AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Podstawy baz danych 2021/22

Projekt systemu bazodanowego

Autorzy:
Zuzanna Olszówka, Dorota Meszka, Marcin Szwed

Spis treści:

Użytkownicy:

1. Biznesowy administrator
2. Pracownik restauracji odpowiedzialny za menu
3. Pracownik restauracji odpowiedzialny za rezerwacje
4. Pracownik restauracji odpowiedzialny za tworzenie raportów i faktur
5. Pracownik restauracji odpowiedzialny za zamówienia
6. Pracownik restauracji odpowiedzialny za backup
7. Klient indywidualny
8. Klient firma

Funkcje użytkowników:

1. Biznesowy administrator
   - tworzenie, edycja usuwanie, wyszukiwanie kont użytkowników
   - przypisywanie ról do kont użytkowników
2. Pracownik restauracji odpowiedzialny za menu
   - definiowanie dań (tworzenie, edycja, usuwanie / zaznaczanie jako nieaktywne)
   - definiowanie pozycji menu (danie, cena)
   - definiowanie menu (daty obowiązywania)
3. Pracownik restauracji odpowiedzialny za rezerwacje
   - sprawdzanie czy możliwe jest wykonanie rezerwacji (sprawdzanie dostępności stolików)
     - akceptacja i dodanie rezerwacji do systemu
     - generowanie informacji z potwierdzeniem wraz ze wskazaniem stolika lub informacji o braku możliwości wykonania rezerwacji
     - anulowanie rezerwacji - sprawdzenie, czy anulowanie jest możliwe, oznaczenie rezerwacji jako niezrealizowanej
   - potwierdzenie anulowania zamówienia klientowi
4. Pracownik restauracji odpowiedzialny za tworzenie raportów i faktur
   - wystawianie faktury dla danego zamówienia
   - wystawianie faktury zbiorczej raz na miesiąc
   - generowanie raportów tygodniowych dotyczących:
   - Rezerwacji stolików
   - Rabatów
   - Menu
   - generowanie raportów miesięcznych dotyczących:
   - Rezerwacji stolików
   - Rabatów
   - Menu
   - generowanie statystyk zamówień oraz rabatów dla klienta indywidualnego
   - generowanie statystyk zamówień dla firmy
5. Pracownik restauracji odpowiedzialny za zamówienia
   - dodawanie/usuwanie zamówień
   - obsługa płatności
   - wystawianie faktur do zamówienia
   - obsługa zrealizowanych zamówień
6. Pracownik restauracji odpowiedzialny za backup

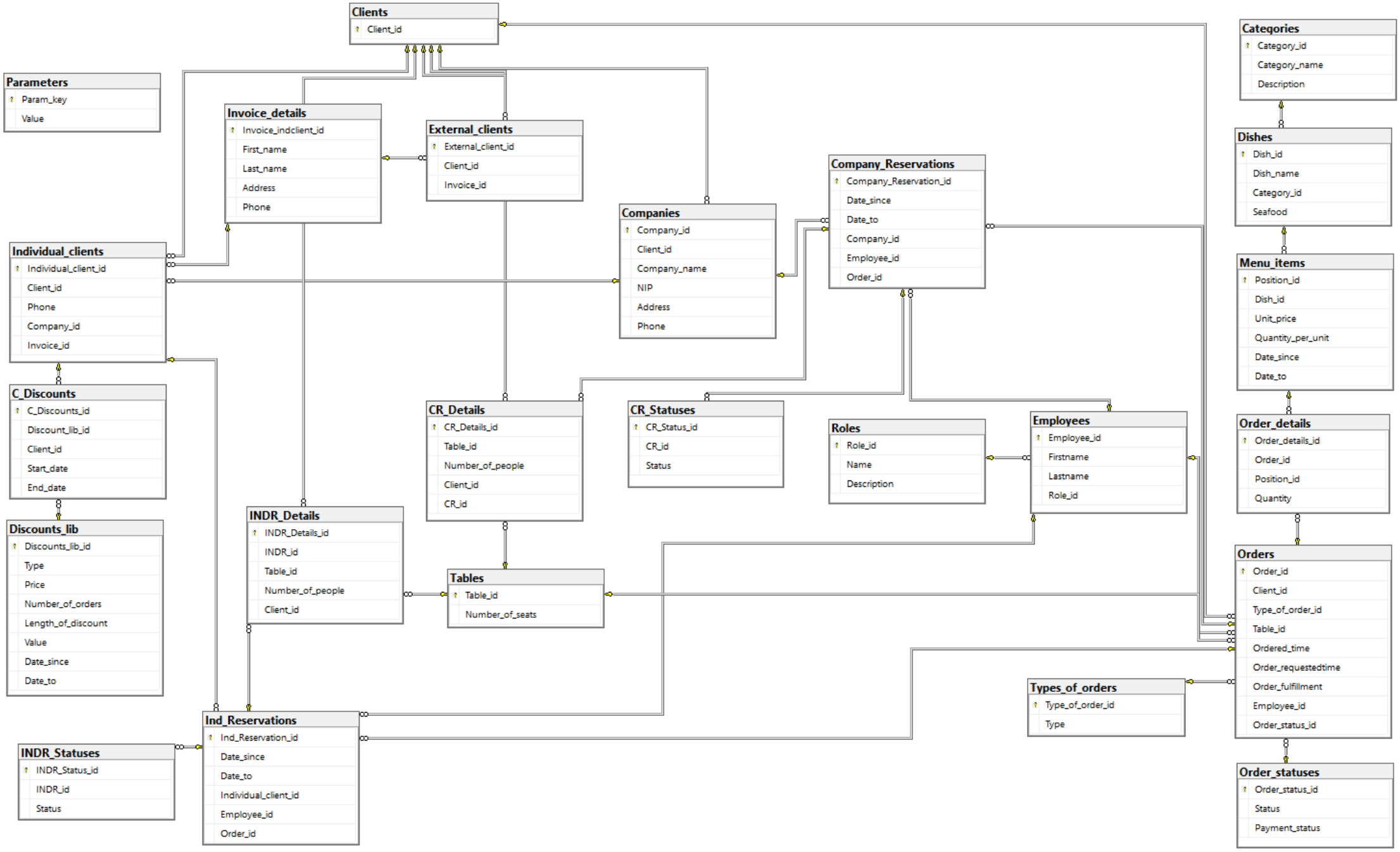
- tworzenie backupu systemu bazodanowego
- obsługa awarii systemu

7. Klient indywidualny
   - rezerwacja stolika na klienta indywidualnego
   - zamawianie jedzenia na miejscu
   - zamawianie jedzenia na wynos
   - otrzymywanie faktur
   - zgłoszenie chęci anulowania rezerwacji pracownikowi
8. Klient firma
   - rezerwacja stolika na firmę/pracownika firmy
   - zamawianie jedzenia na miejscu
   - zamawianie jedzenia na wynos
   - otrzymywanie faktur
   - zgłoszenie chęci anulowania rezerwacji pracownikowi

Funkcje systemowe:

- naliczanie rabatów
- sprawdzanie możliwości rezerwacji (czy są dostępne stoliki)
- kontrolowanie menu - czy w ostatnim czasie dokonano aktualizacji
- sprawdzenie czy anulowanie zamówienia jest możliwe (limit czasowy)

Schemat:


Clients
Client_id
Parameters
Param_key
Value
Invoice_details
Invoice_indclient_id
First_name
Last_name
Address
Phone
External_clients
External_client_id
Client_id
Invoice_id
Companies
Company_id
Client_id
Company_name
NIP
Address
Phone
Company_Reservations
Company_Reservation_id
Date_since
Date_to
Company_id
Employee_id
Order_id
Categories
Category_id
Category_name
Description
Dishes
Dish_id
Dish_name
Category_id
Seafood
Menu_items
Position_id
Dish_id
Unit_price
Quantity_per_unit
Date_since
Date_to
Individual_clients
Individual_client_id
Client_id
Phone
Company_id
Invoice_id
C_Discounts
C_Discounts_id
Discount_lib_id
Client_id
Start_date
End_date
CR_Details
CR_Details_id
Table_id
Number_of_people
Client_id
CR_id
CR_Statuses
CR_Status_id
CR_id
Status
Roles
Role_id
Name
Description
Employees
Employee_id
Firstname
Lastname
Role_id
Order_details
Order_details_id
Order_id
Position_id
Quantity
Discounts_lib
Discounts_lib_id
Type
Price
Number_of_orders
Length_of_discount
Value
Date_since
Date_to
INDR_Details
INDR_Details_id
INDR_id
Table_id
Number_of_people
Client_id
Tables
Table_id
Number_of_seats
Orders
Order_id
Client_id
Type_of_order_id
Table_id
Ordered_time
Order_requestedtime
Order_fulfillment
Employee_id
Order_status_id
Types_of_orders
Type_of_order_id
Type
Ind_Reservations
Ind_Reservation_id
Date_since
Date_to
Individual_client_id
Employee_id
Order_id
INDR_Statuses
INDR_Status_id
INDR_id
Status
Order_statuses
Order_status_id
Status
Payment_status

Opisy tabel:

Tabela C_Discounts
Reprezentacja informacji o zniżkach klienta w bazie danych.

- C_Discount_id - klucz główny
- Discount_lib_id - klucz obcy
- Client_id - klucz obcy
- Start_date - data uzyskania rabatu
- End_date - końcowa data działania rabatu

Warunki integralności:

- Data uzyskania rabatu jest mniejsza bądź równa obecnej dacie:

```
ALTER TABLE [dbo].[C_Discounts]  WITH CHECK ADD CONSTRAINT [c_cdi_start_date]
CHECK  (([Start_date]<=GETDATE()))
GO
ALTER TABLE [dbo].[C_Discounts] CHECK CONSTRAINT [c_cdi_start_date]
GO
```

```
CREATE TABLE [dbo].[C_Discounts](
	[C_Discounts_id] [int] NOT NULL,
	[Discount_lib_id] [int] NOT NULL,
	[Client_id] [int] NOT NULL,
	[Start_date] [datetime] NOT NULL,
	[End_date] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[C_Discounts_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 Categories
Reprezentacja kategorii potraw w bazie danych.

- Category_id - klucz główny
- Category_name - nazwa kategorii
- Description - opis kategorii

```
CREATE TABLE [dbo].[Categories](
	[Category_id] [int] NOT NULL,
	[Category_name] [varchar](50) NOT NULL,
	[Description] [varchar](100) NULL,
 CONSTRAINT [Categories_pk] PRIMARY KEY CLUSTERED
(
	[Category_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 Clients

Reprezentacja klientów w bazie danych.

- Client_id - klucz główny

```
CREATE TABLE [dbo].[Clients](
      [Client_id] [int] NOT NULL,
 CONSTRAINT [Clients_pk] 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_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Companies

Reprezentacja firm w bazie danych.

- Company_id - klucz główny
- Client_id - klucz obcy
- Company_name - nazwa firmy
- NIP - numer NIP
- Address - adres
- Phone - numer telefonu

Warunki integralności:

- numer NIP jest unikalny:

```
CONSTRAINT [U_NIP] UNIQUE NONCLUSTERED
(
      [NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

- numer NIP składa się z cyfr od 0 do 9:

```
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD CONSTRAINT [c_comp_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].[Companies] CHECK CONSTRAINT [c_comp_nip]
GO
```

- numer telefonu składa się z cyfr od 0 do 9:

```
ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [c_comp_phone]
CHECK  (([Phone] like
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' OR [Phone] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [c_comp_phone]
GO
```

```
CREATE TABLE [dbo].[Companies](
	[Company_id] [int] NOT NULL,
	[Client_id] [int] NOT NULL,
	[Company_name] [varchar](40) NOT NULL,
	[NIP] [varchar](50) NOT NULL,
	[Address] [varchar](60) NOT NULL,
	[Phone] [varchar](24) NOT NULL,
 CONSTRAINT [Companies_pk] 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 [U_NIP] UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Company_Reservations

Reprezentacja rezerwacji dokonanych przez firmę w bazie danych.

- Copmany_Reservation_id - klucz główny
- Date_since - czas rozpoczęcia rezerwacji
- Date_to - czas zakończenia rezerwacji
- Company_id - klucz obcy
- Employee_id - klucz obcy
- Order_id - klucz obcy

Warunki integralności:

- czas rozpoczęcia rezerwacji jest wcześniejszy niż czas zakończenia rezerwacji:

```
ALTER TABLE [dbo].[Company_Reservations]  WITH CHECK ADD  CONSTRAINT
[c_comr_date_since_earlier_than_date_to] CHECK  (([Date_since]<[Date_to]))
GO
ALTER TABLE [dbo].[Company_Reservations] CHECK CONSTRAINT
[c_comr_date_since_earlier_than_date_to]
```

```
GO
```

```
CREATE TABLE [dbo].[Company_Reservations](
	[Company_Reservation_id] [int] NOT NULL,
	[Date_since] [datetime] NOT NULL,
	[Date_to] [datetime] NOT NULL,
	[Company_id] [int] NOT NULL,
	[Employee_id] [int] NOT NULL,
	[Order_id] [int] NULL,
 CONSTRAINT [Company_Reservations_pk] PRIMARY KEY CLUSTERED
(
	[Company_Reservation_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 CR_Details

Reprezentacja szczegółów rezerwacji dokonanej przez firmę w bazie danych.

- CR_Detail_id - klucz główny
- Table_id - klucz obcy
- Client_id - klucz obcy
- CR_id - klucz obcy

```
CREATE TABLE [dbo].[CR_Details](
	[CR_Details_id] [int] NOT NULL,
	[Table_id] [int] NULL,
	[Number_of_people] [int] NOT NULL,
	[Client_id] [int] NULL,
	[CR_id] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[CR_Details_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 CR_Statuses

Reprezentacja statusu rezerwacji dokonanej przez firmę w bazie.

- CR_Status_id - klucz główny
- CR_id - klucz obcy
- Status - status rezerwacji

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

```
	[CR_Status_id] [int] NOT NULL,
	[CR_id] [int] NOT NULL,
	[Status] [varchar](50) NULL,
PRIMARY KEY CLUSTERED
(
	[CR_Status_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 Discounts_lib

Biblioteka rabatów.

- Discount_lib_id - klucz główny
- Price - kwota (K1, K2)
- Number_of_orders - liczba zamówień (Z1)
- Length_of_discount - długość trwania rabatu (D1)
- Value - nakładany rabat (R1,R2)
- Date_since - od kiedy rabat może być nadawany
- Date_to - do kiedy rabat może być nadawany

Warunki integralności:

- długość trwania rabatu > 0:

```
ALTER TABLE [dbo].[Discounts_lib]  WITH CHECK ADD  CONSTRAINT
[length_more_than_zero] CHECK  (([length_of_discount]>(0)))
GO
ALTER TABLE [dbo].[Discounts_lib] CHECK CONSTRAINT [length_more_than_zero]
GO
```

- liczba zamówień > 0:

```
ALTER TABLE [dbo].[Discounts_lib]  WITH CHECK ADD  CONSTRAINT
[number_of_orders_more_than_zero] CHECK  (([number_of_orders]>(0)))
GO
ALTER TABLE [dbo].[Discounts_lib] CHECK CONSTRAINT
[number_of_orders_more_than_zero]
GO
```

- kwota > 0:

```
ALTER TABLE [dbo].[Discounts_lib]  WITH CHECK ADD  CONSTRAINT
[price_more_than_zero] CHECK  (([price]>(0)))
GO
ALTER TABLE [dbo].[Discounts_lib] CHECK CONSTRAINT [price_more_than_zero]
GO
```

- nakładany rabat > 0 i < 1:

```
ALTER TABLE [dbo].[Discounts_lib]  WITH CHECK ADD  CONSTRAINT
[value_more_than_zero_less_than_one] CHECK  (([value]>(0) AND [value]<(1)))
GO
ALTER TABLE [dbo].[Discounts_lib] CHECK CONSTRAINT
[value_more_than_zero_less_than_one]
GO
```

- Date_since wcześniejsza niż date_to:

```
ALTER TABLE [dbo].[Discounts_lib]  WITH CHECK ADD  CONSTRAINT
[c_disl_date_since_earlier_than_date_to] CHECK  (([Date_since]<[Date_to]))
GO
ALTER TABLE [dbo].[Discounts_lib] CHECK CONSTRAINT
[c_disl_date_since_earlier_than_date_to]
GO
```

```
CREATE TABLE [dbo].[Discounts_lib](
      CREATE TABLE [dbo].[Discounts_lib](
      [Discounts_lib_id] [int] NOT NULL,
      [Type] [varchar](10) NOT NULL,
      [Price] [money] NOT NULL,
      [Number_of_orders] [int] NULL,
      [Length_of_discount] [int] NULL,
      [Value] [decimal](10, 2) NOT NULL,
      [Date_since] [datetime] NOT NULL,
      [Date_to] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [Discounts_lib_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 Dishes

Reprezentacja potraw w bazie danych.

- Dish_id - klucz główny
- Dish_name - nazwa potrawy
- Category_id - klucz obcy
- Seafood - informacja o zawartości owoców morza w potrawie

```
CREATE TABLE [dbo].[Dishes](
      [Dish_id] [int] NOT NULL,
      [Dish_name] [varchar](50) NOT NULL,
      [Category_id] [int] NOT NULL,
      [Seafood] [varchar](10) NOT NULL,
 CONSTRAINT [Dishes_pk] PRIMARY KEY CLUSTERED
(
      [Dish_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 Employees

Reprezentacja pracowników w bazie danych.

- Employee_id - klucz główny
- Firstname - imię pracownika
- Lastname - nazwisko pracownika
- Role_id - klucz obcy

```
CREATE TABLE [dbo].[Employees](
      [Employee_id] [int] NOT NULL,
      [Firstname] [varchar](30) NOT NULL,
      [Lastname] [varchar](30) NOT NULL,
      [Role_id] [int] NOT NULL,
 CONSTRAINT [Employees_pk] PRIMARY KEY CLUSTERED
(
      [Employee_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 External_clients

Reprezentacja klientów z zewnątrz w bazie danych.

- External_client_id - klucz głowny
- Client_id - klucz obcy
- Invoice_id - klucz obcy

```
CREATE TABLE [dbo].[External_clients](
      [External_client_id] [int] NOT NULL,
      [Client_id] [int] NOT NULL,
      [Invoice_id] [int] NULL,
 CONSTRAINT [External_clients_pk] PRIMARY KEY CLUSTERED
(
      [External_client_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 Ind_Reservations

Reprezentacja rezerwacji dokonanych przez klientów indywidualnych w bazie danych.

- Ind_Reservation_id - klucz główny

- Date_since - data rozpoczęcia rezerwacji, tj. “na kiedy”
- Date_to - data zakończenia rezerwacji, tj. “do kiedy”
- Individual_client_id - klucz obcy; identyfikator klienta indywidualnego dokonującego rezerwacji
- Employee_id - klucz obcy; identyfikator pracownika obsługującego rezerwację
- Order_id - klucz obcy
- Table_id - klucz obcy

Warunki integralności:

- Date_since wcześniejsza niż Date_to:

```
ALTER TABLE [dbo].[Ind_Reservations]  WITH CHECK ADD  CONSTRAINT
[c_indr_date_since_earlier_than_date_to] CHECK  (([Date_since]<[Date_to]))
GO
ALTER TABLE [dbo].[Ind_Reservations] CHECK CONSTRAINT
[c_indr_date_since_earlier_than_date_to]
GO
```

```
CREATE TABLE [dbo].[Ind_Reservations](
      [Ind_Reservation_id] [int] NOT NULL,
      [Date_since] [datetime] NOT NULL,
      [Date_to] [datetime] NOT NULL,
      [Individual_client_id] [int] NOT NULL,
      [Employee_id] [int] NOT NULL,
      [Order_id] [int] NOT NULL,
 CONSTRAINT [Ind_Reservations_pk] PRIMARY KEY CLUSTERED
(
      [Ind_Reservation_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 Individual_clients

Reprezentacja klientów indywidualnych w bazie danych.

- Individual_client_id - klucz główny
- Client_id - klucz obcy
- Phone - nr telefonu
- Company_id - klucz obcy
- Invoice_id - klucz obcy

Warunki integralności:

- numer telefonu składa się z cyfr od 0 do 9:

```
ALTER TABLE [dbo].[Individual_clients]  WITH CHECK ADD  CONSTRAINT
[c_indc_phone] CHECK  (([Phone] like
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' OR [Phone] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO
```

```
ALTER TABLE [dbo].[Individual_clients] CHECK CONSTRAINT [c_indc_phone]
GO
```

```
CREATE TABLE [dbo].[Individual_clients](
	[Individual_client_id] [int] NOT NULL,
	[Client_id] [int] NOT NULL,
	[Phone] [varchar](24) NOT NULL,
	[Company_id] [int] NULL,
	[Invoice_id] [int] NULL,
 CONSTRAINT [Client_id] PRIMARY KEY CLUSTERED
(
	[Individual_client_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 INDR_Details

Reprezentacja szczegółów rezerwacji dokonanych przez klientów indywidualnych.

- INDR_Detail_id - klucz główny
- INDR_id - klucz obcy
- Table_id - klucz obcy
- Number_of_people - liczba osób
- Client_id - klucz obcy

Warunki integralności:

- liczba osób > 0:

```
ALTER TABLE [dbo].[INDR_Details]  WITH CHECK ADD  CONSTRAINT [c_ir_number]
CHECK  (([number_of_people]>(0)))
GO
ALTER TABLE [dbo].[INDR_Details] CHECK CONSTRAINT [c_ir_number]
GO
```

```
CREATE TABLE [dbo].[INDR_Details](
	[INDR_Details_id] [int] NOT NULL,
	[INDR_id] [int] NOT NULL,
	[Table_id] [int] NULL,
	[Number_of_people] [int] NOT NULL,
	[Client_id] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[INDR_Details_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 INDR_Statuses

Reprezentacja statusów rezerwacji dokonanych przez klientów indywidualnych.

- INDR_Status_id - klucz główny
- INDR_id - klucz obcy
- Status - status rezerwacji

```
CREATE TABLE [dbo].[INDR_Statuses](
	[INDR_Status_id] [int] NOT NULL,
	[INDR_id] [int] NOT NULL,
	[Status] [varchar](50) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[INDR_Status_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 Invoice_details

Dane do faktury dla klientów indywidualnych/klientów “z zewnątrz”.

- Invoice_indclient_id - klucz główny
- First_name - imię klienta
- Last_name - nazwisko klienta
- Address - adres zamieszkania klienta
- Phone - numer telefonu

Warunki integralności:

- numer telefonu składa się z cyfr od 0 do 9:

```
ALTER TABLE [dbo].[Invoice_details]  WITH CHECK ADD  CONSTRAINT
[c_invd_phone] CHECK  (([Phone] like
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' OR [Phone] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO
ALTER TABLE [dbo].[Invoice_details] CHECK CONSTRAINT [c_invd_phone]
GO
```

```
CREATE TABLE [dbo].[Invoice_details](
	[Invoice_indclient_id] [int] NOT NULL,
	[First_name] [varchar](20) NOT NULL,
	[Last_name] [varchar](20) NOT NULL,
	[Address] [varchar](50) NOT NULL,
```

```
	[Phone] [varchar](24) NOT NULL,
 CONSTRAINT [Invoice_details_pk] PRIMARY KEY CLUSTERED
(
	[Invoice_indclient_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 Menu_items

Reprezentacja pozycji w menu.

- Position_id - klucz główny
- Dish_id - klucz obcy
- Unit_price - cena za porcję
- Quantity_per_unit - ile sztuk danego produktu w 1 porcji
- Date_since - data od kiedy pozycja jest w menu
- Date_to - data do kiedy pozycja była w menu

Warunki integralności:

- Date_since wcześniejsza niż Date_to:

```
ALTER TABLE [dbo].[Menu_items]  WITH CHECK ADD  CONSTRAINT
[c_mi_date_since_earlier_than_date_to] CHECK  (([Date_since]<[Date_to]))
GO
ALTER TABLE [dbo].[Menu_items] CHECK CONSTRAINT
[c_mi_date_since_earlier_than_date_to]
GO
```

- cena za porcję > 0:

```
ALTER TABLE [dbo].[Menu_items]  WITH CHECK ADD  CONSTRAINT [c_mi_unit_price]
CHECK  (([unit_price]>(0)))
GO
ALTER TABLE [dbo].[Menu_items] CHECK CONSTRAINT [c_mi_unit_price]
GO
```

```
CREATE TABLE [dbo].[Menu_items](
	[Position_id] [int] NOT NULL,
	[Dish_id] [int] NOT NULL,
	[Unit_price] [money] NOT NULL,
	[Quantity_per_unit] [varchar](20) NOT NULL,
	[Date_since] [datetime] NOT NULL,
	[Date_to] [datetime] NOT NULL,
 CONSTRAINT [Menu_details_pk] PRIMARY KEY CLUSTERED
(
	[Position_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 Order_details

Reprezentacja szczegółów zamówień.

- Order_details_id - klucz główny
- Order_id - klucz obcy
- Position_id - klucz obcy
- Quantity - ile porcji danego dania zamówiono

Warunki integralności:

- liczba porcji > 0:

```
ALTER TABLE [dbo].[Order_details]  WITH CHECK ADD  CONSTRAINT [c_od_quantity]
CHECK  (([quantity]>(0)))
GO
ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [c_od_quantity]
GO
```

```
CREATE TABLE [dbo].[Order_details](
      [Order_details_id] [int] NOT NULL,
      [Order_id] [int] NOT NULL,
      [Position_id] [int] NOT NULL,
      [Quantity] [int] NOT NULL,
 CONSTRAINT [Order_details_pk] PRIMARY KEY CLUSTERED
(
      [Order_details_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 Order_statuses

Reprezentacja statusów zamówień.

- Order_status_id - klucz główny
- Status - status zamówienia
- Payment_status - status płatności zamówienia

```
CREATE TABLE [dbo].[Order_statuses](
      [Order_status_id] [int] NOT NULL,
      [Status] [varchar](50) NOT NULL,
      [Payment_status] [varchar](50) NOT NULL,
 CONSTRAINT [Order_statuses_pk] PRIMARY KEY CLUSTERED
(
```

```
	[Order_status_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 Orders

Reprezentacja zamówień.

- Order_id - klucz główny
- Client_id - klucz obcy
- Type_of_order_id - klucz obcy
- Table_id - klucz obcy
- Ordered_time - data i czas złożenia zamówienia
- Order_requestedtime - data i czas preferowanego odbioru zamówienia, tj. “na kiedy”
- Order_fulfillment - data i czas realizacji zamówienia
- Employee_id - klucz obcy
- Order_status_id - klucz obcy

Warunki integralności:

- Ordered_time wcześniejszy niż Order_fulfillment:

```
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[c_ord_ordered_time_earlier_then_order_fillfilment] CHECK
(([Ordered_time]<[Order_fulfillment]))
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT
[c_ord_ordered_time_earlier_then_order_fillfilment]
GO
```

- Ordered_time wcześniejszy niż Order_requestedtime:

```
ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[c_ord_ordered_time_earlier_then_order_requestedtime] CHECK
(([Ordered_time]<[Order_requestedtime]))
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT
[c_ord_ordered_time_earlier_then_order_requestedtime]
GO
```

```
CREATE TABLE [dbo].[Orders](
	[Order_id] [int] NOT NULL,
	[Client_id] [int] NULL,
	[Type_of_order_id] [int] NOT NULL,
	[Table_id] [int] NULL,
	[Ordered_time] [datetime] NOT NULL,
	[Order_requestedtime] [datetime] NULL,
	[Order_fulfillment] [datetime] NULL,
```

```
      [Employee_id] [int] NOT NULL,
      [Order_status_id] [int] NOT NULL,
 CONSTRAINT [Orders_pk] 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_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Parameters

Reprezentacja parametrów rezerwacji.

- Param_key - klucz główny
- Value - wartość parametru

```
CREATE TABLE [dbo].[Parameters](
      [Param_key] [varchar](30) NOT NULL,
      [Value] [decimal](12, 5) NOT NULL,
PRIMARY KEY CLUSTERED
(
      [Param_key] 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 Roles

reprezentacja posad pracowników

- Role_id - klucz główny
- Name - nazwa posady
- Description - opis posady

```
CREATE TABLE [dbo].[Roles](
      [Role_id] [int] NOT NULL,
      [Name] [varchar](30) NOT NULL,
      [Description] [varchar](100) NULL,
 CONSTRAINT [Roles_pk] PRIMARY KEY CLUSTERED
(
      [Role_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 Tables
Reprezentacja stolików.

- Table_id - klucz główny
- Number_of_seats - liczba miejsc przy danym stoliku

Warunki integralności:

- liczba miejsc > 0:

```
ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT [c_tables_number]
CHECK  (([number_of_seats]>(0)))
GO
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [c_tables_number]
GO
```

```
CREATE TABLE [dbo].[Tables](
	[Table_id] [int] NOT NULL,
	[Number_of_seats] [int] NOT NULL,
 CONSTRAINT [Tables_pk] 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
```

Tabela Types_of_orders
Reprezentacja rodzajów zamówień.

- Type_of_order_id - klucz główny
- Type - rodzaj zamówienia

```
CREATE TABLE [dbo].[Types_of_orders](
	[Type_of_order_id] [int] NOT NULL,
	[Type] [varchar](50) NOT NULL,
 CONSTRAINT [Types_of_orders_pk] PRIMARY KEY CLUSTERED
(
	[Type_of_order_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
```

Widoki:

Widok Menu
Wyświetla aktualne menu.

```
CREATE VIEW Menu AS
SELECT DISTINCT Position_id, Dish_name, Quantity_per_unit, Unit_price,
Seafood
FROM Menu_items
INNER JOIN Dishes ON Menu_items.Dish_id=Dishes.Dish_id
WHERE GETDATE() BETWEEN Date_since AND Date_to
GO
```

Widok UnrealizedOrders
Wyświetla niezrealizowane zamówienia.

```
CREATE VIEW UnrealizedOrders AS
SELECT Order_id, Client_id, Type_of_order_id, Table_id, Ordered_time,
Order_requestedtime, Order_fulfillment, Employee_id, Payment_status,
Status
FROM Orders
INNER JOIN Order_statuses ON
Orders.Order_status_id=Order_statuses.Order_status_id
WHERE Status = 'Niezrealizowane'
GO
```

Widok PendingReservations
Wyświetla oczekujące na potwierdzenie rezerwacje.

```
CREATE VIEW PendingReservations AS
SELECT 'Firma' AS type_of_client, Company_Reservation_id, Date_since,
Date_to, Company_id, Employee_id, Order_id, Number_of_people, Client_id,
Table_id
FROM Company_Reservations
INNER JOIN CR_Details ON
CR_Details.CR_id=Company_Reservations.Company_Reservation_id
INNER JOIN CR_Statuses ON
CR_Statuses.CR_id=Company_Reservations.Company_Reservation_id
WHERE Status = 'Niezatwierdzone'
UNION
SELECT 'Klient Indywidualny' AS type_of_client, Ind_Reservation_id,
Date_since, Date_to, Individual_client_id,Employee_id, Order_id,
Number_of_people, Client_id, Table_id
FROM Ind_Reservations
INNER JOIN INDR_Details ON
INDR_Details.INDR_id=Ind_Reservations.Ind_Reservation_id
INNER JOIN INDR_Statuses ON
INDR_Statuses.INDR_id=Ind_Reservations.Ind_Reservation_id
```

```
WHERE Status = 'Niezatwierdzone'
GO
```

Widok IndClientOrders
Wyświetla zamówienia dla klienta indywidualnego.

```
CREATE VIEW IndClientOrders AS
SELECT Individual_client_id, Orders.Order_id, Type, Status,
Ordered_time, Order_requestedtime, Order_fulfillment, Dish_name,
Payment_status, Quantity, Unit_price, Unit_price*Quantity AS Value
FROM Orders
INNER JOIN Clients ON Clients.Client_id=Orders.Client_id
INNER JOIN Individual_clients ON
Individual_clients.Client_id=Clients.Client_id
INNER JOIN Types_of_orders ON
Orders.Type_of_order_id=Types_of_orders.Type_of_order_id
INNER JOIN Order_details ON Orders.Order_id=Order_details.Order_id
INNER JOIN Order_statuses ON
Orders.Order_status_id=Order_statuses.Order_status_id
INNER JOIN Menu_items ON
Menu_items.Position_id=Order_details.Position_id
INNER JOIN Dishes ON Dishes.Dish_id=Menu_items.Dish_id
GO
```

Widok IndClientDiscounts
Wyświetla rabaty dla klienta indywidualnego.

```
CREATE VIEW IndClientDiscounts AS
SELECT Individual_client_id, Discount_lib_id, Price, Number_of_orders,
Length_of_discount, Value, Start_date, End_date
FROM Individual_clients
INNER JOIN C_Discounts ON
C_Discounts.Client_id=Individual_clients.Client_id
INNER JOIN Discounts_lib ON
Discounts_lib.Discounts_lib_id=C_Discounts.Discount_lib_id
GO
```

Widok CompanyOrders
Wyświetla zamówienia dla firmy.

```
CREATE VIEW CompanyOrders AS
SELECT Company_id, Type, Status, Ordered_time, Order_requestedtime,
Order_fulfillment, Dish_name, Payment_status, Quantity, Unit_price,
Unit_price*Quantity AS Value
FROM Orders
INNER JOIN Clients ON Clients.Client_id=Orders.Client_id
INNER JOIN Companies ON Companies.Client_id=Clients.Client_id
```

```
INNER JOIN Types_of_orders ON
Orders.Type_of_order_id=Types_of_orders.Type_of_order_id
INNER JOIN Order_details ON Orders.Order_id=Order_details.Order_id
INNER JOIN Order_statuses ON
Orders.Order_status_id=Order_statuses.Order_status_id
INNER JOIN Menu_items ON
Menu_items.Position_id=Order_details.Position_id
INNER JOIN Dishes ON Dishes.Dish_id=Menu_items.Dish_id
GO
```

Widok TablesAllMonthly
Wyświetla miesięczny raport dotyczący rezerwacji stolików.

```
CREATE VIEW [dbo].[TablesAllMonthly] AS
SELECT T1.Table_id, YEAR(O1.Order_fulfillment) as 'year',
MONTH(O1.Order_fulfillment) as 'month',
isnull((SELECT TableINDR_Monthly.[number of reservations] FROM
TableINDR_Monthly
WHERE T1.Table_id = TableINDR_Monthly.[table id] AND
YEAR(O1.Order_fulfillment) = TableINDR_Monthly.year
AND MONTH(O1.Order_fulfillment) = TableINDR_Monthly.month), 0) +
isnull((SELECT TableCR_Monthly.[number of reservations] FROM
TableCR_Monthly
WHERE T1.Table_id = TableCR_Monthly.[table id] AND
YEAR(O1.Order_fulfillment) = TableCR_Monthly.year
AND MONTH(O1.Order_fulfillment) = TableCR_Monthly.month), 0) as 'number
of times used'
from Tables as T1
INNER JOIN Orders AS O1 ON O1.Table_id = T1.Table_id
GROUP BY T1.Table_id, YEAR(O1.Order_fulfillment),
MONTH(O1.Order_fulfillment)
GO
```

Widok TablesAllWeekly
Wyświetla tygodniowy raport dotyczący rezerwacji stolików.

```
CREATE VIEW [dbo].[TablesAllWeekly] AS
SELECT T1.Table_id, YEAR(O1.Order_fulfillment) as 'year',
MONTH(O1.Order_fulfillment) as 'month',
DATEPART(week, O1.Order_fulfillment) as 'week',
isnull((SELECT TableINDR_Weekly.[number of reservations] FROM
TableINDR_Weekly
WHERE T1.Table_id = TableINDR_Weekly.[table id] AND
YEAR(O1.Order_fulfillment) = TableINDR_Weekly.year
AND MONTH(O1.Order_fulfillment) = TableINDR_Weekly.month
AND DATEPART(week, O1.Order_fulfillment) = TableINDR_Weekly.week), 0) +
isnull((SELECT TableCR_Weekly.[number of reservations] FROM
TableCR_Weekly
WHERE T1.Table_id = TableCR_Weekly.[table id] AND
YEAR(O1.Order_fulfillment) = TableCR_Weekly.year
AND MONTH(O1.Order_fulfillment) = TableCR_Weekly.month
```

```
AND DATEPART(week, O1.Order_fulfillment) = TableCR_Weekly.week), 0) as
'number of times used'
from Tables as T1
INNER JOIN Orders AS O1 ON O1.Table_id = T1.Table_id
GROUP BY T1.Table_id, YEAR(O1.Order_fulfillment),
MONTH(O1.Order_fulfillment),
DATEPART(week, O1.Order_fulfillment)
GO
```

Widok TableCR_Monthly
Wyświetla miesięczny raport dotyczący rezerwacji stolików (rezerwacje firmowe).

```
CREATE VIEW [dbo].[TableCR_Monthly] AS
SELECT Tables.Table_id as 'table id',
YEAR(Company_Reservations.Date_since) as 'year',
MONTH(Company_Reservations.Date_since) as 'month',
isnull(count(*), 0) 'number of reservations' FROM Tables
INNER JOIN (CR_Details INNER JOIN Company_Reservations
ON Company_Reservations.Company_Reservation_id = CR_Details.CR_id
INNER JOIN CR_Statuses on CR_Status_id =
Company_Reservations.Company_Reservation_id)
ON Tables.Table_id = CR_Details.Table_id
WHERE CR_Statuses.Status = 'Zatwierdzone'
GROUP BY Tables.Table_id, YEAR(Company_Reservations.Date_since),
MONTH(Company_Reservations.Date_since)
GO
```

Widok TableCR_Weekly
Wyświetla tygodniowy raport dotyczący rezerwacji stolików (rezerwacje firmowe).

```
CREATE VIEW [dbo].[TableCR_Weekly] AS
SELECT Tables.Table_id as 'table
id',YEAR(Company_Reservations.Date_since) as 'year',
MONTH(Company_Reservations.Date_since) as 'month', DATEPART(week,
Company_Reservations.Date_since) as 'week',
isnull(count(*), 0) 'number of reservations' FROM Tables
INNER JOIN (CR_Details INNER JOIN Company_Reservations
ON Company_Reservations.Company_Reservation_id = CR_Details.CR_id
INNER JOIN CR_Statuses on CR_Status_id =
Company_Reservations.Company_Reservation_id)
ON Tables.Table_id = CR_Details.Table_id
WHERE CR_Statuses.Status = 'Zatwierdzone'
GROUP BY Tables.Table_id, YEAR(Company_Reservations.Date_since),
MONTH(Company_Reservations.Date_since),
DATEPART(week, Company_Reservations.Date_since)
GO
```

Widok TableINDR_Monthly
Wyświetla miesięczny raport dotyczący rezerwacji stolików (rezerwacje firmowe).

```
CREATE VIEW [dbo].[TableINDR_Monthly] AS
SELECT Tables.Table_id as 'table id', YEAR(Ind_Reservations.Date_since)
as 'year',
MONTH(Ind_Reservations.Date_since) as 'month',
isnull(count(*), 0) 'number of reservations' FROM Tables
INNER JOIN (INDR_Details INNER JOIN Ind_Reservations
ON Ind_Reservations.Ind_Reservation_id = INDR_Details.INDR_id
INNER JOIN INDR_Statuses on INDR_Status_id =
Ind_Reservations.Ind_Reservation_id)
ON Tables.Table_id = INDR_Details.Table_id
WHERE INDR_Statuses.Status = 'Zatwierdzone'
GROUP BY Tables.Table_id, YEAR(Ind_Reservations.Date_since),
MONTH(Ind_Reservations.Date_since)
GO
```

Widok TableINDR_Weekly

Wyświetla tygodniowy raport dotyczący rezerwacji stolików (rezerwacje firmowe).

```
CREATE VIEW [dbo].[TableINDR_Weekly] AS
SELECT Tables.Table_id as 'table id',YEAR(Ind_Reservations.Date_since)
as 'year',
MONTH(Ind_Reservations.Date_since) as 'month', DATEPART(week,
Ind_Reservations.Date_since) as 'week',
isnull(count(*), 0) 'number of reservations' FROM Tables
INNER JOIN (INDR_Details INNER JOIN Ind_Reservations
ON Ind_Reservations.Ind_Reservation_id = INDR_Details.INDR_id
INNER JOIN INDR_Statuses on INDR_Status_id =
Ind_Reservations.Ind_Reservation_id)
ON Tables.Table_id = INDR_Details.Table_id
WHERE INDR_Statuses.Status = 'Zatwierdzone'
GROUP BY Tables.Table_id, YEAR(Ind_Reservations.Date_since),
MONTH(Ind_Reservations.Date_since),
DATEPART(week, Ind_Reservations.Date_since)
GO
```

Widok last_month_discounts

Zawiera listę aktywnych rabatów klientów w ostatnim miesiącu, daty rabatów są przycięte do granic miesiąca.

```
CREATE VIEW last_month_discounts AS
SELECT  o.client_id,
       dl.value,
       (CASE WHEN cd.start_date<DATEADD(DD,-30,GETDATE()) THEN
DATEADD(DD,-30,GETDATE())
             ELSE cd.start_date END) AS start_m,
       (CASE WHEN cd.end_date>GETDATE() THEN GETDATE()
             ELSE cd.end_date END) AS end_m
FROM C_Discounts AS cd
        INNER JOIN Discounts_lib as dl on
```

```
cd.Discount_lib_id=dl.Discounts_lib_id
        INNER JOIN Individual_clients AS ic on ic.client_id=cd.client_id
        INNER JOIN clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
WHERE ((GETDATE()>cd.Start_date and GETDATE()<cd.End_date) or
      cd.end_date>DATEADD(DD,-30,GETDATE()) and cd.start_date<GETDATE())
```

Widok last_month_client_sales_discount

Zawiera informacje o kliencie, liczbę jego aktywnych rabatach, wartość zakupów bez rabatu i wartość rabatu.

```
CREATE VIEW last_month_client_sales_discount AS
SELECT o.client_id,
      (SELECT COUNT(*) FROM last_month_discounts where
last_month_discounts.client_id=o.client_id) as count_of_discounts,
      sum(mi.unit_price*od.quantity) as value_without_discount,
sum(mi.unit_price*od.quantity*discounts.value) as discount_value
FROM Orders as o
        INNER JOIN Order_details as od on od.order_id=o.order_id
        INNER JOIN Menu_items as mi ON mi.Position_id=od.Position_id
        INNER JOIN last_month_discounts
   AS discounts ON discounts.client_id=o.client_id
WHERE (o.Ordered_time>discounts.start_m and
o.Ordered_time<=discounts.end_m)
GROUP BY o.client_id
```

Widok last_week_discounts

Zawiera listę aktywnych rabatów klientów w ostatnim tygodniu, daty rabatów są przycięte do granic tygodnia (ostatnich 7 dni).

```
CREATE VIEW last_week_discounts AS
SELECT  o.client_id,
       dl.value,
       (CASE WHEN cd.start_date<DATEADD(DD,-7,GETDATE()) THEN
DATEADD(DD,-7,GETDATE())
             ELSE cd.start_date END) AS start_m,
       (CASE WHEN cd.end_date>GETDATE() THEN GETDATE()
             ELSE cd.end_date END) AS end_m
FROM C_Discounts AS cd
        INNER JOIN Discounts_lib as dl on
cd.Discount_lib_id=dl.Discounts_lib_id
        INNER JOIN Individual_clients AS ic on ic.client_id=cd.client_id
        INNER JOIN clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
WHERE ((GETDATE()>=cd.Start_date and GETDATE()<cd.End_date) or
      cd.end_date>DATEADD(DD,-7,GETDATE()) and cd.start_date<=GETDATE())
```

Widok last_week_client_sales_discount

Zawiera informacje o kliencie, liczbę jego aktywnych rabatach, wartość zakupów bez rabatu i wartość rabatu w ostatnim tygodniu.

```
CREATE VIEW last_week_client_sales_discount AS
SELECT o.client_id,
      (SELECT COUNT(*) FROM last_week_discounts where
last_week_discounts.client_id=o.client_id) as count_of_discounts,
      sum(mi.unit_price*od.quantity) as value_without_discount,
sum(mi.unit_price*od.quantity*discounts.value) as discount_value
FROM Orders as o
        INNER JOIN Order_details as od on od.order_id=o.order_id
        INNER JOIN Menu_items as mi ON mi.Position_id=od.Position_id
        INNER JOIN last_week_discounts
   AS discounts ON discounts.client_id=o.client_id
WHERE (o.Ordered_time>discounts.start_m and
o.Ordered_time<=discounts.end_m)
GROUP BY o.client_id
```

Widok last_month_menu

Dla każdej pozycji menu zawiera cenę jednostkową, daty wprowadzenia i usunięcia z menu (przycięte do granic miesiąca), liczbę sprzedanych dań, oraz informacje czy pozycja została w ostatnim miesiącu dodana lub usunięta z menu.

```
CREATE VIEW last_month_menu AS
SELECT  mi.position_id,
       mi.unit_price,
       (CASE WHEN mi.date_since<DATEADD(DD,-30,GETDATE()) THEN
DATEADD(DD,-30,GETDATE())
             ELSE mi.date_since END) AS start_m,
       (CASE WHEN mi.date_to>GETDATE() THEN GETDATE()
             ELSE mi.date_to END) AS end_m,
       (SELECT sum(quantity) from order_details where
order_details.position_id=mi.position_id group by position_id) as
sold_items,
       (CASE WHEN mi.date_since>DATEADD(DD,-30,GETDATE()) THEN 1
             ELSE 0 END) AS added_to_menu,
       (CASE WHEN mi.date_to<GETDATE() THEN 1
             ELSE 0 END) AS removed_from_menu
FROM Menu_items AS mi
WHERE ((GETDATE()>=mi.date_since and GETDATE()<mi.date_to) or
      mi.date_to>DATEADD(DD,-30,GETDATE()) and mi.date_to<=GETDATE())
```

Widok last_week_menu

Dla każdej pozycji menu zawiera cenę jednostkową, daty wprowadzenia i usunięcia z menu (przycięte do granic tygodnia - ostatnich 7 dni), liczbę sprzedanych dań, oraz informacje czy pozycja została w ostatnim miesiącu dodana lub usunięta z menu.

```
CREATE VIEW last_week_menu AS
SELECT  mi.position_id,
       mi.unit_price,
       (CASE WHEN mi.date_since<DATEADD(DD,-7,GETDATE()) THEN
DATEADD(DD,-7,GETDATE())
             ELSE mi.date_since END) AS start_m,
       (CASE WHEN mi.date_to>GETDATE() THEN GETDATE()
             ELSE mi.date_to END) AS end_m,
       (SELECT sum(quantity) from order_details where
order_details.position_id=mi.position_id group by position_id) as
sold_items,
       (CASE WHEN mi.date_since>DATEADD(DD,-7,GETDATE()) THEN 1
             ELSE 0 END) AS added_to_menu,
       (CASE WHEN mi.date_to<GETDATE() THEN 1
             ELSE 0 END) AS removed_from_menu
FROM Menu_items AS mi
WHERE ((GETDATE()>=mi.date_since and GETDATE()<mi.date_to) or
      mi.date_to>DATEADD(DD,-7,GETDATE()) and mi.date_to<=GETDATE())
```

Widok client_orders_monthly

Zawiera informacje o sumarycznej wartości zamówień dla klienta indywidualnego w ostatnim miesiącu.

```
GO
CREATE VIEW client_orders_monthly AS
SELECT ic.client_id, sum(od.quantity*mi.unit_price) as summary_value
FROM Individual_clients as ic
        INNER JOIN Clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-30,GETDATE())
GROUP BY ic.client_id
```

Widok client_orders_weekly

Zawiera informacje o sumarycznej wartości zamówień dla klienta indywidualnego w ostatnim tygodniu.

```
CREATE VIEW client_orders_weekly AS
SELECT ic.client_id, sum(od.quantity*mi.unit_price) as summary_value
FROM Individual_clients as ic
        INNER JOIN Clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-7,GETDATE())
GROUP BY ic.client_id
```

Widok company_orders_monthly
Zawiera informacje o sumarycznej wartości zamówień dla klienta firmowego w ostatnim miesiącu.

```
CREATE VIEW company_orders_monthly AS
SELECT cc.client_id, sum(od.quantity*mi.unit_price) as summary_value
FROM Companies as cc
        INNER JOIN Clients as c on c.client_id=cc.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-30,GETDATE())
GROUP BY cc.client_id
```

Widok company_orders_weekly
Zawiera informacje o sumarycznej wartości zamówień dla klienta firmowego ostatnim tygodniu.

```
CREATE VIEW company_orders_weekly AS
SELECT cc.client_id, sum(od.quantity*mi.unit_price) as summary_value
FROM Companies as cc
        INNER JOIN Clients as c on c.client_id=cc.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-7,GETDATE())
GROUP BY cc.client_id
```

Widok last_month_all_client_orders
Zawiera informacje o wartości oraz dacie wszystkich zamówień dla każdego klienta indywidualnego z ostatniego miesiąca.

```
CREATE VIEW last_month_all_client_orders AS
SELECT ic.client_id, sum(od.quantity*mi.unit_price) as summary_value,
o.ordered_time
FROM Individual_clients as ic
        INNER JOIN Clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-30,GETDATE())
GROUP BY ic.client_id, o.ordered_time
```

Widok last_week_all_client_orders
Zawiera informacje o wartości oraz dacie wszystkich zamówień dla każdego klienta indywidualnego z ostatniego tygodnia.

```
CREATE VIEW last_week_all_client_orders AS
SELECT ic.client_id, sum(od.quantity*mi.unit_price) as summary_value,
o.ordered_time
FROM Individual_clients as ic
        INNER JOIN Clients as c on c.client_id=ic.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-7,GETDATE())
GROUP BY ic.client_id, o.ordered_time
```

Widok last_month_all_company_orders

Zawiera informacje o wartości oraz dacie wszystkich zamówień dla każdego klienta firmowego z ostatniego miesiąca.

```
CREATE VIEW last_month_all_company_orders AS
SELECT cc.client_id, sum(od.quantity*mi.unit_price) as summary_value,
o.ordered_time
FROM Companies as cc
        INNER JOIN Clients as c on c.client_id=cc.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-30,GETDATE())
GROUP BY cc.client_id, o.ordered_time
```

Widok last_week_all_company_orders

Zawiera informacje o wartości oraz dacie wszystkich zamówień dla każdego klienta firmowego z ostatniego tygodnia.

```
CREATE VIEW last_week_all_company_orders AS
SELECT cc.client_id, sum(od.quantity*mi.unit_price) as summary_value,
o.ordered_time
FROM Companies as cc
        INNER JOIN Clients as c on c.client_id=cc.client_id
        INNER JOIN Orders as o on o.client_id=c.client_id
        INNER JOIN Order_details as od on od.Order_id=o.order_id
        INNER JOIN menu_items as mi on mi.position_id=od.position_id
where o.ordered_time>=DATEADD(DD,-7,GETDATE())
GROUP BY cc.client_id, o.ordered_time
```

Widok Reservations_Today

Wyświetla rezerwacje na dzień dzisiejszy.

```
CREATE VIEW Reservations_Today AS
SELECT 'Company' AS Client_type, Company_Reservation_id, Date_since,
Date_to, Companies.Client_id AS Client_id, Companies.Company_id AS
```

```
Company_or_Ind_Client_id, Order_id, Table_id, Number_of_people FROM
Company_Reservations
INNER JOIN CR_Details ON
CR_Details.CR_id=Company_Reservations.Company_Reservation_id
INNER JOIN Companies ON
Companies.Company_id=Company_Reservations.Company_id
WHERE YEAR(GETDATE())=YEAR(Date_since) AND
MONTH(GETDATE())=MONTH(Date_since) AND DAY(GETDATE())=DAY(Date_since)
UNION
SELECT 'Individual Client' AS Client_type, Ind_Reservation_id,
Date_since, Date_to, Individual_clients.Client_id AS Client_id,
Individual_clients.Individual_client_id AS Company_or_Ind_Client_id,
Order_id, Table_id, Number_of_people  FROM Ind_Reservations
INNER JOIN INDR_Details ON
INDR_Details.INDR_id=Ind_Reservations.Ind_Reservation_id
INNER JOIN Individual_clients On
Individual_clients.Individual_client_id=Ind_Reservations.Individual_clie
nt_id
WHERE YEAR(GETDATE())=YEAR(Date_since) AND
MONTH(GETDATE())=MONTH(Date_since) AND DAY(GETDATE())=DAY(Date_since)
```

Procedury:

Procedura AddCategory
Procedura dodaje kategorię potraw.

```
CREATE PROCEDURE AddCategory
@Category_name varchar(50),
@Description varchar(100)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		DECLARE @Category_id INT
		SELECT @Category_id = ISNULL(MAX(Category_id), 0) + 1
		FROM Categories
		INSERT INTO Categories(Category_id, Category_name,
Description)
		VALUES(@Category_id, @Category_name, @Description);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu kategorii: ' +
ERROR_MESSAGE();
		THROW 5200, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddDish
Procedura dodaje potrawę.

```
CREATE PROCEDURE AddDish
@Dish_name varchar(50),
@Category_name varchar(50),
@Seafood varchar(10)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF EXISTS(
			SELECT * FROM Dishes
			WHERE Dish_name=@Dish_name
			)
		BEGIN;
			THROW 52000, 'Potrawa już istnieje', 1
		END
		IF NOT EXISTS(
			SELECT * FROM Categories
			WHERE Category_name=@Category_name
			)
		BEGIN;
			THROW 52000, 'Nie istnieje taka kategoria', 1
```

```
            END
            DECLARE @Category_id INT
            SELECT @Category_id = Category_id
            FROM Categories
            WHERE Category_name=@Category_name
            DECLARE @Dish_id INT
            SELECT @Dish_id = ISNULL(MAX(Dish_id), 0) + 1
            FROM Dishes
            INSERT INTO Dishes(Dish_id, Dish_name, Category_id, Seafood)
            VALUES(@Dish_id, @Dish_name, @Category_id, @Seafood);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu potrawy: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddMenu_item

Procedura dodaje pozycję do tabeli Menu_items.

```
CREATE PROCEDURE AddMenuItem
@Dish_name varchar(50),
@Unit_price money,
@Quantity_per_unit varchar(20),
@Date_since datetime,
@Date_to datetime
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Dishes
                  WHERE Dish_name=@Dish_name
                  )
            BEGIN;
                  THROW 52000, 'Nie istnieje taka potrawa', 1
            END
            DECLARE @Dish_id INT
            SELECT @Dish_id = Dish_id
            FROM Dishes
            WHERE Dish_name=@Dish_name
            DECLARE @Position_id INT
            SELECT @Position_id = ISNULL(MAX(Position_id), 0) + 1
            FROM Menu_items
            INSERT INTO Menu_items(Position_id, Dish_id, Unit_price,
Quantity_per_unit, Date_since, Date_to )
            VALUES(@Position_id, @Dish_id, @Unit_price,
@Quantity_per_unit, @Date_since, @Date_since);
      END TRY
      BEGIN CATCH
```

```
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu pozycji menu: ' +
ERROR_MESSAGE();\
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddClient
Procedura dodaje klienta.

```
CREATE PROCEDURE AddClient
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            DECLARE @Client_id INT
            SELECT @Client_id = ISNULL(MAX(Client_id), 0) + 1
            FROM Clients
            INSERT INTO Clients(Client_id)
            VALUES(@Client_id);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu klienta: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddExternalClient
Procedura dodaje klienta zewnętrznego.

```
CREATE PROCEDURE AddExternalClient
@Invoice_id INT
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            DECLARE @External_client_id INT
            SELECT @External_client_id = ISNULL(MAX(External_client_id),
0) + 1
            FROM External_clients
            EXEC AddClient
            DECLARE @Client_id INT
            SELECT @Client_id = ISNULL(MAX(Client_id), 0)
            FROM Clients
            INSERT INTO External_clients(External_client_id, Client_id,
Invoice_id)
```

```
            VALUES(@External_client_id, @Client_id, @Invoice_id);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu klienta zewnętrznego: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddIndClient
Procedura dodaje klienta indywidualnego.

```
CREATE PROCEDURE AddIndClient
@Phone varchar(24),
@Company_id INT,
@Invoice_id INT
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF EXISTS(
                  SELECT * FROM Individual_clients
                  WHERE Phone=@Phone
                  )
            BEGIN;
                  THROW 52000, 'Ten numer telefonu jest już użyty.', 1
            END
            DECLARE @Individual_client_id INT
            SELECT @Individual_client_id =
ISNULL(MAX(Individual_client_id), 0) + 1
            FROM Individual_clients
            EXEC AddClient
            DECLARE @Client_id INT
            SELECT @Client_id = ISNULL(MAX(Client_id), 0)
            FROM Clients
            INSERT INTO Individual_clients(Individual_client_id,
Client_id, Phone, Company_id, Invoice_id)
            VALUES(@Individual_client_id, @Client_id, @Phone,
@Company_id, @Invoice_id);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu klienta indywidualnego: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddCompany
Procedura dodaje firmę.

```
CREATE PROCEDURE AddCompany
@Company_name varchar(40),
@NIP varchar(50),
@Address varchar(60),
@Phone varchar(24)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF EXISTS(
			SELECT * FROM Companies
			WHERE Phone=@Phone
			)
		BEGIN;
			THROW 52000, 'Ten numer telefonu jest już użyty.', 1
		END
		DECLARE @Company_id INT
		SELECT @Company_id = ISNULL(MAX(Company_id), 0) + 1
		FROM Companies
		EXEC AddClient
		DECLARE @Client_id INT
		SELECT @Client_id = ISNULL(MAX(Client_id), 0)
		FROM Clients
		INSERT INTO Companies(Company_id, Company_name, Client_id,
NIP, Address, Phone)
		VALUES(@Company_id,@Company_id, @Client_id, @NIP, @Address,
@Phone);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu firmy: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddDiscount
Procedura dodaje rabat do Discounts_lib.

```
CREATE PROCEDURE AddDiscount
@Type varchar(10),
@Price money,
@Number_of_orders int,
@Length_of_discount int,
@Value decimal(10,2),
@Date_since datetime,
```

```
@Date_to datetime
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF (@Type<>'R1' AND @Type<>'R2')
		BEGIN;
			THROW 52000, 'Rabaty mogą być tylko typu R1 lub R2', 1
		END
		IF EXISTS(
			SELECT * FROM Discounts_lib
			WHERE @Date_since<(SELECT MAX(Date_to) FROM
Discounts_lib WHERE @Type=Type)
			)
		BEGIN;
			THROW 52000, 'Data początkowa nowego rabatu musi być
późniejsza niż data końcowa rabatu tego samego typu.', 1
		END
		IF (@Type='R1' AND (@Number_of_orders IS NULL OR
@Length_of_discount IS NOT NULL))
		BEGIN;
			THROW 52000, 'Rabat typu R1 musi mieć podaną liczbę
zamówień i nie może mieć podanej długości trwania, ponieważ jest
jednorazowy', 1
		END
		IF (@Type='R2' AND (@Length_of_discount IS NULL OR
@Number_of_orders IS NOT NULL))
		BEGIN;
			THROW 52000, 'Rabat typu R2 musi mieć podaną długość
trwania i nie uwzględnia liczby zamówień', 1
		END
		DECLARE @Discounts_lib_id INT
		SELECT @Discounts_lib_id = ISNULL(MAX(Discounts_lib_id), 0)
+ 1
		FROM Discounts_lib
		INSERT INTO Discounts_lib(Discounts_lib_id, Type, Price,
Number_of_orders, Length_of_discount, Value, Date_since, Date_to)
		VALUES(@Discounts_lib_id, @Type, @Price, @Number_of_orders,
@Length_of_discount, @Value, @Date_since, @Date_to);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu rabatu: ' +
ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddC_Discount
Procedura dodaje rabat dla klienta do C_Discounts.

```
CREATE PROCEDURE AddC_Discounts
@Client_id int,
@Discounts_lib_id int,
@Start_date datetime
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
				SELECT * FROM Individual_clients
				WHERE Individual_client_id=@Client_id
				)
		BEGIN;
		 THROW 52000, 'Nie istnieje taki klient indywidualny.', 1
		END
			IF NOT EXISTS(
				SELECT * FROM Discounts_lib
				WHERE Discounts_lib_id=@Discounts_lib_id
				)
		BEGIN;
		 THROW 52000, 'Nie istnieje taki rabat.', 1
		END
		DECLARE @C_Discounts_id INT
		SELECT @C_Discounts_id = ISNULL(MAX(C_Discounts_id), 0) + 1
		FROM C_Discounts
			DECLARE @Length_of_discount int = ISNULL((SELECT
Length_of_discount FROM Discounts_lib WHERE
Discounts_lib_id=@Discounts_lib_id),0)
			DECLARE @End_date datetime
			IF @Length_of_discount = 0
				SELECT @End_date=NULL
			ELSE
				SELECT @End_date=DATEADD(day,
@Length_of_discount, @Start_date)
		INSERT INTO C_Discounts(C_Discounts_id, Discount_lib_id,
Client_id, Start_date, End_date)
		VALUES(@C_Discounts_id, @Discounts_lib_id, @Client_id,
@Start_date, @End_date);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu rabatu: ' +
ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddRole
Procedura dodaje rolę.

```
CREATE PROCEDURE AddRole
@Name varchar(30),
@Description varchar(100)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF EXISTS(
			SELECT * FROM Roles
			WHERE Name=@Name
			)
		BEGIN;
				THROW 52000, 'Rola już istnieje', 1
		END
		DECLARE @Role_id INT
		SELECT @Role_id = ISNULL(MAX(Role_id), 0) + 1
		FROM Roles
		INSERT INTO Roles(Role_id, Name, Description)
		VALUES(@Role_id, @Name, @Description);
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
			='Błąd przy dodawaniu roli: ' + ERROR_MESSAGE();
		THROW 5200, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddEmployee
Procedura dodaje pracownika.

```
CREATE PROCEDURE AddEmployee
@Firstname varchar(30),
@Lastname varchar(30),
@Role_name varchar(30)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM Roles
			WHERE Name=@Role_name
			)
	BEGIN;
			THROW 52000, 'Nie istnieje taka rola', 1
	END
		DECLARE @Role_id INT
	SELECT @Role_id = Role_id
	FROM Roles
	WHERE Name=@Role_name
```

```
            DECLARE @Employee_id INT
            SELECT @Employee_id = ISNULL(MAX(Employee_id), 0) + 1
            FROM Employees
            INSERT INTO Employees(Employee_id, Firstname, Lastname,
Role_id)
            VALUES(@Employee_id, @Firstname, @Lastname, @Role_id);
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu pracownika: ' +
ERROR_MESSAGE();
            THROW 5200, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddIndClientInvoiceDetails
Procedura dodaje szczegóły do faktury dla klienta indywidualnego.

```
CREATE PROCEDURE AddIndClientInvoiceDetails
@Individual_client_id int,
@First_name varchar(20),
@Last_name varchar(20),
@Address varchar(50),
@Phone varchar(24)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Individual_clients
                  WHERE Individual_client_id=@Individual_client_id
                  )
            BEGIN;
                  THROW 52000, 'Podany klient indywidualny nie
istnieje.', 1
            END
            DECLARE @Invoice_id INT
            SELECT @Invoice_id = ISNULL(MAX(Invoice_indclient_id), 0) +
1
            FROM Invoice_details
            INSERT INTO Invoice_details(Invoice_indclient_id,
First_name, Last_name, Address, Phone)
            VALUES(@Invoice_id, @First_name, @Last_name, @Address,
@Phone);
            BEGIN
                  UPDATE Individual_clients
                        SET Invoice_id = @Invoice_id
                        WHERE
Individual_clients.Individual_client_id=@Individual_client_id
            END

      END TRY
```

```
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu szczegółów do faktury
klienta indywidualnego' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddExtClientInvoiceDetails

Procedura dodaje szczegóły do faktury dla klienta zewnętrznego.

```
CREATE PROCEDURE AddExtClientInvoiceDetails
@External_client_id int,
@First_name varchar(20),
@Last_name varchar(20),
@Address varchar(50),
@Phone varchar(24)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM External_clients
			WHERE External_client_id=@External_client_id
			)
		BEGIN;
			THROW 52000, 'Podany klient zewnętrzny nie
istnieje.',1
		END
		DECLARE @Invoice_id INT
		SELECT @Invoice_id = ISNULL(MAX(Invoice_indclient_id), 0)+1
		FROM Invoice_details
		INSERT INTO Invoice_details(Invoice_indclient_id,
First_name, Last_name, Address, Phone)
		VALUES(@Invoice_id, @First_name, @Last_name, @Address,
@Phone);
		BEGIN
			UPDATE External_clients
				SET Invoice_id = @Invoice_id
				WHERE
External_clients.External_client_id=@External_client_id
		END

	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu szczegółów do faktury
klienta zewnętrznego:' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddOrder
Procedura dodaje zamówienie.

```
CREATE PROCEDURE AddOrder
@Client_id int,
@Type_of_order_id int,
@Table_id int,
@Order_requestedtime datetime,
@Employee_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM Clients
			WHERE Client_id=@Client_id
			)
		BEGIN;
			THROW 52000, 'Podany klient nie istnieje.', 1
		END
		IF NOT EXISTS(
			SELECT * FROM Employees
			WHERE Employee_id=@Employee_id
			)
		BEGIN;
			THROW 52000, 'Podany pracownik nie istnieje.', 1
		END
		IF NOT EXISTS(
			SELECT * FROM Tables
			WHERE Table_id=@Table_id
			)
		BEGIN;
			THROW 52000, 'Podany stolik nie istnieje.', 1
		END
		IF @Type_of_order_id=2 AND @Table_id IS NOT NULL
		BEGIN;
			THROW 52000, 'Jeśli rodzaj zamówienia jest "na wynos",
nie podawaj id stolika.', 1
		END
		IF @Type_of_order_id=1 AND @Table_id IS NULL
		BEGIN;
			THROW 52000, 'Jeśli rodzaj zamówienia jest "na
miejscu", podaj id stolika.', 1
		END
		DECLARE @Ordered_time datetime = GETDATE()
		DECLARE @Order_fulfillment datetime = NULL
		DECLARE @Order_status_id int = 1
		DECLARE @Order_id INT
		SELECT @Order_id = ISNULL(MAX(Order_id), 0) + 1
		FROM Orders
		INSERT INTO Orders(Order_id, Client_id, Type_of_order_id,
Table_id, Ordered_time, Order_requestedtime, Order_fulfillment,
Employee_id, Order_status_id)
```

```
            VALUES(@Order_id, @Client_id, @Type_of_order_id, @Table_id,
@Ordered_time, @Order_requestedtime, @Order_fulfillment, @Employee_id,
@Order_status_id);

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu zamówienia:' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddOrderDetails

Procedura dodaje szczegóły zamówienia.

```
CREATE PROCEDURE AddOrderDetails
@Order_id int,
@Position_id int,
@Quantity int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Orders
                  WHERE Order_id=@Order_id
                  )
            BEGIN;
                  THROW 52000, 'Podane zamówienie nie istnieje.', 1
            END
            IF (NOT EXISTS(
                  SELECT * FROM Menu_items
                  WHERE Position_id=@Position_id
                  )) OR NOT ((SELECT Orders.Order_requestedtime FROM
Orders WHERE Orders.Order_id = @Order_id) BETWEEN
                  (SELECT Menu_items.Date_since FROM Menu_items WHERE
Menu_items.Position_id = @Position_id) AND
                  (SELECT Menu_items.Date_to FROM Menu_items WHERE
Menu_items.Position_id = @Position_id))
            BEGIN;
                  THROW 52000, 'Podana pozycja menu nie istnieje.', 1
            END
            IF (SELECT Dishes.Seafood FROM Dishes INNER JOIN Menu_items
ON Menu_items.Dish_id = Dishes.Dish_id
            WHERE Menu_items.Position_id = @Position_id) = 'yes'
            AND (DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime
FROM Orders WHERE Orders.Order_id = @Order_id)) NOT LIKE 'Thursday'
```

```
            AND DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime
FROM Orders WHERE Orders.Order_id = @Order_id)) NOT LIKE 'Friday'
            AND DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime
FROM Orders WHERE Orders.Order_id = @Order_id)) NOT LIKE 'Saturday')
            BEGIN;
                  THROW 52000, 'Nie można zamówić owoców morza na ten
dzień', 1
            END
            IF (SELECT Dishes.Seafood FROM Dishes INNER JOIN Menu_items
ON Menu_items.Dish_id = Dishes.Dish_id
            WHERE Menu_items.Position_id = @Position_id) = 'yes' AND
            ((DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime FROM
Orders WHERE Orders.Order_id = @Order_id)) LIKE 'Thursday'
            AND DATEDIFF(day, (SELECT Orders.Ordered_time FROM Orders
WHERE Orders.Order_id = @Order_id),
            (SELECT Orders.Order_requestedtime FROM Orders WHERE
Orders.Order_id = @Order_id)) <=2) OR
            (DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime FROM
Orders WHERE Orders.Order_id = @Order_id)) LIKE 'Friday'
            AND DATEDIFF(day, (SELECT Orders.Ordered_time FROM Orders
WHERE Orders.Order_id = @Order_id),
            (SELECT Orders.Order_requestedtime FROM Orders WHERE
Orders.Order_id = @Order_id)) <=3) OR
            (DATENAME(WEEKDAY, (SELECT Orders.Order_requestedtime FROM
Orders WHERE Orders.Order_id = @Order_id)) LIKE 'Saturday'
            AND DATEDIFF(day, (SELECT Orders.Ordered_time FROM Orders
WHERE Orders.Order_id = @Order_id),
            (SELECT Orders.Order_requestedtime FROM Orders WHERE
Orders.Order_id = @Order_id)) <=4))
            BEGIN;
                  THROW 52000, 'Owoce morza zamówione za późno', 1
            END
            DECLARE @Order_details_id INT
            SELECT @Order_details_id= ISNULL(MAX(Order_details_id), 0) +
1
            FROM Order_details
            INSERT INTO Order_details(Order_details_id, Order_id,
Position_id, Quantity)
            VALUES(@Order_details_id, @Order_id, @Position_id,
@Quantity);

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu szczegółów zamówienia:' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddIndReservation
Procedura dodaje rezerwację dla klienta indywidualnego.

```
CREATE PROCEDURE AddIndReservation
@Date_since datetime,
@Date_to datetime,
@Individual_client_id int,
@Employee_id int,
@Order_id int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY

         IF NOT EXISTS(
                    SELECT * FROM Individual_clients
                    WHERE Individual_client_id=@Individual_client_id
                    )
         BEGIN;
                    THROW 52000, 'Podany klient indywidualny nie
istnieje.', 1
         END
                  IF ([dbo].isWZfulfilled(@Order_id) = 0 OR
[dbo].isWKfulfilled(@Individual_client_id) = 0)
                  BEGIN;
                        THROW 52000, 'Nie spełnia warunków rezerwacji', 1
                  END
         IF NOT EXISTS(
                    SELECT * FROM Employees
                    WHERE Employee_id=@Employee_id
                    )
         BEGIN;
                    THROW 52000, 'Podany pracownik nie istnieje.', 1
         END
         DECLARE @Ind_reservation_id INT
         SELECT @Ind_reservation_id= ISNULL(MAX(Ind_reservation_id), 0)
+ 1
         FROM Ind_Reservations
         INSERT INTO Ind_Reservations(Ind_reservation_id, Date_since,
Date_to, Individual_client_id, Employee_id, Order_id)
         VALUES(@Ind_reservation_id, @Date_since, @Date_to,
@Individual_client_id, @Employee_id, @Order_id);

      END TRY
      BEGIN CATCH
```

```
        DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu rezerwacji klienta
indywidualnego: ' + ERROR_MESSAGE();
        THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddINDRDetails

Procedura dodaje szczegóły rezerwacji dla klienta indywidualnego.

```
CREATE PROCEDURE AddINDRDetails
@INDR_id int,
@Table_id int,
@Number_of_people int,
@Client_id int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Ind_Reservations
                  WHERE Ind_Reservation_id=@INDR_id
                  )
            BEGIN;
                  THROW 52000, 'Podana rezerwacja nie istnieje.', 1
            END
            IF @Number_of_people > (SELECT Tables.Number_of_seats from
Tables WHERE Tables.Table_id = @Table_id)
             BEGIN;
                   THROW 52000, 'Przy tym stoliku nie ma tyle miejsc.',
1
             END

            IF NOT EXISTS(
                  SELECT * FROM Tables
                  WHERE Table_id=@Table_id
                  )
            BEGIN;
                  THROW 52000, 'Podany stolik nie istnieje.', 1
            END
                  IF @Number_of_people<2
            BEGIN;
                  THROW 52000, 'Rezerwację można zarezerwować na
minimalnie 2 osoby', 1
            END
            DECLARE @INDR_details_id INT
            SELECT @INDR_details_id= ISNULL(MAX(INDR_details_id), 0) + 1
            FROM INDR_details
            INSERT INTO INDR_details(INDR_details_id, INDR_id, Table_id,
Number_of_people, Client_id)
```

```
            VALUES(@INDR_details_id, @INDR_id, @Table_id,
@Number_of_people, @Client_id);

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy dodawaniu szczegółów rezerwacji
klienta indywidualnego: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddCompanyReservation
Procedura dodaje rezerwację dla firmy.

```
CREATE PROCEDURE AddCompanyReservation
@Date_since datetime,
@Date_to datetime,
@Company_id int,
@Employee_id int,
@Order_id int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Companies
                  WHERE Company_id=@Company_id
                  )
            BEGIN;
                  THROW 52000, 'Podana firma nie istnieje.', 1
            END
            IF NOT EXISTS(
                  SELECT * FROM Employees
                  WHERE Employee_id=@Employee_id
                  )
            BEGIN;
                  THROW 52000, 'Podany pracownik nie istnieje.', 1
            END

            DECLARE @Company_Reservation_id INT
            SELECT @Company_Reservation_id=
ISNULL(MAX(Company_Reservation_id), 0) + 1
            FROM Company_Reservations
            INSERT INTO Company_Reservations(Company_Reservation_id,
Date_since, Date_to, Company_id, Employee_id, Order_id)
            VALUES(@Company_Reservation_id, @Date_since, @Date_to,
@Company_id, @Employee_id, @Order_id);

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
```

```
                        ='Błąd przy dodawaniu rezerwacji firmy: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura AddCRDetails

Procedura dodaje szczegóły rezerwacji dla klienta firmy.

```
CREATE PROCEDURE AddCRDetails
@CR_id int,
@Table_id int,
@Number_of_people int,
@Client_id int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Company_Reservations
                  WHERE Company_Reservation_id=@CR_id
                  )
            BEGIN;
                  THROW 52000, 'Podana rezerwacja nie istnieje.', 1
            END
            IF NOT EXISTS(
                  SELECT * FROM Tables
                  WHERE Table_id=@Table_id
                  )
            BEGIN;
                  THROW 52000, 'Podany stolik nie istnieje.', 1
            END
            IF @Number_of_people > (SELECT Tables.Number_of_seats from
Tables WHERE Tables.Table_id = @Table_id)
             BEGIN;
                   THROW 52000, 'Przy tym stoliku nie ma tyle miejsc.',
1
             END
            IF @Number_of_people<2
            BEGIN;
                  THROW 52000, 'Rezerwację można zarezerwować na
minimalnie 2 osoby', 1
            END
            DECLARE @CR_details_id INT
            SELECT @CR_details_id= ISNULL(MAX(CR_details_id), 0) + 1
            FROM CR_details
            INSERT INTO CR_details(CR_details_id, CR_id, Table_id,
Number_of_people, Client_id)
            VALUES(@CR_details_id, @CR_id, @Table_id, @Number_of_people,
@Client_id);
```

```
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
				='Błąd przy dodawaniu szczegółów rezerwacji
klienta indywidualnego: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura AddTable
Procedura dodaje stolik.

```
CREATE PROCEDURE AddTable
@Number_of_seats int
AS
BEGIN
SET NOCOUNT ON
BEGIN TRY
IF @Number_of_seats <= 1
BEGIN;
THROW 52000, 'Przy stoliku muszą być co najmniej 2 miejsca', 1
END
DECLARE @Table_id INT
SELECT @Table_id = ISNULL(MAX(Table_id), 0) + 1
FROM Tables
INSERT INTO Tables(Table_id, Number_of_seats)
VALUES (@Table_id, @Number_of_seats);
END TRY
BEGIN CATCH
DECLARE @errorMsg nvarchar(2048)
='Błąd przy dodawaniu stolika: ' + ERROR_MESSAGE();
THROW 52000, @errorMsg, 1
END CATCH
END
GO
```

Procedura RemoveCategory
Procedura usuwa kategorię potraw.

```
CREATE PROCEDURE RemoveCategory
@Category_name varchar(50)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Categories
		WHERE Category_name = @Category_name
```

```
                  )
            BEGIN;
            THROW 52000,'Nie istnieje taka kategoria.',1
            END
            DELETE FROM Categories
            WHERE Category_name = @Category_name
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048) =
                        'Błąd usuwania kategorii: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1;
      END CATCH
END
GO
```

Procedura RemoveDish
Procedura usuwa potrawę.

```
CREATE PROCEDURE RemoveDish
@Dish_name varchar(50)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT *
            FROM Dishes
            WHERE Dish_name = @Dish_name
                  )
            BEGIN;
            THROW 52000,'Nie istnieje taka potrawa.',1
            END
            DELETE FROM Dishes
            WHERE Dish_name = @Dish_name
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048) =
                        'Błąd usuwania potrawy: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1;
      END CATCH
END
GO
```

Procedura RemoveMenuItem
Procedura usuwa pozycję menu.

```
CREATE PROCEDURE RemoveMenuItem
@Position_id int
AS
BEGIN
```

```
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Menu_items
		WHERE Position_id = @Position_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taka pozycja menu.',1
		END
		DELETE FROM Menu_items
		WHERE Position_id = @Position_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania pozycji menu: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveClient
Procedura usuwa klienta.

```
CREATE PROCEDURE RemoveClient
@Client_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Clients
	    WHERE Client_id = @Client_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taki klient.',1
		END
		DELETE FROM Clients
		WHERE Client_id = @Client_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania klienta: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveExternalClient
Procedura usuwa klienta zewnętrznego.

```
CREATE PROCEDURE RemoveExternalClient
@External_client_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM External_clients
		WHERE External_client_id = @External_client_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taki klient zewnętrzny.',1
		END
		DELETE FROM External_clients
		WHERE External_client_id = @External_client_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania klienta zewnętrznego: ' +
ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveIndClient
Procedura usuwa klienta indywidualnego.

```
CREATE PROCEDURE RemoveIndClient
@Individual_client_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Individual_clients
		WHERE Individual_client_id = @Individual_client_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taki klient indywidualny.',1
		END
	   DELETE FROM Individual_clients
		WHERE Individual_client_id = @Individual_client_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania klienta indywidualnego: ' +
ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
```

```
END
GO
```

Procedura RemoveCompany
Procedura usuwa firmę.

```
CREATE PROCEDURE RemoveCompany
@Company_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Companies
		WHERE Company_id = @Company_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taka firma.',1
		END
		DELETE FROM Companies
		WHERE Company_id = @Company_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania firmy: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveDiscount
Procedura usuwa rabat.

```
CREATE PROCEDURE RemoveDiscount
@Discount_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Discounts_lib
		WHERE Discounts_lib_id = @Discount_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taki rabat.',1
		END
		DELETE FROM Discounts_lib
		WHERE Discounts_lib_id = @Discount_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania rabatu: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveRole

Procedura usuwa rolę.

```
CREATE PROCEDURE RemoveRole
@Role_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		FROM Roles
		WHERE Role_id = @Role_id
			)
		BEGIN;
		THROW 52000,'Nie istnieje taka rola.',1
		END
		DELETE FROM Roles
		WHERE Role_id = @Role_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania roli: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveEmployee
Procedura usuwa pracownika.

```
CREATE PROCEDURE RemoveEmployee
@Employee_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		    FROM Employees
		  WHERE Employee_id = @Employee_id
			)
		  BEGIN;
		  THROW 52000,'Nie istnieje taki pracownik.',1
		  END
		  DELETE FROM Employees
		  WHERE Employee_id = @Employee_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048) =
				'Błąd usuwania pracownika: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1;
	END CATCH
END
GO
```

Procedura RemoveIndClientInvoiceDetails
Procedura usuwa szczegóły do faktury dla klienta indywidualnego.

```
CREATE PROCEDURE RemoveIndClientInvoiceDetails
@Individual_client_id int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT *
		  FROM Individual_clients
		  WHERE Individual_client_id = @Individual_client_id
			)
		  BEGIN;
		  THROW 52000,'Nie istnieje taki klient indywidualny.',1
		  END
		  IF NOT EXISTS(
			SELECT *
		  FROM Invoice_details
		  WHERE Client_id = @Individual_client_id
			)
		  BEGIN;
		  THROW 52000,'Nie istnieją szczegóły faktury dla tego klienta
```

```
indywidualnego.',1
            END
            DELETE FROM Invoice_details
            WHERE Client_id = @Individual_client_id
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048) =
                        'Błąd usuwania szczegółów faktury: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1;
      END CATCH
END
GO
```

Procedura RemoveExtClientInvoiceDetails
Procedura usuwa szczegóły do faktury dla klienta zewnętrznego.

```
CREATE PROCEDURE RemoveExtClientInvoiceDetails
@External_client_id int
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT *
            FROM External_clients
            WHERE External_client_id = @External_client_id
                  )
            BEGIN;
            THROW 52000,'Nie istnieje taki klient zewnętrzny.',1
            END
            IF NOT EXISTS(
                  SELECT *
            FROM Invoice_details
            WHERE Client_id = @External_client_id
                  )
            BEGIN;
            THROW 52000,'Nie istnieją szczegóły faktury dla tego klienta
zewnętrznego.',1
            END
            DELETE FROM Invoice_details
            WHERE Client_id = @External_client_id
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048) =
                        'Błąd usuwania szczegółów faktury: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1;
      END CATCH
END
GO
```

Procedura ModifyTable
Procedura zmienia ilość miejsc przy danym stoliku.

```
CREATE PROCEDURE [dbo].ModifyTable
@table_id int, @seats int
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		IF NOT EXISTS (SELECT * FROM Tables WHERE Table_id =
		@table_id)
		BEGIN;
			THROW 52000, 'Nie ma takiego stolika', 1;
		END
	UPDATE Tables SET Tables.Number_of_seats = @seats WHERE table_id =
	@table_id
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)='Błąd przy zmianie
	parametrów stolika: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura ModifyDateItemMenu
Procedura uaktualnia czas obowiązywania pozycji w Menu_items o danym Position_id.

```
CREATE PROCEDURE ModifyDateItemMenu
@Position_id int,
@Date_since datetime,
@Date_to datetime
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM Menu_items
			WHERE Position_id=@Position_id
			)
		BEGIN;
			THROW 52000, 'Nie istnieje taka pozycja menu.', 1
		END
		IF EXISTS(
			SELECT * FROM Menu_items
			WHERE Position_id=@Position_id AND Date_to < GETDATE()
			)
		BEGIN;
			THROW 52000, 'Ta pozycja skończyła już obowiązywać i
nie można zmienić dat jej obowiązywania', 1
```

```
            END
            IF @Date_since < GETDATE() OR @Date_to <GETDATE()
            BEGIN;
                  THROW 52000, 'Data nie może być z przeszłości.', 1
            END
            IF @Date_since IS NOT NULL
            BEGIN
                  UPDATE Menu_items
                        SET Date_since = @Date_since
                        WHERE Menu_items.Position_id=@Position_id
            END
            IF @Date_to IS NOT NULL
            BEGIN
                  UPDATE Menu_items
                        SET Date_to = @Date_to
                        WHERE Menu_items.Position_id=@Position_id
            END
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy zmianie czasu pozycji menu: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ModifyDateDiscount

Procedura edytuje czas obowiązywania rabatu w Discounts_lib o podanym Discounts_lib_id.

```
CREATE PROCEDURE ModifyDateDiscount
@Discount_lib_id int,
@Date_since datetime,
@Date_to datetime
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Discounts_lib
                  WHERE Discounts_lib_id=@Discount_lib_id
                  )
            BEGIN;
                  THROW 52000, 'Nie istnieje taka pozycja menu.', 1
            END
            IF EXISTS(
                  SELECT * FROM Discounts_lib
                  WHERE Discounts_lib_id=@Discount_lib_id AND Date_to <
GETDATE()
                  )
            BEGIN;
                  THROW 52000, 'Ten rabat skończył już obowiązywać i nie
można zmienić dat jego obowiązywania', 1
```

```
            END
            IF @Date_since < GETDATE() OR @Date_to < GETDATE()
            BEGIN;
                  THROW 52000, 'Data nie może być z przeszłości.', 1
            END
            IF @Date_since IS NOT NULL
            BEGIN
                  UPDATE Discounts_lib
                        SET Date_since = @Date_since
                        WHERE
Discounts_lib.Discounts_lib_id=@Discount_lib_id
            END
            IF @Date_to IS NOT NULL
            BEGIN
                  UPDATE Discounts_lib
                        SET Date_to = @Date_to
                        WHERE
Discounts_lib.Discounts_lib_id=@Discount_lib_id
            END
      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                        ='Błąd przy zmianie czasu rabatu: ' +
ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ModifyTable

Procedura zmienia wartość parametru.

```
CREATE PROCEDURE ModifyParameters
@Param_key varchar(30),
@Value decimal(12,5)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
      IF NOT EXISTS (SELECT * FROM Parameters WHERE Param_key =
@Param_key)
      BEGIN;
            THROW 52000, 'Nie ma takiego parametru.', 1;
      END
      UPDATE Parameters SET Parameters.Value = @Value WHERE Param_key =
@Param_key
END TRY
BEGIN CATCH
      DECLARE @errorMsg nvarchar(2048)='Błąd przy zmianie parametru: ' +
ERROR_MESSAGE();
      THROW 52000, @errorMsg, 1
END CATCH
END
```

Procedura ChangeIndReservationStatus
Procedura zmienia status rezerwacji klienta indywidualnego.

```
CREATE PROCEDURE ChangeIndReservationStatus
@Ind_Reservation_id int,
@Status varchar(50)
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM Ind_Reservations
			WHERE Ind_Reservation_id=@Ind_Reservation_id
			)
		BEGIN;
			THROW 52000, 'Nie istnieje taka rezerwacja.', 1
		END


		UPDATE INDR_Statuses
		    SET Status = @Status
			WHERE INDR_id=@Ind_Reservation_id

	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
			 ='Błąd przy zmianie statusu: ' + ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura ChangeCReservationStatus
Procedura zmienia status rezerwacji klienta (firma).

```
CREATE PROCEDURE ChangeCReservationStatus
@C_Reservation_id int, @Status varchar
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
		IF NOT EXISTS(
			SELECT * FROM Company_Reservations
			WHERE
Company_Reservations.Company_Reservation_id=@C_Reservation_id)
		BEGIN;
			THROW 52000, 'Nie istnieje taka rezerwacja.', 1
		END
```

```
            UPDATE CR_Statuses
             SET Status = @Status
             WHERE CR_id =@C_Reservation_id

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                  ='Błąd przy zmianie statusu: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ChangeOrderStatus
Procedura zmienia status zamówienia.

```
CREATE PROCEDURE ChangeOrderStatus
@Order_id int,
@Order_status_id varchar(50)
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
            IF NOT EXISTS(
                  SELECT * FROM Orders
                  WHERE Order_id=@Order_id
                  )
            BEGIN;
                  THROW 52000, 'Nie istnieje taka rezerwacja.', 1
            END
            IF NOT EXISTS(
                  SELECT * FROM Order_statuses
                  WHERE Order_status_id=@Order_status_id
                  )
            BEGIN;
              THROW 52000, 'Nie istnieje taki status zamówienia.', 1
            END

            UPDATE Orders
            SET Order_status_id = @Order_status_id
                  WHERE Order_id=@Order_id

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                  ='Błąd przy zmianie statusu: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ViewIndClientOrders
Procedura wyświetla widok IndClientOrders dla danego klienta indywidualnego.

```
CREATE PROCEDURE ViewIndClientOrders
@Individual_client_id INT
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
	IF NOT EXISTS(
			SELECT * FROM Individual_clients
			WHERE Individual_client_id=@Individual_client_id
			)
	BEGIN;
			THROW 52000, 'Nie istnieje taki klient', 1
	END
		SELECT * FROM IndClientOrders
		WHERE Individual_client_id=@Individual_client_id

	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
			='Błąd przy wyświetlaniu widoku IndClientOrders: ' +
ERROR_MESSAGE();
		THROW 52000, @errorMsg, 1
	END CATCH
END
GO
```

Procedura ViewIndClientDiscounts
Procedura wyświetla widok IndClientDiscounts dla danego klienta indywidualnego.

```
CREATE PROCEDURE ViewIndClientDiscounts
@Individual_client_id INT
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
	IF NOT EXISTS(
			SELECT * FROM Individual_clients
			WHERE Individual_client_id=@Individual_client_id
			)
	BEGIN;
			THROW 52000, 'Nie istnieje taki klient', 1
	END
		SELECT * FROM IndClientDiscounts
		WHERE Individual_client_id=@Individual_client_id

	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar(2048)
			='Błąd przy wyświetlaniu widoku IndClientDiscounts: ' +
ERROR_MESSAGE();
```

```
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ViewCompanyOrders
Procedura wyświetla widok CompanyOrders dla danej firmy.

```
CREATE PROCEDURE ViewCompanyOrders
@Company_id INT
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
      IF NOT EXISTS(
                  SELECT * FROM Companies
                  WHERE Company_id=@Company_id
                  )
      BEGIN;
                  THROW 52000, 'Nie istnieje taka firma', 1
      END
             SELECT * FROM CompanyOrders
             WHERE Company_id=@Company_id

      END TRY
      BEGIN CATCH
            DECLARE @errorMsg nvarchar(2048)
                              ='Błąd przy wyświetlaniu widoku
CompanyOrders: ' + ERROR_MESSAGE();
            THROW 52000, @errorMsg, 1
      END CATCH
END
GO
```

Procedura ChangeMenu
Procedura zmienia co najmniej połowę menu w podanym przedziale czasowym.

```
CREATE PROCEDURE
[dbo].[ChangeMenu]
(@HowMany float, @From datetime, @To datetime)
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    IF @HowMany < 0.5
    BEGIN;
        THROW 52000, 'Wymieniamy co najmniej połowę pozycji w menu', 1;
    END
    IF @From < getdate()
    BEGIN;
        THROW 52000, 'Nie możesz ustalać menu w przeszłości', 1;
    END
```

```
    DECLARE @count int = 0, @menu_items_to_change int,
@menu_position_to_change int,
    @new_menu_id int, @dish_ID int, @unit_price money,
@quantity_per_unit int
    SET @new_menu_id = (select count(*) from Menu_items) + 1
    SET @menu_items_to_change =(select count(*) from Menu_items

      WHERE @From BETWEEN Menu_items.Date_since AND Menu_items.Date_to)
* @HowMany
WHILE @count < @menu_items_to_change

       BEGIN

        SET @menu_position_to_change = (SELECT TOP 1
Menu_items.Position_id From Menu_items

        WHERE Menu_items.Date_to > @From Order by Menu_items.Date_since)

        UPDATE [Menu_items]

        SET Menu_items.Date_to = @From

        WHERE Menu_items.Position_id = @menu_position_to_change



        SET @dish_id = (select top 1 dish_id from Dishes where dish_id
not in (SELECT Menu_items.Dish_id

        FROM Menu_items WHERE Menu_items.Date_to > @From))

        SET @unit_price = isnull((select top 1 menu_items.Unit_price
from menu_items

        where menu_items.Dish_id =@dish_id order by unit_price desc),
10.00)

        SET @quantity_per_unit = isnull((select top 1
menu_items.Quantity_per_unit from menu_items

        where menu_items.Dish_id =@dish_id order by unit_price desc), 1)

        insert into menu_items values (@new_menu_id, @dish_id,
@unit_price, @quantity_per_unit,

        @from, @to)

        SET @count = @count + 1

        SET @new_menu_id = @new_menu_id + 1

        END

END TRY

BEGIN CATCH

DECLARE @errorMsg nvarchar(2048) = 'Błąd przy zmianie menu' +
```

```
ERROR_MESSAGE();

THROW 52000, @errorMsg, 1;

END CATCH

END

GO
```

Funkcje:

Funkcja CheckDiscount_R1
Funkcja zwraca Discounts_lib_id rabatu R1, który można przyznać klientowi,jeśli spełnił wymagania jego otrzymania.

```
CREATE FUNCTION [dbo].[CheckDiscount_R1]
(
    @Individual_client_id int,
    @Order_id int
)
RETURNS INT
AS
BEGIN
DECLARE @Client_ID int = (SELECT Client_ID From Orders  WHERE
Orders.Order_id=@Order_id)
DECLARE @Ordered_time datetime = (SELECT Ordered_time FROM Orders WHERE
Orders.Order_id=@Order_id)
DECLARE @Discount_R1_id int = (SELECT Discounts_lib_id FROM
Discounts_lib
WHERE Type= 'R1' AND (@Ordered_time BETWEEN Date_since AND Date_to))
DECLARE @Number_of_orders int = (SELECT Number_of_orders FROM
Discounts_lib
WHERE @Discount_R1_id=Discounts_lib_id)
DECLARE @Minimal_sum_price money = (SELECT Price FROM Discounts_lib
WHERE @Discount_R1_id=Discounts_lib_id)
DECLARE @Client_orders int = (select count(clients.client_id) from
(select o1.client_id from order_details
inner join  menu_items on order_details.Position_id =
Menu_items.Position_id
inner join orders as o1 on o1.order_id = order_details.Order_id
where Order_details.Order_id = o1.Order_id and client_id = @Client_ID
group by o1.client_id, o1.order_id
having sum(unit_price * quantity) > @Minimal_sum_price) as clients
group by clients.client_id)
IF ISNULL(@Number_of_orders, 0) > ISNULL(@Client_orders, 0)
BEGIN
    RETURN 0
END
    RETURN ISNULL(@Discount_R1_id, 0)
END
GO
```

Funkcja CheckDiscount_R2
Funkcja zwraca Discounts_lib_id rabatu R2, który można przyznać klientowi, jeśli spełnił wymagania jego otrzymania.

```
CREATE FUNCTION CheckDiscount_R2
      (
            @Individual_client_id int,
            @Order_id int
```

```
      )
      RETURNS INT
AS
BEGIN
      DECLARE @Ordered_time datetime = (SELECT Ordered_time FROM Orders
WHERE Orders.Order_id=@Order_id)
      DECLARE @Discount_R2_id int = (SELECT Discounts_lib_id From
Discounts_lib WHERE Type='R2' AND (@Ordered_time BETWEEN Date_since AND
Date_to))
      DECLARE @Price_R2 money = (SELECT Price FROM Discounts_lib WHERE
Discounts_lib_id=@Discount_R2_id)
      DECLARE @Total_Cost_R2 int = (SELECT SUM(Value) FROM
IndClientOrders WHERE Individual_client_id=@Individual_client_id GROUP
BY Individual_client_id)
      RETURN(ISNULL((
            SELECT Discounts_lib_id FROM Discounts_lib WHERE
Discounts_lib_id=@Discount_R2_id AND @Total_Cost_R2>@Price_R2
      ), 0));
END
GO
```

Funkcja CheckDiscounts
Funkcja zwraca informację czy można przydzielić rabat klientowi i jaki to rabat.

```
CREATE FUNCTION [dbo].[CheckDiscounts](@Individual_client_id int,
@Order_id int)
RETURNS decimal(10, 2)
AS
BEGIN
DECLARE @R1_id int = [dbo].[CheckDiscount_R1](@Individual_client_id,
@Order_id)
DECLARE @R2_id int = [dbo].[CheckDiscount_R2](@Individual_client_id,
@Order_id)
DECLARE @Value_R1 decimal(10,2) = ISNULL((SELECT Discounts_lib.Value
FROM Discounts_lib WHERE Discounts_lib_id = @R1_id), 0)
DECLARE @Value_R2 decimal(10,2) = ISNULL((SELECT Discounts_lib.Value
FROM Discounts_lib WHERE Discounts_lib_id = @R2_id), 0)
IF @Value_R1 < @Value_R2 AND @Value_R1 != 0
    BEGIN
       RETURN @Value_R1
    END
RETURN @Value_R2
END

GO
```

Funkcja GetFreeTables
Funkcja zwraca listę wolnych stolików.

```
CREATE FUNCTION Free_Tables(@Order_date datetime)
```

```
RETURNS TABLE
AS
RETURN
((SELECT Tables.Table_id AS 'Table id' FROM Tables INNER JOIN CR_Details
ON CR_Details.Table_id = Tables.Table_id
INNER JOIN Company_Reservations ON Company_Reservation_id =
CR_Details.CR_id
WHERE @Order_date NOT BETWEEN Date_since AND Date_to)
UNION
(SELECT Tables.Table_id AS 'Table id' FROM Tables INNER JOIN
INDR_Details ON INDR_Details.Table_id = Tables.Table_id
INNER JOIN Ind_Reservations ON Ind_Reservation_id = INDR_Details.INDR_id
WHERE @Order_date NOT BETWEEN Date_since AND Date_to)
UNION
(SELECT Tables.Table_id AS 'Table_id' FROM Tables INNER JOIN Orders ON
Orders.Table_id = Tables.Table_id
WHERE Orders.Type_of_order_id = 2 OR (Orders.Type_of_order_id = 1 AND
Order_requestedtime IS NULL AND Order_fulfillment IS NULL AND
DATEDIFF(hour, Orders.Ordered_time, @Order_date) > 2) OR
(Orders.Type_of_order_id = 1 AND Order_requestedtime IS NOT NULL
AND Order_fulfillment IS NULL AND DATEDIFF(hour,
Orders.Order_requestedtime, @Order_date) > 2) OR (Order_fulfillment IS
NOT NULL AND
Order_fulfillment < @Order_date)
))
GO
```

Funkcja PriceWithDiscount
Funkcja zwraca cenę zamówienia (uwzględniającą rabat).

```
CREATE FUNCTION PriceWithDiscount(@Order_id int)
RETURNS MONEY
AS
BEGIN
DECLARE @Client_id int = (SELECT Individual_client_id FROM
Individual_clients
INNER JOIN Orders ON Orders.Client_id = Individual_clients.Client_id
WHERE Order_id = @Order_id)
DECLARE @Discount decimal = [dbo].[CheckDiscount](@Client_id, @Order_id)
DECLARE @NoDiscountPrice int = (SELECT SUM(Order_details.Quantity *
Menu_items.Unit_price) FROM Order_details
INNER JOIN Menu_items on Order_details.Position_id =
Menu_items.Position_id
WHERE Order_details.Order_id = @Order_id
GROUP BY Order_details.Order_id)
RETURN (@NoDiscountPrice - (@NoDiscountPrice * @Discount))
END
GO
```

Funkcja GetInvoiceInd
Funckja wystawia fakturę dla klienta indywidualnego za podane zamówienie.

```
CREATE FUNCTION GetInvoiceInd(@Order_id int)
RETURNS TABLE
AS
RETURN
(SELECT ID.First_name, ID.Last_name, ID.Address, ID.Phone, @Order_id AS
Order_id, [dbo].[PriceWithDiscount](@Order_id) AS 'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN Individual_clients AS IC ON C.Client_id=IC.Client_id
INNER JOIN Invoice_details AS ID ON
ID.Invoice_indclient_id=IC.Invoice_id
WHERE O.Order_id=@Order_id
)
GO
```

Funkcja GetInvoiceExt
Funkcja wystawia fakturę dla klienta zewnętrznego za podane zamówienie.

```
CREATE FUNCTION GetInvoiceExt(@Order_id int)
RETURNS TABLE
AS
RETURN
(SELECT ID.First_name, ID.Last_name, ID.Address, ID.Phone, @Order_id AS
Order_id, [dbo].[PriceWithDiscount](@Order_id) AS 'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN External_clients AS EC ON C.Client_id=EC.Client_id
INNER JOIN Invoice_details AS ID ON
ID.Invoice_indclient_id=EC.Invoice_id
WHERE O.Order_id=@Order_id)
GO
```

Funkcja GetInvoiceC
Funkcja wystawia fakturę dla firmy za podane zamówienie.

```
CREATE FUNCTION GetInvoiceC(@Order_id int)
RETURNS TABLE
AS
RETURN
(SELECT CC.Company_name, CC.NIP, CC.Address, CC.Phone, @Order_id AS
Order_id, [dbo].[PriceWithDiscount](@Order_id) AS 'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN Companies AS CC ON C.Client_id=CC.Client_id
WHERE O.Order_id=@Order_id)
GO
```

Funkcja GetInvoiceIndMonth
Funkcja wystawia fakturę dla klienta indywidualnego za zamówienia z podanego miesiąca.

```
CREATE FUNCTION GetInvoiceIndMonth(@Month int, @Year int,
@Individual_client_id int)
RETURNS TABLE
AS
RETURN
(SELECT ID.First_name, ID.Last_name, ID.Address, ID.Phone, @Month AS
'Month', @Year AS 'Year', SUM([dbo].[PriceWithDiscount](Order_id)) AS
'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN Individual_clients AS IC ON C.Client_id=IC.Client_id
INNER JOIN Invoice_details AS ID ON
ID.Invoice_indclient_id=IC.Invoice_id
WHERE MONTH(O.Order_fulfillment)=@Month AND
YEAR(O.Order_fulfillment)=@Year AND
IC.Individual_client_id=@Individual_client_id
GROUP BY ID.First_name, ID.Last_name, ID.Address, ID.Phone)
GO
```

Funkcja GetInvoiceExtMonth
Funkcja wystawia fakturę dla klienta zewnętrznego za zamówienia z podanego miesiąca.

```
CREATE FUNCTION GetInvoiceExtMonth(@Month int, @Year int,
@External_client_id int)
RETURNS TABLE
AS
RETURN
(SELECT ID.First_name, ID.Last_name, ID.Address, ID.Phone, @Month AS
'Month', @Year AS 'Year', SUM([dbo].[PriceWithDiscount](Order_id)) AS
'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN External_clients AS EC ON C.Client_id=EC.Client_id
INNER JOIN Invoice_details AS ID ON
ID.Invoice_indclient_id=EC.Invoice_id
WHERE MONTH(O.Order_fulfillment)=@Month AND
YEAR(O.Order_fulfillment)=@Year AND
EC.External_client_id=@External_client_id
GROUP BY ID.First_name, ID.Last_name, ID.Address, ID.Phone)
GO
```

Funkcja GetInvoiceCMonth
Funkcja wystawia fakturę dla firmy za zamówienia z podanego miesiąca.

```
CREATE FUNCTION GetInvoiceCMonth(@Month int, @Year int, @Company_id int)
RETURNS TABLE
AS
```

```
RETURN
(SELECT CC.Company_name, CC.NIP, CC.Address, CC.Phone, @Month AS
'Month', @Year AS 'Year', SUM([dbo].[PriceWithDiscount](Order_id)) AS
'Value'
FROM Orders AS O
INNER JOIN Clients AS C ON O.Client_id=C.Client_id
INNER JOIN Companies AS CC ON C.Client_id=CC.Client_id
WHERE MONTH(O.Order_fulfillment)=@Month AND
YEAR(O.Order_fulfillment)=@Year AND CC.Company_id=@Company_id
GROUP BY CC.Company_name, CC.NIP, CC.Address, CC.Phone)
GO
```

Funkcja ReservationsForTheDay
Funkcja wyszukuje rezerwacje na dany dzień.

```
CREATE FUNCTION ReservationsForTheDay(@Date date)
RETURNS TABLE
AS
RETURN ((SELECT Company_Reservations.Company_Reservation_id, 'Company
Reservation' as 'Reservation type' FROM Company_Reservations
WHERE YEAR(Company_Reservations.Date_since) = YEAR(@Date)
AND MONTH(Company_Reservations.Date_since) = MONTH(@Date)
AND DAY(Company_Reservations.Date_since) = DAY(@Date)) UNION ALL
(SELECT Ind_Reservations.Ind_Reservation_id, 'Individual Reservation'
FROM Ind_Reservations
WHERE YEAR(Ind_Reservations.Date_since) = YEAR(@Date)
AND MONTH(Ind_Reservations.Date_since) = MONTH(@Date)
AND DAY(Ind_Reservations.Date_since) = DAY(@Date)))
GO
```

Funkcja IncompleteOrders
Funkcja wyszukuje niezrealizowanych zamówień na dany dzień na wynos lub przy stoliku.

```
CREATE FUNCTION IncompleteOrders(@Date date)
RETURNS TABLE
AS
RETURN
(SELECT Order_id FROM Orders WHERE (Order_status_id = 1 OR
Order_status_id = 2)
AND YEAR(@Date) = YEAR(Orders.Ordered_time) AND MONTH(@Date) =
MONTH(Orders.Ordered_time)
AND DAY(@Date) = DAY(Orders.Ordered_time))
GO
```

Funkcja OrdersOfEmployee
Funkcja wyświetla zamówienia obsłużone przez określonego pracownika w konkretnym przedziale czasowym.

```
CREATE FUNCTION OrdersOfEmployee (@EmployeeID int, @Date_since datetime,
@Date_to datetime)
RETURNS TABLE
AS
RETURN
(SELECT Orders.Order_id FROM Orders WHERE Orders.Employee_id =
@EmployeeID AND
@Date_since > Orders.Order_fulfillment AND @Date_to <
Orders.Order_fulfillment)
GO
```

Funkcja CategoryMenu
Funkcja wyświetla dania w menu z konkretnej kategorii w konkretnym dniu.

```
CREATE FUNCTION CategoryMenu(@Category_name varchar, @Date datetime)
RETURNS TABLE
AS
RETURN
(SELECT Dishes.Dish_name FROM Dishes
INNER JOIN Categories ON Dishes.Category_id = Categories.Category_id
INNER JOIN Menu_items ON Menu_items.Dish_id = Dishes.Dish_id
WHERE @Category_name=Categories.Category_name AND @Date BETWEEN
Menu_items.Date_since AND Menu_items.Date_to)
GO
```

Funkcja getDishesByName
Funkcja zwraca tablicę dań znajdujących się w menu, których nazwa zawiera parametr funkcji.

```
CREATE FUNCTION getDishesByName
(
   @partial_name VARCHAR(30)
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT d.dish_id, d.dish_name, d.category_id, d.seafood
FROM Dishes as d
                INNER JOIN menu_items on menu_items.dish_id=d.dish_id
               WHERE d.dish_name like '%'+@partial_name+'%' and
menu_items.date_to>getdate()
           );
GO
```

Wywołanie:
select * from getDishesByName('zupa');

Funkcja getDishesByCategoryName
Funkcja zwraca tablicę dań znajdujących się w menu, których nazwa kategorii zawiera parametr funkcji.

```
CREATE FUNCTION getDishesByCategoryName
(
   @partial_name VARCHAR(30)
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT d.dish_id, d.dish_name, d.category_id, d.seafood
FROM Dishes as d
                INNER JOIN menu_items on menu_items.dish_id=d.dish_id
                INNER JOIN Categories on
Categories.Category_id=d.Category_id
               WHERE Categories.Category_name like '%'+@partial_name+'%'
and menu_items.date_to>getdate()
           );
GO
```

Wywołanie:
select * from getDishesByCategoryName('zupy');

Funkcja getDishesByCriteria
Funkcja zwraca tablicę dań znajdujących się w menu, spełniających kryteria określone przez parametry funkcji.

```
CREATE FUNCTION getDishesByCriteria
(
   @partial_dish_name AS VARCHAR(30) = null,
   @partial_category_name AS VARCHAR(30)=null,
   @price_from AS DECIMAL = 0,
   @price_to AS DECIMAL = 10000000,
   @contains_seafood as VARCHAR(30) = null
)
   RETURNS TABLE
       AS
       RETURN
           (

               SELECT d.dish_id, d.dish_name, d.category_id, d.seafood,
categories.Category_name, menu_items.Unit_price FROM Dishes as d
               INNER JOIN menu_items on menu_items.dish_id=d.dish_id
               INNER JOIN Categories on
Categories.Category_id=d.Category_id
               WHERE((@partial_dish_name is null OR d.dish_name like
'%'+@partial_dish_name+'%')
                   AND (@partial_category_name is null OR
Categories.Category_name like '%'+@partial_category_name+'%' )
```

```
                    AND (@contains_seafood is null OR
d.Seafood=@contains_seafood)
                    AND @price_from<=Menu_items.Unit_price
                    AND @price_to>=Menu_items.Unit_price
                    AND menu_items.date_to>getdate())
            );
GO
```

Przykłady wywołań:
select * from getDishesByCriteria(default, default, default, default, default);
select * from getDishesByCriteria('zupa', default, default, default, default);
select * from getDishesByCriteria(default, 'Napoj', default, default, default);
select * from getDishesByCriteria(default, default, 11, default, default);
select * from getDishesByCriteria(default, default, default, 11.90, default);
select * from getDishesByCriteria(default, default, default, default, 'yes');
select * from getDishesByCriteria('Spaghett', default, 10, 13, 'no');

Funkcja getReservationForDay
Funkcja zwraca tablicę rezerwacji na dzień będący drugim parametrem funkcji, w zależności od pierwszego parametru funkcji będą to rezerwacji firmowe, lub indywidualne.

```
CREATE FUNCTION getReservationForDay
(
   @type AS VARCHAR(30),
   @date AS DATE
)
   RETURNS @results TABLE (
          id INT NOT NULL,
          date_s DATE,
          date_t DATE,
          employee_id INT,
          phone VARCHAR(50),
          number_of_people INT,
          table_id INT
                          )
BEGIN
   IF (@type like '%comp%')
       BEGIN
           INSERT INTO @results
           SELECT Company_Reservation_id , cr.date_since, cr.date_to,
cr.Employee_id, Companies.phone as phone, CR_Details.Number_of_people,
Tables.Table_id
           FROM Company_reservations as cr
            INNER JOIN CR_Details on
cr.company_reservation_id=CR_Details.CR_Details_id
            INNER JOIN Tables on CR_Details.Table_id=Tables.Table_id
            INNER JOIN Companies on Companies.Company_id=cr.Company_id
           WHERE @date<=cast(cr.Date_to as date) and
@date>=cast(cr.Date_since as date);
       END
   ELSE IF (@type like '%ind%')
       BEGIN
```

```
            INSERT INTO @results
            SELECT indr.Ind_Reservation_id , indr.date_since,
indr.date_to, indr.Employee_id, Individual_clients.phone as phone,
INDR_Details.Number_of_people, Tables.Table_id
            FROM Ind_Reservations as indr
             INNER JOIN INDR_Details on
indr.Ind_reservation_id=INDR_Details.INDR_Details_id
             INNER JOIN Tables on INDR_Details.Table_id=Tables.Table_id
             INNER JOIN Individual_clients on
Individual_clients.Individual_client_id=indr.Individual_client_id
        END
    RETURN;
END;
GO
```

Wywołania:
select * from getReservationForDay('comp', '2021-12-12');
select * from getReservationForDay('ind', '2021-11-09');
select * from getReservationForDay('individual', getdate());
select * from getReservationForDay('company', getdate());

Funkcja getOrdersByTypeAndStatus
Funkcja zwraca tablicę zamówień spełniających podane kryteria (status niezrealizowany, lub zrealizowany, na miejscu, lub na wynos oraz zapłacony, lub niezapłacony).

```
CREATE FUNCTION getOrdersByTypeAndStatus
(
   @status AS VARCHAR(30),
   @type AS VARCHAR(30)
)
RETURNS TABLE
AS
        RETURN
            (
                SELECT o.order_id, Types_of_orders.Type,
Order_statuses.status, Order_statuses.Payment_status FROM Orders as o
                INNER JOIN Types_of_orders on
Types_of_orders.Type_of_order_id=o.Type_of_order_id
                INNER JOIN Order_statuses on
Order_statuses.Order_status_id=o.Order_status_id
                where Types_of_orders.Type=@type and
Order_statuses.Status=@status
            );
GO
```

Wywołanie:
select * from getOrdersByTypeAndStatus('Zrealizowane', 'Na miejscu', 'Zaplacone');

Funkcja isWKfulfilled
Funkcja sprawdza czy klient indywidualny dokonał co najmniej WK zamówień, żeby zarezerwować stolik.

```
CREATE FUNCTION isWKfulfilled
(
   @Ind_client_id INT
)
RETURNS INT
AS
BEGIN
      DECLARE @WK int = (SELECT Value FROM Parameters WHERE
Param_key='WK')
      DECLARE @Number_of_orders int = (SELECT count(*) FROM Orders
      INNER JOIN Clients ON Clients.Client_id=Orders.Client_id
      INNER JOIN Individual_clients ON
Clients.Client_id=Individual_clients.Client_id
      WHERE Individual_client_id=@Ind_client_id)
      IF @Number_of_orders>=@WK
      BEGIN
            RETURN 1
      END
      RETURN 0
END
GO
```

Funkcja isWZfulfilled
Funkcja sprawdza czy klient złożył zamówienie o minimalnej wartości WZ przy rezerwacji stolika.

```
CREATE FUNCTION isWZfulfilled
(
   @Order_id INT
)
RETURNS INT
AS
BEGIN
      DECLARE @WZ int = (SELECT Value FROM Parameters WHERE
Param_key='WZ')
      DECLARE @Value Money = (SELECT
[dbo].[PriceWithDiscount](@Order_id))
      IF @Value>=CAST(@WZ AS money)
      BEGIN
            RETURN 1
      END
      RETURN 0
END
GO
```

Funkcja CheckIfChanged
Funkcja sprawdza czy co najmniej połowa obecnie obowiązującego menu ma pozycje dodane w ciągu ostatnich 2 tygodni.

```
CREATE FUNCTION CheckIfChanged(@Date datetime)
RETURNS BIT
AS
BEGIN
	DECLARE @AllInMenu int = (SELECT COUNT(*) FROM Menu_items
	WHERE @Date BETWEEN Menu_items.Date_since AND Menu_items.Date_to)
	DECLARE @MenuChanged int = (SELECT COUNT(*) FROM Menu_items
	WHERE DATEDIFF(day, Date_since, @Date) < 14)
	DECLARE @Ratio decimal(10,2) = @MenuChanged/@AllInMenu
	IF @Ratio < 1/2
	BEGIN
		RETURN 0
	END
	RETURN 1
END
GO
```

Funkcja getFreeTablesBetweenDates
Funkcja zwraca listę wolnych stolików między dwiema datami.

```
CREATE FUNCTION getFreeTablesBetweenDates(@date1 datetime, @date2
datetime)
RETURNS TABLE
AS
RETURN
   (
       (
           SELECT Tables.Table_id AS 'Table id' FROM Tables INNER JOIN
CR_Details ON CR_Details.Table_id = Tables.Table_id
                                                                INNER JOIN
Company_Reservations ON Company_Reservation_id = CR_Details.CR_id
           WHERE (@date2<Date_since OR @date1>Date_to)
       )
       UNION
       (
           SELECT Tables.Table_id AS 'Table id' FROM Tables INNER JOIN
INDR_Details ON INDR_Details.Table_id = Tables.Table_id
                                                                INNER JOIN
Ind_Reservations ON Ind_Reservation_id = INDR_Details.INDR_id
           WHERE (@date2<Date_since OR @date1>Date_to)
       )
   )
GO
```

Triggery:

Trigger CancelOrderInd
Anuluje zamówienie, gdy rezerwacja klienta indywidualnego została anulowana.

```
CREATE TRIGGER CancelOrderInd
ON INDR_Statuses
AFTER UPDATE
AS
BEGIN
SET NOCOUNT ON;
UPDATE Orders SET Order_status_id=4
WHERE Order_id IN (SELECT Order_id FROM Ind_Reservations
INNER JOIN INDR_Statuses ON
INDR_Statuses.INDR_id=Ind_Reservations.Ind_Reservation_id
WHERE INDR_Statuses.Status='Anulowana')
END
GO
```

Trigger CancelOrderC
Anuluje zamówienie, gdy rezerwacja firmy została anulowana.

```
CREATE TRIGGER CancelOrderC
ON CR_Statuses
AFTER UPDATE
AS
BEGIN
SET NOCOUNT ON;
UPDATE Orders SET Order_status_id=4
WHERE Order_id IN (SELECT Order_id FROM Company_Reservations
INNER JOIN CR_Statuses ON
CR_Statuses.CR_id=Company_Reservations.Company_Reservation_id
WHERE CR_Statuses.Status='Anulowana')
END
GO
```

Trigger SeaFoodCheck
Blokuje możliwość zamówienia potrawy z owocami morza w inne dni tygodnia niż czwartek, piątek i sobotę.

```
CREATE TRIGGER SeaFoodCheck
ON Order_details
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
      IF EXISTS(
      SELECT * FROM inserted AS i
      INNER JOIN Orders AS O ON O.Order_id=i.Order_id
```

```
      INNER JOIN Menu_items AS MI ON MI.Position_id=i.Position_id
      INNER JOIN Dishes AS D On D.Dish_id=MI.Dish_id
      WHERE DATENAME(WEEKDAY, O.Order_requestedtime) NOT LIKE 'Thursday'
      AND DATENAME(WEEKDAY, O.Order_requestedtime) NOT LIKE 'Friday'
      AND DATENAME(WEEKDAY, O.Order_requestedtime) NOT LIKE 'Saturday'
      AND D.Seafood = 'yes')
      BEGIN;
            THROW 50001, 'Nie można zamówić dania z owocami morza na inne
dni niż czwartek, piątek lub sobota', 1
      END
END
GO
```

Trigger SeaFoodCheckDaysBefore

Blokuje możliwość zamówienia potrawy z owocami morza, jeśli złożone później niż w poniedziałek poprzedzający zamówienie.

```
CREATE TRIGGER SeaFoodCheckDaysBefore
ON Order_details
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
      IF EXISTS(
      SELECT * FROM inserted AS i
      INNER JOIN Orders AS O ON O.Order_id=i.Order_id
      INNER JOIN Menu_items AS MI ON MI.Position_id=i.Position_id
      INNER JOIN Dishes AS D ON D.Dish_id=MI.Dish_id
      WHERE (DATENAME(WEEKDAY, O.Order_requestedtime) LIKE 'Thursday'
      AND DATEDIFF(day, O.Ordered_time, O.Order_requestedtime)<=2
      AND D.Seafood='yes')
      OR (DATENAME(WEEKDAY, O.Order_requestedtime) LIKE 'Friday'
      AND DATEDIFF(day, O.Ordered_time, O.Order_requestedtime)<=3
      AND D.Seafood='yes')
      OR (DATENAME(WEEKDAY, O.Order_requestedtime) LIKE 'Saturday'
      AND DATEDIFF(day, O.Ordered_time, O.Order_requestedtime)<=4
      AND D.Seafood='yes')
      )
      BEGIN;
            THROW 50001, 'Nie można zamówić dania z owocami morza później
niż w poniedziałek poprzedzający zamówienie', 1
      END

END
GO
```

Trigger DeleteCancelledOrderDetails

Usuwa szczegóły zamówienia, które zostało anulowane.

```
CREATE TRIGGER DeleteCancelledOrderDetails
```

```
ON Order_details
FOR DELETE
AS
BEGIN
   SET NOCOUNT ON;
   DELETE FROM Order_details
   WHERE Order_id IN (SELECT Orders.Order_id FROM Orders
                      WHERE Orders.Order_status_id = 4)
END
GO
```

Trigger DeleteINDReservationDetails
Usuwa szczegóły zamówienia (indywidualnego), które zostało anulowane.

```
CREATE TRIGGER DeleteINDReservationDetails
ON INDR_Details
FOR DELETE
AS
BEGIN
   SET NOCOUNT ON;
   DELETE FROM INDR_Details
   WHERE INDR_id IN (SELECT INDR_Statuses.INDR_id FROM INDR_Statuses
                     WHERE Status = 'Anulowane')
END
GO
```

Trigger DeleteCReservationDetails
usuwa szczegóły zamówienia (firmowego), które zostało anulowane.

```
CREATE TRIGGER DeleteCReservationDetails
ON CR_Details
FOR DELETE
AS
BEGIN
   SET NOCOUNT ON;
   DELETE FROM CR_Details
   WHERE CR_id IN (SELECT CR_Statuses.CR_id FROM CR_Statuses
                   WHERE Status = 'Anulowane')
END
GO
```

Trigger UpdateOrderStatus
Zmienia status zamówienia gdy fulfillment date nie jest wartością null.

```
CREATE TRIGGER UpdateCReservationStatus
ON Order_statuses
AFTER UPDATE
AS
```

```
BEGIN
   SET NOCOUNT ON;
   UPDATE Orders SET Order_status_id=3
   WHERE Order_id in (SELECT Order_id FROM Orders
                                                INNER JOIN Order_statuses
ON Order_statuses.Order_status_id = Orders.Order_status_id
                      WHERE Orders.Order_fulfillment IS NOT NULL)
END
GO
```

Trigger table_type_of_order

Wyrzuca wyjątek i cofa transakcję jeśli:

1. table_id nie jest nullem i typ zamówienia na wynos
2. table_id jest nullem i typ zamówienia na miejscu

```
CREATE TRIGGER table_type_of_order
   ON orders
   INSTEAD OF INSERT
   AS
BEGIN
   IF EXISTS(SELECT * FROM INSERTED
             WHERE (Type_of_order_id=2 AND table_id is not null)
                OR (Type_of_order_id=1 AND table_id is null)
       )
       BEGIN
           RAISERROR ('If table_id is not null type_of_order_id must be
1 or If table_id is null type_of_order_id must be 2',10,1)
           ROLLBACK TRANSACTION
       END
   ELSE
       BEGIN
           INSERT INTO orders SELECT * FROM INSERTED
       END
END
GO
```

Przykłady wywołania:

```
   INSERT INTO orders values(2000,2000,1,null,
'2022-01-19','2022-01-20',null,1000,1)
   INSERT INTO orders values(2000,2000,2,1,
'2022-01-19','2022-01-20',null,1000,1)
```

Trigger discount_length_does_not_match

Wyrzuca wyjątek i cofa transakcję jeśli:

1. długość rabatu w C_discounts nie jest równa długości w discounts_lib_id
2. rabat jest nieaktualny (to_date<getdate())

```
CREATE TRIGGER discount_length_does_not_match
   ON C_Discounts
```

```
   INSTEAD OF INSERT
   AS
BEGIN
   IF EXISTS(SELECT * FROM INSERTED
                                INNER JOIN Discounts_lib ON
Discounts_lib.Discounts_lib_id=Discount_lib_id
            WHERE
discounts_lib.length_of_discount<>ABS(DATEDIFF(day,end_date,start_date))
                OR discounts_lib.date_to<getdate()
       )
       BEGIN
           RAISERROR('length of discount does not match parameters in
discounts_lib or discount expired',10,1)
           ROLLBACK TRANSACTION
       END
   ELSE
       BEGIN
           INSERT INTO C_Discounts SELECT * FROM INSERTED
       END
END
GO
```

Przykład wywołania:

```
   insert into C_Discounts
values(9999,4,9999,'2021-12-30','2022-03-31')
```

Trigger individualCheckTableAvailability

Wyrzuca wyjątek i cofa transakcję w przypadku, gdy zarezerwowany stolik (indywidualna rezerwacja) jest już zajęty na daną datę.

```
CREATE TRIGGER individualCheckTableAvailability
   ON Ind_Reservations
   INSTEAD OF INSERT
   AS
BEGIN
   Declare @start_date as datetime
   Declare @End_date as datetime
   Declare @tid int
   SELECT @start_date=date_since, @End_date=date_to  from inserted
   SELECT @tid=INDR_Details.table_id FROM INSERTED
                                              INNER JOIN INDR_Details ON
INDR_Details.INDR_id=Ind_reservation_id
   IF @tid NOT IN(
       SELECT * FROM getFreeTablesBetweenDates(@start_date, @End_date)
   )
       BEGIN
           RAISERROR('table not available',10,1)
           ROLLBACK TRANSACTION
       END
   ELSE
       BEGIN
           INSERT INTO Ind_reservations SELECT * FROM INSERTED
       END
```

```
END
GO
```

Trigger companyCheckTableAvailability

Wyrzuca wyjątek i cofa transakcję w przypadku, gdy zarezerwowany stolik (firmowa rezerwacja) jest już zajęty na daną datę.

```
CREATE TRIGGER companyCheckTableAvailability
   ON Company_Reservations
   INSTEAD OF INSERT
   AS
BEGIN
   Declare @start_date as datetime
   Declare @End_date as datetime
   Declare @tid int
   SELECT @start_date=date_since, @End_date=date_to  from inserted
   SELECT @tid=Company_Reservation_id FROM INSERTED
                                                INNER JOIN CR_Details ON
CR_Details.CR_id=Company_Reservation_id
   IF @tid NOT IN(
       SELECT * FROM getFreeTablesBetweenDates(@start_date, @End_date)
   )
       BEGIN
           RAISERROR('table not available',10,1)
           ROLLBACK TRANSACTION
       END
   ELSE
       BEGIN
           INSERT INTO Company_Reservations SELECT * FROM INSERTED
       END
END
GO
```

Indeksy:

Indeks Companies_client_id
Ustawienie indeksu na Client_id w tabeli Companies.

```
CREATE INDEX Companies_client_id ON Companies (Client_id)
```

Indeks Companies_company_name
Ustawienie indeksu na Company_name w tabeli Comapnies.

```
CREATE INDEX Companies_company_name ON Companies (Company_name)
```

Indeks Companies_NIP
Ustawienie indeksu na NIP w tabeli Companies.

```
CREATE INDEX Companies_NIP ON Companies (NIP)
```

Indeks Companies_phone
Ustawienie indeksu na Phone w tabeli Companies.

```
CREATE INDEX Companies_phone ON Companies (Phone)
```

Indeks Ind_clients_client_id
Ustawienie indeksu na Client_id w tabeli Individual_clients.

```
CREATE INDEX Ind_clients_client_id ON Individual_clients (Client_id)
```

Indeks Ind_clients_phone
Ustawienie indeksu na Phone w tabeli Individual_clients.

```
CREATE INDEX Ind_clients_phone ON Individual_clients (Phone)
```

Indeks Ext_clients_client_id
Ustawienie indeksu na Client_id w tabeli External_clients.

```
CREATE INDEX Ext_clients_client_id ON External_clients (Client_id)
```

Indeks Categories_category_name
Ustawienie indeksu na Category_name w tabeli External_clients.

```
CREATE INDEX Categories_category_name ON Categories (Category_name)
```

Indeks Dishes_dish_name
Ustawienie indeksu na Dish_name w tabeli Dishes.

```
CREATE INDEX Dishes_dish_name ON Dishes (Dish_name)
```

Indeks Dishes_category_id
Ustawienie indeksu na Category_id w tabeli Dishes.

```
CREATE INDEX Dishes_category_id ON Dishes (Category_id)
```

Indeks Menu_items_dish_id
Ustawienie indeksu na Dish_id w tabeli Menu_items.

```
CREATE INDEX Menu_items_dish_id ON Menu_items (Dish_id)
```

Indeks Menu_items_date_since
Ustawienie indeksu na Date_since w tabeli Menu_items.

```
CREATE INDEX Menu_items_date_since ON Menu_items (Date_since)
```

Indeks Menu_items_date_to
Ustawienie indeksu na Date_to w tabeli Menu_items.

```
CREATE INDEX Menu_items_date_to ON Menu_items (Date_to)
```

Indeks Employees_role_id
Ustawienie indeksu na Role_id w tabeli Employees.

```
CREATE INDEX Employees_role_id ON Employees (Role_id)
```

Indeks Employees_name
Ustawienie indeksu na Firstname oraz Lastname w tabeli Employees.

```
CREATE INDEX Employees_name ON Employees (Firstname, Lastname)
```

Indeks Ind_Reservations_Individual_client_id
Ustawienie indeksu na Individual_client_id w tabeli Ind_Reservations

```
CREATE INDEX Ind_Reservations_Individual_client_id ON Ind_Reservations
(Individual_client_id)
```

Indeks Ind_Reservations_order_id
Ustawienie indeksu na order_id w tabeli Ind_Reservations

```
CREATE INDEX Ind_reservations_order_id ON Ind_Reservations (order_id)
```

Indeks Ind_Reservations_Individual_employee_id
Ustawienie indeksu na employee_id w tabeli Ind_Reservations

```
CREATE INDEX Ind_reservations_employee_id ON Ind_Reservations
(employee_id)
```

Indeks INDR_Statuses_INDR_id
Ustawienie indeksu na INDR_id w tabeli INDR_Statuses

```
CREATE INDEX INDR_Statuses_INDR_id ON INDR_Statuses (INDR_id)
```

Indeks Company_reservations_company_id
Ustawienie indeksu na company_id w tabeli Company_reservations

```
CREATE INDEX Company_reservations_company_id ON company_reservations
(company_id)
```

Indeks Company_reservations_employee_id
Ustawienie indeksu na employee_id w tabeli Company_reservations

```
CREATE INDEX Company_reservations_employee_id ON company_reservations
(employee_id)
```

Indeks Company_reservations_order_id
Ustawienie indeksu na order_id w tabeli Company_reservations

```
CREATE INDEX Company_reservations_order_id ON Company_reservations
(order_id)
```

Indeks CR_Statuses_CR_id
Ustawienie indeksu na CR_id w tabeli CR_Statuses

```
CREATE INDEX CR_Statuses_CR_id ON CR_Statuses (CR_id)
```

Indeks CR_Details_CR_id
Ustawienie indeksu na CR_id w tabeli CR_Details

```
CREATE INDEX CR_Details_CR_id ON CR_Details (CR_id)
```

Indeks CR_Details_Client_id
Ustawienie indeksu na client_id w tabeli CR_Details

```
CREATE INDEX CR_Details_Client_id ON CR_Details (Client_id)
```

Indeks CR_Details_Table_id
Ustawienie indeksu na table_id w tabeli CR_Details

```
CREATE INDEX CR_Details_Table_id ON CR_Details (Table_id)
```

Indeks INDR_Details_INDR_id
Ustawienie indeksu na INDR_id w tabeli INDR_Details

```
CREATE INDEX INDR_Details_INDR_id ON INDR_Details (INDR_id)
```

Indeks INDR_Details_Table_id
Ustawienie indeksu na table_id w tabeli INDR_Details

```
CREATE INDEX INDR_Details_Table_id ON INDR_Details (Table_id)
```

Indeks INDR_Details_client_id
Ustawienie indeksu na client_id w tabeli INDR_Details

```
CREATE INDEX INDR_Details_Client_id ON INDR_Details (Client_id)
```

Indeks Orders_client_id
Ustawienie indeksu na client_id w tabeli Orders

```
CREATE INDEX Orders_client_id ON orders (client_id)
```

Indeks Orders_type_of_order_id
Ustawienie indeksu na type_of_order_id w tabeli Orders

```
CREATE INDEX Orders_Type_of_order_id ON Orders (type_of_order_id)
```

Indeks Orders_table_id
Ustawienie indeksu na table_id w tabeli Orders

```
CREATE INDEX Orders_table_id ON Orders (table_id)
```

Indeks Orders_employee_id
Ustawienie indeksu na employee_id w tabeli Orders

```
CREATE INDEX Orders_employee_id ON Orders (Employee_id)
```

Indeks Orders_status_id
Ustawienie indeksu na status_id w tabeli Orders

```
CREATE INDEX Order_Orders_status_id ON Orders (Order_status_id)
```

Indeks Order_details_order_id
Ustawienieindeksu na order_id w tabeli Order_details

```
CREATE INDEX Order_details_order_id ON Order_details (order_id)
```

Indeks Order_details_position_id
Ustawienieindeksu na position_id w tabeli Order_details

```
CREATE INDEX Order_details_position_id ON Order_details (position_id)
```

Uprawnienia:

Rola business_admin i jej uprawnienia

```
CREATE ROLE business_admin
GRANT SELECT, INSERT, UPDATE, DELETE ON Employees to business_admin
GRANT SELECT ON Roles to business_admin
GRANT EXECUTE ON AddRole to business_admin
GRANT EXECUTE ON AddEmployee to business_admin
GRANT EXECUTE ON RemoveRole to business_admin
GRANT EXECUTE ON RemoveEmployee to business_admin
```

Rola menu_manager i jej uprawnienia

```
CREATE ROLE menu_manager
GRANT SELECT, INSERT, UPDATE ON Categories to menu_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Dishes to menu_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Menu_Items to Menu_manager
GRANT SELECT ON Menu to Menu_manager
GRANT EXECUTE ON AddCategory to Menu_manager
GRANT EXECUTE ON AddDish to Menu_manager
Grant EXECUTE ON AddMenuItem to Menu_manager
GRANT EXECUTE ON RemoveCategory to menu_manager
GRANT EXECUTE ON RemoveDish to menu_manager
GRANT EXECUTE ON RemoveMenuItem to menu_manager
GRANT EXECUTE ON ModifyDateItemMenu to menu_manager
GRANT EXECUTE ON ChangeMenu to menu_manager
```

Rola reservation_manager i jej uprawnienia

```
CREATE ROLE reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Ind_Reservations to
reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON INDR_Details to
reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON INDR_Statuses to
reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Company_Reservations to
reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON CR_Details to
reservation_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON CR_Statuses to
reservation_manager
GRANT SELECT ON Tables to reservation_manager
GRANT SELECT, INSERT, UPDATE ON Individual_clients to
```

```
reservation_manager
GRANT SELECT, INSERT, UPDATE ON Companies to reservation_manager
GRANT SELECT, INSERT ON Clients to reservation_manager
GRANT SELECT, INSERT, UPDATE ON External_clients to reservation_manager
GRANT SELECT ON Menu to reservation_manager
GRANT SELECT ON PendingReservations to reservation_manager
GRANT SELECT ON Reservations_Today to reservation_manager
GRANT EXECUTE ON AddClient to reservation_manager
GRANT EXECUTE ON AddExternalClient to reservation_manager
GRANT EXECUTE ON AddIndClient to reservation_manager
GRANT EXECUTE ON AddCompany to reservation_manager
GRANT EXECUTE ON AddIndReservation to reservation_manager
GRANT EXECUTE ON AddCompanyReservation to reservation_manager
GRANT EXECUTE ON AddCRDetails to reservation_manager
GRANT EXECUTE ON AddTable to reservation_manager
GRANT EXECUTE ON RemoveClient to reservation_manager
GRANT EXECUTE ON RemoveExternalClient to reservation_manager
GRANT EXECUTE ON RemoveIndClient to reservation_manager
GRANT EXECUTE ON RemoveCompany to reservation_manager
GRANT EXECUTE ON ModifyTable to reservation_manager
GRANT EXECUTE ON ChangeIndReservationStatus to reservation_manager
GRANT EXECUTE ON ChangeCReservationStatus to reservation_manager
```

Rola invoice_manager i jej uprawnienia

```
CREATE ROLE invoice_manager
GRANT SELECT ON IndClientDiscounts to invoice_manager
GRANT SELECT ON TablesAllMonthly to invoice_manager
GRANT SELECT ON TablesAllWeekly to invoice_manager
GRANT SELECT ON TableCR_Monthly to invoice_manager
GRANT SELECT ON TableCR_Weekly to invoice_manager
GRANT SELECT ON TableINDR_Monthly to invoice_manager
GRANT SELECT ON TableINDR_Weekly to invoice_manager
GRANT SELECT ON last_month_discounts to invoice_manager
GRANT SELECT ON last_month_client_sales_discount to invoice_manager
GRANT SELECT ON last_week_discounts to invoice_manager
GRANT SELECT ON last_week_client_sales_discount to invoice_manager
GRANT SELECT ON last_month_menu to invoice_manager
GRANT SELECT ON last_week_menu to invoice_manager
GRANT SELECT ON client_orders_monthly to invoice_manager
GRANT SELECT ON client_orders_weekly to invoice_manager
GRANT SELECT ON company_orders_monthly to invoice_manager
GRANT SELECT ON company_orders_weekly to invoice_manager
GRANT SELECT ON last_month_all_client_orders to invoice_manager
GRANT SELECT ON last_week_all_client_orders to invoice_manager
GRANT SELECT ON last_month_all_company_orders to invoice_manager
GRANT SELECT ON last_week_all_company_orders to invoice_manager
GRANT EXECUTE ON AddIndClientInvoiceDetails to invoice_manager
```

```
GRANT EXECUTE ON AddExtClientInvoiceDetails to invoice_manager
GRANT EXECUTE ON RemoveIndClientInvoiceDetails to invoice_manager
GRANT EXECUTE ON RemoveExtClientInvoiceDetails to invoice_manager
```

Rola order_manager i jej uprawnienia

```
CREATE ROLE order_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Orders to order_manager
GRANT SELECT ON Types_of_orders to order_manager
GRANT SELECT ON Order_statuses to order_manager
GRANT SELECT, INSERT, UPDATE, DELETE ON Order_Details to order_manager
GRANT SELECT ON Menu_Items to order_manager
GRANT SELECT ON Dishes to order_manager
GRANT SELECT ON Categories to order_manager
GRANT SELECT ON Menu to order_manager
GRANT SELECT ON UnrealizedOrders to order_manager
GRANT SELECT ON IndClientOrders to order_manager
GRANT SELECT ON CompanyOrders to order_manager
GRANT EXECUTE ON AddOrder to order_manager
GRANT EXECUTE ON AddOrderDetails to order_manager
GRANT EXECUTE ON ChangeOrderStatus to order_manager
GRANT EXECUTE ON ViewIndCLientOrders to order_manager
GRANT EXECUTE ON ViewCompanyOrders to order_manager
```