AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

PODSTAWY BAZ DANYCH
PROJEKT

KATEDRA INFORMATYKI WIET

Tomasz Boroń, Michał Skorek

3 semestr studiów stacjonarnych
Kierunek Informatyka
Rok akademicki 2020/2021

Spis treści

1. Opis projektu

a. Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. System winien być możliwy do równoległego użytkowania przez kilka tego typu firm.

b. Ogólne informacje

W ofercie jest żywność (np. ciastka, lunch, drobne przekąski) oraz napoje bezalkoholowe (np. kawa, koktajle, woda). Usługi świadczone są na miejscu oraz na wynos. Zamówienie na wynos może być zlecone na miejscu lub z wyprzedzeniem (z wykorzystaniem formularza WWW i wyboru preferowanej daty i godziny odbioru zamówienia). Firma dysponuje ograniczoną liczbą stolików, w tym miejsc siedzących. Istnieje możliwość wcześniejszej rezerwacji stolika dla co najmniej dwóch osób.

Z uwagi na zmieniające się ograniczenia związane z COVID-19, w poszczególnych dniach może być dostępna ograniczona liczba miejsc (w odniesieniu do powierzchni lokalu), zmienna w czasie.

Klientami są osoby indywidualne oraz firmy, odbierające większe ilości posiłków w porze lunchu lub jako catering (bez dostawy). Dla firm istnieje możliwość wystawienia faktury dla danego zamówienia lub faktury zbiorczej raz na miesiąc.

c. Menu

Menu ustalane jest co najmniej dziennym wyprzedzeniem. W firmie panuje zasada, że co najmniej połowa pozycji menu zmieniana jest co najmniej raz na dwa tygodnie, przy czym pozycja zdjęta może powtórzyć się nie wcześniej niż za 1 miesiąc.

Ponadto, w dniach czwartek-piątek-sobota istnieje możliwość wcześniejszego zamówienia dań zawierających owoce morza. Z uwagi na indywidualny import takie zamówienie winno być złożone co maksymalnie do poniedziałku poprzedzającego zamówienie. Istnieje możliwość, że pozycja w menu zostanie usunięta na skutek wyczerpania się półproduktów.

d. Wcześniejsza rezerwacja zamówienia/stolika

Internetowy formularz umożliwia klientowi indywidualnemu rezerwację stolika, przy jednoczesnym złożeniu zamówienia, z opcją płatności przed lub po zamówieniu, przy minimalnej wartości zamówienia 50 zł, w przypadku klientów, którzy dokonali wcześniej co najmniej 5 zamówień i/lub mniej, ale w tym przypadku na kwotę co najmniej 200 zł.

Informacja wraz z potwierdzeniem zamówienia oraz wskazaniem stolika. Wysyłana jest po akceptacji przez obsługę.

Internetowy formularz umożliwia także rezerwację stolików dla firm, w dwóch opcjach: rezerwacji stolików na firmę i/lub rezerwację stolików dla konkretnych pracowników firmy (imiennie).

e. Rabaty

System umożliwia realizację programów rabatowych dla klientów indywidualnych:

- Po realizacji ustalonej liczby zamówień Z1 (przykładowo Z1 = 10) za co najmniej określoną kwotę K1 (np. K1 = 30 zł każde zamówienie): R1% (np. R1 = 3%) zniżki na wszystkie zamówienia
- Po realizacji kolejnych Z1 zamówień (np. Z1 = 10) za co najmniej określoną kwotę K1 każde (np. K1 = 30 zł): dodatkowe R1% zniżki na wszystkie zamówienia (np. R1 = 3%)
- Po realizacji zamówień za łączną kwotę K2 (np. 1000 zł): jednorazowa zniżka R2% (np. 5%) na zamówienia złożone przez D1 dni (np. D1 = 7), począwszy od dnia przyznania zniżki • Po realizacji zamówień za łączną kwotę K3 (np. K3 = 5000 zł): jednorazowa zniżka R3% (np. R3 = 5%) na zamówienia złożone przez D2 dni (np. D2 = 7), począwszy od dnia przyznania zniżki.

W przypadku firm rabat udzielany jest zgodnie z zasadami:

- Za każdy kolejny miesiąc, w którym dokonano co najmniej FZ zamówień (np. FZ = 5) za łączną kwotę co najmniej FK1 (np. FK1 = 500 zł): rabat FR1% (np. FR1 = 0,1%). W przypadku braku ciągłości w zamówieniach rabat zeruje się. Łączny, maksymalny rabat, to FM% (np. FM = 4%).
- Za każdy kolejny kwartał, w którym dokonano zamówień za łączną kwotę FK2 (np. FK2 = 10000 zł): rabat kwotowy równy FR2% (np. FR2 = 5%) z łącznej kwoty, z którą zrealizowano zamówienie.

f. Raporty

System umożliwia generowanie raportów miesięcznych i tygodniowych, dotyczących rezerwacji stolików, rabatów, menu, a także statystyk zamówienia – dla klientów indywidualnych oraz firm – dotyczących kwot oraz czasu składania zamówień. System umożliwia także generowanie raportów dotyczących zamówień oraz rabatów dla klienta indywidualnego oraz firm.

2. Analiza wymagań

Struktura bazy danych:

1. Klienci
    a. Klientem może być osoba indywidualna bądź firma
    b. Informacje o klientach indywidualnych i firmach są przetrzymywane w innych grupach (Klienci indywidualni oraz klienci biznesowi)
    c. Każdy klient ma przypisany unikalny identyfikator, który jednoznacznie go określa (id_klienta)
    d. Jeżeli klient należy do firmy to rejestrujemy go jako klienta indywidualnego oraz notujemy informację w jakiej firmie pracuje. Zatem trzymamy takich użytkowników w dodatkowym zbiorze, zapisując ich identyfikator klienta jako klienta indywidualnego oraz identyfikator firmy, w której pracuje. Ponadto zawieramy również informację od kiedy pracuje w danej firmie.
    e. W bazie danych klient indywidualny może być identyfikowany, poza unikalnym ID, po imieniu i nazwisku. W celu kontaktu z klientem utrzymujemy również jego numer telefonu.
    f. Szczegóły klientów biznesowych zawierają informacje niezbędne do wystawienia faktury, tj. numer NIP, Nazwę firmy i Adres firmy.
    g. Część wspólna każdego typu klienta to: adres email, numer telefonu. Dane te są zawarte w głównym zbiorze wszystkich klientów.

2. Zamówienia
    a. Zamówienia dzielimy na dania przygotowywane na miejscu oraz na wynos.
    b. Każde zamówienie ma swój unikalny identyfikator, nr klienta składającego zamówienie, datę złożenia zamówienia, datę odbioru zamówienia oraz informacje czy zamówienie ma zostać zrealizowane na miejscu czy jest ono na wynos.
    c. Każde zamówienie składa się z jednego lub więcej dań
        i. Dania mają unikalny dla siebie atrybut pozwalający na ich jednoznaczną identyfikację. Każde danie zawiera jeden lub więcej półproduktów. Jednocześnie jeden półprodukt może być składnikiem wielu dań. Dania posiadają cenę dania.
        ii. Każde danie ma swój Przepis, w którym wyszczególnione są potrzebne półprodukty oraz ilość do wykonania porcji.
    d. Szczegóły zamówień zawierają informację o złożonych zamówieniach, posiadają identyfikator potrawy, ilość zamówionych jednostek danej potrawy, cenę jednostkową dania w chwili jego zamawiania.

3. Menu
    a. Menu jest listą dostępnych dań w danym przedziale czasowym. Notuje również historię dań, które wystąpiły w przeszłości i zostały zdjęte.
    b. Dodatkowo w menu przechowujemy informację kiedy dane danie zostało dodane do menu. Stanowi to podstawę do ustalenia, które pozycje powinny zostać zdjęte z karty w najbliższej kolejności. Szczegóły odnośnie wyboru dań do zdjęcia z karty menu zostały opisane w funkcjach baz danych w punkcie 3.
    c. Każda zdjęta pozycja zawiera również informację o dacie zniknięcia z menu (w celu spełnienia wymagań funkcjonalnych opisanych niżej)
    d. Dodatkowo w czwartek, piątek i sobotę do menu dołączane są dania z kategorii "owoce morza". Z uwagi na indywidualny import dania zawierające owoce morza nie podlegają rotacji i nie są przechowywane w Menu. Każde danie z owocami morza

może zostać zrealizowane jeśli tylko spełnia warunki dotyczące terminu jego złożenia.

4. Półprodukty
    a. Zawiera informację na temat produktu, jego numer identyfikacyjny, nazwę produktu, liczbę dostępnych jednostek półproduktu w magazynie.
    b. Każdy półprodukt na przestrzeni czasu może posiadać wielu dostawców.
    c. Każdy półprodukt może być składnikiem wielu dań.

5. Kategorie dań
    a. Dania podzielone są na kategorie. Każde danie należy do jednej kategorii
    b. Kategoria identyfikowana jest po jej numerze identyfikacyjnym.

6. Aktualnie przyznane rabaty
    a. Zawiera informacje o kliencie i aktualnie posiadanych przez niego rabatach (unikalna para klient – konkretny rabat). Przechowywana jest również informacja o dacie, od której dany rabat obowiązuje i do kiedy jest on ważny.

7. Rabaty
    a. Określa jakie typy rabatów przysługują klientom firmy gastronomicznej oraz ich jednostkową wysokość.
    b. Rabaty dzielone są na cztery grupy – dla klientów indywidualnych (jednorazowe i stałe) i dla klientów biznesowych (miesięczne i kwartalne)
    c. Każda grupa ma swoje warunki niezbędne do jego przyznania.
    d. Część wspólną wszystkich rabatów stanowi minimalna kwota zamówień oraz jego jednostkowa wysokość
    e. Rabaty stałe dla klientów indywidualnych zawierają ponadto minimalną liczbę zamówień niezbędną do przyznania rabatu
    f. Rabaty jednorazowe dla klientów indywidualnych ponadto zawierają informacje o ważności rabatu.
    g. Rabaty firmowe miesięczne ponadto zawierają minimalną liczbę zamówień i maksymalną wysokość rabatu.
    h. Rabaty firmowe kwartalne z racji, że nie posiadają cech charakterystycznych znajdują się w głównym zbiorze rabatów
8. Dostawcy
    a. Każdy półprodukt może mieć wielu różnych dostawców na przestrzeni czasu.
    b. Dostawca jest identyfikowany po jego unikalnym identyfikatorze.
    c. Dodatkowo każdy rekord zawiera informacje takie jak adres dostawcy, dane osoby kontaktowej, numer telefonu i tym podobne informacje

9. Aktualnie Dostarcza
    a. Opisuje listę produktów, którą dostarcza aktualnie dany dostawca. Zawiera numer dostawcy oraz numer produktu.

10. Historia Dostaw
    a. Notowane są wszystkie dostawy, trzymamy informację o dostawcy, dostarczonym półprodukcie, ilości jednostek półproduktu oraz dacie dostawy.

11. Rezerwacje
    a. Rezerwacje podzielone są na trzy typy: składane przez klienta indywidualnego, firmowego oraz na nazwiska pracowników firmy. Podział na takie trzy zbiory pozwala nam jednoznacznie określić, z jakim typem rezerwacji mamy do czynienia.

b. Część wspólna dla każdego typu przechowywana jest w głównym zbiorze rezerwacji, który każdej rezerwacji przypisuje jednoznaczny identyfikator. W tym miejscu znajduje się również informacja o dacie złożenia rezerwacji oraz o dacie na kiedy dana rezerwacja została złożona.
c. Rezerwacje dla klientów indywidualnych posiadają identyfikator zamówienia powiązanego z tą rezerwacją.
d. W przypadku złożenia rezerwacji przez firmę na konkretnych pracowników informacje te są reprezentowane jako unikalne trójki, zawierające numer rezerwacji, identyfikator firmy i pracownika

12. Stoliki
    a. Każdy stolik ma swój unikalny numer identyfikacyjny.
    b. Każdy stolik ma również ustaloną maksymalną liczbę miejsc
    c. Na podstawie obowiązujących ograniczeń związanych z epidemią COVID-19 stolik ma ustaloną maksymalną dostępną liczbę miejsc

13. Obostrzenia
    a. Zawierają informację o stolikach i dostępnej liczbie miejsc przy danym stoliku w czasie od wprowadzenia obostrzenia.
    b. Dodatkowo przechowujemy datę wprowadzenia ograniczenia w celu ustalenia aktualnego obostrzenia i liczby miejsc przy stolikach.
    c. W przypadku gdy w danym okresie nie ma obostrzeń, liczba dostępnych miejsc jest równa maksymalnej liczbie miejsc przy danym stoliku.

Funkcje bazy danych:

1. Klienci
    a. Każdemu klientowi może zostać naliczony rabat (jego szczegóły zostały opisane w punkcie 6)
    b. Dla firm istnieje możliwość wystawienia faktury dla konkretnego zamówienia, bądź zbiorczej raz na miesiąc
    c. Każdy klient ma możliwość zamówienia na miejscu (jeśli jest wolny stolik) lub na wynos (szczegóły w punkcie 2)
    d. Każdy klient ma możliwość zlecenia zamówienia z wyprzedzeniem poprzez stronę WWW (szczegóły w punkcie 2)

2. Zamówienia
    a. Jeden klient może dokonać wielu zamówień.
    b. Zamówienie może zostać złożone stacjonarnie w lokalu bądź z wyprzedzeniem (na wynos, z ustalonym czasem odbioru). Zamówienia na wynos mogą być złożone na oba sposoby, tj. stacjonarnie lub z wyprzedzeniem.
    c. W przypadku zamówienia na miejscu, musi być przynajmniej jeden niezarezerwowany stolik.
    d. Zamówienie można złożyć na dowolną pozycję z aktualnego Menu, w ilości takiej, że będzie możliwe jego realizacja.
    e. Zrealizowanie zamówienia jest możliwe tylko w przypadku, gdy w magazynie jest wystarczająca liczba półproduktów potrzebnych do wykonania zamówienia. W przypadku gdy w ciągu dnia zabraknie półproduktu, zamówienia zawierające potrawy z nim nie powinny zostać przyjmowane lub powinna się pojawić informacja, że nie można zamówić danej potrawy.
    f. Przy zamówieniach dań zawierających owoce morza musimy sprawdzać czy zamówienie zostało złożone odpowiednio wcześniej. Z racji że danie musi być złożone najpóźniej w poniedziałek poprzedzający zamówienie obliczamy poprawność dodając do daty zamówienia 3, 4 lub 5 dni w zależności czy zamówienie zostało złożone kolejno na dzień, który jest czwartkiem, piątkiem czy sobotą. Jeśli po dodaniu odpowiedniej ilości dni osiągnięta data będzie wcześniejsza lub równa

dacie odbioru to zamówienie takie możemy przyjąć do akceptacji. Doliczone dni gwarantują nam poprawność w sytuacji skrajnej, gdy zamówienie zostało złożone najpóźniej, czyli w poniedziałek poprzedzający jego odbiór, zatem w przypadku późniejszego zamówienia na ten sam tydzień zamówienia zostaną odrzucone z informacją, że na ten tydzień nie można już takiego zamówienia złożyć oraz z propozycją złożenia na kolejny tydzień.

g. W momencie potwierdzenia przyjęcia zamówienia automatycznie redukowana jest liczba jednostek półproduktów w magazynie o ilość potrzebną do wykonania wszystkich zamówionych pozycji.
h. Jeśli klient posiada rabat to odliczany jest odpowiedni procent od wartości zamówienia, zgodny z aktualnymi zniżkami posiadanymi przez klienta.
i. Jeśli klient indywidualny należy do firmy posiadającej zniżkę, to zniżka firmowa zostanie doliczona do indywidualnego rabatu klienta.

3. Menu

a. Menu ustalane jest z co najmniej dziennym wyprzedzeniem.
b. Co 14 dni zmieniona zostaje połowa pozycji z menu.
c. Nie można dopuścić pozycji wcześniej niż miesiąc od jej ostatniego zniknięcia z karty. Weryfikacji dokonujemy korzystając z Historii Menu, gdzie posiadamy informacje o dacie zniknięcia pozycji z Menu.
d. W przypadku braku półproduktów potrzebnych do wykonania potrawy, pozycja ta nie może znaleźć się wśród nowych pozycji z karty dań.
e. Dla dań zawierających produkty z kategorii owoce morza można otrzymać takie dania w dniach czwartek-piątek-sobota. Zamówienie takie zostanie potwierdzone pod warunkiem złożenia zamówienia najpóźniej w poniedziałek poprzedzający datę odbioru zamówienia.
f. Dana pozycja może zniknąć z menu na skutek wyczerpania półproduktów.
g. Z Menu w trakcie rotacji znikają pozycje, które znajdują się w nim najdłużej.

4. Dostawy

a. W przypadku, gdy stan magazynowy danego półprodukty jest zerowy, dania zawierające dany półprodukt nie mogą znajdować się w menu, do momentu gdy dany półprodukt nie zostanie ponownie uzupełniony. Istnieje możliwość wygenerowania listy brakujących rzeczy, z podanymi informacjami o dostawcy, w celu ewentualnego zamówienia danej rzeczy.
b. Dany półprodukt może zostać zamówiony od dowolnego dostawcy, od którego mamy deklarację, że go aktualnie dostarcza (którą przechowujemy w Aktualnie Dostarcza punkt 8 - struktura bazy danych).
c. W momencie odebrania dostawy notowana ona jest w historii oraz zwiększany jest stan magazynowy danego półproduktu o dostarczoną ilość jednostek.
d. Każdego wtorku realizowane jest zamówienie na półprodukty dań z kategorii "owoce morza". Zamawiane produkty są zależne od zleconych zamówień klientów na dany tydzień (czwartek, piątek, sobota, czyli wtedy, kiedy są one dostępne).

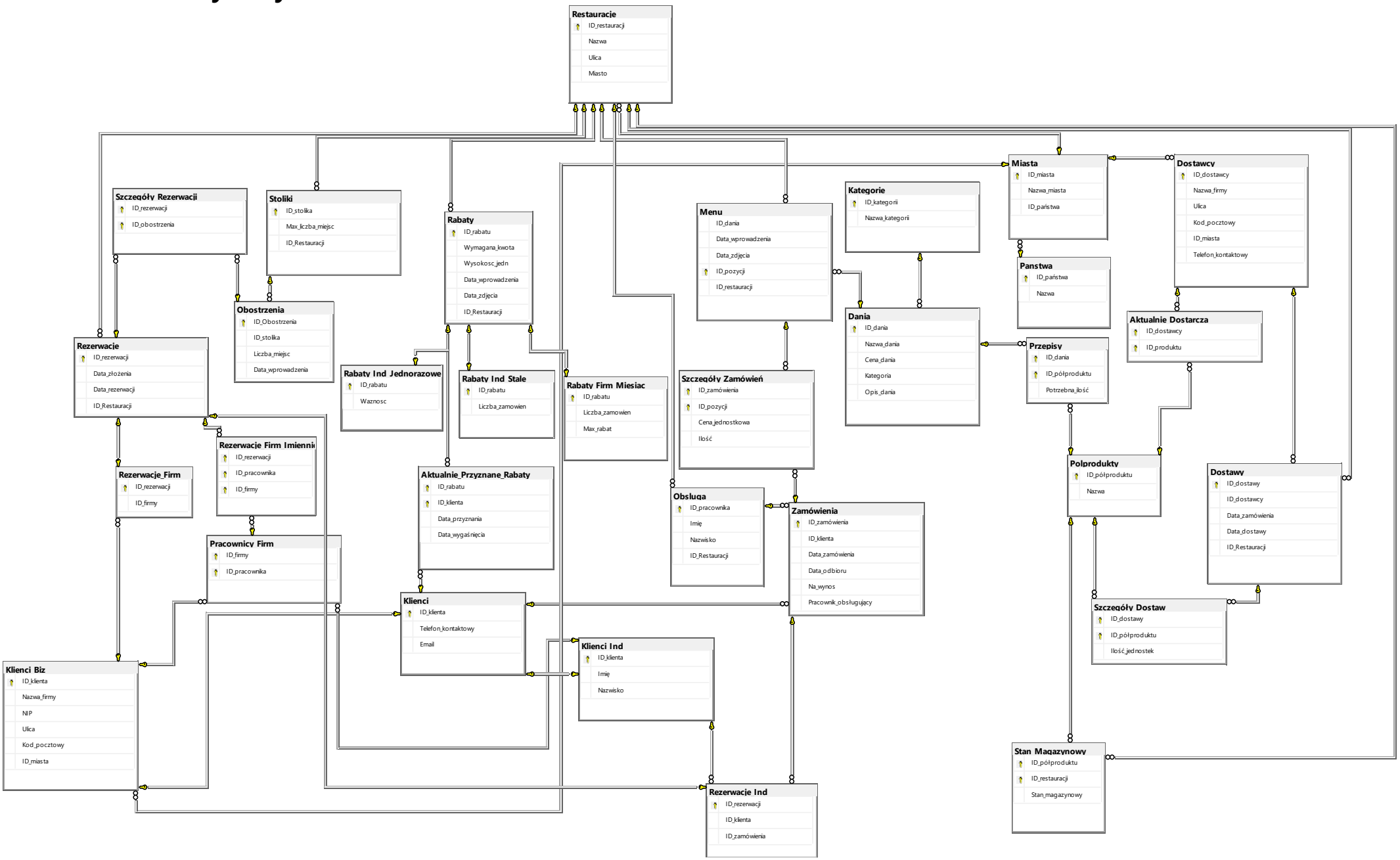
5. Rabaty

a. Ciągłość rabatu sprawdzamy na podstawie daty przyznania rabatu.
b. W przypadku niespełnienia warunku ciągłości rabat jest usuwany.
c. Dla klientów indywidualnych aktualne rabaty sprawdzane oraz ewentualnie naliczane są po każdym zamówieniu.
d. W przypadku firm, na podstawie daty przyznania rabatu wyznaczamy aktualny procent zniżki.
e. Klient indywidualny po przekroczeniu odpowiedniej ilości zamówień za minimalną daną kwotę otrzymuje rabat. Po dokonaniu kolejnej liczby zamówień rabat zostaje powiększony o kolejny procent.
f. Klient indywidualny może również otrzymać rabat jednorazowy, do wykorzystania przez odpowiednią ilość dni. Po wygaśnięciu rabatu jest on kasowany.

6. Raporty
    a. Możliwość generowania raportów podsumowujących działalność firmy
    b. Raport może zawierać podsumowanie finansowe przedsiębiorstwa z danego tygodnia/miesiąca i/lub statystyki opisane poniżej:
        i. Statystyka zamówień zawiera listę najczęściej zamawianych przez klientów dań (wskazówka sugerująca, co warto pozostawić w menu)
        ii. Statystyka zamówień jak wyżej, lecz dla konkretnego klienta
        iii. Statystyka stolików zawiera listę najczęściej rezerwowanych stolików przez klientów
        iv. Statystyka rabatów zawiera informację, jak często klienci korzystają z danego rabatu, oraz jaką kwotę łącznie zaoszczędzili dzięki niemu (indywidualny dla klienta lub zbiorczy dla wszystkich)
        v. Statystyka rabatów jak wyżej, lecz dla konkretnego klienta
        vi. Istnieje również możliwość wygenerowanego raportu dla klienta biznesowego, który zawiera statystykę czasów składania zamówień oraz kwot zamówień firmy z danego tygodnia/miesiąca
7. Rezerwacje
    a. Każdy klient ma możliwość rezerwacji stolika
    b. Stolik może zostać zarezerwowany dla minimum dwóch osób.
    c. W przypadku rezerwacji dla klienta indywidualnego, rezerwacja posiada informację identyfikującą jednoznacznie klienta, który rezerwacji dokonał.
    d. Liczba dostępnych miejsc jest ustalana na podstawie znanej, ustalonej powierzchni lokalu i obowiązujących w danym dniu ograniczeń związanych z COVID-19 (minimalna powierzchnia na jedną osobę)
    e. Procedura składania rezerwacji:
        a. Klient deklaruje w rezerwacji wymaganą liczbę miejsc przy stoliku
        b. Akceptacja rezerwacji jest obsługiwana przez lokal
        c. Klient ma możliwość wybrania przy rezerwacji płatności przed lub po zamówieniu
        d. Lokal w rezerwacji otrzymuje informację, czy klient (indywidualny) dokonał wcześniej co najmniej 5 zamówień (jest to potrzebna informacja, żeby ustalić minimalną kwotę zamówienia 50 zł lub 200 zł przy rezerwacji aby rezerwacja została zaakceptowana)
        e. Po akceptacji rezerwacji klient otrzymuje identyfikację zwrotną, ze wskazaniem przydzielonych stolików

8. Obostrzenia
    a. W momencie wprowadzenia dodatkowych obostrzeń, należy zmienić aktualną dostępną liczbę miejsc przy każdym stoliku. W celu ustalenia maksymalnej liczby osób przy określonym stoliku w danym dniu, wybierane jest obostrzenie o najpóźniejszej dacie przed danym dniem.
    b. W przypadku zniesienia obostrzeń, przy każdym stoliku jest dostępna maksymalna liczba miejsc przypisana do niego.
    c. Jeśli ograniczenie powoduje obniżenie liczby dostępnych miejsc sprawdzamy czy wszystkie rezerwacje mogą się odbyć, czyli czy sumaryczna liczba zarezerwowanych miejsc jest większa niż liczba dostępnych. W przypadku gdy nie możemy jednak przyjąć wszystkich rezerwacji, informujemy klientów, którzy dokonali rezerwacji najpóźniej, że rezerwacja nie może się odbyć w danym terminie. Postępujemy tak do momentu gdy zejdziemy poniżej dostępnej liczby miejsc na sali.

3. Schemat bazy danych



4. Tabele i warunki integralności

Nazwa pola - pogrubiona

Typ danych - w nawiasie obok nazwy pol

1. **Tabela Klienci -** reprezentuje zbiór wszystkich klientów w bazie danych. Zawiera informację o:
    a. **ID_klienta (int),** które stanowi klucz główny jednoznacznie identyfikujący klienta
    b. **Telefon_kontaktowy (varchar),** numer telefonu klienta
    c. **Email (varchar),** będący adresem e-mail klienta

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

```
CREATE TABLE [dbo].[Klienci](
	[ID_klienta] [int] IDENTITY(1,1) NOT NULL,
	[Telefon_kontaktowy] [varchar](9) NOT NULL,
	[Email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Klienci] PRIMARY KEY CLUSTERED
(
	[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Un_Klienci_Email] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Klienci]  WITH CHECK ADD  CONSTRAINT [CK_Klienci_Email] CHECK  (([Email] like '%@%.%'))
GO

ALTER TABLE [dbo].[Klienci] CHECK CONSTRAINT [CK_Klienci_Email]
GO

ALTER TABLE [dbo].[Klienci]  WITH CHECK ADD  CONSTRAINT [CK_Klienci_Telefon] CHECK  (([Telefon_kontaktowy] like '[0-
9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Klienci] CHECK CONSTRAINT [CK_Klienci_Telefon]
GO
```

2. **Tabela Klienci_Ind -** reprezentuje klientów indywidualnych w bazie danych. Zawiera informację o:
    a. **ID_klienta (int),** które jest kluczem głównym
    b. **Imię (varchar),** czyli imię klienta
    c. **Nazwisko (varchar),** czyli nazwisko konkretnego klienta

W imieniu oraz nazwisku nie mogą występować cyfry.

```
CREATE TABLE [dbo].[Klienci_Ind](
	[ID_klienta] [int] NOT NULL,
```

```
	[Imię] [varchar](30) NOT NULL,
	[Nazwisko] [varchar](30) NOT NULL,
 CONSTRAINT [PK_Klienci_Ind] PRIMARY KEY CLUSTERED
(
	[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Klienci_Ind]  WITH CHECK ADD  CONSTRAINT [FK_Klienci_Ind_Klienci1] FOREIGN
KEY([ID_klienta])
REFERENCES [dbo].[Klienci] ([ID_klienta])
GO

ALTER TABLE [dbo].[Klienci_Ind] CHECK CONSTRAINT [FK_Klienci_Ind_Klienci1]
GO

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

ALTER TABLE [dbo].[Klienci_Ind] CHECK CONSTRAINT [CK_Klienci_Ind_Imie]
GO

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

ALTER TABLE [dbo].[Klienci_Ind] CHECK CONSTRAINT [CK_Klienci_Ind_Nazw]
GO
```

3. **Tabela Klienci_Biz -** reprezentuje klientów firmowych w bazie danych. Zawiera informację o:
    a. **ID_klienta (int),** stanowiące klucz główny
    b. **Nazwa_firmy (varchar),** będące nazwą firmy
    c. Numer **NIP (varchar)**
    d. Adres, na który składa się: **Ulica (varchar), Kod_pocztowy (varchar)** oraz **ID_miasta (int),** które jest kluczem obcym określającym miasto, w którym znajduje się firma

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

```
CREATE TABLE [dbo].[Klienci_Biz](
	[ID_klienta] [int] NOT NULL,
	[Nazwa_firmy] [varchar](50) NOT NULL,
	[NIP] [varchar](10) NOT NULL,
	[Ulica] [varchar](50) NOT NULL,
	[Kod_pocztowy] [varchar](6) NOT NULL,
	[ID_miasta] [int] NOT NULL,
 CONSTRAINT [PK_Klienci_Biz] PRIMARY KEY CLUSTERED
(
	[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Un_Klienci_Biz_NIP] UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Klienci_Biz]  WITH CHECK ADD  CONSTRAINT [FK_Klienci_Biz_Klienci] FOREIGN
KEY([ID_klienta])
REFERENCES [dbo].[Klienci] ([ID_klienta])
GO

ALTER TABLE [dbo].[Klienci_Biz] CHECK CONSTRAINT [FK_Klienci_Biz_Klienci]
GO

ALTER TABLE [dbo].[Klienci_Biz]  WITH CHECK ADD  CONSTRAINT [FK_Klienci_Biz_Miasta] FOREIGN
KEY([ID_miasta])
REFERENCES [dbo].[Miasta] ([ID_miasta])
GO

ALTER TABLE [dbo].[Klienci_Biz] CHECK CONSTRAINT [FK_Klienci_Biz_Miasta]
GO

ALTER TABLE [dbo].[Klienci_Biz]  WITH CHECK ADD  CONSTRAINT [CK_Klienci_Biz_Kod_pocztowy] CHECK
(([Kod_pocztowy] like '[0-9][0-9][0-9][0-9][0-9]' OR [Kod_pocztowy] like '[0-9][0-9]-[0-9][0-9][0-
9]'))
GO

ALTER TABLE [dbo].[Klienci_Biz] CHECK CONSTRAINT [CK_Klienci_Biz_Kod_pocztowy]
GO

ALTER TABLE [dbo].[Klienci_Biz]  WITH CHECK ADD  CONSTRAINT [CK_Klienci_Biz_NIP] CHECK  (([NIP] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Klienci_Biz] CHECK CONSTRAINT [CK_Klienci_Biz_NIP]
GO

ALTER TABLE [dbo].[Klienci_Biz]  WITH CHECK ADD  CONSTRAINT [CK_Klienci_Biz_Ulica] CHECK  (([Ulica]
like '%[0-9][a-z]' OR [Ulica] like '%[0-9]'))
GO

ALTER TABLE [dbo].[Klienci_Biz] CHECK CONSTRAINT [CK_Klienci_Biz_Ulica]
GO
```

4. **Tabela Pracownicy_firm -** zawiera informację o aktualnym zatrudnieniu pracowników (klientów indywidualnych) w firmach (klientach biznesowych).
    a. Klucz główny stanowi para **ID_firmy (int), ID_pracownika (int)**
    b. **ID_firmy** jest kluczem obcym oznaczającym firmę
    c. **ID_pracownika** jest kluczem obcym oznaczającym osobę - pracownika firmy

```
CREATE TABLE [dbo].[Pracownicy_Firm](
	[ID_firmy] [int] NOT NULL,
	[ID_pracownika] [int] NOT NULL,
 CONSTRAINT [PK_Pracownicy_Firm] PRIMARY KEY CLUSTERED
(
	[ID_firmy] ASC,
	[ID_pracownika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Pracownicy_Firm]  WITH CHECK ADD  CONSTRAINT [FK_Pracownicy_Firm_Klienci_Biz]
FOREIGN KEY([ID_firmy])
REFERENCES [dbo].[Klienci_Biz] ([ID_klienta])
GO

ALTER TABLE [dbo].[Pracownicy_Firm] CHECK CONSTRAINT [FK_Pracownicy_Firm_Klienci_Biz]
GO
```

```
ALTER TABLE [dbo].[Pracownicy_Firm]  WITH CHECK ADD  CONSTRAINT [FK_Pracownicy_Firm_Klienci_Ind]
FOREIGN KEY([ID_pracownika])
REFERENCES [dbo].[Klienci_Ind] ([ID_klienta])
GO

ALTER TABLE [dbo].[Pracownicy_Firm] CHECK CONSTRAINT [FK_Pracownicy_Firm_Klienci_Ind]
GO
```

5. **Tabela Rezerwacje -** tabela reprezentuje wszystkie złożone rezerwacje. Posiada informację o:
    a. **ID_rezerwacji (int),** które jest kluczem głównym
    b. Dacie złożenia rezerwacji przez klienta **Data_złożenia (date)** oraz dacie, na kiedy dana rezerwacja została złożona **Data_rezerwacji (date)**
    c. **ID_Restauracji (int),** które jest kluczem obcym oznaczającym restaurację, do której została złożona rezerwacja.

Data złożenia ma domyślnie wartość getDate()

```
CREATE TABLE [dbo].[Rezerwacje](
	[ID_rezerwacji] [INT] IDENTITY(1,1) NOT NULL,
	[Data_złożenia] [DATETIME] NOT NULL,
	[Data_rezerwacji] [DATETIME] NOT NULL,
	[ID_Restauracji] [INT] NOT NULL,
 CONSTRAINT [PK_Rezerwacje] PRIMARY KEY CLUSTERED
(
	[ID_rezerwacji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Rezerwacje]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Restauracje] FOREIGN
KEY([ID_Restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

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

6. **Tabela Stoliki -** reprezentuje zbiór informacji dotyczących stolików w restauracjach.
    a. **ID_stolika (int)** stanowi klucz główny, potrzebny do identyfikacji stolików.
    b. **Max_liczba_miejsc (int)** określa liczbę miejsc dostępnych przy stoliku, przy założeniu, że na dany stolik nie byłoby narzucone obostrzenie.
    c. **ID_Restauracji (int)** stanowi klucz obcy określający lokal, który jest właścicielem stolika

Maksymalna liczba miejsc przy stoliku musi być dodatnia.

```
CREATE TABLE [dbo].[Stoliki](
	[ID_stolika] [int] IDENTITY(1,1) NOT NULL,
	[Max_liczba_miejsc] [int] NOT NULL,
	[ID_Restauracji] [int] NOT NULL,
 CONSTRAINT [PK_Stoliki] PRIMARY KEY CLUSTERED
(
	[ID_stolika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Stoliki]  WITH CHECK ADD  CONSTRAINT [FK_Stoliki_Restauracje] FOREIGN
KEY([ID_Restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Stoliki] CHECK CONSTRAINT [FK_Stoliki_Restauracje]
GO

ALTER TABLE [dbo].[Stoliki]  WITH CHECK ADD  CONSTRAINT [CK_Stoliki_Liczba_Miejsc] CHECK
(([Max_liczba_miejsc]>(0)))
GO

ALTER TABLE [dbo].[Stoliki] CHECK CONSTRAINT [CK_Stoliki_Liczba_Miejsc]
GO
```

7. **Tabela Obostrzenia -** określa ograniczenia miejsc na stoliki w danym czasie.
    a. **ID_obostrzenia (int)** jest kluczem głównym
    b. **ID_stolika (int)** to klucz obcy, który określa stolik
    c. **Liczba_miejsc (int)** oznacza dostępną aktualnie górną granicę na ilość osób, które mogą siedzieć przy stoliku
    d. Posiada także informację o dacie wprowadzenia każdego obostrzenia, w celu ustalenia aktualnie obowiązującego **Data_wprowadzenia (date)**

    Domyślnie ma wartość getDate()

    Liczba miejsc dostępnych przy stoliku jest nieujemna.

```
CREATE TABLE [dbo].[Obostrzenia](
	[ID_Obostrzenia] [INT] IDENTITY(1,1) NOT NULL,
	[ID_stolika] [INT] NOT NULL,
	[Liczba_miejsc] [INT] NOT NULL,
	[Data_wprowadzenia] [DATE] NOT NULL,
 CONSTRAINT [PK_Obostrzenia_1] PRIMARY KEY CLUSTERED
(
	[ID_Obostrzenia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Obostrzenia] ADD  CONSTRAINT [DF_Obostrzenia_Data_wprowadzenia]  DEFAULT
(GETDATE()) FOR [Data_wprowadzenia]
GO

ALTER TABLE [dbo].[Obostrzenia]  WITH CHECK ADD  CONSTRAINT [FK_Obostrzenia_Stoliki] FOREIGN
KEY([ID_stolika])
REFERENCES [dbo].[Stoliki] ([ID_stolika])
GO

ALTER TABLE [dbo].[Obostrzenia] CHECK CONSTRAINT [FK_Obostrzenia_Stoliki]
GO

ALTER TABLE [dbo].[Obostrzenia]  WITH CHECK ADD  CONSTRAINT [CK_Obostrzenia_Liczba_Miejsc] CHECK
(([Liczba_miejsc]>=(0)))
GO

ALTER TABLE [dbo].[Obostrzenia] CHECK CONSTRAINT [CK_Obostrzenia_Liczba_Miejsc]
GO
```

8. **Tabela Szczegóły_Rezerwacji -** zawiera informację o stoliku, na jaki złożono rezerwację.
    a. Klucz główny stanowi para **ID_rezerwacji (int), ID_obostrzenia (int)**

b. **ID_rezerwacji** to klucz obcy, określający rezerwację
c. **ID_obostrzenia** to klucz obcy, wskazujący na aktualne warunki danego stolika.

```
CREATE TABLE [dbo].[Szczegóły_Rezerwacji](
	[ID_rezerwacji] [INT] NOT NULL,
	[ID_obostrzenia] [INT] NOT NULL,
 CONSTRAINT [PK_Szczegóły_Rezerwacji_1] PRIMARY KEY CLUSTERED
(
	[ID_rezerwacji] ASC,
	[ID_obostrzenia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Szczegóły_Rezerwacji]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegóły_Rezerwacji_Obostrzenia] FOREIGN KEY([ID_obostrzenia])
REFERENCES [dbo].[Obostrzenia] ([ID_Obostrzenia])
GO

ALTER TABLE [dbo].[Szczegóły_Rezerwacji] CHECK CONSTRAINT [FK_Szczegóły_Rezerwacji_Obostrzenia]
GO

ALTER TABLE [dbo].[Szczegóły_Rezerwacji]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegóły_Rezerwacji_Rezerwacje] FOREIGN KEY([ID_rezerwacji])
REFERENCES [dbo].[Rezerwacje] ([ID_rezerwacji])
GO

ALTER TABLE [dbo].[Szczegóły_Rezerwacji] CHECK CONSTRAINT [FK_Szczegóły_Rezerwacji_Rezerwacje]
GO
```

9. **Tabela Rezerwacje_Ind -** tabela, której celem jest spełnienie założenia, że klient indywidualny musi złożyć zamówienie przy składaniu rezerwacji.
   a. **ID_rezerwacji (int)** określa rezerwację, jest kluczem głównym tabeli
   b. **ID_klienta (int)** to klucz obcy, będący numerem klienta indywidualnego, który złożył rezerwację
   c. **ID_zamówienia (int)** to klucz obcy, który określa zamówienie złożone przy danej rezerwacji

```
CREATE TABLE [dbo].[Rezerwacje_Ind](
	[ID_rezerwacji] [INT] NOT NULL,
	[ID_klienta] [INT] NOT NULL,
	[ID_zamówienia] [INT] NOT NULL,
 CONSTRAINT [PK_Rezerwacje_Ind] PRIMARY KEY CLUSTERED
(
	[ID_rezerwacji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rezerwacje_Ind]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Ind_Klienci_Ind]
FOREIGN KEY([ID_klienta])
REFERENCES [dbo].[Klienci_Ind] ([ID_klienta])
GO

ALTER TABLE [dbo].[Rezerwacje_Ind] CHECK CONSTRAINT [FK_Rezerwacje_Ind_Klienci_Ind]
GO

ALTER TABLE [dbo].[Rezerwacje_Ind]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Ind_Rezerwacje]
FOREIGN KEY([ID_rezerwacji])
REFERENCES [dbo].[Rezerwacje] ([ID_rezerwacji])
GO

ALTER TABLE [dbo].[Rezerwacje_Ind] CHECK CONSTRAINT [FK_Rezerwacje_Ind_Rezerwacje]
GO
```

```
ALTER TABLE [dbo].[Rezerwacje_Ind]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Ind_Zamówienia]
FOREIGN KEY([ID_zamówienia])
REFERENCES [dbo].[Zamówienia] ([ID_zamówienia])
GO

ALTER TABLE [dbo].[Rezerwacje_Ind] CHECK CONSTRAINT [FK_Rezerwacje_Ind_Zamówienia]
GO
```

10. **Tabela Rezerwacje_Firm -** reprezentuje rezerwacje złożone przez klienta firmowego na firmę. Zawiera informację o:
    a. Numerze rezerwacji **ID_rezerwacji (int)**, który jest kluczem głównym
    b. **ID_firmy (int)**, które jest kluczem obcym określającym firmę, składającą rezerwację

```
CREATE TABLE [dbo].[Rezerwacje_Firm](
	[ID_rezerwacji] [INT] NOT NULL,
	[ID_firmy] [INT] NOT NULL,
 CONSTRAINT [PK_Rezerwacje_Firm] PRIMARY KEY CLUSTERED
(
	[ID_rezerwacji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rezerwacje_Firm]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Firm_Klienci_Biz]
FOREIGN KEY([ID_firmy])
REFERENCES [dbo].[Klienci_Biz] ([ID_klienta])
GO

ALTER TABLE [dbo].[Rezerwacje_Firm] CHECK CONSTRAINT [FK_Rezerwacje_Firm_Klienci_Biz]
GO

ALTER TABLE [dbo].[Rezerwacje_Firm]  WITH CHECK ADD  CONSTRAINT [FK_Rezerwacje_Firm_Rezerwacje]
FOREIGN KEY([ID_rezerwacji])
REFERENCES [dbo].[Rezerwacje] ([ID_rezerwacji])
GO

ALTER TABLE [dbo].[Rezerwacje_Firm] CHECK CONSTRAINT [FK_Rezerwacje_Firm_Rezerwacje]
GO
```

11. **Tabela Rezerwacje_Firm_Imiennie -** tabela, reprezentująca rezerwacje składane przez firmę na nazwiska jej konkretnych pracowników
    a. Klucz główny stanowi trójka **ID_rezerwacji (int), ID_pracownika (int), ID_firmy (int)**
    b. Ponadto **ID_rezerwacji** jest kluczem obcym oznaczającym numer rezerwacji
    c. Para **ID_pracownika, ID_firmy** to klucz obcy określający osobę, pracownika danej firmy, który jest uwzględniony przy danej rezerwacji

```
CREATE TABLE [dbo].[Rezerwacje_Firm_Imiennie](
	[ID_rezerwacji] [INT] NOT NULL,
	[ID_pracownika] [INT] NOT NULL,
	[ID_firmy] [INT] NOT NULL,
 CONSTRAINT [PK_Rezerwacje_Firm_Imiennie] PRIMARY KEY CLUSTERED
(
	[ID_rezerwacji] ASC,
	[ID_pracownika] ASC,
	[ID_firmy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rezerwacje_Firm_Imiennie]  WITH CHECK ADD  CONSTRAINT
[FK_Rezerwacje_Firm_Imiennie_Pracownicy_Firm] FOREIGN KEY([ID_firmy], [ID_pracownika])
```

```
REFERENCES [dbo].[Pracownicy_Firm] ([ID_firmy], [ID_pracownika])
GO

ALTER TABLE [dbo].[Rezerwacje_Firm_Imiennie] CHECK CONSTRAINT
[FK_Rezerwacje_Firm_Imiennie_Pracownicy_Firm]
GO

ALTER TABLE [dbo].[Rezerwacje_Firm_Imiennie]  WITH CHECK ADD  CONSTRAINT
[FK_Rezerwacje_Firm_Imiennie_Rezerwacje] FOREIGN KEY([ID_rezerwacji])
REFERENCES [dbo].[Rezerwacje] ([ID_rezerwacji])
GO

ALTER TABLE [dbo].[Rezerwacje_Firm_Imiennie] CHECK CONSTRAINT
[FK_Rezerwacje_Firm_Imiennie_Rezerwacje]
GO
```

12. **Tabela Rabaty -** zbiera wszystkie dostępne typy rabatu wraz z warunkami, które dotyczą wszystkich typów rabatów
    a. **ID_rabatu (int)** to klucz główny
    b. **Wymagana_kwota (money)** oznacza minimalną kwotę, która może pozwolić na spełnienie warunków przyznania rabatu
    c. **Wysokość_jedn (float)** to podstawowy rabat, który jest przyznawany lub naliczany przy spełnieniu warunków
    d. Każdy rabat posiada informację o dacie wprowadzenia **Data_wprowadzenia (date)** oraz dacie anulowania zasad, jeśli ten rabat już nie obowiązuje **Data_zdjęcia (date)**. Daty te pozwalają na sprawdzenie warunków w przypadku rabatów ciągłych, jeśli w okresie ciągłości zasady naliczania rabatu uległy zmianie.
    e. **ID_restauracji (int)** stanowi klucz obcy określający lokal, w którym dany rabat obowiązuje

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

```
CREATE TABLE [dbo].[Rabaty](
	[ID_rabatu] [INT] IDENTITY(1,1) NOT NULL,
	[Wymagana_kwota] [MONEY] NOT NULL,
	[Wysokosc_jedn] [FLOAT] NOT NULL,
	[Data_wprowadzenia] [DATE] NOT NULL,
	[Data_zdjęcia] [DATE] NULL,
	[ID_Restauracji] [INT] NOT NULL,
 CONSTRAINT [PK_Rabaty] PRIMARY KEY CLUSTERED
(
	[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rabaty] ADD  CONSTRAINT [DF_Rabaty_Data_wprowadzenia]  DEFAULT (GETDATE()) FOR
[Data_wprowadzenia]
GO

ALTER TABLE [dbo].[Rabaty]  WITH CHECK ADD  CONSTRAINT [FK_Rabaty_Restauracje] FOREIGN
KEY([ID_Restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Rabaty] CHECK CONSTRAINT [FK_Rabaty_Restauracje]
GO
```

```
ALTER TABLE [dbo].[Rabaty]  WITH CHECK ADD  CONSTRAINT [CK_Rabaty_Wymagana_Kwota] CHECK
(([Wymagana_kwota]>(0)))
GO

ALTER TABLE [dbo].[Rabaty] CHECK CONSTRAINT [CK_Rabaty_Wymagana_Kwota]
GO

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

ALTER TABLE [dbo].[Rabaty] CHECK CONSTRAINT [CK_Rabaty_Wys_Jedn]
GO
```

13. **Tabela Aktualnie_przyznane_rabaty -** tabela zawierająca wszystkie rabaty przyznane klientom, które aktualnie obowiązują
    a. Klucz główny stanowi para **ID_rabatu (int), ID_klienta (int)**
    b. **ID_rabatu** to klucz obcy określający typ przyznanego rabatu
    c. **ID_klienta** jest kluczem obcym oznaczającym klienta
    d. **Data_przyznania (date)** określa dzień, w którym rabat został nadany klientowi, natomiast **Data_wygaśnięcia (date)** określa dzień, w którym rabat traci swoją ważność, o ile taki dzień znamy

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

```
CREATE TABLE [dbo].[Aktualnie_Przyznane_Rabaty](
	[ID_rabatu] [INT] NOT NULL,
	[ID_klienta] [INT] NOT NULL,
	[Data_przyznania] [DATE] NOT NULL,
	[Data_wygaśnięcia] [DATE] NULL,
 CONSTRAINT [PK_Aktualnie_Przyznane_Rabaty] PRIMARY KEY CLUSTERED
(
	[ID_rabatu] ASC,
	[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_Przyznane_Rabaty_Klienci] FOREIGN KEY([ID_klienta])
REFERENCES [dbo].[Klienci] ([ID_klienta])
GO

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] CHECK CONSTRAINT
[FK_Aktualnie_Przyznane_Rabaty_Klienci]
GO

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_Przyznane_Rabaty_Rabaty1] FOREIGN KEY([ID_rabatu])
REFERENCES [dbo].[Rabaty] ([ID_rabatu])
GO

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] CHECK CONSTRAINT
[FK_Aktualnie_Przyznane_Rabaty_Rabaty1]
GO
```

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

ALTER TABLE [dbo].[Aktualnie_Przyznane_Rabaty] CHECK CONSTRAINT [CK_Aktualnie_Przyznane_Rabaty_Daty]
GO
```

14. **Tabela Rabaty_Ind_Jednorazowe -** tabela zawierająca dodatkowe warunki dotyczące rabatów dla klientów indywidualnych, które są jednorazowe
    a. **ID_rabatu (int)** to klucz główny i określa dodatkowo numer po którym możemy rozpoznać dany rabat
    b. **Waznosc (int)** to ustalona przez zasady liczba dni, przez które dany rabat obowiązuje od chwili jego przyznania

Podawana w dniach ważność rabatu musi być dodatnia.

```
CREATE TABLE [dbo].[Rabaty_Ind_Jednorazowe](
	[ID_rabatu] [INT] NOT NULL,
	[Waznosc] [INT] NOT NULL,
 CONSTRAINT [PK_Rabaty_Ind_Jednorazowe] PRIMARY KEY CLUSTERED
(
	[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rabaty_Ind_Jednorazowe]  WITH CHECK ADD  CONSTRAINT
[FK_Rabaty_Ind_Jednorazowe_Rabaty] FOREIGN KEY([ID_rabatu])
REFERENCES [dbo].[Rabaty] ([ID_rabatu])
GO

ALTER TABLE [dbo].[Rabaty_Ind_Jednorazowe] CHECK CONSTRAINT [FK_Rabaty_Ind_Jednorazowe_Rabaty]
GO

ALTER TABLE [dbo].[Rabaty_Ind_Jednorazowe]  WITH CHECK ADD  CONSTRAINT
[CK_Rabaty_Ind_Jednorazowe_Waznosc] CHECK  (([Waznosc]>(0)))
GO

ALTER TABLE [dbo].[Rabaty_Ind_Jednorazowe] CHECK CONSTRAINT [CK_Rabaty_Ind_Jednorazowe_Waznosc]
GO
```

15. **Tabela Rabaty_Ind_Stale -** tabela zawierająca dodatkowe warunki, które klient indywidualny musi spełnić, aby mu przyznano rabat stały.
    a. **ID_rabatu (int)** to klucz główny i określa dodatkowo numer po którym możemy rozpoznać dany rabat
    b. **Liczba_zamowien (int)** określa ilość złożonych zamówień, po której przekroczeniu klientowi jest przyznawany ten typ rabatu

Liczba zamówień musi być liczbą dodatnią.

```
CREATE TABLE [dbo].[Rabaty_Ind_Stale](
	[ID_rabatu] [INT] NOT NULL,
	[Liczba_zamowien] [INT] NOT NULL,
 CONSTRAINT [PK_Rabaty_Ind_Stale] PRIMARY KEY CLUSTERED
(
	[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rabaty_Ind_Stale]  WITH CHECK ADD  CONSTRAINT [FK_Rabaty_Ind_Stale_Rabaty1]
FOREIGN KEY([ID_rabatu])
REFERENCES [dbo].[Rabaty] ([ID_rabatu])
GO

ALTER TABLE [dbo].[Rabaty_Ind_Stale] CHECK CONSTRAINT [FK_Rabaty_Ind_Stale_Rabaty1]
GO

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

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

16. **Tabela Rabaty_Firm_Miesiac -** określa dodatkowe warunki, jakie musi spełnić klient biznesowy aby otrzymać/utrzymać rabat typu miesięczny
    a. **ID_rabatu (int)** to klucz główny i określa dodatkowo numer po którym możemy rozpoznać dany rabat
    b. **Liczba_zamowien (int)** określa ilość złożonych zamówień, po której przekroczeniu klientowi jest przyznawany ten typ rabatu
    c. **Max_rabat (float)** to górna granica naliczanego rabatu.

    Liczba zamówień musi być liczbą dodatnią.
    Maksymalny możliwy do zebrania rabat musi być liczbą z przedziału [0,1].

```
CREATE TABLE [dbo].[Rabaty_Firm_Miesiac](
	[ID_rabatu] [INT] NOT NULL,
	[Liczba_zamowien] [INT] NOT NULL,
	[Max_rabat] [FLOAT] NOT NULL,
 CONSTRAINT [PK_Rabaty_Firm_Miesiac] PRIMARY KEY CLUSTERED
(
	[ID_rabatu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Rabaty_Firm_Miesiac]  WITH CHECK ADD  CONSTRAINT [FK_Rabaty_Firm_Miesiac_Rabaty]
FOREIGN KEY([ID_rabatu])
REFERENCES [dbo].[Rabaty] ([ID_rabatu])
GO

ALTER TABLE [dbo].[Rabaty_Firm_Miesiac] CHECK CONSTRAINT [FK_Rabaty_Firm_Miesiac_Rabaty]
GO

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

ALTER TABLE [dbo].[Rabaty_Firm_Miesiac] CHECK CONSTRAINT [CK_Rabaty_Firm_Miesiac_L_Zam]
GO

ALTER TABLE [dbo].[Rabaty_Firm_Miesiac]  WITH CHECK ADD  CONSTRAINT
[CK_Rabaty_Firm_Miesiac_Max_Rabat] CHECK  (([Max_rabat]>=(0) AND [Max_rabat]<=(1)))
GO

ALTER TABLE [dbo].[Rabaty_Firm_Miesiac] CHECK CONSTRAINT [CK_Rabaty_Firm_Miesiac_Max_Rabat]
GO
```

17. **Tabela Restauracje** - tabela wszystkich restauracji, które korzystają z bazy danych

a. **ID_Restauracji (int)** to klucz główny identyfikujący każdą restaurację
b. **Nazwa (varchar)** - nazwa danej restauracji
c. **Ulica (varchar)** - ulica na której znajduje się dana restauracja
d. **Miasto (int)** - identyfikator miasta w którym znajduje się dana restauracja

Ulica kończy się cyfrą, co oznacza numer domu lub małą literą (w sytuacji gdy numer domu to np. 47a)

```
CREATE TABLE [dbo].[Restauracje](
	[ID_restauracji] [int] IDENTITY(1,1) NOT NULL,
	[Nazwa] [varchar](50) NOT NULL,
	[Ulica] [varchar](50) NOT NULL,
	[Miasto] [int] NOT NULL,
 CONSTRAINT [PK_Restauracje] PRIMARY KEY CLUSTERED
(
	[ID_restauracji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Restauracje]  WITH CHECK ADD  CONSTRAINT [FK_Restauracje_Miasta] FOREIGN
KEY([Miasto])
REFERENCES [dbo].[Miasta] ([ID_miasta])
GO

ALTER TABLE [dbo].[Restauracje] CHECK CONSTRAINT [FK_Restauracje_Miasta]
GO

ALTER TABLE [dbo].[Restauracje]  WITH CHECK ADD  CONSTRAINT [CK_Restauracje_Ulica] CHECK
(([Ulica] like '%[0-9][a-z]' OR [Ulica] like '%[0-9]'))
GO

ALTER TABLE [dbo].[Restauracje] CHECK CONSTRAINT [CK_Restauracje_Ulica]
GO
```

18. **Tabela Miasta** - tabela wszystkich miast istniejących w bazie danych (słownik miast)
    a. **ID_Miasta (int)** to klucz główny jednoznacznie identyfikujący każde miasto
    b. **Nazwa miasta (varchar)** to nazwa danego miasta
    c. **ID_Państwa (int**) to identyfikator państwa w którym znajduje się dane miasto

Długość nazwy miasta musi być dłuższa niż 1

```
CREATE TABLE [dbo].[Miasta](
	[ID_miasta] [int] IDENTITY(1,1) NOT NULL,
	[Nazwa_miasta] [varchar](50) NOT NULL,
	[ID_państwa] [int] NOT NULL,
 CONSTRAINT [PK_Miasta] PRIMARY KEY CLUSTERED
(
	[ID_miasta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Miasta]  WITH CHECK ADD  CONSTRAINT [FK_Miasta_Państwa] FOREIGN KEY([ID_państwa])
REFERENCES [dbo].[Państwa] ([ID_państwa])
GO

ALTER TABLE [dbo].[Miasta] CHECK CONSTRAINT [FK_Miasta_Państwa]
```

```
GO

ALTER TABLE [dbo].[Miasta]  WITH CHECK ADD  CONSTRAINT [CK_Miasta_Nazwa] CHECK
((len([Nazwa_miasta])>(1)))
GO

ALTER TABLE [dbo].[Miasta] CHECK CONSTRAINT [CK_Miasta_Nazwa]
GO
```

19. **Tabela Państwa** - tabela wszystkich państw znajdujących się w bazie danych (słownik państw)
    a. **ID_Państwa (int)** to klucz główny jednoznacznie identyfikujący każde państwo
    b. **Nazwa (varchar)** to nazwa danego państwa

Nazwa państwa jest wartością UNIQUE

```
CREATE TABLE [dbo].[Państwa](
      [ID_państwa] [int] IDENTITY(1,1) NOT NULL,
      [Nazwa] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Państwa] PRIMARY KEY CLUSTERED
(
      [ID_państwa] 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 [Un_Państwa_Nazwa] UNIQUE NONCLUSTERED
(
      [Nazwa] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

20. **Dostawcy** - tabela zawierająca dane o wszystkich dostawcach półproduktów w bazie danych
    a. **ID_Dostawcy (int)** to klucz główny jednoznacznie identyfikujący każdego dostawcę
    b. **Nazwa_firmy (varchar)** jak sama nazwa wskazuje, określa nazwę firmy będącej dostawcą
    c. **Ulica (varchar)** określa ulicę na której znajduje się siedziba/oddział firmy
    d. **Kod pocztowy (varchar)** określa kod pocztowy adresu podanego w punkcie powyżej
    e. **ID_Miasta (int)** określa miasto adresu podanego w punkcie wyżej
    f. **Telefon_Kontaktowy (varchar)** to numer telefonu do danej firmy

Ulica kończy się cyfrą, co oznacza numer domu lub małą literą (w sytuacji gdy numer domu to np. 47a)
Kod pocztowy formy XXXXX albo XX-XXX
Telefon kontaktowy musi składać się z 9 cyfr

```
CREATE TABLE [dbo].[Dostawcy](
      [ID_dostawcy] [int] IDENTITY(1,1) NOT NULL,
      [Nazwa_firmy] [varchar](50) NOT NULL,
      [Ulica] [varchar](50) NOT NULL,
      [Kod_pocztowy] [varchar](6) NOT NULL,
      [ID_miasta] [int] NOT NULL,
      [Telefon_kontaktowy] [varchar](9) NOT NULL,
 CONSTRAINT [PK_Dostawcy] PRIMARY KEY CLUSTERED
(
      [ID_dostawcy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Dostawcy]  WITH CHECK ADD  CONSTRAINT [FK_Dostawcy_Miasta] FOREIGN
KEY([ID_miasta])
REFERENCES [dbo].[Miasta] ([ID_miasta])
GO

ALTER TABLE [dbo].[Dostawcy] CHECK CONSTRAINT [FK_Dostawcy_Miasta]
GO

ALTER TABLE [dbo].[Dostawcy]  WITH CHECK ADD  CONSTRAINT [CK_Dostawcy_Kod_pocztowy] CHECK
(([Kod_pocztowy] like '[0-9][0-9][0-9][0-9][0-9]' OR [Kod_pocztowy] like '[0-9][0-9]-[0-9][0-9][0-
9]'))
GO

ALTER TABLE [dbo].[Dostawcy] CHECK CONSTRAINT [CK_Dostawcy_Kod_pocztowy]
GO

ALTER TABLE [dbo].[Dostawcy]  WITH CHECK ADD  CONSTRAINT [CK_Dostawcy_Telefon] CHECK
(([Telefon_kontaktowy] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Dostawcy] CHECK CONSTRAINT [CK_Dostawcy_Telefon]
GO

ALTER TABLE [dbo].[Dostawcy]  WITH CHECK ADD  CONSTRAINT [CK_Dostawcy_Ulica] CHECK  (([Ulica] like
'%[0-9][a-z]' OR [Ulica] like '%[0-9]'))
GO

ALTER TABLE [dbo].[Dostawcy] CHECK CONSTRAINT [CK_Dostawcy_Ulica]
GO
```

21. **Aktualnie_Dostarcza** - tabela zawierająca półprodukty, które każdy z dostawców może dostarczyć do restauracji, zawiera unikalne pary dostawca - półprodukt
    a. **ID_Dostawcy (int)** to klucz główny określający konkretnego dostawcę
    b. **ID_Produktu (int)** to drugi klucz główny określający produkt, który dostarcza dostawca wskazany w punkcie wyżej

```
CREATE TABLE [dbo].[Aktualnie_Dostarcza](
	[ID_dostawcy] [int] NOT NULL,
	[ID_produktu] [int] NOT NULL,
 CONSTRAINT [PK_Aktualnie_Dostarcza] PRIMARY KEY CLUSTERED
(
	[ID_dostawcy] ASC,
	[ID_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Aktualnie_Dostarcza]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_Dostarcza_Dostawcy] FOREIGN KEY([ID_dostawcy])
REFERENCES [dbo].[Dostawcy] ([ID_dostawcy])
GO

ALTER TABLE [dbo].[Aktualnie_Dostarcza] CHECK CONSTRAINT [FK_Aktualnie_Dostarcza_Dostawcy]
GO

ALTER TABLE [dbo].[Aktualnie_Dostarcza]  WITH CHECK ADD  CONSTRAINT
[FK_Aktualnie_Dostarcza_Półprodukty] FOREIGN KEY([ID_produktu])
REFERENCES [dbo].[Półprodukty] ([ID_półproduktu])
GO

ALTER TABLE [dbo].[Aktualnie_Dostarcza] CHECK CONSTRAINT [FK_Aktualnie_Dostarcza_Półprodukty]
GO
```

22. **Dostawy** - tabela zawierająca historię wszystkich dostaw, zrealizowanych i jeszcze niezrealizowanych
    a. **ID_Dostawy (int)** to klucz główny, unikalny identyfikator każdej dostawy

b. **ID_Dostawcy (int)** to identyfikator dostawcy, u którego zamówiono dostawę półproduktów
c. **Data_Zamówienia (date)** to data określająca kiedy zlecono dostawę półproduktów
d. **Data_dostawy (date)** to data określająca kiedy dostawa została zrealizowana. W przypadku, gdy dostawa nie została jeszcze zrealizowana, w tym miejscu występuje wartość null
e. **ID_Restauracji (int)** to identyfikator restauracji, która zleciła dostawę

Data dostawy ma albo wartość null, lub wartość późniejszą niż data zamówienia
Data zamówienia ma domyślną wartość getDate().

```
CREATE TABLE [dbo].[Dostawy](
	[ID_dostawy] [INT] IDENTITY(1,1) NOT NULL,
	[ID_dostawcy] [INT] NOT NULL,
	[Data_zamówienia] [DATE] NOT NULL,
	[Data_dostawy] [DATE] NULL,
	[ID_Restauracji] [INT] NOT NULL,
 CONSTRAINT [PK_Dostawy] PRIMARY KEY CLUSTERED
(
	[ID_dostawy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Dostawy] ADD  CONSTRAINT [DF_Dostawy_Data_zamówienia]  DEFAULT (GETDATE()) FOR
[Data_zamówienia]
GO

ALTER TABLE [dbo].[Dostawy]  WITH CHECK ADD  CONSTRAINT [FK_Dostawy_Dostawcy] FOREIGN
KEY([ID_dostawcy])
REFERENCES [dbo].[Dostawcy] ([ID_dostawcy])
GO

ALTER TABLE [dbo].[Dostawy] CHECK CONSTRAINT [FK_Dostawy_Dostawcy]
GO

ALTER TABLE [dbo].[Dostawy]  WITH CHECK ADD  CONSTRAINT [FK_Dostawy_Restauracje] FOREIGN
KEY([ID_Restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Dostawy] CHECK CONSTRAINT [FK_Dostawy_Restauracje]
GO

ALTER TABLE [dbo].[Dostawy]  WITH CHECK ADD  CONSTRAINT [CK_Dostawy_Daty] CHECK  (([Data_dostawy] IS
NULL OR [Data_dostawy]>=[Data_zamówienia]))
GO

ALTER TABLE [dbo].[Dostawy] CHECK CONSTRAINT [CK_Dostawy_Daty]
GO
```

23. **Szczegóły dostaw** - tabela zawierająca konkretne półprodukty których dotyczy każda dostawa
    a. **ID_Dostawy (int)** to klucz główny określający identyfikator dostawy, której dotyczy rekord
    b. **ID_Półproduktu (int)** to drugi klucz główny określający id półproduktu ujętego w dostawie
    c. **Ilość_jednostek (float)** określa w jakiej ilości produkt został/ma zostać dostarczony

Ilość jednostek musi być większa od zera

```
 CREATE TABLE [dbo].[Szczegóły_Dostaw](
	[ID_dostawy] [int] NOT NULL,
	[ID_półproduktu] [int] NOT NULL,
	[Ilość_jednostek] [float] NOT NULL,
  CONSTRAINT [PK_Szczegóły_Dostaw] PRIMARY KEY CLUSTERED
```

```
(
	[ID_dostawy] ASC,
	[ID_półproduktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw]  WITH CHECK ADD  CONSTRAINT [FK_Szczegóły_Dostaw_Dostawy]
FOREIGN KEY([ID_dostawy])
REFERENCES [dbo].[Dostawy] ([ID_dostawy])
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw] CHECK CONSTRAINT [FK_Szczegóły_Dostaw_Dostawy]
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw]  WITH CHECK ADD  CONSTRAINT [FK_Szczegóły_Dostaw_Półprodukty]
FOREIGN KEY([ID_półproduktu])
REFERENCES [dbo].[Półprodukty] ([ID_półproduktu])
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw] CHECK CONSTRAINT [FK_Szczegóły_Dostaw_Półprodukty]
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw]  WITH CHECK ADD  CONSTRAINT [CK_Szczegóły_Dostaw_Ilosc_Jedn]
CHECK  (([Ilość_jednostek]>(0)))
GO

ALTER TABLE [dbo].[Szczegóły_Dostaw] CHECK CONSTRAINT [CK_Szczegóły_Dostaw_Ilosc_Jedn]
GO
```

24. Półprodukty - tabela zawierająca dane o wszystkich półproduktach istniejących w przepisach

a. **ID_Półproduktu (int)** to klucz główny jednoznacznie identyfikujący każdy półprodukt

b. **Nazwa (varchar)** to nazwa własna danego półproduktu

Nazwa półproduktu jest wartością unikalną

```
CREATE TABLE [dbo].[Półprodukty](
	[ID_półproduktu] [int] IDENTITY(1,1) NOT NULL,
	[Nazwa] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Półprodukty] PRIMARY KEY CLUSTERED
(
	[ID_półproduktu] 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 [Un_Półprodukty_Nazwa] UNIQUE NONCLUSTERED
(
	[Nazwa] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

25. **Stan_Magazynowy** - tabela określająca dostępność każdego półproduktu w każdej restauracji
    a. **ID_Półproduktu (int)** to klucz główny określający półprodukt
    b. **ID_Restauracji (int)** to drugi klucz główny określający restauracje, której stan magazynowy danego półproduktu określamy
    c. **Stan_Magazynowy (float)** określa ilość półproduktu, która aktualnie znajduje się w magazynie wskazanej restauracji

Wartość aktualnego stanu w magazynie musi być większa bądź równa zero

```
CREATE TABLE [dbo].[Stan_Magazynowy](
	[ID_półproduktu] [int] NOT NULL,
	[ID_restauracji] [int] NOT NULL,
	[Stan_magazynowy] [float] NOT NULL,
 CONSTRAINT [PK_StanMagazynowy] PRIMARY KEY CLUSTERED
(
	[ID_półproduktu] ASC,
	[ID_restauracji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Stan_Magazynowy]  WITH CHECK ADD  CONSTRAINT [FK_StanMagazynowy_Półprodukty]
FOREIGN KEY([ID_półproduktu])
REFERENCES [dbo].[Półprodukty] ([ID_półproduktu])
GO

ALTER TABLE [dbo].[Stan_Magazynowy] CHECK CONSTRAINT [FK_StanMagazynowy_Półprodukty]
GO

ALTER TABLE [dbo].[Stan_Magazynowy]  WITH CHECK ADD  CONSTRAINT [FK_StanMagazynowy_Restauracje]
FOREIGN KEY([ID_restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Stan_Magazynowy] CHECK CONSTRAINT [FK_StanMagazynowy_Restauracje]
GO

ALTER TABLE [dbo].[Stan_Magazynowy]  WITH CHECK ADD  CONSTRAINT [CK_Stan_Magazynowy] CHECK
(([Stan_magazynowy]>=(0)))
GO

ALTER TABLE [dbo].[Stan_Magazynowy] CHECK CONSTRAINT [CK_Stan_Magazynowy]
GO
```

26. **Przepisy** - tabela zawierająca szczegóły wszystkich dań. Określa półprodukty potrzebne do stworzenia danego dania oraz wymaganej ilości danych półproduktów
    a. **ID_Dania (int)** to klucz główny określający danie, którego szczegóły są opisywane
    b. **ID_Półproduktu (int)** to drugi klucz główny określający jeden półprodukt, który zawiera się w składzie dania
    c. **Potrzebna_ilość (float)** określa w jakiej ilości dany półprodukt jest potrzebny do stworzenia wskazanego dania

Potrzebna ilość półproduktu musi być większa niż zero

```
CREATE TABLE [dbo].[Przepisy](
	[ID_dania] [int] NOT NULL,
	[ID_półproduktu] [int] NOT NULL,
	[Potrzebna_ilość] [float] NOT NULL,
```

```
 CONSTRAINT [PK_Przepisy] PRIMARY KEY CLUSTERED
(
      [ID_dania] ASC,
      [ID_półproduktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD  CONSTRAINT [FK_Przepisy_Dania] FOREIGN
KEY([ID_dania])
REFERENCES [dbo].[Dania] ([ID_dania])
GO

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

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD  CONSTRAINT [FK_Przepisy_Półprodukty] FOREIGN
KEY([ID_półproduktu])
REFERENCES [dbo].[Półprodukty] ([ID_półproduktu])
GO

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

ALTER TABLE [dbo].[Przepisy]  WITH CHECK ADD  CONSTRAINT [CK_Przepisy_Ilosc] CHECK
(([Potrzebna_ilość]>(0)))
GO

ALTER TABLE [dbo].[Przepisy] CHECK CONSTRAINT [CK_Przepisy_Ilosc]
GO
```

27. Dania - tabela zawierająca wszystkie dania, które mogą się znaleźć w ofercie restauracji

a. **ID_Dania (int)** to klucz główny, jednoznacznie identyfikujący każde danie
b. **Nazwa_dania (varchar)** określa nazwę każdego dania
c. **Cena_dania (money)** określa cenę danego dania
d. **Kategoria (int)** określa identyfikator kategorii, do której należy danie
e. **Opis dania (varchar)** opcjonalne pole, zawierające opis dania

Cena dania musi być większa od zera

Długość nazwy dania musi być większa niż 2

```
CREATE TABLE [dbo].[Dania](
      [ID_dania] [int] IDENTITY(1,1) NOT NULL,
      [Nazwa_dania] [varchar](50) NOT NULL,
      [Cena_dania] [money] NOT NULL,
      [Kategoria] [int] NOT NULL,
      [Opis_dania] [varchar](255) NULL,
 CONSTRAINT [PK_Dania] PRIMARY KEY CLUSTERED
(
      [ID_dania] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Dania]  WITH CHECK ADD  CONSTRAINT [FK_Dania_Kategorie] FOREIGN
KEY([Kategoria])
REFERENCES [dbo].[Kategorie] ([ID_kategorii])
GO

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

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

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

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

ALTER TABLE [dbo].[Dania] CHECK CONSTRAINT [CK_Dania_Nazwa_Dania_Min]
GO
```

28. **Kategorie** - tabela zawierające kategorie, do których możemy przypisać każde danie
    a. **ID_Kategorii (int)** to klucz główny, jednoznacznie identyfikujący każdą kategorię
    b. **Nazwa_Kategorii (varchar)** określa nazwę danej kategorii

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

```
CREATE TABLE [dbo].[Kategorie](
      [ID_kategorii] [int] IDENTITY(1,1) NOT NULL,
      [Nazwa_kategorii] [varchar](30) NOT NULL,
 CONSTRAINT [PK_Kategorie] PRIMARY KEY CLUSTERED
(
      [ID_kategorii] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Nazwa_Kategorie] UNIQUE NONCLUSTERED
(
      [Nazwa_kategorii] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Kategorie]  WITH CHECK ADD  CONSTRAINT [CK_Kategorie_Nazwa] CHECK
((len([Nazwa_kategorii])>(2)))
GO

ALTER TABLE [dbo].[Kategorie] CHECK CONSTRAINT [CK_Kategorie_Nazwa]
GO
```

29. **Menu** - tabela zawierająca wykaz dań, które znajdują się w menu aktualnie oraz tych obecnych w menu w przeszłości
    a. **ID_Pozycji (int)** to klucz główny jednoznacznie określający daną pozycję, która pojawiła się w menu (teraz bądź w przeszłości)
    b. **ID_Dania (int)** określa danie, które znajduje się pod wskazaną pozycją menu
    c. **Data_wprowadzenia (date)** określa dzień, w którym dana pozycja została wprowadzona do menu
    d. **Data_zdjęcia (date)** określa dzień, w którym dana pozycja została zdjęta z menu
    e. **ID_Restauracji (int)** określa restaurację, której dana pozycja w menu dotyczy

Data wprowadzenia domyślnie ma wartość getDate()

```
CREATE TABLE [dbo].[Menu](
	[ID_dania] [INT] NOT NULL,
	[Data_wprowadzenia] [DATE] NOT NULL,
	[Data_zdjęcia] [DATE] NULL,
	[ID_pozycji] [INT] IDENTITY(1,1) NOT NULL,
	[ID_restauracji] [INT] NOT NULL,
 CONSTRAINT [PK_Menu_1] PRIMARY KEY CLUSTERED
(
	[ID_pozycji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [FK_Menu_Dania1] FOREIGN KEY([ID_dania])
REFERENCES [dbo].[Dania] ([ID_dania])
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Dania1]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [FK_Menu_Restauracje] FOREIGN
KEY([ID_restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [FK_Menu_Restauracje]
 GO
```

30. **Zamówienia** - tabela zawierająca informacje o zamówieniach realizowanych w restauracjach
    a. **ID_Zamówienia (int)** to klucz główny jednoznacznie identyfikujący każde zamówenie
    b. **ID_Klienta (int)** określa klienta, który złożył wskazane zamówienie
    c. **Data_zamówienia (date)** określa datę, kiedy zamówienie zostało złożone
    d. **Data_odbioru (date)** określa datę, kiedy zamówienie zostało zrealizowane, bądź na kiedy ma takie być (obsługa zamawiania z wyprzedzeniem)
    e. **Na_wynos (varchar)** określa, czy dane zamówienie ma być zrealizowane na wynos bądź nie
    f. **Pracownik_obsługujący (int)** zawiera identyfikator pracownika, który zamówienie przyjął (na tej podstawie określamy też której restauracji dotyczy zamówienie)

Data zamówienia domyślnie ma wartość getDate().
Pole na_wynos przyjmuje tylko wartości „T" albo „N"

```
CREATE TABLE [dbo].[Zamówienia](
	[ID_zamówienia] [INT] IDENTITY(1,1) NOT NULL,
	[ID_klienta] [INT] NOT NULL,
	[Data_zamówienia] [DATETIME] NOT NULL,
	[Data_odbioru] [DATETIME] NOT NULL,
	[Na_wynos] [VARCHAR](1) NOT NULL,
	[Pracownik_obsługujący] [INT] NOT NULL,
 CONSTRAINT [PK_Zamówienia] PRIMARY KEY CLUSTERED
(
	[ID_zamówienia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Zamówienia] ADD  CONSTRAINT [DF_Zamówienia_Data_zamówienia]  DEFAULT (GETDATE())
FOR [Data_zamówienia]
```

```
GO

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD  CONSTRAINT [FK_Zamówienia_Klienci] FOREIGN
KEY([ID_klienta])
REFERENCES [dbo].[Klienci] ([ID_klienta])
GO

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

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD  CONSTRAINT [FK_Zamówienia_Obsługa] FOREIGN
KEY([Pracownik_obsługujący])
REFERENCES [dbo].[Obsluga] ([ID_pracownika])
GO

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

ALTER TABLE [dbo].[Zamówienia]  WITH CHECK ADD  CONSTRAINT [CK_Zamówienia_Na_wynos] CHECK
(([Na_wynos] LIKE '[TN]'))
GO

ALTER TABLE [dbo].[Zamówienia] CHECK CONSTRAINT [CK_Zamówienia_Na_wynos]
GO
```

31. **Szczegóły_Zamówień** - tabela zawierająca szczegóły złożonych zamówień
    a. **ID_Zamówienia (int)** to klucz główny określający którego zamówienia dotyczą szczegóły
    b. **ID_Pozycji (int)** to drugi klucz główny, który mówi która pozycja z menu została zamówiona
    c. **Cena_Jednostkowa (money)** zawiera informację o jednostkowej cenie dania w chwili składania zamówienia
    d. **Ilość (int)** to informacja w jakiej ilości wskazane danie zostało zamówiona

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

```
CREATE TABLE [dbo].[Szczegóły_Zamówień](
	[ID_zamówienia] [INT] NOT NULL,
	[ID_pozycji] [INT] NOT NULL,
	[Cena_jednostkowa] [MONEY] NOT NULL,
	[Ilość] [INT] NOT NULL,
 CONSTRAINT [PK_Szczegóły_Zamówień] PRIMARY KEY CLUSTERED
(
	[ID_zamówienia] ASC,
	[ID_pozycji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień]  WITH CHECK ADD  CONSTRAINT [FK_Szczegóły_Zamówień_Menu]
FOREIGN KEY([ID_pozycji])
REFERENCES [dbo].[Menu] ([ID_pozycji])
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień] CHECK CONSTRAINT [FK_Szczegóły_Zamówień_Menu]
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień]  WITH CHECK ADD  CONSTRAINT
[FK_Szczegóły_Zamówień_Zamówienia] FOREIGN KEY([ID_zamówienia])
REFERENCES [dbo].[Zamówienia] ([ID_zamówienia])
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień] CHECK CONSTRAINT [FK_Szczegóły_Zamówień_Zamówienia]
GO
```

```
ALTER TABLE [dbo].[Szczegóły_Zamówień]  WITH CHECK ADD  CONSTRAINT [CK_Szczegóły_Zamówień_cena]
CHECK  (([Cena_jednostkowa]>(0)))
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień] CHECK CONSTRAINT [CK_Szczegóły_Zamówień_cena]
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień]  WITH CHECK ADD  CONSTRAINT [CK_Szczegóły_Zamówień_ilość]
CHECK  (([Ilość]>(0)))
GO

ALTER TABLE [dbo].[Szczegóły_Zamówień] CHECK CONSTRAINT [CK_Szczegóły_Zamówień_ilość]
 GO
```

32. Obsługa - tabela zawierająca dane pracowników pracujących w każdej restauracji

a. **ID_Pracownika (int)** to klucz główny jednoznacznie identyfikujący każdego pracownika restauracji
b. **Imie (varchar)** określa imię pracownika
c. **Nazwisko (varchar)** określa nazwisko pracownika
d. **ID_Restauracji (int)** określa restaurację, w której dany pracownik jest zatrudniony

Imie nie może zawierać cyfr

Nazwisko nie może zawierać cyfr

```
CREATE TABLE [dbo].[Obsluga](
       [ID_pracownika] [INT] IDENTITY(1,1) NOT NULL,
       [Imię] [VARCHAR](30) NOT NULL,
       [Nazwisko] [VARCHAR](30) NOT NULL,
       [ID_Restauracji] [INT] NOT NULL,
 CONSTRAINT [PK_Obsługa] PRIMARY KEY CLUSTERED
(
       [ID_pracownika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Obsluga]  WITH CHECK ADD  CONSTRAINT [FK_Obsługa_Restauracje] FOREIGN
KEY([ID_Restauracji])
REFERENCES [dbo].[Restauracje] ([ID_restauracji])
GO

ALTER TABLE [dbo].[Obsluga] CHECK CONSTRAINT [FK_Obsługa_Restauracje]
GO

ALTER TABLE [dbo].[Obsluga]  WITH CHECK ADD  CONSTRAINT [CK_Obsługa_Imie] CHECK  ((NOT [Imię] LIKE
'%[0-9]%'))
GO

ALTER TABLE [dbo].[Obsluga] CHECK CONSTRAINT [CK_Obsługa_Imie]
GO

ALTER TABLE [dbo].[Obsluga]  WITH CHECK ADD  CONSTRAINT [CK_Obsługa_Nazwisko] CHECK  ((NOT
[Nazwisko] LIKE '%[0-9]%'))
GO

ALTER TABLE [dbo].[Obsluga] CHECK CONSTRAINT [CK_Obsługa_Nazwisko]
GO
```

5. Widoki

V_Braki_W_Magazynie - Lista braków w magazynach restauracji.

```
CREATE VIEW [dbo].[V_Braki_W_Magazynie]
AS
SELECT dbo.Restauracje.Nazwa, dbo.Restauracje.Ulica, dbo.Miasta.Nazwa_miasta, dbo.Polprodukty.Nazwa
AS Polprodukt, dbo.Stan_Magazynowy.Stan_magazynowy
FROM     dbo.Restauracje INNER JOIN
                  dbo.Stan_Magazynowy ON dbo.Restauracje.ID_restauracji =
dbo.Stan_Magazynowy.ID_restauracji INNER JOIN
                  dbo.Miasta ON dbo.Restauracje.Miasto = dbo.Miasta.ID_miasta INNER JOIN
                  dbo.Polprodukty ON dbo.Stan_Magazynowy.ID_półproduktu =
dbo.Polprodukty.ID_półproduktu
WHERE  (dbo.Stan_Magazynowy.Stan_magazynowy = 0)
GO
```

V_Najpopularniejsze_Dania - Lista najczęściej zamawianych potraw.

```
CREATE VIEW [dbo].[V_Najpopularniejsze_Dania]
AS
SELECT TOP (20) PERCENT dbo.Dania.Nazwa_dania, dbo.Dania.Cena_dania,
SUM(dbo.Szczegóły_Zamówień.Ilość) AS Liczba_zamówionych_jednostek
FROM     dbo.Menu INNER JOIN
                  dbo.Dania ON dbo.Menu.ID_dania = dbo.Dania.ID_dania INNER JOIN
                  dbo.Szczegóły_Zamówień ON dbo.Menu.ID_pozycji = dbo.Szczegóły_Zamówień.ID_pozycji
GROUP BY dbo.Dania.Nazwa_dania, dbo.Dania.Cena_dania, dbo.Dania.ID_dania
ORDER BY Liczba_zamówionych_jednostek DESC
GO
```

V_Pozycje_Niemozliwe_Do_Stworzenia - Lista potraw z menu, których nie można obecnie zreazlizować przez braki półproduktów

```
CREATE VIEW [dbo].[V_Pozycje_Niemozliwe_Do_Stworzenia]
AS
SELECT dbo.Restauracje.Nazwa, dbo.Dania.Nazwa_dania, dbo.Polprodukty.Nazwa AS Polprodukt,
dbo.Przepisy.Potrzebna_ilość, dbo.Stan_Magazynowy.Stan_magazynowy
FROM     dbo.Dania INNER JOIN
                  dbo.Menu ON dbo.Dania.ID_dania = dbo.Menu.ID_dania INNER JOIN
                  dbo.Przepisy ON dbo.Dania.ID_dania = dbo.Przepisy.ID_dania INNER JOIN
                  dbo.Polprodukty ON dbo.Przepisy.ID_półproduktu = dbo.Polprodukty.ID_półproduktu
INNER JOIN
                  dbo.Restauracje ON dbo.Menu.ID_restauracji = dbo.Restauracje.ID_restauracji INNER
JOIN
                  dbo.Stan_Magazynowy ON dbo.Polprodukty.ID_półproduktu =
dbo.Stan_Magazynowy.ID_półproduktu AND dbo.Restauracje.ID_restauracji =
dbo.Stan_Magazynowy.ID_restauracji AND
                  dbo.Przepisy.Potrzebna_ilość > dbo.Stan_Magazynowy.Stan_magazynowy
WHERE  (dbo.Menu.Data_zdjęcia IS NULL)
GO
```

V_Klienci_Wydatki - Statystyka łącznych kwot wydanych przez klientów.

```
CREATE VIEW [dbo].[V_Klienci_Wydatki]
AS
SELECT dbo.Klienci.ID_klienta, SUM(dbo.Szczegóły_Zamówień.Ilość *
dbo.Szczegóły_Zamówień.Cena_jednostkowa) AS Łączna_wart_zam
FROM     dbo.Klienci INNER JOIN
                  dbo.Zamówienia ON dbo.Klienci.ID_klienta = dbo.Zamówienia.ID_klienta INNER JOIN
                  dbo.Szczegóły_Zamówień ON dbo.Zamówienia.ID_zamówienia =
dbo.Szczegóły_Zamówień.ID_zamówienia
GROUP BY dbo.Klienci.ID_klienta
GO
```

V_Pracownicy_firm – Lista klientów indywidualnych pracująca w firmach również będących klientami.

```
CREATE VIEW [dbo].[V_Pracownicy_Firm]
AS
SELECT dbo.Klienci_Biz.Nazwa_firmy, dbo.Klienci_Ind.Imię, dbo.Klienci_Ind.Nazwisko
FROM     dbo.Klienci_Biz INNER JOIN
                  dbo.Pracownicy_Firm ON dbo.Klienci_Biz.ID_klienta = dbo.Pracownicy_Firm.ID_firmy
INNER JOIN
                  dbo.Klienci_Ind ON dbo.Pracownicy_Firm.ID_pracownika = dbo.Klienci_Ind.ID_klienta
GO
```

V_Owoce_morza – Lista dań z kategorii owoce morza.

```
CREATE VIEW [dbo].[V_Owoce_Morza]
AS
SELECT        dbo.Dania.Nazwa_dania, dbo.Dania.Cena_dania
FROM            dbo.Dania INNER JOIN
                         dbo.Kategorie ON dbo.Dania.Kategoria = dbo.Kategorie.ID_kategorii
WHERE        (dbo.Kategorie.Nazwa_kategorii = 'seafood')
GO
```

6. Widoki z parametrami

Pracownicy_Firmy - lista pracowników danej firmy

```
CREATE FUNCTION [dbo].[Pracownicy_Firmy]
(
	@id_firmy int
)
RETURNS TABLE
AS
RETURN
(
	SELECT i.Imię,i.Nazwisko
	FROM Klienci_Ind i
	INNER JOIN Pracownicy_Firm f on i.ID_klienta=f.ID_pracownika AND @id_firmy=f.ID_firmy
	INNER JOIN Klienci k on k.ID_klienta=i.ID_klienta
)
GO
```

Aktualne_Rabaty_Klienta - lista wszystkich rabatów, które klient aktualnie posiada

```
CREATE FUNCTION [dbo].[Aktualne_Rabaty_Klienta]
(
	@id_klienta int,
	@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
	SELECT a.Data_przyznania,a.Data_wygaśnięcia,ra.Wysokosc_jedn
	FROM Restauracje r
	INNER JOIN Rabaty ra ON r.ID_restauracji=ra.ID_Restauracji
	INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=ra.ID_rabatu
	WHERE a.ID_klienta=@id_klienta AND r.ID_restauracji=@id_restauracji
)
```

```
GO
```

Aktualne_Zamowienia_Klienta - lista zamówień złożonych jeszcze nie odebranych

```
CREATE FUNCTION [dbo].[Aktualne_Zamowienia_Klienta]
(
	@id_klienta int
)
RETURNS TABLE
AS
RETURN
(
	SELECT d.Nazwa_dania,sz.Cena_jednostkowa,sz.Ilość,z.Data_zamówienia,z.Data_odbioru,r.Nazwa
	FROM Zamówienia z
	INNER JOIN Szczegóły_Zamówień sz ON sz.ID_zamówienia=z.ID_zamówienia
	INNER JOIN Menu m ON m.ID_pozycji=sz.ID_pozycji
	INNER JOIN Dania d ON d.ID_dania=m.ID_dania
	INNER JOIN Restauracje r ON r.ID_restauracji=m.ID_restauracji
	WHERE z.Data_odbioru>GETDATE() AND @id_klienta=z.ID_klienta
)
GO
```

Dania_Z_Kategorii - lista dań z podanej kategorii

```
CREATE FUNCTION [dbo].[Dania_Z_Kategorii]
(
	@id_kategorii int
)
RETURNS TABLE
AS
RETURN
(
	SELECT Nazwa_dania,Cena_dania FROM Dania
	WHERE Kategoria=@id_kategorii
)
GO
```

Dostawy_Niezrealizowane - lista dostaw złożonych jeszcze niedostarczonych do danej restauracji

```
CREATE FUNCTION [dbo].[Dostawy_Niezrealizowane]
(
	@id_lokalu int
)
RETURNS TABLE
AS
RETURN
(
	SELECT
d.ID_dostawy,p.Nazwa,sd.Ilość_jednostek,d.Data_zamówienia,do.Nazwa_firmy,do.Telefon_kontaktowy FROM
Dostawy d
	INNER JOIN Dostawcy do ON do.ID_dostawcy=d.ID_dostawcy
	INNER JOIN Szczegóły_Dostaw sd ON sd.ID_dostawy=d.ID_dostawy
	INNER JOIN Półprodukty p ON p.ID_półproduktu=sd.ID_półproduktu
	WHERE d.ID_Restauracji=@id_lokalu AND d.Data_dostawy is NULL
)
GO
```

Lista_Rezerwacja_Imiennie - spis pracowników firmy na rezerwacje, które dana firma złożyła na przyszłość (dotyczy tylko rezerwacji składanej na konkretnych pracowników)

```
CREATE FUNCTION [dbo].[Lista_Rezerwacja_Imiennie]
(
	@id_firmy int
)
RETURNS TABLE
AS
RETURN
(
	SELECT k.Imię,k.Nazwisko,r.ID_rezerwacji,r.Data_rezerwacji,re.Nazwa,o.ID_stolika
	FROM Rezerwacje_Firm_Imiennie rfi
	INNER JOIN Rezerwacje r ON r.ID_rezerwacji=rfi.ID_rezerwacji
	INNER JOIN Restauracje re ON re.ID_restauracji=r.ID_Restauracji
	INNER JOIN Pracownicy_Firm pf ON pf.ID_firmy=rfi.ID_firmy AND
pf.ID_pracownika=rfi.ID_pracownika
	INNER JOIN Klienci_Ind k ON k.ID_klienta=pf.ID_pracownika
	INNER JOIN Szczegóły_Rezerwacji sr ON sr.ID_rezerwacji=r.ID_rezerwacji
	INNER JOIN Obostrzenia o ON o.ID_Obostrzenia=sr.ID_obostrzenia
	WHERE @id_firmy=rfi.ID_firmy AND r.Data_rezerwacji>GETDATE()
)
GO
```

Pokaz_Menu - aktualna karta dań danego lokalu

```
CREATE FUNCTION [dbo].[Pokaz_Menu]
(
	@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
	SELECT d.Nazwa_dania,m.ID_dania,m.Data_wprowadzenia,d.Cena_dania,d.Opis_dania,m.ID_pozycji
FROM Menu m
	INNER JOIN Dania d on m.ID_dania=d.ID_dania
	WHERE ID_restauracji=@id_restauracji AND Data_zdjęcia is NULL
)
GO
```

Przepis - określa półprodukty i ich ilość do wykonania danego dania

```
CREATE FUNCTION [dbo].[Przepis]
(
	@id_dania int
)
RETURNS TABLE
AS
RETURN
(
	SELECT po.Nazwa,p.Potrzebna_ilość
	FROM Przepisy p
	INNER JOIN Półprodukty po ON po.ID_półproduktu=p.ID_półproduktu
	WHERE p.ID_dania=@id_dania
)
GO
```

Rezerwacje_Klienta_Biz - spis rezerwacji dla firmy, które się jeszcze nie odbyły (dotyczy tylko rezerwacji składanych na firmę)

```
CREATE FUNCTION [dbo].[Rezerwacje_Klienta_Biz]
(
	@id_firmy int
)
RETURNS TABLE
AS
RETURN
(
	SELECT r.ID_rezerwacji,r.Data_rezerwacji,re.Nazwa,o.ID_stolika
	FROM Rezerwacje_Firm fr
	INNER JOIN Rezerwacje r ON r.ID_rezerwacji=fr.ID_rezerwacji
	INNER JOIN Szczegóły_Rezerwacji sr ON r.ID_rezerwacji=sr.ID_rezerwacji
	INNER JOIN Obostrzenia o ON o.ID_Obostrzenia=sr.ID_obostrzenia
	INNER JOIN Restauracje re ON re.ID_restauracji=r.ID_Restauracji
	WHERE @id_firmy=fr.ID_firmy AND r.Data_rezerwacji>GETDATE()
)
GO
```

Rezerwacje_Klienta_Ind - lista rezerwacji dla klienta indywidualnego, które się jeszcze nie odbyły

```
CREATE FUNCTION [dbo].[Rezerwacje_Klienta_Ind]
(
	@id_klienta int
)
RETURNS TABLE
AS
RETURN
(
	SELECT
r.Data_złożenia,r.Data_rezerwacji,re.Nazwa,o.ID_stolika,o.Liczba_miejsc,ri.ID_zamówienia
	FROM Rezerwacje r
	INNER JOIN Restauracje re ON r.ID_Restauracji=re.ID_restauracji
	INNER JOIN Rezerwacje_Ind ri ON ri.ID_rezerwacji=r.ID_rezerwacji
	INNER JOIN Szczegóły_Rezerwacji sr ON sr.ID_rezerwacji=r.ID_rezerwacji
	INNER JOIN Obostrzenia o ON o.ID_Obostrzenia=sr.ID_obostrzenia
	WHERE ri.ID_klienta=@id_klienta AND r.Data_rezerwacji>GETDATE()
)
GO
```

Stan_magazynu - określa zapas wszystkich półproduktów w magazynie konkretnej restauracji

```
CREATE FUNCTION [dbo].[Stan_Magazynu]
(
	@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
	SELECT p.Nazwa,s.Stan_magazynowy
	FROM Stan_Magazynowy s
	INNER JOIN Półprodukty p ON p.ID_półproduktu=s.ID_półproduktu
	WHERE s.ID_restauracji=@id_restauracji
)
GO
```

Znajdz_Dostawce_Polproduktu - zwraca listę dostawców od których można zakupić dany składnik

```
CREATE FUNCTION [dbo].[Znajdz_Dostawce_Polproduktu]
(
	@id_polproduktu int
)
RETURNS TABLE
AS
RETURN
(
	SELECT d.Nazwa_firmy,d.Telefon_kontaktowy FROM Aktualnie_Dostarcza a
	INNER JOIN Dostawcy d on d.ID_dostawcy=a.ID_dostawcy
	WHERE a.ID_produktu=@id_polproduktu
)
GO
```

Generuj_Fakture_Zamowienie – generuje fakturę za pojedyncze zamówienie

```
CREATE FUNCTION [dbo].[Generuj_Fakture_Zamowienie]
(
	-- Add the parameters for the function here
	@id_zamowienia INT
)
RETURNS TABLE
AS
RETURN
(
	SELECT CONCAT('Faktura za zamówienie nr: ', ID_zamówienia) AS 'SZCZEGÓŁY', NULL AS 'WARTOŚĆ'
	FROM dbo.Zamówienia
	WHERE ID_zamówienia=@id_zamowienia

	UNION ALL
	SELECT ' ', NULL
	UNION ALL

	SELECT CONCAT('Nazwa firmy: ',kb.Nazwa_firmy), NULL
	FROM dbo.Zamówienia z
	INNER JOIN dbo.Klienci_Biz kb ON kb.ID_klienta=z.ID_klienta
	WHERE ID_zamówienia=@id_zamowienia
	UNION ALL
	SELECT CONCAT('NIP: ',kb.NIP), NULL
	FROM dbo.Zamówienia z
	INNER JOIN dbo.Klienci_Biz kb ON kb.ID_klienta=z.ID_klienta
	WHERE ID_zamówienia=@id_zamowienia
	UNION ALL
	SELECT CONCAT('Adres: ',kb.Ulica, ', ',kb.Kod_pocztowy,', ',m.Nazwa_miasta, ', ',p.Nazwa),
NULL
	FROM dbo.Zamówienia z
	INNER JOIN dbo.Klienci_Biz kb ON kb.ID_klienta=z.ID_klienta
	INNER JOIN dbo.Miasta m ON m.ID_miasta=KB.ID_miasta
	INNER JOIN dbo.Panstwa p ON p.ID_państwa = m.ID_państwa
	WHERE ID_zamówienia=@id_zamowienia
	UNION ALL
	SELECT ' ', NULL
	UNION ALL
	SELECT 'Szczegóły zamówienia', NULL
	UNION ALL
	SELECT CONCAT('Zamówione danie:', d.Nazwa_dania, ',   Ilość: ', sz.Ilość, ',   Cena
jednostkowa:',
	sz.Cena_jednostkowa
	) AS 'SZCZEGÓŁY', sz.Ilość*sz.Cena_jednostkowa  AS 'SUMA' FROM dbo.Zamówienia z
	INNER JOIN dbo.Szczegóły_Zamówień sz ON z.ID_zamówienia=sz.ID_zamówienia
	INNER JOIN dbo.Menu m ON m.ID_pozycji=sz.ID_pozycji
	INNER JOIN dbo.Dania d ON d.ID_dania=m.ID_dania
	WHERE z.ID_zamówienia=@id_zamowienia
```

```
	UNION ALL

	SELECT 'ŁĄCZNA WARTOŚĆ', dbo.Wartosc_Zamowienia_Z_Rabatem(@id_zamowienia)

)
GO
```

Generuj_Fakture_Miesieczna – generuje fakturę za wszystkie zamówienia złożone w ciągu 30 dni od dnia dzisiejszego dla danego klienta biznesowego

```
CREATE FUNCTION [dbo].[Generuj_Fakture_Miesieczna]
(
	@id_klienta INT
)
RETURNS
@faktura TABLE
(
	Szczegol VARCHAR(150),
	Wartosc MONEY
)
AS
BEGIN
	DECLARE @nazwa_firmy varchar(50) = (SELECT Nazwa_firmy FROM Klienci_Biz WHERE
ID_klienta=@id_klienta)
	DECLARE @nip varchar(10) = (SELECT NIP FROM Klienci_Biz WHERE ID_klienta=@id_klienta)
	DECLARE @ulica varchar(50) = (SELECT Ulica FROM Klienci_Biz WHERE ID_klienta=@id_klienta)
	DECLARE @kod_pocztowy varchar(6) = (SELECT Kod_pocztowy FROM Klienci_Biz WHERE
ID_klienta=@id_klienta)
	DECLARE @miasto varchar(50) = (SELECT Nazwa_miasta FROM Miasta
	INNER JOIN Klienci_Biz ON Klienci_Biz.ID_miasta=Miasta.ID_miasta WHERE
@id_klienta=ID_klienta)
	DECLARE @panstwo varchar(50) = (SELECT Panstwa.Nazwa FROM Panstwa INNER JOIN Miasta ON
Miasta.ID_państwa=Panstwa.ID_państwa
	INNER JOIN Klienci_Biz ON Klienci_Biz.ID_miasta=Miasta.ID_miasta WHERE
@id_klienta=ID_klienta)
	DECLARE @adres varchar(100) = CONCAT(@ulica,', ',@kod_pocztowy,' ',@miasto,', ',@panstwo)
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES (CONCAT('Faktura miesieczna dla klienta: ', @nazwa_firmy),NULL)
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES (CONCAT('NIP: ',@nip),NULL)
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES (CONCAT('Adres: ',@adres),NULL)
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES ('',NULL)
	DECLARE @sumaryczna_wart MONEY = 0
	DECLARE iter CURSOR
	FOR
		SELECT ID_zamówienia FROM Zamówienia
		WHERE ID_klienta=@id_klienta AND DATEDIFF(DAY,Data_zamówienia,GETDATE())<=30
	DECLARE @id_zamowienia INT
	OPEN iter
	FETCH NEXT FROM iter INTO @id_zamowienia
	WHILE @@FETCH_STATUS=0
	BEGIN
		INSERT INTO @faktura(Szczegol,Wartosc)
		VALUES ('',NULL)
		DECLARE @data_zlozenia DATE =(SELECT Data_zamówienia FROM Zamówienia WHERE
@id_zamowienia=ID_zamówienia)
		INSERT INTO @faktura(Szczegol,Wartosc) VALUES
		(CONCAT('Zamowienie: ', @id_zamowienia,' zlozone dnia: ',@data_zlozenia),NULL)
		DECLARE danie CURSOR
		FOR
			SELECT ID_pozycji FROM Szczegóły_Zamówień
			WHERE ID_zamówienia=@id_zamowienia
		DECLARE @id_pozycji INT
		OPEN danie
```

```
		FETCH NEXT FROM danie INTO @id_pozycji
		WHILE @@FETCH_STATUS=0
		BEGIN
			DECLARE @nazwa_dania VARCHAR(50) =(SELECT Nazwa_dania FROM Menu m INNER JOIN
Dania d ON d.ID_dania=m.ID_dania WHERE m.ID_pozycji=@id_pozycji)
			DECLARE @ilosc INT =(SELECT Ilość FROM Szczegóły_Zamówień WHERE
@id_zamowienia=ID_zamówienia AND ID_pozycji=@id_pozycji)
			DECLARE @cena MONEY =(SELECT Cena_jednostkowa FROM Szczegóły_Zamówień WHERE
@id_zamowienia=ID_zamówienia AND ID_pozycji=@id_pozycji)
			INSERT INTO @faktura(Szczegol,Wartosc)
			VALUES(CONCAT('Zamówione danie:', @nazwa_dania, ',   Ilość: ', @ilosc, ',
Cena jednostkowa:',
				@cena),@cena*@ilosc)
			FETCH NEXT FROM danie INTO @id_pozycji
		END
		CLOSE danie
		DEALLOCATE danie
		DECLARE @laczna_wart_zam MONEY = dbo.Wartosc_Zamowienia_Z_Rabatem(@id_zamowienia)
		INSERT INTO @faktura(Szczegol,Wartosc)
		VALUES('Laczna kwota za zamowienie: ',@laczna_wart_zam)
		SET @sumaryczna_wart=@sumaryczna_wart+@laczna_wart_zam
		FETCH NEXT FROM iter INTO @id_zamowienia
	END
	CLOSE iter
	DEALLOCATE iter
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES ('',NULL)
	INSERT INTO @faktura(Szczegol,Wartosc)
	VALUES('Laczna kwota za wszystkie zamowienia: ',@sumaryczna_wart)
	RETURN
END
GO
```

Statystyki_Obslugi – posortowana lista pracowników danej restauracji pod względem sumarycznej wartości wszystkich obsłużonych zamówień.

```
CREATE FUNCTION [dbo].[Statystyki_Obslugi]
(
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT o.ID_pracownika,o.Imię, o.Nazwisko,
COUNT(DISTINCT z.ID_zamówienia) AS 'Liczba obsłużonych zamówień',
SUM(sz.Cena_jednostkowa*sz.Ilość) AS 'Łączna wartość obsłużonych zamówień'
FROM dbo.Obsluga o
INNER JOIN dbo.Zamówienia z ON o.ID_pracownika=z.Pracownik_obsługujący
INNER JOIN dbo.Szczegóły_Zamówień sz ON sz.ID_zamówienia = z.ID_zamówienia
WHERE o.ID_Restauracji=@id_restauracji
GROUP BY o.ID_pracownika,o.Imię,o.Nazwisko
ORDER BY 5 DESC OFFSET 0 ROWS
)
GO
```

Statystyki_Zamowien_Klientow_Ind – lista klientów indywidualnych uporządkowana po łącznej wartości zamówień w danej restauracji

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

```
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT ki.Imię,ki.Nazwisko,sum(sz.Cena_jednostkowa*sz.Ilość) as 'łączna wartość zamówień'
FROM Klienci_Ind ki
INNER JOIN Klienci k ON k.ID_klienta=ki.ID_klienta
INNER JOIN Zamówienia z ON z.ID_klienta=k.ID_klienta
INNER JOIN Szczegóły_Zamówień sz ON sz.ID_zamówienia=z.ID_zamówienia
WHERE z.Pracownik_obsługujący IN (SELECT o.ID_pracownika FROM Obsluga o WHERE
o.ID_Restauracji=@id_restauracji)
GROUP BY ki.ID_klienta,ki.Imię,ki.Nazwisko
ORDER BY 3 DESC OFFSET 0 ROWS
)
GO
```

Statystyki_Zamowien_Klientow_Biz - lista firmuporządkowana po łącznej wartości zamówień w danej restauracji

```
CREATE FUNCTION [dbo].[Statystyki_Zamowien_Klientow_Biz]
(
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT kb.Nazwa_firmy,sum(sz.Cena_jednostkowa*sz.Ilość) as 'łączna wartość zamówień'
FROM Klienci_Biz kb
INNER JOIN Klienci k ON k.ID_klienta=kb.ID_klienta
INNER JOIN Zamówienia z ON z.ID_klienta=k.ID_klienta
INNER JOIN Szczegóły_Zamówień sz ON sz.ID_zamówienia=z.ID_zamówienia
WHERE z.Pracownik_obsługujący IN (SELECT o.ID_pracownika FROM Obsluga o WHERE
o.ID_Restauracji=@id_restauracji)
GROUP BY kb.ID_klienta,kb.Nazwa_firmy
ORDER BY 2 DESC OFFSET 0 ROWS
)
GO
```

Statystyki_Rezerwacji_Stolikow – lista stolików wraz z ilością rezerwacji na nie w danym przedziale czasowym

```
CREATE FUNCTION  [dbo].[Statystyki_Rezerwacji_Stolikow]
(
@id_restauracji int,
@data_od date,
@data_do date
)
RETURNS TABLE
AS
RETURN
(
SELECT s.ID_stolika,COUNT(*) as 'liczba rezerwacji' FROM Rezerwacje r
INNER JOIN Szczegóły_Rezerwacji sr ON sr.ID_rezerwacji=r.ID_rezerwacji
INNER JOIN Obostrzenia o ON o.ID_Obostrzenia=sr.ID_obostrzenia
```

```
INNER JOIN Stoliki s ON s.ID_stolika=o.ID_stolika
WHERE r.ID_Restauracji=@id_restauracji AND r.Data_rezerwacji BETWEEN @data_od AND @data_do
AND s.ID_stolika IN (SELECT ID_stolika FROM dbo.Stoliki WHERE @id_restauracji=ID_Restauracji)
GROUP BY s.ID_stolika
ORDER BY 2 DESC OFFSET 0 ROWS
)
GO
```

Raport_Rabatow – podsumowanie ile rabatów danego typu jest aktualnie przyznanych przez restauracje klientom

```
CREATE FUNCTION [dbo].[Raport_Rabatow]
(
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT CONCAT('Raport rabatow dla restauracji ',@id_restauracji,'.') AS 'SZCZEGOLY',NULL AS 'LICZBA'
UNION ALL
SELECT 'Liczba klientow indywidualnych posiadajacych rabat staly:', COUNT(*)
FROM dbo.Aktualnie_Przyznane_Rabaty apr
INNER JOIN dbo.Rabaty_Ind_Stale ris ON ris.ID_rabatu=apr.ID_rabatu
INNER JOIN dbo.Rabaty r
ON r.ID_rabatu=ris.ID_rabatu AND r.ID_Restauracji=r.ID_Restauracji AND r.Data_zdjęcia IS NULL
UNION ALL

SELECT 'Liczba klientow indywidualnych posiadajacych rabat jednorazowy:', COUNT(*)
FROM dbo.Aktualnie_Przyznane_Rabaty apr
INNER JOIN dbo.Rabaty_Ind_Jednorazowe rij ON rij.ID_rabatu=apr.ID_rabatu
INNER JOIN dbo.Rabaty r
ON r.ID_rabatu=rij.ID_rabatu AND r.ID_Restauracji=r.ID_Restauracji AND r.Data_zdjęcia IS NULL

UNION ALL

SELECT 'Liczba klientow firmowych posiadajacych rabat miesieczny:', COUNT(*)
FROM dbo.Aktualnie_Przyznane_Rabaty apr
INNER JOIN dbo.Rabaty_Firm_Miesiac rfm ON rfm.ID_rabatu=apr.ID_rabatu
INNER JOIN dbo.Rabaty r
ON r.ID_rabatu=rfm.ID_rabatu AND r.ID_Restauracji=r.ID_Restauracji AND r.Data_zdjęcia IS NULL

UNION ALL

SELECT 'Liczba klientow firmowych posiadajacych rabat kwartalny:', COUNT(*)
FROM dbo.Aktualnie_Przyznane_Rabaty apr
INNER JOIN dbo.Rabaty r ON apr.ID_rabatu=r.ID_rabatu AND r.Data_zdjęcia IS NULL AND
r.ID_Restauracji=@id_restauracji
WHERE apr.ID_rabatu not IN (SELECT id_rabatu FROM dbo.Rabaty_Firm_Miesiac)
AND apr.ID_rabatu NOT IN (SELECT id_rabatu FROM dbo.Rabaty_Ind_Jednorazowe)
AND apr.ID_rabatu NOT IN (SELECT id_rabatu FROM dbo.Rabaty_Ind_Stale)
)
GO
```

Statystyki_Rezerwacji_Stolikow_Miesiac – zbiera dane dla restauracji dla rezerwacji z ostatnich 30 dni.

```
CREATE FUNCTION [dbo].[Statystyki_Rezerwacji_Stolikow_Miesiac]
```

```
(
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT * FROM dbo.Statystyki_Rezerwacji_Stolikow(@id_restauracji, DATEADD(DAY,-
30,GETDATE()),GETDATE())
)
GO
```

Statystyki_Rezerwacji_Stolikow_Tydzien– zbiera dane dla restauracji dla rezerwacji z ostatnich 7 dni

```
CREATE FUNCTION [dbo].[Statystyki_Rezerwacji_Stolikow_Tydzien]
(
@id_restauracji int
)
RETURNS TABLE
AS
RETURN
(
SELECT * FROM dbo.Statystyki_Rezerwacji_Stolikow(@id_restauracji, DATEADD(DAY,-
7,GETDATE()),GETDATE())

)
GO
```

Pokaz_Menu_Dnia – karta dań w restauracji z podanego dnia w historii

```
CREATE FUNCTION [dbo].[Pokaz_Menu_Dnia]
(
 @id_restauracji INT,
 @data date
)
RETURNS TABLE
AS
RETURN
(
SELECT ID_pozycji,Data_wprowadzenia,Data_zdjęcia,Nazwa_dania,Cena_dania, Opis_dania FROM dbo.Menu
INNER JOIN dbo.Dania ON dbo.Menu.ID_dania=dbo.Dania.ID_dania
INNER JOIN dbo.Kategorie ON Dania.Kategoria = dbo.Kategorie.ID_kategorii
WHERE ID_restauracji=@id_restauracji
AND (
(Data_zdjęcia IS NULL AND Data_wprowadzenia<=@data)
OR
(Data_zdjęcia  IS NOT NULL and @data BETWEEN Data_wprowadzenia AND Data_zdjęcia)
)
)
GO
```

Możliwe_Dania_Do_Wstawienia – Wyświetla listę dań które nie były obecne w menu przez co najmniej 30 dni, które mogą zostać dodane do menu w danej restauracji w danym dniu

```
CREATE FUNCTION [dbo].[Mozliwe_Dania_Do_Wstawienia]
(
	@id_restauracji INT,
    @data DATE
)
RETURNS TABLE
AS
RETURN
(
	(SELECT DISTINCT d.ID_dania,d.Nazwa_dania FROM Dania d  LEFT OUTER JOIN Menu m ON
m.ID_dania=d.ID_dania
	AND m.ID_restauracji=@id_restauracji
	WHERE m.Data_wprowadzenia IS NULL OR m.Data_zdjęcia<=DATEADD(DAY,-30,@data) AND d.ID_Dania
NOT IN (SELECT m2.ID_dania FROM Menu m2
                                                                    WHERE
m2.Data_wprowadzenia>DATEADD(DAY,-30,@data) AND m2.ID_restauracji=@id_restauracji))
)
GO
```

Rezerwacje_Na_Dany_Okres – lista rezerwacji dla restauracji w przedziale dat

```
CREATE FUNCTION [dbo].[Rezerwacje_Na_Dany_Okres]
(
@id_restauracji int,
@od date,
@do date
)
RETURNS TABLE
AS
RETURN
(
SELECT rez.ID_rezerwacji,rez.Data_złożenia,rez.Data_rezerwacji,ki.Imię+' '+ki.Nazwisko AS
Nazwa_Klienta FROM Restauracje r
INNER JOIN Rezerwacje rez ON rez.ID_Restauracji=r.ID_restauracji
INNER JOIN Rezerwacje_Ind ri ON ri.ID_rezerwacji=rez.ID_rezerwacji
INNER JOIN Klienci_Ind ki ON ki.ID_klienta=ri.ID_klienta
WHERE @id_restauracji=r.ID_restauracji AND rez.Data_rezerwacji BETWEEN @od AND @do

UNION

SELECT rez.ID_rezerwacji,rez.Data_złożenia,rez.Data_rezerwacji,kb.Nazwa_firmy AS Nazwa_Klienta FROM
Restauracje r
INNER JOIN Rezerwacje rez ON rez.ID_Restauracji=r.ID_restauracji
INNER JOIN Rezerwacje_Firm rf ON rf.ID_rezerwacji=rez.ID_rezerwacji
INNER JOIN Klienci_Biz kb ON kb.ID_klienta=rf.ID_firmy
WHERE @id_restauracji=r.ID_restauracji AND rez.Data_rezerwacji BETWEEN @od AND @do

UNION

SELECT DISTINCT rez.ID_rezerwacji,rez.Data_złożenia,rez.Data_rezerwacji,kb.Nazwa_firmy AS
Nazwa_Klienta FROM Restauracje r
INNER JOIN Rezerwacje rez ON rez.ID_Restauracji=r.ID_restauracji
INNER JOIN Rezerwacje_Firm_Imiennie rfi ON rfi.ID_rezerwacji=rez.ID_rezerwacji
INNER JOIN Pracownicy_Firm pf ON pf.ID_firmy=rfi.ID_firmy
INNER JOIN Klienci_Biz kb ON kb.ID_klienta=pf.ID_firmy
```

```
WHERE @id_restauracji=r.ID_restauracji AND rez.Data_rezerwacji BETWEEN @od AND @do
)
GO
```

7. Funkcje

Data_Rezerwacji – zwraca date wskazanej rezerwacji

```
CREATE FUNCTION [dbo].[Data_Rezerwacji]
(
	@id_rezerwacji INT
)
RETURNS DATE
AS
BEGIN
	RETURN(
		SELECT Data_rezerwacji FROM Rezerwacje
		WHERE @id_rezerwacji=ID_rezerwacji
	)

END
GO
```

Ilosc_Zamowien_Powyzej_Kwoty – zwraca liczbe zamowien powyżej wskazanej kwoty dla wskazanego klienta w danej restauracji

```
CREATE FUNCTION [dbo].[Ilosc_Zamowien_Powyzej_Kwoty]
(
	@id_restauracji INT,
	@id_klienta INT,
	@kwota MONEY
)
RETURNS INT
AS
BEGIN
	RETURN (
		SELECT COUNT(liczba_zam) FROM
		(
			SELECT COUNT(DISTINCT z.ID_zamówienia) AS liczba_zam FROM Zamówienia z
			INNER JOIN Szczegóły_Zamówień sz ON sz.ID_zamówienia=z.ID_zamówienia
			WHERE z.ID_klienta=@id_klienta AND z.Pracownik_obsługujący IN (SELECT
ID_pracownika FROM Obsluga WHERE ID_Restauracji=@id_restauracji)
			GROUP BY z.ID_zamówienia
			HAVING SUM(sz.Ilość*sz.Cena_jednostkowa)>@kwota
		) AS zamowienia
	)
END
GO
```

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

```
CREATE FUNCTION [dbo].[Liczba_Wolnych_Miejsc]
(
	@id_restauracji INT,
	@data_rezerwacji DATE
)
RETURNS INT
AS
BEGIN
	DECLARE @wszystkie_miejsca INT = (SELECT SUM(o.Liczba_miejsc)
		FROM Obostrzenia o
		INNER JOIN Stoliki s ON s.ID_stolika=o.ID_stolika
```

```
		WHERE s.ID_Restauracji=@id_restauracji AND
		o.Data_wprowadzenia=(SELECT MAX(o2.Data_wprowadzenia) FROM Obostrzenia o2
					WHERE o2.ID_stolika=o.ID_stolika)
		)
	DECLARE @zajete_miejsca INT =(SELECT SUM(o.Liczba_miejsc)
		FROM Obostrzenia o
		INNER JOIN Szczegóły_Rezerwacji sr ON sr.ID_obostrzenia=o.ID_Obostrzenia
		INNER JOIN Rezerwacje r ON r.ID_rezerwacji=sr.ID_rezerwacji
		WHERE r.ID_Restauracji=@id_restauracji AND
DATEDIFF(DAY,r.Data_rezerwacji,@data_rezerwacji)=0)
	RETURN (@wszystkie_miejsca-@zajete_miejsca)

END
GO
```

Nalicz_Rabat_Firm_Kwartalny – zwraca wysokość posiadanego rabatu kwartalnego przez danego klienta firmowego w danej restauracji

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Firm_Kwartalny]
(
	@id_restauracji INT,
	@id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @id_rabatu INT = (
		SELECT TOP 1 r.ID_rabatu FROM Rabaty r
		INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
		AND r.ID_rabatu NOT IN
		(SELECT ID_rabatu FROM Rabaty_Ind_Jednorazowe)
		AND r.ID_rabatu NOT IN
		(SELECT ID_rabatu FROM Rabaty_Ind_Stale)
		AND r.ID_rabatu NOT IN
		(SELECT ID_rabatu FROM Rabaty_Firm_Miesiac)
		WHERE r.ID_Restauracji=@id_restauracji AND r.Data_zdjęcia IS NULL
	)

	IF @id_rabatu IS NULL
	BEGIN
		RETURN 0
	END

	DECLARE @wysokosc float = (SELECT Wysokosc_jedn FROM Rabaty WHERE @id_rabatu=ID_rabatu)

	RETURN @wysokosc
END
GO
```

Nalicz_Rabat_Firm_Miesieczny - zwraca wysokość posiadanego rabatu miesięcznego przez danego klienta firmowego w danej restauracji

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Firm_Miesieczny]
(
	@id_restauracji INT,
	@id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @id_rabatu INT = (
		SELECT TOP 1 r.ID_rabatu FROM Rabaty r
		INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
		INNER JOIN Rabaty_Firm_Miesiac rf ON rf.ID_rabatu=r.ID_rabatu
		WHERE r.ID_Restauracji=@id_restauracji AND r.Data_zdjęcia IS NULL
	)

	IF @id_rabatu IS NULL
	BEGIN
		RETURN 0
	END

	DECLARE @data_przyznania DATE = (SELECT Data_przyznania FROM Aktualnie_Przyznane_Rabaty WHERE
@id_rabatu=ID_rabatu AND ID_klienta=@id_klienta)

	DECLARE @wysokosc FLOAT = (SELECT Wysokosc_jedn FROM Rabaty WHERE @id_rabatu=ID_rabatu)

	DECLARE @max_rabat FLOAT = (SELECT Max_rabat FROM Rabaty_Firm_Miesiac WHERE
@id_rabatu=ID_rabatu)

	IF DATEDIFF(MONTH,@data_przyznania,GETDATE())*@wysokosc<@max_rabat
	BEGIN
		RETURN DATEDIFF(MONTH,@data_przyznania,GETDATE())*@wysokosc
	END
	RETURN @max_rabat
END
GO
```

Nalicz_Rabat_Ind_Jednorazowy – zwraca wysokość posiadanego rabatu jednorazowego przez danego klienta indywidualnego w danej restauracji

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Ind_Jednorazowy]
(
	@id_restauracji INT,
	@id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @id_rabatu INT = (
		SELECT TOP 1 r.ID_rabatu FROM Rabaty r
		INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
		INNER JOIN Rabaty_Ind_Jednorazowe rj ON rj.ID_rabatu=r.ID_rabatu
		WHERE r.ID_Restauracji=@id_restauracji AND r.Data_zdjęcia IS NULL
	)

	IF @id_rabatu IS NULL
	BEGIN
		RETURN 0
	END
```

```
	DECLARE @kwota MONEY = (SELECT Wymagana_kwota FROM Rabaty WHERE ID_rabatu=@id_rabatu)

	IF GETDATE() NOT BETWEEN (SELECT Data_przyznania FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
	AND (SELECT Data_wygaśnięcia FROM Aktualnie_Przyznane_Rabaty WHERE ID_klienta=@id_klienta AND
@id_rabatu=ID_rabatu)
	BEGIN
		RETURN 0
	END

	DECLARE @laczna_wart_zam MONEY = (SELECT SUM(sz.Ilość*sz.Cena_jednostkowa) FROM
Szczegóły_Zamówień sz
		INNER JOIN Zamówienia z ON z.ID_zamówienia=sz.ID_zamówienia
		WHERE @id_klienta=z.ID_klienta)

	IF @laczna_wart_zam<@kwota
	BEGIN
		RETURN 0
	END

	RETURN (SELECT Wysokosc_jedn FROM Rabaty WHERE @id_rabatu=ID_rabatu)
END
GO
```

Nalicz_Rabat_Ind_Staly – zwraca wysokość posiadanego rabatu stałego przez danego klienta indywidualnego w danej restauracji

```
CREATE FUNCTION [dbo].[Nalicz_Rabat_Ind_Staly]
(
	@id_restauracji INT,
	@id_klienta INT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @id_rabatu INT = (
		SELECT TOP 1 r.ID_rabatu FROM Rabaty r
		INNER JOIN Aktualnie_Przyznane_Rabaty a ON a.ID_rabatu=r.ID_rabatu AND
a.ID_klienta=@id_klienta
		INNER JOIN Rabaty_Ind_Stale rs ON rs.ID_rabatu=r.ID_rabatu
		WHERE r.ID_Restauracji=@id_restauracji AND r.Data_zdjęcia IS NULL
	)

	IF @id_rabatu IS NULL
	BEGIN
		RETURN 0
	END

	DECLARE @kwota MONEY = (SELECT Wymagana_kwota FROM Rabaty WHERE ID_rabatu=@id_rabatu)

	DECLARE @ilosc_powyzej_kwoty INT =
dbo.Ilosc_Zamowien_Powyzej_Kwoty(@id_restauracji,@id_klienta,@kwota)

	DECLARE @liczba_zamowien INT = (SELECT Liczba_zamowien FROM Rabaty_Ind_Stale WHERE
ID_rabatu=@id_rabatu)

	RETURN @ilosc_powyzej_kwoty/@liczba_zamowien*(SELECT Wysokosc_jedn FROM Rabaty WHERE
ID_rabatu=@id_rabatu)

END
GO
```

Ostatnie_Usuniecie_Z_Menu – zwraca datę ostatniego usunięcia z menu danego dania w danej restauracji

```
CREATE FUNCTION [dbo].[Ostatnie_Usuniecie_Z_Menu]
(
	@id_dania INT,
	@id_restauracji INT
)
RETURNS DATE
AS
BEGIN
	RETURN (SELECT TOP 1 Data_zdjęcia FROM Menu WHERE @id_dania=ID_dania AND
@id_restauracji=ID_restauracji ORDER BY Data_zdjęcia DESC)
END
GO
```

Pobierz_Numer_Stolika – zwraca numer stolika (z obostrzeń, czyli z określoną ograniczoną liczbą miejsc) który nie jest jeszcze zarezerwowany danego dnia oraz nie został wykluczony z użytkowania

```
CREATE FUNCTION [dbo].[Pobierz_Numer_Stolika]
(
	@id_restauracji INT,
	@data_rezerwacji DATE
)
RETURNS INT
AS
BEGIN
	RETURN (SELECT TOP 1 o.ID_stolika FROM Obostrzenia o
			INNER JOIN Stoliki s ON s.ID_stolika=o.ID_stolika
			INNER JOIN Restauracje r ON r.ID_restauracji=s.ID_Restauracji
			WHERE r.ID_restauracji=@id_restauracji AND s.ID_stolika
			NOT IN(
				SELECT o2.ID_stolika FROM Szczegóły_Rezerwacji sr2
				INNER JOIN Obostrzenia o2 ON sr2.ID_obostrzenia=o2.ID_Obostrzenia
				INNER JOIN Rezerwacje r2 ON r2.ID_rezerwacji=sr2.ID_rezerwacji
				WHERE r2.ID_Restauracji=@id_restauracji
				AND DATEDIFF(DAY,r2.Data_rezerwacji,@data_rezerwacji)=0
			)
			AND(
			SELECT TOP 1 o3.Liczba_miejsc FROM Obostrzenia o3
			WHERE o3.ID_Obostrzenia=o.ID_Obostrzenia
			ORDER BY Data_wprowadzenia DESC)>0
			ORDER BY Data_wprowadzenia DESC
			)
END
GO
```

Pobierz_Obostrzenie_Do_Rezerwacji – zwraca numer obostrzenia (czyli stan stolika w danym dniu) do rezerwacji, który nie jest danego dnia zajęty

```
CREATE FUNCTION [dbo].[Pobierz_Obostrzenie_Do_Rezerwacji]
(
	@id_restauracji INT,
	@data_rezerwacji DATE
)
RETURNS INT
AS
BEGIN
	RETURN (
	SELECT TOP 1 o.ID_Obostrzenia
	FROM Obostrzenia o
	INNER JOIN Stoliki s ON s.ID_stolika=o.ID_stolika
	WHERE s.ID_Restauracji=@id_restauracji AND
	o.Data_wprowadzenia=(SELECT MAX(o2.Data_wprowadzenia) FROM Obostrzenia o2
					WHERE o2.ID_stolika=o.ID_stolika)
	AND ID_Obostrzenia NOT IN
```

```
	(SELECT o.ID_Obostrzenia
		FROM Obostrzenia o
		INNER JOIN Szczegóły_Rezerwacji sr ON sr.ID_obostrzenia=o.ID_Obostrzenia
		INNER JOIN Rezerwacje r ON r.ID_rezerwacji=sr.ID_rezerwacji
		WHERE r.ID_Restauracji=@id_restauracji AND
DATEDIFF(DAY,r.Data_rezerwacji,@data_rezerwacji)=0)
	)
END
GO
```

Wartosc_Zamowienia_Z_Rabatem – zwraca łączną wartość zamówienia z uwzględnieniem przyznanych rabatów

```
CREATE FUNCTION [dbo].[Wartosc_Zamowienia_Z_Rabatem]
(
	@id_zamowienia INT
)
RETURNS MONEY
AS
BEGIN
	RETURN (SELECT SUM(Ilość*Cena_jednostkowa)
			FROM Szczegóły_Zamówień
			WHERE ID_zamówienia=@id_zamowienia
)
END
GO
```

8. Procedury

Dodaj_Panstwo - dodanie nowego państwa do bazy

```
CREATE PROCEDURE [dbo].[Dodaj_Panstwo]
	@nazwa_panstwa VARCHAR(50)
AS
BEGIN
	SET NOCOUNT ON;
	INSERT INTO Panstwa(Nazwa) VALUES (@nazwa_panstwa)
END
GO
```

Dodaj_Miasto - dodanie nowego miasta do bazy, jeśli jest w państwie, którego wcześniej nie odnotowaliśmy to dodaje również nowe państwo

```
CREATE PROCEDURE [dbo].[Dodaj_Miasto]
	@nazwa_miasta VARCHAR(50),
	@nazwa_panstwa VARCHAR(50)
AS
BEGIN
	IF EXISTS (SELECT * FROM Panstwa WHERE Panstwa.Nazwa=@nazwa_panstwa)
		BEGIN
			DECLARE @id_panstwa INT = (SELECT ID_państwa FROM Panstwa WHERE
Panstwa.Nazwa=@nazwa_panstwa)
			INSERT INTO Miasta(Nazwa_miasta,ID_państwa) VALUES (@nazwa_miasta,@id_panstwa)
		END
	ELSE
		BEGIN
			EXEC dbo.Dodaj_Panstwo
				@nazwa_panstwa
			INSERT INTO Miasta(Nazwa_miasta,ID_państwa) VALUES (@nazwa_miasta, @@IDENTITY)
		END
END
GO
```

Dodaj_Klienta - procedura pomocnicza przy rejestracji wszystkich rodzajów klientów w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta]
	@email VARCHAR(50),
	@telefon VARCHAR(9)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		IF EXISTS(
			SELECT * FROM Klienci
			WHERE Email=@email
		)
		BEGIN
		 ;THROW 52000, 'Email jest juz zajety',1
		END
		IF EXISTS(
			SELECT * FROM Klienci
			WHERE Telefon_kontaktowy=@telefon
		)
		BEGIN
		 ;THROW 52000, 'Telefon jest juz zajety',1
		END
		INSERT INTO Klienci(Telefon_kontaktowy,Email)
		VALUES (@telefon,@email)
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta: '+ ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Klienta_Ind - pełna rejestracja klienta indywidualnego w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta_Ind]
	@imię VARCHAR(30),
	@nazwisko VARCHAR(30),
	@email VARCHAR(50),
	@telefon VARCHAR(9)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Klienta_Ind
			EXEC dbo.Dodaj_Klienta
				@email,
				@telefon
			DECLARE @id INT = @@IDENTITY
			INSERT INTO Klienci_Ind(ID_klienta,Imię,Nazwisko)
			VALUES (@id,@imię,@nazwisko)
		COMMIT TRAN Dodaj_Klienta_Ind
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Klienta_Ind
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta indywidualnego: '+
ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Klienta_Biz - pełna rejestracja klienta firmowego w bazie

```
CREATE PROCEDURE [dbo].[Dodaj_Klienta_Biz]
```

```
	@email VARCHAR(50),
	@telefon VARCHAR(9),
	@nazwa_firmy VARCHAR(50),
	@nip VARCHAR(10),
	@ulica VARCHAR(50),
	@kod VARCHAR(6),
	@nazwa_miasta VARCHAR(50),
	@nazwa_panstwa VARCHAR(50)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Klienta_Biz
			EXEC dbo.Dodaj_Klienta
				@email,
				@telefon
			DECLARE @id INT = @@IDENTITY
			IF EXISTS(SELECT * FROM Miasta WHERE Nazwa_miasta=@nazwa_miasta)
				BEGIN
					DECLARE @id_miasta INT = (SELECT ID_miasta FROM Miasta WHERE
Nazwa_miasta=@nazwa_miasta)
					INSERT INTO
Klienci_Biz(ID_klienta,Nazwa_firmy,NIP,Ulica,Kod_pocztowy,ID_miasta)
					VALUES (@id,@nazwa_firmy,@nip,@ulica,@kod,@id_miasta)
				END
			ELSE
				BEGIN
					EXEC dbo.Dodaj_Miasto
						@nazwa_miasta,
						@nazwa_panstwa
					INSERT INTO
Klienci_Biz(ID_klienta,Nazwa_firmy,NIP,Ulica,Kod_pocztowy,ID_miasta)
					VALUES (@id,@nazwa_firmy,@nip,@ulica,@kod,@@IDENTITY)
				END
		COMMIT TRAN Dodaj_Klienta_Biz
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Klienta_Biz
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania klienta biznesowego: '+
ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Pracownika_Firmy - rejestruje klienta indywidualnego w firmie, do której przynależy, jeśli pracownika nie ma w bazie, to jest on automatycznie dodawany jako klient

```
CREATE PROCEDURE [dbo].[Dodaj_Pracownika_Firmy]
	@imie_pracownika VARCHAR(30),
	@nazwisko_pracownika VARCHAR(30),
	@telefon_pracownika VARCHAR(9),
	@email_pracownika VARCHAR(50),
	@email_firmy VARCHAR(50)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		IF NOT EXISTS (SELECT b.ID_klienta FROM Klienci_Biz b
					INNER JOIN Klienci k ON k.ID_klienta=b.ID_klienta
					WHERE @email_firmy=k.Email)
		BEGIN
			;THROW 52000, 'Nie ma takiej firmy',1
		END
	 BEGIN TRAN Dodaj_Pracownika_Firmy
		  IF NOT EXISTS (SELECT i.ID_klienta FROM Klienci_Ind i
					INNER JOIN Klienci k ON k.ID_klienta=i.ID_klienta
					WHERE @email_pracownika=k.Email)
```

```
                        BEGIN
                                EXEC dbo.Dodaj_Klienta_Ind
                                        @imie_pracownika,
                                        @nazwisko_pracownika,
                                        @email_pracownika,
                                        @telefon_pracownika
                        END
                        DECLARE @id_firmy INT = (SELECT ID_klienta FROM Klienci WHERE
Email=@email_firmy)
                        DECLARE @id_pracownika INT = (SELECT ID_klienta FROM Klienci WHERE
Email=@email_pracownika)
                        INSERT INTO Pracownicy_Firm(ID_firmy,ID_pracownika)
                        VALUES (@id_firmy,@id_pracownika)
        COMMIT TRAN Dodaj_Pracownika_Firmy
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN Dodaj_Pracownika_Firmy
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania pracownika firmy: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Lokal - rejestracja restauracji w systemie

```
CREATE PROCEDURE [dbo].[Dodaj_Lokal]
       @nazwa_lokalu VARCHAR(50),
       @ulica VARCHAR(50),
       @nazwa_miasta VARCHAR(50),
       @nazwa_panstwa VARCHAR(50)
AS
BEGIN
       SET NOCOUNT ON;
       BEGIN TRY
              IF NOT EXISTS(
                     SELECT * FROM Miasta WHERE Nazwa_miasta=@nazwa_miasta
              )
              BEGIN
                     EXEC Dodaj_Miasto
                            @nazwa_miasta,
                            @nazwa_panstwa
              END
              DECLARE @id_miasta INT = (SELECT ID_miasta FROM Miasta WHERE
@nazwa_miasta=Nazwa_miasta)
              INSERT INTO Restauracje(Nazwa,Ulica,Miasto) VALUES (@nazwa_lokalu,@ulica,@id_miasta)
       END TRY
       BEGIN CATCH
              DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania restauracji: '+ ERROR_MESSAGE () ;
           THROW 52000 , @errorMsg ,1;
       END CATCH
END
GO
```

Dodaj_Pracownika - rejestracja pracownika restauracji w systemie

```
CREATE PROCEDURE [dbo].[Dodaj_Pracownika]
       @imie VARCHAR(30),
       @nazwisko VARCHAR(30),
       @nazwa_restauracji VARCHAR(50),
       @ulica VARCHAR(50),
       @miasto VARCHAR(50)
AS
BEGIN
       SET NOCOUNT ON;
       BEGIN TRY
              IF NOT EXISTS(
                     SELECT * FROM Restauracje r
                     INNER JOIN Miasta m
```

```
                ON m.ID_miasta=r.Miasto
                WHERE @nazwa_restauracji=r.Nazwa
                AND @ulica=r.Ulica
                AND @miasto=m.Nazwa_miasta
        )
        BEGIN
                ;THROW 52000, 'Nie ma takiej restauracji',1
        END
        DECLARE @id_lokalu INT = (
                SELECT r.ID_restauracji FROM Restauracje r
                INNER JOIN Miasta m
                ON m.ID_miasta=r.Miasto
                WHERE @nazwa_restauracji=r.Nazwa
                AND @ulica=r.Ulica
                AND @miasto=m.Nazwa_miasta
        )
        INSERT INTO Obsluga(Imię,Nazwisko,ID_Restauracji)
        VALUES (@imie,@nazwisko,@id_lokalu)
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania pracownika: '+ ERROR_MESSAGE () ;
      THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Stolik - dodanie stolika z danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Stolik]
    @max_liczba_miejsc INT,
    @nazwa_lokalu VARCHAR(50),
    @ulica VARCHAR(50),
    @miasto VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @id_restauracji INT = (
            SELECT r.ID_restauracji FROM Restauracje r
            INNER JOIN Miasta m ON r.Miasto=m.ID_miasta
            WHERE @nazwa_lokalu=r.Nazwa
            AND @ulica=r.Ulica
            AND @miasto=m.Nazwa_miasta)
        INSERT INTO Stoliki(Max_liczba_miejsc,ID_Restauracji)
        VALUES (@max_liczba_miejsc,@id_restauracji)
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania stolika: '+ ERROR_MESSAGE () ;
      THROW 52000 , @errorMsg ,1;
    END CATCH
END
GO
```

Dodaj_Obostrzenie – dodanie nowego obostrzenia dla wskazanego stolika

```
CREATE PROCEDURE [dbo].[Dodaj_Obostrzenie]
    @id_stolika INT,
    @liczba_miejsc INT,
    @data_wprowadzenia DATE=NULL
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT * FROM Stoliki WHERE ID_stolika=@id_stolika
        )
```

```
		BEGIN
			;THROW 52000, 'Nie istnieje taki stolik',1
		END
		IF @liczba_miejsc>(SELECT Max_liczba_miejsc FROM Stoliki WHERE ID_stolika=@id_stolika)
		BEGIN
			;THROW 52000, 'Nie mozna dac wiecej miejsc niz jest przy stoliku bez
obostrzen',1
		END
		IF @liczba_miejsc<0
		BEGIN
			;THROW 52000, 'Liczba miejsc nie moze byc ujemna',1
		END
		IF @data_wprowadzenia is null
		BEGIN
			SET @data_wprowadzenia=GETDATE()
		END
		INSERT INTO Obostrzenia(ID_stolika,Liczba_miejsc,Data_wprowadzenia)
		VALUES (@id_stolika,@liczba_miejsc,@data_wprowadzenia)
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg nvarchar (2048) = 'Blad dodania obostrzenia na stolik: '+
ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Dostawe – dodanie nowej dostawy

```
CREATE PROCEDURE [dbo].[Dodaj_Dostawe]
	@nazwa_restauracji VARCHAR(50),
	@nazwa_dostawcy VARCHAR(50),
	@data_zamowienia DATE
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	BEGIN TRAN Dodaj_Dostawe
			IF NOT EXISTS (SELECT * FROM Dostawcy WHERE @nazwa_dostawcy=Nazwa_firmy)
			BEGIN
				;THROW 52000, 'Nie istnieje taki dostawca',1
			END
			IF NOT EXISTS (SELECT * FROM Restauracje WHERE @nazwa_restauracji=Nazwa)
			BEGIN
				;THROW 52000, 'Nie istnieje taka restauracja',1
			END
			DECLARE @id_restauracji INT = (SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
			DECLARE @id_dostawcy INT = (SELECT ID_dostawcy FROM Dostawcy WHERE
Nazwa_firmy=@nazwa_dostawcy)
			INSERT INTO Dostawy(ID_dostawcy,Data_zamówienia,Data_dostawy,ID_Restauracji)
			VALUES (@id_dostawcy,@data_zamowienia,NULL,@id_restauracji)
	COMMIT TRAN Dodaj_Dostawe
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Dostawe
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodawania dostawy: '+ ERROR_MESSAGE () ;
	THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Produkt_Dostawy – dodanie elementu do wskazanej dostawy

```
CREATE PROCEDURE [dbo].[Dodaj_Produkt_Dostawy]
	@id_dostawy INT,
	@nazwa_polproduktu VARCHAR(50),
```

```
	@ilosc_jednostek FLOAT
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	BEGIN TRAN Dodaj_Produkt_Dostawy
			IF NOT EXISTS (SELECT * FROM Dostawy WHERE @id_dostawy=ID_dostawy)
			BEGIN
				;THROW 52000, 'Nie istnieje taka dostawa',1
			END
			IF NOT EXISTS (SELECT * FROM Polprodukty WHERE @nazwa_polproduktu=Nazwa)
			BEGIN
				;THROW 52000, 'Nie istnieje taki polprodukt',1
			END
			DECLARE @id_produktu INT =(SELECT ID_półproduktu FROM Polprodukty WHERE
@nazwa_polproduktu=Nazwa)
			IF @ilosc_jednostek<=0
			BEGIN
				;THROW 52000, 'Liczba dostarczanych jednostek musi byc wartoscia
dodatnia',1
			END
			INSERT INTO Szczegóły_Dostaw(ID_dostawy,ID_półproduktu,Ilość_jednostek)
			VALUES (@id_dostawy,@id_produktu,@ilosc_jednostek)
	COMMIT TRAN Dodaj_Produkt_Dostawy
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Produkt_Dostawy
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodawania elementu do dostawy: '+
ERROR_MESSAGE () ;
	THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Odbierz_Dostawe – odebranie wskazanej dostawy i aktualizacja stanu magazynu półproduktów

```
CREATE PROCEDURE [dbo].[Odbierz_Dostawe]
	@id_dostawy INT,
	@nazwa_restauracji VARCHAR(50),
	@data_dostawy DATE
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	BEGIN TRAN Odbierz_Dostawe
			IF NOT EXISTS (SELECT * FROM Dostawy WHERE @id_dostawy=ID_dostawy)
			BEGIN
				;THROW 52000, 'Nie istnieje taka dostawa',1
			END
			IF NOT EXISTS (SELECT * FROM Restauracje WHERE @nazwa_restauracji=Nazwa)
			BEGIN
				;THROW 52000, 'Nie istnieje taka restauracja',1
			END
			IF @data_dostawy<(SELECT Data_zamówienia FROM Dostawy WHERE
@id_dostawy=ID_dostawy)
			BEGIN
				;THROW 52000, 'Data odbioru nie moze byc wczesniejsza niz zamowienia',1
			END
			UPDATE Dostawy SET Data_dostawy=@data_dostawy WHERE ID_dostawy=@id_dostawy
			DECLARE @id_restauracji INT = (SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
			DECLARE iter CURSOR
			FOR
				SELECT ID_półproduktu
				FROM Szczegóły_Dostaw
				WHERE @id_dostawy=ID_dostawy
			DECLARE @id_polproduktu INT
			OPEN iter
			FETCH NEXT FROM iter INTO @id_polproduktu
```

```
                    WHILE @@FETCH_STATUS = 0
                    BEGIN
                            DECLARE @nazwa_polproduktu VARCHAR(50)=(SELECT Nazwa FROM Polprodukty
WHERE @id_polproduktu=ID_półproduktu)
                            DECLARE @ilosc_jednostek FLOAT =(SELECT Ilość_jednostek FROM
Szczegóły_Dostaw WHERE ID_dostawy=@id_dostawy
                                    AND @id_polproduktu=ID_półproduktu)
                            EXEC Dodaj_Do_Magazynu
                                    @nazwa_restauracji,
                                    @nazwa_polproduktu,
                                    @ilosc_jednostek
                            FETCH NEXT FROM iter INTO @id_polproduktu
                    END
                    CLOSE iter
                    DEALLOCATE iter
                    COMMIT TRAN Odbierz_Dostawe
        END TRY
        BEGIN CATCH
                ROLLBACK TRAN Odbierz_Dostawe
                DECLARE @errorMsg NVARCHAR (2048) = 'Blad przy odbiorze dostawy: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Dostawce – rejestracja nowego dostawcy w systemie

```
CREATE PROCEDURE [dbo].[Dodaj_Dostawce]
        @nazwa_firmy VARCHAR(50),
        @ulica VARCHAR(50),
        @kod VARCHAR(6),
        @miasto VARCHAR(50),
        @panstwo VARCHAR(50),
        @telefon VARCHAR(9)
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
                BEGIN TRAN Dodaj_Dostawce
                        IF EXISTS (SELECT * FROM Dostawcy WHERE Telefon_kontaktowy=@telefon) OR
                                EXISTS (SELECT * FROM Klienci WHERE Telefon_kontaktowy=@telefon)
                        BEGIN
                                ;THROW 52000, 'Telefon musi byc unikalny',1
                        END
                        IF NOT EXISTS (SELECT * FROM Miasta WHERE Nazwa_miasta=@miasto)
                        BEGIN
                                EXEC Dodaj_Miasto
                                        @miasto,
                                        @panstwo
                        END
                        DECLARE @id_miasta INT =(SELECT ID_miasta FROM Miasta WHERE
Nazwa_miasta=@miasto)
                        INSERT INTO
Dostawcy(Nazwa_firmy,Ulica,Kod_pocztowy,ID_miasta,Telefon_kontaktowy)
                        VALUES (@nazwa_firmy,@ulica,@kod,@id_miasta,@telefon)
                COMMIT TRAN Dodaj_Dostawce
        END TRY
        BEGIN CATCH
                ROLLBACK TRAN Dodaj_Dostawce
                DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania dostawcy: '+ ERROR_MESSAGE () ;
            THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Do_Menu - wprowadzenie dania do menu danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Do_Menu]
	@nazwa_dania varchar(50),
	@data_wprowadzenia date,
	@nazwa_restauracji varchar(50)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	BEGIN TRAN Dodaj_Do_Menu
			IF NOT EXISTS (SELECT * FROM Dania WHERE Nazwa_dania=@nazwa_dania)
			BEGIN
				;THROW 52000, 'Nie ma takiego dania',1
			END
			IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
			BEGIN
				;THROW 52000, 'Nie ma takiej restauracji',1
			END
			DECLARE @id_dania int = (SELECT ID_dania FROM Dania WHERE
Nazwa_dania=@nazwa_dania)
			DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
			IF NOT EXISTS (SELECT * FROM Menu WHERE @id_dania=ID_dania AND
ID_restauracji=@id_restauracji)
			BEGIN
				INSERT INTO Menu(ID_dania,Data_wprowadzenia,ID_restauracji)
				VALUES (@id_dania,@data_wprowadzenia,@id_restauracji)
			END
			ELSE IF (SELECT dbo.Ostatnie_Usuniecie_Z_Menu(@id_dania,@id_restauracji)) IS
NULL
			BEGIN
				;THROW 52000, 'Nie mozna dodac do menu, poniewaz jest w menu',1
			END
			ELSE IF (SELECT
dbo.Ostatnie_Usuniecie_Z_Menu(@id_dania,@id_restauracji))>@data_wprowadzenia
			BEGIN
				;THROW 52000, 'Danie nie moze zostac znow dodane przed data jego
ostatniego usuniecia',1
			END
			ELSE IF (DATEDIFF(DAY,(SELECT
dbo.Ostatnie_Usuniecie_Z_Menu(@id_dania,@id_restauracji)),@data_wprowadzenia)<30)
			BEGIN
				;THROW 52000, 'Nie mozna dodac do menu, poniewaz nie minal miesiac od
zdjecia',1
			END
			ELSE
			BEGIN
				INSERT INTO Menu(ID_dania,Data_wprowadzenia,ID_restauracji)
				VALUES (@id_dania,@data_wprowadzenia,@id_restauracji)
			END
	COMMIT TRAN Dodaj_Do_Menu
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Do_Menu
		DECLARE @errorMsg nvarchar (2048) = 'Blad dodania do menu: '+ ERROR_MESSAGE () ;
	THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

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

```
CREATE PROCEDURE [dbo].[Usun_Polowe_Pozycji_Z_Menu]
@id_restauracji int,
@data date
AS
BEGIN
SET NOCOUNT ON;

UPDATE Menu SET Data_zdjęcia=GETDATE() WHERE ID_pozycji IN(
SELECT TOP 50 PERCENT a.ID_pozycji FROM dbo.Pokaz_Menu_Dnia(@id_restauracji,DATEADD(day,-1,@data)) a
INNER JOIN dbo.Pokaz_Menu_Dnia(@id_restauracji,DATEADD(day,-14,@data)) b ON a.ID_pozycji =
b.ID_pozycji
ORDER BY a.Data_wprowadzenia ASC)
END
GO
```

Usun_Z_Menu - dodaje datę zdjęcia dania w tabeli Menu

```
CREATE PROCEDURE [dbo].[Usun_Z_Menu]
	@nazwa_dania varchar(50),
	@nazwa_restauracji varchar(50),
	@data_zdjecia date
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	 BEGIN TRAN Usun_Z_Menu
			DECLARE @id_dania int = (SELECT ID_dania FROM Dania WHERE
Nazwa_dania=@nazwa_dania)
			DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
			IF NOT EXISTS (SELECT * FROM Menu WHERE @id_dania=ID_dania AND
ID_restauracji=@id_restauracji AND Data_zdjęcia IS NULL)
			BEGIN
				;THROW 52000, 'Nie ma takiej pozycji obecnie w Menu dla tej
restauracji',1
			END
			DECLARE @id_pozycji int = (SELECT ID_pozycji FROM Menu WHERE
ID_restauracji=@id_restauracji
                                                    AND ID_dania=@id_dania AND
Data_zdjęcia IS NULL)
			IF @data_zdjecia<GETDATE()
			BEGIN
				;THROW 52000, 'Data zdjecia nie moze byc chwila z przeszlosci',1
			END
			IF @data_zdjecia<(SELECT Data_wprowadzenia FROM Menu WHERE
ID_restauracji=@id_restauracji
                                                    AND ID_dania=@id_dania AND
Data_zdjęcia IS NULL)
			BEGIN
				;THROW 52000, 'Data zdjecia nie moze wczesniejsza niz dodania',1
			END
			UPDATE Menu
			SET Data_zdjęcia=@data_zdjecia

		COMMIT TRAN Usun_Z_Menu
```

```
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Usun_Z_Menu
		DECLARE @errorMsg nvarchar (2048) = 'Blad usuniecia z menu: '+ ERROR_MESSAGE () ;
	 THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

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

```
CREATE PROCEDURE Dodaj_Danie
	@nazwa_dania varchar(50),
    @cena money,
	@nazwa_kategorii varchar(50),
	@opis varchar(255)=null
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	   BEGIN TRAN Dodaj_Danie
			IF NOT EXISTS(SELECT * FROM Kategorie WHERE Nazwa_kategorii=@nazwa_kategorii)
			BEGIN
				INSERT INTO Kategorie(Nazwa_kategorii) VALUES (@nazwa_kategorii)
			END
			DECLARE @id_kategorii int = (SELECT ID_kategorii FROM Kategorie WHERE
Nazwa_kategorii=@nazwa_kategorii)
			IF @cena<=0
			BEGIN
				;THROW 52000, 'Cena musi byc wartoscia dodatnia',1
			END
			INSERT INTO Dania(Nazwa_dania,Cena_dania,Kategoria,Opis_dania)
			VALUES (@nazwa_dania,@cena,@id_kategorii,@opis)
	   COMMIT TRAN Dodaj_Danie
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Danie
		DECLARE @errorMsg nvarchar (2048) = 'Blad dodania nowego dania: '+ ERROR_MESSAGE () ;
	   THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Do_Magazynu - jeśli produkt jest odnotowany w magazynie danej restauracji to zwiększa liczbę jednostek o podaną, natomiast gdy produktu nie ma to wstawia nowy rekord

```
CREATE PROCEDURE Dodaj_Do_Magazynu
	@nazwa_restauracji varchar(50),
	@nazwa_produktu varchar(50),
	@ilosc_jednostek float
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
	   BEGIN TRAN Dodaj_Do_Magazynu
			IF NOT EXISTS (SELECT * FROM Półprodukty WHERE @nazwa_produktu=Nazwa)
			BEGIN
```

```
                        EXEC Dodaj_Polprodukt
                              @nazwa_produktu
                  END
                  IF @ilosc_jednostek<=0
                  BEGIN
                        ;THROW 52000, 'Liczba dodawanych jednostek musi byc wieksza od 0',1
                  END
                  IF NOT EXISTS (SELECT * FROM Restauracje WHERE @nazwa_restauracji=Nazwa)
                  BEGIN
                        ;THROW 52000, 'Nie istnieje taka restauracja',1
                  END
                  DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
                  DECLARE @id_produktu int = (SELECT ID_półproduktu FROM Półprodukty WHERE
Nazwa=@nazwa_produktu)
                  IF EXISTS(SELECT * FROM Stan_Magazynowy WHERE ID_restauracji=@id_restauracji
                  AND @id_produktu=ID_półproduktu)
                  BEGIN
                        UPDATE Stan_Magazynowy SET
Stan_magazynowy=Stan_magazynowy+@ilosc_jednostek
                        WHERE ID_restauracji=@id_restauracji AND @id_produktu=ID_półproduktu
                  END
                  ELSE
                  BEGIN
                        INSERT INTO
Stan_Magazynowy(ID_półproduktu,ID_restauracji,Stan_magazynowy)
                        VALUES (@id_produktu,@id_restauracji,@ilosc_jednostek)
                  END
            COMMIT TRAN Dodaj_Do_Magazynu
      END TRY
      BEGIN CATCH
            ROLLBACK TRAN Dodaj_Do_Magazynu
            DECLARE @errorMsg nvarchar (2048) = 'Blad zwiekszenia ilosci jednostek w magazynie: '+
ERROR_MESSAGE () ;
            THROW 52000 , @errorMsg ,1;
      END CATCH
END
GO
```

Dodaj_Polprodukt - wstawia nowy składnik dań do bazy

```
CREATE PROCEDURE Dodaj_Polprodukt
      @nazwa_produktu varchar(50)
AS
BEGIN
      SET NOCOUNT ON;
      BEGIN TRY
            BEGIN TRAN Dodaj_Polprodukt
                  IF EXISTS (SELECT * FROM Półprodukty WHERE Nazwa=@nazwa_produktu)
                  BEGIN
                        ;THROW 52000, 'Półprodukt o takiej nazwie jest już wpisany',1
                  END
                  INSERT INTO Półprodukty(Nazwa) VALUES (@nazwa_produktu)
            COMMIT TRAN Dodaj_Polprodukt
      END TRY
      BEGIN CATCH
            ROLLBACK TRAN Dodaj_Polprodukt
            DECLARE @errorMsg nvarchar (2048) = 'Blad przy dodawaniu polproduktu: '+ ERROR_MESSAGE
() ;
```

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

Dodaj_Element_Przepisu - notuje składnik danego dania wraz z ilością potrzebną na wykonanie porcji; ułatwia wstawianie półproduktu - jeśli jest nowy, nie odnotowany w bazie to go wstawia do słownika

```
CREATE PROCEDURE Dodaj_Element_Przepisu
	@nazwa_dania varchar(50),
	@nazwa_produktu varchar(50),
	@potrzebna_ilosc float
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Skladnik
			IF NOT EXISTS (SELECT * FROM Dania WHERE Nazwa_dania=@nazwa_dania)
			BEGIN
				;THROW 52000, 'Nie ma takiego dania',1
			END
			IF @potrzebna_ilosc<=0
			BEGIN
				;THROW 52000, 'Potrzebna ilosc skladnika musi byc liczba dodatnia',1
			END
			IF NOT EXISTS (SELECT * FROM Półprodukty WHERE Nazwa=@nazwa_produktu)
			BEGIN
				EXEC Dodaj_Polprodukt
					@nazwa_produktu
			END
			DECLARE @id_dania int =(SELECT ID_dania FROM Dania WHERE
@nazwa_dania=Nazwa_dania)
			DECLARE @id_polproduktu int = (SELECT ID_półproduktu FROM Półprodukty WHERE
@nazwa_produktu=Nazwa)
			INSERT INTO Przepisy (ID_dania,ID_półproduktu,Potrzebna_ilość)
			VALUES(@id_dania,@id_polproduktu,@potrzebna_ilosc)
		COMMIT TRAN Dodaj_Skladnik
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Skladnik
		DECLARE @errorMsg nvarchar (2048) = 'Blad dodania skladnika: '+ ERROR_MESSAGE () ;
		THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Zamowienie – dodanie nowego zamówienia do systemu

```
CREATE PROCEDURE [dbo].[Dodaj_Zamowienie]
	@nazwa_restauracji VARCHAR(50),
	@id_klienta INT,
	@data_zamowienia DATETIME,
	@data_odbioru DATETIME =NULL,
	@na_wynos VARCHAR(1),
	@id_pracownika INT
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Zamowienie
```

```
	IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
	BEGIN
		;THROW 52000, 'Nie ma takiej restauracji',1
	END
	DECLARE @id_restauracji INT = (SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
	IF NOT EXISTS (SELECT * FROM Klienci WHERE @id_klienta=ID_klienta)
	BEGIN
		;THROW 52000, 'Nie ma takiego klienta',1
	END
	IF @data_odbioru IS NULL AND @na_wynos='T'
	BEGIN
		;THROW 52000, 'Przy zamawianiu na wynos trzeba podac date odbioru',1
	END
	IF @data_odbioru is NULL
	BEGIN
		SET @data_odbioru=DATEADD(hh,1,@data_zamowienia)
	END
	IF NOT EXISTS (SELECT * FROM Obsluga WHERE ID_pracownika=@id_pracownika AND
ID_Restauracji=@id_restauracji)
	BEGIN
		;THROW 52000, 'Zamowienie obsluguje nieuprawniona osoba',1
	END
	DELETE FROM Aktualnie_Przyznane_Rabaty WHERE @id_klienta=ID_klienta AND
Data_wygaśnięcia IS NOT NULL AND Data_wygaśnięcia<GETDATE()
	DELETE FROM Aktualnie_Przyznane_Rabaty WHERE @id_klienta=ID_klienta AND
ID_rabatu IN(
		SELECT ID_rabatu FROM Rabaty WHERE Data_zdjęcia IS NOT NULL)
	IF @id_klienta IN (SELECT ID_klienta FROM Klienci_Ind)
	BEGIN
		EXEC Aktualizuj_Rabat_Ind_Staly
			@id_klienta,
			@id_restauracji
		EXEC Aktualizuj_Rabat_Ind_Jednorazowy
			@id_klienta,
			@id_restauracji
	END
	ELSE
	BEGIN
		EXEC Aktualizuj_Rabat_Firm_Miesieczny
			@id_klienta,
			@id_restauracji
		EXEC Aktualizuj_Rabat_Firm_Kwartalny
			@id_klienta,
			@id_restauracji
	END
	INSERT INTO
Zamówienia(ID_klienta,Data_zamówienia,Data_odbioru,Na_wynos,Pracownik_obsługujący)
	VALUES (@id_klienta,@data_zamowienia,@data_odbioru,@na_wynos,@id_pracownika)
	DECLARE iter CURSOR
	FOR
		SELECT a.ID_rabatu
		FROM Aktualnie_Przyznane_Rabaty a
		INNER JOIN Rabaty r ON r.ID_rabatu=a.ID_rabatu
		WHERE @id_klienta=a.ID_klienta AND r.ID_Restauracji=@id_restauracji
	DECLARE @rabat int
	OPEN iter
	FETCH NEXT FROM iter INTO @rabat
	WHILE @@FETCH_STATUS = 0
	BEGIN
		DECLARE @data_wygasniecia date=(SELECT Data_wygaśnięcia FROM
Aktualnie_Przyznane_Rabaty
		WHERE @rabat=ID_rabatu AND @id_klienta=ID_klienta)
		IF @data_wygasniecia<GETDATE()
		BEGIN
			EXEC Odbierz_Rabat_Klientowi
				@rabat,
				@id_klienta
		END
```

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

Dodaj_Element_Do_Zamowienia – dodanie nowego elementu do istniejącego zamówienia

```
CREATE PROCEDURE [dbo].[Dodaj_Element_Do_Zamowienia]
        @id_zamowienia INT,
        @nazwa_restauracji VARCHAR(50),
        @nazwa_dania VARCHAR(50),
        @ilosc INT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        BEGIN TRAN Dodaj_Element
                    IF @nazwa_restauracji <> (
                            SELECT r.Nazwa FROM Restauracje r
                            INNER JOIN Obsluga o ON o.ID_Restauracji=r.ID_restauracji
                            INNER JOIN Zamówienia z ON z.Pracownik_obsługujący=o.ID_pracownika
                            WHERE @id_zamowienia=z.ID_zamówienia)
                    BEGIN
                            ;THROW 52000, 'Proba dostania sie do danych innego lokalu',1
                    END
                    IF NOT EXISTS (SELECT * FROM Zamówienia WHERE ID_zamówienia=@id_zamowienia)
                    BEGIN
                            ;THROW 52000, 'Nie ma takiego zamowienia',1
                    END
                    IF NOT EXISTS (SELECT * FROM Restauracje WHERE @nazwa_restauracji=Nazwa)
                    BEGIN
                            ;THROW 52000, 'Nie istnieje taka restauracja',1
                    END
                    IF NOT EXISTS (SELECT * FROM Dania WHERE @nazwa_dania=Nazwa_dania)
                    BEGIN
                            ;THROW 52000, 'Nie ma takiego dania',1
                    END
                    IF @ilosc<=0
                    BEGIN
                            ;THROW 52000, 'Trzeba zamowic przynajmniej jedna sztuke',1
                    END
                    DECLARE @id_dania int = (SELECT ID_dania FROM Dania WHERE
Nazwa_dania=@nazwa_dania)
                    DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
                    IF (SELECT Kategoria FROM Dania WHERE @id_dania=ID_dania) = (SELECT
ID_kategorii FROM Kategorie WHERE Nazwa_kategorii='Seafood')
                    BEGIN
                            DECLARE @data_odbioru date =(SELECT Data_odbioru FROM Zamówienia WHERE
@id_zamowienia=ID_zamówienia)
                            DECLARE @data_zamowienia date =(SELECT Data_zamówienia FROM Zamówienia
WHERE @id_zamowienia=ID_zamówienia)
                            IF(DATEPART(w,@data_odbioru)=5 AND
DATEDIFF(day,@data_zamowienia,@data_odbioru)<3)
                            BEGIN
                                    ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce
morza',1
                            END
                            IF(DATEPART(w,@data_odbioru)=6 AND
DATEDIFF(day,@data_zamowienia,@data_odbioru)<4)
```

```
                        BEGIN
                                ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce
morza',1
                        END
                        IF(DATEPART(w,@data_odbioru)=7 AND
DATEDIFF(DAY,@data_zamowienia,@data_odbioru)<5)
                        BEGIN
                                ;THROW 52000, 'Niespelnione warunki na danie z kategorii owoce
morza',1
                        END
                END
                IF NOT EXISTS (SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania
                AND ID_restauracji=@id_restauracji AND Data_zdjęcia is NULL)
                BEGIN
                        ;THROW 52000, 'To danie nie jest obecnie w ofercie restauracji',1
                END
                DECLARE @id_pozycji int =(SELECT ID_pozycji FROM Menu WHERE ID_dania=@id_dania
                AND ID_restauracji=@id_restauracji AND Data_zdjęcia is NULL)
                DECLARE iter CURSOR
                FOR
                        SELECT ID_półproduktu
                        FROM Przepisy
                        WHERE @id_dania=ID_dania
                DECLARE @id_polproduktu INT
                OPEN iter
                FETCH NEXT FROM iter INTO @id_polproduktu
                WHILE @@FETCH_STATUS = 0
                BEGIN
                        DECLARE @nazwa_polproduktu VARCHAR(50)=(SELECT Nazwa FROM Polprodukty
WHERE @id_polproduktu=ID_półproduktu)
                        PRINT @nazwa_polproduktu
                        DECLARE @ilosc_jednostek FLOAT =(SELECT Potrzebna_ilość FROM Przepisy
WHERE ID_dania=@id_dania
                                AND @id_polproduktu=ID_półproduktu)*@ilosc
                        EXEC Pobierz_Z_Magazynu
                                @nazwa_restauracji,
                                @nazwa_polproduktu,
                                @ilosc_jednostek
                        FETCH NEXT FROM iter INTO @id_polproduktu
                END
                CLOSE iter
                DEALLOCATE iter
                DECLARE @rabat FLOAT=0.0
                DECLARE @id_klienta INT =(SELECT ID_klienta FROM Zamówienia WHERE
ID_zamówienia=@id_zamowienia)
                IF @id_klienta IN (SELECT ID_klienta FROM Klienci_Ind)
                BEGIN
                        SET
@rabat=@rabat+dbo.Nalicz_Rabat_Ind_Staly(@id_restauracji,@id_klienta)+dbo.Nalicz_Rabat_Ind_Jednorazo
wy(@id_restauracji,@id_klienta)
                        IF @id_klienta IN (SELECT ID_pracownika FROM Pracownicy_Firm)
                        BEGIN
                                DECLARE @id_firmy INT = (SELECT TOP 1 ID_firmy FROM
Pracownicy_Firm WHERE ID_pracownika=@id_klienta)
                                SET
@rabat=@rabat+dbo.Nalicz_Rabat_Firm_Miesieczny(@id_restauracji,@id_firmy)+dbo.Nalicz_Rabat_Firm_Kwar
talny(@id_restauracji,@id_firmy)
                        END
                END
                ELSE
                BEGIN
                        SET
@rabat=@rabat+dbo.Nalicz_Rabat_Firm_Miesieczny(@id_restauracji,@id_klienta)+dbo.Nalicz_Rabat_Firm_Kw
artalny(@id_restauracji,@id_klienta)
                END
                IF @rabat>1
                BEGIN
                        SET @rabat=1
                END
```

```
                    DECLARE @cena MONEY = (SELECT Cena_dania FROM Dania WHERE
ID_dania=@id_dania)*(1-@rabat)
                    INSERT INTO Szczegóły_Zamówień(ID_zamówienia,ID_pozycji,Cena_jednostkowa,Ilość)
                    VALUES(@id_zamowienia,@id_pozycji,@cena,@ilosc)
                    COMMIT TRAN Dodaj_Element
        END TRY
        BEGIN CATCH
         ROLLBACK TRAN Dodaj_Element
              DECLARE @errorMsg NVARCHAR (2048) = 'Blad przy dodaniu dania do zamowienia: '+
ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Zamknij_Zamowienie – zamknięcie zamówienia usuwające istniejący rabat jednorazowy

```
CREATE PROCEDURE [dbo].[Zamknij_Zamowienie]
        @id_zamowienia INT
AS
BEGIN
        SET NOCOUNT ON;
        DECLARE @id_klienta INT =(SELECT ID_klienta FROM Zamówienia WHERE
ID_zamówienia=@id_zamowienia)
        DECLARE @id_restauracji INT =(SELECT o.ID_Restauracji FROM Obsluga o
        INNER JOIN Zamówienia z ON z.Pracownik_obsługujący=o.ID_pracownika WHERE
z.ID_zamówienia=@id_zamowienia)
        DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r INNER JOIN Rabaty_Ind_Jednorazowe
ri ON ri.ID_rabatu=r.ID_rabatu
        WHERE r.Data_zdjęcia IS NULL AND r.ID_Restauracji=@id_restauracji)
        IF @id_rabatu IS NOT NULL
        BEGIN
              DELETE FROM Aktualnie_Przyznane_Rabaty WHERE ID_rabatu=@id_rabatu AND
ID_klienta=@id_klienta
        END
END
GO
```

Pobierz_Z_Magazynu – pobranie odpowiedniej wartości półproduktu z magazynu

```
CREATE PROCEDURE [dbo].[Pobierz_Z_Magazynu]
        @nazwa_restauracji VARCHAR(50),
        @nazwa_produktu VARCHAR(50),
        @ilosc_jednostek FLOAT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        BEGIN TRAN Pobierz_Z_Magazynu
                    IF NOT EXISTS (SELECT * FROM Polprodukty WHERE @nazwa_produktu=Nazwa)
                    BEGIN
                          ;THROW 52000, 'Nie istnieje taki polprodukt',1
                    END
                    IF NOT EXISTS (SELECT * FROM Restauracje WHERE @nazwa_restauracji=Nazwa)
                    BEGIN
                          ;THROW 52000, 'Nie istnieje taka restauracja',1
                    END
                    DECLARE @id_restauracji INT = (SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
                    DECLARE @id_produktu INT = (SELECT ID_półproduktu FROM Polprodukty WHERE
Nazwa=@nazwa_produktu)
                    IF @ilosc_jednostek<=0
                    BEGIN
                          ;THROW 52000, 'Liczba pobieranych jednostek musi byc wieksza od 0',1
                    END
                    IF NOT EXISTS(
```

```
                        SELECT * FROM Stan_Magazynowy WHERE @id_restauracji=ID_restauracji
                        AND @id_produktu=ID_półproduktu
                )
                BEGIN
                        ;THROW 52000, 'W magazynie nie odnotowano tego produktu do tej pory',1
                END
                IF @ilosc_jednostek>(SELECT Stan_magazynowy FROM Stan_Magazynowy WHERE
@id_restauracji=ID_restauracji
                        AND @id_produktu=ID_półproduktu)
                BEGIN
                        ;THROW 52000, 'Proba pobrania zbyt wielu jednostek',1
                END
                UPDATE Stan_Magazynowy SET Stan_magazynowy=Stan_magazynowy-@ilosc_jednostek
                        WHERE ID_restauracji=@id_restauracji AND @id_produktu=ID_półproduktu
        COMMIT TRAN Pobierz_Z_Magazynu
        END TRY
        BEGIN CATCH
                ROLLBACK TRAN Pobierz_Z_Magazynu
                DECLARE @errorMsg NVARCHAR (2048) = 'Blad pobierania polproduktu z magazynu: '+
ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Rezerwacje_Ind – tworzenie nowej rezerwacji dla klienta indywidualnego

```
CREATE PROCEDURE [dbo].[Dodaj_Rezerwacje_Ind]
        @data_zlozenia DATE,
        @data_rezerwacji DATE,
        @nazwa_restauracji VARCHAR(50),
        @id_klienta INT,
        @id_zamowienia INT,
        @liczba_osob INT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
        BEGIN TRAN Dodaj_Rezerwacje_Ind
                IF @liczba_osob<2
                BEGIN
                        ;THROW 52000, 'Rezerwacja musi byc dla min 2 osob',1
                END
                IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
                BEGIN
                        ;THROW 52000, 'Nie ma takiej restauracji',1
                END
                DECLARE @id_restauracji INT =(SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
                IF NOT EXISTS (SELECT * FROM Zamówienia WHERE @id_zamowienia=ID_zamówienia)
                BEGIN
                        ;THROW 52000, 'Najpierw trzeba zlozyc zamowienie',1
                END
                DECLARE @wart_zam MONEY=(SELECT SUM(Cena_jednostkowa*Ilość) FROM
Szczegóły_Zamówień WHERE ID_zamówienia=@id_zamowienia)
                DECLARE @ilosc_zam INT = (SELECT COUNT(*) FROM Zamówienia z
                INNER JOIN Obsluga o ON o.ID_pracownika = z.Pracownik_obsługujący
                INNER JOIN Restauracje r ON r.ID_restauracji=o.ID_Restauracji
                WHERE @id_klienta=ID_klienta AND @id_restauracji=r.ID_restauracji)-1 -- aby
wykluczyc wlasnie zlozone
                IF @ilosc_zam<5 AND @wart_zam<200
                BEGIN
                        ;THROW 52000, 'Niespelniony warunek zamowienia',1
                END
                IF @ilosc_zam>=5 AND @wart_zam<50
                BEGIN
                        ;THROW 52000, 'Niespelniony warunek zamowienia',1
                END
```

```
                    DECLARE @liczba_wolnych_miejsc INT =
dbo.Liczba_Wolnych_Miejsc(@id_restauracji,@data_rezerwacji)
                    IF @liczba_wolnych_miejsc<@liczba_osob
                    BEGIN
                           ;THROW 52000, 'Zbyt malo wolnych miejsc',1
                    END
                    INSERT INTO Rezerwacje(Data_złożenia,Data_rezerwacji,ID_Restauracji)
                    VALUES (@data_zlozenia,@data_rezerwacji,@id_restauracji)
                    DECLARE @id INT=@@IDENTITY
                    INSERT INTO Rezerwacje_Ind(ID_rezerwacji,ID_klienta,ID_zamówienia)
                    VALUES (@id,@id_klienta,@id_zamowienia)
                    DECLARE @liczba_usadzonych INT = 0
                    WHILE @liczba_usadzonych<@liczba_osob
                    BEGIN
                           DECLARE @potwierdzenie_stolika INT =
dbo.Pobierz_Obostrzenie_Do_Rezerwacji(@id_restauracji,@data_rezerwacji)
                           DECLARE @zajete INT = (SELECT Liczba_miejsc FROM Obostrzenia WHERE
ID_Obostrzenia=@potwierdzenie_stolika)
                           SET @liczba_usadzonych=@liczba_usadzonych+@zajete
                           INSERT INTO Szczegóły_Rezerwacji(ID_rezerwacji,ID_obostrzenia)
                           VALUES (@id,@potwierdzenie_stolika)
                    END
       COMMIT TRAN Dodaj_Rezerwacje_Ind
       END TRY
       BEGIN CATCH
              ROLLBACK TRAN Dodaj_Rezerwacje_Ind
              DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodawania rezerwacji: '+ ERROR_MESSAGE () ;
       THROW 52000 , @errorMsg ,1;
       END CATCH
END
GO
```

Dodaj_Rezerwacje_Firm – tworzenie nowej rezerwacji dla klienta firmowego

```
CREATE PROCEDURE [dbo].[Dodaj_Rezerwacje_Firm]
       @data_zlozenia DATE,
       @data_rezerwacji DATE,
       @nazwa_restauracji VARCHAR(50),
       @id_klienta INT,
       @liczba_osob INT
AS
BEGIN
       SET NOCOUNT ON;
       BEGIN TRY
       BEGIN TRAN Dodaj_Rezerwacje_Firm
                    IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
                    BEGIN
                           ;THROW 52000, 'Nie ma takiej restauracji',1
                    END
                    DECLARE @id_restauracji INT =(SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
                    DECLARE @liczba_wolnych_miejsc INT =
dbo.Liczba_Wolnych_Miejsc(@id_restauracji,@data_rezerwacji)
                    IF @liczba_wolnych_miejsc<@liczba_osob
                    BEGIN
                           ;THROW 52000, 'Zbyt malo wolnych miejsc',1
                    END
                    INSERT INTO Rezerwacje(Data_złożenia,Data_rezerwacji,ID_Restauracji)
                    VALUES (@data_zlozenia,@data_rezerwacji,@id_restauracji)
                    DECLARE @id INT=@@IDENTITY
                    INSERT INTO Rezerwacje_Firm(ID_rezerwacji,ID_firmy)
                    VALUES (@id,@id_klienta)
                    DECLARE @liczba_usadzonych INT = 0
                    WHILE @liczba_usadzonych<@liczba_osob
                    BEGIN
                           DECLARE @potwierdzenie_stolika INT =
dbo.Pobierz_Obostrzenie_Do_Rezerwacji(@id_restauracji,@data_rezerwacji)
                           DECLARE @zajete INT = (SELECT Liczba_miejsc FROM Obostrzenia WHERE
ID_Obostrzenia=@potwierdzenie_stolika)
```

```
                        SET @liczba_usadzonych=@liczba_usadzonych+@zajete
                        INSERT INTO Szczegóły_Rezerwacji(ID_rezerwacji,ID_obostrzenia)
                        VALUES (@id,@potwierdzenie_stolika)
                END
        COMMIT TRAN Dodaj_Rezerwacje_Firm
        END TRY
        BEGIN CATCH
                ROLLBACK TRAN Dodaj_Rezerwacje_Firm
                DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodawania rezerwacji: '+ ERROR_MESSAGE () ;
        THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Rezerwacje_Firm_Im – tworzenie nowej rezerwacji dla klienta firmowego – imienne

```
CREATE PROCEDURE [dbo].[Dodaj_Rezerwacje_Firm_Im]
@data_zlozenia datetime,
@data_rezerwacji datetime,
@nazwa_restauracji varchar(50),
@id_firmy int,
@id_pracownika int
AS
BEGIN
SET NOCOUNT ON;
        BEGIN TRY
        BEGIN TRAN Dodaj_Rezerwacje_Firm_Im
IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
BEGIN
        ;THROW 52000, 'Nie ma takiej restauracji',1
END
DECLARE @id_restauracji int =(SELECT ID_restauracji FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
IF NOT EXISTS(SELECT * FROM Rezerwacje WHERE @data_rezerwacji=Data_rezerwacji
AND Data_złożenia=@data_zlozenia AND ID_Restauracji=@id_restauracji)
BEGIN
        INSERT INTO Rezerwacje(Data_złożenia,Data_rezerwacji,ID_Restauracji)
        VALUES (@data_zlozenia,@data_rezerwacji,@id_restauracji)
END
DECLARE @id_rezerwacji int=(SELECT ID_rezerwacji FROM Rezerwacje WHERE
@data_rezerwacji=Data_rezerwacji
AND Data_złożenia=@data_zlozenia AND ID_Restauracji=@id_restauracji)
DECLARE @miejsca_siedzace INT = (SELECT SUM(o.Liczba_miejsc) FROM dbo.Szczegóły_Rezerwacji sr
INNER JOIN obostrzenia o ON o.ID_Obostrzenia=sr.ID_obostrzenia
WHERE ID_rezerwacji=@id_rezerwacji)
DECLARE @aktualnie_przypisani INT = (SELECT COUNT(*) FROM dbo.Rezerwacje_Firm_Imiennie WHERE
        ID_rezerwacji=@id_rezerwacji)
IF @miejsca_siedzace>@aktualnie_przypisani
BEGIN
        INSERT INTO Rezerwacje_Firm_Imiennie(ID_rezerwacji,ID_firmy,ID_pracownika)
        VALUES (@id_rezerwacji,@id_firmy,@id_pracownika)
END
ELSE
BEGIN
        DECLARE @pobrane_obostrzenie int =
        dbo.Pobierz_Obostrzenie_Do_Rezerwacji(@id_restauracji,@data_rezerwacji)
        IF @pobrane_obostrzenie IS NULL
                BEGIN
                EXEC dbo.Anuluj_Rezerwacje @id_rezerwacji = @id_rezerwacji -- int
                ;THROW 52000, 'Nie mozna dodac wiekszej ilosci osob. Brak wolnych miejsc w danym
                terminie',1
        END
        ELSE
```

```
	BEGIN
		INSERT INTO dbo.Szczegóły_Rezerwacji( ID_rezerwacji,ID_obostrzenia)
		VALUES(@id_rezerwacji,@pobrane_obostrzenie)
		INSERT INTO Rezerwacje_Firm_Imiennie(ID_rezerwacji,ID_firmy,ID_pracownika)
		VALUES (@id_rezerwacji,@id_firmy,@id_pracownika)
	END
END
	COMMIT TRAN Dodaj_Rezerwacje_Firm_Im
END TRY
BEGIN CATCH
		ROLLBACK TRAN Dodaj_Rezerwacje_Firm_Im
		DECLARE @errorMsg nvarchar (2048) = 'Blad dodawania rezerwacji: '+ ERROR_MESSAGE () ;
	THROW 52000 , @errorMsg ,1;
END CATCH
END
GO
```

Anuluj_Rezerwacje – anulowanie wskazanej rezerwacji

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

Przyznaj_Rabat_Klientowi – przyznanie klientowi rabatu

```
CREATE PROCEDURE [dbo].[Przyznaj_Rabat_Klientowi]
	@id_rabatu INT,
	@id_klienta INT,
	@data_przyznania DATE=NULL,
	@data_wygasniecia DATE=NULL
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		IF NOT EXISTS (SELECT * FROM Rabaty WHERE ID_rabatu=@id_rabatu)
		BEGIN
			;THROW 52000, 'Nie istnieje taki rabat',1
		END
		DECLARE @data_zdjecia DATE = (SELECT Data_zdjęcia FROM Rabaty WHERE
@id_rabatu=ID_rabatu)
		IF NOT(@data_zdjecia IS NULL OR @data_zdjecia>=GETDATE())
		BEGIN
			;THROW 52000, 'Ten rabat utracil waznosc',1
		END
		IF NOT EXISTS (SELECT * FROM Klienci WHERE ID_klienta=@id_klienta)
		BEGIN
			;THROW 52000, 'Nie istnieje taki klient',1
		END
		IF EXISTS(SELECT * FROM Klienci_Biz WHERE @id_klienta=ID_klienta) AND
		(EXISTS (SELECT * FROM Rabaty_Ind_Jednorazowe WHERE ID_rabatu=@id_rabatu) OR
		EXISTS(SELECT * FROM Rabaty_Ind_Stale WHERE ID_rabatu=@id_rabatu))
		BEGIN
			;THROW 52000, 'Proba przyznania rabatu dla klienta indywidualnego firmie',1
		END
		IF EXISTS(SELECT * FROM Klienci_Ind WHERE @id_klienta=ID_klienta) AND
		NOT EXISTS (SELECT * FROM Rabaty_Ind_Jednorazowe WHERE ID_rabatu=@id_rabatu) AND
```

```
		NOT EXISTS(SELECT * FROM Rabaty_Ind_Stale WHERE ID_rabatu=@id_rabatu)
		BEGIN
			;THROW 52000, 'Proba przyznania rabatu dla firmy klientowi indywidualnemu',1
		END
		IF @data_przyznania IS NULL
		BEGIN
			SET @data_przyznania=GETDATE()
		END
		IF EXISTS(SELECT * FROM Rabaty_Ind_Jednorazowe WHERE ID_rabatu=@id_rabatu)
		BEGIN
			DECLARE @waznosc INT = (SELECT Waznosc FROM Rabaty_Ind_Jednorazowe WHERE
ID_rabatu=@id_rabatu)
			SET @data_wygasniecia=DATEADD(dd,@waznosc,@data_przyznania)
		END
		INSERT INTO
Aktualnie_Przyznane_Rabaty(ID_rabatu,ID_klienta,Data_przyznania,Data_wygaśnięcia)
		VALUES(@id_rabatu,@id_klienta,@data_przyznania,@data_wygasniecia)
	END TRY
	BEGIN CATCH
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad przyznania rabatu: '+ ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Odbierz_Rabat_Klientowi – skasowanie rabatu klientowi

```
CREATE PROCEDURE [dbo].[Odbierz_Rabat_Klientowi]
	@id_rabatu INT,
	@id_klienta INT
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Odbierz_Rabat
			IF NOT EXISTS (SELECT * FROM Rabaty WHERE ID_rabatu=@id_rabatu)
			BEGIN
				;THROW 52000, 'Nie istnieje taki rabat',1
			END
			IF NOT EXISTS (SELECT * FROM Klienci WHERE ID_klienta=@id_klienta)
			BEGIN
				;THROW 52000, 'Nie istnieje taki klient',1
			END
			DELETE FROM Aktualnie_Przyznane_Rabaty WHERE @id_rabatu=ID_rabatu
		COMMIT TRAN Odbierz_Rabat
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Odbierz_Rabat
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad wygaszania rabatu: '+ ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

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

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat]
@wymagana_kwota money,
@wysokosc_jedn float,
@data_wprowadzenia date=null,
@data_zdjecia date=null,
@nazwa_restauracji varchar(50)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
```

```
IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
BEGIN
;THROW 52000, 'Nie ma takiej restauracji',1
END
DECLARE @id_restauracji int=(SELECT ID_restauracji FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
IF @wymagana_kwota<=0
BEGIN
;THROW 52000, 'Kwota musi byc wartoscia dodatnia',1
END
IF NOT @wysokosc_jedn BETWEEN 0 AND 1
BEGIN
;THROW 52000, 'Jednostkowy rabat jest wartoscia z przedzialu od 0 do 1',1
END
IF @data_wprowadzenia is NULL
BEGIN
SET @data_wprowadzenia=GETDATE()
END
INSERT INTO Rabaty(Wymagana_kwota,Wysokosc_jedn,Data_wprowadzenia,Data_zdjęcia,ID_Restauracji)
VALUES (@wymagana_kwota,@wysokosc_jedn,@data_wprowadzenia,@data_zdjecia,@id_restauracji)
PRINT 'Dotarlo'
END TRY
BEGIN CATCH
DECLARE @errorMsg nvarchar (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
    THROW 52000 , @errorMsg ,1;
END CATCH
END
GO
```

Dodaj_Rabat_Ind_Staly – dodanie rabatu stałego dla klientów indywidualnych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Ind_Staly]
	@wymagana_kwota MONEY,
	@wysokosc_jedn FLOAT,
	@data_wprowadzenia DATE=NULL,
	@data_zdjecia DATE=NULL,
	@nazwa_restauracji VARCHAR(50),
	@liczba_zamowien INT
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Rabat_Ind_Staly
			IF @liczba_zamowien<=0
			BEGIN
				;THROW 52000, 'Minimalna ilosc zamowien musi byc liczba dodatnia
calkowita',1
			END
			DECLARE @id_restauracji INT =(SELECT ID_restauracji FROM Restauracje WHERE
@nazwa_restauracji=Nazwa)
			DECLARE @id_poprzedniego INT = (SELECT ri.ID_rabatu FROM Rabaty_Ind_Stale ri
INNER JOIN Rabaty r
                                        ON r.ID_rabatu=ri.ID_rabatu WHERE r.Data_zdjęcia IS NULL
AND @id_restauracji=r.ID_Restauracji)
			IF @id_poprzedniego IS NOT NULL
			BEGIN
				UPDATE Rabaty SET Data_zdjęcia=@data_wprowadzenia WHERE
@id_poprzedniego=ID_rabatu
			END
			EXEC Dodaj_Rabat
				@wymagana_kwota,
				@wysokosc_jedn,
				@data_wprowadzenia,
```

```
                        @data_zdjecia,
                        @nazwa_restauracji
                DECLARE @id_rabatu INT=@@IDENTITY
                INSERT INTO Rabaty_Ind_Stale(ID_rabatu,Liczba_zamowien)
                VALUES (@id_rabatu,@liczba_zamowien)
            COMMIT TRAN Dodaj_Rabat_Ind_Staly
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN Dodaj_Rabat_Ind_Staly
            DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
          THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Rabat_Ind_Jednorazowy - dodanie rabatu jednorazowego dla klientów indywidualnych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Ind_Jednorazowy]
        @wymagana_kwota MONEY,
        @wysokosc_jedn FLOAT,
        @data_wprowadzenia DATE=NULL,
        @data_zdjecia DATE=NULL,
        @nazwa_restauracji VARCHAR(50),
        @waznosc INT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
            BEGIN TRAN Dodaj_Rabat_Ind_Jedn
                IF @waznosc<=0
                BEGIN
                    ;THROW 52000, 'Rabat musi miec waznosc przynajmniej 1 dzien',1
                END
                IF @data_wprowadzenia IS NULL
                BEGIN
                    SET @data_wprowadzenia=GETDATE()
                END
                EXEC Dodaj_Rabat
                    @wymagana_kwota,
                    @wysokosc_jedn,
                    @data_wprowadzenia,
                    @data_zdjecia,
                    @nazwa_restauracji
                DECLARE @id INT = @@IDENTITY
                INSERT INTO Rabaty_Ind_Jednorazowe(ID_rabatu,Waznosc)
                VALUES (@id,@waznosc)
            COMMIT TRAN Dodaj_Rabat_Ind_Jedn
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN Dodaj_Rabat_Ind_Jedn
            DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
          THROW 52000 , @errorMsg ,1;
        END CATCH
END
GO
```

Dodaj_Rabat_Firm_Mies – dodanie rabatu miesięcznego dla klientów biznesowych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Firm_Mies]
        @wymagana_kwota MONEY,
        @wysokosc_jedn FLOAT,
        @data_wprowadzenia DATE=NULL,
        @data_zdjecia DATE=NULL,
        @nazwa_restauracji VARCHAR(50),
        @liczba_zamowien INT,
        @max_rabat FLOAT
AS
```

```
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		BEGIN TRAN Dodaj_Rabat_Firm_Mies
			IF @liczba_zamowien<=0
			BEGIN
				;THROW 52000, 'Minimalna ilosc zamowien musi byc liczba dodatnia
calkowita',1
			END
			IF NOT @max_rabat BETWEEN 0 AND 1
			BEGIN
				;THROW 52000, 'Maksymalny mozliwy rabat musi byc liczba pomiedzy 0 a
1',1
			END
			EXEC Dodaj_Rabat
				@wymagana_kwota,
				@wysokosc_jedn,
				@data_wprowadzenia,
				@data_zdjecia,
				@nazwa_restauracji
			DECLARE @id INT=@@IDENTITY
			INSERT INTO Rabaty_Firm_Miesiac(ID_rabatu,Liczba_zamowien,Max_rabat)
			VALUES (@id,@liczba_zamowien,@max_rabat)
		COMMIT TRAN Dodaj_Rabat_Firm_Mies
	END TRY
	BEGIN CATCH
		ROLLBACK TRAN Dodaj_Rabat_Firm_Mies
		DECLARE @errorMsg NVARCHAR (2048) = 'Blad dodania rabatu: '+ ERROR_MESSAGE () ;
	    THROW 52000 , @errorMsg ,1;
	END CATCH
END
GO
```

Dodaj_Rabat_Firm_Kwartal – dodanie rabatu kwartalnego dla klientów biznesowych w danej restauracji

```
CREATE PROCEDURE [dbo].[Dodaj_Rabat_Firm_Kwartal]
	@wymagana_kwota MONEY,
	@wysokosc_jedn FLOAT,
	@data_wprowadzenia DATE=NULL,
	@data_zdjecia DATE=NULL,
	@nazwa_restauracji VARCHAR(50)
AS
BEGIN
	SET NOCOUNT ON;
	BEGIN TRY
		IF NOT EXISTS (SELECT * FROM Restauracje WHERE Nazwa=@nazwa_restauracji)
		BEGIN
			;THROW 52000, 'Nie ma takiej restauracji',1
		END
		DECLARE @id_restauracji INT=(SELECT ID_restauracji FROM Restauracje WHERE
Nazwa=@nazwa_restauracji)
		IF @wymagana_kwota<=0
		BEGIN
			;THROW 52000, 'Kwota musi byc wartoscia dodatnia',1
		END
		IF NOT @wysokosc_jedn BETWEEN 0 AND 1
		BEGIN
			;THROW 52000, 'Jednostkowy rabat jest wartoscia z przedzialu od 0 do 1',1
		END
		IF @data_wprowadzenia IS NULL
		BEGIN
			SET @data_wprowadzenia=GETDATE()
		END
		INSERT INTO
Rabaty(Wymagana_kwota,Wysokosc_jedn,Data_wprowadzenia,Data_zdjęcia,ID_Restauracji)
		VALUES
(@wymagana_kwota,@wysokosc_jedn,@data_wprowadzenia,@data_zdjecia,@id_restauracji)
	END TRY
	BEGIN CATCH
```

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

Aktualizuj_Rabat_Ind_Staly – aktualizowanie klientowi indywidualnemu rabatu stałego w danej restauracji

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Ind_Staly]
    @id_klienta INT,
    @id_restauracji INT
AS
BEGIN.
    SET NOCOUNT ON;

  DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
    INNER JOIN Rabaty_Ind_Stale ri ON ri.ID_rabatu=r.ID_rabatu
    WHERE ID_Restauracji=@id_restauracji AND Data_zdjęcia IS NULL)

    IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
    BEGIN
        DECLARE @liczba_zamowien INT = (SELECT Liczba_zamowien FROM Rabaty_Ind_Stale WHERE
ID_rabatu=@id_rabatu)
        DECLARE @wymagana_kwota MONEY = (SELECT Wymagana_kwota FROM Rabaty WHERE
ID_rabatu=@id_rabatu)
        DECLARE @ilosc_powyzej_kwoty INT =(SELECT
dbo.Ilosc_Zamowien_Powyzej_Kwoty(@id_restauracji,@id_klienta,@wymagana_kwota))
        IF (@ilosc_powyzej_kwoty>=@liczba_zamowien)
        BEGIN
            EXEC Przyznaj_Rabat_Klientowi
                @id_rabatu,
                @id_klienta,
                NULL,
                NULL
        END
    END
END
GO
```

Aktualizuj_Rabat_Ind_Jednorazowy – aktualizowanie klientowi indywidualnemu rabatu jednorazowego w danej restauracji

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Ind_Jednorazowy]
    @id_klienta INT,
    @id_restauracji INT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

  DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
    INNER JOIN Rabaty_Ind_Jednorazowe rj ON rj.ID_rabatu=r.ID_rabatu
    WHERE ID_Restauracji=@id_restauracji AND Data_zdjęcia IS NULL)

    IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
    BEGIN
        DECLARE @laczna_wart_zam MONEY = (SELECT SUM(sz.Ilość*sz.Cena_jednostkowa) FROM
Szczegóły_Zamówień sz
        INNER JOIN Zamówienia z ON z.ID_zamówienia=sz.ID_zamówienia
        WHERE @id_klienta=z.ID_klienta)

        DECLARE @wymagana_kwota MONEY=(SELECT Wymagana_kwota FROM Rabaty WHERE
ID_rabatu=@id_rabatu)

        IF @laczna_wart_zam>=@wymagana_kwota
```

```
		BEGIN
			EXEC Przyznaj_Rabat_Klientowi
			@id_rabatu,
			@id_klienta,
			NULL,
			NULL
		END
	END
END
GO
```

Aktualizuj_Rabat_Firm_Miesieczny – aktualizowanie klientowi firmowemu rabatu miesięcznego w danej restauracji

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Firm_Miesieczny]
	@id_klienta INT,
	@id_restauracji INT
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
	INNER JOIN Rabaty_Firm_Miesiac rf ON rf.ID_rabatu=r.ID_rabatu
	WHERE ID_Restauracji=@id_restauracji AND Data_zdjęcia IS NULL)

	IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
	BEGIN
		DECLARE @laczna_wart_zam MONEY = (SELECT SUM(sz.Ilość*sz.Cena_jednostkowa) FROM
Szczegóły_Zamówień sz
		INNER JOIN Zamówienia z ON z.ID_zamówienia=sz.ID_zamówienia
		WHERE @id_klienta=z.ID_klienta AND DATEDIFF(DAY,z.Data_zamówienia,GETDATE())<=30)

		DECLARE @wymagana_kwota MONEY=(SELECT Wymagana_kwota FROM Rabaty WHERE
ID_rabatu=@id_rabatu)

		DECLARE @wymagana_ilosc_zam INT =(SELECT Liczba_zamowien FROM Rabaty_Firm_Miesiac
WHERE ID_rabatu=@id_rabatu)

		DECLARE @ilosc_zam INT =(SELECT COUNT(*) FROM Zamówienia WHERE ID_klienta=@id_klienta
AND DATEDIFF(DAY,Data_zamówienia,GETDATE())<=30)

		IF @laczna_wart_zam>=@wymagana_kwota AND @ilosc_zam>=@wymagana_ilosc_zam
		BEGIN
			EXEC Przyznaj_Rabat_Klientowi
			@id_rabatu,
			@id_klienta,
			NULL,
			NULL
		END
	END

END
GO
```

Aktualizuj_Rabat_Firm_Kwartalny – aktualizowanie klientowi firmowemu rabatu kwartalnego w danej restauracji

```
CREATE PROCEDURE [dbo].[Aktualizuj_Rabat_Firm_Kwartalny]
	@id_klienta INT,
	@id_restauracji INT
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @id_rabatu INT =(SELECT r.ID_rabatu FROM Rabaty r
	INNER JOIN Rabaty_Firm_Miesiac rf ON rf.ID_rabatu=r.ID_rabatu
```

```
	WHERE ID_Restauracji=@id_restauracji AND Data_zdjęcia IS NULL)

	IF @id_rabatu IS NOT NULL AND NOT EXISTS (SELECT * FROM Aktualnie_Przyznane_Rabaty WHERE
ID_klienta=@id_klienta AND @id_rabatu=ID_rabatu)
	BEGIN
		DECLARE @laczna_wart_zam MONEY = (SELECT SUM(sz.Ilość*sz.Cena_jednostkowa) FROM
Szczegóły_Zamówień sz
		INNER JOIN Zamówienia z ON z.ID_zamówienia=sz.ID_zamówienia
		WHERE @id_klienta=z.ID_klienta AND DATEDIFF(DAY,z.Data_zamówienia,GETDATE())<=91)

		DECLARE @wymagana_kwota MONEY=(SELECT Wymagana_kwota FROM Rabaty WHERE
ID_rabatu=@id_rabatu)

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

END
GO
```

Aktualizuj_Cene_Dania – ustawianie nowej ceny dla dania

```
CREATE PROCEDURE [dbo].[Aktualizuj_Cene_Dania]
-- Add the parameters for the stored procedure here
@nazwa_dania varchar(50),
@nowa_cena money
AS
BEGIN
DECLARE @id_dania int = (SELECT ID_dania FROM dbo.Dania WHERE @nazwa_dania=Nazwa_dania)
IF @id_dania IS NULL
	BEGIN
		;THROW 52000, 'Podane danie nie istnieje!',1
	END
ELSE
	BEGIN
	UPDATE dbo.Dania SET Cena_dania=@nowa_cena WHERE ID_dania=@id_dania
	PRINT 'Zaktualizowano pomyślnie.'
END
SET NOCOUNT ON;

END
GO
```

9.Triggery

TRIG_Rezerwacje_Daty – pilnuje, aby data rezerwacji nie mogła być wcześniejsza niż data złożenia rezerwacji

```
CREATE TRIGGER [dbo].[TRIG_Rezerwacje_Daty]
 ON [dbo].[Rezerwacje]
 AFTER INSERT
AS
```

```
BEGIN
 SET NOCOUNT ON
 DECLARE @data_zlozenia date =(SELECT Data_złożenia FROM inserted)
 DECLARE @data_rezerwacji date =(SELECT Data_rezerwacji FROM inserted)
 IF DATEDIFF(day,@data_zlozenia,@data_rezerwacji)<0
 BEGIN
;THROW 50002, 'Data rezerwacji nie moze byc wczesniejsza niz data zlozenia',1
 END
END
GO
```

TRIG_Obostrzenia_Daty – pilnuje, aby obostrzenie na dany stolik nie mogło zostać wstawione z datą wcześniejszą niż poprzednie wprowadzone obostrzenie

```
CREATE TRIGGER [dbo].[TRIG_Obostrzenia_Daty]
 ON [dbo].[Obostrzenia]
 AFTER INSERT
AS
BEGIN
 SET NOCOUNT ON
 DECLARE @id_stolika int =(SELECT ID_stolika FROM inserted)
 DECLARE @data_poprzedniego date = (SELECT TOP 1 Data_wprowadzenia FROM Obostrzenia WHERE
ID_stolika=@id_stolika ORDER BY Data_wprowadzenia DESC)
 DECLARE @data_wprowadzanego date =(SELECT Data_wprowadzenia FROM inserted)
 IF DATEDIFF(day,@data_poprzedniego,@data_wprowadzanego)<0
 BEGIN
;THROW 50002, 'Nie mozna wprowadzic obostrzenia na date wczesniejsza niz poprzednie',1
 END
END
GO
```

TRIG_Menu_Wstawianie_Pozycji – nie pozwala na wstawienie dania, jeśli nie minęło 30 dni od jego ostatniego zniknięcia z menu w danej restauracji

```
CREATE TRIGGER [dbo].[TRIG_Menu_Wstawianie_Pozycji]
 ON [dbo].[Menu]
 AFTER INSERT
AS
BEGIN
 SET NOCOUNT ON
 DECLARE @id_pozycji int =(SELECT ID_pozycji FROM inserted)
 DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Menu WHERE ID_pozycji=@id_pozycji)
 DECLARE @id_dania int = (SELECT ID_dania FROM Menu WHERE ID_pozycji=@id_pozycji)
 DECLARE @data_poprzedniego_zdjecia date = (SELECT TOP 1 Data_zdjęcia FROM Menu WHERE
@id_dania=ID_dania AND @id_restauracji=ID_restauracji
ORDER BY Data_zdjęcia DESC)
 DECLARE @data_wprowadzanego date =(SELECT Data_wprowadzenia FROM inserted)
 IF DATEDIFF(day,@data_poprzedniego_zdjecia,@data_wprowadzanego)<30
 BEGIN
;THROW 50002, 'Nie minelo 30 dni od ostatniego zdjecia potrawy w tej restauracji',1
 END
END
GO
```

TRIG_Zamowienia_Daty – data odbioru zamówienia nie może być wcześniejsza niż data złożenia

```
CREATE TRIGGER [dbo].[TRIG_Zamowienia_Daty]
```

```
 ON [dbo].[Zamówienia]
 AFTER INSERT
AS
BEGIN
 SET NOCOUNT ON
 DECLARE @data_zlozenia date =(SELECT Data_zamówienia FROM inserted)
 DECLARE @data_odbioru date =(SELECT Data_odbioru FROM inserted)
 IF DATEDIFF(day,@data_zlozenia,@data_odbioru)<0
 BEGIN
;THROW 50002, 'Data odbioru nie moze byc wczesniejsza niz data zlozenia zamowienia',1
 END
END
GO
```

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

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

TRIG_Blokuj_Wstawianie_Powtorek – zapobiega powtórzeniu w aktualniej ofercie restauracji kilkukrotnie tego samego dania

```
CREATE TRIGGER [dbo].[TRIG_Menu_Blokuj_Wstawianie_Powtorek]
 ON [dbo].[Menu]
 AFTER INSERT
AS
BEGIN
 SET NOCOUNT ON
 DECLARE @id_pozycji int =(SELECT ID_pozycji FROM inserted)
 DECLARE @id_restauracji int = (SELECT ID_restauracji FROM Menu WHERE ID_pozycji=@id_pozycji)
 DECLARE @id_dania int = (SELECT ID_dania FROM Menu WHERE ID_pozycji=@id_pozycji)
 IF EXISTS(SELECT * FROM Menu WHERE @id_dania=ID_dania AND @id_restauracji=ID_restauracji AND
Data_zdjęcia IS NULL AND ID_pozycji<>@id_pozycji)
 BEGIN
;THROW 50002, 'To danie znajduje sie obecnie w ofercie restauracji',1
 END
END
GO
```

10.Generator

Powyższą bazę danych na potrzeby testów wypełniliśmy odpowiednimi danymi wykorzystując dostępny w sieci program SQL Data Generator firmy Redgate. Dane zostały wygenerowane z zachowaniem zasad spójności oraz rozłączności poszczególnych danych w tabelach. Generator nie udostępniał jednak danych takich jak lista półproduktów, lista dań czy nazwy kategorii. W związku z tym wykorzystane zostały dostępne w sieci (głównie z serwisu GitHub.com) tak zwane „word lists", czyli pliki tekstowe zawierające listy słów ze wskazanych kategorii, uzupełniane również ręcznie, które następnie zostały zaimportowane do generatora, który losowo je dodał do poszczególnych tabel.

Wykorzystane word listy z sieci:
https://github.com/imsky/wordlists/blob/master/nouns/food.txt
https://github.com/imsky/wordlists/blob/master/nouns/fish.txt
https://github.com/schollz/food-identicon/blob/master/ingredients.txt

11.Indeksy

IX_Aktualnie_Dostarcza

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

IX_Aktualnie_Dostarcza_1

```
CREATE NONCLUSTERED INDEX [IX_Aktualnie_Dostarcza_1] ON [dbo].[Aktualnie_Dostarcza]
(
	[ID_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Aktualnie_Przyznane_Rabaty

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

IX_Aktualnie_Przyznane_Rabaty_1

```
CREATE NONCLUSTERED INDEX [IX_Aktualnie_Przyznane_Rabaty_1] ON [dbo].[Aktualnie_Przyznane_Rabaty]
(
	[ID_klienta] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Dostawy

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

IX_Dostawy_1

```
CREATE NONCLUSTERED INDEX [IX_Dostawy_1] ON [dbo].[Dostawy]
(
	[ID_Restauracji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Klienci_Biz

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

IX_Klienci_Biz_1

```
CREATE NONCLUSTERED INDEX [IX_Klienci_Biz_1] ON [dbo].[Klienci_Biz]
(
[ID_miasta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Klienci_Ind

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

IX_Menu

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

IX_Menu_1

```
CREATE NONCLUSTERED INDEX [IX_Menu_1] ON [dbo].[Menu]
(
[ID_restauracji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Miasta

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

IX_Miasta_1

```
CREATE NONCLUSTERED INDEX [IX_Miasta_1] ON [dbo].[Miasta]
(
[ID_państwa] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Obostrzenia

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

IX_Obsluga

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

IX_Panstwa

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

IX_Panstwa_2

```
CREATE NONCLUSTERED INDEX [IX_Panstwa_2] ON [dbo].[Panstwa]
(
[Nazwa] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Polprodukty

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

IX_Pracownicy_Firm

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

IX_Pracownicy_Firm_1

```
CREATE NONCLUSTERED INDEX [IX_Pracownicy_Firm_1] ON [dbo].[Pracownicy_Firm]
(
[ID_pracownika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Przepisy

```
CREATE NONCLUSTERED INDEX [IX_Przepisy] ON [dbo].[Przepisy]
(
[ID_półproduktu] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Rabaty

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

IX_Rabaty_Firm_Miesiac

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

IX_Rabaty_Ind_Jednorazowe

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

IX_Rabaty_Ind_Stale

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

IX_Rezerwacje

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

IX_Rezerwacje_Firm

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

IX_Rezerwacje_Firm_1

```
CREATE NONCLUSTERED INDEX [IX_Rezerwacje_Firm_1] ON [dbo].[Rezerwacje_Firm]
(
[ID_firmy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Rezerwacje_Firm_Imiennie

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

IX_Rezerwacje_Firm_Imiennie_1

```
CREATE NONCLUSTERED INDEX [IX_Rezerwacje_Firm_Imiennie_1] ON [dbo].[Rezerwacje_Firm_Imiennie]
(
[ID_pracownika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Rezerwacje_Firm_Imiennie_2

```
CREATE NONCLUSTERED INDEX [IX_Rezerwacje_Firm_Imiennie_2] ON [dbo].[Rezerwacje_Firm_Imiennie]
(
[ID_firmy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Rezerwacje_Ind

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

IX_Rezerwacje_Ind_1

```
CREATE NONCLUSTERED INDEX [IX_Rezerwacje_Ind_1] ON [dbo].[Rezerwacje_Ind]
(
[ID_klienta] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Rezerwacje_Ind_2

```
CREATE NONCLUSTERED INDEX [IX_Rezerwacje_Ind_2] ON [dbo].[Rezerwacje_Ind]
(
[ID_zamówienia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Stan_Magazynowy

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

IX_Stoliki

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

IX_Szczegoly_Dostaw

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

IX_Szczegoly_Dostaw_1

```
CREATE NONCLUSTERED INDEX [IX_Szczegoly_Dostaw_1] ON [dbo].[Szczegóły_Dostaw]
(
[ID_półproduktu] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Szczegoly_Rezerwacji

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

IX_Szczegoly_Rezerwacji_1

```
CREATE NONCLUSTERED INDEX [IX_Szczegoly_Rezerwacji_1] ON [dbo].[Szczegóły_Rezerwacji]
(
[ID_obostrzenia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Szczegoly_Zamowien

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

IX_Szczegoly_Zamowien_1

```
CREATE NONCLUSTERED INDEX [IX_Szczegoly_Zamowien_1] ON [dbo].[Szczegóły_Zamówień]
(
[ID_pozycji] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
GO
```

IX_Zamowienia

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

12.Uprawnienia

Do zarządzanie systemem zaleca się zdefiniowanie następujących ról:

1. **Administrator -** całkowity dostęp do bazy i jej funkcjonalności
2. **Menedżer restauracji -** dostęp do danych dotyczących jego restauracji i zarządzanie nimi
3. **Pracownik restauracji -** dostęp do funkcji potrzebnych do obsługi klientów
4. **Klient biznesowy -** możliwość składania zamówień, rezerwacji, zarządzania listą pracowników jego firmy. Dostęp do danych określających jego zamówienia, rezerwacje i rabaty.
5. **Klient indywidualny -** dostęp do danych określających jego zamówienia, rezerwacje i rabaty, możliwość składania zamówień i rezerwacji w swoim imieniu.
6. **Użytkownik niezidentyfikowany -** dostęp do listy menu danej restauracji.

13.Funkcje użytkowników

1. Aktualizacja rabatów (Menedżer restauracji)
2. Zdefiniowanie nowego rabatu (Menedżer restauracji)
3. Dodanie dania (Menedżer restauracji)
4. Dodanie potrawy do menu (Menedżer restauracji)
5. Usunięcie potrawy z menu (Menedżer restauracji)
6. Rejestracja nowego dostawcy (Administrator)
7. Zlecenie dostawy (Menedżer restauracji)
8. Dodanie produktu do dostawy (Menedżer restauracji)
9. Potwierdzenie odbioru dostawy (Menedżer restauracji/ Pracownik restauracji)
10. Wygenerowanie listy dostaw niezrealizowanych (Menedżer restauracji)
11. Przyjęcie zamówienia i jego potwierdzenie (Pracownik restauracji)
12. Rejestracja klienta (Klient indywidualny/ Klient biznesowy)
13. Rejestracja lokalu (Administrator)
14. Dodanie miasta oraz państwa (Administrator)
15. Wprowadzenie i zarządzanie obostrzeniami w obrębie restauracji (Menedżer restauracji)
16. Dodanie stolików (Menedżer restauracji)
17. Rejestracja pracownika restauracji (Menedżer restauracji)
18. Rejestracja informacji o pracowniku firmy (Klient biznesowy)
19. Lista zarejestrowanych pracowników firmy (Klient biznesowy)
20. Złożenie rezerwacji (Klient indywidualny/ Klient biznesowy)
21. Anulowanie rezerwacji (Klient indywidualny/ Klient biznesowy)
22. Wgląd do aktualnych rabatów, rezerwacji i zamówień klienta (Klient indywidualny/ Klient biznesowy/ Pracownik Restauracji/ Menedżer restauracji)
23. Wygenerowanie faktury za miesiąc usług (Klient Biznesowy)
24. Wygenerowanie faktury za pojedyncze zamówienie (Klient Biznesowy)
25. Wygenerowanie listy osób na rezerwację imienną dla firmy (Klient Biznesowy/ Pracownik restauracji)
26. Wgląd do menu restauracji (Każdy)
27. Wygenerowanie przepisu konkretnego dania (Pracownik restauracji)
28. Generowanie statystyki zamówień, stolików, rezerwacji i wyników obsługi dla danej restauracji (Menedżer restauracji)

29. Wgląd do stanu magazynowego (Menedżer restauracji/ Pracownik restauracji)
30. Lista kontaktowa dostawców półproduktu (Menedżer restauracji)

14.Funkcje systemowe

1. Automatyczne zwiększenie lub zmniejszenie stanu półproduktu w magazynie
2. Obliczenie rabatów dla danego klienta
3. Automatyczna obsługa systemu rabatów (przyznanie i usuwanie rabatu)
4. Kontrola dostępności stolików przy rezerwacjach
5. Obliczenie wartości zamówienia
6. Obliczenie liczby wolnych miejsc w lokalu (przy rezerwacjach)
7. Kontrola zasad związanych ze szczegółowymi wymaganiami klienta (rabaty, menu, owoce morza itp.)