

Piotr Płoszczyca
Dominika Kwaśniewska



Projekt i implementacja systemu bazodanowego.
System Zarządzania lokalami gastronomicznymi.

Podstawy Baz Danych
Kraków, 2020

Spis treści

1.	Wprowadzenie	6
2.	Cel Projektu	6
3.	Schemat utworzonej bazy danych	6
4.	Analiza wymagań	8
5.	Opis tabel wraz z warunkami integralnościowymi	14
5.1.	Opis klientów	14
5.1.1.	Tabela Clients	14
5.1.2.	Tabela Individual_Clients	16
5.1.3.	Tabela Companies	17
5.1.4.	Tabela Company_Members	19
5.2.	Dania oraz składniki	20
5.2.1.	Tabela Dishes	20
5.2.2.	Tabela Dish_details	22
5.2.3.	Tabela Ingredients	23
5.2.4.	Tabela Categories	26
5.2.5.	Tabela Menu	27
5.3.	Zamówienia	29
5.3.1.	Tabela Orders	29
5.3.2.	Tabela Order_details	31
5.3.3.	Tabela Orders_unpaid	32
5.4.	Rezerwacje i stoliki	33
5.4.1.	Tabela Reservations	33
5.4.2.	Tabela Personal_Reservations	34
5.4.3.	Tabela Company_Reservations	35
5.4.4.	Tabela Company_Name_Reservations	35
5.4.5.	Tabela Restrictions	36
5.4.6.	Tabela Tables	38
5.5.	Rabaty	39
5.5.1.	Tabela Discounts	39
5.5.2.	Tabela Discount_dictionary	41
5.6.	Dostawcy	43
5.6.1.	Tabela Suppliers	43
5.7.	Pracownicy	46
5.7.1.	Tabela Employees	46
5.7.2.	Tabela Positions	49

5.8.	Adresy	50
5.8.1.	Tabela Towns	50
5.8.2.	Tabela Countries	50
5.8.3.	Tabela Towns_connections	51
5.9.	Restauracje	52
5.9.1.	Tabela Restaurants	52
6.	Generator danych	54
7.	Widoki, funkcje, procedury, triggerzy	55
7.1.	Funkcje użytkownika	55
7.1.1.	Rezerwacje na dzień dzisiejszy(TodaysReservations)	55
7.1.2.	Dostępne stoliki w danym dniu (SpecificDaysFreeTables)	56
7.1.3.	Obecne zamówienia (CurrentOrdersToMake)	56
7.1.4.	Wszystkie zamówienia w danym dniu (SpecificDayOrders)	57
7.1.5.	Zamówienie z pozycjami(OrderDetails)	57
7.1.6.	Aktualni pracownicy (CurrentEmployees)	58
7.1.7.	Menu - obecne pozycje (MenuToday)	58
7.1.8.	Brakujące składniki do zamówienia (MissingIngredientsForOrder)	59
7.1.9.	Jakie rabaty przysługują klientowi (ClientsDiscount)	59
7.1.10.	Tworzenie faktury - funkcja ogólna (Invoice)	60
7.1.11.	Tworzenie faktury dla jednego zamówienia dla firm (OrderInvoice)	61
7.1.12.	Tworzenie faktury zbiorczej raz na miesiąc dla firmy (MonthInvoice)	61
7.1.13.	Niezapłacone zamówienia (z informacją o kliencie) (UnpaidOrders)	61
7.1.14.	Wyświetlenie klientów danej restauracji (AllClients)	62
7.1.15.	Wyświetlenie pracowników danej firmy (CustomersBelongToCompany)	62
7.1.16.	Wyświetlanie aktualnych typów rabatów dla danej restauracji (CurrentRestaurantDiscountTypes)	63
7.1.17.	Wyświetlanie wszystkich dań, które nie są zablokowane (AllUnlockedDishes)	63
7.1.18.	Wyświetlenie składników danego dania (DishIngredients)	64
7.1.19.	Wyświetlenie dań, które można dodać do menu (były co najmniej miesiąc wcześniej usunięte) (dbo.DishesThatCanBeAddedToMenu)	64
7.1.20.	Wyświetlanie wszystkich zamówień danego klienta (GetClientOrderHistory)	65
7.2.	Funkcje do generowania raportów	65
7.2.1.	Obliczanie średniego rabatu (AverageDiscount)	65
7.2.2.	Średnia ilość zamówień (AverageOrders)	66
7.2.3.	Średnia kwota zamówień na dany dzień (AverageOrderCost)	66
7.2.4.	Dzień z największą liczbą zamówień (DayWithMaxOrderCount)	67

7.2.5.	Ilość rezerwacji w danej restauracji w danym dniu(CountReservationsInOneDay)	68
7.2.6.	Liczba pozycji w menu (AverageMenuPositions)	68
7.2.7.	Średnia ilość rezerwacji na dzień (AverageReservations)	69
7.2.8.	Raport (Report)	69
7.2.9.	Raport tygodniowy (WeeklyReport)	70
7.2.10.	Raport miesięczny (MonthlyReport)	70
7.2.11.	Funkcja do generowania faktury (Invoice)	70
7.2.12.	Faktura dla firm dla pojedynczego zamówienia (OrderInvoice)	71
7.2.13.	Faktura zbiorcza miesięczna (MonthInvoice)	72
7.3.	Funkcje systemowe	72
7.3.1.	Funkcja obliczająca wartość zamówienia (SumOrder)	72
7.3.2.	Funkcja sprawdzająca, czy można zamówić owoce morza (czwartek-piątek-sobota) (CanOrderSeafood)	72
7.3.3.	Sprawdzenie, czy klient indywidualny może dokonać rezerwacji (CanIndividualMakeReservation)	73
7.3.4.	Sprawdzenie warunków na przydzielenie rabatu dla klienta indywidualnego (CheckIndividualDiscount)	73
7.3.5.	Znalezienie wartości nowego rabatu dla klienta indywidualnego (GetFirstOrSecondDiscountValue)	74
7.3.6.	Sprawdzenie warunków na 3 i 4 typ rabatu dla klienta indywidualnego (GetThirdAndFourthDiscountID)	75
7.3.7.	Sprawdzenie warunków na 1. rabat dla firmy (GetCompanyFirstDiscountValue)	76
7.3.8.	Sprawdzenie warunków na 2. rabat dla firmy (GetCompanySecondDiscountID)	77
7.4.	Procedury	78
7.4.1.	Dodanie nowego klienta indywidualnego (AddIndividualClient)	78
7.4.2.	Dodanie nowego klienta jako firmy (AddCompanyClient)	79
7.4.3.	Dodanie restauracji do bazy (AddRestaurant)	80
7.4.4.	Dodanie pracownika restauracji (AddEmployee)	81
7.4.5.	Tworzenie rezerwacji dla klienta indywidualnego (AddIndividualReservation)	82
7.4.6.	Tworzenie rezerwacji dla firmy (AddCompanyReservation)	83
7.4.7.	Dodanie imiennej rezerwacji do rezerwacji firmowej (AddNameReservation)	84
7.4.8.	Dodanie stolika (AddTable)	85
7.4.9.	Tworzenie ograniczenia na stoliki (AddRestriction)	85
7.4.10.	Dodanie pozycji menu (AddDishToMenu)	86
7.4.11.	Dodanie typu rabatu (AddDiscountDictionary)	87
7.4.12.	Złożenie zamówienia (AddOrder)	88
7.4.13.	Dodawanie szczegółów do zamówieni (AddOrderDetails)	89

7.4.14.	Dodanie dostawcy (AddSupplier)	90
7.4.15.	Dodanie składnika (AddIngredient)	91
7.4.16.	Dodanie dania (AddDish)	92
7.4.17.	Łączenie składnika z daniem (ConnectDishWithIngredient)	93
7.4.18.	Dodanie klienta do firmy (AddIndividualToCompany)	93
7.4.19.	Aktualizowanie składnika w magazynie (UpdateIngredientAmount)	94
7.4.20.	Usuwanie rekordu z tabeli Orders_Unpaid, jeżeli zamówienie zostało opłacone (DeleteUnpaidOrder)	94
7.4.21.	Dodanie kategorii (AddCategory)	95
7.4.22.	Dodanie miasta (AddTown)	95
7.4.23.	Dodanie kraju (AddCountry)	96
7.4.24.	Pozycje będące w menu ponad 2 tygodnie (DishesInMenuOverTwoWeeks)	96
7.4.25.	Przydzielenie firmie rabatu (AddNewDiscountToCompanies)	97
7.4.26.	Procedura wysyłająca informację do użytkownika, jeżeli ponad połowa pozycji w menu jest ponad 2 tygodnie (MenuMessage)	98
7.5.	Triggery	99
7.5.1.	Przydzielenie klientowi indywidualnemu rabatu (AddNewDiscountToIndividualClient)	99
7.5.2.	Jeżeli składnik zostanie zablokowany to wszystkie dania zawierające ten składnik też zostaną zablokowane (LockDish)	101
7.5.3.	Jeżeli danie zostało zablokowane i znajduje się w menu, danie zostaje usunięte z aktualnego menu (LockMenu)	101
7.5.4.	Blokada składnika, jeżeli nie będzie wystarczającej ilości w magazynie (LockIngredientWithLowAmount)	102
7.5.5.	Sprawdzenie przy rezerwacji czy zamówienie klienta indywidualnego nie jest czasem na wynos (checkCorrectnessOfAddedReservation)	102
7.6.	Widoki	103
7.6.1.	Informacje o restauracjach w bazie (RestaurantsInformations)	103
7.6.2.	Wszystkie kategorie dań w systemie (CategoriesInformations)	103
7.6.3.	Wszystkie miasta obecne w bazie (TownsAndCountriesNames)	103
8.	Użytkownicy systemu i ich uprawnienia	104

1. Wprowadzenie

Projekt został utworzony w ramach przedmiotu Podstawy Baz Danych na kierunku Informatyka. Dotyczy on systemu wspomagania działalności firmy świadczącej usługi gastronomiczne dla klientów indywidualnych oraz firm. Projekt zawiera elementy systemu bazodanowego jak opis poszczególnych tabel wchodzących w skład tworzonej bazy danych oraz szczegółowe informacje na temat funkcji umożliwiających korzystanie z systemu.

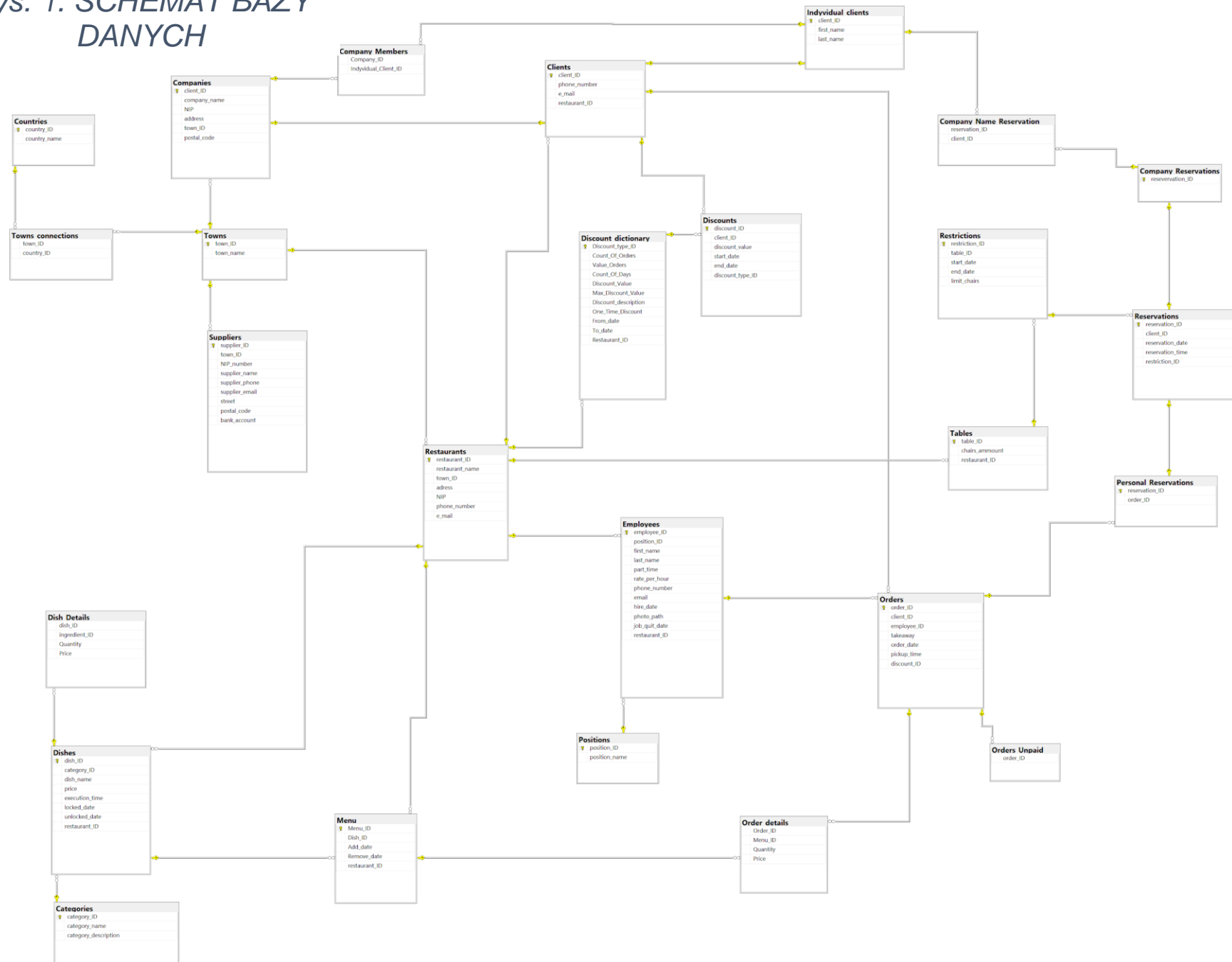
2. Cel Projektu

Celem projektu jest implementacja systemu wspierającego działalność firm gastronomicznych. System ten musi wspomagać procesy zachodzące w firmach, umożliwiać równoległe korzystanie z niego dla wielu firm.

3. Schemat utworzonej bazy danych

Na rysunku 1 został przedstawiony schemat utworzonej bazy danych. Przedstawia on 27 tabel, które szczegółowo zostały opisane w punkcie 5.

Rys. 1. SCHEMAT BAZY
DANYCH



4. Analiza wymagań

Na podstawie opisu reguł biznesowych dotyczących firmy, po ich wnikliwej analizie ustalone zostały wymagania, jakie system musi spełniać.

Po dogłębnym przeanalizowaniu reguł miniświata i potrzeb użytkowników ustalono następujące wymagania:

System musi umożliwiać dokonywania zamówień poszczególnym klientom, stąd w bazie będą przechowywane szczegółowe informacje na temat każdego klienta, który kiedykolwiek składał zamówienie w restauracji. W bazie wszyscy klienci posiadają unikatowe atrybuty:

- numer (client_ID), po takim numerze następować będzie identyfikacja klienta,
- nr telefonu (phone_number) – w celu kontaktowania się z klientem w ramach zamówienia, problemów nagłych związanych z danym zamówieniem złożonym przez klienta. Zaleca się podanie takiego numeru w bazie, jednak nie jest to pole obowiązkowe
- adres email (E-mail) - umożliwi przesyłanie informacji na temat zamówienia lub kontakt z klientem (również pole nieobowiązkowe, gdyby klient nie zgodził się na podanie prywatnych danych).
- restaurant_ID - identyfikuje restaurację, do której należy klient

Klienci ponadto zostali podzieleni na 2 osobne grupy ze względu na dodatkowe informacje, jakie będą przechowywane w bazie na temat każdej z grup oraz różnice w postępowaniu z danym klientem. Każdy klient z grupy ogólnej musi posiadać szczegółowe informacje tylko w jednej z tych grup, relacja ta będzie się opierać o numer identyfikujący każdego z klientów.

Podział klientów:

- Klienci indywidualni posiadający:
- imię (first_name)
- nazwisko (last_name)

Tu będą się zaliczać wszyscy klienci indywidualni, którzy kiedykolwiek zrobili zamówienia prywatne w restauracji.

- Firmy posiadają:
- nazwę firmy (company_name)

W systemie będą przechowywane również informacje na temat rabatów:

- ID rabatu (discount_ID)
- ID klienta (client ID)
- Wartość rabatu (%) - (discount_value)
- Data od kiedy przysługuje rabat (start_date)
- Data do kiedy przysługuje rabat (end_date)
- Typ rabatu (discount_type_ID) - określony w słowniku rabatu

Baza musi również przechowywać szczegóły na temat aktualnych typów rabatów w postaci słownika rabatów:

- Discount_type_ID - identyfikuje typ rabatu
- Count_Of_Orders - liczba zamówień, które muszą być złożone przez klienta jako warunek otrzymania rabatu
- Value_Orders - minimalna wartość zamówień z podpunktu wyżej
- Count_Of_Days - na ile obowiązuje rabat (jeżeli posiada ograniczenie czasowe)
- Discount_value - wartość rabatu
- Max_discount_Value - maksymalna wartość rabatu
- Discount_description - opis dodatkowy rabatu
- One_Time_discount - określa czy rabat jest jednorazowy
- From_date - określa od kiedy obowiązuje dany typ rabatu
- To_date - określa do kiedy obowiązuje dany typ rabatu
- Restaurant_ID - typy rabatu obowiązują tylko w danej restauracji

Restauracja oferuje pewną pulę dań, z której dania będą wybierane do menu. W związku z tym w bazie będą przechowywane zarówno ogólne i szczegółowe informacje na temat każdego dania. System będzie również umożliwiać aktualizację puli poprzez dodanie dania, gdy zajdzie taka potrzeba oraz blokowanie dania, jeżeli zabraknie potrzebnych składników w magazynie lub tymczasowo nie będzie możliwe wykonanie takiego dania z przyczyn losowych.

Każde danie serwowane w restauracji musi więc posiadać:

- unikatowy numer (dish_ID), po którym danie będzie identyfikowane w bazie
- nazwę dania (dish_name) - posłuży do informowania klientów jakie danie jest sprzedawane
- cenę jednostkową (price)- cena, po której danie będzie sprzedawane w restauracji
- czas wykonania dań wyrażony w minutach (execution_time).
- data zablokowania dania, w przypadku, gdy składnik nie jest możliwy do zakupu lub wykorzystania w restauracji (locked_date)
- data odblokowania dania, wtedy można ponownie używać produktu. (unlocked_date)
- Data usunięcia z menu (Remove_date)
- Data dodania do menu (Add_date)
- Numer identyfikujący restaurację (Restaurant_ID)

System przechowuje również informację czy danie jest ekskluzywne tzn. czy wymaga wcześniejszego skompletowania składników, możliwe do zamówienia w dni czwartek-piątek-sobota (np. Dania posiadające w składzie owoce morza) w formie KATEGORII (jeżeli nazwa kategorii to "dania ekskluzywne"). Umożliwi to sprawdzenie czy zamówienie posiadające na liście takie dania będzie złożone co najmniej w poniedziałek poprzedzający datę zamówienia.

Dania są pogrupowane po kategoriach. Wynika to z konieczności uporządkowania dań w prezentowanym przez restaurację menu, ułatwi to klientom wyszukiwania konkretnego posiłku na liście pozycji. Taka kategoria będzie stwierdzać, czy dane danie jest lunchem, napojem bezalkoholowym czy przekąską.

Każda kategoria dysponować będzie więc:

- unikatowym numerem (`category_ID`) - identyfikującym każdą kategorię, pozwoli ograniczenie dublowania informacji o kategorii przy każdym daniu
- nazwą kategorii (`category_name`) - będzie informować o typie dania i przynależności do grupy dań
- opis kategorii (`category_description`) - nieobowiązkowe pole, do informowania klientów o tym co się znajduje w kategorii

W bazie istnieje konieczność przechowywania również informacji na temat składników, z których przygotowywane będą dania w restauracji. Pozwoli to na ciągłe zapewnienie podstawowej ilości półproduktów w magazynie poprzez wykonanie zamówienia na dany składnik u jego dostawcy. Ponadto patrząc na dzisiejsze realia, w których popularna jest kultura wegetariańska, wegańska, oraz problemy zdrowotne klientów w postaci nietolerancji oraz alergii, baza musi przechowywać również dodatkowy opis dotyczący składu produktu (np. gluten-free). Ułatwi to informowanie klientów na ich prywatną prośbę. Związane jest to z obecnie panującym standardem w sieciach gastronomicznych.

Do przygotowania każdego dania potrzeba więcej niż jednego składnika, jak również każdy składnik może być przypisany do więcej niż jednego dania (Relacja wiele do wielu).

Każdy składnik posiada:

- unikatowy numer (`ingredient_ID`) - identyfikujący składnik jednoznacznie, pozwoli to na ukazanie relacji wiele do wielu poprzez łączenie ID składnika z ID dania (ograniczenie dublowania informacji w bazie)
- nazwę (`ingredient_name`)
- informacje o dostępnej ilości danego składnika w magazynie (`ingredient_in_stock`),
- cenę (`ingredient_price`) - za jaką składnik kupowany jest w hurtowni,
- ilość na jednostkę składnika (`ingredient_quantity`) - np. "pół litra", "100 g", itp.
- dostawcę składnika (`supplier_ID`) - od kogo kupujemy towar, aby w łatwy sposób można było go uzupełnić.
- Wartość bezpieczna (`safe_amount`) – minimalna ilość składnika jaka powinna znajdować się w magazynie. Jeśli przekroczymy tą wartość to towar powinien niezwłocznie zostać uzupełniony.
- Dodatkowe informacje dotyczące składu (`extra_information`)
- Data zablokowania składnika, w przypadku gdy składnik nie jest możliwy do zakupu lub wykorzystania w restauracji (`locked_date`)
- Data odblokowania składnika, wtedy można ponownie używać produktu. (`unlocked_date`)

- Restaurant_ID - numer restauracji, która wykorzystuje dany składnik (dana restauracja przechowuje informacje o ilości składnika w magazynie, więc nawet gdyby więcej restauracji korzystało z tego samego składnika, nie można by tego zamodelować, stąd muszą być różne pozycje dla każdego składnika w każdej restauracji)

W związku z koniecznością uzupełniania potrzeb magazynowych przechowywane również będą w systemie szczegółowe informacje dotyczące dostawcy, zarówno jego dane kontaktowe, jak i NIP do wystawiania faktur. Każdy dostawca dostarczać może więcej niż jeden produkt (relacja jeden do wielu).

W systemie będą przechowywani dostawcy:

- o unikalnym numerze (supplier_ID) - do identyfikacji dostawcy
- NIP-ie firmy - umożliwi automatyczne generowanie faktur/paragonów
- nazwie firmy dostawczej (supplier_name)
- numerze telefonu (supplier_phone) - do szybkiego kontaktowania się z firmą
- adresie e-mail (supplier_email) - nieobowiązkowe pole
- adresie firmy (supplier_adress)
- Ulica (street)
- Miasto (town)
- Kod pocztowy (postal_code)
- Państwo (country)
- Numerze konta bankowego (bank_account) - do wykonywania przelewów (nieobowiązkowe, zależy to od umowy, w której określono sposób płacenia za dobra materialne)

Każdy klient składa zamówienie, na które złożone jest z aktualnie dostępnych dań w menu. Zamówienia mogą być z odbiorem na wynos lub na miejscu. Każde zamówienie posiada:

- numer zamówienia (order_ID) - każde zamówienie będzie posiadać unikatowy numer
- numer klienta, który składa dane zamówienie (client_ID)
- charakter zamówienia (na miejscu/na wynos) (takeaway)
- numer pracownika odpowiedzialnego za skompletowanie dania (employee_ID)
- datę odbioru zamówienia (order_date)
- datę i godzinę odbioru zamówienia (pickup_time)

W bazie będą przechowywane również szczegóły każdego zamówienia w postaci:

- Order ID
- dań (dish_ID), jakie zostały zamówione
- ilości (quantity) - można zamówić jedną, lub więcej porcji danego dania
- ceny (price) - obowiązująca w czasie składania zamówienia
- rabacie (discount_ID) przyznanym do zamówienia - informacja o rabacie jest konieczna, gdyż klient może mieć w tym samym czasie więcej niż jeden rabat

Firma gastronomiczna umożliwia składanie rezerwacji poprzez internetowy formularz, zarówno przez klienta indywidualnego, przy jednoczesnym złożeniu zamówienia oraz przez firmę w dwóch opcjach, na firmę albo imiennie.

W systemie więc przechowywane będą szczegółowe informacje na temat każdej rezerwacji.

Każda rezerwacja posiada:

- unikatowy numer (reservation_ID),
- numer klienta, który sporządził rezerwację (client_ID),
- numer zamówienia (order_ID),
- datę rezerwacji (reservation_date),
- godzinę rezerwacji (reservation_time).

Jeżeli w szczególności firma dokona rezerwacji imiennej, to takie imię będzie również przechowywane w bazie (name_reservation).

Do każdej rezerwacji przydzielony może być więcej niż 1 stolik (table_ID) (relacja jeden do wielu).

W bazie muszą być przechowywane informacje na temat liczby dostępnych stolików oraz maksymalnej ilości miejsc siedzących przy danym stoliku. Na każdy stół składają się więc informacje:

- Unikatowy numer stolika (table_ID),
- Maksymalna liczba krzeseł przydzielonych do stolika (chars_ammount),

Informacje w ramach restrykcji nałożonych ze względu na pandemię będą przechowywane osobno:

- Numer stolika (table_ID)
- Data nałożenia ograniczenia (start_date)
- Data zniesienia ograniczenia (end_date)
- Ograniczenie liczby krzeseł < maksymalna liczba krzeseł oraz >= 0 (limit_chairs)

W systemie będą przechowywane też informacje o osobach pracujących w firmie. Pracownicy będą odpowiedzialni za skompletowanie zamówienia na wynos lub dostarczenie wszystkich dań klientowi w lokalu. System będzie również rozróżniać stanowiska poszczególnych osób w firmie. Każdy pracownik będzie posiadać:

- unikalny numer (employee_ID),
- imię (first_name),
- nazwisko (last_name),
- stanowisko (position_ID) - Etat (part_time) - pracownik może być zatrudniony na pełny etat/pół etatu
- stawkę godzinową (rate_per_hour) - kwota, jaką zarabia pracownik,

- numer telefonu (phone_number),
- e-mail (email) - nieobowiązkowy,
- datę zatrudnienia (hire_date),
- zdjęcie (photo_path) - nieobowiązkowe pole, przydatne do sporządzenia plaketek pracowniczych
- restaurant_ID - do identyfikacji restauracji, w której pracuje dany pracownik

Positions:

- position_ID – unikalny numer stanowiska
- position_name - nazwa stanowiska (np. kucharz/kelner/szef/manager)

Kraje i Miasta - w bazie będą przechowywane kraje oraz miasta jako atrybuty klientów oraz dostawców:

Kraje:

- Unikalny numer (country_ID)
- Nazwa (country_name)

Miasta:

- Unikalny numer (town_ID)
- Nazwa miasta (town_name)

Kraje będą połączone z miastami za pomocą relacji jeden kraj do wielu miast (ID kraju – ID miasta)

W związku z wykorzystywaniem bazy danych równolegle przez różne firmy, w systemie istnieje konieczność również przechowywania informacji na temat restauracji:

- restaurant_ID - numer identyfikujący daną restaurację w bazie
- restaurant_name - nazwa restauracji
- town_ID - określa miasto w jakim znajduje się restauracja
- adress - adres restauracji
- NIP - numer NIP
- e_mail - email restauracji do kontaktu
- phone_number - numer telefonu danej restauracji

5. Opis tabel wraz z warunkami integralnościowymi

Tabele zostały opisane w tabelach (nazwa kolumny - opis). Zawarte zostały również oryginalne tabele z bazy danych dla ukazania ustalonego typu danych dla kolumny oraz czy zmienna może być nullem.

5.1. Opis klientów

5.1.1. Tabela Clients

Clients			
	Column Name	Data Type	Allow Nulls
?	client_ID	int	<input type="checkbox"/>
	phone_number	varchar(10)	<input checked="" type="checkbox"/>
	e_mail	varchar(50)	<input checked="" type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela Clients posiada informacje ogólne na temat klientów restauracji. Jest połączona z tabelami: Indywidual_Clients oraz Companies relacją 1 do 1 (ten sam Client_ID), tabelą Discounts relacją jeden do wielu (każdy klient może mieć więcej niż jeden rabat) oraz z tabelą Orders relacją jeden do wielu (każdy klient może złożyć dowolną liczbę zamówień). Jest również połączona z tabelą przechowującą informacje na temat restauracji (równoległe korzystanie wielu firm z bazy danych).

Warunki integralnościowe:

- Email musi być unikalny, zawiera w sobie "@"
- Numer telefonu musi być unikalny, składa się tylko z cyfr

client_ID	Unikatowy numer klienta, jednoznacznie go identyfikuje, klucz główny tabeli
phone_number	Numer telefonu klienta
e_mail	Email klienta
restaurant_ID	Klucz obcy identyfikujący restaurację

```

CREATE TABLE [dbo].[Clients](
    [client_ID] [int] NOT NULL,
    [phone_number] [varchar](10) NULL,
    [e_mail] [varchar](50) NOT NULL,
    [restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Clients_1] PRIMARY KEY CLUSTERED
(
    [client_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [EMAIL] UNIQUE NONCLUSTERED
(
    [e_mail] 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 [phone] UNIQUE NONCLUSTERED
(
    [phone_number] 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].[Clients] WITH CHECK ADD CONSTRAINT
[FK_Clients_Restaurants] FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [FK_Clients_Restaurants]
GO

ALTER TABLE [dbo].[Clients] WITH CHECK ADD CONSTRAINT [CK_Clients] CHECK
(((e_mail like '%@%'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [CK_Clients]
GO

ALTER TABLE [dbo].[Clients] WITH CHECK ADD CONSTRAINT [phone_numeric]
CHECK ((isnumeric([phone_number])=(1)))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [phone_numeric]
GO

```

5.1.2. Tabela Indyvidual_Clients

Indyvidual clients			
	Column Name	Data Type	Allow Nulls
🔑	client_ID	int	<input type="checkbox"/>
	first_name	varchar(30)	<input type="checkbox"/>
	last_name	varchar(30)	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela Indyvidual_Clients przechowuje szczegółowe informacje na temat klientów indywidualnych. Jako że każdy klient indywidualny może być jednocześnie pracownikiem firmy również będącej klientem, tabela ta jest połączona z tabelą Company_Members, która łączy ze sobą tabele Indyvidual_Clients oraz Companies zawierając tylko klucze obce.

W celu identyfikacji rezerwacji imiennych firm, tabela również jest połączona relacją wiele do wielu z tabelą Company_Name_Reservations.

Warunki integralnościowe:

- Imię i nazwisko klienta (First_name i last_name) zaczynają się z wielkiej litery

client_ID	Klucz główny, unikatowy numer klienta
first_name	Imię klienta
last_name	Nazwisko klienta

```
CREATE TABLE [dbo].[Indyvidual_clients](
    [client_ID] [int] NOT NULL,
    [first_name] [varchar](30) NOT NULL,
    [last_name] [varchar](30) NOT NULL,
    CONSTRAINT [PK_Indyvidual_clients_1] PRIMARY KEY CLUSTERED
(
    [client_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Indyvidual_clients] WITH CHECK ADD CONSTRAINT
[FK_Indyvidual_clients_Clients] FOREIGN KEY([client_ID])
REFERENCES [dbo].[Clients] ([client_ID])
GO
```

```
ALTER TABLE [dbo].[Indyvidual_clients] CHECK CONSTRAINT
[FK_Indyvidual_clients_Clients]
GO
```

```
ALTER TABLE [dbo].[Indyvidual_clients] WITH CHECK ADD CONSTRAINT
[CK_Indyvidual_clients_first_name] CHECK (((first_name) like '[A-Z]%%'))
```


GO

```
ALTER TABLE [dbo].[Indyvidual_clients] CHECK CONSTRAINT  
[CK_Indyvidual_clients_first_name]  
GO
```

```
ALTER TABLE [dbo].[Indyvidual_clients] WITH CHECK ADD CONSTRAINT  
[CK_Indyvidual_clients_last_name] CHECK ((([last_name] like '[A-Z]%''))  
GO
```

```
ALTER TABLE [dbo].[Indyvidual_clients] CHECK CONSTRAINT  
[CK_Indyvidual_clients_last_name]  
GO
```

5.1.3. Tabela Companies


Companies			
	Column Name	Condensed Type	Nullable
	client_ID	int	No
	company_na...	varchar(20)	No
	NIP	varchar(10)	No
	address	varchar(50)	No
	town_ID	int	No
	postal_code	varchar(10)	No

Tabela Companies przechowuje informacje na temat klientów będących firmami. Tabela jest również połączona z tabelą Towns, aby w jak najoptymalniejszy sposób przechowywać dane adresowe konieczne do wystawiania faktur.

Warunki integralnościowe:

- Numer NIP musi być unikalny, numer również musi składać się tylko z cyfr
- Kod pocztowy jest tylko w postaci XX-XXX, XXXXXX, XXXXXXX, gdzie X to cyfra
- Nazwa firmy (Company_name) musi być unikalne

client_ID	Klucz główny, unikatowy numer klienta
company_name	Nazwa firmy
NIP	Numer NIP
adres	Adres firmy (np. ulica)
town_ID	Klucz obcy, identyfikuje miasto, w jakim znajduje się firma
postal_code	Kod pocztowy

```

CREATE TABLE [dbo].[Companies](
    [client_ID] [int] NOT NULL,
    [company_name] [varchar](20) NOT NULL,
    [NIP] [varchar](10) NOT NULL,
    [address] [varchar](50) NOT NULL,
    [town_ID] [int] NOT NULL,
    [postal_code] [varchar](10) NOT NULL,
    CONSTRAINT [PK_Companies_1] PRIMARY KEY CLUSTERED
(
    [client_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [Company_name] UNIQUE NONCLUSTERED
(
    [company_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [NIP_companies] 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].[Companies] WITH CHECK ADD CONSTRAINT
[FK_Companies_Clients] FOREIGN KEY([client_ID])
REFERENCES [dbo].[Clients] ([client_ID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Clients]
GO

ALTER TABLE [dbo].[Companies] WITH CHECK ADD CONSTRAINT
[FK_Companies_Towns] FOREIGN KEY([town_ID])
REFERENCES [dbo].[Towns] ([town_ID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Towns]
GO

ALTER TABLE [dbo].[Companies] WITH CHECK ADD CONSTRAINT [NIP] CHECK
((isnumeric([NIP])=(1)))
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [NIP]
GO

```

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

```
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [Postal_code]
GO
```

5.1.4. Tabela Company_Members

Company Members			
	Column Name	Data Type	Allow Nu...
	Company_ID	int	<input type="checkbox"/>
	Indyvidual_Client_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela Company_members łączy ze sobą tabele Indyvidual_Clients wraz z Companies. Pozwala określić, który klient indywidualny jest również pracownikiem firmy. Tabela określa więc zależność "Klient X pracuje w firmie Y".

company_ID	Klucz obcy, identyfikuje Firmę, w której pracuje klient indywidualny
indyvidual_client_ID	Klucz obcy, identyfikuje klienta – pracownika

```
CREATE TABLE [dbo].[Company_Members] (
    [Company_ID] [int] NOT NULL,
    [Indyvidual_Client_ID] [int] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Company_Members] WITH CHECK ADD CONSTRAINT
[FK_Company_Members_Companies] FOREIGN KEY([Company_ID])
REFERENCES [dbo].[Companies] ([client_ID])
GO
```

```
ALTER TABLE [dbo].[Company_Members] CHECK CONSTRAINT
[FK_Company_Members_Companies]
GO
```

```
ALTER TABLE [dbo].[Company_Members] WITH CHECK ADD CONSTRAINT
[FK_Company_Members_Indyvidual_clients] FOREIGN KEY([Indyvidual_Client_ID])
REFERENCES [dbo].[Indyvidual_clients] ([client_ID])
GO
```

```
ALTER TABLE [dbo].[Company_Members] CHECK CONSTRAINT
[FK_Company_Members_Indyvidual_clients]
GO
```

5.2. Dania oraz składniki

5.2.1. Tabela Dishes

Dishes			
	Column Name	Data Type	Allow Nulls
🔑	dish_ID	int	<input type="checkbox"/>
	category_ID	int	<input type="checkbox"/>
	dish_name	varchar(30)	<input type="checkbox"/>
	price	money	<input type="checkbox"/>
	execution_time	int	<input type="checkbox"/>
	locked_date	date	<input checked="" type="checkbox"/>
	unlocked_date	date	<input checked="" type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela opisuje wszystkie dania, jakie są przyrządzane przez daną restaurację. Jest połączona z tabelą ingredients za pomocą tabeli łączącej Dish_details (relacja wiele do wielu). Ponadto połączona została z tabelą Restaurants za pomocą klucza obcego restaurant_ID, gdyż każda restauracja ma własną osobną listę dań, które udostępnia (relacja jeden do wielu).

Warunki integralnościowe:

- Cena dania musi być większa od 0
- Czas wykonania dania (execution_time) musi być większy od 0
- Data zablokowania dania (locked_date) musi być wcześniejszą datą niż data odblokowania dania (unlocked_date)
- Domyślna wartość (Default) dla daty zablokowania dania(locked_date) to data dzisiejsza

dish_ID	Klucz główny, identyfikuje danie
category_ID	Klucz obcy, identyfikuje kategorie, do jakiej należy danie, pozwoli zidentyfikować danie luksusowe, wymagające specjalnych składników koniecznych do zamówienia wcześniej
dish_name	Nazwa dania
price	Cena dania
execution_time	Średni czas przygotowania dania w minutach
locked_date	Data zablokowania dania, jeżeli danie aktualnie nie może zostać wykonane z przyczyn losowych lub po zablokowaniu składnika dania
unlocked_date	Data odblokowania dania
restaurant_ID	Klucz obcy identyfikujący restaurację, przygotowującą to danie

```

CREATE TABLE [dbo].[Dishes](
    [dish_ID] [int] NOT NULL,
    [category_ID] [int] NOT NULL,
    [dish_name] [varchar](30) NOT NULL,
    [price] [money] NOT NULL,
    [execution_time] [int] NOT NULL,
    [locked_date] [date] NULL,
    [unlocked_date] [date] NULL,
    [restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Dishes_1] PRIMARY KEY CLUSTERED
(
    [dish_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [Dish_name] UNIQUE NONCLUSTERED
(
    [dish_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Dishes] ADD CONSTRAINT [DF_Dishes_locked_date] DEFAULT
(CONVERT([date],getdate())) FOR [locked_date]
GO

ALTER TABLE [dbo].[Dishes] WITH CHECK ADD CONSTRAINT
[FK_Dishes_Categories] FOREIGN KEY([category_ID])
REFERENCES [dbo].[Categories] ([category_ID])
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Categories]
GO

ALTER TABLE [dbo].[Dishes] WITH CHECK ADD CONSTRAINT
[FK_Dishes_Restaurants] FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [FK_Dishes_Restaurants]
GO

ALTER TABLE [dbo].[Dishes] WITH CHECK ADD CONSTRAINT [Dishes_dates] CHECK
(([locked_date]>=[unlocked_date]))
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [Dishes_dates]
GO

ALTER TABLE [dbo].[Dishes] WITH CHECK ADD CONSTRAINT
[Dishes_execution_time] CHECK (([execution_time]>(0)))
GO

ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [Dishes_execution_time]

```

GO

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

```
ALTER TABLE [dbo].[Dishes] CHECK CONSTRAINT [Dishes_Price]
GO
```

5.2.2. Tabela Dish_details

Dish Details			
	Column Name	Data Type	Allow Nulls
	dish_ID	int	<input type="checkbox"/>
	ingredient_ID	int	<input type="checkbox"/>
	Quantity	float	<input type="checkbox"/>
	Price	money	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela łącząca tabelę dań z tabelą składników.

Warunki integralnościowe:

- Liczebność składnika wchodząca w skład dania (Quantity) musi być większa od 0
- Cena składnika wchodzącego do dania (Price) musi być większa od 0

dish_ID	Klucz obcy, identyfikuje danie
ingredient_ID	Klucz obcy, identyfikuje składnik dania
quantity	Liczba opisująca ilość składnika wchodzącego w skład dania
price	Cena składnika w danym czasie

```
CREATE TABLE [dbo].[Dish_Details] (
    [dish_ID] [int] NOT NULL,
    [ingredient_ID] [int] NOT NULL,
    [Quantity] [float] NOT NULL,
    [Price] [money] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Dish_Details] WITH CHECK ADD CONSTRAINT
[FK_Dish_Details_Dishes] FOREIGN KEY([dish_ID])
REFERENCES [dbo].[Dishes] ([dish_ID])
GO
```

```
ALTER TABLE [dbo].[Dish_Details] CHECK CONSTRAINT [FK_Dish_Details_Dishes]
GO
```

```
ALTER TABLE [dbo].[Dish_Details] WITH CHECK ADD CONSTRAINT
[FK_Dish_Details_Igredients] FOREIGN KEY([ingredient_ID])
REFERENCES [dbo].[Igredients] ([ingredient_ID])
GO
```

```
ALTER TABLE [dbo].[Dish_Details] CHECK CONSTRAINT
[FK_Dish_Details_Igredients]
GO
```

```
ALTER TABLE [dbo].[Dish_Details] WITH CHECK ADD CONSTRAINT
[Dish_Details_Price] CHECK (([Price]>(0)))
GO
```

```
ALTER TABLE [dbo].[Dish_Details] CHECK CONSTRAINT [Dish_Details_Price]
GO
```

```
ALTER TABLE [dbo].[Dish_Details] WITH CHECK ADD CONSTRAINT
[Dish_Details_Quantity] CHECK (([Quantity]>(0)))
GO
```

```
ALTER TABLE [dbo].[Dish_Details] CHECK CONSTRAINT [Dish_Details_Quantity]
GO
```

5.2.3. Tabela Ingredients

Igredients			
	Column Name	Data Type	Allow Nulls
🔑	ingredient_ID	int	<input type="checkbox"/>
	supplier_ID	int	<input type="checkbox"/>
	ingredient_name	varchar(20)	<input type="checkbox"/>
	ingredient_in_stock	int	<input type="checkbox"/>
	ingredient_price	money	<input type="checkbox"/>
	ingredient_quantity	varchar(20)	<input type="checkbox"/>
	safe_amout	int	<input type="checkbox"/>
	extra_informations	varchar(50)	<input checked="" type="checkbox"/>
	locked_date	date	<input checked="" type="checkbox"/>
	unlocked_date	date	<input checked="" type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela zawierająca informacje na temat składników, z których robione są dania. Jest połączona z tabelą dostawców (Suppliers) relacją jeden do wielu (jeden dostawca może dostarczać wiele składników), co pozwoli na zamawianie brakujących składników, gdy ich ilość w magazynie spadnie poniżej pewnej ustalonej normy. Składniki również są połączone z tabelą restauracji relacją jeden do wielu.

Warunki integralnościowe:

- Nazwa składnika (Ingredient_name) musi być unikatowa
- Data zablokowania składnika (Locked_date) musi być wcześniejszą datą niż data odblokowania składnika (unlocked_date)
- Cena składnika (Ingredient_price) musi być większe od 0
- Ilość składnika w magazynie (Ingredient_in_stock) musi być większa lub równa 0
- Wartość bezpieczna składnika (Safe_amount) musi być większa lub równa 0
- Wartość domyślna dla daty zablokowania składnika (locked_date) to data dzisiejsza

ingredient_ID	Klucz główny, identyfikuje składnik
supplier_ID	Klucz obcy, identyfikuje jednoznacznie dostawcę
ingredient_name	Nazwa składnika
ingredient_in_stock	Ilość składnika w magazynie
ingredient_price	Cena składnika
ingredient_quantity	Ilość składnika na jednostkę składnika (np. 1litr)
safe_amount	Wartość bezpieczna składnika w magazynie, jeżeli ilość składnika spadnie poniżej tej wartości, należy wykonać zamówienie u dostawcy danego składnika
extra_information	Dodatkowe informacje na temat składnika, może zawierać informacje na temat alergenów, glutenu obecnego w składniku
locked_date	Data zablokowania składnika, jeżeli składnik jest od pewnej daty niedostępny tzn. nie jest możliwe zamówienie go lub firma z niego zrezygnowała
unlocked_date	Data odblokowania składnika, jeżeli składnik po jakimś czasie jest znowu możliwy do zamówienia
restaurant_ID	Klucz obcy, identyfikuje restaurację korzystającą ze składnika

```
CREATE TABLE [dbo].[Ingredients] (
    [ingredient_ID] [int] NOT NULL,
    [supplier_ID] [int] NOT NULL,
    [ingredient_name] [varchar](20) NOT NULL,
    [ingredient_in_stock] [int] NOT NULL,
    [ingredient_price] [money] NOT NULL,
    [ingredient_quantity] [varchar](20) NOT NULL,
    [safe_amount] [int] NOT NULL,
```



```

        [extra_informations] [varchar](50) NULL,
        [locked_date] [date] NULL,
        [unlocked_date] [date] NULL,
        [restaurant_ID] [int] NOT NULL,
        CONSTRAINT [PK_Igredients_1] PRIMARY KEY CLUSTERED
    (
        [ingredient_ID] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
    OFF) ON [PRIMARY],
        CONSTRAINT [IX_Igredients_name] UNIQUE NONCLUSTERED
    (
        [ingredient_name] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
    OFF) ON [PRIMARY]
    ) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Igredients] ADD CONSTRAINT [DF_Igredients_locked_date]
DEFAULT (CONVERT([date],getdate())) FOR [locked_date]
GO

ALTER TABLE [dbo].[Igredients] WITH CHECK ADD CONSTRAINT
[FK_Igredients_Restaurants] FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

ALTER TABLE [dbo].[Igredients] CHECK CONSTRAINT [FK_Igredients_Restaurants]
GO

ALTER TABLE [dbo].[Igredients] WITH CHECK ADD CONSTRAINT
[FK_Igredients_Suppliers] FOREIGN KEY([supplier_ID])
REFERENCES [dbo].[Suppliers] ([supplier_ID])
GO

ALTER TABLE [dbo].[Igredients] CHECK CONSTRAINT [FK_Igredients_Suppliers]
GO

ALTER TABLE [dbo].[Igredients] WITH CHECK ADD CONSTRAINT
[Igredients_dates] CHECK ((([locked_date]<=[unlocked_date])))
GO

ALTER TABLE [dbo].[Igredients] CHECK CONSTRAINT [Igredients_dates]
GO

ALTER TABLE [dbo].[Igredients] WITH CHECK ADD CONSTRAINT
[Igredients_price] CHECK ((([ingredient_price]>(0)))
GO

ALTER TABLE [dbo].[Igredients] CHECK CONSTRAINT [Igredients_price]
GO

ALTER TABLE [dbo].[Igredients] WITH CHECK ADD CONSTRAINT
[Igredients_safe] CHECK ((([safe_amout]>=(0)))
GO

```

```
ALTER TABLE [dbo].[Igreredients] CHECK CONSTRAINT [Igreredients_safe]
GO
```

```
ALTER TABLE [dbo].[Igreredients] WITH CHECK ADD CONSTRAINT
[Igreredients_stock] CHECK (([ingredient_in_stock]>=(0)))
GO
```

```
ALTER TABLE [dbo].[Igreredients] CHECK CONSTRAINT [Igreredients_stock]
GO
```

5.2.4. Tabela Categories


Categories			
	Column Name	Data Type	Allow Nulls
	category_ID	int	<input type="checkbox"/>
	category_name	varchar(15)	<input type="checkbox"/>
	category_description	varchar(40)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Tabela opisująca kategorię dania, pozwala określić, czy danie jest luksusowe wymaga zamówienia składników

Warunki integralnościowe:

- Nazwa kategorii (category_name) musi być unikatowa

category_ID	Klucz główny, identyfikuje kategorię
category_name	Nazwa kategorii
category_Description	Dodatkowy opis kategorii, pozwalający klientom mieć ogólny obraz na dania należące do danej kategorii

```
CREATE TABLE [dbo].[Categories](
    [category_ID] [int] NOT NULL,
    [category_name] [varchar](15) NOT NULL,
    [category_description] [varchar](40) NULL,
    CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
    [category_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [U_Categories] UNIQUE NONCLUSTERED
(
    [category_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

5.2.5. Tabela Menu


Menu			
	Column Name	Data Type	Allow ...
	Menu_ID	int	<input type="checkbox"/>
	Dish_ID	int	<input type="checkbox"/>
	Add_date	date	<input type="checkbox"/>
	Remove_date	date	<input type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela menu opisuje aktualne pozycje dań dostępne dla klientów w danym czasie. Jest połączona z daniami relacją jeden do jednego oraz z tabelą restauracji relacją jeden do wielu. Ponadto występuje również połączenie z tabelą zamówień (Orders) za pomocą tabeli łączącej Order_Details.

Warunki integralnościowe:

- Data dodania dania do menu (Add_date) musi być wcześniejsza niż data usunięcia dania z menu (Remove_date)
- Wartość domyślna (Default) dla daty dodania do menu (add_date) to data dzisiejsza
- Wartość domyślna (Default) dla daty usunięcia z menu (add_date) to data dzisiejsza + 2 tyg

Menu_ID	Klucz główny, identyfikuje pozycję menu
Dish_ID	Klucz obcy, identyfikuje danie, którego dotyczy pozycja
Add_date	Data dodania dania do menu, data pozwoli na trzymanie porządku w menu, usuwanie dań, które już 2 tygodnie znajdują się menu, uniemożliwia wprowadzenie ich ponownie do menu przed upłynięciem określonego czasu
Remove_date	Data usunięcia dania z menu, dania aktualnie dostępne dla klientów to dania, które zostały dodane do menu przed datą zamówienia oraz jeszcze nie zostały usunięte z menu
restaurant_ID	Klucz obcy, identyfikuje restaurację, do której należy pozycja menu

```
CREATE TABLE [dbo].[Menu] (
    [Menu_ID] [int] NOT NULL,
    [Dish_ID] [int] NOT NULL,
    [Add_date] [date] NOT NULL,
    [Remove_date] [date] NOT NULL,
    [restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Menu] PRIMARY KEY CLUSTERED
(
    [Menu_ID] ASC
```

```

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

ALTER TABLE [dbo].[Menu] ADD CONSTRAINT [DF_Menu_Add_date] DEFAULT
(CONVERT([date],getdate())) FOR [Add_date]
GO

ALTER TABLE [dbo].[Menu] ADD CONSTRAINT [DF_Menu_Remove_date] DEFAULT
(CONVERT([date],dateadd(day,14,getdate()))) FOR [Remove_date]
GO

ALTER TABLE [dbo].[Menu] WITH CHECK ADD CONSTRAINT [FK_Menu_Dishes]
FOREIGN KEY([Dish_ID])
REFERENCES [dbo].[Dishes] ([dish_ID])
GO

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

ALTER TABLE [dbo].[Menu] WITH CHECK ADD CONSTRAINT [FK_Menu_Restaurants]
FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

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

ALTER TABLE [dbo].[Menu] WITH CHECK ADD CONSTRAINT [CK_Menu] CHECK
(([Add_date]<[Remove_date]))
GO

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

```

5.3. Zamówienia

5.3.1. Tabela Orders


Orders			
	Column Name	Data Type	Allow Nulls
	order_ID	int	<input type="checkbox"/>
	client_ID	int	<input type="checkbox"/>
	employee_ID	int	<input type="checkbox"/>
	takeaway	bit	<input type="checkbox"/>
	order_date	date	<input type="checkbox"/>
	pickup_time	datetime	<input type="checkbox"/>
	discount_ID	int	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Tabela opisująca szczegółowo informacje dotyczące zamówień. Jest połączona z tabelą dań za pomocą tabeli łączącej Order_details (relacja wiele do wielu). Tabela zawiera informację, który pracownik jest odpowiedzialny za dane zamówienie poprzez przechowywanie klucza obcego z tabeli Employees, która zawiera dane pracowników. Jest to połączenie opisujące relację jeden do wielu, gdyż każde danie ma dokładnie jedną osobę odpowiedzialną. Tabela zamówień jest również połączona z tabelą Personal_reservations (rezerwacje klientów indywidualnych), co umożliwia zapewnienie spełnionego wymagania: każdy klient indywidualny może złożyć rezerwację tylko wraz ze złożeniem zamówienia ponad daną kwotę.

Każde zamówienie ma również przypisany rabat, który w danym czasie klient wykorzystał, stąd tabela jest też połączona z tabelą rabatów - Discounts (każde zamówienie może mieć maksymalnie jeden przypisany rabat).

W celu określenia, czy dane zamówienie nie zostało jeszcze zapłacone, a więc czy zamówienie wymaga uiszczenia kwoty należnej podczas wizyty w restauracji, tabela jest również połączona do tabeli Orders_unpaid. Jeżeli zamówienie wystąpi również w orders_unpaid oznacza to, iż jeszcze nie zostało zapłacone.

Warunki integralnościowe:

- Wartość domyślna dla daty i godziny zamówienia to data aktualna

Order_ID	Klucz główny, identyfikuje jednoznacznie zamówienie
Client_ID	Klucz obcy, identyfikuje klienta zamawiającego
Employee_ID	Klucz obcy, identyfikuje pracownika odpowiedzialnego za skompletowanie zamówienia
Takeaway	Informuje, czy zamówienie jest na wynos (1/0)
Order_date	Data zamówienia
Pickup_time	Godzina zamówienia, na którą godzinę zostało złożone zamówienie
Discount_ID	Rabat dotyczący zamówienia, gdyż klient składający zamówienie może mieć więcej niż jeden rabat przydzielony (stały lub jednorazowy)

```

CREATE TABLE [dbo].[Orders] (
    [order_ID] [int] NOT NULL,
    [client_ID] [int] NOT NULL,
    [employee_ID] [int] NOT NULL,
    [takeaway] [bit] NOT NULL,
    [order_date] [date] NOT NULL,
    [pickup_time] [time](7) NOT NULL,
    [discount] [float] NOT NULL,
    CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
    [order_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders] ADD CONSTRAINT [DF_Orders_order_date] DEFAULT
(CONVERT([date],getdate())) FOR [order_date]
GO

ALTER TABLE [dbo].[Orders] ADD CONSTRAINT [DF_Orders_pickup_time] DEFAULT
(CONVERT([time],getdate())) FOR [pickup_time]
GO

ALTER TABLE [dbo].[Orders] WITH CHECK ADD CONSTRAINT [FK_Orders_Clients]
FOREIGN KEY([client_ID])
REFERENCES [dbo].[Clients] ([client_ID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Clients]
GO

ALTER TABLE [dbo].[Orders] WITH CHECK ADD CONSTRAINT [FK_Orders_Employee]
FOREIGN KEY([employee_ID])
REFERENCES [dbo].[Employees] ([employee_ID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Employee]

```

GO

```
ALTER TABLE [dbo].[Orders] WITH CHECK ADD CONSTRAINT [CK_Orders_discount]
CHECK (([discount]>=(0) AND [discount]<=(1)))
GO
```

```
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders_discount]
GO
```

5.3.2. Tabela Order_details

Order_details			
	Column Name	Condensed Type	Nullable
	Order_ID	int	No
	Menu_ID	int	No
	Quantity	int	No
	Price	money	No

Tabela łącząca tabelę Orders oraz tabelę Menu. Pozwala na zamodelowanie relacji wiele do wielu wraz z przechowywaniem szczegółowych informacji, na temat zamówienia.

Warunki integralnościowe:

- Cena (Price) większa od 0
- Ilość (Quantity) większa od zera

Order_ID	Klucz obcy, identyfikuje jednoznacznie zamówienie
Menu_ID	Klucz obcy, jednoznacznie identyfikują pozycję menu
Quantity	Liczba zamówionego dania
Price	Cena dania w dniu zamówienia

```
CREATE TABLE [dbo].[Order_details] (
    [Order_ID] [int] NOT NULL,
    [Menu_ID] [int] NOT NULL,
    [Quantity] [int] NOT NULL,
    [Price] [money] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Order_details] WITH CHECK ADD CONSTRAINT
[FK_Order_details_Menu] FOREIGN KEY([Menu_ID])
REFERENCES [dbo].[Menu] ([Menu_ID])
GO
```

```
ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [FK_Order_details_Menu]
GO
```

```
ALTER TABLE [dbo].[Order_details] WITH CHECK ADD CONSTRAINT
[FK_Order_details_Orders] FOREIGN KEY([Order_ID])
REFERENCES [dbo].[Orders] ([order_ID])
GO
```

```
ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT
[FK_Order_details_Orders]
GO
```

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

```
ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [CK_Order_details_price]
GO
```

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

```
ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT
[CK_Order_details_quantity]
GO
```

5.3.3. Tabela Orders_unpaid

Orders Unpaid			
	Column Name	Data Type	Allow ...
	order_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela posiada listę zamówień, które jeszcze nie zostały zapłacone.

Order_ID	Numer identyfikujący zamówienie, które jeszcze nie zostało zapłacone
----------	--

```
CREATE TABLE [dbo].[Orders_Unpaid] (
    [order_ID] [int] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Orders_Unpaid] WITH CHECK ADD CONSTRAINT
[FK_Orders_Unpaid_Orders] FOREIGN KEY([order_ID])
REFERENCES [dbo].[Orders] ([order_ID])
GO
```

```
ALTER TABLE [dbo].[Orders_Unpaid] CHECK CONSTRAINT
[FK_Orders_Unpaid_Orders]
GO
```


5.4. Rezerwacje i stoliki

5.4.1. Tabela Reservations


Reservations			
	Column Name	Data Type	Allow Nulls
	reservation_ID	int	<input type="checkbox"/>
	client_ID	int	<input type="checkbox"/>
	reservation_date	date	<input type="checkbox"/>
	reservation_time	time(7)	<input type="checkbox"/>
	restriction_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela reservations zawiera ogólne informacje na temat rezerwacji składanych przez klientów. Rezerwacje łączą się relacją jeden do jednego z 2 tabelami: Personal_Reservations i Company_Reservations (tabele zawierające szczegółowe informacje na temat rezerwacji dokonywanych przez kolejno indywidualnych klientów oraz firmy).

Rezerwacje są dokonywane na pozycje z tabeli restrictions, a więc na stoliki z narzuconymi ograniczeniami miejsc.

Reservation_ID	Klucz główny, identyfikuje rezerwacje
Client_ID	Klucz obcy, identyfikuje klienta składającego rezerwacje
Reservation_date	Data rezerwacji
Reservation_time	Godzina rezerwacji
Restriction_ID	Klucz obcy, stół w danej restauracji wraz z ograniczeniem liczby miejsc

```
CREATE TABLE [dbo].[Reservations](
    [reservation_ID] [int] NOT NULL,
    [client_ID] [int] NOT NULL,
    [reservation_date] [date] NOT NULL,
    [reservation_time] [time](7) NOT NULL,
    [restriction_ID] [int] NOT NULL,
    CONSTRAINT [PK_Reservation] PRIMARY KEY CLUSTERED
(
    [reservation_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations] WITH CHECK ADD CONSTRAINT
[FK_Reservation_Company_Reservations] FOREIGN KEY([reservation_ID])
REFERENCES [dbo].[Company_Reservations] ([reservation_ID])
GO
```

```
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[FK_Reservation_Company_Reservations]
GO
```

```
ALTER TABLE [dbo].[Reservations] WITH CHECK ADD CONSTRAINT
[FK_Reservation_Personal_Reservation] FOREIGN KEY([reservation_ID])
REFERENCES [dbo].[Personal_Reservations] ([reservation_ID])
GO
```

```
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[FK_Reservation_Personal_Reservation]
GO
```

```
ALTER TABLE [dbo].[Reservations] WITH CHECK ADD CONSTRAINT
[FK_Reservation_Restrictions] FOREIGN KEY([restriction_ID])
REFERENCES [dbo].[Restrictions] ([restriction_ID])
GO
```

```
ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[FK_Reservation_Restrictions]
GO
```

5.4.2. Tabela Personal_Reservations


Personal Reservations			
	Column Name	Data Type	Allow Nulls
	reservation_ID	int	<input type="checkbox"/>
	order_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela opisuje szczegółowe informacje na temat rezerwacji klientów indywidualnych. Przy takiej rezerwacji konieczne jest złożenie zamówienia powyżej ustalonej kwoty, stąd przechowywany jest tutaj klucz obcy z tabeli zamówień. Występuje relacja 1 do 1 (rezerwacja może mieć maksymalnie jedną pozycję w personal_reservations)

Reservation_ID	Unikalny numer identyfikujący rezerwację, klucz główny tabeli
Order_ID	Klucz obcy, identyfikujący numer zamówienia

```
CREATE TABLE [dbo].[Personal_Reservations](
    [reservation_ID] [int] NOT NULL,
    [order_ID] [int] NOT NULL,
    CONSTRAINT [PK_Personal_Reservation] PRIMARY KEY CLUSTERED
(
    [reservation_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Personal_Reservations] WITH CHECK ADD CONSTRAINT
[FK_Personal_Reservation_Orders] FOREIGN KEY([order_ID])
REFERENCES [dbo].[Orders] ([order_ID])
GO
```

```
ALTER TABLE [dbo].[Personal_Reservations] CHECK CONSTRAINT
[FK_Personal_Reservation_Orders]
GO
```

5.4.3. Tabela Company_Reservations


Company Reservations			
	Column Name	Data Type	Allow...
	resevation_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela przechowuje listę ID rezerwacji, które zostały złożone przez firmę. Tabela łączy się z tabelą rezerwacji imiennych (Company_Name_Reservations).

Reservation_ID	Klucz główny, identyfikuje rezerwację
----------------	---------------------------------------

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

5.4.4. Tabela Company_Name_Reservations

Company Name Reservation			
	Column Name	Data Type	Allow...
	reservation_ID	int	<input type="checkbox"/>
	client_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela przechowuje rezerwacje firmowe imienne, modeluje relację wiele do wielu, gdyż rezerwacja imienna mogła być złożona na więcej niż jedną osobę. Łączy się z tabelą Individual_clients.

Reservation_ID	Klucz obcy, numer rezerwacji
Client_ID	Klucz obcy, identyfikuje klienta indywidualnego

```
CREATE TABLE [dbo].[Company_Name_Reservation] (
    [reservation_ID] [int] NOT NULL,
    [client_ID] [int] NOT NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Company_Name_Reservation] WITH CHECK ADD CONSTRAINT
[FK_Company_Name_Reservation_Company_Reservations] FOREIGN
KEY([reservation_ID])
REFERENCES [dbo].[Company_Reservations] ([reservation_ID])
GO
```

```
ALTER TABLE [dbo].[Company_Name_Reservation] CHECK CONSTRAINT
[FK_Company_Name_Reservation_Company_Reservations]
GO
```

```
ALTER TABLE [dbo].[Company_Name_Reservation] WITH CHECK ADD CONSTRAINT
[FK_Company_Name_Reservation_Individual_clients] FOREIGN KEY([client_ID])
REFERENCES [dbo].[Individual_clients] ([client_ID])
GO
```

```
ALTER TABLE [dbo].[Company_Name_Reservation] CHECK CONSTRAINT
[FK_Company_Name_Reservation_Individual_clients]
GO
```

5.4.5. Tabela Restrictions


Restrictions			
	Column Name	Data Type	Allow Nulls
	restriction_ID	int	<input type="checkbox"/>
	table_ID	int	<input type="checkbox"/>
	start_date	date	<input type="checkbox"/>
	end_date	date	<input type="checkbox"/>
	limit_chairs	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela opisuje aktualną sytuację stolikową dla danej restauracji, które stoliki są wolne, w jakim czasie, ile miejsc posiadają ze względu na obecne ograniczenia związane z obostrzeniami rządowymi. Pozycje z tej tabeli są dopiero rezerwowane przez klientów. Tabela jest połączona z tabelą Tables, przechowującą informacje na temat stolików oraz maksymalnej liczby krzeseł mogących być przy każdym ze stolików.

Warunki integralnościowe:

- Data rozpoczęcia ograniczenia (Start_date) musi być wcześniejszą datą niż data końca ograniczenia (end_date)
- Liczba krzeseł (limit_chairs) musi być większa bądź równa 0
- Data rozpoczęcia ograniczenia (Start_date) przyjmują jako domyślną wartość datę dzisiejszą

Restriction_ID	Klucz główny, numer ograniczenia
Table_ID	Klucz obcy, numer stolika
Start_date	Od kiedy ograniczenie obowiązuje
End_date	Do kiedy ograniczenie obowiązuje
Limit_chairs	Liczba krzeseł, mniejsza niż maksymalna liczba krzeseł przy stoliku

```
CREATE TABLE [dbo].[Restrictions] (
    [restriction_ID] [int] NOT NULL,
    [table_ID] [int] NOT NULL,
    [start_date] [date] NOT NULL,
    [end_date] [date] NOT NULL,
    [limit_chairs] [int] NOT NULL,
    CONSTRAINT [PK_Restrictions] PRIMARY KEY CLUSTERED
(
    [restriction_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Restrictions] ADD CONSTRAINT
[DF_Restrictions_start_date] DEFAULT (CONVERT([date],getdate())) FOR
[start_date]
GO

ALTER TABLE [dbo].[Restrictions] WITH CHECK ADD CONSTRAINT
[FK_Restrictions_Tables] FOREIGN KEY([table_ID])
REFERENCES [dbo].[Tables] ([table_ID])
GO

ALTER TABLE [dbo].[Restrictions] CHECK CONSTRAINT [FK_Restrictions_Tables]
GO

ALTER TABLE [dbo].[Restrictions] WITH CHECK ADD CONSTRAINT
[CK_Restrictions_dates] CHECK (([start_date]<=[end_date]))
GO

ALTER TABLE [dbo].[Restrictions] CHECK CONSTRAINT [CK_Restrictions_dates]
GO

ALTER TABLE [dbo].[Restrictions] WITH CHECK ADD CONSTRAINT
[CK_Restrictions_limit] CHECK ((([limit_chairs]>=(0)))
GO

ALTER TABLE [dbo].[Restrictions] CHECK CONSTRAINT [CK_Restrictions_limit]
GO
```

5.4.6. Tabela Tables

Tables			
	Column Name	Data Type	Allow Nulls
🔑	table_ID	int	<input type="checkbox"/>
	chairs_ammount	int	<input type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela opisująca informację o krzesłach znajdujących się w restauracjach. Połączona jest z tabelą Restrictions relacją jeden do wiele (można nałożyć wiele restrykcji w różnych odstępach czasu dla jednego stolika), oraz z tabelą Restaurants relacją jeden do wiele (w jednej restauracji może znajdować się wiele stolików).

Warunki integralnościowe:

- Maksymalna liczba miejsc przy stoliku (chairs_amount) musi być większa od 0

Table_ID	Unikalny numer stolika
Chairs_amount	Maksymalna ilość miejsc przy stoliku
restaurant_ID	Numer restauracji, w której znajduje się dany stolik

```
CREATE TABLE [dbo].[Tables](
    [table_ID] [int] NOT NULL,
    [chairs_ammount] [int] NOT NULL,
    [restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
    [table_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Tables] WITH CHECK ADD CONSTRAINT
[FK_Tables_Restaurants] FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO
```

```
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [FK_Tables_Restaurants]
GO
```

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

```
ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [chairs_amount]
GO
```

5.5. Rabaty

5.5.1. Tabela Discounts

Discounts			
	Column Name	Data Type	Allow Nulls
🔑	discount_ID	int	<input type="checkbox"/>
	client_ID	int	<input type="checkbox"/>
	discount_value	float	<input type="checkbox"/>
	start_date	date	<input type="checkbox"/>
	end_date	date	<input type="checkbox"/>
	discount_type_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela, w której przechowywane są informacje o przyznanych klientom rabatach. Połączona ona jest z tabelą Discount_dictionary relacją jeden do wiele (typ danego rabatu może się powtarzać) poprzez Discount_type_ID, oraz z tabelą Clients relacją jeden do wiele (klient może uzyskać wiele rabatów).

Warunki integralnościowe:

- Data, od której obowiązuje rabat (Start_date) musi być wcześniejszą datą niż data, do kiedy ten rabat obowiązuje (end_date)
- Zniżka (Discount_value) jest większa bądź równa 0 i jednocześnie mniejsza bądź równa 1
- Data, od której obowiązuje rabat (Start_date) jest jako wartość domyślną przyjmuje dzień dzisiejszy

Discount_ID	Unikatowy numer rabatu
Client_ID	Numer klienta, który uzyskał rabat
Discount_Value	Wartość rabatu
Start_date	Data, od momentu, w którym rabat zaczął obowiązywać
End_date	Data zakończenia rabatu
Discount_type_ID	Numer typu rabatu w słowniku rabatów

```
CREATE TABLE [dbo].[Discounts] (
    [discount_ID] [int] NOT NULL,
    [client_ID] [int] NOT NULL,
    [discount_value] [float] NOT NULL,
    [start_date] [date] NOT NULL,
    [end_date] [date] NOT NULL,
    [discount_type_ID] [int] NOT NULL,
    CONSTRAINT [PK_Discounts_1] PRIMARY KEY CLUSTERED
(
    [discount_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```

ALTER TABLE [dbo].[Discounts] ADD CONSTRAINT [DF_Discounts_start_date]
DEFAULT (CONVERT([date],getdate())) FOR [start_date]
GO

ALTER TABLE [dbo].[Discounts] WITH CHECK ADD CONSTRAINT
[FK_Discounts_Clients] FOREIGN KEY([client_ID])
REFERENCES [dbo].[Clients] ([client_ID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [FK_Discounts_Clients]
GO

ALTER TABLE [dbo].[Discounts] WITH CHECK ADD CONSTRAINT
[FK_Discounts_Discount_dictionary] FOREIGN KEY([discount_type_ID])
REFERENCES [dbo].[Discount_dictionary] ([Discount_type_ID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT
[FK_Discounts_Discount_dictionary]
GO

ALTER TABLE [dbo].[Discounts] WITH CHECK ADD CONSTRAINT
[FK_Discounts_Discount_types] FOREIGN KEY([discount_type_ID])
REFERENCES [dbo].[Discount_types] ([discount_type_ID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT
[FK_Discounts_Discount_types]
GO

ALTER TABLE [dbo].[Discounts] WITH CHECK ADD CONSTRAINT [CK_Discounts]
CHECK (([discount_value]<=(1) AND [discount_value]>=(0)))
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [CK_Discounts]
GO

ALTER TABLE [dbo].[Discounts] WITH CHECK ADD CONSTRAINT [Discounts_dates]
CHECK (([start_date]<=[end_date]))
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [Discounts_dates]
GO

```


5.5.2. Tabela Discount_dictionary


Discount_dictionary			
	Column Name	Data Type	Allow N...
	Discount_type_ID	int	<input type="checkbox"/>
	Count_Of_Orders	int	<input checked="" type="checkbox"/>
	Value_Orders	money	<input type="checkbox"/>
	Count_Of_Days	int	<input checked="" type="checkbox"/>
	Discount_Value	float	<input type="checkbox"/>
	Max_Discount_Val...	float	<input checked="" type="checkbox"/>
	Discount_descripti...	varchar(50)	<input type="checkbox"/>
	One_Time_Discount	bit	<input type="checkbox"/>
	From_date	date	<input type="checkbox"/>
	To_date	date	<input type="checkbox"/>
	Restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela będąca słownikiem rabatów. W niej znajdują się informacje dotyczące warunków uzyskania rabatu oraz czas, czas od kiedy dane warunki obowiązują, oraz w jakiej restauracji można uzyskać dany rabat. Tabela jest połączona z tabelą Discounts relacją jeden do wiele (dany typ rabatu może się powtarzać), oraz z tabelą Restaurants relacją jeden do wiele (jedna restauracja może mieć kilka różnych rabatów).

Warunki integralnościowe:

- Data, od kiedy dany typ rabatu obowiązuje(From_date) musi być wcześniejszą datą od daty, do której dany typ rabatu obowiązuje (To_date)
- Liczba dni, przez którą obowiązuje rabat (Count_Of_Days) musi być większa bądź równa od 0
- Liczba zamówień konieczna do uzyskania rabatu (Count_Of_Orders) musi być większa bądź równa 0
- Wartość rabatu (Discount Value) musi być większa od 0 oraz mniejsza bądź równa maksymalnemu rabatowi lub 1 w przypadku, gdy maksymalny rabat(Max_Discount_Value) jest nullem
- Maksymalny rabat (Max_Discount_Value) musi być większy bądź równy 0, oraz mniejszy bądź równy 1
- Data, od kiedy dany typ rabatu obowiązuje(From_date) przyjmuje jako wartość domyślną dzień dzisiejszy

Discount_type_ID	Unikatowy numer typu rabatu
Count_Of_Orders	Liczba zamówień
Value_Orders	Kwota zamówienia/zamówień
Count_OF_Days	Liczba dni
Discount_Value	Wartość rabatu
Max_Discount_Value	Maksymalna wartość rabatu, jaką można uzyskać
Discount_description	Opis typu rabatu
One_Time_Discount	Określa, czy można danego rabatu użyć więcej niż jeden raz
From_date	Data, od kiedy dany typ rabatu obowiązuje
To_date	Data, zakończenia obowiązywania rabatu
restaurant_ID	Restauracja, w której typ rabatu obowiązuje

```

CREATE TABLE [dbo].[Discount_dictionary] (
    [Discount_type_ID] [int] NOT NULL,
    [Count_Of_Orders] [int] NULL,
    [Value_Orders] [money] NOT NULL,
    [Count_Of_Days] [int] NULL,
    [Discount_Value] [float] NOT NULL,
    [Max_Discount_Value] [float] NULL,
    [Discount_description] [varchar](50) NOT NULL,
    [One_Time_Discount] [bit] NOT NULL,
    [From_date] [date] NOT NULL,
    [To_date] [date] NOT NULL,
    [Restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Discount_dictionary] PRIMARY KEY CLUSTERED
(
    [Discount_type_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discount_dictionary] ADD CONSTRAINT
[DF_Discount_dictionary_From_date] DEFAULT (CONVERT([date],getdate())) FOR
[From_date]
GO

ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT
[FK_Discount_dictionary_Restaurants] FOREIGN KEY([Restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT
[FK_Discount_dictionary_Restaurants]
GO

ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT
[CK_Discount_dictionary] CHECK ((([Discount_Value]>=(0) AND
[Discount_Value]<=isnull([Max_Discount_Value],(1)))))
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT
[CK_Discount_dictionary]
GO

```

```

ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT [Dates]
CHECK (([From_date]<=[To_date]))
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT [Dates]
GO

ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT [Days]
CHECK (([Count_Of_Days]>=(0)))
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT [Days]
GO

ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT
[Max_discount] CHECK (([Max_Discount_Value]>=(0) AND
[Max_Discount_Value]<=(1)))
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT [Max_discount]
GO


ALTER TABLE [dbo].[Discount_dictionary] WITH CHECK ADD CONSTRAINT
[Orders_dictionary] CHECK (([Count_Of_Orders]>=(0)))
GO

ALTER TABLE [dbo].[Discount_dictionary] CHECK CONSTRAINT
[Orders_dictionary]
GO

```

5.6. Dostawcy

5.6.1. Tabela Suppliers

Suppliers			
	Column Name	Data Type	Allow Nulls
	supplier_ID	int	<input type="checkbox"/>
	town_ID	int	<input type="checkbox"/>
	NIP_number	varchar(10)	<input type="checkbox"/>
	supplier_name	varchar(30)	<input type="checkbox"/>
	supplier_phone	varchar(12)	<input type="checkbox"/>
	supplier_email	varchar(20)	<input checked="" type="checkbox"/>
	street	varchar(20)	<input type="checkbox"/>
	postal_code	varchar(10)	<input type="checkbox"/>
	bank_account	varchar(26)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

W tabeli są przechowywane szczegółowe informacje na temat dostawców poszczególnych składników. Tabela jest połączona relacją jeden do wiele z tabelą Ingredients (jeden producent może mieć w swojej ofercie wiele składników), oraz relacją jeden do wiele z tabelą Towns (w jednym mieście może się znajdować kilka siedzib różnych firm).

Warunki integralnościowe:

- Nazwa dostawcy (supplier_name) musi być unikalna
- Numer NIP (NIP_number) musi być unikalny i składać się tylko z cyfr
- Kod pocztowy (Postal_code) postaci XX-XXX, XXXXXX, XXXXXX, gdzie X to cyfra
- Numer telefonu (Supplier_phone) musi być unikatowy, oraz składać się z samych cyfr
- Numer Konta Bankowego (Bank_account) musi składać się tylko z cyfr i być unikatowy
- Adres email (Supplier_email) musi być unikatowy i zawierać znak "@"

Supplier_ID	Unikatowy numer firmy dostawczej
Town_ID	Numer miasta, w którym firma ma swoją siedzibę
NIP_number	Numer NIP
Supplier_name	Nazwa dostawcy
Supplier_phone	Numer telefonu do dostawcy
Supplier_email	Adres email
Street	Ulica, na której firma ma swoją siedzibę
Postal_code	Kod pocztowy
Bank_account	Numer konta bankowego

```
CREATE TABLE [dbo].[Suppliers] (
    [supplier_ID] [int] NOT NULL,
    [town_ID] [int] NOT NULL,
    [NIP_number] [varchar](10) NOT NULL,
    [supplier_name] [varchar](30) NOT NULL,
    [supplier_phone] [varchar](12) NOT NULL,
    [supplier_email] [varchar](20) NULL,
    [street] [varchar](20) NOT NULL,
    [postal_code] [varchar](10) NOT NULL,
    [bank_account] [varchar](26) NULL,
    CONSTRAINT [PK_Suppliers] PRIMARY KEY CLUSTERED
(
    [supplier_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [IX_Suppliers_bank_account] UNIQUE NONCLUSTERED
(
    [bank_account] 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 [IX_Suppliers_nip] UNIQUE NONCLUSTERED
(
    [NIP_number] 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 [IX_Suppliers_phone_number] UNIQUE NONCLUSTERED
(
    [supplier_phone] 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 [IX_Suppliers_supplier_email] UNIQUE NONCLUSTERED
(
    [supplier_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],
    CONSTRAINT [IX_Suppliers_supplier_name] UNIQUE NONCLUSTERED
(
    [supplier_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[FK_Suppliers_Towns] FOREIGN KEY([town_ID])
REFERENCES [dbo].[Towns] ([town_ID])
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [FK_Suppliers_Towns]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[CK_Suppliers_bank_account] CHECK ((isnumeric([bank_account])=(1)))
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [CK_Suppliers_bank_account]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[CK_Suppliers_email] CHECK (([supplier_email] like '%@%'))
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [CK_Suppliers_email]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[CK_Suppliers_nip_number] CHECK ((isnumeric([NIP_number])=(1)))
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [CK_Suppliers_nip_number]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[CK_Suppliers_phone_number] CHECK ((isnumeric([supplier_phone])=(1)))
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [CK_Suppliers_phone_number]
GO

ALTER TABLE [dbo].[Suppliers] WITH CHECK ADD CONSTRAINT
[CK_Suppliers_postal_code] CHECK (([postal_code] like '[0-9][0-9]-[0-9][0-9][0-9]' OR [postal_code] like '[0-9][0-9][0-9][0-9][0-9]' OR [postal_code]
like '[0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Suppliers] CHECK CONSTRAINT [CK_Suppliers_postal_code]

```

5.7. Pracownicy

5.7.1. Tabela Employees


Employees			
	Column Name	Data Type	Allow Nulls
	employee_ID	int	<input type="checkbox"/>
	position_ID	int	<input type="checkbox"/>
	first_name	varchar(40)	<input type="checkbox"/>
	last_name	varchar(40)	<input type="checkbox"/>
	part_time	varchar(15)	<input type="checkbox"/>
	rate_per_hour	money	<input type="checkbox"/>
	phone_number	varchar(12)	<input type="checkbox"/>
	email	varchar(50)	<input checked="" type="checkbox"/>
	hire_date	date	<input type="checkbox"/>
	photo_path	varchar(50)	<input checked="" type="checkbox"/>
	job_quit_date	date	<input checked="" type="checkbox"/>
	restaurant_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela Employees przechowuje szczegółowe informacje na temat pracowników pracujących w restauracjach. Jest ona połączona z tabelą Restaurants relacją jeden do wielu poprzez restaurant_ID (firma może zatrudniać wielu pracowników), z tabelą Positions relacją jeden do wielu poprzez position_ID, oraz z tabelą Orders relacją jeden do wielu poprzez employee_ID, (jeden pracownik może odpowiadać za wiele dań).

Warunki integralnościowe:

- Etat (Part_time) musi przyjmować jedną z opcji "Full_time", "Half_time", "Quarter_time"
- Imię i nazwisko (first_name i last_name) powinno się zaczynać od dużej litery
- Stawka godzinowa (rate_per_hour) musi być większa, bądź równa 0
- Numer telefonu (phone_number) musi być unikatowy, oraz składać się z samych cyfr
- Adres email (email) musi być unikatowy i zawierać znak "@"
- Data zatrudnienia (hire_date) musi być mniejsza od daty odejścia z firmy (o ile taka istnieje), (job_quit_date)
- Data zatrudnienia (hire_date) przyjmuje wartość domyślną jako dzień dzisiejszy

employee_ID	Unikalny numer pracownika
position_ID	Numer identyfikujący zawód pracownika
first_name	Imię
last_name	Nazwisko
part_time	Informacja czy pracownik jest zatrudniony na pełny etat lub pół etatu
rate_per_hour	Stawka godzinowa, którą pracownik zarabia
phone_number	Numer telefonu
email	Adres email
hire_date	Data zatrudnienia
photo_path	Ścieżka do pliku zawierająca zdjęcia pracownika
job_quit_date	Data odejścia z firmy, jeżeli pracownik opuścił już firmę, ale chcemy przechowywać dalej historię jego aktywności
restaurant_ID	Restauracja, w której pracownik pracuje

```

CREATE TABLE [dbo].[Employees] (
    [employee_ID] [int] NOT NULL,
    [position_ID] [int] NOT NULL,
    [first_name] [varchar](40) NOT NULL,
    [last_name] [varchar](40) NOT NULL,
    [part_time] [varchar](15) NOT NULL,
    [rate_per_hour] [money] NOT NULL,
    [phone_number] [varchar](12) NOT NULL,
    [email] [varchar](50) NULL,
    [hire_date] [date] NOT NULL,
    [photo_path] [varchar](50) NULL,
    [job_quit_date] [date] NULL,
    [restaurant_ID] [int] NOT NULL,
    CONSTRAINT [PK_Employee_1] PRIMARY KEY CLUSTERED
(
    [employee_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [IX_Employees] 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],
    CONSTRAINT [IX_Employees_1] UNIQUE NONCLUSTERED
(
    [phone_number] 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].[Employees] ADD CONSTRAINT [DF_Employees_hire_date]
DEFAULT (CONVERT([date],getdate())) FOR [hire_date]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT
[FK_Employee_Positions] FOREIGN KEY([position_ID])
REFERENCES [dbo].[Positions] ([position_ID])

```

```

GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employee_Positions]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT
[FK_Employees_Restaurants] FOREIGN KEY([restaurant_ID])
REFERENCES [dbo].[Restaurants] ([restaurant_ID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Restaurants]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT [CK_Employees]
CHECK (([hire_date]<=isnull([job_quit_date],getdate()))))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT [Employees_email]
CHECK (([email] like '%@%'))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_email]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT [Employees_names]
CHECK ((([first_name] like '[A-Z]%' AND [last_name] like '[A-Z]%')))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_names]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT
[Employees_part_time] CHECK (([part_time]='Full_time' OR
[part_time]='Half_time' OR [part_time]='Quarter_time'))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_part_time]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT [Employees_phone]
CHECK ((isnumeric([phone_number])=(1)))
GO


ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_phone]
GO

ALTER TABLE [dbo].[Employees] WITH CHECK ADD CONSTRAINT [Employees_rate]
CHECK (([rate_per_hour]>=(0)))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_rate]
GO

```


5.7.2. Tabela Positions

Positions			
	Column Name	Data Type	Allow ...
	position_ID	int	<input type="checkbox"/>
	position_name	varchar(20)	<input type="checkbox"/>
			<input type="checkbox"/>

W tabeli Position będą przechowywane informacje o stanowiskach, na jakich można pracować w restauracji. Takie jak np. kelner, kucharz, itp. Tabela ta jest połączona z tabelą Employees relacją jeden do wiele (jedno stanowisko może mieć kilku pracowników)

Warunki integralnościowe:

- Nazwa zawodu (Position_name) musi przyjmować jedną z opcji:
"Waiter", "Cook", "Manager", "Chef", "Dishwasher", "Cleaner"

position_ID	Numer identyfikujący zawód pracownika
position_name	Nazwa stanowiska


```
CREATE TABLE [dbo].[Positions] (
    [position_ID] [int] NOT NULL,
    [position_name] [varchar](20) NOT NULL,
    CONSTRAINT [PK_Positions] PRIMARY KEY CLUSTERED
(
    [position_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Positions] WITH CHECK ADD CONSTRAINT
[CK_Positions_name] CHECK (([position_name]='Cleaner' OR
[position_name]='Dishwasher' OR [position_name]='Chef' OR
[position_name]='Manager' OR [position_name]='Cook' OR
[position_name]='Waiter'))
GO
```

```
ALTER TABLE [dbo].[Positions] CHECK CONSTRAINT [CK_Positions_name]
GO
```

5.8. Adresy

5.8.1. Tabela Towns


Towns			
	Column Name	Data Type	Allow ...
	town_ID	int	<input type="checkbox"/>
	town_name	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela, w której przechowywane są informacje o miastach. Połączona jest z tabelą Towns_connections relacją jeden do wiele (jeden kraj może mieć wiele miast), z tabelą Suppliers relacją jeden do wiele (w jednym mieście może być kilku dostawców), z tabelą Restaurant relacją jeden do wiele (w jednym mieście może być kilka restauracji), z tabelą Companies relacją jeden do wiele (w jednym mieście może być siedzib różnych firm)

town_ID	Unikalny numer miasta
town_name	Nazwa miasta

```
CREATE TABLE [dbo].[Towns] (  
    [town_ID] [int] NOT NULL,  
    [town_name] [varchar](50) NOT NULL,  
    CONSTRAINT [PK_Towns_1] PRIMARY KEY CLUSTERED  
(  
        [town_ID] ASC  
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,  
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =  
    OFF) ON [PRIMARY]  
) ON [PRIMARY]  
GO
```

5.8.2. Tabela Countries

Countries			
	Column Name	Data Type	Allow Nulls
	country_ID	int	<input type="checkbox"/>
	country_name	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

W tej tabeli przechowywane są informacje o krajach. Połączona ona jest z tabelą Towns_connections relacją jeden do wiele (poprzez town_ID) .

Warunki integralnościowe:

- Nazwy krajów (country_name) powinny być unikalne

country_ID	Unikalny numer państwa
country_name	Nazwa miasta

```
CREATE TABLE [dbo].[Countries](
    [country_ID] [int] NOT NULL,
    [country_name] [varchar](50) NOT NULL,
    CONSTRAINT [PK_Countries_1] PRIMARY KEY CLUSTERED
(
    [country_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [Country_name] UNIQUE NONCLUSTERED
(
    [country_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

5.8.3. Tabela Towns_connections

Towns_connections			
	Column Name	Data Type	Allow Nulls
	town_ID	int	<input type="checkbox"/>
	country_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Tabela, w której przechowywane są informacje, w jakich krajach znajdują się dane miasta. Jest ona połączona z tabelą Countries relacją jeden do wiele (poprzez country_ID), oraz z tabelą Towns relacją jeden do wiele (poprzez town_ID).

country_ID	Numer państwa
town_ID	Numer miasta

```

CREATE TABLE [dbo].[Towns_connections](
    [town_ID] [int] NOT NULL,
    [country_ID] [int] NOT NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Towns_connections] WITH CHECK ADD CONSTRAINT
[FK_TownsConnections_Countries] FOREIGN KEY([country_ID])
REFERENCES [dbo].[Countries] ([country_ID])
GO

ALTER TABLE [dbo].[Towns_connections] CHECK CONSTRAINT
[FK_TownsConnections_Countries]
GO

ALTER TABLE [dbo].[Towns_connections] WITH CHECK ADD CONSTRAINT
[FK_TownsConnections_Towns] FOREIGN KEY([town_ID])
REFERENCES [dbo].[Towns] ([town_ID])
GO

ALTER TABLE [dbo].[Towns_connections] CHECK CONSTRAINT
[FK_TownsConnections_Towns]
GO

```

5.9. Restauracje

5.9.1. Tabela Restaurants


Restaurants			
	Column Name	Condensed Type	Nullable
	restaurant_ID	int	No
	restaurant_name	varchar(20)	No
	town_ID	int	No
	adress	varchar(30)	No
	NIP	varchar(10)	No
	phone_number	varchar(15)	No
	e_mail	varchar(20)	No

Tabela opisująca wszystkie restauracje którymi zarządzamy. Jest ona połączona z tabelą Dishes relacją jeden do wiele (poprzez restaurant_ID), z tabelą Menu relacją jeden do wiele (każda restauracja ma swoje menu), z tabelą Employees relacją jeden do wiele (każda restauracja zatrudnia swoich pracowników i może ich mieć więcej niż jednego), z tabelą Tables relacją jeden do wiele (każda restauracja może mieć więcej niż jeden stół), z Tabelą Discount_disctionary relacją jeden do wiele (każda restauracja ma swoje warunki rabatów i może mieć więcej niż jeden typ), z tabelą Clients relacją jeden do wiele (każda restauracja ma swoich klientów i ilość tych klientów może być większa bądź równa 1), z tabelą Towns relacją jeden do wiele (poprzez town_ID), z tabelą Igrédients relacją jeden do wiele (każda restauracja ma swoje składniki i tych składników może być więcej niż 1)

Warunki integralnościowe:

- Nazwa restauracji (restaurant_name) musi być unikalna
- Numer NIP (NIP) musi być unikalny i składać się tylko z cyfr
- Numer telefonu (Phone_number) musi być unikatowy, oraz składać się z samych cyfr
- Adres email (E_mail) musi być unikatowy i zawierać znak "@"

restaurant_ID	Unikatowy numer identyfikujący pojedynczą restaurację
restaurant_name	Nazwa restauracji
town_ID	Numer odpowiadający miastu w tabeli Towns
Adress	Adres lokalu
NIP	Numer NIP restauracji
Phone_number	Telefon komórkowy do lokalu
E_mail	Adres email do kontaktu z lokalem przez pocztę elektroniczną

```
CREATE TABLE [dbo].[Restaurants](
    [restaurant_ID] [int] NOT NULL,
    [restaurant_name] [varchar](20) NOT NULL,
    [town_ID] [int] NOT NULL,
    [adress] [varchar](30) NOT NULL,
    [NIP] [varchar](10) NOT NULL,
    [phone_number] [varchar](15) NOT NULL,
    [e_mail] [varchar](20) NOT NULL,
    CONSTRAINT [PK_Restaurants] PRIMARY KEY CLUSTERED
(
    [restaurant_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
    CONSTRAINT [IX_Restaurants_email] UNIQUE NONCLUSTERED
(
    [e_mail] 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 [IX_Restaurants_name] UNIQUE NONCLUSTERED
(
    [restaurant_name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY],
```

```

CONSTRAINT [IX_Restaurants_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],
    CONSTRAINT [IX_Restaurants_phone] UNIQUE NONCLUSTERED
(
    [phone_number] 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].[Restaurants] WITH CHECK ADD CONSTRAINT
[FK_Restaurants_Towns] FOREIGN KEY([town_ID])
REFERENCES [dbo].[Towns] ([town_ID])
GO

ALTER TABLE [dbo].[Restaurants] CHECK CONSTRAINT [FK_Restaurants_Towns]
GO

ALTER TABLE [dbo].[Restaurants] WITH CHECK ADD CONSTRAINT
[CK_Restaurants_email] CHECK (([e_mail] like '%@%'))
GO

ALTER TABLE [dbo].[Restaurants] CHECK CONSTRAINT [CK_Restaurants_email]
GO

ALTER TABLE [dbo].[Restaurants] WITH CHECK ADD CONSTRAINT
[CK_Restaurants_NIP] CHECK ((isnumeric([NIP])=(1)))
GO

ALTER TABLE [dbo].[Restaurants] CHECK CONSTRAINT [CK_Restaurants_NIP]
GO

ALTER TABLE [dbo].[Restaurants] WITH CHECK ADD CONSTRAINT
[CK_Restaurants_phone] CHECK ((isnumeric([phone_number])=(1)))
GO

ALTER TABLE [dbo].[Restaurants] CHECK CONSTRAINT [CK_Restaurants_phone]
GO

```

6. Generator danych

W celu przetestowania poprawności działania systemu bazodanowego wygenerowane zostały dane losowe odpowiadające danym rzeczywistym. Do generowania danych wykorzystano SQL Data Generator firmy RedGate.

7. Widoki, funkcje, procedury, triggerzy

7.1. Funkcje użytkownika

7.1.1. Rezerwacje na dzień dzisiejszy(TodaysReservations)

- zwraca tabelę z informacjami na temat rezerwacji z terminem dzisiejszym - do przemyślenia

```
CREATE FUNCTION [dbo].[TodaysReservations]
(
    @restaurant_ID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT reservation_ID AS 'Numer rezerwacji', Reservations.client_ID AS
    'Numer klienta',
        IIF( Reservations.client_ID IN (SELECT client_ID FROM
    Indyvidual_clients), 'Klient Indywidualny', 'Firma') AS 'Typ klienta',
        IIF( Reservations.client_ID IN (SELECT client_ID FROM
    Indyvidual_clients), (SELECT first_name + ' ' + last_name FROM
    Indyvidual_clients WHERE Clients.client_ID = Indyvidual_clients.client_ID),
    (SELECT company_name FROM Companies WHERE Clients.client_ID =
    Companies.client_ID)) AS 'Klient',
        table_ID AS 'Numer stolika', limit_chairs AS 'Liczba miejsc',
    reservation_time AS 'Godzina rezerwacji'
    FROM Reservations
    INNER JOIN Restrictions
    ON Restrictions.restriction_ID = Reservations.restriction_ID
    INNER JOIN Clients
    ON Clients.client_ID = Reservations.client_ID
    WHERE Clients.restaurant_ID = @restaurant_ID AND CAST(GETDATE() AS
    DATE) = reservation_date
)
GO
```

7.1.2. Dostępne stoliki w danym dniu (SpecificDaysFreeTables)

- zwraca tabelę z ID stolika oraz liczbę miejsc obowiązujących. Funkcja przyjmuje dzień i godzinę, kiedy teoretycznie byśmy chcieli dokonać rezerwacji i zwraca wolne opcje w danej firmie. Stoliki zajęte to takie, na które dokonano rezerwacji mniej niż 3h wcześniej lub mniej niż 3h później

```
CREATE FUNCTION [dbo].[SpecificDaysFreeTables]
(
    @restaurant_ID int,
    @date date,
    @time time
)
RETURNS TABLE
AS
RETURN
(
    SELECT Restrictions.table_ID as 'Numer stolika', limit_chairs AS
    'Liczba miejsc'
    FROM restrictions
    INNER JOIN [Tables]
    ON [Tables].table_ID = Restrictions.table_ID
    WHERE restriction_ID not in
    (SELECT restriction_ID FROM Reservations
    WHERE reservation_date = @date AND (DATEDIFF(MINUTE,
reservation_time,@time) BETWEEN -180 AND 180))
    AND (DATEDIFF(DAY,[start_date],@date) >= 0 AND
DATEDIFF(DAY,@date,end_date) >= 0)
    AND restaurant_ID = @restaurant_ID
)
GO
```

7.1.3. Obecne zamówienia (CurrentOrdersToMake)

- Zwraca tabelę z numerem zamówienia, godziną zamówienia, numerem klienta i typem klienta. Informacje dotyczą tylko zamówień na dzisiaj, na późniejszą godzinę niż obecna

```
CREATE FUNCTION [dbo].[CurrentOrdersToMake]
(
    @companyID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT order_ID AS 'Numer zamowienia',pickup_time AS 'Godzina
zamowienia',Orders.client_ID AS 'Numer klienta', IIF(Orders.client_ID IN
(SELECT client_ID FROM Indyvidual_clients),'Indyvidual Client','Company')
AS 'Typ klienta'
    FROM Orders
    INNER JOIN Clients
    ON Orders.client_ID = Clients.client_ID
```



```

WHERE order_date = GETDATE() AND restaurant_ID = @companyID
AND DATEDIFF(minute,GETDATE(),pickup_time) > 0
)
GO

```

7.1.4. Wszystkie zamówienia w danym dniu (SpecificDayOrders)

- Zwraca tabelę z numerem zamówienia, godziną zamówienia, numerem klienta i typem klienta. Informacje dotyczą tylko zamówień na dokładny dzień podany jako parametr

```

CREATE FUNCTION [dbo].[SpecificDayOrders]
(
    @companyID int,
    @specificDay date
)
RETURNS TABLE
AS
RETURN
(
    SELECT order_ID AS 'Numer zamowienia', CAST(pickup_time AS TIME(0)) AS
    'Godzina zamowienia',Orders.client_ID AS 'Numer klienta',
    IIF(Orders.client_ID IN (SELECT client_ID FROM Indyvidual_clients),'Klient
    indywidualny','Firma') AS 'Typ klienta'
    FROM Orders
    INNER JOIN Clients
    ON Orders.client_ID = Clients.client_ID
    WHERE CAST(pickup_time AS DATE) = @specificDay AND restaurant_ID =
    @companyID
)
GO

```

7.1.5. Zamówienie z pozycjami(OrderDetails)

- zwraca tabelę z nazwami dań, liczbą oraz ceną dla wybranego zamówienia

```

CREATE FUNCTION [dbo].[OrdersDetails]
(
    @OrderID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT dish_name AS 'Nazwa dania',
    quantity AS 'Liczebnosc',
    Order_Details.Price AS 'Cena' FROM Order_details
    INNER JOIN Menu
    ON Menu.Menu_ID = Order_details.Menu_ID
    INNER JOIN Dishes
    ON Dishes.dish_ID = Menu.Dish_ID
    WHERE Order_ID = @OrderID
)
GO

```

7.1.6. Aktualni pracownicy (CurrentEmployees)

- Wyświetla wszystkich aktualnie pracujących pracowników. Po podaniu restauracji można się tylko ograniczyć do pracowników wskazanego lokalu.

```
CREATE FUNCTION [dbo].[CurrentEmployees]
(
    @restaurantID int = NULL
)
RETURNS TABLE
AS
RETURN
(
    SELECT e.employee_ID, e.first_name, e.last_name, p.position_name,
    e.part_time, e.rate_per_hour, e.restaurant_ID, e.phone_number, e.email,
    e.hire_date
    FROM Employees AS e
    INNER JOIN Positions p ON p.position_ID = e.position_ID
    WHERE ISNULL(@restaurantID, restaurant_ID) = restaurant_ID AND
    e.hire_date <= GETDATE() AND e.job_quit_date IS NULL
)
GO
```

7.1.7. Menu - obecne pozycje (MenuToday)

- Zwraca tabelę z ID dania oraz z nazwą dania dla pozycji będących dzisiaj w menu dla danej restauracji

```
CREATE FUNCTION [dbo].[MenuToday]
(
    @companyID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT Dishes.dish_ID AS 'Numer dania', dish_name AS 'Nazwa dania' FROM
    Dishes
    INNER JOIN Menu
    ON Dishes.dish_ID = Menu.Dish_ID
    WHERE CAST(GETDATE() AS DATE) >= Add_date AND CAST(GETDATE() AS DATE)
    <= Remove_date
    AND Menu.restaurant_ID = @companyID
)
GO
```

7.1.8. Brakujące składniki do zamówienia (MissingIngredientsForOrder)

- Zwraca tabelę z id składnika, nazwą składnika i w jakiej ilości go brakuje, aby wykonać dane zamówienie

```
CREATE FUNCTION [dbo].[MissingIngredientsForOrder]
(
    @orderID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT i.ingredient_ID as 'ID Składnika', i.ingredient_name as 'Nazwa',
    (dd.Quantity*od.Quantity) - i.ingredient_in_stock as 'Brakująca ilość'
FROM Order_details AS od
    INNER JOIN Menu m ON od.Menu_ID = m.Menu_ID
    INNER JOIN Dishes d ON d.dish_ID = m.Dish_ID
    INNER JOIN Dish_Details dd ON d.dish_ID = dd.dish_ID
    INNER JOIN Ingredients i ON dd.ingredient_ID = i.ingredient_ID
WHERE od.Order_ID = @orderID AND i.ingredient_in_stock -
(dd.Quantity*od.Quantity) < 0
)
GO
```

7.1.9. Jakie rabaty przysługują klientowi (ClientsDiscount)

- zwraca tabelę z numerem rabatu, wartością i datami od kiedy i do kiedy obowiązuje rabat
- zwraca tylko te rabaty, które aktualnie przysługują klientowi w danej restauracji

```
CREATE FUNCTION [dbo].[ClientsDiscount]
(
    @client_ID int,
    @restaurant_ID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT discount_ID AS 'Numer rabatu', Discounts.discount_value AS
    'Wartosc rabatu',
    [start_date] AS 'Od kiedy', end_date AS 'Do kiedy' FROM Discounts
    INNER JOIN Discount_dictionary
    ON Discounts.discount_type_ID = Discount_dictionary.Discount_type_ID
WHERE Restaurant_ID = @restaurant_ID AND client_ID = @client_ID
    AND DATEDIFF(DAY,[start_date],GETDATE()) >= 0 AND
DATEDIFF(DAY,GETDATE(),end_date) >= 0
)
GO
```

7.1.10. Tworzenie faktury - funkcja ogólna (Invoice)

```
CREATE FUNCTION [dbo].[Invoice]
(
    @clientID AS INT,
    @startDate AS DATE,
    @endDate AS DATE,
    @orderID AS INT
)
RETURNS TABLE
AS

RETURN
(
    (SELECT
        CONCAT('Nazwa firmy: ', c.company_name, ' Adres firmy: ', c.address, '
        ', t.town_name, ' ', c.postal_code, ' NIP: ', c.NIP, ' Data utworzenia
        faktury: ', GETDATE()) AS 'Name'
        FROM dbo.Companies AS c
        INNER JOIN dbo.Towns t ON t.town_ID = c.town_ID
        WHERE c.client_ID = @clientID)

    UNION ALL

    SELECT
        CONCAT('Numer zamówienia: ', o.order_ID, ' Data zamówienia:
        ', o.order_date, ' Nazwa zamówionej pozycji: ', d.dish_name, ' Ilość: ',
        od.Quantity, ' Cena detaliczna: ', od.Price, 'zł Rabat: ',
        ISNULL(disc.discount_value,0)*100, '% Suma: ', od.Price*od.Quantity*(1-
        ISNULL(disc.discount_value,0)))
        FROM dbo.Orders AS o
        INNER JOIN dbo.Order_details od ON od.Order_ID = o.order_ID
        INNER JOIN dbo.Menu m ON m.Menu_ID = od.Menu_ID
        INNER JOIN dbo.Dishes d ON d.dish_ID = m.Dish_ID
        INNER JOIN dbo.Clients c ON c.client_ID = o.client_ID
        LEFT OUTER JOIN dbo.Discounts disc ON disc.discount_ID = o.discount_ID
        WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
        @clientID = o.client_ID AND ISNULL(@orderID, o.order_ID) = o.order_ID

    UNION ALL

    SELECT
        CONCAT('Summaryczna kwota: ', SUM(dbo.SumOrder(o.order_ID)), 'zł')
        FROM dbo.Orders AS o
        WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
        @clientID = o.client_ID
    )
)

GO
```

7.1.11. Tworzenie faktury dla jednego zamówienia dla firm (OrderInvoice)

```
CREATE FUNCTION [dbo].[OrderInvoice]
(
    @orderID AS INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Invoice((SELECT client_ID FROM Orders WHERE @orderID
= order_ID), (SELECT order_date FROM Orders WHERE @orderID = order_ID),
(SELECT order_date FROM Orders WHERE @orderID = order_ID), @orderID)
)
GO
```

7.1.12. Tworzenie faktury zbiorczej raz na miesiąc dla firmy (MonthInvoice)

```
CREATE FUNCTION [dbo].[MonthInvoice]
(
    @clientID AS INT,
    @endDate AS DATE
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Invoice(@clientID, DATEADD(DAY, -30, @endDate) ,
@endDate, NULL)
)
GO
```

7.1.13. Niezapłacone zamówienia (z informacją o kliencie) (UnpaidOrders)

- zwraca tabelę z numerem zamówienia, numerem klienta, typem klienta oraz imieniem i nazwiskiem klienta lub nazwą firmy (w zależności od typu klienta)

```
CREATE FUNCTION [dbo].[UnpaidOrders]
(
    @restaurantID AS INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT Orders_Unpaid.order_ID,Orders.client_ID,
    IIF(Orders.client_ID IN (SELECT client_ID FROM
Individual_clients),'Klient Indywidualny','Firma') AS 'Typ klienta',
    IIF(Orders.client_ID IN (SELECT client_ID FROM
Individual_clients),(SELECT first_name + ' ' + last_name FROM
Individual_clients WHERE Orders.client_ID = Individual_clients.client_ID),
```

```

(SELECT company_name FROM Companies WHERE Orders.client_ID =
Companies.client_ID)) AS 'Klient' FROM Orders_Unpaid
INNER JOIN ORDERS
ON Orders_Unpaid.order_ID = Orders.order_ID
INNER JOIN Clients
ON Clients.client_ID = Orders.client_ID
WHERE @restaurantID = restaurant_ID
)
GO

```

7.1.14. Wyświetlenie klientów danej restauracji (AllClients)

- dla danej restauracji wyświetla wszystkich klientów, typ oraz nazwę firmy/imie i nazwisko

```

CREATE FUNCTION [dbo].[AllClients]
(
    @restaurant_ID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT Clients.client_ID AS 'Numer klienta',
    IIF(Clients.client_ID IN (SELECT client_ID FROM
Indyvidual_clients), 'Klient Indywidualny', 'Firma') AS 'Typ klienta',
    IIF(Clients.client_ID IN (SELECT client_ID FROM
Indyvidual_clients), (SELECT first_name + ' ' + last_name FROM
Indyvidual_clients WHERE Clients.client_ID = Indyvidual_clients.client_ID),
(SELECT company_name FROM Companies WHERE Clients.client_ID =
Companies.client_ID)) AS 'Klient' FROM Clients
    WHERE restaurant_ID = @restaurant_ID
)
GO

```

7.1.15. Wyświetlenie pracowników danej firmy (CustomersBelongToCompany)

- zwraca tablicę z imionami i nazwiskami klientów indywidualnych, którzy należą do danej firmy

```

CREATE FUNCTION [dbo].[CustomersBelongToCompany]
(
    @companyID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT ic.client_ID, ic.first_name, ic.last_name FROM Company_Members
AS cm
    INNER JOIN Indyvidual_clients ic ON cm.Indyvidual_Client_ID =
ic.client_ID
    WHERE cm.Company_ID = @companyID
)
GO

```

7.1.16. Wyświetlanie aktualnych typów rabatów dla danej restauracji

(CurrentRestaurantDiscountTypes)

- **Zwraca tabelę z aktualnymi typami rabatów, oraz informacjami o nich dla danej restauracji**

```
CREATE FUNCTION [dbo].[CurrentRestaurantDiscountTypes]
(
    @restaurantID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Discount_dictionary
    WHERE Restaurant_ID = @restaurantID AND From_date <= GETDATE() AND
    GETDATE() <= To_date
)
GO
```

7.1.17. Wyświetlanie wszystkich dań, które nie są zablokowane (AllUnlockedDishes)

- **zwraca tabelę z numerem dania, nazwą dania dla wszystkich dań z danej restauracji, które nie są zablokowane dzisiaj.**

```
CREATE FUNCTION [dbo].[AllUnlockedDishes]
(
    @restaurant_ID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT dish_ID, dish_name FROM Dishes
    WHERE (DATEDIFF(DAY,GETDATE(), locked_date) > 0 OR
    DATEDIFF(DAY,unlocked_date, GETDATE()) >= 0)
    AND restaurant_ID = @restaurant_ID
)
GO
```

7.1.18. Wyświetlenie składników danego dania (DishIngredients)

- wyświetla podstawowe informacje o składnikach dla danego dania

```
CREATE FUNCTION [dbo].[DishIngredients]
(
    @dishID int,
    @restaurantID int = NULL
)
RETURNS TABLE
AS
RETURN
(
    SELECT DISTINCT i.ingredient_ID, i.ingredient_name, dd.Quantity as
'Ilość do dania', i.ingredient_quantity, i.extra_information
FROM Dish_Details AS dd
INNER JOIN Ingredients i ON dd.ingredient_ID = i.ingredient_ID
WHERE dd.dish_ID = @dishID AND ISNULL(@restaurantID, i.restaurant_ID) =
i.restaurant_ID
)
GO
```

7.1.19. Wyświetlenie dań, które można dodać do menu (były co najmniej miesiąc wcześniej usunięte) (dbo.DishesThatCanBeAddedToMenu)

- Wyświetla numer, nazwę dania oraz jego ostatnią datę usunięcia z menu, jeżeli interwał czasu jaki minął od tej daty jest większy niż miesiąc

```
CREATE FUNCTION [dbo].[DishesThatCanBeAddedToMenu]
(
    @restaurant_ID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT Menu.dish_ID AS 'Numer dania', dish_name AS 'Nazwa dania',
MAX(Remove_date) AS 'Ostatnia data usuniecia z menu'
FROM Menu
INNER JOIN Dishes
ON Menu.Dish_ID = Dishes.dish_ID
WHERE Menu.restaurant_ID = @restaurant_ID AND (GETDATE() < locked_date
OR GETDATE() >= unlocked_date)
GROUP BY Menu.Dish_ID, dish_name
HAVING DATEDIFF(DAY, MAX(Remove_date), GETDATE()) > 30
)
GO
```


7.1.20. Wyświetlanie wszystkich zamówień danego klienta (GetClientOrderHistory)

- wyświetla całą historię danego klienta, z uwzględnieniem numeru zamówienia, daty, godziny, nazwy dania które zamówił, w jakiej ilości, cenie, niższe

```
CREATE FUNCTION [dbo].[GetClientOrderHistory]
(
    @clientID int
)
RETURNS TABLE
AS
RETURN
(
    SELECT o.order_ID, o.order_date, o.pickup_time, d.dish_name, od.price,
    od.Quantity, disc.discount_value FROM Orders AS o
    INNER JOIN Order_details od ON o.order_ID = od.Order_ID
    INNER JOIN Menu m ON m.Menu_ID = od.Menu_ID
    INNER JOIN Dishes d ON d.dish_ID = m.Dish_ID
    INNER JOIN dbo.Clients c ON c.client_ID = o.client_ID
    LEFT OUTER JOIN dbo.Discounts disc ON disc.discount_ID = o.discount_ID
    WHERE o.client_ID = @clientID
)
GO
```

7.2. Funkcje do generowania raportów

7.2.1. Obliczanie średniego rabatu (AverageDiscount)

- funkcja oblicz średnią wartość aktualnych rabatów klientów danej restauracji

```
CREATE FUNCTION [dbo].[AverageDiscount]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS FLOAT
AS
BEGIN
    RETURN ISNULL((SELECT AVG(d.discount_value) FROM dbo.Discounts AS d
    INNER JOIN dbo.Discount_dictionary dd ON d.discount_type_ID =
    dd.Discount_type_ID
    WHERE @restaurantID = dd.Restaurant_ID AND @startDate <= d.start_date
    AND d.end_date <= @endDate),0)
END
GO
```

7.2.2. Średnia ilość zamówień (AverageOrders)

- średnia liczba zamówień jaką składali klienci w danym czasie w danej restauracji

```
CREATE FUNCTION [dbo].[AverageOrders]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @differenceDate AS INT;
    SET @differenceDate =DATEDIFF(DAY, @startDate, @endDate)+1;

    RETURN CAST((SELECT COUNT(*) FROM dbo.Orders AS o
        INNER JOIN dbo.Employees e ON e.employee_ID = o.employee_ID
        INNER JOIN dbo.Restaurants r ON r.restaurant_ID = e.restaurant_ID
        WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
@restaurantID = e.restaurant_ID) AS FLOAT )/@differenceDate
END
GO
```

7.2.3. Średnia kwota zamówień na dany dzień (AverageOrderCost)

- Funkcja zwraca średnią wartość zamówień na dany dzień jakie złożyli klienci danej restauracji

```
CREATE FUNCTION [dbo].[AverageOrderCost]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT AVG(dbo.SumOrder(o.order_ID)) FROM dbo.Orders AS o
        INNER JOIN dbo.Employees e ON e.employee_ID = o.employee_ID
        INNER JOIN dbo.Restaurants r ON r.restaurant_ID = e.restaurant_ID
        WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
@restaurantID = e.restaurant_ID);
END
GO
```

7.2.4. Dzień z największą liczbą zamówień (DayWithMaxOrderCount)

- funkcja zwraca dzień z maksymalną liczbą zamówień (dla wybranego przedziału czasowego)

```
CREATE FUNCTION [dbo].[DayWithMaxOrderCount]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS DATE
AS
BEGIN
    DECLARE @iterDate AS DATE;
    DECLARE @maxCount AS INT
    DECLARE @maxDayCount AS DATE;
    DECLARE @tmpCount AS INT;

    SET @iterDate = @startDate;
    SET @maxCount = 0;
    SET @maxDayCount = @startDate;

    WHILE @iterDate <= @endDate
    BEGIN
        SET @tmpCount = dbo.AverageOrders(@iterDate, @iterDate,
@restaurantID);

        IF @tmpCount > @maxCount
        BEGIN
            SET @maxCount = @tmpCount;
            SET @maxDayCount = @iterDate;
        END

        SET @iterDate = DATEADD(DAY, 1, @iterDate);
    END

    RETURN @maxDayCount;
END
GO
```

7.2.5. Ilość rezerwacji w danej restauracji w danym dniu(CountReservationsInOneDay)

- **Zwraca liczbę rezerwacji w danej restauracji w danym dniu**

```
CREATE FUNCTION [dbo].[CountReservationInOneDay]
(
    @reservationDate AS DATE,
    @restaurantID AS INT
)
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM dbo.Reservations AS r
        INNER JOIN dbo.Restrictions res ON res.restriction_ID =
r.restriction_ID
        INNER JOIN dbo.Tables t ON t.table_ID = res.table_ID
        WHERE @reservationDate = r.reservation_date AND @restaurantID =
t.restaurant_ID)
END
GO
```

7.2.6. Liczba pozycji w menu (AverageMenuPositions)

- **wyświetla średnią liczbę pozycji w menu w danym czasie**

```
CREATE FUNCTION [dbo].[AverageMenuPositions]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @daysCount AS INT;
    DECLARE @sumOfCountMenu AS INT;
    DECLARE @dayIter AS DATE;
    SET @daysCount = DATEDIFF(DAY, @startDate, @endDate) + 1;
    SET @sumOfCountMenu = 0;
    SET @dayIter = @startDate;

    WHILE @dayIter <= @endDate
    BEGIN
        SET @sumOfCountMenu = @sumOfCountMenu + (SELECT COUNT(*) FROM
dbo.Menu WHERE @restaurantID = restaurant_ID AND Add_date <= @dayIter AND
@dayIter <= Remove_date);
        SET @dayIter = DATEADD(DAY, 1, @dayIter);
    END

    RETURN CAST(@sumOfCountMenu AS FLOAT)/@daysCount
END
GO
```

7.2.7. Średnia ilość rezerwacji na dzień (AverageReservations)

- Funkcja obliczająca średnią ilość rezerwacji na dzień w danym przedziale czasowym

```
CREATE FUNCTION [dbo].[AverageReservations]
(
    @startDate AS DATE,
    @endDate AS DATE,
    @restaurantID AS INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @days AS INT;
    SET @days = DATEDIFF(DAY, @startDate, @endDate) + 1

    RETURN (SELECT CAST(COUNT(*) AS FLOAT)/@days FROM dbo.Reservations AS r
        INNER JOIN dbo.Restrictions res ON res.restriction_ID =
r.restriction_ID
        INNER JOIN dbo.Tables t ON t.table_ID = res.table_ID
        WHERE r.reservation_date <= @startDate AND r.reservation_date <=
@endDate AND @restaurantID = t.restaurant_ID)

END
GO
```

7.2.8. Raport (Report)

- funkcja ogólna do tworzenia raportów tygodniowych i miesięcznych, jako parametry przyjmuje przedział czasowy oraz numer identyfikujący restaurację, a następnie generuje raport składający się z informacji ogólnych dotyczących danego przedziału.

```
CREATE FUNCTION [dbo].[Report]
(
    @restaurantID AS INT,
    @startDate AS DATE,
    @endDate AS DATE
)
RETURNS TABLE
AS
RETURN (
    SELECT dbo.AverageDiscount(@startDate, @endDate, @restaurantID) AS
'Sredni rabat',
    dbo.AverageOrderCost(@startDate, @endDate, @restaurantID) AS 'Średnia
wartość zamówień',
    dbo.AverageOrders(@startDate, @endDate, @restaurantID) AS 'Średnia
liczba zamówień na dzień',
    dbo.DayWithMaxOrderCount(@startDate, @endDate, @restaurantID) AS 'Dzień
z największą liczbą zamówień',
    dbo.AverageReservations(@startDate, @endDate, @restaurantID) AS
'Srednia liczba rezerwacji',
    dbo.AverageMenuPositions(@startDate, @endDate, @restaurantID) AS
'Srednia liczba pozycji w menu'
)
GO
```

7.2.9. Raport tygodniowy (WeeklyReport)

```
CREATE FUNCTION [dbo].[WeeklyReport]
(
    @startDate AS DATE,
    @restaurantID AS INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Report(@restaurantID, @startDate, DATEADD(DAY, 7,
@startDate))
)
GO
```

7.2.10. Raport miesięczny (MonthlyReport)

```
CREATE FUNCTION [dbo].[MonthlyReport]
(
    @startDate AS DATE,
    @restaurantID AS INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Report(@restaurantID, @startDate, DATEADD(DAY, 30,
@startDate))
)
GO
```

7.2.11. Funkcja do generowania faktury (Invoice)

- funkcja pozwala na generowanie faktur na podstawie parametrów, wykorzystywana w funkcjach dotyczących generowania faktur

```
CREATE FUNCTION [dbo].[Invoice]
(
    @clientID AS INT,
    @startDate AS DATE,
    @endDate AS DATE,
    @orderID AS INT
)
RETURNS TABLE
AS

RETURN
(
    (SELECT
```

```

        CONCAT('Nazwa firmy: ', c.company_name, ' Adres firmy: ', c.address, '
', t.town_name, ' ', c.postal_code, ' NIP: ', c.NIP, ' Data utworzenia
faktury: ', GETDATE()) AS 'Name'
    FROM dbo.Companies AS c
    INNER JOIN dbo.Towns t ON t.town_ID = c.town_ID)

    UNION ALL
    SELECT
        CONCAT('Numer zamówienia: ', o.order_ID, ' Data zamówienia:
', o.order_date, ' Nazwa zamówionej pozycji: ', d.dish_name, ' Ilość: ',
od.Quantity, ' Cena detaliczna: ', od.Price, 'zł Rabat: ',
disc.discount_value*100, '%' Suma: ', od.Price*od.Quantity*(1-
disc.discount_value))
    FROM dbo.Orders AS o
    INNER JOIN dbo.Order_details od ON od.Order_ID = o.order_ID
    INNER JOIN dbo.Menu m ON m.Menu_ID = od.Menu_ID
    INNER JOIN dbo.Dishes d ON d.dish_ID = m.Dish_ID
    INNER JOIN dbo.Clients c ON c.client_ID = o.client_ID
    INNER JOIN dbo.Discounts disc ON disc.client_ID = c.client_ID
    WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
@clientID = o.client_ID AND ISNULL(@orderID, o.order_ID) = o.order_ID

    UNION ALL
    SELECT
        CONCAT('Sumaryczna kwota: ', SUM(dbo.SumOrder(o.order_ID)), 'zł')
    FROM dbo.Orders AS o
    WHERE @startDate <= o.order_date AND o.order_date <= @endDate AND
@clientID = o.client_ID
)

GO

```

7.2.12. Faktura dla firm dla pojedynczego zamówienia (OrderInvoice)

```

CREATE FUNCTION [dbo].[OrderInvoice]
(
    @clientID AS INT,
    @orderID AS INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Invoice(@clientID, (SELECT order_date FROM Orders
WHERE @orderID = order_ID), (SELECT order_date FROM Orders WHERE @orderID =
order_ID), @orderID)
)

GO

```

7.2.13. Faktura zbiorcza miesięczna (MonthInvoice)

```
CREATE FUNCTION [dbo].[MonthInvoice]
(
    @clientID AS INT,
    @endDate AS DATE
)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM dbo.Invoice(@clientID, DATEADD(DAY, -30, @endDate) ,
@endDate, NULL)
)
GO
```

7.3. Funkcje systemowe

7.3.1. Funkcja obliczająca wartość zamówienia (SumOrder)

- **funkcja oblicza całkowitą wartość zamówienia**

```
CREATE FUNCTION [dbo].[SumOrder]
(
    @orderIndex int
)
RETURNS MONEY
AS
BEGIN
    RETURN (
        SELECT ROUND(SUM(od.Price*od.Quantity*(1-ISNULL(d.discount_value,
0))),2) FROM Order_details AS od
        INNER JOIN Orders o ON od.Order_ID = o.order_ID
        INNER JOIN dbo.Clients c ON c.client_ID = o.client_ID
        LEFT OUTER JOIN dbo.Discounts d ON d.discount_ID = o.discount_ID
        WHERE @orderIndex = od.Order_ID
    )
END
GO
```

7.3.2. Funkcja sprawdzająca, czy można zamówić owoce morza (czwartek-piątek-sobota) (CanOrderSeafood)

- **Zamówienie możemy złożyć maksymalnie w poniedziałek poprzedzający dzień odebrania zamówienia. Różnica między datą zamówienia, a datą odebrania musi być większa niż liczba porządkowa tygodnia od 0**

```
CREATE FUNCTION [dbo].[CanOrderSeafood]
(
    @pickUpDate AS DATETIME,
    @orderDate AS DATE
```



```

)
RETURNS BIT
AS
BEGIN
    IF DATEPART(WEEKDAY, @pickUpDate) >= 5 AND DATEPART(WEEKDAY,
@pickUpDate) <= 7 AND DATEDIFF(DAY, @orderDate, @pickUpDate) >=
DATEPART(WEEKDAY, @pickUpDate) -2
    BEGIN
        RETURN 1;
    END

    RETURN 0;

END
GO

```

7.3.3. Sprawdzenie, czy klient indywidualny może dokonać rezerwacji
(CanIndyvidualMakeReservation)

- sprawdza, czy klient indywidualny może złożyć rezerwację

```

CREATE FUNCTION [dbo].[CanIndyvidualMakeReservation]
(
    @clientID AS INT,
    @clientsOrderID AS INT
)
RETURNS BIT
AS
BEGIN
    IF dbo.SumOrder(@clientsOrderID) >= 50 AND (
        SELECT COUNT(order_ID) FROM dbo.Orders
        WHERE dbo.SumOrder(order_ID) >= 200) >= 5 AND @clientID IN (SELECT
client_ID FROM dbo.Indyvidual_clients)

    BEGIN
        RETURN 1;
    END

    RETURN 0;

END
GO

```

7.3.4. Sprawdzenie warunków na przydzielenie rabatu dla klienta indywidualnego
(CheckIndyvidualDiscount)

- na podstawie parametrów wysłanych do funkcji sprawdzamy czy klientowi należy przyznać rabat

```

CREATE FUNCTION [dbo].[CheckIndyvidualDiscount]
(
    @clientID AS INT,
    @countOfOrders AS INT,
    @valueOfOrder AS MONEY,
    @allOrderValue AS MONEY,

```

```

        @lastDiscountDate AS DATE
    )
    RETURNS BIT
    AS
    BEGIN
        IF @clientID NOT IN (SELECT client_ID FROM dbo.Individual_clients)
        BEGIN
            RETURN 0;
        END
        IF (SELECT COUNT(*) FROM dbo.Orders WHERE @clientID = client_ID AND
        dbo.SumOrder(order_ID) > @valueOfOrder AND order_date > @lastDiscountDate)
        > @countOfOrders
        AND
        (SELECT SUM(dbo.SumOrder(order_ID)) FROM dbo.Orders WHERE @clientID =
        client_ID AND order_date > @lastDiscountDate) > @allOrderValue
        BEGIN
            RETURN 1;
        END
    END
    RETURN 0;
END
GO

```

7.3.5. Znalezienie wartości nowego rabatu dla klienta indywidualnego (GetFirstOrSecondDiscountValue)

- sprawdza warunki czy klient indywidualny ma dostać rabat, jeżeli tak, zwraca wartość rabatu (uwzględniając rabaty, które już klient posiada), jeżeli nie - zwraca 0

```

CREATE FUNCTION [dbo].[GetFirstOrSecondDiscountValue]
(
    @clientID AS INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @countOfOrders AS INT;
    DECLARE @valueOfOrder AS MONEY;
    DECLARE @lastDiscountDate AS DATE;
    DECLARE @lastDiscountValue AS FLOAT;

    SET @countOfOrders = (SELECT Count_Of_Orders FROM
    dbo.Discount_dictionary WHERE From_date <= GETDATE() AND GETDATE() <=
    To_date AND Count_Of_Orders IS NOT NULL);
    SET @valueOfOrder = (SELECT Value_Orders FROM dbo.Discount_dictionary
    WHERE From_date <= GETDATE() AND GETDATE() <= To_date AND Count_Of_Orders
    IS NOT NULL);
    SET @lastDiscountDate = (SELECT MAX(start_date) FROM dbo.Discounts AS d
    INNER JOIN dbo.Discount_dictionary dd ON dd.Discount_type_ID =
    d.discount_type_ID
    WHERE @clientID = client_ID AND GETDATE() <= d.end_date AND
    dd.Count_Of_Orders = @countOfOrders AND dd.Value_Orders = @valueOfOrder);

```

```

        SET @lastDiscountValue = (SELECT d.discount_value FROM dbo.Discounts AS
d
        INNER JOIN dbo.Discount_dictionary dd ON dd.Discount_type_ID =
d.discount_type_ID
        WHERE @clientID = client_ID AND GETDATE() <= end_date AND
start_date = @lastDiscountDate AND dd.Count_Of_Orders = @countOfOrders AND
dd.Value_Orders = @valueOfOrder);

        IF @lastDiscountDate IS NULL AND dbo.CheckIndyvidualDiscount(@clientID,
@countOfOrders, @valueOfOrder, -1, DATEADD(YEAR, -100, GETDATE())) = 1
        BEGIN
            RETURN (SELECT Discount_Value FROM dbo.Discount_dictionary WHERE
From_date <= GETDATE() AND GETDATE() <= To_date AND Count_Of_Orders IS NOT
NULL)
        END

        IF @lastDiscountDate IS NOT NULL AND
dbo.CheckIndyvidualDiscount(@clientID, @countOfOrders, @valueOfOrder, -1,
@lastDiscountDate) = 1
        BEGIN
            RETURN (SELECT Discount_Value FROM dbo.Discount_dictionary WHERE
From_date <= GETDATE() AND GETDATE() <= To_date AND Count_Of_Orders IS NOT
NULL) + @lastDiscountValue;
        END

        RETURN 0;
    END
GO

```

7.3.6. Sprawdzenie warunków na 3 i 4 typ rabatu dla klienta indywidualnego (GetThirdAndFourthDiscountID)

- funkcja sprawdza warunki rabatu oraz jeżeli można klientowi indywidualnemu przyznać rabat zwraca id typu rabatu, w przeciwnym razie zwraca -1

```

CREATE FUNCTION [dbo].[GetThirdAndFourthDiscountID]
(
    @clientID AS INT
)
RETURNS INT
AS
BEGIN
    DECLARE @discounTypeID AS INT;
    DECLARE @sumOfOrders AS INT;

    SET @sumOfOrders = (SELECT SUM(dbo.SumOrder(order_ID)) FROM dbo.Orders
WHERE @clientID = client_ID);
    SET @discounTypeID = (SELECT MAX(Value_Orders) FROM
dbo.Discount_dictionary WHERE Count_Of_Days IS NULL AND Value_Orders <=
@sumOfOrders );

```

```

    IF @discountTypeID NOT IN (SELECT discount_type_ID FROM dbo.Discounts
WHERE @clientID = client_ID)
    BEGIN
        RETURN @discountTypeID;
    END

    RETURN -1;

END
GO

```

7.3.7. Sprawdzenie warunków na 1. rabat dla firmy (GetCompanyFirstDiscountValue)

- funkcja sprawdza czy firmie można przydzielić rabat, jeśli tak to zwraca jego wartość, jeżeli nie to zwraca 0

```

CREATE FUNCTION [dbo].[GetCompanyFirstDiscountValue]
(
    @clientID AS INT,
    @startDate AS DATE
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @endDate AS DATE;
    DECLARE @discountTypeID AS INT;
    DECLARE @valueOfLastDiscount AS FLOAT;
    DECLARE @maxDiscountTypeValue AS FLOAT;
    DECLARE @valueOfDiscount AS FLOAT;

    SET @endDate = DATEADD(MONTH, -1, @startDate);
    SET @discountTypeID = (SELECT Discount_type_ID FROM
dbo.Discount_dictionary WHERE Max_Discount_Value IS NOT NULL AND
One_Time_Discount = 0 AND Count_Of_Days IS NULL AND @startDate > From_date
AND @startDate < To_date );
    SET @valueOfLastDiscount = (SELECT Discount_Value FROM dbo.Discounts
WHERE Discount_type_ID = @discountTypeID AND @endDate = [start_date] AND
@clientID = client_ID);
    SET @valueOfLastDiscount = ISNULL(@valueOfLastDiscount, 0);
    SET @maxDiscountTypeValue = (SELECT Max_Discount_Value FROM
dbo.Discount_dictionary WHERE Discount_type_ID = @discountTypeID);
    SET @valueOfDiscount = (SELECT Discount_Value FROM
dbo.Discount_dictionary WHERE Discount_type_ID = @discountTypeID);

    IF dbo.CheckIfCompanyShoudGetFirstTypeOfDiscount(@clientID, (SELECT
Count_Of_Orders FROM dbo.Discount_dictionary WHERE Discount_type_ID =
@discountTypeID), (SELECT Value_Orders FROM dbo.Discount_dictionary WHERE
Discount_type_ID = @discountTypeID), @startDate) = 0
    BEGIN
        RETURN 0;
    END
END

```

```

    IF @maxDiscountTypeValue < @valueOfLastDiscount + @valueOfDiscount
    BEGIN
        RETURN @maxDiscountTypeValue;
    END

    RETURN @valueOfLastDiscount + @valueOfDiscount;

END
GO

```

7.3.8. Sprawdzenie warunków na 2. rabat dla firmy (GetCompanySecondDiscountID)

- jeżeli firmie można przydzielić dany rabat zwracamy id rabatu, w przeciwnym razie -1

```

CREATE FUNCTION [dbo].[GetCompanySecondDiscountID]
(
    @clientID AS INT,
    @startDate AS DATE
)
RETURNS INT
AS
BEGIN
    DECLARE @endDate AS DATE;
    DECLARE @discountTypeID AS INT;
    DECLARE @orderCostFromDiscount AS FLOAT;

    SET @endDate = DATEADD(MONTH, -3, @startDate);
    SET @discountTypeID = (SELECT Discount_type_ID FROM
dbo.Discount_dictionary WHERE Max_Discount_Value IS NOT NULL AND
One_Time_Discount = 0 AND Count_Of_Days IS NULL AND Count_Of_Orders IS NULL
AND @startDate > From_date AND @startDate < To_date );
    SET @orderCostFromDiscount = (SELECT Value_Orders FROM
dbo.Discount_dictionary WHERE @discountTypeID = Discount_type_ID);

    IF (SELECT SUM(dbo.SumOrder(order_ID)) FROM dbo.Orders WHERE @clientID
= client_ID AND @startDate <= order_date AND order_date <= @endDate) >=
@orderCostFromDiscount
    BEGIN
        RETURN @discountTypeID;
    END

    RETURN -1;

END
GO

```

7.4. Procedury

7.4.1. Dodanie nowego klienta indywidualnego (AddIndyvidualClient)

- Dodaje klienta indywidualnego do tabeli "Indyvidual_clients" i "Clients"

```
CREATE PROCEDURE [dbo].[AddIndyvidualClient]
    @phoneNumber AS VARCHAR(10),
    @email AS VARCHAR(50),
    @firstName AS VARCHAR(30),
    @lastName AS VARCHAR(30),
    @restaurantID AS INT
AS

DECLARE @newClientID AS INT;
DECLARE @error AS VARCHAR(20);

SET @newClientID = (SELECT MAX(client_ID)+1 FROM dbo.Clients)

IF @restaurantID IS NULL
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Clients
(
    client_ID,
    phone_number,
    e_mail,
    restaurant_ID
)
VALUES
(
    @newClientID, -- client_ID - int
    @phoneNumber, -- phone_number - varchar(10)
    @email, -- e_mail - varchar(50)
    @restaurantID -- restaurant_ID - int
)

INSERT INTO dbo.Indyvidual_clients
(
    client_ID,
    first_name,
    last_name
)
VALUES
(
    @newClientID, -- client_ID - int
    @firstName, -- first_name - varchar(30)
    @lastName -- last_name - varchar(30)
)

GO
```

7.4.2. Dodanie nowego klienta jako firmy (AddCompanyClient)

- Dodaje klienta indywidualnego do tabeli "Indywidual_clients" i "Clients"

```
CREATE PROCEDURE [dbo].[AddCompanyClient]
    @companyName AS VARCHAR(20),
    @NIP AS VARCHAR(10),
    @address AS VARCHAR(50),
    @townName AS VARCHAR(50),
    @countryName AS VARCHAR(50),
    @postalCode AS VARCHAR(10),
    @phoneNumber AS VARCHAR(10),
    @email AS VARCHAR(50),
    @restaurantID AS INT
AS

DECLARE @newClientID AS INT;
SET @newClientID = (SELECT MAX(client_ID)+1 FROM dbo.Clients);

IF @townName NOT IN (
    SELECT town_name FROM dbo.Towns AS t
    INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
    INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
    WHERE t.town_name = @townName AND c.country_name = @countryName
)
BEGIN
    EXEC dbo.AddTown @townName = @townName, @countryName = @countryName;
END

INSERT INTO dbo.Clients
(
    client_ID,
    phone_number,
    e_mail,
    restaurant_ID
)
VALUES
(
    @newClientID, -- client_ID - int
    @phoneNumber, -- phone_number - varchar(10)
    @email, -- e_mail - varchar(50)
    @restaurantID -- restaurant_ID - int
)

INSERT INTO dbo.Companies
(
    client_ID,
    company_name,
    NIP,
    address,
    town_ID,
    postal_code
)
VALUES
(
    @newClientID, -- client_ID - int
    @companyName, -- company_name - varchar(20)
    @NIP, -- NIP - varchar(10)
    @address, -- address - varchar(50)
    (SELECT tc.town_ID FROM dbo.Towns AS t
```

```

INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
WHERE t.town_name = @townName AND c.country_name = @countryName
), -- town_ID - int
@postalCode -- postal_code - varchar(10)
)

```

GO

7.4.3. Dodanie restauracji do bazy (AddRestaurant)

- Dodaje restaurację do tabeli "Restaurants"

```

CREATE PROCEDURE [dbo].[AddRestaurant]
    @restaurantName AS VARCHAR(20),
    @townName AS VARCHAR(50),
    @countryName AS VARCHAR(50),
    @address AS VARCHAR(30),
    @NIP AS VARCHAR(10),
    @phoneNumber AS VARCHAR(9),
    @email AS VARCHAR(20)
AS

DECLARE @newRestaurantID AS INT
SET @newRestaurantID = (SELECT MAX(restaurant_ID)+1 FROM dbo.Restaurants)

IF @townName NOT IN (
    SELECT town_name FROM dbo.Towns AS t
    INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
    INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
    WHERE t.town_name = @townName AND c.country_name = @countryName
)
BEGIN
    EXEC dbo.AddTown @townName = @townName, @countryName = @countryName;
END

INSERT INTO dbo.Restaurants
(
    restaurant_ID,
    restaurant_name,
    town_ID,
    address,
    NIP,
    phone_number,
    e_mail
)
VALUES
(
    @newRestaurantID, -- restaurant_ID - int
    @restaurantName, -- restaurant_name - varchar(20)
    (SELECT tc.town_ID FROM dbo.Towns AS t
    INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
    INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
    WHERE t.town_name = @townName AND c.country_name = @countryName
    ), -- town_ID - int
    @address, -- address - varchar(30)

```



```

@NIP, -- NIP - varchar(10)
@phoneNumber, -- phone_number - varchar(15)
@email -- e_mail - varchar(20)
)

```

GO

- 7.4.4. Dodanie pracownika restauracji (AddEmployee)
- Dodaje nowego pracownika do tabeli "Employees"

```

CREATE PROCEDURE [dbo].[AddEmployee]
    @positionName AS VARCHAR(20),
    @firstName AS VARCHAR(40),
    @lastName AS VARCHAR(40),
    @partTime AS VARCHAR(15),
    @hireDate AS DATE,
    @rate_per_hour AS MONEY,
    @phoneNumber AS VARCHAR(12),
    @email AS VARCHAR(50),
    @photoPath AS VARCHAR(50),
    @restaurantID AS INT
AS
DECLARE @positionID AS INT;
DECLARE @newEmployeeID AS INT;
DECLARE @error AS VARCHAR(20);

IF @positionName NOT IN (SELECT position_name FROM dbo.Positions)
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

SET @positionID = (SELECT position_ID FROM dbo.Positions WHERE
position_name = @positionName);
SET @newEmployeeID = (SELECT MAX(employee_ID)+1 FROM dbo.Employees);

INSERT INTO dbo.Employees
(
    employee_ID,
    position_ID,
    first_name,
    last_name,
    part_time,
    rate_per_hour,
    phone_number,
    email,
    hire_date,
    photo_path,
    job_quit_date,
    restaurant_ID
)
VALUES

```

```
(
    @newEmployeeID,          -- employee_ID - int
    @positionID,             -- position_ID - int
    @firstName,              -- first_name - varchar(40)
    @lastName,               -- last_name - varchar(40)
    @partTime,               -- part_time - varchar(15)
    @rate_per_hour,          -- rate_per_hour - money
    @phoneNumber,            -- phone_number - varchar(12)
    @email,                  -- email - varchar(50)
    ISNULL(@hireDate, GETDATE()), -- hire_date - date
    @photoPath,              -- photo_path - varchar(50)
    NULL,                    -- job_quit_date - date
    @restaurantID            -- restaurant_ID - int
)
```

GO

7.4.5. Tworzenie rezerwacji dla klienta indywidualnego

(AddIndyvidualReservation)

- dodaje rezerwacje klienta indywidualnego

```
CREATE PROCEDURE [dbo].[AddIndyvidualReservation]
    @clientID AS INT,
    @orderID AS INT,
    @reservationDate AS DATE,
    @reservationTime AS TIME(7),
    @tableID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @restaurantID AS INT;
DECLARE @restrictionID AS INT;
DECLARE @reservationID AS INT;

SET @restaurantID = (SELECT restaurant_ID FROM dbo.Tables WHERE @tableID =
table_ID);
SET @restrictionID = (SELECT restriction_ID FROM dbo.Restrictions WHERE
start_date <= @reservationDate AND end_date <= @reservationDate AND
@tableID = table_ID)
SET @reservationID = (SELECT MAX(reservation_ID)+1 FROM dbo.Reservations)

IF (dbo.CanIndyvidualMakeReservation(@clientID,@orderID)) = 0 OR
@reservationDate >= GETDATE() OR @tableID NOT IN (SELECT [Numer stolika]
FROM dbo.SpecificDaysFreeTables(@restaurantID, @reservationDate,
@reservationTime))
BEGIN
    SET @error = 'Nie mozesz zlozyc rezerwacji!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Reservations
(
    reservation_ID,
    client_ID,
    reservation_date,
```

```

        reservation_time,
        restriction_ID
    )
VALUES
(
    @reservationID,          -- reservation_ID - int
    @clientID,               -- client_ID - int
    @reservationDate,        -- reservation_date - date
    @reservationTime,        -- reservation_time - time(7)
    @restrictionID           -- restriction_ID - int
)

INSERT INTO dbo.Personal_Reservations
(
    reservation_ID,
    order_ID
)
VALUES
(
    @reservationID, -- reservation_ID - int
    @orderID -- order_ID - int
)

GO

```

7.4.6. Tworzenie rezerwacji dla firmy (AddCompanyReservation)

- Dodanie rezerwacji na firmę z możliwością podania na kogo jest ta rezerwacja

```

CREATE PROCEDURE [dbo].[AddCompanyReservation]
    @clientID AS INT,
    @reservationDate AS DATE,
    @reservationTime AS TIME(7),
    @tableID AS INT,
    @nameID AS INT = NULL
AS
DECLARE @error AS VARCHAR(20);
DECLARE @restaurantID AS INT;
DECLARE @restrictionID AS INT;
DECLARE @reservationID AS INT;

SET @restaurantID = (SELECT restaurant_ID FROM dbo.Tables WHERE @tableID =
table_ID);
SET @restrictionID = (SELECT restriction_ID FROM dbo.Restrictions WHERE
[start_date] <= @reservationDate AND @reservationDate <= [end_date] AND
@tableID = table_ID)
SET @reservationID = (SELECT MAX(reservation_ID)+1 FROM dbo.Reservations)

IF
@clientID NOT IN (SELECT client_ID FROM dbo.Companies) OR @reservationDate
< CAST(GETDATE() AS DATE) OR @tableID NOT IN (SELECT [Numer stolika] FROM
dbo.SpecificDaysFreeTables(@restaurantID, @reservationDate,
@reservationTime))
BEGIN
    SET @error = 'Nie możesz złożyć rezerwacji!';
    RAISERROR(@error, 16, 1);
    RETURN;

```

```

END

INSERT INTO dbo.Reservations
(
    reservation_ID,
    client_ID,
    reservation_date,
    reservation_time,
    restriction_ID
)
VALUES
(
    @reservationID,          -- reservation_ID - int
    @clientID,               -- client_ID - int
    @reservationDate,        -- reservation_date - date
    @reservationTime,        -- reservation_time - time(7)
    @restrictionID           -- restriction_ID - int
)

INSERT INTO dbo.Company_Reservations
(
    resevervation_ID
)
VALUES
(@reservationID -- resevervation_ID - int
)

IF @nameID IS NOT NULL
BEGIN
    EXEC AddNameReservation @reservationID = @reservationID, @nameID =
@nameID
END
GO

```

- 7.4.7. Dodanie imiennej rezerwacji do rezerwacji firmowej
(AddNameReservation)
- łączy rezerwację firmową z klientem indywidualnym

```

CREATE PROCEDURE [dbo].[AddNameReservation]
    @reservationID AS INT,
    @nameID AS INT
AS

DECLARE @companyID AS INT;
SET @companyID = (SELECT client_ID FROM dbo.Reservations WHERE
@reservationID = reservation_ID)

DECLARE @error AS VARCHAR(20);

IF @nameID NOT IN (SELECT Indyvidual_Client_ID FROM dbo.Company_Members
WHERE @companyID = Company_ID)
BEGIN
    SET @error = 'Nie możesz złożyć rezerwacji!';
    RAISERROR(@error, 16, 1);
END

```

```

IF @nameID IS NOT NULL
BEGIN
    INSERT INTO dbo.Company_Name_Reservation
    (
        reservation_ID,
        client_ID
    )
    VALUES
    (
        @reservationID, -- reservation_ID - int
        @nameID -- client_ID - int
    )
END

GO

```

7.4.8. Dodanie stolika (AddTable)

- dodaje stolik konkretnej restauracji do tabeli "Tables"

```

CREATE PROCEDURE [dbo].[AddTable]
    @chairsAmmout AS INT,
    @restaurantID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newTableID AS INT;

SET @newTableID = (SELECT MAX(table_ID)+1 FROM dbo.Tables);

IF @restaurantID NOT IN (SELECT restaurant_ID FROM dbo.Restaurants)
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Tables
(
    table_ID,
    chairs_ammount,
    restaurant_ID
)
VALUES
(
    @newTableID, -- table_ID - int
    @chairsAmmout, -- chairs_ammount - int
    @restaurantID -- restaurant_ID - int
)

GO

```

7.4.9. Tworzenie ograniczenia na stoliki (AddRestriction)

- dodaje ograniczenie na stolik

```

CREATE PROCEDURE [dbo].[AddRestriction]
    @tableID AS INT,
    @startDate AS DATE,

```

```

        @endDate AS DATE,
        @charsLimit AS FLOAT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newRestrictionID AS INT;
DECLARE @newLimit AS INT;

IF 0 > @charsLimit OR @charsLimit > 1
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

SET @newRestrictionID = (SELECT MAX(restriction_ID)+1 FROM
dbo.Reservations);
SET @newLimit = FLOOR((SELECT chairs_ammount FROM dbo.Tables WHERE @tableID
= table_ID) * @charsLimit);

INSERT INTO dbo.Restrictions
(
    restriction_ID,
    table_ID,
    start_date,
    end_date,
    limit_chairs
)
VALUES
(
    @newRestrictionID,          -- restriction_ID - int
    @tableID,                  -- table_ID - int
    @startDate, -- start_date - date
    @endDate, -- end_date - date
    @newLimit                  -- limit_chairs - int
)

GO

```

7.4.10. Dodanie pozycji menu (AddDishToMenu)

- dodaje nową pozycję do menu dla konkretnej restauracji

```

CREATE PROCEDURE [dbo].[AddDishToMenu]
    @dishID AS INT,
    @addDate AS DATE,
    @removeDate AS DATE,
    @restaurantID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newMenuID AS INT;

IF @restaurantID NOT IN (SELECT restaurant_ID FROM dbo.Restaurants) OR
@dishID NOT IN (SELECT [Numer dania] FROM
dbo.DishesThatCanBeAddedToMenu(@restaurantID))
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);

```

```

END

SET @newMenuID = (SELECT MAX(Menu_ID)+1 FROM dbo.Menu)

INSERT INTO dbo.Menu
(
    Menu_ID,
    Dish_ID,
    Add_date,
    Remove_date,
    restaurant_ID
)
VALUES
(
    @newMenuID,          -- Menu_ID - int
    @dishID,             -- Dish_ID - int
    @addDate, -- Add_date - date
    @removeDate, -- Remove_date - date
    @restaurantID        -- restaurant_ID - int
)

GO

```

7.4.11. Dodanie typu rabatu (AddDiscountDictionary)

- dodaje nowy typ rabatu dla konkretnej restauracji

```

CREATE PROCEDURE [dbo].[AddDiscountDictionary]
    @countOfOrders AS INT,
    @valueOrders AS MONEY = NULL,
    @countOfDays AS INT = NULL,
    @discountValue AS FLOAT,
    @maxDiscountValue AS FLOAT,
    @discountDescription AS VARCHAR(50),
    @oneTimeDiscount AS BIT,
    @fromDate AS DATE,
    @toDate AS DATE,
    @restaurantID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newDiscountTypeID AS INT;

IF @restaurantID NOT IN (SELECT restaurant_ID FROM dbo.Restaurants)
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

SET @newDiscountTypeID = (SELECT MAX(Discount_type_ID)+1 FROM
dbo.Discount_dictionary)

INSERT INTO dbo.Discount_dictionary
(
    Discount_type_ID,
    Count_Of_Orders,
    Value_Orders,
    Count_Of_Days,

```

```

Discount_Value,
Max_Discount_Value,
Discount_description,
One_Time_Discount,
From_date,
To_date,
Restaurant_ID
)
VALUES
(
    @newDiscountTypeID,          -- Discount_type_ID - int
    @countOfOrders,              -- Count_Of_Orders - int
    @valueOrders,                -- Value_Orders - money
    @countOfDays,                -- Count_Of_Days - int
    @discountValue,              -- Discount_Value - float
    @maxDiscountValue,           -- Max_Discount_Value - float
    @discountDescription,         -- Discount_description - varchar(50)
    @oneTimeDiscount,            -- One_Time_Discount - bit
    @fromDate, -- From_date - date
    @toDate, -- To_date - date
    @restaurantID                -- Restaurant_ID - int
)

GO

```

7.4.12. Złożenie zamówienia (AddOrder)

- dodaje zamówienie do tabeli "Orders" i zaznacza w systemie, czy zamówienie jest opłacone

```

CREATE PROCEDURE [dbo].[AddOrder]
    @clientID AS INT,
    @employeeID AS INT,
    @takeaway AS BIT,
    @orderDate AS DATE,
    @pickupTime AS DATETIME,
    @discountID AS INT,
    @isPaid AS BIT
AS
DECLARE @newOrderID AS INT;
DECLARE @error AS VARCHAR(20);

IF @orderDate < GETDATE()
BEGIN
    SET @error = 'Nie możesz złożyć zamówienia!';
    RAISERROR(@error, 30, 1);
    RETURN;
END

SET @newOrderID = ISNULL((SELECT MAX(order_ID)+1 FROM dbo.Orders), 0);

INSERT INTO dbo.Orders
(
    order_ID,
    client_ID,
    employee_ID,
    takeaway,

```



```

        order_date,
        pickup_time,
        discount_ID
    )
VALUES
(
    @newOrderID,          -- order_ID - int
    @clientID,            -- client_ID - int
    @employeeID,          -- employee_ID - int
    @takeaway,            -- takeaway - bit
    @orderDate,           -- order_date - date
    @pickupTime,          -- pickup_time - datetime
    @discountID           -- discount - float
)

IF @isPaid = 0
BEGIN
    INSERT INTO dbo.Orders_Unpaid
    (
        order_ID
    )
VALUES
(@newOrderID -- order_ID - int
)
END

GO

```

7.4.13. Dodawanie szczegółów do zamówieni (AddOrderDetails)

- Dodaje danie z menu do zamówienia

```

CREATE PROCEDURE [dbo].[AddOrderDetails]
    @orderID AS INT,
    @menuID AS INT,
    @quantity AS INT,
    @price AS MONEY
AS

INSERT INTO dbo.Order_details
(
    Order_ID,
    Menu_ID,
    Quantity,
    Price
)
VALUES
(
    @orderID,    -- Order_ID - int
    @menuID,     -- Menu_ID - int
    @quantity,   -- Quantity - int
    @price -- Price - money
)

GO

```

7.4.14. Dodanie dostawcy (AddSupplier)

- dodaje nowego dostawcę do tabeli "Suppliers"

```
CREATE PROCEDURE [dbo].[AddSupplier]
    @supplierName AS VARCHAR(30),
    @townName AS VARCHAR(50),
    @countryName AS VARCHAR(50),
    @NIP AS VARCHAR(10),
    @phoneNumber AS VARCHAR(12),
    @email AS VARCHAR(20),
    @street AS VARCHAR(20),
    @postalCode AS VARCHAR(10),
    @bankAccount AS VARCHAR(26)
AS

DECLARE @newSupplierID AS INT;
SET @newSupplierID = (SELECT MAX(supplier_ID)+1 FROM dbo.Suppliers);

IF @townName NOT IN (
    SELECT town_name FROM dbo.Towns AS t
    INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
    INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
    WHERE t.town_name = @townName AND c.country_name = @countryName
)
BEGIN
    EXEC dbo.AddTown @townName = @townName, @countryName = @countryName;
END

INSERT INTO dbo.Suppliers
(
    supplier_ID,
    town_ID,
    NIP_number,
    supplier_name,
    supplier_phone,
    supplier_email,
    street,
    postal_code,
    bank_account
)
VALUES
(
    @newSupplierID, -- supplier_ID - int
    (SELECT tc.town_ID FROM dbo.Towns AS t
    INNER JOIN dbo.Towns_connections tc ON t.town_ID = tc.town_ID
    INNER JOIN dbo.Countries c ON c.country_ID = tc.country_ID
    WHERE t.town_name = @townName AND c.country_name = @countryName
    ), -- town_ID - int
    @NIP, -- NIP_number - varchar(10)
    @supplierName, -- supplier_name - varchar(30)
    @phoneNumber, -- supplier_phone - varchar(12)
    @email, -- supplier_email - varchar(20)
    @street, -- street - varchar(20)
    @postalCode, -- postal_code - varchar(10)
    @bankAccount -- bank_account - varchar(26)
)

GO
```

7.4.15. Dodanie składnika (AddIngredient)

- dodaje nowy składnik dla konkretnej restauracji do tabeli "Ingredients"

```
CREATE PROCEDURE [dbo].[AddIngredient]
    @name AS VARCHAR(20),
    @supplierID AS INT,
    @inStock AS INT = 0,
    @price AS MONEY = 0.01,
    @quantity AS VARCHAR(20),
    @extraInformation AS VARCHAR(50) = NULL,
    @restaurantID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newIngredientID AS INT;

SET @newIngredientID = (SELECT MAX(ingredient_ID)+1 FROM dbo.Ingredients);

IF @supplierID NOT IN (SELECT supplier_ID FROM dbo.Suppliers) OR
@restaurantID NOT IN (SELECT @restaurantID FROM dbo.Restaurants)
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Ingredients
(
    ingredient_ID,
    supplier_ID,
    ingredient_name,
    ingredient_in_stock,
    ingredient_price,
    ingredient_quantity,
    safe_amout,
    extra_informations,
    locked_date,
    unlocked_date,
    restaurant_ID
)
VALUES
(
    @newIngredientID,          -- ingredient_ID - int
    @supplierID,              -- supplier_ID - int
    @name,                    -- ingredient_name - varchar(20)
    @inStock,                 -- ingredient_in_stock - int
    @price,                   -- ingredient_price - money
    @quantity,                -- ingredient_quantity - varchar(20)
    5,                        -- safe_amout - int
    @extraInformation,         -- extra_informations - varchar(50)
    NULL, -- locked_date - date
    NULL, -- unlocked_date - date
    @restaurantID             -- restaurant_ID - int
)

GO
```

7.4.16. Dodanie dania (AddDish)

- dodaje nowe danie dla konkretnej restauracji do tabeli "Dishes"

```
CREATE PROCEDURE [dbo].[AddDish]
    @name AS VARCHAR(30),
    @categoryName AS VARCHAR(15),
    @price AS INT,
    @executionTime AS INT,
    @restaurantID AS INT
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newDishID AS INT;

IF @restaurantID NOT IN (SELECT restaurant_ID FROM dbo.Restaurants)
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

SET @newDishID = (SELECT MAX(dish_ID)+1 FROM dbo.Dishes);

IF @categoryName NOT IN (SELECT category_name FROM dbo.Categories)
BEGIN
    EXEC dbo.AddCategory @categoryName = @categoryName          -- varchar(15)
END

INSERT INTO dbo.Dishes
(
    dish_ID,
    category_ID,
    dish_name,
    price,
    execution_time,
    locked_date,
    unlocked_date,
    restaurant_ID
)
VALUES
(
    @newDishID,          -- dish_ID - int
    (SELECT category_ID FROM dbo.Categories WHERE category_name =
@categoryName),        -- category_ID - int
    @name,               -- dish_name - varchar(30)
    @price,              -- price - money
    @executionTime,      -- execution_time - int
    NULL, -- locked_date - date
    NULL, -- unlocked_date - date
    @restaurantID        -- restaurant_ID - int
)

GO
```

7.4.17. Łączenie składnika z daniem (ConnectDishWithIngredient)

- dodaje do tabeli "Dish_details" informacje o składniku dla poszczególnego dania

```
CREATE PROCEDURE [dbo].[ConnectDishWithIngredient]
    @dishID AS INT,
    @ingredientID AS INT,
    @quantity AS FLOAT,
    @price AS MONEY
AS

DECLARE @error AS VARCHAR(20);
DECLARE @dishRestaurantID AS INT;
DECLARE @ingredientRestaurantID AS INT;

SET @dishRestaurantID = (SELECT restaurant_ID FROM dbo.Dishes WHERE dish_ID
= @dishID);
SET @ingredientRestaurantID = (SELECT restaurant_ID FROM dbo.Ingredients
WHERE ingredient_ID = @ingredientID)

IF @dishID NOT IN (SELECT dish_ID FROM dbo.Dishes) OR @ingredientID NOT IN
(SELECT ingredient_ID FROM dbo.Ingredients) OR @dishRestaurantID !=
@ingredientRestaurantID
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Dish_Details
(
    dish_ID,
    ingredient_ID,
    Quantity,
    Price
)
VALUES
(
    @dishID,    -- dish_ID - int
    @ingredientID, -- ingredient_ID - int
    @quantity, -- Quantity - float
    @price -- Price - money
)

GO
```

7.4.18. Dodanie klienta do firmy (AddIndyvidualToCompany)

- łączy klienta indywidualnego z daną firmą

```
CREATE PROCEDURE [dbo].[AddIndyvidualToCompany]
    @indyvidualID AS INT,
    @companyID AS INT
AS

DECLARE @error AS VARCHAR(20);

IF @indyvidualID NOT IN (SELECT client_ID FROM dbo.Indyvidual_clients) OR
@companyID NOT IN (SELECT client_ID FROM dbo.Companies)
BEGIN
```

```

        SET @error = 'Błędne dane!';
        RAISERROR(@error, 16, 1);
    END

    INSERT INTO dbo.Company_Members
    (
        Company_ID,
        Indyvidual_Client_ID
    )
    VALUES
    (
        @companyID, -- Company_ID - int
        @indyvidualID -- Indyvidual_Client_ID - int
    )

GO

```

- 7.4.19. Aktualizowanie składnika w magazynie (UpdateIngeredientAmmount)
- zaznacza w systemie, jak ilość składnika została dodana/zużyta

```

CREATE PROCEDURE [dbo].[UpdateIngeredientAmmount]
    @ingredientID AS INT,
    @ammount AS INT
AS

UPDATE dbo.Ingredients
SET ingredient_in_stock = ingredient_in_stock + @ammount
WHERE ingredient_ID = @ingredientID

GO

```

- 7.4.20. Usuwanie rekordu z tabeli Orders_Unpaid, jeżeli zamówienie zostało opłacone (DeleteUnpaidOrder)
- Usuwa rekord z tabeli "Orders_Unpaid"

```

CREATE PROCEDURE [dbo].[DeleteUnpaidOrder]
    @deletingOrderID AS INT
AS

DECLARE @error AS VARCHAR(20);

IF @deletingOrderID IS NULL
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

DELETE FROM dbo.Orders_Unpaid
WHERE order_ID = @deletingOrderID

GO

```

7.4.21. Dodanie kategorii (AddCategory)

- Dodaje nową kategorię do tabeli “Categories”

```
CREATE PROCEDURE [dbo].[AddCategory]
    @categoryName VARCHAR(15),
    @categoryDescription VARCHAR(40) = NULL
AS

DECLARE @error AS VARCHAR(20);

IF @categoryName IS NULL
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Categories
(
    category_ID,
    category_name,
    category_description
)
VALUES
(
    (SELECT MAX(category_ID)+1 FROM dbo.Categories), -- category_ID - int
    @categoryName, -- category_name - varchar(15)
    @categoryDescription -- category_description - varchar(40)
)

GO
```

7.4.22. Dodanie miasta (AddTown)

- Dodaje miasto do tabeli “Towns”, oraz łączy miasto z krajem

```
CREATE PROCEDURE [dbo].[AddTown]
    @townName VARCHAR(50),
    @countryName VARCHAR(50)
AS

DECLARE @error AS VARCHAR(20);
DECLARE @newTownID AS INT

IF @townName IS NULL OR @countryName IS NULL
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

SET @newTownID = (SELECT MAX(town_ID)+1 FROM dbo.Towns);

INSERT INTO dbo.Towns
(
    town_ID,
    town_name
)
VALUES
(
    @newTownID, -- town_ID - int
```

```

        @townName -- town_name - varchar(50)
    )

IF @countryName NOT IN (SELECT country_name FROM dbo.Countries)
BEGIN
    EXEC dbo.AddCountry @countryName = @countryName;
END

INSERT INTO dbo.Towns_connections
(
    town_ID,
    country_ID
)
VALUES
(
    @newTownID, -- town_ID - int
    (SELECT country_ID FROM dbo.Countries WHERE country_name =
@countryName) -- country_ID - int
)

GO

```

7.4.23. Dodanie kraju (AddCountry)

- Dodaje kraj do tabeli "Countries"

```

CREATE PROCEDURE [dbo].[AddCountry]
    @countryName VARCHAR(50)
AS
DECLARE @error AS VARCHAR(20);

IF @countryName IS NULL
BEGIN
    SET @error = 'Błędne dane!';
    RAISERROR(@error, 16, 1);
END

INSERT INTO dbo.Countries
(
    country_ID,
    country_name
)
VALUES
(
    (SELECT MAX(country_ID)+1 FROM dbo.Countries),
    @countryName -- country_name - varchar(50)
)

GO

```

7.4.24. Pozycje będące w menu ponad 2 tygodnie (DishesInMenuOverTwoWeeks)

```

CREATE FUNCTION [dbo].[DishesInMenuOverTwoWeeks]
(
    @restaurantID AS INT
)
RETURNS TABLE

```



```

AS
RETURN
(
    SELECT m.Menu_ID FROM dbo.MenuToday(@restaurantID) AS mt
    INNER JOIN dbo.Menu m ON mt.[Numer dania] = m.Dish_ID
    WHERE DATEDIFF(DAY, m.Add_date, GETDATE()) > 14 AND m.Remove_date >
GETDATE()
)
GO

```

7.4.25. Przydzielenie firmie rabatu (AddNewDiscountToCompanies)

- procedura uruchamiana co miesiąc, tak by sprawdzić czy firmom można naliczyć rabat miesięczny/kwartalny

```

CREATE PROCEDURE [dbo].[AddNewDiscountToCompanies]
    @clientID AS INT
AS
BEGIN
    DECLARE @valueOfFirstTypeOfDiscount FLOAT;
    DECLARE @firstDiscountTypeID INT;
    DECLARE @newDiscountID INT;

    SET @valueOfFirstTypeOfDiscount =
    dbo.GetCompanyFirstDiscountValue(@clientID, GETDATE());

    IF @valueOfFirstTypeOfDiscount > 0
    BEGIN
        SET @firstDiscountTypeID = (SELECT Discount_type_ID FROM
        dbo.Discount_dictionary WHERE Max_Discount_Value IS NOT NULL AND
        One_Time_Discount = 0 AND Count_Of_Days IS NULL AND GETDATE() >= From_date
        AND GETDATE() <= To_date );
        SET @newDiscountID = ISNULL((SELECT MAX(discount_ID)+1 FROM
        dbo.Discounts),0);

        INSERT INTO dbo.Discounts
        (
            discount_ID,
            client_ID,
            discount_value,
            start_date,
            end_date,
            discount_type_ID
        )
        VALUES
        (
            @newDiscountID,          -- discount_ID - int
            @clientID,              -- client_ID - int
            @valueOfFirstTypeOfDiscount, -- discount_value - float
            GETDATE(),              -- start_date - date
            DATEADD(MONTH, 1, GETDATE()), -- end_date - date
            @firstDiscountTypeID     -- discount_type_ID - int
        )
    END

    DECLARE @secondDiscountTypeID INT;

```

```

DECLARE @valueOfSecondTypeOfDiscount FLOAT;

SET @secondDiscountTypeID = dbo.GetCompanySecondDiscountID(@clientID,
GETDATE());

IF @secondDiscountTypeID != -1
BEGIN
    SET @valueOfSecondTypeOfDiscount = (SELECT Discount_Value FROM
dbo.Discount_dictionary WHERE Discount_type_ID = @secondDiscountTypeID);
    SET @newDiscountID = ISNULL((SELECT MAX(discount_ID)+1 FROM
dbo.Discounts),0);

    INSERT INTO dbo.Discounts
    (
        discount_ID,
        client_ID,
        discount_value,
        start_date,
        end_date,
        discount_type_ID
    )
    VALUES
    (
        @newDiscountID,          -- discount_ID - int
        @clientID,              -- client_ID - int
        @valueOfSecondTypeOfDiscount, -- discount_value - float
        GETDATE(), -- start_date - date
        DATEADD(MONTH, 3, GETDATE()), -- end_date - date
        @secondDiscountTypeID    -- discount_type_ID - int
    )
END

END
GO

```

7.4.26. Procedura wysyłająca informację do użytkownika, jeżeli ponad połowa pozycji w menu jest ponad 2 tygodnie (MenuMessage)

```

CREATE PROCEDURE [dbo].[MenuMessage]
@restaurantID AS INT
AS

DECLARE @countAllMenu AS INT;
DECLARE @countMenuOverTwoWeeks AS INT;

SET @countAllMenu = (SELECT COUNT(*) FROM dbo.MenuToday(@restaurantID));
SET @countMenuOverTwoWeeks = (SELECT COUNT(*) FROM
dbo.DishesInMenuOverTwoWeeks(@restaurantID));

IF @countMenuOverTwoWeeks > 0.5 * @countAllMenu
BEGIN
    PRINT 'Trzeba zmienić menu!'
END
GO

```

7.5. Triggery

7.5.1. Przydzielenie klientowi indywidualnemu rabatu (AddNewDiscountToIndyvidualClient)

```
CREATE TRIGGER [dbo].[AddNewDiscountToIndyvidualClient]
ON [dbo].[Orders]
AFTER INSERT
AS
BEGIN
    DECLARE @clientsTable TABLE (
        ind INT PRIMARY KEY IDENTITY(1,1),
        clientID INT
    )

    INSERT @clientsTable
    SELECT DISTINCT Inserted.client_ID FROM Inserted

    DECLARE @i INT;
    DECLARE @maxIndex INT;
    DECLARE @clientID INT;
    DECLARE @firstDiscountValue FLOAT;
    DECLARE @newDiscountID INT;

    SET @i = (SELECT MIN(ind) FROM @clientsTable);
    SET @maxIndex = (SELECT MAX(ind) FROM @clientsTable);

    WHILE @i <= @maxIndex
    BEGIN
        SET @clientID = (SELECT clientID FROM @clientsTable WHERE ind =
        @i);
        SET @firstDiscountValue =
        dbo.GetFirstOrSecondDiscountValue(@clientID);

        IF @firstDiscountValue > 0
        BEGIN
            DECLARE @newDiscountTypeID INT;
            SET @newDiscountTypeID = (SELECT Discount_type_ID FROM
            dbo.Discount_dictionary WHERE From_date <= GETDATE() AND GETDATE() <=
            To_date AND Count_Of_Orders IS NOT NULL);
            SET @newDiscountID = ISNULL( (SELECT MAX(discount_ID)+1 FROM
            dbo.Discounts), 0);

            INSERT INTO dbo.Discounts
            (
                discount_ID,
                client_ID,
                discount_value,
                start_date,
                end_date,
                discount_type_ID
            )
            VALUES
            (
                @newDiscountID,          -- discount_ID - int
                @clientID,                -- client_ID - int
                @firstDiscountValue,      -- discount_value - float
            )
        END
    END
END
```

```

        GETDATE(), -- start_date - date
        DATEADD(YEAR, 200, GETDATE()), -- end_date - date
        @newDiscountTypeID             -- discount_type_ID - int
    )

END

DECLARE @thirdAndFourthDiscountType INT;
SET @thirdAndFourthDiscountType =
dbo.GetThirdAndFourthDiscountID(@clientID);

IF @thirdAndFourthDiscountType != -1
BEGIN
    DECLARE @discountValue FLOAT;
    DECLARE @daysOfUse INT;

    SET @discountValue = (SELECT Discount_Value FROM
        dbo.Discount_dictionary WHERE @thirdAndFourthDiscountType =
        Discount_type_ID);
    SET @daysOfUse = (SELECT Count_Of_Days FROM
        dbo.Discount_dictionary WHERE @thirdAndFourthDiscountType =
        Discount_type_ID);
    SET @newDiscountID = ISNULL( (SELECT MAX(discount_ID)+1 FROM
        dbo.Discounts) ,0);

    INSERT INTO dbo.Discounts
    (
        discount_ID,
        client_ID,
        discount_value,
        start_date,
        end_date,
        discount_type_ID
    )
    VALUES
    (
        @newDiscountID,             -- discount_ID - int
        @clientID,                 -- client_ID - int
        @discountValue,             -- discount_value - float
        GETDATE(), -- start_date - date
        DATEADD(DAY, @daysOfUse, GETDATE()), -- end_date - date
        @thirdAndFourthDiscountType -- discount_type_ID -
int
    )

END

SET @i = @i + 1;
END

END
GO

ALTER TABLE [dbo].[Orders] ENABLE TRIGGER
[AddNewDiscountToIndyvidualClient]
GO

```

7.5.2. Jeżeli składnik zostanie zablokowany to wszystkie dania zawierające ten składnik też zostaną zablokowane (LockDish)

```
CREATE TRIGGER [dbo].[LockDish]
ON [dbo].[Ingredients]
AFTER UPDATE
AS
    UPDATE dbo.Dishes
    SET locked_date = GETDATE(), unlocked_date=DATEADD(DAY, 30, GETDATE())
    WHERE dish_ID IN (SELECT dd.dish_ID FROM Inserted AS i
    INNER JOIN dbo.Dish_Details dd ON dd.ingredient_ID = i.ingredient_ID
    )
GO

ALTER TABLE [dbo].[Ingredients] ENABLE TRIGGER [LockDish]
GO
```

7.5.3. Jeżeli danie zostało zablokowane i znajduje się w menu, danie zostaje usunięte z aktualnego menu(LockMenu)

- usunięcie z menu polega na zmianie daty usunięcia z menu (remove_date) na datę dzisiejszą

```
CREATE TRIGGER [dbo].[LockMenu]
ON [dbo].[Dishes]
AFTER UPDATE
AS
    UPDATE dbo.Menu
    SET Remove_date = GETDATE()
    WHERE Menu_ID IN (SELECT m.Menu_ID FROM Inserted i
    INNER JOIN dbo.Menu m ON m.Dish_ID = i.dish_ID)
GO

ALTER TABLE [dbo].[Dishes] ENABLE TRIGGER [LockMenu]
GO
```

7.5.4. Blokada składnika, jeżeli nie będzie wystarczającej ilości w magazynie
(LockIngredientWithLowAmount)

```
CREATE TRIGGER [dbo].[LockIngredientWithLowAmount] ON [dbo].[Ingredients]
AFTER UPDATE
AS
    UPDATE dbo.Ingredients
    SET locked_date = GETDATE(), unlocked_date = DATEADD(MONTH, 1,
GETDATE())
    WHERE ingredient_ID IN (SELECT Inserted.ingredient_ID FROM Inserted)
AND ingredient_in_stock < safe_amout
GO

ALTER TABLE [dbo].[Ingredients] ENABLE TRIGGER
[LockIngredientWithLowAmount]
GO
```

7.5.5. Sprawdzenie przy rezerwacji czy zamówienie klienta indywidualnego nie
jest czasem na wynos (checkCorrectnessOfAddedReservation)

```
CREATE TRIGGER [dbo].[checkCorrectnessOfAddedReservation]
ON [dbo].[Reservations]
AFTER INSERT
AS

IF EXISTS (SELECT o.takeaway FROM Inserted AS i
INNER JOIN dbo.Personal_Reservations pr ON pr.reservation_ID =
i.reservation_ID
INNER JOIN dbo.Orders o ON o.client_ID = i.client_ID
WHERE o.takeaway = 1)
BEGIN
    RAISERROR('Nie można dodać rezerwacji', 16, 1);
    ROLLBACK TRANSACTION
    RETURN
END
GO

ALTER TABLE [dbo].[Reservations] ENABLE TRIGGER
[checkCorrectnessOfAddedReservation]
GO
```

7.6. Widoki

7.6.1. Informacje o restauracjach w bazie (RestaurantsInformations)

```
CREATE VIEW [dbo].[RestaurantsInformations]
AS
SELECT      r.restaurant_name, r.address + ',' + t.town_name + ',' +
c.country_name AS Address, r.phone_number
FROM        dbo.Restaurants AS r INNER JOIN
            dbo.Towns AS t ON t.town_ID = r.town_ID INNER JOIN
            dbo.Towns_connections AS tc ON tc.town_ID =
t.town_ID INNER JOIN
            dbo.Countries AS c ON c.country_ID = tc.country_ID
GO
```

7.6.2. Wszystkie kategorie dań w systemie (CategoriesInformations)

```
CREATE VIEW [dbo].[CategoriesInformations]
AS
SELECT      category_name, category_description
FROM        dbo.Categories
GO
```

7.6.3. Wszystkie miasta obecne w bazie (TownsAndCountriesNames)

```
CREATE VIEW [dbo].[TownsAndCountriesNames]
AS
SELECT      TOP (100) PERCENT dbo.Countries.country_name,
dbo.Towns.town_name
FROM        dbo.Towns INNER JOIN
            dbo.Towns_connections ON dbo.Towns.town_ID =
dbo.Towns_connections.town_ID INNER JOIN
            dbo.Countries ON dbo.Towns_connections.country_ID
= dbo.Countries.country_ID
ORDER BY   dbo.Countries.country_name, dbo.Towns.town_name
GO
```

8. Użytkownicy systemu i ich uprawnienia

Na podstawie opisu miniświata oraz analizując funkcjonalności systemu zidentyfikowani zostali użytkownicy systemu.

Użytkownicy systemu:

- Administrator systemu - dostęp do wszystkich danych i operacji, umożliwiających zarządzanie bazą danych
- Klient Indywidualny
 - dostęp do aktualnego menu wybranej restauracji
 - dostęp do historii swoich zamówień
 - dostęp do obecnych i archiwalnych rezerwacji wykonanych w danej restauracji
 - dostęp do danych kontaktowych wybranych restauracji
 - możliwość dokonywania rezerwacji stolika (przy spełnionych warunkach), wykonywania zamówień na wynos, wyświetlenia przysługujących mu rabatów
- Firma
 - możliwość wystawienia faktury za zamówienie lub zbiorczej miesięcznej
 - możliwość dokonywania rezerwacji imiennych lub na firmę (bez ograniczeń)
 - dostęp do aktualnego menu restauracji
 - dostęp do danych kontaktowych wybranych restauracji
 - możliwość wyświetlenia wszystkich przysługujących rabatów
 - dostęp do historii swoich zamówień
 - dostęp do obecnych i archiwalnych rezerwacji wykonanych w danej restauracji
 - dodawanie klientów indywidualnych jako pracowników firmy
- Pracownik restauracji
 - dostęp do dań oferowanych w danej restauracji
 - dostęp do wszystkich pozycji (również przeszłych) menu
 - dostęp do listy składników wraz z informacjami o ich dostępności w magazynie
 - dostęp do informacji o dostawcach składników wykorzystywanych przez restaurację
 - dostęp do informacji na temat klientów restauracji oraz ich rabatów
 - dostęp do zamówień i rezerwacji
 - możliwość dodawania nowych dań oferowanych przez restaurację
 - możliwość modyfikowania aktualnego menu poprzez dodawanie i usuwanie pozycji
 - możliwość usuwania zamówień z listy zamówień niezapłaconych
 - dodawanie ograniczeń ilościowych na miejsca przy stoliku
 - aktualizowanie ilości składnika w magazynie
 - możliwość dodawania nowych składników do bazy
 - możliwość generowania faktur dla firm
- szef restauracji
 - wszystkie uprawnienia co ma pracownik restauracji
 - generowanie raportów dotyczących danej restauracji
 - dodawanie nowych pracowników restauracji
 - dodawanie typów rabatów

