOrder to manager
GRANT EXECUTE ON addProductToOrder to manager
GRANT EXECUTE ON addReservation to manager
GRANT EXECUTE ON ApproveReservation to manager
GRANT EXECUTE ON addDish to manager
GRANT EXECUTE ON addProductToMenu to manager
GRANT EXECUTE ON SetAsActiveInMenu to manager
GRANT EXECUTE ON SetAsNonActiveInMenu to manager

GRANT SELECT, INSERT, DELETE ON clients to manager
GRANT SELECT, INSERT, DELETE ON company to manager
GRANT SELECT, INSERT, DELETE ON individualClient to manager
```

Uprawnienia admina

Posiada wszystkie uprawnienia w systemie, nadzoruje jego działanie.

```
CREATE ROLE admin

grant all privileges ON u_smigala.dbo TO admin
```