Projekt i implementacja systemu bazodanowego

Podstawy baz danych

Mateusz Praski
Dzimitry Mikialevich

Grupa PN 12:50

SPIS TREŚCI

[SCHEMAT BAZY]

TABELE

Nazwa tabeli: Table_Restrictions

Opis tablicy: Tablica odpowiada za obostrzenia ”Covidowe” (albo inne) i zawiera informację o obostrzeniu dotyczącym poszczególnych stolików (Na przykład obostrzenia odnośnie odległości między stolikami są rozstrzygane w nast. sposób: Dodajemy w tabeli Table_Restrictions rekord o obostrzeniu dla nieodpowiednio bliskich stolików, wprowadzając datę rozpoczęcia i końca obostrzenia)

Opis pól:

- *Restriction_ID (int):* Klucz główny
- *Table_ID (int):* Klucz obcy do Tabeli Stoliki - Indeks
- *Seats (int):* Liczba możliwych do zajęcia miejsc (w przypadku całkowitego zakazu wprowadzamy 0)
- *Start_Date (datetime):* Data rozpoczęcia obostrzenia
- *End_Date (datetime):* Data końca obostrzenia (w przypadku braku danych - wartość null)

Warunki Integralności:

- Koniec obostrzenia musi być większy od początku jeśli istnieje
- Liczba miejsc >=0

Kod, generujący tabelę:

```
create table Tables_Restrictions
(
   Table_ID       int      not null
       constraint FK_Tables_Restrictions_Tables
           references Tables,
   Seats          int      not null
       constraint CK_Restriction_Seats
           check ([Seats] >= 0),
   Start_Date     datetime not null,
   End_Date       datetime,
   Restriction_ID int      not null
       constraint PK_Tables_Restrictions
           primary key,
   constraint CK_Restriction_End_Date
       check ([End_Date] IS NULL OR [End_Date] > [Start_Date])
)
```

Nazwa tabeli: Tables

Opis tablicy: Tablica Odpowiada za stolik w lokalu, przechowuje informację o maksymalnej liczbie miejsc dostępnych dla poszczególnego stolika. Aby sprawdzić aktualny stan stolika, patrzymy na obostrzenia w relacji z danym stolikiem w danym przedziale czasowym i na podstawie tej informacji, określamy liczbę dostępnych miejsc przy stoliku. Brak obostrzenia odpowiadającego stolikowi w danym czasie oznacza, że cały stolik jest dostępny.

Opis pól:

- *Table_ID (int):* Klucz główny
- *Local_ID (int):* Klucz obcy z tabeli Locals
- *Seats (int):* Maksymalna liczba miejsc dla danego stolika

Warunki Integralności:

- Liczba miejsc >=0

Kod, generujący tabelę:

```
create table Tables
(
   Table_ID int not null
       constraint PK_Tables
           primary key,
   Local_ID int not null
       constraint FK_Tables_Local
           references Locals,
   Seats    int not null
       constraint CK_Tables_Seats
           check ([Seats] > 0)
)
```

Nazwa tabeli: Company_Reservation_Tables

Opis tablicy: Tabela łącząca tabele Company_Reservations i Tables

Opis pól:

- *Table_ID (int):* Klucz główny, klucz obcy tabeli Stoliki
- *Company_Reservation_ID (int):* Klucz główny, klucz obcy tabeli Rezerwacja_Firmowa

Kod, generujący tabelę:

```
create table Company_Reservation_Tables
(
   Table_ID               int not null
       constraint FK_Company_Reservation_Tables
           references Tables,
   Company_Reservation_ID int not null
       constraint FK_Companty_Reservation_Tables
           references Company_Reservation,
   constraint PK_Company_Reservation_Table
       primary key (Table_ID, Company_Reservation_ID)
)
```

Nazwa tabeli: Company_Reservation_Workers

Opis tablicy: Tabela zawierająca przypisanych do rezerwacji firmowej pracowników

Opis pól:

- *Company_Reservation_ID (int):* Klucz Główny, Klucz obcy do rezerwacji firmowej, reprezentujący rezerwację firmową
- *Firstname (char (50)):* Klucz Główny, Imię pracownika
- *Lastname (char (50)):* Klucz Główny, Nazwisko pracownika

Kod, generujący tabelę:

```
create table Company_Reservation_Workers
(
   Company_Reservation_ID int      not null
       constraint FK_Company_Reservation_Workers
           references Company_Reservations,
   Firstname              char(50) not null,
   Lastname               char(50) not null,
   constraint PK_Company_Reservation_Workers
       primary key (Company_Reservation_ID, Firstname, Lastname)
)
```

Nazwa tabeli: Company_Reservations

Opis tablicy: Tabela zawierająca informacje na temat rezerwacji firmowej.

Opis pól:

- *Company_Reservation_ID (int):* Klucz główny
- *Company_ID (int):* Klucz Obcy do Firmy, ID Firmy która robi tą rezerwację - Indeks
- *Start_Date (datetime):* Data rozpoczęcia rezerwacji
- *End_Date (datetime):* Data końca rezerwacji
- *Confirmation_Date (datetime):* Data potwierdzenia rezerwacji (null w przypadku niepotwierdzonych rezerwacji)

Warunki integralności:

- Data potwierdzenia jest nullem albo (data potwierdzenia >=data rozpoczęcia)
- Data rozpoczęcia jest mniejsza od daty zakończenia

Kod, generujący tabelę:

```
create table Company_Reservations
(
   Company_Reservation_ID int      not null
       constraint PK_Company_Reservation
           primary key,
   Company_ID             int      not null
       constraint FK_Company_Reservation
           references Companies,
   Start_Date             datetime not null,
   End_Date               datetime not null,
   Confirmation_Date      datetime,
   constraint CK_Company_Reservation_Confirmation
       check ([Confirmation_Date] IS NULL OR [Confirmation_Date] <
[Start_Date]),
   constraint Ck_Company_Reservation_Begin_End
       check ([End_Date] > [Start_Date])
)
```

Nazwa tabeli: Individual_Reservations

Opis tablicy: Tabela reprezentująca rezerwację zrobioną przez klienta indywidualnego.

Opis pól:

- *Individual_Reservation_ID (int):* Klucz główny
- *Start_Date (datetime):* Data rozpoczęcia rezerwacji
- *End_Date (datetime):* Data końca rezerwacji
- *Confrimation_Date (datetime):* Data potwierdzenia rezerwacji (null w przypadku niepotwierdzonych rezerwacji)
- *Order_ID (int):* Klucz obcy do tabeli Zamówienia, reprezentujący zamówienie, powiązane z daną rezerwacją - Indeks
- Table_ID (int): Klucz obcy, reprezentujący zarezerwowany stolik

Warunki integralności:

- Data rozpoczęcia < Data zakończenia
- Data potwierdzenia jest nullem albo mniejsza lub równa dacie rozpoczęcia

Kod, generujący tabelę:

```
create table Individual_Reservations
(
   Individual_Reservation_ID int      not null
       constraint PK_Individual_Reservation
           primary key,
   Start_Date                datetime not null,
   End_Date                  datetime not null,
   Confirmation_Date         datetime,
   Order_ID                  int      not null
       constraint FK_Individual_Reservation_Order
           references Orders,
   Table_ID                  int      not null
       constraint FK_Individual_Reservation_Tables
           references Tables,
   constraint CK_Individual_Reservation_Confirmation
       check ([Confirmation_Date] IS NULL OR [Confirmation_Date] <=
[Start_Date]),
   constraint CK_Individual_Reservation_Start_End
       check ([End_Date] > [Start_Date])
)
```

Nazwa tabeli: Orders

Opis tablicy: Tabela zawierające informacje na temat zamówień.

Opis pól:

- *Order_ID (int):* Klucz główny
- *Client_ID (int):* Klucz obcy do tabeli Client, wskazujący klienta robiącego zamówienie - Indeks
- *Local_ID (int):* Klucz obcy do Tabeli Locals
- *Order_Date (datetime):* Data złożenia zamówienia
- *Pref_Date* (datetime) : Preferowana data doręczenia (Null oznacza, że klient nie sprecyzował preferowanej daty dostawy)
- *Delivery_Date (datetime):* Data doręczenia zamówienia (Null oznacza, że zamówienie nie zostało doręczone)
- *Paid (bit):* Czy zamówienie jest opłacone
- *Payment_Method (int):* Klucz obcy tabeli Payment_Methods
- Discount_Percent (float): Procent kwoty odjęty od końcowej wartości zamówienia

Warunki integralności:

- Data preferowana jest nullem albo data preferowana > data złożenia
- Data doręczenia jest nullem albo data złożenia <= dacie doręczenia
- Procentowy rabat znajduje się w przedziale [0,1]

Kod, generujący tabelę:

```
create table Orders
(
   Order_ID         int identity
       constraint PK_Zamowienia
           primary key,
   Client_ID        int             not null
       constraint FK_Orders_Clients
           references Clients,
   Local_ID         int             not null
       constraint FK_Orders_Local
           references Locals,
   Order_Date       datetime        not null,
   Pref_Date        datetime,
   Delivery_Date    datetime,
   Paid             bit             not null,
   Payment_Method   int             not null
       constraint FK_Orders_Payment_Methods
           references Payment_Methods,
   Discount_Percent float default 0 not null
       constraint CK_Discount_Percent
           check ([Discount_Percent] >= 0 AND [Discount_Percent] <= 1),
   constraint CK_Delivery_Date
       check ([Delivery_Date] IS NULL OR [Delivery_Date] >= [Order_Date]),
   constraint CK_Orders_Order_Pref
       check ([Pref_Date] > [Order_Date])
)
```

Nazwa tabeli: Locals

Opis tablicy: Tablica zawierające informacje na temat lokalu oraz jego lokalizacji

Opis pól:

- *Local_ID (int):* Klucz główny
- *City_ID (int):* Klucz obcy do tabeli Cities - Indeks
- *Name (char (50)):* Nazwa lokalu
- *Street (char (50)):* Ulica
- *House_no (char (50)):* Numer domu
- *Apartment_no (char (50)):* Numer lokalu w budynku, null jeśli nie dotyczy
- *Postal_Code (char (20)):* Kod pocztowy

Kod, generujący tabelę:

```
create table Locals
(
   Local_ID     int      not null
       constraint PK_Locals
           primary key,
   City_ID      int      not null
       constraint FK_Locals_Cities
           references Cities,
   Name         char(50) not null,
   Street       char(50) not null,
   House_no     char(50) not null,
   Apartment_no char(50),
   Postal_Code  char(20) not null
)
```

Nazwa tabeli: Payment_Methods

Opis tablicy: Słownik form płatności za zamówienie

Opis pól:

- *Payment_ID (int):* Klucz główny
- *Name (char (50)):* Nazwa płatności

Warunki integralności:

- Nazwa jest unikatowa

Kod, generujący tabelę:

```
create table Payment_Methods
(
   Payment_ID int      not null
       constraint PK_Payment_Method
           primary key,
   Name       char(50) not null
       constraint UNQ_Payment_Name
           unique
)
```

Nazwa tabeli: Order_Details

Opis tablicy: Tablica odpowiadająca za informację o poszczególnych produktach, będących w zamówieniu

Opis pól:

- *Order_ID (int):* Klucz główny, Klucz obcy tabeli Orders
- *Product_ID (int):* Klucz główny, Klucz obcy Products
- *Quantity (int):* Ilość zamówionego produktu
- *Unit_Price (money):* Cena za sztukę danego produktu

Warunki integralności:

- Ilość > 0
- Cena za sztukę > 0

Kod, generujący tabelę:

```
create table Order_Details
(
   Order_ID   int   not null
       constraint FK_Order_Details
           references Orders,
   Product_ID int   not null
       constraint FK_Order_Details_Product
           references Products,
   Quantity   int   not null
       constraint CK_Order_Details_Quantity
           check ([Quantity] > 0),
   Unit_Price money not null
       constraint CK_Order_Details_Price
           check ([Unit_Price] > 0),
   constraint PK_Order_Details
       primary key (Order_ID, Product_ID)
)
```

Nazwa tabeli: Individual_Discounts

Opis tablicy: Tabela odpowiadająca za informację o rabatach dla klientów indywidualnych

Opis pól:

- *Individual_Discount_ID (int):* klucz główny
- *Client_ID (int):* klucz obcy Individual_Clients - Indeks
- *Param_ID (int):* Klucz obcy Discount_Params
- *Start_Date (datetime):* Data rozpoczęcia ważności rabatu
- *End_Date (datetime):* Data końca ważności rabatu (null w przypadku nieograniczonego czasowo rabatu)
- Multiplier (int) : mnożnik rabatu, wykorzystywany przy rabatach długoterminowych. Wartość zero oznacza, że rabat jest już nieważny

Warunki integralności:

- Data końca jest nullem albo (Data końca > Data rozpoczęcia)

Kod, generujący tabelę:

```
create table Individual_Discounts
(
   Client_ID              int           not null
       constraint FK_Individual_Discount_Client
           references Individual_Clients,
   Param_ID               char(50)      not null
       constraint FK_Individual_Discount_Param
           references Discount_Params,
   Start_Date             datetime      not null,
   End_Date               datetime,
   Individual_Discount_ID int           not null
       constraint PK_Individual_Discounts
           primary key,
   multiplier             int default 1 not null,
   constraint CK_Individual_Client_End
       check ([End_Date] IS NULL OR [End_Date] > [Start_Date])
)
```

Nazwa tabeli: Company_Discounts

Opis tablicy: Tabela, odpowiadająca za informację o rabatach dla klientów firmowych

Opis pól:

- *Company_Discount_ID (int):* klucz główny
- *Client_ID (int):* klucz obcy do Companies
- *Param_ID (int):* Klucz obcy Discount_Params
- *Start_Date (datetime):* Data rozpoczęcia ważności rabatu
- Multiplier (int) : Mnożnik rabatu używany dla rabatów długoterminowych. Wartość 0 oznacza, że rabat został wykorzystany

Kod, generujący tabelę:

```
create table Company_Discounts
(
   Client_ID           int           not null
       constraint FK_Company_Discount
           references Companies,
   Param_ID            char(50)      not null
       constraint FK_Company_Discount_Param
           references Discount_Params,
   Start_Date          datetime      not null,
   End_Date            datetime,
   Company_Discount_ID int           not null
       constraint PK_Company_Discount
           primary key,
   multiplier          int default 1 not null,
   constraint CK_Company_Discount_Start_End
       check ([End_Date] IS NULL OR [End_Date] > [Start_Date])
)
```

Nazwa tabeli: Discount_Params

Opis tablicy: Słownik zawierający informację o poszczególnych rodzajach rabatów

Opis pól:

- *Param_ID (int):* klucz główny
- *Value (float):* Współczynnik rabatu

Warunki integralności:

- Wartość > 0

Kod, generujący tabelę:

```
create table Discount_Params
(
   Param_ID char(50) not null
       constraint PK_Discount_Params
           primary key,
   Value    float    not null
       constraint CK_Discount_Params_Value
           check ([Value] >= 0)
)
```

Nazwa tabeli: Kategorie_Produktów

Opis tablicy: Słownik zawierający kategorie gotowych produktów

Opis pól:

- *Category_ID (int) : klucz główny*
- *Name (char(50)) : Nazwa kategorii produktu*
- *Description (varchar(max)) : Opis tekstowy kategorii produktu*

Warunki integralności:

- Nazwa jest unikatowa
- Wartość defaultowa dla opisu jest “brak opisu”

Kod, generujący tabelę:

```
create table Product_Categories
(
   Category_ID int      not null
       constraint PK_Product_Categories
           primary key,
   Name         char(50) not null
       constraint UNQ_Category_Name
           unique,
   Description varchar(max) default 'no description'
)
```

Nazwa tabeli: Products

Opis tablicy: Słownik produktów, jakie mogą być wystawione na sprzedaż w dowolnym lokalu

Opis pól:

- *Product_ID (int) : Klucz główny*
- *Category_ID (int) : ID kategorii do której produkt należy*
- *Name (char(50)) : Nazwa produktu*

Warunki integralności:

- *Nazwa powinna być unikatowa*

Kod, generujący tabelę:

```
create table Products
(
   Product_ID  int      not null
       constraint PK_Products
           primary key,
   Category_ID int      not null
       constraint FK_Products_Category
           references Product_Categories,
   Name        char(50) not null
       constraint UNQ_Product_Name
           unique
)
```

Nazwa tabeli: Menu

Opis tablicy: Opisuje relację zachodzącą pomiędzy Produktami a Lokalami, prezentując listę produktów dostępnych w danym lokalu

Opis pól:

- *Product_ID (int) : Klucz główny, klucz obcy tablicy Gotowe_Produkty*
- *Local_ID (int) : Klucz główny, klucz obcy tablicy Lokale*
- *Date_Added (datetime) : Data ostatniego dodania produktu do menu w danym lokalu*
- *Date_Removed (datetime) : Data ostatniego usunięcia produktu z menu w danym lokalu (Null oznacza, że produkt znajduje się aktualnie w menu)*
- *Unit_Price (money) : Cena jednostkowa danego produktu w danym lokalu*
- *Quantity (int) : Ilość dostępnych jednostek produktu w lokalu*

Warunki integralności:

- Ilosc >= 0
- Cena > 0
- Data usunięcia jest nullem albo Data usunięcia większa niż data dodania

Kod, generujący tabelę:

```
create table Menu
(
   Product_ID   int      not null
       constraint FK_MENU_PRODUCT
           references Products,
   Local_ID     int      not null
       constraint FK_Menu_Local
           references Locals,
   Date_Added   datetime not null,
   Date_Removed datetime,
   Unit_Price   money    not null
       constraint CK_Menu_Unit_Price
           check ([Unit_Price] > 0),
   constraint PK_Menu
       primary key (Product_ID, Local_ID),
   constraint CK_Menu_Removed
       check ([Date_Removed] IS NULL OR [Date_Added] < [Date_Removed])
)
```

Nazwa tabeli: Products_Ingredients

Opis tablicy: Tabela łącząca tabele Products z Ingredients, opisując jakich półproduktów potrzebuje produkt do wykonania

Opis pól:

- *Product_ID (int) : Klucz główny, klucz obcy Products*
- *Ingredient_ID (int) : Klucz główny, klucz obcy Ingredients*

Kod, generujący tabelę:

```
create table Products_Ingredients
(
   Product_ID    int not null
       constraint FK_Products_Products_Ingredients
           references Products,
   Ingredient_ID int not null
       constraint FK_Products_Ingredients_Ingredients
           references Ingredients,
   constraint PK_Produkty_Polprodukty
       primary key (Product_ID, Ingredient_ID)
)
```

Nazwa tabeli: Ingredients

Opis tablicy: Tabela opisująca listę półproduktów możliwych do wykorzystywania

Opis pól:

- *Ingredient_ID (int) : Klucz główny - Indeks*
- *Category_ID (int) : ID Kategorii do której półprodukt należy*
- *Name (char(30)) : Nazwa półproduktu*

Warunki integralności:

- Nazwa powinna być unikatowa

Kod, generujący tabelę:

```
create table Ingredients
(
   Ingredient_ID int      not null
       constraint PK_Ingredients
           primary key,
   Category_ID   int      not null
       constraint FK_Ingredient_Category
           references Ingredient_Categories,
   Name          char(30) not null
       constraint UNQ_Ingredient_Name
           unique
)
```

Nazwa tabeli: Ingredient_Categories

Opis tablicy: Słownik zawierający kategorie półproduktów

Opis pól:

- *Category_ID (int) : Klucz główny*
- *Name (char(50)) : nazwa kategorii*

Warunki integralności:

- Nazwa powinna być unikatowa

Kod, generujący tabelę:

```
create table Ingredient_Categories
(
   Category_ID int      not null
       constraint PK_Ingredients_Categories
           primary key,
   Name        char(50) not null
       constraint UNQ_Ingredient_Category_Name
           unique
)
```

Nazwa tabeli: Local_Ingredients

Opis tablicy: Lista półproduktów w danym lokalu, wraz z ich stanem oraz relacją z informacjami o dostawcach.

Opis pól:

- *Local_ID (int) : Klucz główny, klucz obcy tabeli Locals*
- *Igredient_ID (int) : Klucz główny, klucz obcy tabeli Ingredients*
- *Supplier_ID (int) : ID dostawcy danego półproduktu do lokalu - Indeks*
- *Quantity (int) : aktualna ilość półproduktów w danym lokalu*

Warunki integralności:

- Ilość >=0

Kod, generujący tabelę:

```
create table Local_Ingredients
(
   Local_ID      int not null
       constraint FK_Local_Ingredients_Local
           references Locals,
   Ingredient_ID int not null
       constraint FK_Local_Ingredients_Ingredients
           references Ingredients,
   Supplier_ID   int not null
       constraint FK_Local_Ingredients_Suppliers
           references Suppliers,
   Quantity      int not null
       constraint CK_Ingredient_Quantity
           check ([Quantity] >= 0),
   constraint PK_Local_Ingredients
       primary key (Local_ID, Ingredient_ID)
)
```

Nazwa tabeli: Suppliers

Opis tablicy: Tabela z listą dostawców wraz informacjami dotyczącymi

Opis pól:

- *Supplier_ID (int) : Klucz główny*
- *Name (char(50)) : Nazwa dostawcy*
- *City_ID (int) : Klucz obcy słownika Miasta - Indeks*
- *Phone (char(15)) : Numer telefonu kontaktowego do dostawcy, null oznacza brak*
- *Street (char(50)) : Ulica w adresie firmy*
- *House_no (char(50)) : Numer budynku w adresie firmy*
- *Apartment_no (char(50)) : Numer lokalu w adresie firmy (Null oznacza, że adres firmy nie zawiera numeru lokalu)*
- *Postal_Code (char(20)) : Kod pocztowy adresu firmy*

Warunki integralności:

- Telefon powinien zawierać tylko cyfry (i opcjonalnie plusa na początku)

Kod, generujący tabelę:

```
create table Suppliers
(
   Supplier_ID  int      not null
       constraint PK_Suppliers
           primary key,
   Name         char(50) not null,
   Phone        char(50),
   Street       char(50) not null,
   House_no     char(50) not null,
   Apartment_no char(50),
   Postal_Code  char(20),
   City_ID      int      not null
       constraint FK_Suppliers_Cities
           references Cities
)
```

Nazwa tabeli: Clients

Opis tablicy: Lista ID klientów wspólnych dla firm i klientów indywidualnych, używana do składania zamówień.

Opis pól:

- Client_ID (int) : Klucz główny

Kod, generujący tabelę:

```
create table Clients
(
   Client_ID int identity
       constraint PK_Clients
           primary key
)
```

Nazwa tabeli: Companies

Opis tablicy: Lista firm dokonujących zamówień oraz rezerwacji, wraz z informacjami kontaktowymi oraz do faktury

Opis pól:

- *Client_ID (int) : Klucz główny, klucz obcy tabeli Clients*
- *Name (char(50)) : Nazwa firmy*
- *City_ID (int) : Klucz obcy słownika Cities - **CITY_ID***
- *NIP (char(10)) : Numer identyfikacji podatkowej do faktury VAT*
- *Email (char(30)) : Kontaktowy adres email do firmy*
- *Street (char(50)) : Ulica w adresie firmy*
- *House_no (char(50)) : Numer budynku w adresie firmy*
- *Apartment_no (char(50)) : Numer lokalu w adresie firmy (Null oznacza, że adres firmy nie zawiera numeru lokalu)*
- *Postal_Code (char(20)) : Kod pocztowy adresu firmy*

Warunki integralności:

- Nip powinien być unikatowy
- Nip powinien być numerem
- Email powinien zawierać @

Kod, generujący tabelę:

```
create table Companies
(
   Client_ID    int identity
       constraint PK_Companies
           primary key
       constraint FK_Companies_Clients
           references Clients,
   Name         char(50) not null,
   City_ID      int      not null
       constraint FK_Companies_Cities
           references Cities,
   NIP          char(10) not null
       constraint UNQ_Company_NIP
           unique
       constraint CK_Companies_NIP
           check (NOT [NIP] like '%[^0-9]%'),
   Email        char(50) not null
       constraint CK_Company_Email
           check ([Email] like '%@%_._%'),
   Street       char(50) not null,
   House_no     char(50) not null,
   Apartment_no char(50),
   Postal_Code  char(20) not null
)
```

Nazwa tabeli: Individual_Clients

Opis tablicy: Lista klientów indywidualnych dokonujących zamówień oraz rezerwacji, wraz z informacjami do faktury

Opis pól:

- *Client_ID (int) : Klucz główny, klucz obcy tabeli Clients*
- *City_ID (int) : Klucz obcy słownika Cities - Indeks*
- *Phone (char(15)) : Numer telefonu kontaktowego do klienta*
- *Street (char(50)) : Ulica w adresie firmy*
- *House_no (char(50)) : Numer budynku w adresie firmy*
- *Apartment_no (char(50)) : Numer lokalu w adresie firmy (Null oznacza, że adres firmy nie zawiera numeru lokalu)*
- *Postal_Code (char(20)) : Kod pocztowy adresu firmy*
- *Firstname (char(50)) : Imię klienta*
- *Lastname (char(50)) : Nazwisko klienta*

Null, w dowolnym polu adresu, bądź numeru telefonu oznacza, że klient nie podał danych.

Warunki integralności:

- Numer telefonu powinien się składać tylko z cyfr

Kod, generujący tabelę:

```
create table Individual_Clients
(
   Client_ID    int not null
       constraint PK_Individual_Client
           primary key
       constraint FK_Individual_Clients
           references Clients,
   City_ID      int
       constraint FK_Individual_Clients_Cities
           references Cities,
   Phone        nchar(30) not null
       constraint CK_Individual_Client_Phone
           check (NOT [Phone] like '%[^0-9\+]%'),
   Street       char(50) not null,
   House_no     char(50) not null,
   Apartment_no char(50),
   Postal_Code  char(20) not null,
   Firstname    char(50) not null,
   Lastname     char(50) not null
)
```

Nazwa tabeli: Cities

Opis tablicy: Słownik Miast

Opis pól:

- *City_ID (int) : Klucz główny*
- *Country_ID (int) : Klucz obcy tabeli Państwa*
- *Name (char(50)) : Nazwa Miasta*

Warunki integralności:

- Nazwa powinna być unikatowa

Kod, generujący tabelę:

```
create table Cities
(
   City_ID    int identity
       constraint PK_Cities
           primary key,
   Name       char(50) not null
       constraint UNQ_City_Name
           unique,
   Country_ID int      not null
       constraint FK_City_Country
           references Countries
)
```

Nazwa tabeli: Countries

Opis tablicy: Słownik Państw

Opis pól:

- *Country_ID (int) : Klucz główny*
- *Country_Name (char(50)) : Nazwa Państwa*

Warunki integralności:

- Nazwa powinna być unikatowa

Kod, generujący tabelę:

```
create table Countries
(
   Country_ID   int identity
       constraint PK_Panstwa
           primary key,
   Country_Name char(50) not null
       constraint UNQ_Nazwa_Panstwa
           unique
)
```

Nazwa tabeli: Log

Opis tablicy: Tablica przechowująca informacje logowane przez procedury w jobach.

Opis pól:

- *Message_ID (int) : klucz główny*
- *Message_date (datetime): data dodania wiadomości do logu*
- *Message (varchara(MAX)): treść wiadomości*

Warunki integralności:

- Żadne z pól nie może być nullem

Kod, generujący tabelę:

```
create table Log
(
   Message_ID   int          not null
       constraint Log_pk
           primary key nonclustered,
   Message_date datetime     not null,
   Message      varchar(max) not null
)
```

Widoki

Nazwa widoku: VW_Reservations

Opis widoku:
Widok reprezentujący łączący informację na temat zarezerwowanych stolików przez firmy oraz klientów indywidualnych.

Kod generujący widok:

```
CREATE VIEW dbo.VW_Reservations AS
SELECT T.Table_ID                   AS [Table ID],
      T.Local_ID                    AS [Local ID],
      CR.Start_Date                 AS [Start date],
      CR.End_Date                   AS [End date],
      'COMPANY'                     AS [Reservation type],
      CR.Company_ID                 AS [Client ID],
      CR.Company_Reservation_ID     AS [Reservation ID]

FROM dbo.Tables T
  INNER JOIN Company_Reservation_Tables CRT ON T.Table_ID = CRT.Table_ID
  INNER JOIN Company_Reservations CR ON CRT.Company_Reservation_ID =
CR.Company_Reservation_ID

UNION

SELECT T2.Table_ID                  AS [Table ID],
      T2.Local_ID                   AS [Local ID],
      IR.Start_Date                 AS [Start date],
      IR.End_date                   AS [End date],
      'INDIVIDUAL'                  AS [Reservation type],
      O.Client_ID                   AS [Client ID],
      IR.Individual_Reservation_ID AS [Reservation ID]

FROM dbo.Tables T2
  INNER JOIN Individual_Reservations IR on T2.Table_ID = IR.Table_ID
  INNER JOIN Orders O on IR.Order_ID = O.Order_ID
```

Nazwa widoku: VW_Current_Reservations

Opis widoku:
Widok na VW_Reservations, pokazujący tylko przyszłe rezerwacje.

Kod generujący widok:

```
CREATE VIEW dbo.VW_Current_Reservations AS
SELECT * FROM dbo.VW_Reservations WHERE [Start date] > GETDATE()
```

Nazwa widoku: VW_Local_Address

Opis widoku:
Widok pokazujący pełne dane adresowe wszystkich lokali.
Kod generujący widok:

```
CREATE VIEW dbo.VW_Local_Address AS
SELECT L.Local_ID,
     L.Street,
     L.House_no,
     L.Apartment_no,
     L.Postal_Code,
     'Postal code: ' + L.Postal_Code + 'Street: ' + L.Street +
'House number:  ' + L.House_no + 'Apartments: ' +
ISNULL(L.Apartment_no, '') + ' ' AS Address,
     C.Name AS City,
     C2.Country_Name AS Country
FROM Locals L
  INNER JOIN Cities C on C.City_ID = L.City_ID
  INNER JOIN Countries C2 on C2.Country_ID = C.Country_ID
```

Nazwa widoku: VW_Order_Value

Opis widoku:
Widok odpowiadający za informację o zamówieniach i ich wartości.

Kod generujący widok:

```
CREATE VIEW VW_Order_Value AS
SELECT O.Order_ID,
    O.Client_ID,
    O.Local_ID,
    O.Order_Date,
    SUM(OD.Quantity * OD.Unit_Price) * (1 - O.Discount_Percent)AS Value
FROM Orders O
 INNER JOIN Order_Details OD on O.Order_ID = OD.Order_ID
GROUP BY O.Order_ID, O.Client_ID, O.Local_ID, O.Order_Date,
O.Discount_Percent
```

Nazwa widoku: VW_IsProductFromSeafood

Opis widoku:
Widok na liczbę półproduktów w lokalach

Kod generujący widok:

```
create view dbo.VW_IsProductFromSeafood as
   select p.Product_ID,p.Name,
       CAST(
            CASE
                 WHEN IC.Name='Seafood'
                    THEN 1
                 ELSE 0
            END AS bit) as IsSeaFood
   from Products p
   join Products_Ingredients PI on p.Product_ID = PI.Product_ID
   join Ingredients I on PI.Ingredient_ID = I.Ingredient_ID
   join Ingredient_Categories IC on I.Category_ID = IC.Category_ID
```

Nazwa widoku: VW_Menu_Stockpile

Opis widoku:
Widok na liczbę półproduktów w lokalach

Kod generujący widok:

```
CREATE VIEW VW_Menu_Stockpile AS
SELECT M.Product_ID,
      M.Local_ID,
      M.Date_Added,
      M.Date_Removed,
      M.Unit_Price,
      ISNULL(MIN(LI.Quantity), 0) AS 'Stockpile'
FROM Menu M
   LEFT OUTER JOIN Products_Ingredients PI on M.Product_ID = PI.Product_ID
   LEFT OUTER JOIN Local_Ingredients LI on M.Local_ID = LI.Local_ID AND
PI.Ingredient_ID = LI.Ingredient_ID
GROUP BY M.Product_ID, M.Local_ID, M.Date_Added, M.Date_Removed,
M.Unit_Price
```

Nazwa widoku: VW_Last_Company_Discount

Opis widoku:
Widok na ostatnie rabaty uzyskane przez firmy.

Kod generujący widok:

```
CREATE VIEW VW_Last_Company_Discount
AS
SELECT CD.Company_Discount_ID,
      CD.Client_ID,
      CD.Param_ID,
      CD.Start_Date,
      CD.multiplier
FROM Company_Discounts CD
   INNER JOIN (
       SELECT Param_ID,
              Client_ID,
              MAX(Start_Date) Start_Date
       FROM Company_Discounts
       GROUP BY Param_ID, Client_ID
   ) T
   ON CD.Param_ID = T.Param_ID AND CD.Client_ID = T.Client_ID AND CD.Start_Date
= T.Start_Date
```

Nazwa widoku: VW_Last_Individual_Discount

Opis widoku:
Widok na ostatnie rabaty dla klientów indywidualnych.

Kod generujący widok:

```
CREATE VIEW VW_Last_Individual_Discount
AS
SELECT ID.Individual_Discount_ID,
      ID.Param_ID,
      ID.Client_ID,
      ID.Start_Date,
      ID.End_Date,
      Id.multiplier
FROM Individual_Discounts ID
   INNER JOIN (
       SELECT Param_ID,
              Client_ID,
              MAX(Start_Date) Start_Date
       FROM Individual_Discounts
       GROUP BY Param_ID, Client_ID
   ) T
   ON ID.Param_ID = T.Param_ID AND ID.Start_Date = T.Start_Date AND ID.Client_ID
= T.Client_ID
```

Nazwa widoku: VW_Current_Menu

Opis widoku:
Widok na obecne menu

Kod generujący widok:

```
create view VW_Current_Menu as
   select *
   from Menu
   where Menu.Date_Removed is null
```

PROCEDURY

Nazwa procedury: Add_Individual_Client

Opis: Dodaje nowego klienta indywidualnego do bazy

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Individual_Client
@FirstName char(50),
@LastName char(50),
@City char(50),
@Street char(50),
@House_no char(50),
@Postal_Code char(20),
@Phone char(30),
@Apartment_no char(50) = NULL
AS
 SET NOCOUNT ON
 -- Create new Client ID
 DECLARE @Client_ID int;
 INSERT INTO Clients
 DEFAULT VALUES
 SET @Client_ID = SCOPE_IDENTITY()
 -- Get City ID
 DECLARE @City_ID int
 IF (@City IS NULL)
     BEGIN
         SET @City_ID = NULL
     END
 ELSE
     BEGIN
       SET @City_ID = (SELECT City_ID FROM Cities WHERE Name = @City)
     END
 -- Insert data
 INSERT INTO Individual_Clients(
    Client_ID, Firstname, Lastname, City_ID, Street, House_No,
    Apartment_no, Postal_Code, Phone
 )
 VALUES (
    @Client_ID, @FirstName, @LastName, @City_ID, @Street, @House_no,
    @Apartment_no, @Postal_Code, @Phone
 )
```

Nazwa procedury: Add_Company

Opis: Dodaje nowego klienta firmowego do bazy

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Company
@Name char(50),
@NIP char(10),
@Email char(50),
@City char(50),
@Street char(50),
@House_no char(50),
@Postal char(20),
@Apartment_no char(50) = NULL
AS
  SET NOCOUNT ON
  -- Create new Client ID
  DECLARE @Client_ID int;
  INSERT INTO Clients
  DEFAULT VALUES;
  SET @Client_ID = SCOPE_IDENTITY()
  -- Get City ID
  DECLARE @City_ID int
  IF (@City IS NULL)
    BEGIN
      SET @City_ID = NULL
    END
  ELSE
    BEGIN
      SET @City_ID = (SELECT City_ID FROM Cities WHERE Name = @City)
    END
  INSERT INTO Companies(
   Client_ID, Name, NIP, Email, Street, House_no, Apartment_no,
   Postal_Code, City_ID)
  VALUES (
   @Client_ID, @Name, @NIP, @Email, @Street, @House_no, @Apartment_no,
   @Postal, @City_ID)
  INSERT INTO Company_Discounts(Client_ID, Param_ID, Start_Date,
multiplier)
  VALUES(@Client_ID, 'FR1', GETDATE(), 0)
```

Nazwa procedury: Add_Product

Opis: Dodaje nowy produkt o podanej nazwie i kategorii do bazy

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Product
  @ProductName char(50),
  @CategoryName char(50)
AS
   SET NOCOUNT ON
   -- Get category ID
  DECLARE @Category_ID int
  SET @Category_ID = (SELECT Category_ID FROM Product_Categories WHERE Name
= @CategoryName)

  INSERT INTO Products(Category_ID, Name)
  VALUES (@Category_ID, @ProductName)
GO
```

Nazwa procedury: Add_Ingredient

Opis: Dodaje nowy składnik o podanej nazwie i kategorii do bazy

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Ingredient
  @IngredientName char(50),
  @CategoryName char(50)
AS
  SET NOCOUNT ON
  -- Get category ID
  DECLARE @Category_ID int
  SET @Category_ID = (SELECT Category_ID FROM Ingredient_Categories WHERE
Name = @CategoryName)

  INSERT INTO Ingredient_Categories(Category_ID, Name)
  VALUES (@Category_ID, @IngredientName)
GO
```

Nazwa procedury: Add_Ingredient_To_Product

Opis: Łączy składnik z produktem, w celu automatycznej dekrementacji liczby składników i liczenia liczby produktów na stanie

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Ingredient_To_Product
  @ProductName char(50),
  @IngredientName char(50)
AS
  SET NOCOUNT ON
  -- Get product ID
  DECLARE @Product_ID int
  SET @Product_ID = (SELECT Product_ID FROM Products WHERE Name =
@ProductName)
  -- Get ingredient ID
  DECLARE @Ingredient_ID int
  SET @Ingredient_ID = (SELECT Ingredient_ID FROM Ingredients WHERE Name =
@IngredientName)
  -- Create link
  INSERT INTO Products_Ingredients(Product_ID, Ingredient_ID)
  VALUES (@Product_ID, @Ingredient_ID)
GO
```

Nazwa procedury: Remove_Ingredient_From_Product

Opis: Usuwa połączenie produktu ze składnikiem

Kod generujący procedurę:

```
CREATE PROCEDURE Remove_Ingredient_From_Product
  @ProductName char(50),
  @IngredientName char(50)
AS
  SET NOCOUNT ON
  -- Get product ID
  DECLARE @Product_ID int
  SET @Product_ID = (SELECT Product_ID FROM Products WHERE Name =
@ProductName)
  -- Get ingredient ID
  DECLARE @Ingredient_ID int
  SET @Ingredient_ID = (SELECT Ingredient_ID FROM Ingredients WHERE Name =
@IngredientName)
  -- Drop link
  DELETE FROM Products_Ingredients
  WHERE Ingredient_ID = @Ingredient_ID AND Product_ID = @Product_ID
GO
```

Nazwa procedury: Add_New_Order

Opis: Dodaje nowe puste zamówienie

Kod generujący procedurę:

```
create procedure Add_New_Order(@Client_ID int,
                        @Local_ID int, @Order_Date datetime,
@Payment_Method int, @Pref_Date datetime = null,
                        @Delivery_Date datetime = null, @Paid bit = 0)
as
begin
set nocount on
insert into Orders(Client_ID, Local_ID, Order_Date, Pref_Date,
Delivery_Date, Paid, Payment_Method)
VALUES (@Client_ID, @Local_ID, @Order_Date, @Pref_Date,
        @Delivery_Date, @Paid, @Payment_Method)
DECLARE @Order_ID int = SCOPE_IDENTITY()
IF dbo.Is_Company(@Client_ID) = 1
   BEGIN
       EXEC dbo.Apply_Company_Discount @Client_ID, @Order_ID
   END
ELSE
    BEGIN
        EXEC dbo.Apply_Individual_Discount @Client_ID, @Order_ID
    END
RETURN(@Order_ID)
end
```

Nazwa procedury: Append_Product_To_Order

Opis: Dodaje daną liczbę produktów do zamówienia

Kod generujący procedurę:

```
CREATE PROCEDURE Append_Product_To_Order @Order_ID int,
                                         @ProductName char(50),
                                         @Quantity int = 1
AS
 -- Get local ID
  DECLARE @Local_ID int = (SELECT Local_ID
                  FROM Orders
                  WHERE Order_ID = @Order_ID)
 -- Get product ID
  DECLARE @Product_ID int = (SELECT Product_ID
                    FROM Products
                    WHERE Name = @ProductName)

 -- Check for Real Product Quantity to be less than wanted
  DECLARE @Real_Quantity int = (select
[dbo].Check_Product_Status(@ProductName, @Local_ID))
 if (@Real_Quantity < @Quantity)
     begin
         RAISERROR ('Cannot add product, due to lack of products. Currently
having: ', 1, 1);
     end
 else
     begin

         DECLARE @Unit_Price int
         SET @Unit_Price = (
             SELECT Unit_Price
             FROM Menu
             WHERE Product_ID = @Product_ID
               AND Local_ID = @Local_ID)
         INSERT INTO Order_Details(Order_ID, Product_ID, Quantity,
Unit_Price)
         VALUES (@Order_ID, @Product_ID, @Quantity, @Unit_Price)

         exec Change_Product_Ingredients_Quantity @Product_ID, @Local_ID,
@Quantity
     end
```

Nazwa procedury: Remove_Product_From_Order

Opis: Usuwa produkt z zamówienia

Kod generujący procedurę:

```
CREATE PROCEDURE Remove_Product_From_Order @Order_ID int,
                                    @ProductName char(50)
AS

 -- Get product ID
    DECLARE @Product_ID int = (SELECT Product_ID
                    FROM Products
                    WHERE Name = @ProductName)
        begin
            delete from Order_Details
            where Product_ID=@Product_ID and Order_ID=@Order_ID
        end
```

Nazwa procedury: Change_Product_Ingredients_Quantity

Opis: Dodaje daną liczbę produktów do zamówienia

Kod generujący procedurę:

```
create procedure Change_Product_Ingredients_Quantity
   @ProductID int,
   @LocalID int,
   @UsedQuantity int
as
UPDATE Local_Ingredients
SET Quantity =
          (select QI.Quantity
           from VW_Quantity_Of_Ingredient QI
           where (QI.Ingredient_ID = Local_Ingredients.Ingredient_ID and
QI.Product_ID = @ProductID AND QI.Local_ID = @LocalID))
          - @UsedQuantity
WHERE Local_Ingredients.Ingredient_ID IN (
  SELECT Ingredient_ID
  FROM Products_Ingredients
  WHERE Product_ID = @ProductID AND Local_ID = @LocalID)
GO
```

Nazwa procedury: Apply_Individual_Discount

Opis: Nakłada wszystkie możliwe zniżki dla klienta indywidualnego o podanym ID dla zamówienia

Kod generujący procedurę:

```
CREATE PROCEDURE Apply_Individual_Discount
   @Client_id int,
   @Order_id int
AS
   DECLARE @lifetime float = dbo.Get_Individual_Lifetime_Discount(@Client_id)
   DECLARE @single int = dbo.Get_Individual_R2_Discount(@Client_id)
   IF @single IS NOT NULL
       BEGIN
           SET @lifetime = @lifetime + (SELECT Value FROM Discount_Params WHERE
Param_ID = 'R2')
           UPDATE Individual_Discounts
           SET multiplier = 0
           WHERE Individual_Discount_ID = @single
       END
   SET @single = dbo.Get_Individual_R3_Discount(@Client_id)
   IF @single IS NOT NULL
       BEGIN
           SET @lifetime = @lifetime + (SELECT Value FROM Discount_Params WHERE
Param_ID = 'R2')
           UPDATE Individual_Discounts
           SET multiplier = 0
           WHERE Individual_Discount_ID = @single
       END
   UPDATE Orders
   SET Discount_Percent = @lifetime
   WHERE Order_ID = @Order_id
```

Nazwa procedury: Apply_Company_Discount

Opis: Nakłada wszystkie możliwe zniżki dla klienta firmowego o podanym ID dla zamówienia

Kod generujący procedurę:

```
CREATE PROCEDURE Apply_Company_Discount
   @Client_id int,
   @Order_id int
AS
   DECLARE @monthly float = dbo.Get_Company_Monthly_Discount(@Client_id)
   DECLARE @qt int = dbo.Get_Company_Quarter_Discount(@Client_id)
   if @qt IS NOT NULL
       BEGIN
           SET @monthly = @monthly + (SELECT VALUE FROM Discount_Params WHERE
Param_ID = 'FR2')
           UPDATE Company_Discounts
           SET multiplier = 0
           WHERE Company_Discount_ID = @qt
       END
   UPDATE Orders
   SET Discount_Percent = @monthly
   WHERE Order_ID = @Order_id
```

Nazwa procedury: Check_Menu

Opis: Sprawdza warunek w menu, czy przynajmniej połowa produktów została zmieniona w przeciągu dwóch tygodni. Jeśli tak nie jest, dodaje informację odnośnie tego do tablicy *Log.*

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Menu
   @Local_ID int
AS
   DECLARE @FreshProducts int = (
       SELECT COUNT(*)
       FROM Menu
       WHERE Local_ID = @Local_ID AND
             DATEDIFF(week, Date_Added, GETDATE()) < 2
   )
   DECLARE @TotalProducts int = (
       SELECT COUNT(*)
       FROM Menu
       WHERE Local_ID = @Local_ID
   )
   IF @FreshProducts * 2 < @TotalProducts
       BEGIN
           INSERT INTO Log(Message_date, Message)
           VALUES (GETDATE(), CONCAT('Menu should be updated in local no. ',
@Local_ID))
       END
```

Nazwa procedury: Check_All_Menus

Opis: Wywołuje sprawdzenie warunku Menu dla wszystkich lokali. Procedura ta powinna zostać dodana do jobów bazy danych, z wykonywaniem jej co jeden dzień.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_All_Menus
AS
  DECLARE @local_id int
  DECLARE @local_cursor CURSOR
  SET @local_cursor = CURSOR FOR
  SELECT Local_ID FROM Menu
  OPEN @local_cursor
  FETCH NEXT FROM @local_cursor
  INTO @local_id
  WHILE @@FETCH_STATUS = 0
  BEGIN
      EXEC Check_Menu @local_id;
      FETCH NEXT FROM @local_cursor
      INTO @local_id
  END
```

Nazwa procedury: Check_Individual_Discount

Opis: Procedura sprawdzająca, czy klient indywidualny zasługuje na nowy rabat, jeśli tak, dodaje go.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Individual_Discount
   @client_id int
AS
   -- Discount params
   DECLARE @D1 int = (SELECT CAST(Value as int) FROM Discount_Params WHERE
Param_ID = 'D1')
   DECLARE @D2 int = (SELECT CAST(Value as int) FROM Discount_Params WHERE
Param_ID = 'D2')
   DECLARE @K1 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'K1')
   DECLARE @K2 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'K2')
   DECLARE @K3 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'K3')
   DECLARE @Z1 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'Z1')
   -- Check long term
   DECLARE @LONG_COUNT int = (
       SELECT multiplier
       FROM dbo.VW_Last_Individual_Discount
       WHERE Param_ID = 'R1' AND Client_ID = @client_id
   )
   DECLARE @DISCOUNT_ORDERS int = (
       SELECT COUNT(*)
       FROM VW_Order_Value
       WHERE Client_ID = @client_id AND Value >= @K1
   )
   IF @LONG_COUNT = 0 AND @DISCOUNT_ORDERS >= @Z1
       BEGIN
           INSERT INTO Individual_Discounts(Client_ID, Param_ID,
Start_Date, End_Date)
           VALUES (@client_id, 'R1', GETDATE(), NULL)
       END
   ELSE IF @LONG_COUNT = 1 AND @DISCOUNT_ORDERS >= 2 * @Z1
       BEGIN
           UPDATE Individual_Discounts
           SET multiplier = 2
           WHERE Client_ID = @client_id AND Param_ID = 'R1'
       end
   -- Check short term 1
   DECLARE @LAST_TIME datetime = (
       SELECT Start_Date
       FROM VW_Last_Individual_Discount
       WHERE Param_ID = 'R2' AND Client_ID = @client_id
```

```
    )
    DECLARE @LAST_VALUE money = (
        SELECT SUM(Value)
        FROM VW_Order_Value
        WHERE Client_ID = @client_id AND (
            @LAST_TIME IS NULL OR
            Order_Date > @LAST_TIME
    ))
    IF @LAST_VALUE > @K2
        BEGIN
            INSERT INTO Individual_Discounts(Client_ID, Param_ID,
Start_Date, End_Date)
            VALUES (@client_id, 'R2', GETDATE(), DATEADD(day, @D1,
GETDATE()))
        END
    -- Check short term 2
    SET @LAST_TIME = (
        SELECT Start_Date
        FROM VW_Last_Individual_Discount
        WHERE Param_ID = 'R3' AND Client_ID = @client_id
    )
    SET @LAST_VALUE = (
        SELECT SUM(Value)
        FROM VW_Order_Value
        WHERE Client_ID = @client_id AND (
            @LAST_TIME IS NULL OR
            Order_Date > @LAST_TIME
    ))
    IF @LAST_VALUE > @K3
        BEGIN
            INSERT INTO Individual_Discounts(Client_ID, Param_ID,
Start_Date, End_Date)
            VALUES (@client_id, 'R3', GETDATE(), DATEADD(day, @D2,
GETDATE()))
        END
```

Nazwa procedury: Check_Monthly_Discount

Opis: Procedura dokonująca sprawdzenia warunków rabatowych miesięcznych dla wszystkich klientów firmowych. Procedura ta powinna zostać dodana do jobów bazy danych z wykonywaniem się co miesiąc.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Monthly_Discount AS
   DECLARE @Company_ID int
   DECLARE @cursor CURSOR
   SET @cursor = CURSOR FOR
   SELECT Client_ID FROM Companies
   OPEN @cursor
   FETCH NEXT FROM @cursor INTO @Company_ID
   WHILE @@FETCH_STATUS = 0
       BEGIN
           EXEC Check_Month_Company_Discount @Company_ID
           FETCH NEXT FROM @cursor INTO @Company_ID
       END
```

Nazwa procedury: Check_Month_Company_Discount

Opis: Procedura sprawdzająca czy klient firmowy zasługuje na przedłużenie rabatu miesięcznego, a jeśli nie to go zeruje.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Month_Company_Discount
  @client_id int
AS
  -- Get discount params
  DECLARE @FZ int = (SELECT CAST(Value AS int) FROM Discount_Params WHERE
Param_ID = 'FZ')
  DECLARE @FK1 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'FK1')
  DECLARE @FM float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'FM')
  --
  DECLARE @LAST_MONTH int = (
      SELECT COUNT(*)
      FROM VW_Order_Value
      WHERE Client_ID = @client_id AND
            DATEDIFF(month, Order_Date, GETDATE()) < 1 AND
            Value >= @FK1
  )
  IF @LAST_MONTH >= @FZ AND dbo.Get_Company_Monthly_Discount(@client_id) <
@FM
      BEGIN
          DECLARE @last int = (SELECT multiplier FROM Company_Discounts
WHERE Client_ID = @client_id AND Param_ID = 'FR1')
          UPDATE Company_Discounts
          SET multiplier = @last + 1, Start_Date = GETDATE()
          WHERE Client_ID = @client_id AND Param_ID = 'FR1'
      END
  ELSE
      BEGIN
          UPDATE Company_Discounts
          SET multiplier = 0, Start_Date = GETDATE()
          WHERE Client_ID = @client_id AND Param_ID = 'FR1'
      end
```

Nazwa procedury: Check_Quarter_Discount

Opis: Procedura dokonująca sprawdzenia warunków rabatowych kwartalnych dla wszystkich klientów firmowych. Procedura ta powinna zostać dodana do jobów bazy danych z wykonywaniem się co kwartał.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Quarter_Discount AS
   DECLARE @Company_ID int
   DECLARE @cursor CURSOR
   SET @cursor = CURSOR FOR
   SELECT Client_ID FROM Companies
   OPEN @cursor
   FETCH NEXT FROM @cursor INTO @Company_ID
   WHILE @@FETCH_STATUS = 0
       BEGIN
           EXEC Check_Quarter_Company_Discount @Company_ID
           FETCH NEXT FROM @cursor INTO @Company_ID
       END
```

Nazwa procedury: Check_Quarter_Company_Discount

Opis: Procedura sprawdzająca czy klient firmowy zasługuje na otrzymanie rabatu kwartalnego, jeśli to go dodaje do bazy.

Kod generujący procedurę:

```
CREATE PROCEDURE Check_Quarter_Company_Discount
   @client_id int
AS
   DECLARE @FK2 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'K2')
   DECLARE @QT float = (
       SELECT SUM(Value) FROM VW_Order_Value
       WHERE Client_ID = @client_id AND DATEDIFF(quarter, Order_Date, GETDATE())
< 1
       )
   IF @FK2 <= @QT
       BEGIN
          INSERT INTO Company_Discounts(CLIENT_ID, PARAM_ID, START_DATE)
          VALUES(@client_id, 'FR2', GETDATE())
       END
```

Nazwa procedury: Reserve_Individual_Table

Opis: Procedura dokonująca procesu rezerwacji dla klienta indywidualnego.

Kod generujący procedurę:

```
CREATE PROCEDURE Reserve_Individual_Table
  @START datetime,
  @END datetime,
  @ORDER_ID int,
  @Table_ID int
AS
  DECLARE @LOCAL int = (SELECT Local_ID FROM Orders WHERE Order_ID =
@ORDER_ID)
  IF @Table_ID IN (SELECT Table_ID FROM dbo.Check_Available_Tables(@LOCAL,
@START, @END)) AND
     @ORDER_ID NOT IN (SELECT Order_ID FROM Individual_Reservations)
       BEGIN
       INSERT INTO Individual_Reservations(Start_Date, End_Date,
Confirmation_Date, Order_ID, Table_ID)
       VALUES(@START, @END, NULL, @ORDER_ID, @Table_ID)
       END
  ELSE
       BEGIN
           RAISERROR('Cannot create reservation', 1, 1)
       END
END
```

Nazwa procedury: Add_Company_Reservation

Opis: Procedura tworząca pustą rezerwację dla firmy.

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Company_Reservation
   @START datetime,
   @END datetime,
   @Client_ID int
AS
   INSERT INTO Company_Reservations(Company_ID, Start_Date, End_Date,
Confirmation_Date)
   VALUES(@Client_ID, @START, @END, NULL)
```

Nazwa procedury: Add_Table_Company_Reservation

Opis: Procedura dodająca stolik do rezerwacji firmowej.

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Table_Company_Reservation
  @Reservation_ID int,
  @Table_ID int
AS
  DECLARE @LOCAL int = (SELECT Local_ID FROM Tables WHERE Table_ID =
@Table_ID)
  DECLARE @START datetime = (SELECT Start_date FROM Company_Reservations
WHERE Company_Reservation_ID = @Reservation_ID)
  DECLARE @END datetime = (SELECT End_Date FROM Company_Reservations WHERE
Company_Reservation_ID = @Reservation_ID)
  IF @Table_ID IN (SELECT Table_ID FROM dbo.Check_Available_Tables(@LOCAL,
@START, @END))
      BEGIN
       INSERT INTO Company_Reservation_Tables(Table_ID,
Company_Reservation_ID)
       VALUES(@Table_ID, @Reservation_ID)
      END
  ELSE
      BEGIN
          RAISERROR('Chosen table is not available!', 1, 1)
      end
```

Nazwa procedury: Add_Worker_Company_Reservation

Opis: Procedura dodająca pracownika do zamówienia firmowego.

Kod generujący procedurę:

```
CREATE PROCEDURE Add_Worker_Company_Reservation
   @Reservation_ID int,
   @Firstname char(50),
   @Lastname char(50)
AS
   INSERT INTO Company_Reservation_Workers(Company_Reservation_ID, Firstname,
Lastname)
   VALUES(@Reservation_ID, @Firstname, @Lastname)
```

FUNKCJE

Nazwa funkcji: Is_Company

Opis: Funkcja zwracająca bit informujący czy klient jest firmą

Kod generujący procedurę:

```
CREATE FUNCTION Is_Company (@Client_ID int)
RETURNS BIT
AS
BEGIN
   DECLARE @RET BIT;
   IF EXISTS(SELECT * FROM Companies WHERE Client_ID = @Client_ID)
       BEGIN
           SET @RET = 1
       END
   ELSE
       BEGIN
           SET @RET = 0
       END
   RETURN @RET
END
```

Nazwa funkcji: Check_Product_Status

Opis: Zwraca maksymalna liczbę produktów jaką można wykonać w lokalu

Kod generujący funkcję:

```
CREATE FUNCTION Check_Product_Status (@ProductName char(50), @LocalID int)
RETURNS INT
AS
BEGIN
 -- Get product ID
 DECLARE @Product_ID int
 SET @Product_ID = (SELECT Product_ID FROM Products WHERE Name =
@ProductName)
 RETURN (
      SELECT MIN(LI.Quantity)
      FROM Local_Ingredients LI
          INNER JOIN Ingredients I on I.Ingredient_ID = LI.Ingredient_ID
          INNER JOIN Products_Ingredients PI on I.Ingredient_ID =
PI.Ingredient_ID
      WHERE PI.Product_ID = @Product_ID AND LI.Local_ID = @LocalID
      GROUP BY PI.Product_ID
)
END
```

Nazwa funkcji: Get_Individual_Lifetime_Discount

Opis: Zwraca procentowy rabat przysługujący klientowi na wszystkie przyszłe zamówienia

Kod generujący funkcję:

```
CREATE FUNCTION Get_Individual_Lifetime_Discount(@client_id int)
RETURNS float
AS
BEGIN
   DECLARE @R1 float = (SELECT Value FROM Discount_Params WHERE Param_ID = 'R1')
   DECLARE @MULT int = (SELECT multiplier FROM Individual_Discounts WHERE
Param_ID = 'R1' AND Client_ID = @client_id)
   DECLARE @RET float
   IF @MULT IS NULL
       BEGIN
           SET @RET = 0
       END
   ELSE
       BEGIN
           SET @RET = @MULT * @R1
       END
   RETURN @RET
END
```

Nazwa funkcji: Get_Individual_R2_Discount

Opis: Zwraca ID rabatu jednorazowego R2 jeśli klient takowy posiada, w przeciwnym przypadku zwraca null.

Kod generujący funkcję:

```
CREATE FUNCTION Get_Individual_R2_Discount(@client_id int)
RETURNS INT
as
BEGIN
   DECLARE @RET int = NULL
   IF EXISTS(
       SELECT * FROM Individual_Discounts
       WHERE Client_ID = @client_id AND
             multiplier != 0 AND
             Param_ID = 'R2' AND
             GETDATE() BETWEEN Start_Date AND End_Date
       )
       BEGIN
          SET @RET = (SELECT TOP 1 Individual_Discount_ID FROM
Individual_Discounts
               WHERE Client_ID = @client_id AND
               multiplier != 0 AND
               Param_ID = 'R2' AND
               GETDATE() BETWEEN Start_Date AND End_Date
              )
       END
   RETURN @RET
END
```

Nazwa funkcji: Get_Individual_R3_Discount

Opis: Zwraca ID rabatu jednorazowego R3 jeśli klient takowy posiada, w przeciwnym przypadku zwraca null.

Kod generujący funkcję:

```
CREATE FUNCTION Get_Individual_R3_Discount(@client_id int)
RETURNS INT
as
BEGIN
   DECLARE @RET int = NULL
   IF EXISTS(
       SELECT * FROM Individual_Discounts
       WHERE Client_ID = @client_id AND
             multiplier != 0 AND
             Param_ID = 'R3' AND
             GETDATE() BETWEEN Start_Date AND End_Date
       )
       BEGIN
          SET @RET = (SELECT TOP 1 Individual_Discount_ID FROM
Individual_Discounts
               WHERE Client_ID = @client_id AND
               multiplier != 0 AND
               Param_ID = 'R3' AND
               GETDATE() BETWEEN Start_Date AND End_Date
              )
       END
   RETURN @RET
END
```

Nazwa funkcji: Get_Company_Quarter_Discount

Opis: Zwraca ID rabatu kwartalnego jeśli firma takowy posiada, w przeciwnym przypadku zwraca null.

Kod generujący funkcję:

```
CREATE FUNCTION Get_Company_Quarter_Discount (@client_id int)
RETURNS int AS
BEGIN
   DECLARE @RET int = NULL
   IF EXISTS(
       SELECT * FROM Company_Discounts
       WHERE Client_ID = @client_id AND Param_ID = 'FR2' AND multiplier > 0
       )
       BEGIN
           SET @RET = (SELECT TOP 1 Company_Discount_ID FROM Company_Discounts
               WHERE Param_ID = 'FR2' AND Client_ID = @client_id AND multiplier
> 0)
       END
   RETURN @RET
END
```

Nazwa funkcji: Check_Company_Monthly_Discount

Opis: Zwraca procentowy rabat przysługujący klientowi firmowemu na wszystkie przyszłe zamówienia

Kod generujący funkcję:

```
CREATE FUNCTION Get_Company_Monthly_Discount(@client_id int)
RETURNS float
AS
BEGIN
   DECLARE @FR1 float = (SELECT Value FROM Discount_Params WHERE Param_ID =
'FR1')
   DECLARE @MULT int = (SELECT multiplier FROM Company_Discounts WHERE Client_ID
= @client_id AND Param_ID = 'FR1')
   DECLARE @RET float;
   IF @MULT IS NULL
       BEGIN
           SET @RET = 0
       END
   ELSE
       BEGIN
           SET @RET = @FR1 * @MULT
       end
   RETURN @MULT
End
```

Nazwa funkcji: Check_Available_Tables

Opis: Funkcja tablicowa zwracająca minimalną liczbę siedzeń dostępnych przy stole w lokalu w danym przedziale czasowym.

Kod generujący funkcję:

```
CREATE FUNCTION Check_Available_Tables(@local_id int, @from datetime, @to
datetime)
RETURNS TABLE
AS RETURN (
   SELECT Table_ID,
          Seats
   FROM dbo.Get_Tables_Capacity(@from, @to)
   WHERE Local_ID = @local_id
   )
```

Nazwa funkcji: Get_Tables_Capacity

Opis: Funkcja tablicowa zwracająca minimalną liczbę siedzeń dostępnych przy stole w danym przedziale czasowym.

Kod generujący funkcję:

```
CREATE FUNCTION Get_Tables_Capacity(@from datetime, @to datetime)
RETURNS TABLE AS RETURN
SELECT T.Table_ID,
     T.Local_ID,
     ISNULL(MIN(TR.Seats), T.Seats) AS [Seats]
FROM Tables T
   LEFT OUTER JOIN (
       SELECT * FROM Tables_Restrictions
       WHERE Start_Date BETWEEN @FROM AND @TO OR
             End_Date BETWEEN @FROM AND @TO
   ) TR on T.Table_ID = TR.Table_ID
   LEFT OUTER JOIN (
       SELECT * FROM VW_Reservations
       WHERE [Start date] BETWEEN @FROM AND @TO OR
             [End date] BETWEEN @FROM AND @TO
       ) R ON R.[Table ID] = T.Table_ID
WHERE R.[Table ID] IS NULL
GROUP BY T.Table_ID, T.Local_ID, T.Seats
```

Nazwa funkcji: generate_Invoice_For_Firms

Opis: Funkcja generuję fakturę dla firmy dla poszczególnego zamówienia

Kod generujący funkcję:

```
create function generate_Invoice_For_Firms(@Order_ID int)
returns table
return
select ('Bill from: ' + L.Name + 'Address: ' + VLA.Address) as 'Order Details'
from Orders
inner join Locals L on L.Local_ID = Orders.Local_ID
inner join VW_Local_Address VLA on L.Apartment_no = VLA.Apartment_no
where Orders.Order_ID=@Order_ID
union all
select top 1 CONCAT('Bill to: ', C.Name, 'NIP: ', C.NIP,
   'Postal code: ' + C.Postal_Code + 'Street: ' + C.Street + 'House number:  ' +
C.House_no + 'Apartments: ' + ISNULL(C.Apartment_no, '') + ' ')
from Orders
inner join Companies C on Orders.Client_ID = C.Client_ID
union all
select CONCAT('Order value is: ', VOV.Value)
from Orders
inner join VW_Order_Value VOV on Orders.Client_ID = VOV.Client_ID
where Orders.Order_ID=@Order_ID and VOV.Order_ID=@Order_ID
union all
select 'Product name: ' + P.Name+' Category name: '+ PC.Name+' Price: ' +
CONVERT(VARCHAR,Order_Details.Unit_Price )+ ' Quantity: ' +
CONVERT(VARCHAR,Order_Details.Quantity)
from Order_Details
inner join Products P on P.Product_ID = Order_Details.Product_ID
inner join Product_Categories PC on PC.Category_ID = P.Category_ID
where Order_ID=@Order_ID
```

TRIGGERY

Nazwa triggera: TR_Menu_Update

Opis: Trigger odpowiadający za sprawdzenie poprawności dat przy aktualizacji Menu. Sprawdza czy nowa data dodania jest z przynajmniej dziennym wyprzedzeniem, oraz czy produkt nie pojawia się ponownie po okresie krótszym niż jeden miesiąc.

Kod generujący trigger:

```
CREATE TRIGGER TR_Menu_Update
   ON Menu
FOR UPDATE
AS
   -- Check date_added
   IF DATEDIFF(day, GETDATE(), (SELECT MIN(Date_Added) FROM INSERTED)) < 1
       BEGIN
           RAISERROR ('Cannot update date_added to the day before tomorrow'
, 1, 1)
           ROLLBACK TRANSACTION
       END
   -- Check date_end
   IF (
       SELECT
              MAX(ISNULL(DATEDIFF(month, D.Date_Removed, I.Date_Added), 1))
       FROM INSERTED I
                INNER JOIN DELETED D ON D.Product_ID = I.Product_ID AND
D.Local_ID = I.Local_ID
   ) < 1
       BEGIN
           RAISERROR ('Product cannot be added again to the menu within a
month', 1, 1)
           ROLLBACK TRANSACTION
       END
```

Nazwa triggera: TR_Check_Seafood

Opis: Trigger odpowiadający za sprawdzenie czy zamawiany jest owoc morza. Jeśli tak, sprawdza wymagane warunki dla zamawiania dań z owocami morza.

Kod generujący trigger:

```
CREATE TRIGGER TR_Check_Seafood
   ON Order_Details
FOR INSERT
AS
   IF EXISTS(
       SELECT *
       FROM INSERTED I
           INNER JOIN Orders O ON I.Order_ID = O.Order_ID
           INNER JOIN Products P ON P.Product_ID = I.Product_ID
           INNER JOIN Products_Ingredients PI on P.Product_ID = PI.Product_ID
           INNER JOIN Ingredients I2 on PI.Ingredient_ID = I2.Ingredient_ID
           INNER JOIN Ingredient_Categories IC on IC.Category_ID =
I2.Category_ID
       WHERE IC.Name = 'Seafood' AND (
               DATEPART(week, ISNULL(O.Pref_Date, GETDATE())) > DATEPART(week,
O.Order_Date) OR
               DATEPART(week, ISNULL(O.Pref_Date, GETDATE())) = DATEPART(week,
O.Order_Date) AND DATENAME(dw, O.Order_Date) != 'Monday'
           ))
       BEGIN
          RAISERROR ('Cannot order seafood for the closest friday', 1, 1)
           ROLLBACK TRANSACTION
       END
```

Nazwa triggera: TR_Restrictions_Dates

Opis: Trigger sprawdzający, czy nowo dodawane obostrzenia nie nachodzą, na już znajdujące się w tabeli (przy założeniu, że obostrzenie z datą końcową NULL oznacza, że obostrzenie nie ma końca)

Kod generujący trigger:

```
CREATE TRIGGER TR_Check_Restrictions_Dates
ON Tables_Restrictions
FOR INSERT, UPDATE
AS
BEGIN
DECLARE @start datetime
DECLARE @stop datetime
DECLARE @table int
DECLARE @new_cursor CURSOR
SET @new_cursor = CURSOR FOR
SELECT Start_Date, End_Date, Table_ID FROM inserted
OPEN @new_cursor
FETCH NEXT FROM @new_cursor INTO @start, @stop, @table
WHILE @@FETCH_STATUS = 0
    BEGIN
        IF EXISTS(
             SELECT * FROM Tables_Restrictions WHERE
                 Table_ID = @table AND
                 ((@start > Start_Date AND (End_Date IS NULL OR End_Date >
@start)) OR
                  (@stop > Start_Date AND (End_Date IS NULL OR End_Date >
@stop))
                 ))
            BEGIN
                RAISERROR ('New restrictions overlap existing one', 1, 1)
                ROLLBACK TRANSACTION
            end
        FETCH NEXT FROM @new_cursor INTO @start, @stop, @table
    END
END
```

Nazwa triggera: TR_Check_Company_Restricted_Tables

Opis: Trigger sprawdzający, czy nie został zarezerwowany w firmowym zamówieniu stolik, który jest oznaczony jako zablokowany przez obostrzenie.

Kod generujący trigger:

```
CREATE TRIGGER TR_Check_Company_Restricted_Tables
ON Company_Reservation_Tables
FOR INSERT, UPDATE AS
BEGIN
   IF EXISTS(
       SELECT * FROM Tables_Restrictions TR
           INNER JOIN INSERTED I ON TR.Table_ID = I.Table_ID
           INNER JOIN Company_Reservations CR ON I.Company_Reservation_ID =
CR.Company_Reservation_ID
           WHERE TR.Seats = 0 AND (
                   (CR.Start_Date > TR.Start_Date AND (TR.End_Date IS NULL OR
TR.End_Date > TR.Start_Date)) OR
                   (CR.End_Date > TR.End_Date AND (TR.End_Date IS NULL OR
TR.End_Date > CR.End_Date))
               ))
       BEGIN
           RAISERROR ('Chosen table is marked as restricted', 1, 1)
           ROLLBACK TRANSACTION
       END
end
```

Nazwa triggera: TR_Check_Individual_Restricted_Tables

Opis: Trigger sprawdzający, czy nie został zarezerwowany w indywidualnym zamówieniu stolik, który jest oznaczony jako zablokowany przez obostrzenie.

Kod generujący trigger:

```
CREATE TRIGGER TR_Check_Individual_Restricted_Tables
ON Individual_Reservations
FOR INSERT, UPDATE AS
BEGIN
   IF EXISTS(
       SELECT * FROM Tables_Restrictions TR
               INNER JOIN INSERTED I ON I.Table_ID = TR.Table_ID
               WHERE TR.Seats = 0 AND
               ((I.Start_Date > TR.Start_Date AND (TR.End_Date IS NULL OR
TR.End_Date > I.Start_Date)) OR
                (I.End_Date > TR.Start_Date AND (TR.End_Date IS NULL OR
TR.End_Date > I.End_Date)))
       )
       BEGIN
           RAISERROR ('Chosen table is marked as restricted', 1, 1)
           ROLLBACK TRANSACTION
       END
END
```

Nazwa triggera: TR_Add_New_Discount

Opis: Trigger odpowiadający za sprawdzenie czy klient zasługuje na nową zniżkę.

Kod generujący trigger:

```
CREATE TRIGGER TR_Add_New_Discount
ON Orders
AFTER INSERT, UPDATE
AS
DECLARE @client_id int
DECLARE client_cursor CURSOR FOR
SELECT I.Client_ID FROM inserted I
   INNER JOIN Individual_Clients IC ON IC.Client_ID = I.Client_ID
OPEN client_cursor
FETCH NEXT FROM client_cursor INTO @client_id
WHILE @@FETCH_STATUS = 0
    BEGIN
       EXEC dbo.CHECK_INDIVIDUAL_DISCOUNT @client_id;
       FETCH NEXT FROM client_cursor INTO @client_id
    END
```

Nazwa triggera: TR_Check_Local_Table

Opis: Trigger odpowiadający za sprawdzenie czy stolik który został zarezerwowany znajduje się w tym samym lokalu, w którym zostało złożone zamówienie

Kod generujący trigger:

```
CREATE TRIGGER TR_CHECK_LOCAL_TABLE
ON Individual_Reservations
FOR INSERT, UPDATE AS
BEGIN
  IF EXISTS(
      SELECT * FROM INSERTED I
           INNER JOIN Tables T ON T.Table_ID = I.Table_ID
           INNER JOIN Orders O on I.Order_ID = O.Order_ID
      WHERE T.Local_ID != O.Local_ID
      )
       BEGIN
          RAISERROR ('Table is located in the different local than the order!',
1, 1)
          ROLLBACK TRANSACTION
       END
END
```

Nazwa triggera: TR_Check_Order_Menu

Opis: Trigger sprawdzający, czy zamawiany produkt znajduje się aktualnie w menu

Kod generujący trigger:

```
CREATE TRIGGER TR_CHECK_ORDER_MENU
ON Order_Details
FOR INSERT, UPDATE AS
BEGIN
   IF EXISTS(
       SELECT * FROM INSERTED I1
       WHERE NOT EXISTS(
           SELECT * FROM Menu M
               INNER JOIN Orders O ON O.Local_ID = M.Local_ID
           WHERE I1.Product_ID = M.Product_ID AND O.Order_ID = I1.Order_ID AND
                 M.Date_Removed IS NULL OR M.Date_Removed > O.Order_Date AND
                 M.Date_Added >= O.Order_Date
       ))
       BEGIN
           RAISERROR ('Ordered item is not available in the menu', 1, 1)
           ROLLBACK TRANSACTION
       END
END
```

Nazwa triggera: TR_Guard_Individual_ID

Opis: Trigger sprawdzający, czy ID klienta dodawane do tablicy Individual_Clients nie nachodzi ID dodane wcześniej do tablicy Companies.

Kod generujący trigger:

```
CREATE TRIGGER TR_Guard_Individual_ID ON Individual_Clients
FOR INSERT, UPDATE AS
BEGIN
  IF EXISTS(
      SELECT * FROM INSERTED I
           INNER JOIN Companies C ON C.Client_ID = I.Client_ID
      )
       BEGIN
          RAISERROR('Client ID is already used in Company clients', 1, 1)
           ROLLBACK TRANSACTION
       END
END
```

Nazwa triggera: TR_Guard_Company_ID

Opis: Trigger sprawdzający, czy ID klienta dodawane do tablicy Companies nie nachodzi ID dodane wcześniej do tablicy Individual_Clients

Kod generujący trigger:

```
CREATE TRIGGER TR_Guard_Company_ID ON Companies
FOR INSERT, UPDATE AS
BEGIN
  IF EXISTS(
      SELECT * FROM INSERTED I
           INNER JOIN Individual_Clients IC ON I.Client_ID = IC.Client_ID)
       BEGIN
           RAISERROR('Client ID is already used in Individual clients', 1, 1)
           ROLLBACK TRANSACTION
       END
END
```

INDEKSY

Indeksy są tworzone automatycznie dla kluczy głównych i unikalnych. Oprócz tych indeksów były dodane następujące indeksy dla kluczy obcych:

```
create index FK_Companies_City_ID
   ON Companies(City_ID)

create index FK_CompanyReservations_Company_ID
   ON Company_Reservations(Company_ID)

create index FK_Tables_Local_ID
   ON Tables(Local_ID)

create index FK_TableRestrictions_Table_ID
   ON Tables_Restrictions(Table_ID)

create index FK_IndividualReservations_Table_Order_ID
   ON Individual_Reservations(Order_ID)

create index FK_Locals_City_ID
   ON Locals(City_ID)

create index FK_Company_Discounts_Client_ID
   ON Company_Discounts(Client_ID)

create index FK_Individual_Discounts_Client_ID
   ON Individual_Discounts(Client_ID)

create index FK_Orders_Client_ID
   ON Orders(Client_ID)

create index FK_Individual_Clients_City_ID
   ON Individual_Clients(City_ID)

create index FK_Ingredients_Category_ID
   on Ingredients(Category_ID)

create index FK_Local_Ingredients_Supplier_ID
   on Local_Ingredients(Supplier_ID)

create index FK_Suppliers_City_ID
   on Suppliers(City_ID)
```

RAPORTY

Nazwa raportu: get_Month_Report_For_Menu

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Miesięczny dla Menu

Kod generujący raport:

```
create function get_Month_Report_For_Menu (@Local_ID int)
returns table
return
   select Menu.Product_ID, P.Name as 'Product Name',PC.Category_ID,
PC.Name as 'Category Name', Menu.Unit_Price,PC.Description
   from Menu
   inner join Products P on Menu.Product_ID = P.Product_ID
   inner join Product_Categories PC on P.Category_ID = PC.Category_ID
   where Menu.Local_ID= @Local_ID and (DATEDIFF(day,Menu.Date_Added,
getdate()) <=30 and DATEDIFF(day,Menu.Date_Added, getdate()) >=0)
```

Nazwa raportu: get_Month_Report_For_Discounts

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Miesięczny dla Zniżki

Kod generujący raport:

```
create function get_Month_Report_For_Discounts()
returns table
return
select 'INDIVIDUAL CLIENT' as Type, ID.Client_ID, DP.Param_ID,
ID.Start_Date ,ID.End_Date, ID.multiplier, DP.Value
from Individual_Discounts ID
inner join Discount_Params DP on DP.Param_ID = ID.Param_ID
where (DATEDIFF(day,ID.Start_Date, getdate()) <=30 and
DATEDIFF(day,ID.Start_Date, getdate()) >=0) and ID.multiplier>0
union all
select 'COMPANY' ,CD.Client_ID, D.Param_ID, CD.Start_Date, NULL,
CD.multiplier, D.Value
from Company_Discounts CD
inner join Discount_Params D on D.Param_ID = CD.Param_ID
where (DATEDIFF(day,CD.Start_Date, getdate()) <=30 and
DATEDIFF(day,CD.Start_Date, getdate()) >=0) and CD.multiplier>0
```

Nazwa raportu: get_Month_Report_For_Orders_Individual

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Miesięczny dla Zamówień Indywidualnych

Kod generujący raport:

```
create function get_Month_Report_For_Orders_Individual(@Local_ID
int)
returns table
return
select IC.Client_ID, IC.Firstname, IC.Lastname, VOV.Order_ID,
VOV.Value,VOV.Order_Date
from Clients
inner join Individual_Clients IC on Clients.Client_ID =
IC.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where VOV.Local_ID=@Local_ID and (DATEDIFF(day,VOV.Order_Date,
getdate()) <=30 and DATEDIFF(day,VOV.Order_Date, getdate()) >=0)
```

Nazwa raportu: get_Month_Report_For_Orders_Firms

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Miesięczny dla Zamówień Firmowych

Kod generujący raport:

```
create function get_Month_Report_For_Orders_Firms(@Local_ID int)
returns table
return
select Clients.Client_ID, C.Name,C.NIP, VOV.Order_ID, VOV.Value,
VOV.Order_Date
from Clients
inner join Companies C on Clients.Client_ID = C.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where VOV.Local_ID=@Local_ID and (DATEDIFF(day,VOV.Order_Date,
getdate()) <=30 and DATEDIFF(day,VOV.Order_Date, getdate()) >=0)
```

Nazwa raportu: get_Week_Report_For_Menu

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Tygodniowy dla Menu

Kod generujący raport:

```
create function get_Week_Report_For_Menu (@Local_ID int)
returns table
return
   select Menu.Product_ID, P.Name as 'Product Name',PC.Category_ID,
PC.Name as 'Category Name', Menu.Unit_Price,PC.Description
   from Menu
   inner join Products P on Menu.Product_ID = P.Product_ID
   inner join Product_Categories PC on P.Category_ID = PC.Category_ID
   where Menu.Local_ID= @Local_ID and (DATEDIFF(day,Menu.Date_Added,
getdate()) <=7 and DATEDIFF(day,Menu.Date_Added, getdate()) >=0)
```

Nazwa raportu: get_Week_Report_For_Discounts

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Tygodniowy dla Zniżek

Kod generujący raport:

```
create function get_Week_Report_For_Discounts()
returns table
return
select 'INDIVIDUAL CLIENT' as Type, ID.Client_ID, DP.Param_ID,
ID.Start_Date ,ID.End_Date, ID.multiplier, DP.Value
from Individual_Discounts ID
inner join Discount_Params DP on DP.Param_ID = ID.Param_ID
where (DATEDIFF(day,ID.Start_Date, getdate()) <=30 and
DATEDIFF(day,ID.Start_Date, getdate()) >=0) and ID.multiplier>0
union all
select 'COMPANY' ,CD.Client_ID, D.Param_ID, CD.Start_Date, NULL,
CD.multiplier, D.Value
from Company_Discounts CD
inner join Discount_Params D on D.Param_ID = CD.Param_ID
where (DATEDIFF(day,CD.Start_Date, getdate()) <=7 and
DATEDIFF(day,CD.Start_Date, getdate()) >=0)and CD.multiplier>0
```

Nazwa raportu: get_Week_Report_For_Orders_Individual

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Tygodniowy dla Zamówień dla klientów Indywidualnych

Kod generujący raport:

```
create function get_Week_Report_For_Orders_Individual(@Local_ID
int)
returns table
return
select IC.Client_ID, IC.Firstname, IC.Lastname, VOV.Order_ID,
VOV.Value, VOV.Order_Date
from Clients
inner join Individual_Clients IC on Clients.Client_ID =
IC.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where VOV.Local_ID=@Local_ID and (DATEDIFF(day,VOV.Order_Date,
getdate()) <=7 and DATEDIFF(day,VOV.Order_Date, getdate()) >=0)
```

Nazwa raportu: get_Week_Report_For_Orders_Firms

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Tygodniowy dla Zamówień dla klientów Firmowych

Kod generujący raport:

```
create function get_Week_Report_For_Orders_Firms(@Local_ID int)
returns table
return
select Clients.Client_ID, C.Name,C.NIP, VOV.Order_ID, VOV.Value,
VOV.Order_Date
from Clients
inner join Companies C on Clients.Client_ID = C.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where VOV.Local_ID=@Local_ID and (DATEDIFF(day,VOV.Order_Date,
getdate()) <=7 and DATEDIFF(day,VOV.Order_Date, getdate()) >=0)
```

Nazwa raportu: get_All_Orders_Individual

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Zawierający wszystkie zamówienia dla Klienta Indywidualnego

Kod generujący raport:

```
create function get_All_Orders_Individual(@Client_ID int)
returns table
return
select IC.Client_ID, IC.Firstname, IC.Lastname, VOV.Order_ID,
VOV.Value, VOV.Order_Date
from Clients
inner join Individual_Clients IC on Clients.Client_ID =
IC.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where Clients.Client_ID=@Client_ID
```

Nazwa raportu: get_All_Orders_Firm

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Zawierający wszystkie zamówienia dla Klienta Firmowego

Kod generujący raport:

```
create function get_All_Orders_Firm(@Client_ID int)
returns table
return
select Clients.Client_ID, C.Name,C.NIP, VOV.Order_ID, VOV.Value,
VOV.Order_Date
from Clients
inner join Companies C on Clients.Client_ID = C.Client_ID
inner join VW_Order_Value VOV on Clients.Client_ID =
VOV.Client_ID
where Clients.Client_ID=@Client_ID
```

Nazwa raportu: get_All_Report_Discounts

Opis: Funkcja, zwracająca tabelę, reprezentującą Raport Zawierający wszystkie zniżki dla Klientów (Jak indywidualnego, tak i firmowego)

Kod generujący raport:

```
create function get_All_Report_Discounts(@Client_ID int )
returns table
return
select 'INDIVIDUAL CLIENT' as Type, ID.Client_ID, DP.Param_ID,
ID.Start_Date ,ID.End_Date, ID.multiplier, DP.Value
from Individual_Discounts ID
inner join Discount_Params DP on DP.Param_ID = ID.Param_ID
where ID.Client_ID = @Client_ID
union all
select 'COMPANY' ,CD.Client_ID, D.Param_ID, CD.Start_Date, NULL,
CD.multiplier, D.Value
from Company_Discounts CD
inner join Discount_Params D on D.Param_ID = CD.Param_ID
where CD.Client_ID = @Client_ID
```

ROLE

W systemie zdefiniowaliśmy następujące role:

- **Klient** - Klient indywidualny, mający możliwość rezerwacji stolika poprzez formularz internetowy.
    - Add_New_Order
    - Reserve_Individual_Table
- **Pracownicy firmy**
    - Właściciel restauracji - osoba, mająca uprawnienia administratora
    - Księgowy - odpowiada za generowanie raportów, więc posiada wszystkie funkcję, dotyczące za generowanie raportów
    - Kelner
        - Add_New_Order
        - Check_Menu
        - Append_Product_To_Order
        - Check_Individual_Discount
        - VW_Menu_Stockpile
        - Check_Month_Company_Discount
        - VW_IsProductFromSeafood
    - Menedżer magazynu restauracji
        - Add_Product
        - Add_Ingredient
        - VW_Menu_Stockpile
        - Add_Ingredient_To_Product
        - Remove_Ingredient_From_Product
        - Change_Product_Ingredients_Quantity
    - Menedżer zamówień, Menedżer rezerwacji
        - Add_Individual_Client
        - Check_Available_Tables
        - Get_Tables_Capacity
        - Get_Tables_Capacity
        -
        - Reserve_Individual_Table
        - Check_Month_Company_Discount
        - Check_Individual_Discount
        - VW_IsProductFromSeafood
        - Add_Company_Reservation
        - Add_Company
        - Check_Menu
        - Add_New_Order
        - Append_Product_To_Order

    - Reserve_Individual_Table
    - Check_Available_Tables
    - Get_Tables_Capacity

- **Administrator** - dostęp do wszystkich tabel, widoków oraz procedur

Funkcję, realizowane przez system

Dodanie klienta (Pracownik):
Możliwość dodawanie klienta do systemu

Usunięcie klienta (Pracownik):
Możliwość usunięcia klienta z systemu

Dodanie pracownika (Administrator bazy danych)
Możliwość dodania pracownika

Usunięcie pracownika (Administrator bazy danych)
Możliwość usunięcia pracownika

Złożenie zamówienia (Pracownik, firmy i klienci indywidualni):
Możliwość złożenia zamówienia na wynos lub na miejscu, z wyborem preferowanej godziny odbioru.

Potwierdzenie zamówienia (Pracownik):
Potwierdzenie złożonego zamówienia przez pracownika.

Zmiana zamówienia (Pracownik):
Możliwość zmiany szczegółów zamówienia.

Ustawienie liczby dostępnych miejsc (Pracownik):
Możliwość zmiany liczby dostępnych miejsc w lokalu w danym dniu, w związku z obostrzeniami związanymi z COVID-19.

Wystawienie faktury (Pracownik):
Możliwość wystawienia faktury dla zamówienia lub zbiorczej przez pracownika lokalu.

Dodanie pozycji do menu (Pracownik):
Możliwość dodania pozycji do menu w danym dniu.

Usuwanie pozycji w menu (Pracownik):

Możliwość usunięcia pozycji w menu w danym dniu

Rezerwacja stolika (Klient indywidualny, pracownik firmy):

Możliwość zarezerwowania stolika przez klienta

Rezerwacja stolików na firmę (Firma):

Możliwość zarezerwowania dowolnej liczby dostępnych stolików na firmę

Potwierdzenie rezerwacji (Pracownik):

Potwierdzenie przez pracownika rezerwacji internetowej

Anulowanie rezerwacji (Pracownik, Klient):

Możliwość anulowania poprzedniej rezerwacji

Zmiana parametrów rabatów (Administrator):

Możliwość zmiany parametrów rabatów (np. Z1, F2) przez administratora systemu

Generowanie raportów (pracownicy, klienci)

Generowanie miesięcznych, tygodniowych raportów dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia. Dla klientów dodatkowo o kwotach i czasie składania zamówień

GENERATOR

W projekcie do wygenerowania danych na temat firm czy miast został wykorzystany SQL Data Generator firmy RedGate. Program sprawdził się przy generowaniu danych statycznych, jednak nie był w stanie stworzyć rekordów dla tabel zawierających bardziej złożone relacje, dla których został napisany prosty generator w języku Python, z wykorzystaniem biblioteki pyodbc do połączenia się z bazą danych.