



Państwowa Wyższa Szkoła Zawodowa w Tarnowie
Wydział Politechniczny
Informatyka

Projekt Bazy danych

System zarządzania pizzerią

Wykonawca:

Ernest Bieś

Prowadzący:

mgr inż. Tomasz Potempa

Tarnów, 2020

Spis treści

1. Sformułowanie zadania projektowego.....	3
2. Analiza stanu wyjściowego.....	4
2.1. Analiza stanu zastanego, obieg dokumentacji.....	4
2.2. Uwarunkowania prawne.....	4
3. Grupy użytkowników w systemie.....	5
4. Określenie scenariuszy użytkownika i przypadków użycia.....	7
4.1. Scenariusze użytkownika.....	7
4.1.1. Przyjęcie zamówienia.....	7
4.1.2. Wybór pizzy.....	8
4.1.3. Założenie konta użytkownika.....	8
4.1.4. Realizacja zamówienia.....	9
4.1.5. Doręczenie zamówienia.....	9
4.1.6. Obsługa systemu zarządzania pizzerią.....	10
4.2. Przypadki użycia.....	11
5. Diagram ERD.....	12
6. Tabele.....	14
7. Sekwencje.....	22
8. Widoki.....	23
8.1 Adresy klientów.....	24
8.2 Zamówienia klientów.....	24
8.3 Pełne dane klientów.....	24
8.4 Konta klientów.....	25
8.5 Cena zamówienia.....	25
8.5 Ranking klientów.....	25
8.6 Szczegóły zamówienia.....	26
8.7 Kwerendy zapisane w widokach.....	26
9. Funkcje.....	29
9.1 Funkcja generuj_rekordy().....	30
9.2 Funkcja hash(character varying, character varying, integer).....	34
9.3 Funkcja oblicz_rabat(integer, numeric).....	35
9.4 Funkcja sprawdz_login(character varying).....	37
9.5 Funkcja zamowienia_max_dzien_tyg(date, date).....	38
10. Wyzwalacze.....	40
10.1 Funkcja wyzwalacza oblicz_cene().....	41
10.2 Funkcja wyzwalacza oblicz_punkty_klienta().....	42
10.3 Funkcja wyzwalacza oblicz_punkty_zamowienia().....	43
10.4 Funkcja wyzwalacza hash_haslo().....	45

1. Sformułowanie zadania projektowego

Głównym celem projektu było stworzenie bazy danych dla pizzerii, która umożliwi klientowi składanie zamówień dotyczących wybranego rodzaju pizzy wraz z dodatkami, a także po wyrażeniu zgody na rejestrację w bazie pizzerii, uzyskiwanie odpowiednich rabatów.

Złożenie zamówienia jest możliwe za pośrednictwem Internetu, telefonicznie lub osobiście. W przypadku złożenia zamówienia telefonicznie lub osobiście pracownik pizzerii, przyjmujący zamówienie wprowadza dane do systemu, a następnie kompletuje zamówienia. Jeżeli klient wyrazi zgodę na rejestrację danych osobowych w systemie, zostaje on objęty rabatami i zniżkami.

Po zarejestrowaniu danych oraz uzyskaniu informacji, że zamawiana pizza wraz z wybranymi dodatkami może zostać przygotowana, klient uiszcza opłatę w wybranej formie. Po potwierdzeniu wpłaty przygotowanie pizzy jest przekazywane do realizacji. Przy zamówieniu przez Internet lub telefonicznie klient dokonuje płatności za pośrednictwem doręczyciela pizzy.

Cele projektu:

- Zarządzanie systemem realizacji zamówień wybranego produktu.
- Przechowywanie informacji na temat produktów (pizzy) oraz dodatków.
- Przechowywanie listy użytkowników i ich podstawowe dane.
- Możliwość złożenia zamówienia danego produktu.
- Możliwość oceny zamówienia przez użytkownika.
- Możliwość ustalenia menu przez Managera.
- Niezawodność oraz szybki dostęp do informacji.

2. Analiza stanu wyjściowego.

2.1. Analiza stanu zastanego, obieg dokumentacji.

Obecnie w pizzerii nie działał żaden system informatyczny, zamówienia były przyjmowane telefonicznie lub osobiście na miejscu. Informacje o zamówieniach zapisywane były na pojedynczych kartkach papieru i przekazywane do kuchni celem realizacji. Informacje o przygotowanym zamówieniu były przekazywane osobiście przez kucharza.

2.2. Uwarunkowania prawne.

W chwili obecnej działalność pizzerii była prowadzona w oparciu o podstawowe przepisy prawne:

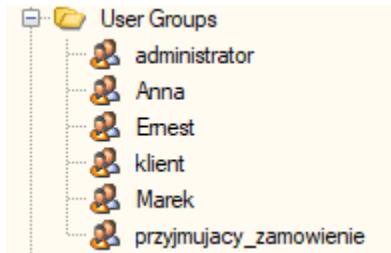
- Ustawa z dnia 2 lipca 2004 r. o swobodzie działalności gospodarczej
- Ustawa z dnia 11.03.2004 r. o podatku od towarów i usług

Z chwilą uruchomienia systemu informatycznego należy wdrożyć dodatkowe przepisy:

- Ustawa z dnia 10 maja 2018 r. o ochronie danych osobowych
- Rozporządzenie Parlamentu Europejskiego i Rady (UE) 2016/679 z dnia 27 kwietnia 2016 r. w sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych i w sprawie swobodnego przepływu takich danych oraz uchylenia dyrektywy 95/46/WE (RODO).

3. Grupy użytkowników w systemie.

W systemie występują 3 grupy użytkowników:



1) **Klient** – osoba, która ma częściowy dostęp do bazy, ma możliwość zapoznania się z ofertą pizzerii, dodatkami, cenami produktów. Klient ma możliwość założenia konta. Po założeniu konta użytkownik będzie mógł złożyć zamówienie. Ma również możliwość do edytowania swoich danych zapisanych na koncie oraz oceny zamówienia. Klient ma możliwość zbierania punktów, które w przyszłości mogą zostać wymienione na rabaty. Może również składać zamówienie bezpośrednio przy kasie lub telefonicznie.

1.1 Klient niezalogowany

- Ma możliwość założenia swojego konta
- Ma dostęp do przeglądania oferty pizzerii
- Ma możliwość przeglądania szczegółowych informacji na temat produktów
- Ma możliwość sprawdzania informacji na temat pizzerii

1.2 Klient zalogowany

- Możliwość edytowania konta przeprowadzana przez użytkownika w razie zmiany adresu zameldowania, numeru telefonu, czy adresu mailowego, danych niezbędnych w celach kontaktowych.
- Możliwość składania zamówienia.
- Ma możliwość wystawiania oceny na temat zamówienia.
- Ma możliwość wyboru adresu dostarczenia pizzy.
- Ma możliwość wyboru dodatków do swojego zamówienia.

- Ma możliwość zbierania punktów, które mogą być wymienione na rabaty.
- Ma możliwość sprawdzania ilości swoich zdobytych punktów.
- Ma możliwość wystawienia komentarza dotyczącego zamówienia.

2) Przyjmujący zamówienie - zadaniem tej osoby jest wprowadzenie zamówienia do systemu, które zostało złożone telefonicznie lub osobiście. Przyjmujący zamówienie może poinformować osobę składającą zamówienie o dostępności danej oferty. Przyjmuje zapłatę za zamówienie złożone osobiście.

- Ma możliwość rejestracji zamówienia złożonego osobiście lub telefonicznie.
- Ma możliwość sprawdzenia aktualnej oferty.
- Ma możliwość sprawdzenia dostępności dodatków.
- Ma możliwość założenia konta klientowi.
- Ma możliwość zmiany adresu doręczenia produktu.
- Ma możliwość zarejestrowania oceny zamówienia w przypadku zamówienia złożonego osobiście w pizzerii.
- Ma wgląd do punktów zebranych przez klientów.
- Odnotowuje fakt realizacji zamówienia.

3) Manager (Administrator) - właściciel pizzerii, będzie miał pełny dostęp do bazy w celu sprawdzania poprawności działania pizzerii. Pełni tu rolę osoby pełniącej kontrolę nad kontami użytkowników, może je usunąć, korygować błędy powstałe, przy wprowadzaniu danych kontaktowych, popełnione przez użytkowników. Do jego zadań należy informowanie o promocji oraz zmianach w ofercie pizzerii. Zajmuje się też wprowadzaniem zmian do menu, sprawuje kontrolę nad cenami widocznymi na stronie pizzerii, w razie zmian wprowadza korekty. Panuje nad prawidłowym działaniem systemu.

- Ma pełną kontrolę bazy danych
- Ma dostęp do edycji bazy danych
- Wprowadzanie korekt cen oferty
- Wprowadzanie zmian w ofercie pizzerii (pizza oraz dodatki)
- Usuwanie konta w przypadku łamania regulaminu przez danego użytkownika.
- Usuwanie danych produktów z oferty (pizzy oraz dodatków)

- Ma dostęp do zmian cen produktów
- Ma możliwość edycji informacji na temat produktów
- Ma możliwość usuwania danych produktów z oferty
- Ma możliwość dodawania nowych produktów
- Ma możliwość edycji zamówień
- Ma możliwość ustalania punktów za zamówienie
- Ma możliwość modyfikacji przyznawanych punktów za zamówienie
- Ma możliwość przeglądania komentarzy.
- Ma możliwość usuwania komentarzy, które łamią regulamin.

4. Określenie scenariuszy użytkownika i przypadków użycia.

4.1. Scenariusze użytkownika

4.1.1. Przyjęcie zamówienia

Nazwa przypadku użycia:	<i>Przyjęcie zamówienia</i>
Aktorzy:	Klient, Przyjmujący zamówienie
Warunki wstępne:	Klient chce zamówić pizzę wraz z dodatkami.
Warunki końcowe:	Przyjmujący zamówienie przyjmuje zamówienie.
Rezultat:	Klientowi został przydzielony termin i godzina dostawy.
Scenariusz:	
	Główny: <ol style="list-style-type: none"> 1. Klient składa zamówienie osobiście, telefonicznie lub przez Internet. 2. Klient wybiera rodzaj pizzy z menu oraz dodatki. 3. Klient podaje termin oraz adres dostawy zamówienia. 4. Dokonuje płatności gotówką lub kartą. 5. Przyjmujący zamówienie przekazuje zamówienie do realizacji. 6. Przyjęcie zamówienia zostało zrealizowane.
	Alternatywny: <ol style="list-style-type: none"> 4.1. System transakcyjny odrzuca płatność kartą. 4.2. Zamówienie zostało anulowane. 4.3. Klient dokonuje płatności gotówką. 4.4. Powrót do punktu 5. scenariusza głównego.

4.1.2. Wybór pizzy

Nazwa przypadku użycia:	<i>Wybór pizzy</i>
Aktorzy:	Klient, Przyjmujący zamówienie.
Warunki wstępne:	Klient wybiera pizzę z karty lub z menu dostępnego na stronie.
Warunki końcowe:	Zamówienie zostało przekazane do realizacji.
Rezultat:	Klient oczekuje na pizzę.
Scenariusz:	
	Główny: 1. Klient z menu wybiera pizzę. 2. Przyjmujący zamówienie sprawdza w bazie dostępność składników potrzebnych do przygotowania pizzy. 3. Klient wybiera dodatki do pizzy. 4. Przyjmujący zamówienie wprowadza zamówienie do bazy danych.
	Alternatywny: 2.1. Przyjmujący zamówienie stwierdza brak składników potrzebnych do zrealizowania zamówienia. 2.2. Klientowi proponowana jest inna pizza. 2.3. Jeśli klient przyjmie propozycję, przejście do punktu 3. scenariusza głównego. 2.4. Jeśli klient odrzuci propozycję, zamówienie nie zostaje zrealizowane.

4.1.3. Założenie konta użytkownika

Nazwa przypadku użycia:	<i>Założenie konta użytkownika</i>
Aktorzy:	Klient
Warunki wstępne:	Klient nie posiada konta w pizzerii.
Warunki końcowe:	Klient zakłada konto za pośrednictwem strony internetowej.
Rezultat:	Klient utworzył swoje Konto użytkownika.
Scenariusz:	
	Główny: 1. Klient podaje swoje dane i wyraża zgodę na założenie konta. 2. Dane klienta zostają wprowadzone do bazy danych. 3. Konto zostało założone.
	Alternatywny: 1.1. Klient nie wyraził zgody na wykorzystanie danych osobowych w celu założenie konta. 1.2. Konto nie zostało założone.

4.1.4. Realizacja zamówienia

Nazwa przypadku użycia:	<i>Realizacja zamówienia</i>
Aktorzy:	Przyjmujący zamówienie
Warunki wstępne:	Przyjmujący zamówienie przekazuje zamówienie do realizacji.
Warunki końcowe:	Zamówienie jest gotowe do wydania.
Rezultat:	Klient dostaje zamówioną pizzę.
Scenariusz:	
	Główny: 1. Przyjmujący zamówienie odczytuje dane dotyczące zamówienia z bazy danych. 2. Przyjmujący zamówienie przekazuje w formie elektronicznej lub osobiście zamówienie do kuchni. 3. Informacje dotyczące zamówienia zostają odczytane z bazy danych. 4. Zamówienie zostaje przekazane doręczycielowi.
	Alternatywny: 3.1 Stwierdzono brak wystarczających informacji dotyczących zamówienia w bazie danych. 3.2 Przyjmujący zamówienie uzupełnia brakujące informacje w bazie danych. 3.3 Powrót do punktu 3. scenariusza głównego.

4.1.5. Doręczenie zamówienia

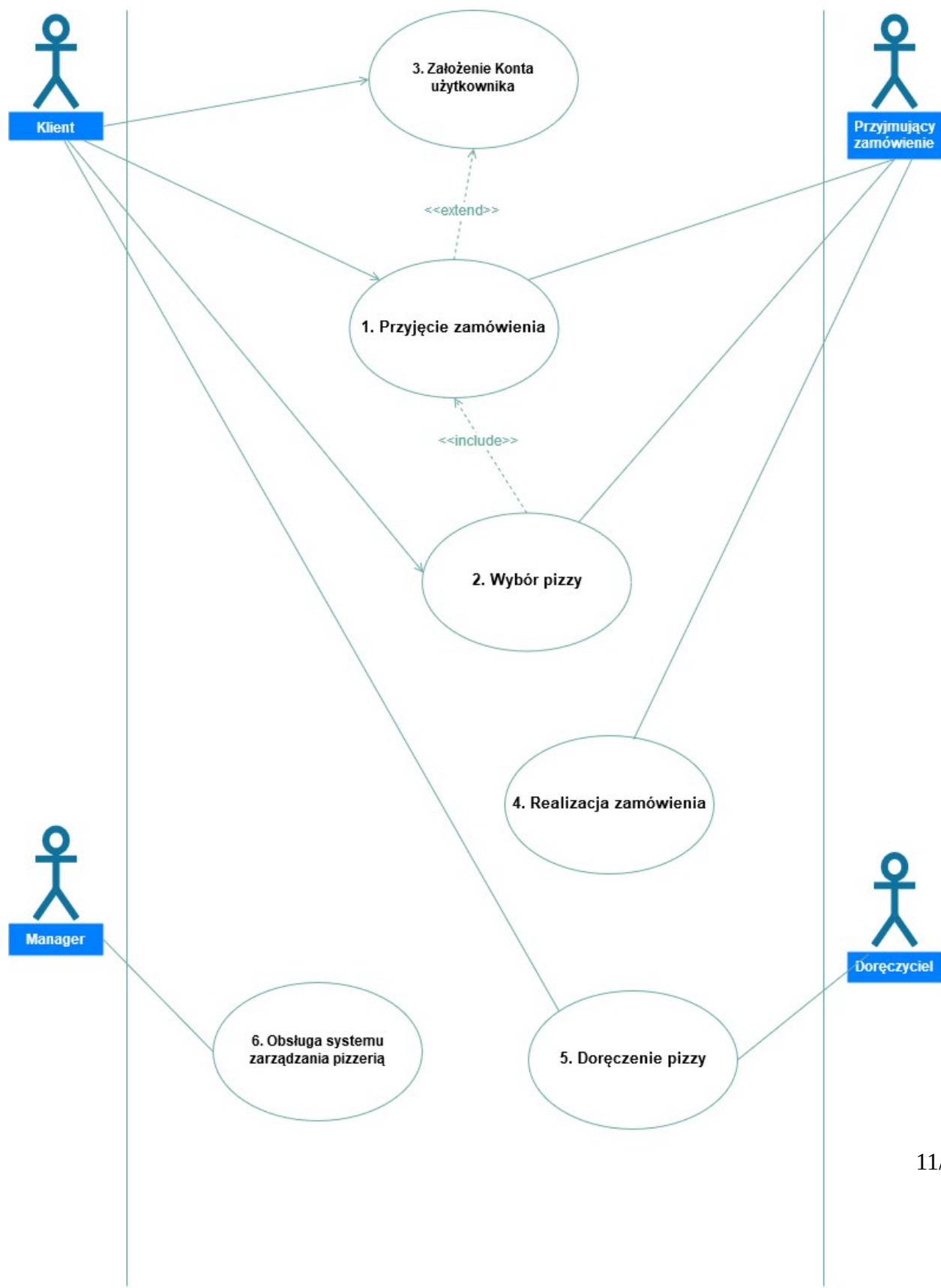
Nazwa przypadku użycia:	<i>Doręczenie zamówienia</i>
Aktorzy:	Przyjmujący zamówienie, Klient
Warunki wstępne:	Zamówienie jest gotowe do wydania.
Warunki końcowe:	Klient otrzymał pizzę.
Rezultat:	Zamówienie zostało doręczone.
Scenariusz:	
	Główny: 1. Przyjmujący zamówienie przekazuje informacje dotyczące adresu dostawy doręczycielowi pizzy. 2. Doręczyciel pizzy udaje się pod wskazany adres dostawy, odbiera płatność i przekazuje zamówienie oraz odbiera.

	<p>3. Przyjmujący uzyskuje informacje od doręczyciela pizzy o doręczeniu zamówienia.</p> <p>4. Przyjmujący zamówienia odnotowuje fakt doręczenia pizzy w bazie danych.</p>
	<p>Alternatywny:</p> <p>3.1 Doręczyciel pizzy nie zastaje klienta pod adresem zamieszkania.</p> <p>3.2 Doręczyciel pizzy zwraca zamówienie do kuchni.</p> <p>3.3 Doręczyciel pizzy ponownie sprawdza adres zamieszkania klienta.</p> <p>3.4 Powrót do punktu 2. scenariusza głównego.</p>

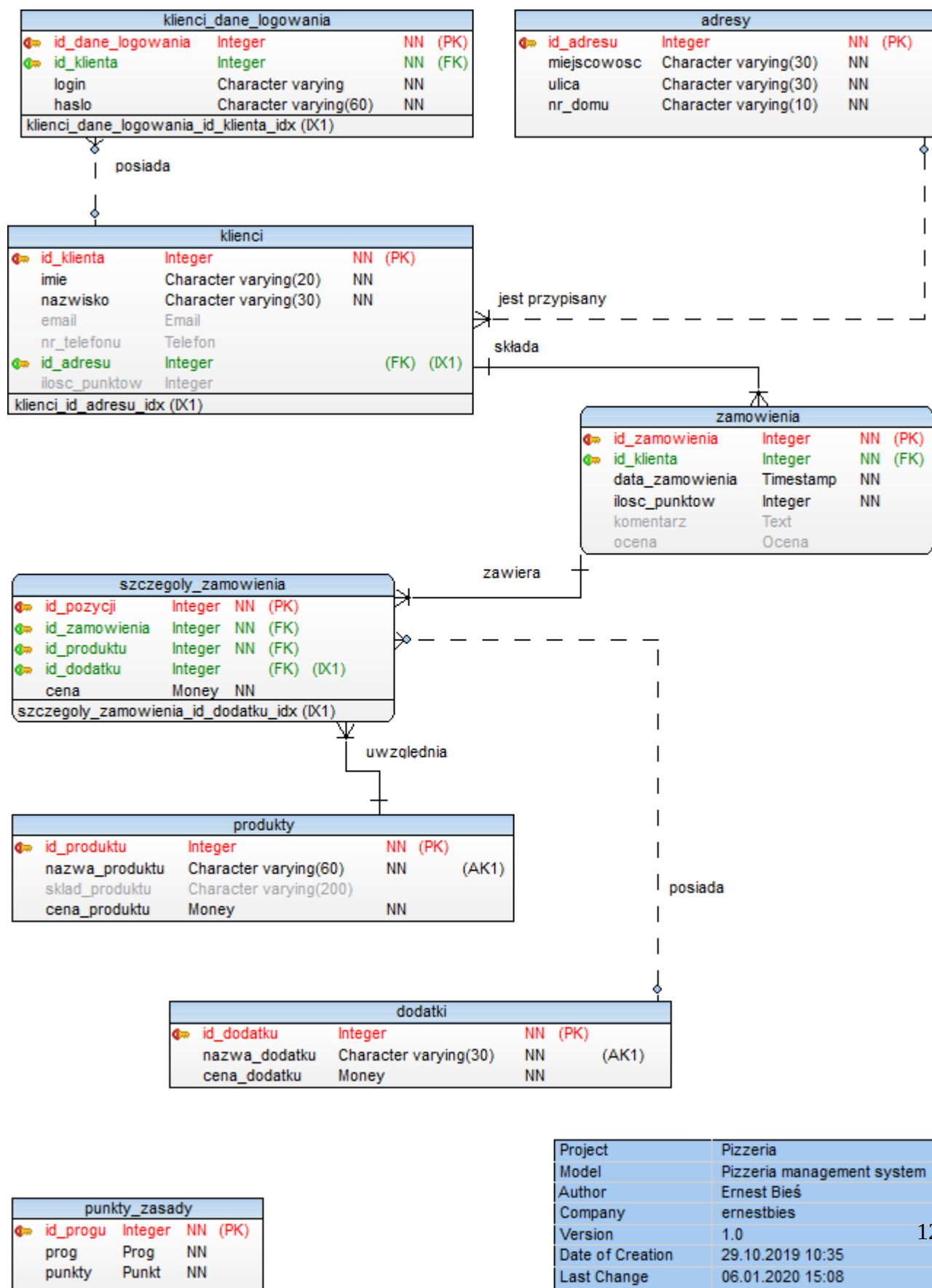
4.1.6. Obsługa systemu zarządzania pizzerią

Nazwa przypadku użycia:	<i>Obsługa systemu zarządzania pizzerią</i>
Aktorzy:	Manager
Warunki wstępne:	Baza danych przygotowana do aktualizacji.
Warunki końcowe:	Zapisanie danych.
Rezultat:	Baza danych dostępna dla użytkowników.
Scenariusz:	
	<p>Główny:</p> <p>1. Manager (Administrator) sprawdza poprawność działania bazy danych.</p> <p>2. Administrator uruchamia optymalizację bazy danych.</p> <p>3. Baza danych jest aktualizowana.</p>
	<p>Alternatywny:</p> <p>1.1 Stwierdzono awarię bazy danych.</p> <p>1.2 Baza danych nie została zaktualizowana.</p> <p>1.3 Administrator uruchamia procedurę odzyskiwania bazy danych.</p> <p>1.4 Powrót do punktu 1. scenariusza głównego</p>

4.2. Przypadki użycia.



5. Diagram ERD



Project	Pizzeria
Model	Pizzeria management system
Author	Ernest Bieś
Company	ernestbies
Version	1.0
Date of Creation	29.10.2019 10:35
Last Change	06.01.2020 15:08

Objaśnienia:

Dziedziny występujące w bazie danych:

Email:

```
CREATE DOMAIN "Email" AS Character varying(100) CONSTRAINT "email_check"  
CHECK (VALUE ~ '^ [A-Za-z0-9._% -]+@[A-Za-z0-9.-]+[.][A-Za-z]+$');
```

Telefon:

```
CREATE DOMAIN "Telefon" AS Character varying(12) CONSTRAINT "telefon_check"  
CHECK (VALUE::text ~* '([(0))([0-9]){9,}'::text OR VALUE::text ~* '([(0))([0-9]){2,}  
([0-9]){7,}'::text);
```

Ocena:

```
CHECK (ocena > 0 AND ocena <= 5);
```

Prog:

```
CHECK (prog > 0::money AND prog <= 500::money);
```

Punkt:

```
CHECK (punkty > 0 AND punkty <= 100);
```

6. Tabele

W bazie danych pizzerii występują następujące tabele:

- klienci



klienci			
 id_klienta	Integer	NN	(PK)
imie	Character varying(20)	NN	
nazwisko	Character varying(30)	NN	
email	Email		
nr_telefonu	Telefon		
 id_adresu	Integer		(FK) (IX1)
ilosc_punktow	Integer		
klienci_id_adresu_idx (IX1)			

Tabela „klienci” zawiera podstawowe informacje na temat klientów w bazie danych, takich jak id klienta (klucz podstawowy), imię, nazwisko, adres e-mail, numer telefonu, id adresu (klucz obcy) oraz łączną ilość punktów, które uzyskał klient.

Kod SQL:

-- Table klienci

```
CREATE TABLE "klienci"(  
  "id_klienta" Integer DEFAULT nextval('klienci_id_klienta_seq') NOT NULL,  
  "imie" Character varying(20) NOT NULL,  
  "nazwisko" Character varying(30) NOT NULL,  
  "email" "Email",  
  "nr_telefonu" "Telefon",  
  "id_adresu" Integer DEFAULT nextval('adresy_id_adresu_seq'),  
  "ilosc_punktow" Integer DEFAULT 0  
)
```

```
WITH (  
  autovacuum_enabled=true);
```

-- Create indexes for table klienci

```
CREATE INDEX "klienci_id_adresu_idx" ON "klienci" ("id_adresu");
```

-- Add keys for table klienci

```
ALTER TABLE "klienci" ADD CONSTRAINT "PK_klienci" PRIMARY KEY ("id_klienta");
```

- klienci_dane_logowania

klienci_dane_logowania			
id_dane_logowania	Integer	NN (PK)	
id_klienta	Integer	NN (FK)	
login	Character varying	NN	
haslo	Character varying(60)	NN	
klienci_dane_logowania_id_klienta_idx (IX1)			==

Tabela „klienci_dane_logowania” zawiera podstawowe informacje na temat danych logowania klientów, takich jak id danych logowania (klucz podstawowy), id klienta (klucz obcy), login (klucz unikatowy) oraz hasło.

Kod SQL:

```
-- Table klienci_dane_logowania
CREATE TABLE "klienci_dane_logowania"(
  "id_dane_logowania" Integer DEFAULT nextval('klienci_dane_logowania_id_dane_logowania_seq')
  NOT NULL,
  "id_klienta" Integer DEFAULT nextval('klienci_id_klienta_seq') NOT NULL,
  "login" Character varying NOT NULL,
  "haslo" Character varying(60) NOT NULL
)
WITH (
  autovacuum_enabled=true)
;
```

```
-- Create indexes for table klienci_dane_logowania
CREATE INDEX "klienci_dane_logowania_id_klienta_idx" ON "klienci_dane_logowania"
("id_klienta");
```

```
-- Add keys for table klienci_dane_logowania
ALTER TABLE "klienci_dane_logowania" ADD CONSTRAINT "PK_klienci_dane_logowania"
PRIMARY KEY ("id_dane_logowania");
```

```
ALTER TABLE "klienci_dane_logowania" ADD CONSTRAINT "login" UNIQUE ("login");
```

- adresy


adresy			
 id_adresu	Integer	NN	(PK)
miescowosc	Character varying(30)	NN	
ulica	Character varying(30)	NN	
nr_domu	Character varying(10)	NN	

Tabela „adresy” zawiera podstawowe informacje na temat miejsca zamieszkania klientów – id adresu (klucz podstawowy), miejscowość, ulica oraz numer domu.

Kod SQL:

-- Table adresy

```
CREATE TABLE "adresy"(  
  "id_adresu" Integer DEFAULT nextval('adresy_id_adresu_seq') NOT NULL,  
  "miescowosc" Character varying(30) NOT NULL,  
  "ulica" Character varying(30) NOT NULL,  
  "nr_domu" Character varying(10) NOT NULL  
)  
WITH (  
  autovacuum_enabled=true)  
;
```

-- Add keys for table adresy

```
ALTER TABLE "adresy" ADD CONSTRAINT "PK_adresy" PRIMARY KEY ("id_adresu");
```

- zamówienia



zamowienia			
 id_zamowienia	Integer	NN	(PK)
 id_klienta	Integer	NN	(FK)
data_zamowienia	Timestamp	NN	
ilosc_punktow	Integer	NN	
komentarz	Text		
ocena	Ocena		

Tabela „zamowienia” zawiera podstawowe informacje na temat zamówienia. Każde zamówienie składa się z poszczególnych szczegółów. Tabela ta określa całość zamówienia (identyfikowana po id zamówienia), która jest powiązana z danym

klientem (identyfikowany po id klienta). Zawiera również takie informacje jak datę złożenia zamówienia, ilość punktów która odpowiada zamówieniu, komentarz oraz ocenę dotyczącą zamówienia, którą może wystawić klient.

Kod SQL:

-- Table zamowienia

```
CREATE TABLE "zamowienia"(  
  "id_zamowienia" Integer DEFAULT nextval('zamowienia_id_zamowienia_seq') NOT NULL,  
  "id_klienta" Integer DEFAULT nextval('klienci_id_klienta_seq') NOT NULL,  
  "data_zamowienia" Timestamp NOT NULL,  
  "ilosc_punktow" Integer DEFAULT 0 NOT NULL,  
  "komentarz" Text DEFAULT NULL,  
  "ocena" Integer DEFAULT NULL  
    CHECK (ocena > 0 AND ocena <= 5)  
)  
WITH (  
  autovacuum_enabled=true)  
;
```

-- Add keys for table zamowienia

```
ALTER TABLE "zamowienia" ADD CONSTRAINT "PK_zamowienia" PRIMARY KEY  
("id_zamowienia");
```

- szczegoly_zamowienia

szczegoly_zamowienia				
🔑 id_pozycji	Integer	NN	(PK)	
🔗 id_zamowienia	Integer	NN	(FK)	
🔗 id_produktu	Integer	NN	(FK)	
🔗 id_dodatku	Integer		(FK)	(IX1)
cena	Money	NN		
szczegoly_zamowienia_id_dodatku_idx (IX1)				

Tabela „szczegoly_zamowienia” zawiera informacje na temat szczegółów zamówienia. Każdy szczegół identyfikowany jest po id jego pozycji (klucz podstawowy), powiązany jest również z danym zamówieniem przez id zamówienia (klucz obcy). Zawiera również informacje o produktach oraz dodatkach, które wchodzi w skład zamówienia (id produktu oraz id dodatku, klucze obce). W tabeli

zawarta jest informacja o cenie danego rekordu (łączna cena produktu oraz dodatku). Zamówienie może składać się z wielu szczegółów.

Kod SQL:

-- Table szczegoly_zamowienia

```
CREATE TABLE "szczegoly_zamowienia"(  
  "id_pozycji" Integer DEFAULT nextval('szczegoly_zamowienia_id_pozycji_seq') NOT NULL,  
  "id_zamowienia" Integer DEFAULT nextval('zamowienia_id_zamowienia_seq') NOT NULL,  
  "id_produktu" Integer DEFAULT nextval('produkty_id_produktu_seq') NOT NULL,  
  "id_dodatku" Integer DEFAULT nextval('dodatki_id_dodatku_seq'),  
  "cena" Money DEFAULT 0::MONEY NOT NULL  
)  
WITH (  
  autovacuum_enabled=true)  
;
```

-- Create indexes for table szczegoly_zamowienia

```
CREATE INDEX "szczegoly_zamowienia_id_dodatku_idx" ON "szczegoly_zamowienia"  
("id_dodatku");
```

-- Add keys for table szczegoly_zamowienia

```
ALTER TABLE "szczegoly_zamowienia" ADD CONSTRAINT "PK_szczegoly_zamowienia" PRIMARY  
KEY ("id_pozycji");
```

- produkty


produkty			
 id_produktu	Integer	NN (PK)	
nazwa_produktu	Character varying(60)	NN	(AK1)
sklad_produktu	Character varying(200)		
cena_produktu	Money	NN	

Tabela „produkty” zawiera informacje na temat produktów. Każdy produkt posiada swoje id produktu (klucz obcy), nazwę produktu (klucz unikatowy), jego skład oraz cenę.

Kod SQL:

-- Table produkty

```
CREATE TABLE "produkty"(  
  "id_produktu" Integer DEFAULT nextval('produkty_id_produktu_seq') NOT NULL,  
  "nazwa_produktu" Character varying(60) NOT NULL,  
  "sklad_produktu" Character varying(200),  
  "cena_produktu" Money NOT NULL  
)  
WITH (  
  autovacuum_enabled=true)  
;
```

-- Add keys for table produkty

```
ALTER TABLE "produkty" ADD CONSTRAINT "PK_produkty" PRIMARY KEY ("id_produktu");  
ALTER TABLE "produkty" ADD CONSTRAINT "nazwa_produktu" UNIQUE ("nazwa_produktu");
```

- dodatki


dodatki				
 id_dodatku	Integer	NN	(PK)	
nazwa_dodatku	Character varying(30)	NN		(AK1)
cena_dodatku	Money	NN		

Tabela „dodatki” zawiera informacje na temat dodatków. Każdy dodatek posiada swoje id dodatku (klucz obcy), nazwę dodatku (klucz unikatowy) oraz cenę.

Kod SQL:

-- Table dodatki

```
CREATE TABLE "dodatki"(  
  "id_dodatku" Integer DEFAULT nextval('dodatki_id_dodatku_seq') NOT NULL,  
  "nazwa_dodatku" Character varying(30) NOT NULL,  
  "cena_dodatku" Money NOT NULL  
)  
WITH (  
  autovacuum_enabled=true);
```

```
-- Add keys for table dodatki
```

```
ALTER TABLE "dodatki" ADD CONSTRAINT "PK_dodatki" PRIMARY KEY ("id_dodatku");
```

```
ALTER TABLE "dodatki" ADD CONSTRAINT "nazwa_dodatku" UNIQUE ("nazwa_dodatku");
```

- punkty_zasady

punkty_zasady			
	id_progu	Integer	NN (PK)
	prog	Prog	NN
	punkty	Punkt	NN

Tabela „punkty_zasady” zawiera informacje na temat zasad przyznawania punktów klientom. Każdy próg identyfikowany jest po jego id progu (klucz podstawowy), wartości prog (dziedzina Prog) oraz ilości punktów (dziedzina Punkt).

Kod SQL:

```
-- Table punkty_zasady
```

```
CREATE TABLE "punkty_zasady" (
```

```
  "id_progu" Integer NOT NULL,
```

```
  "prog" Money NOT NULL
```

```
    CHECK (prog > 0::money AND prog <= 500::money),
```

```
  "punkty" Integer NOT NULL
```

```
    CHECK (punkty > 0 AND punkty <= 100)
```

```
)
```

```
WITH (
```

```
  autovacuum_enabled=true);
```

```
-- Add keys for table punkty_zasady
```

```
ALTER TABLE "punkty_zasady" ADD CONSTRAINT "PK_punkty_zasady" PRIMARY KEY  
("id_progu");
```

Relacje pomiędzy tabelami (kod SQL):

```
ALTER TABLE "klienci" ADD CONSTRAINT "jest przypisany" FOREIGN KEY ("id_adresu")  
REFERENCES "adresy" ("id_adresu") ON DELETE NO ACTION ON UPDATE NO ACTION;
```

```
ALTER TABLE "zamowienia" ADD CONSTRAINT "składa" FOREIGN KEY ("id_klienta")  
REFERENCES "klienci" ("id_klienta") ON DELETE NO ACTION ON UPDATE NO ACTION;
```

```
ALTER TABLE "szczegoly_zamowienia" ADD CONSTRAINT "uwzględnia" FOREIGN KEY  
("id_produktu") REFERENCES "produkty" ("id_produktu") ON DELETE NO ACTION ON UPDATE  
NO ACTION;
```

```
ALTER TABLE "szczegoly_zamowienia" ADD CONSTRAINT "posiada" FOREIGN KEY  
("id_dodatku") REFERENCES "dodatki" ("id_dodatku") ON DELETE NO ACTION ON UPDATE NO  
ACTION;
```

```
ALTER TABLE "szczegoly_zamowienia" ADD CONSTRAINT "zawiera" FOREIGN KEY  
("id_zamowienia") REFERENCES "zamowienia" ("id_zamowienia") ON DELETE NO ACTION ON  
UPDATE NO ACTION;
```

```
ALTER TABLE "klienci_dane_logowania" ADD CONSTRAINT "posiada" FOREIGN KEY  
("id_klienta") REFERENCES "klienci" ("id_klienta") ON DELETE NO ACTION ON UPDATE NO  
ACTION;
```

```
ALTER SEQUENCE "klienci_id_klienta_seq" OWNED BY "klienci"."id_klienta";
```

```
ALTER SEQUENCE "szczegoly_zamowienia_id_pozycji_seq" OWNED BY  
"szczegoly_zamowienia"."id_pozycji";
```

```
ALTER SEQUENCE "zamowienia_id_zamowienia_seq" OWNED BY "zamowienia"."id_zamowienia";
```

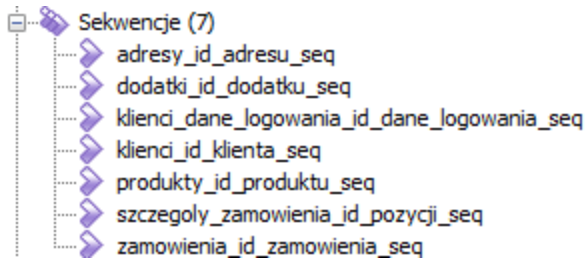
```
ALTER SEQUENCE "adresy_id_adresu_seq" OWNED BY "adresy"."id_adresu";
```

```
ALTER SEQUENCE "produkty_id_produktu_seq" OWNED BY "produkty"."id_produktu";
```

```
ALTER SEQUENCE "dodatki_id_dodatku_seq" OWNED BY "dodatki"."id_dodatku";
```

7. Sekwencje

Lista sekwencji występujących w bazie danych pizzerii:



Kod SQL odpowiedzialny za stworzenie sekwencji:

```
CREATE SEQUENCE "klienci_id_klienta_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

```
CREATE SEQUENCE "szczegoly_zamowienia_id_pozycji_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

```
CREATE SEQUENCE "zamowienia_id_zamowienia_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

```
CREATE SEQUENCE "adresy_id_adresu_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

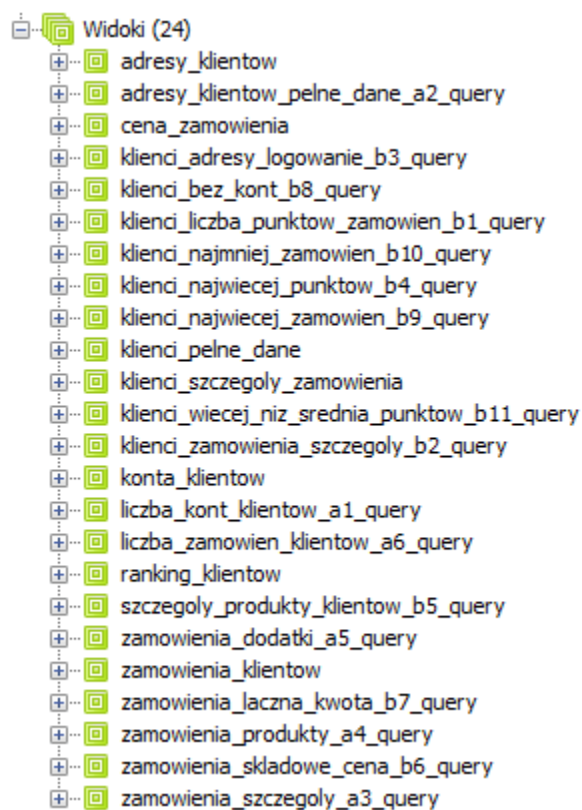
```
CREATE SEQUENCE "produkty_id_produkty_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

```
CREATE SEQUENCE "dodatki_id_dodatku_seq"  
INCREMENT BY 1  
NO MAXVALUE  
NO MINVALUE  
CACHE 1;
```

```
CREATE SEQUENCE "klienci_dane_logowania_id_dane_logowania_seq"  
INCREMENT BY 1  
NO MAXVALUE  
MINVALUE 1  
CACHE 1;
```

8. Widoki

Pełna lista widoków występujących w bazie danych:



8.1 Adresy klientów

Widok zawierający informacje na temat pełnych danych klientów oraz ich adresów.

Kod SQL:

```
CREATE VIEW "adresy_klientow" AS
SELECT "id_klienta", "imie", "nazwisko", "adresy"."id_adresu", "miestowosc", "ulica", "nr_domu"
FROM "klienci", "adresy"
WHERE klienci.id_adresu = adresy.id_adresu;
```

8.2 Zamówienia klientów

Widok zawierający informacje na temat pełnych danych klienta oraz zamówień klientów.

Kod SQL:

```
CREATE VIEW "zamowienia_klientow" AS
SELECT "klienci"."id_klienta", "imie", "nazwisko", "id_zamowienia", "data_zamowienia",
"zamowienia"."ilosc_punktow", "komentarz", "ocena"
FROM "klienci", "zamowienia"
WHERE klienci.id_klienta = zamowienia.id_klienta;
```

8.3 Pełne dane klientów

Widok zawierający informacje na temat pełnych danych klienta wraz z danymi logowania oraz miejscem zamieszkania.

Kod SQL:

```
CREATE VIEW "klienci_pelne_dane" AS
SELECT "klienci"."id_klienta", "imie", "nazwisko", "email", "nr_telefonu", "id_adresu",
"ilosc_punktow", "login", "haslo"
FROM "klienci", "klienci_dane_logowania"
WHERE klienci.id_klienta = klienci_dane_logowania.id_klienta;
```


8.4 Konta klientów

Widok zawierający informacje na temat kont klientów.

Kod SQL:

```
CREATE VIEW "konta_klientow" AS
SELECT "klienci"."id_klienta", "login", "haslo"
FROM "klienci", "klienci_dane_logowania"
WHERE klienci.id_klienta = klienci_dane_logowania.id_klienta;
```

8.5 Cena zamówienia

Widok zawierający informacje na temat łącznej kwoty zamówień.

Kod SQL:

```
CREATE VIEW "cena_zamowienia" AS
SELECT zamowienia.id_zamowienia, SUM(cena) AS wartosc_laczna,
zamowienia.data_zamowienia
FROM zamowienia, szczegoly_zamowienia
WHERE zamowienia.id_zamowienia = szczegoly_zamowienia.id_zamowienia
GROUP BY zamowienia.id_zamowienia;
```

8.5 Ranking klientów

Widok reprezentujący ranking klientów pod względem uzyskanych punktów.

Kod SQL:

```
CREATE VIEW "ranking_klientow" AS
SELECT "id_klienta", "imie", "nazwisko", "email", "ilosc_punktow"
FROM "klienci"
ORDER BY "ilosc_punktow" DESC;
```

8.6 Szczegóły zamówienia

Widok zawierający informacje na temat szczegółów zamówień klientów.

Kod SQL:

```
CREATE VIEW "klienci_szczegoly_zamowienia" AS
SELECT "klienci"."id_klienta", "szczegoly_zamowienia"."id_zamowienia", "id_produktu",
"id_dodatku", "cena"
FROM "klienci", "zamowienia", "szczegoly_zamowienia"
WHERE klienci.id_klienta = zamowienia.id_klienta AND zamowienia.id_zamowienia =
szczegoly_zamowienia.id_zamowienia;
```

8.7 Kwerendy zapisane w widokach.

Kwerendy proste:

A1) Dane o kliencie wraz z liczbą kont.

```
CREATE VIEW liczba_kont_klientow_A1_query AS (
SELECT id_klienta, imie, nazwisko, email, COUNT(kdl.id_klienta)::INTEGER AS liczba_kont_klienta
FROM klienci NATURAL LEFT JOIN klienci_dane_logowania kdl GROUP BY id_klienta ORDER BY
liczba_kont_klienta DESC);
```

A2) Klienci z wprowadzonymi pełnymi danymi.

```
CREATE VIEW adresy_klientow_pelne_dane_A2_query AS (
SELECT id_klienta, imie, nazwisko, miejscowosc, ulica, email, nr_telefonu
FROM klienci NATURAL JOIN adresy WHERE email IS NOT NULL AND nr_telefonu IS NOT NULL);
```

A3) Zamówienia wraz ze szczegółami.

```
CREATE VIEW zamowienia_szczegoly_A3_query AS(
SELECT id_zamowienia, id_klienta, data_zamowienia, COUNT(szczegoly_zamowienia) AS
liczba_pozycji_zamowienia
FROM zamowienia NATURAL JOIN szczegoly_zamowienia GROUP BY id_zamowienia, id_klienta,
data_zamowienia ORDER BY id_zamowienia ASC);
```

A4) Produkty w zamówieniach.

```
CREATE VIEW zamowienia_produkty_A4_query AS(
SELECT id_zamowienia, id_produktu, nazwa_produktu, sklad_produktu, cena_produktu
FROM szczegoly_zamowienia NATURAL JOIN produkty);
```

A5) Dodatki w zamówieniach.

```
CREATE VIEW zamowienia_dodatki_A5_query AS(  
SELECT id_zamowienia, szczegoly_zamowienia.id_dodatku, nazwa_dodatku, cena_dodatku  
FROM szczegoly_zamowienia, dodatki WHERE dodatki.id_dodatku =  
szczegoly_zamowienia.id_dodatku  
ORDER BY id_zamowienia);
```

A6) Liczba zamówień klienta.

```
CREATE VIEW liczba_zamowien_klientow_A6_query AS(  
SELECT klienci.id_klienta, imie, nazwisko, COUNT(id_zamowienia) AS liczba_zamowien FROM  
zamowienia INNER JOIN klienci ON klienci.id_klienta = zamowienia.id_klienta GROUP BY  
klienci.id_klienta);
```

Kwerendy złożone:

B1) Dane na temat klientów, ilości punktów oraz liczby złożonych zamówień.

```
CREATE VIEW klienci_liczba_punktow_zamowien_B1_query AS(  
SELECT klienci.id_klienta, imie, nazwisko, klienci.ilosc_punktow, COUNT(id_zamowienia) AS  
liczba_zamowien FROM zamowienia, klienci WHERE klienci.id_klienta = zamowienia.id_klienta  
GROUP BY klienci.id_klienta ORDER BY ilosc_punktow DESC);
```

B2) Informacje na temat danych klientów, ich zamówień oraz szczegółów zamówień wchodzących w skład zamówienia.

```
CREATE VIEW klienci_zamowienia_szczegoly_B2_query AS(  
SELECT k.id_klienta, k.imie, k.nazwisko, sz.id_zamowienia, p.id_produktu, p.nazwa_produktu,  
d.id_dodatku, d.nazwa_dodatku, sz.cena  
FROM klienci k, zamowienia z, szczegoly_zamowienia sz NATURAL LEFT JOIN produkty p  
NATURAL LEFT JOIN dodatki d  
WHERE k.id_klienta = z.id_klienta AND z.id_zamowienia = sz.id_zamowienia);
```

B3) Informacje na temat klientów, ich adresów oraz danych logowania.

```
CREATE VIEW klienci_adresy_logowanie_B3_query AS(  
SELECT k.id_klienta, k.imie, k.nazwisko, kd.login, kd.haslo, a.id_adresu, a.miejscowosc, a.ulica,  
a.nr_domu FROM klienci k NATURAL JOIN adresy a NATURAL LEFT JOIN klienci_dane_logowania  
kd ORDER BY id_klienta);
```

B4) Klienci o największej liczbie punktów.

```
CREATE VIEW klienci_najwiecej_punktow_B4_query AS(  
SELECT k.id_klienta, k.imie, k.nazwisko, k.ilosc_punktow, kd.login, kd.haslo, a.id_adresu,  
a.miejscowosc, a.ulica, a.nr_domu  
FROM klienci k NATURAL JOIN klienci_dane_logowania kd INNER JOIN adresy a ON a.id_adresu =  
k.id_adresu  
WHERE k.ilosc_punktow = (SELECT MAX(ilosc_punktow) FROM klienci));
```

B5) Zamówienia wraz ze szczegółami, produktami wraz z pełnymi danymi produktów oraz pełnymi danymi klientów wraz z adresami klientów.

```
CREATE VIEW szczegoly_produkty_klientow_B5_query AS(  
SELECT k.id_klienta, k.imie, k.nazwisko, a.id_adresu, a.miejscowosc, a.ulica, a.nr_domu,  
sz.id_zamowienia, p.id_produktu, p.nazwa_produktu, p.sklad_produktu, d.id_dodatku,  
d.nazwa_dodatku, sz.cena  
FROM klienci k, adresy a, zamowienia z, szczegoly_zamowienia sz NATURAL LEFT JOIN produkty  
p NATURAL LEFT JOIN dodatki d  
WHERE k.id_adresu = a.id_adresu AND k.id_klienta = z.id_klienta AND z.id_zamowienia =  
sz.id_zamowienia);
```

B6) Zamówienie wraz z ilością składowych oraz ceną.

```
CREATE VIEW zamowienia_skladowe_cena_B6_query AS(  
SELECT z.id_zamowienia, COUNT(z.id_zamowienia) AS ilosc_skladowych_zamowienia, SUM(cena)  
AS kwota_zamowienia  
FROM zamowienia z INNER JOIN szczegoly_zamowienia sz ON z.id_zamowienia =  
sz.id_zamowienia  
GROUP BY 1 ORDER BY 1);
```

B7) Zamówienie z łączną kwotą.

```
CREATE VIEW zamowienia_laczna_kwota_B7_query AS(  
SELECT z.id_zamowienia AS zamowienie,  
(SELECT SUM(sz.cena) FROM szczegoly_zamowienia sz WHERE sz.id_zamowienia =  
z.id_zamowienia)  
AS laczna_cena FROM zamowienia z GROUP BY z.id_zamowienia);
```

B8) Klienci bez konta.

```
CREATE VIEW klienci_bez_kont_B8_query AS(  
SELECT k.id_klienta, (imie || ' ' || nazwisko)::varchar AS dane_osobowe,  
(SELECT id_adresu FROM adresy a WHERE a.id_adresu = k.id_adresu),
```

```
(SELECT miejscowosc FROM adresy a WHERE a.id_adresu = k.id_adresu),  
(SELECT ulica FROM adresy a WHERE a.id_adresu = k.id_adresu),  
TRUE AS brak_konta FROM klienci k  
WHERE id_klienta NOT IN (SELECT id_klienta FROM klienci_dane_logowania));
```

B9) Klienci z największą liczbą zamówień.

```
SELECT klienci.id_klienta, imie, nazwisko, klienci.ilosc_punktow, COUNT(id_zamowienia) AS  
liczba_zamowien FROM zamowienia, klienci  
WHERE klienci.id_klienta = zamowienia.id_klienta GROUP BY klienci.id_klienta HAVING  
COUNT(id_zamowienia) =  
(SELECT MAX(ilosc) FROM (SELECT COUNT(id_zamowienia) AS ilosc FROM zamowienia GROUP  
BY id_klienta) AS c)  
ORDER BY ilosc_punktow DESC);
```

B10) Klienci z najmniejszą liczbą zamówień.

```
CREATE VIEW klienci_najmniej_zamowien_B10_query AS(  
SELECT k.id_klienta, (imie || ' ' || nazwisko)::varchar AS dane_osobowe, COUNT(id_zamowienia)  
AS ilosc_zamowien  
FROM klienci k NATURAL JOIN zamowienia z GROUP BY k.id_klienta HAVING  
COUNT(id_zamowienia) =  
(SELECT MIN(ilosc) FROM (SELECT COUNT(id_zamowienia) AS ilosc FROM zamowienia GROUP  
BY id_klienta) AS c));
```

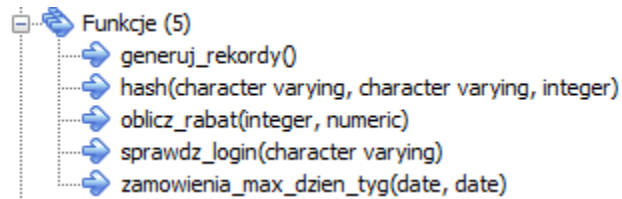
B11) Klienci z liczbą punktów większą od średniej liczby punktów klientów.

```
CREATE VIEW klienci_wiecej_niz_srednia_punktow_B11_query AS(  
SELECT k.id_klienta, k.imie, k.nazwisko, kd.login, kd.haslo, a.id_adresu, a.miejscowosc, a.ulica,  
a.nr_domu, k.ilosc_punktow  
FROM klienci k NATURAL JOIN klienci_dane_logowania kd INNER JOIN adresy a ON a.id_adresu =  
k.id_adresu  
WHERE k.ilosc_punktow >= (SELECT AVG(ilosc_punktow) FROM klienci));
```

9. Funkcje

Funkcje zostały napisane w języku proceduralnym **PL/pgSQL**. Funkcje realizują czynności/zadania, których realizacja nie jest możliwa wyłącznie z wykorzystaniem języka SQL.

Lista funkcji występujących w bazie danych pizzerii:



9.1 Funkcja generuj_rekordy()

Funkcja powoduje wygenerowanie przykładowych rekordów w bazie danych.

Funkcja wprowadza dane do wszystkich tabel istniejących w bazie.

Kod SQL funkcji:

```
CREATE OR REPLACE FUNCTION generuj_rekordy()
RETURNS BOOLEAN AS
$GENERATOR$
DECLARE
  i INTEGER := 0;
  imie VARCHAR [10];
  nazwisko VARCHAR [10];
  miejscowosc VARCHAR [10];
  ulica VARCHAR [10];
  losowa1 INTEGER;
  losowa2 INTEGER;
  losowa3 INTEGER;
  rec RECORD;
  numer VARCHAR := '';
  losowa_data TIMESTAMP WITHOUT TIME ZONE;
BEGIN
  DELETE FROM zamowienia;
  DELETE FROM klienci_dane_logowania;
  DELETE FROM klienci;
  DELETE FROM adresy;
  DELETE FROM punkty_zasady;
  DELETE FROM dodatki;
  DELETE FROM produkty;
```

```

    imie := ARRAY['Ernest', 'Grzegorz', 'Anna', 'Tomasz', 'Janusz', 'Łucja', 'Karolina', 'Józef', 'Elżbieta',
'Jan'];
    nazwisko := ARRAY['Bieś', 'Ptak', 'Sowa', 'Nowak', 'Kowalski', 'Dąb', 'Maj', 'Wiśniewski', 'Tokarczuk',
'Zięba'];
    miejscowosc := ARRAY['Dąbrowa Tarnowska', 'Tarnów', 'Żabno', 'Bolesław', 'Nieczajna Górna',
'Szczucin', 'Radgoszcz', 'Lisia Góra', 'Breń', 'Sieradza'];
    ulica := ARRAY['Owocowa', 'Leśna', 'Dąbrowskiego', 'Tarnowska', 'Żabieńska', 'Krajowa', 'Brzozowa',
'Mała', 'Duża', 'Kwadratowa'];
    LOOP
        losowa1 := (SELECT (random() * 9 + 1)::INTEGER);
        losowa2 := (SELECT (random() * 9 + 1)::INTEGER);
        losowa3 := (SELECT (random() * 100)::INTEGER);
        INSERT INTO adresy (miejscowosc, ulica, nr_domu) VALUES (miejscowosc[losowa1],
ulica[losowa2], losowa3::VARCHAR);
        IF i = 100 THEN
            EXIT;
        END IF;
        i := i + 1;
    END LOOP;

    FOR rec IN (SELECT id_adresu FROM adresy) LOOP
        numer := '(0)';
        i := 0;
        LOOP
            numer := (numer || (SELECT (random() * 9)::INTEGER::VARCHAR));
            IF i = 8 THEN
                EXIT;
            END IF;
            i := i + 1;
        END LOOP;
        losowa1 := (SELECT (random() * 9 + 1)::INTEGER);
        losowa2 := (SELECT (random() * 9 + 1)::INTEGER);
        losowa3 := (SELECT (random() * 100)::INTEGER);

        IF (SELECT (random() * 2)::INTEGER) = 0 THEN
            numer := NULL;

```

```

END IF;

INSERT INTO klienci (imie, nazwisko, email, nr_telefonu, id_adresu) VALUES (imie[losowa1],
nazwisko[losowa1], (losowa3::VARCHAR || '@email.com'), numer, rec.id_adresu);
END LOOP;

IF (SELECT COUNT(*) FROM punkty_zasady) = 0 THEN
    INSERT INTO punkty_zasady VALUES (1, 50::MONEY, 10);
    INSERT INTO punkty_zasady VALUES (2, 100::MONEY, 25);
    INSERT INTO punkty_zasady VALUES (3, 150::MONEY, 40);
    INSERT INTO punkty_zasady VALUES (4, 200::MONEY, 80);
    INSERT INTO punkty_zasady VALUES (5, 250::MONEY, 100);
END IF;

FOR rec IN (SELECT * FROM klienci) LOOP
    IF (SELECT (random() * 3)::INTEGER) = 1 THEN
        INSERT INTO klienci_dane_logowania (id_klienta, login, haslo) VALUES (rec.id_klienta, (rec.imie
|| rec.id_klienta::VARCHAR), md5(rec.nazwisko));
    END IF;
END LOOP;

IF (SELECT COUNT(*) FROM produkty) = 0 THEN
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza
wegetariańska', 'sałata, pieczarka, ogórki, pomidory', 25);
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza
wiejska', 'kiełbasa wiejska, boczek, ogórki, cebula', 30);
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza owoce
morza', 'krewetki, kukurydza, pomidory', 40);
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza
margaritta', 'sos pomidorowy, pieczarki, żółty ser', 15);
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza szynka',
'szynka, pieczarki, żółty ser', 20);
    INSERT INTO produkty (nazwa_produkту, sklad_produkту, cena_produkту) VALUES ('Pizza
firmowa', NULL, 50);
END IF;

IF (SELECT COUNT(*) FROM dodatki) = 0 THEN
    INSERT INTO dodatki (nazwa_dodatku, cena_dodatku) VALUES ('keczup', 5);

```



```

INSERT INTO dodatki (nazwa_dodatku, cena_dodatku) VALUES ('oregano', 3);
INSERT INTO dodatki (nazwa_dodatku, cena_dodatku) VALUES ('pietruszka', 2);
INSERT INTO dodatki (nazwa_dodatku, cena_dodatku) VALUES ('majonez', 5);
END IF;

FOR rec IN (SELECT id_klienta FROM klienci) LOOP
    losowa1 := (SELECT (random() * 10 + 1)::INTEGER);
    i := 0;
    LOOP
        losowa_data := (SELECT '2019-01-01'::TIMESTAMP WITHOUT TIME ZONE + (random() * ('2019-
12-30'::TIMESTAMP WITHOUT TIME ZONE - '2019-01-01'::TIMESTAMP WITHOUT TIME ZONE)));
        INSERT INTO zamowienia (id_klienta, data_zamowienia) VALUES (rec.id_klienta, losowa_data);
        IF i = losowa1 THEN
            EXIT;
        END IF;
        i := i + 1;
    END LOOP;
END LOOP;

FOR rec IN (SELECT id_zamowienia FROM zamowienia) LOOP
    losowa1 := (SELECT (random() * 3 + 1)::INTEGER);
    i := 0;
    LOOP
        losowa2 := (SELECT id_produktu FROM produkty ORDER BY random() LIMIT 1);
        losowa3 := (SELECT id_dodatku FROM dodatki ORDER BY random() LIMIT 1);
        IF (SELECT (random() * 2)::INTEGER) = 0 THEN
            losowa3 := NULL;
        END IF;
        INSERT INTO szczegoly_zamowienia (id_zamowienia, id_produktu, id_dodatku) VALUES
(rec.id_zamowienia, losowa2, losowa3);
        IF i = losowa1 THEN
            EXIT;
        END IF;
        i := i + 1;
    END LOOP;
END LOOP;
RETURN TRUE;

```

```
END;
$GENERATOR$
LANGUAGE 'plpgsql';
```

Dane wyjściowe:

```
SELECT generuj_rekordy();
```

Okno wyjściowe	
Dane wyjściowe	
	generuj_rekordy boolean
1	t

9.2 Funkcja hash(character varying, character varying, integer)

Funkcja korzysta z rozszerzenia pgcrypto. Pozwala na zahashowanie dowolnego ciągu znakowego, dowolnym algorytmem hashującym. Przyjmuje trzy parametry.

- 1) ciąg znaków, który ma zostać zahashowany (typ character varying)
- 2) sposób hashowania (md5, crypt-md5, crypt-bf, crypt-xdes, crypt-des) (typ character varying)
- 3) liczba iteracji (typ integer)

Kod SQL funkcji:

```
CREATE OR REPLACE FUNCTION hash(haslo VARCHAR, sposob_hashowania VARCHAR,
liczba_iteracji INTEGER)
RETURNS TEXT AS
$HASH$
BEGIN
    CREATE EXTENSION IF NOT EXISTS pgcrypto;
    IF $2 = 'md5' THEN
        RETURN md5($1);
    ELSIF $2 = 'crypt-md5' THEN
        RETURN crypt($1, gen_salt('md5'));
    ELSIF $2 = 'crypt-bf' THEN
        RETURN crypt($1, gen_salt('bf', $3));
    ELSIF $2 = 'crypt-xdes' THEN
        RETURN crypt($1, gen_salt('xdes'));
    ELSIF $2 = 'crypt-des' THEN
```

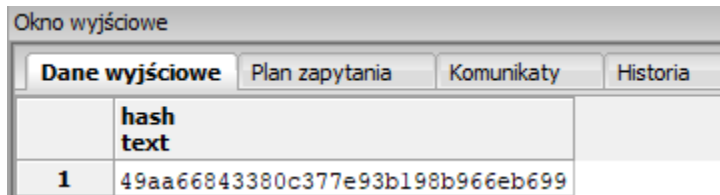
```

RETURN crypt($1, gen_salt('des'));
ELSE
    RAISE NOTICE 'Podano nieprawidłowy parametr!';
    RETURN NULL;
END IF;
END;
$HASH$
LANGUAGE 'plpgsql';

```

Dane wyjściowe:

```
SELECT hash('alamakota','md5',0);
```



	hash	text
1	49aa66843380c377e93b198b966eb699	

9.3 Funkcja oblicz_rabat(integer, numeric)

Funkcja pozwalająca na obliczenie rabatu. Przyjmuje ona dwa parametry:

- 1) id_klienta (typ integer)
- 2) wartość zamówienia (typ numeric)

Na podstawie wartości zamówienia obliczany jest rabat. Funkcja zwraca tabelę składającą się z dwóch kolumn – rabatu oraz obliczonej nowej wartości zamówienia po uzyskaniu rabatu.

Zasady obliczania rabatu są następujące:

- 500 punktów - 2000 punktów – 5% rabatu
- 2001 punktów - 5000 punktów – 10% rabatu
- >5000 punktów - 10 zł + 5% rabatu

Kod SQL funkcji:

```

CREATE OR REPLACE FUNCTION oblicz_rabat(id_klienta INTEGER, wartosc_zamowienia
NUMERIC)
RETURNS TABLE

```

```

(
    rabat VARCHAR,
    zamowienie_z_rabatem MONEY
)AS
$RABAT$
DECLARE
    punkty_klienta INTEGER;
    r VARCHAR;
    w NUMERIC;
BEGIN

    RAISE NOTICE 'Funkcja obliczająca rabat w zależności od punktów klienta
    Zasady przyznawania rabatu: 500 punktów - 2000 punktów - 5 procent rabatu
    2001 punktów - 5000 punktów - 10 procent rabatu
    >5000 punktów - 10 zł + 5 procent rabatu';

    SELECT ilosc_punktow INTO punkty_klienta FROM klienci k WHERE k.id_klienta = $1;

    IF punkty_klienta < 500 THEN
        r := 'brak rabatu';
        w := $2;
    ELSIF punkty_klienta >= 500 AND punkty_klienta <= 2000 THEN
        r := '5% rabatu';
        w := $2 - ($2 * 0.05);
    ELSIF punkty_klienta >= 2001 AND punkty_klienta <= 5000 THEN
        r := '10% rabatu';
        w := $2 - ($2 * 0.1);
    ELSIF punkty_klienta > 5000 THEN
        r := '10zł + 5% rabatu';
        w := $1 - 10 - (($1 - 10)*0.05);
    END IF;
    rabat := r;
    zamowienie_z_rabatem := w::MONEY;
    RETURN NEXT;
END;
$RABAT$
LANGUAGE 'plpgsql';

```

Dane wyjściowe:

```
SELECT * FROM oblicz_rabat(102, 2000);
```

Okno wyjściowe		
Dane wyjściowe		
	rabat character varying	zamowienie_z_rabatem money
1	5% rabatu	1 900,00 zł

9.4 Funkcja sprawdz_login(character varying)

Funkcja pozwalająca na sprawdzenie poprawności nazwy użytkownika.

Korzysta ona z wyrażenia regularnego '[A-Za-z0-9]+\$', które sprawdza czy nazwa użytkownika składa się tylko z liter dowolnej wielkości oraz cyfr. Funkcja przyjmuje jeden parametr – nazwę użytkownika (login) typu varchar. Funkcja zwraca prawdę lub fałsz (typ boolean).

Kod SQL funkcji:

```
CREATE OR REPLACE FUNCTION sprawdz_login(login VARCHAR)
RETURNS BOOLEAN AS
$LOGIN$
BEGIN
    IF login ~ '[A-Za-z0-9]+$' THEN
        RAISE NOTICE 'Nazwa użytkownika jest poprawna';
        RETURN TRUE;
    ELSE
        RAISE NOTICE 'Wprowadzono niepoprawną nazwę użytkownika!';
        RETURN FALSE;
    END IF;
END;
$LOGIN$
LANGUAGE 'plpgsql';
```

Dane wyjściowe:

Przykładowa poprawna nazwa użytkownika zawierająca tylko litery oraz cyfry.

SELECT sprawdz_login('ernest123');

Okno wyjściowe	
Dane wyjściowe Plan zapytania Komunikaty Historia	
	sprawdz_login text
1	true

Przykładowa niepoprawna nazwa użytkownika zawierająca znak „-”.

SELECT sprawdz_login('ernest123-');

Okno wyjściowe	
Dane wyjściowe Plan zapytania Komunikaty Historia	
	sprawdz_login text
1	false

9.5 Funkcja zamówienia_max_dzien_tyg(date, date)

Funkcja zwraca nazwę dnia tygodnia (poniedziałek, wtorek, środa, czwartek, piątek, sobota, niedziela), w którym było najwięcej zamówień z podanego okresu.

Przyjmuje ona dwa parametry:

- 1) datę początkową (typ DATE)
- 2) datę końcową (typ DATE)

Kod SQL funkcji:

```
CREATE OR REPLACE FUNCTION zamówienia_max_dzien_tyg(data_początkowa DATE,  
data_koncowa DATE)
```

```
RETURNS VARCHAR AS
```

```
$ZAMOWIENIA_MAX_DZIEN_TYG$
```

```
DECLARE
```

```
    daty DATE[];
```

```
    dni_tygodnia INTEGER[7];
```

```
    dni_tygodnia_nazwy VARCHAR[7];
```

```
    dzien SMALLINT;
```

```

rozmiar_daty INTEGER;
i INTEGER := 0;
max INTEGER := 0;
ilosc_dat INTEGER := 0;
BEGIN
    IF data_poczatkowa > data_koncowa THEN
        RAISE NOTICE 'Błędne wywołanie! Data końcowa jest wcześniejsza od daty początkowej!';
        RETURN NULL;
    END IF;

    dni_tygodnia := ARRAY[0,0,0,0,0,0,0];
    dni_tygodnia_nazwy :=
ARRAY['poniedziałek','wtorek','środa','czwartek','piątek','sobota','niedziela'];

    ilosc_dat := (SELECT COUNT(data_zamowienia) FROM zamowienia WHERE data_zamowienia
>= data_poczatkowa AND data_zamowienia <= data_koncowa);
    IF ilosc_dat = 0 THEN
        RAISE NOTICE 'Brak zamówień w tym przedziale!';
        RETURN NULL;
    END IF;

    daty := ARRAY(SELECT data_zamowienia FROM zamowienia WHERE data_zamowienia >=
data_poczatkowa AND data_zamowienia <= data_koncowa);
    rozmiar_daty := (SELECT array_length(daty, 1));

    LOOP
        i := i + 1;
        dzien = EXTRACT(dow FROM daty[i]);
        dni_tygodnia[dzien] := dni_tygodnia[dzien] + 1;
        RAISE NOTICE 'Dzień: %', dni_tygodnia;
        IF i = rozmiar_daty THEN
            EXIT;
        END IF;
    END LOOP;

    i := 0;
    max := 0;
    dzien := 0;

```

LOOP

```
i := i + 1;  
IF dni_tygodnia[i] > max THEN  
    dzien := i;  
    max := dni_tygodnia[i];  
END IF;
```

```
IF i = 7 THEN  
    EXIT;  
END IF;
```

END LOOP;

```
RETURN dni_tygodnia_nazwy[dzien];
```

END;

\$ZAMOWIENIA_MAX_DZIEN_TYG\$

LANGUAGE 'plpgsql';

Dane wyjściowe:

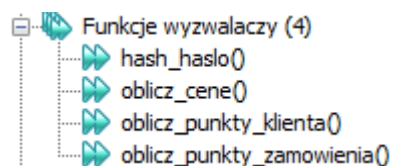
SELECT zamowienia_max_dzien_tyg('2019-01-01','2019-01-30');

Okno wyjściowe	
Dane wyjściowe Plan zapytania Komunikaty Historia	
	zamowienia_max_dzien_tyg character varying
1	wtorek

Oznacza to że w okresie od '2019-01-01' do '2019-01-30' najwięcej zamówień było we wtorki.

10. Wyzwalacze

Lista funkcji wyzwalaczy występujących w bazie danych:



10.1 Funkcja wyzwalacza oblicz_cene()

Funkcja wyzwalacza oblicz_cene() wywoływana jest na tabeli szczegoly_zamowienia przed operacją INSERT lub UPDATE. Oblicza ona cenę danego rekordu (cena produktu + cena dodatku).

Kod SQL:

```
CREATE OR REPLACE FUNCTION oblicz_cene()
  RETURNS TRIGGER AS
$$
DECLARE
  cena_p MONEY;
  cena_d MONEY;
BEGIN
  cena_p = (SELECT cena_produkту FROM produkty WHERE produkty.id_produkту =
NEW.id_produkту);
  cena_d = (SELECT cena_dodatku FROM dodatki WHERE dodatki.id_dodatku =
NEW.id_dodatku);

  NEW.cena = cena_p;
  IF cena_d IS NOT NULL THEN
    NEW.cena = cena_p + cena_d;
  END IF;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql' VOLATILE;

DROP TRIGGER IF EXISTS zamowienia_oblicz_cene ON szczegoly_zamowienia;
CREATE TRIGGER zamowienia_oblicz_cene BEFORE INSERT OR UPDATE
ON szczegoly_zamowienia FOR EACH ROW
EXECUTE PROCEDURE oblicz_cene();
```

Przykład

Pole cena (money) zostało uzupełnione automatycznie na podstawie ceny danego produktu oraz dodatku.

→ cena produktu o id_produktu = 7 wynosi 25 zł

→ cena dodatku o id_dodatku = 8 wynosi 5 zł

→ łączna cena wynosi 30 zł

Okno wyjściowe					
Dane wyjściowe Plan zapytania Komunikaty Historia					
	id_pozycji integer	id_zamowienia integer	id_produktu integer	id_dodatku integer	cena money
1	2531	722	7	8	30,00 zł
2	2532	722	10	7	17,00 zł
3	2533	722	10	7	17,00 zł
4	2534	722	12	8	55,00 zł
5	2535	723	9	8	45,00 zł

10.2 Funkcja wyzwalacza oblicz_punkty_klienta()

Funkcja wyzwalacza oblicz_punkty_klienta() wywoływana jest na tabeli zamówienia po operacji INSERT lub UPDATE. Jeżeli klient posiada konto, jego łączna ilość punktów jest aktualizowana na podstawie ilości punktów, którą klient zgromadził składając zamówienia.

Kod SQL:

```
CREATE OR REPLACE FUNCTION oblicz_punkty_klienta()
  RETURNS TRIGGER AS
$$
DECLARE
  konto INTEGER;
  pkt_klienta INTEGER;
BEGIN
  konto = (SELECT COUNT(*) FROM klienci_dane_logowania k WHERE k.id_klienta = (SELECT
id_klienta FROM zamowienia z WHERE z.id_zamowienia = NEW.id_zamowienia));
  IF konto>0 THEN
    pkt_klienta = (SELECT SUM(ilosc_punktow) FROM zamowienia z WHERE z.id_klienta =
NEW.id_klienta);
```

```

        UPDATE klienci k SET ilosc_punktow = pkt_klienta WHERE k.id_klienta = NEW.id_klienta;
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql' VOLATILE;

```

```

DROP TRIGGER IF EXISTS zamowienia_oblicz_punkty_klienta ON zamowienia;
CREATE TRIGGER zamowienia_oblicz_punkty_klienta AFTER INSERT OR UPDATE
ON zamowienia FOR EACH ROW
EXECUTE PROCEDURE oblicz_punkty_klienta();

```

Przykład

Łączna ilość punktów klienta została automatycznie wyliczona na podstawie punktów przez niego zgromadzonych.

Okno wyjściowe			
Dane wyjściowe		Plan zapytania	Komunikaty
		Historia	
	id_klienta integer	ilosc_punktow integer	
1	131	205	

10.3 Funkcja wyzwalacza oblicz_punkty_zamowienia()

Funkcja wyzwalacza oblicz_punkty_zamowienia() wywoływana jest na tabeli szczegoly_zamowienia po operacji INSERT, UPDATE lub DELETE. Punkty dla danego zamówienia obliczane są na podstawie progów odczytanych z tabeli punkty_zasady. Dla każdego zamówienia przypisywana jest ilość punktów.

Kod SQL:

```

CREATE OR REPLACE FUNCTION oblicz_punkty_zamowienia()
    RETURNS TRIGGER AS

```

```

$$
DECLARE
    pkt INTEGER;
BEGIN

IF (TG_OP != 'DELETE') THEN
    pkt = (SELECT punkty FROM punkty_zasady WHERE prog <= (SELECT SUM(cena) FROM
szczegoly_zamowienia WHERE id_zamowienia = NEW.id_zamowienia) ORDER BY prog DESC
LIMIT 1);
    IF pkt IS NULL THEN
        pkt = 0;
    END IF;
    UPDATE zamowienia SET ilosc_punktow = pkt WHERE zamowienia.id_zamowienia =
NEW.id_zamowienia;

    RETURN NEW;
END IF;
    UPDATE zamowienia SET ilosc_punktow = 0 WHERE zamowienia.id_zamowienia =
OLD.id_zamowienia;
    RETURN NULL;
END;
$$
LANGUAGE 'plpgsql' VOLATILE;

DROP TRIGGER IF EXISTS zamowienia_oblicz_punkty_zamowienia ON szczegoly_zamowienia;
CREATE TRIGGER zamowienia_oblicz_punkty_zamowienia AFTER INSERT OR UPDATE OR
DELETE ON szczegoly_zamowienia FOR EACH ROW
EXECUTE PROCEDURE oblicz_punkty_zamowienia();

```

Przykład

Przykładowe zamówienie, którego łączna kwota wynosi 119 zł.

Okno wyjściowe			
Dane wyjściowe		Plan zapytania	Komunikaty
		Historia	
	zamowienie integer	laczna_cena money	
1	722	119,00 zł	

Szczegółowe informacje na temat tego zamówienia:

Okno wyjściowe						
Dane wyjściowe						
	id_zamowienia integer	id_klienta integer	data_zamowienia timestamp without time zone	ilosc_punktow integer	komentarz text	ocena integer
1	722	102	2019-06-09 05:32:10.988212	25		

Tabela z zasadami przyznawania punktów:

Okno wyjściowe			
Dane wyjściowe			
	id_progu integer	prog money	punkty integer
1	1	50,00 zł	10
2	2	100,00 zł	25
3	3	150,00 zł	40
4	4	200,00 zł	80
5	5	250,00 zł	100

Kwota zamówienia wynosi 119zł, więc jest kwalifikuje się do progu o id = 2, wynika z tego, że ilość punktów dla tego zamówienia jest równa 25.

10.4 Funkcja wyzwalacza hash_haslo()

Funkcja wyzwalacza hash_haslo() wywoływana jest na tabeli klienci_dane_logowania przed operacją INSERT lub UPDATE. Funkcja powoduje hashowanie hasła za pomocą algorytmu MD5.

Kod SQL:

```
CREATE OR REPLACE FUNCTION hash_haslo()
  RETURNS TRIGGER AS
$$
DECLARE

BEGIN
  NEW.haslo = md5(NEW.haslo);
  RETURN NEW;
END;
$$
```

```
LANGUAGE 'plpgsql' VOLATILE;
```

```
DROP TRIGGER IF EXISTS klienci_dane_logowania_hash_haslo ON klienci_dane_logowania;  
CREATE TRIGGER klienci_dane_logowania_hash_haslo BEFORE INSERT OR UPDATE  
ON klienci_dane_logowania FOR EACH ROW  
EXECUTE PROCEDURE hash_haslo();
```

Przykład

Po wykonaniu kodu SQL:

INSERT INTO klienci_dane_logowania (id_klienta, login, haslo) VALUES (1, 'ernestbies', 'tajnehaslo123'); hasło zostało zahashowane za pomocą algorytmu MD5.

Okno wyjściowe				
Dane wyjściowe				
	id_dane_logowania integer	id_klienta integer	login character varying	haslo character varying(60)
35	35	96	Łucja96	ca38caa7313c6aaae0a721027974ab8f
36	36	98	Ernest98	18b7c551646c04b9a18babd54d619164
37	40	5	Łucja665	1bd51fba6259eb5fcd989bf3ee94de18
38	37	5	Łucja66	e79f6bf05011abb329bee192e4a4a7c2
39	41	1	ernestbies	80da6a8c72e09d7f8ff31787cb02f67b