

STUDENCKA PRACOWNIA BAZ DANYCH  
INSTYTUT INFORMATYKI  
UNIwersytetu Wrocławskiego

Łukasz CZAPLIŃSKI

---

# Coffee Shop

DOKUMENTACJA OGÓLNA  
WERSJA 1.0

---

Wrocław 2014

Tabela 1: Historia zmian

Wersja	Data	Opis	Autor
1.0	2014-06-13	Powstanie dokumentu	Łukasz Czapliński

## Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>3</b>
1.1	Cel dokumentu . . . . .	3
<b>2</b>	<b>Zadania projektu</b>	<b>3</b>
<b>3</b>	<b>Model</b>	<b>3</b>
3.1	Model konceptualny . . . . .	3
3.2	Model fizyczny – obiekty . . . . .	3
3.2.1	Dostawca i klient . . . . .	5
3.2.2	Właściciel . . . . .	5
3.2.3	Typ produktu . . . . .	5
3.2.4	Produkt . . . . .	6
3.2.5	Zamówienie . . . . .	6
3.3	Model fizyczny – związki dodatkowe . . . . .	6
3.3.1	Kto dostarcza jaki typ produkt . . . . .	6
3.4	Model fizyczny – wyzwalacze . . . . .	7
3.4.1	Aktualizacja ceny produktu i wartości zamówienia . . . . .	7
3.5	Model fizyczny – role . . . . .	8
3.5.1	Klient . . . . .	8
3.5.2	Właściciel . . . . .	8
3.5.3	Dostawca . . . . .	9

# 1 Wprowadzenie

## 1.1 Cel dokumentu

Dokument ten ma na celu sprecyzowanie zadań projektu CoffeeShop i sposobów ich realizacji.

## 2 Zadania projektu

Celem projektu jest stworzenie aplikacji bazodanowej pozwalającej na obsługę prostego CoffeeShopu. Ma być on zdecentralizowany – kilku właścicieli, z których każdy ma swoje towary. Mogą oni współdzielić dostawców towarów. Klienci mają mieć wybór co kupują od którego właściciela. Powinno się to odbywać zdalnie – zarówno pomiędzy klientami a właścicielami (klient zamawia towary u konkretnego dostawcy, a ten kontaktuje się z nim w sprawie odbioru i zapłaty) oraz dostawcami i właścicielami (właściciel wybiera jaki typ produktu chce zamówić i kontaktuje się z dostawcą lub odwrotnie – dostawca pyta właścicieli z którymi współpracował czego będą potrzebować).

## 3 Model

### 3.1 Model konceptualny

Jest przedstawiony na rys. [1].

Można w nim wyróżnić 3 główne role: klient, dostawca i właściciel oraz 3 obiekty którymi operują: produkty, ich typy oraz zamówienia.

Klient składa zamówienia do właściciela na konkretne produkty.

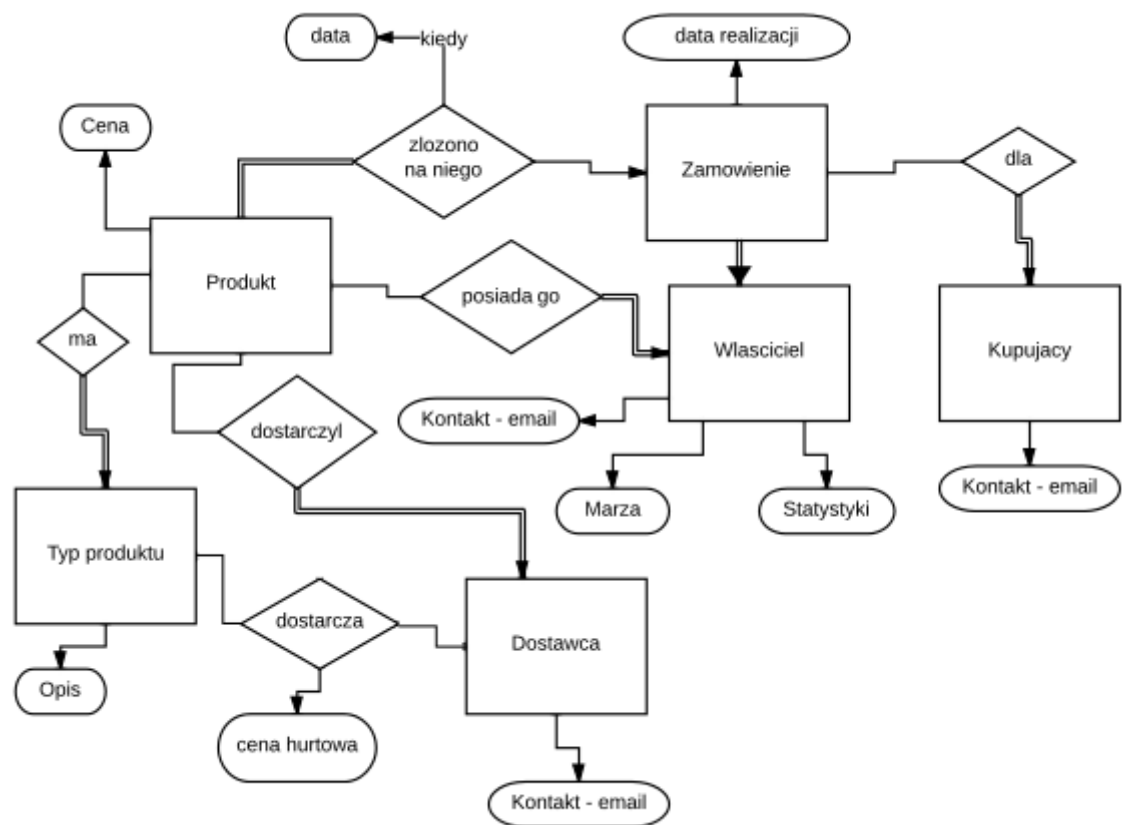
Właściciel dodaje nowe produkty wybierając z oferty dostawców.

Dostawca rejestruje jakie typy produktów w jakiej cenie może zapewnić.

Zamawianie dostaw nie jest kontrolowane przez tą bazę danych – odbywa się pomiędzy właścicielem a dostawcą przez umówione kontakty – maile.

### 3.2 Model fizyczny – obiekty

Każdy obiekt jest reprezentowany przez unikalny klucz – liczbę nadawaną automatycznie z sekwencji oraz informacje dodatkowe.



Rysunek 1: Model konceptualny bazy danych

### 3.2.1 Dostawca i klient

Osoby posiadają pole kontakt – powinien być to mail (jest to automatycznie sprawdzane przy dodawaniu obiektu do bazy).

```
create sequence doid_seq;
create table dostawca (
  doid integer primary key default nextval('doid_seq'),
  mail text unique check (mail ~ '([a-z] | [._, -] | \d)+@(([a-z]
    ↪ | [._, -] | \d)+) \.(com | org | pl | fm) ')
);
```

```
create sequence kuid_seq;
create table kupujacy (
  kuid integer primary key default nextval('kuid_seq'),
  mail text unique check (mail ~ '([a-z] | [._, -] | \d)+@(([a-z]
    ↪ | [._, -] | \d)+) \.(com | org | pl | fm) ')
);
```

### 3.2.2 Właściciel

Dodatkowo właściciel ma określoną marżę – wskaźnik ile procent sprzedawany przez niego produkt będzie droższy od jego ceny hurtowej.

```
create sequence wlid_seq;
create table wlasciciel (
  wlid integer primary key default nextval('wlid_seq'),
  mail text unique check (mail ~ '([a-z] | [._, -] | \d)+@(([a-z]
    ↪ | [._, -] | \d)+) \.(com | org | pl | fm) '),
  marza decimal(3,2) not null check ( marza < 1 and marza >
    ↪ 0)
);
```

### 3.2.3 Typ produktu

```
create sequence tpid_seq;
create table typ_produkta (
```

```

    tpid integer primary key default nextval( 'tpid_seq' ),
    nazwa text ,
    opis text
);

```

### 3.2.4 Produkt

Cena jest automatycznie wyliczaną wartością (na podstawie marży właściciela i ceny hurtowej): [3.4.1].

```

create sequence prid_seq;
create table produkt (
    prid integer primary key default nextval( 'prid_seq' ),
    tpid integer not null references typ_produktu ,
    cena decimal(6,2) not null ,
    zaid integer null references zamowienie ,
    doid integer not null references dostawca ,
    wlid integer not null references wlasciciel
);

```

### 3.2.5 Zamówienie

Wartość zamówiona jest sumą cen produktów na niego się składających: [3.4.1].

```

create sequence zaid_seq;
create table zamowienie (
    zaid integer primary key default nextval( 'zaid_seq' ),
    realizacja date null check (realizacja >= zlozenie),
    zlozenie date null ,
    wartosc integer not null default 0 ,
    kuid integer not null references kupujacy ,
    wlid integer not null references wlasciciel
);

```

## 3.3 Model fizyczny – związki dodatkowe

### 3.3.1 Kto dostarcza jaki typ produkt

Tabela ta pozwala zapisać kto dostarcza jaki towar w jakiej cenie.

```

create table dostarcza (
    tpid integer not null references typ_produktu,
    doid integer not null references dostawca,
    cena decimal(9,2) not null,
    constraint dostarczaj_tylko_raz_dany_typ unique (tpid, doid
    ↪ );
);

```

### 3.4 Model fizyczny – wyzwalacze

#### 3.4.1 Aktualizacja ceny produktu i wartości zamówienia

Odbywa się po zmianie/dodaniu elementu do tabeli **produkt**.

```

create function zwieksz_wartosc_produktu_i_zamowienia()
    ↪ returns trigger as $$
begin
    if (TG_OP = 'UPDATE' and ( old.zaid is not null or new.wlid
    ↪ != old.wlid or new.doid != old.doid )) then
        raise exception 'nie_mozna_zmienic_produktu';
    end if;
    if (exists (select * from zamowienie where zamowienie.zaid
    ↪ = new.zaid and zlozenie is not null)) then
        raise exception 'nie_mozna_dodac_do_zlozonego_zamowienia'
        ↪ ;
    end if;
    if (new.zaid is not null and not exists (select * from
    ↪ zamowienie where zamowienie.zaid = new.zaid)) then
        raise exception 'nie_mozna_dodac_do_nieistniejacego_
        ↪ zamowienia';
    end if;
    if (TG_OP = 'INSERT' and not exists (
        select * from dostarcza where dostarcza.doid = new.doid
        ↪ and dostarcza.tpid = new.tpid)) then
        raise exception 'ten_dostawca_nie_ma_tego_produktu';
    end if;

```

```

select into new.cena (1.00 + Y.marza) * X.cena from (select
    ↪ cena from dostarcza where dostarcza.tpid = new.tpid
    ↪ and dostarcza.doid = new.doid) X
cross join (select marza from wlasciciel where wlasciciel
    ↪ .wlid = new.wlid) Y;
if (new.zaid is not null) then
    update zamowienie set wartosc = wartosc + new.cena where
        ↪ zamowienie.zaid = new.zaid;
end if;
return new;
end
$$ language plpgsql;

```

## 3.5 Model fizyczny – role

### 3.5.1 Klient

```

create role buyer;
revoke all on produkt, zamowienie, dostarcza, typ_produkta,
    ↪ dostawca, kupujacy, wlasciciel from buyer cascade;
grant update on produkt to buyer;
grant insert, update on zamowienie to buyer;
grant select on produkt, typ_produkta, zamowienie to buyer;
grant select on wlasciciel, dostarcza to buyer;
grant all on zaid_seq, doid_seq, wlid_seq, tpid_seq, prid_seq to
    ↪ buyer;

```

### 3.5.2 Właściciel

```

create role owner;
revoke all on produkt, zamowienie, dostarcza, typ_produkta,
    ↪ dostawca, kupujacy, wlasciciel from owner cascade;
grant insert on produkt to owner;
grant update on zamowienie to owner;
grant select on zamowienie, dostarcza, typ_produkta, produkt
    ↪ to owner;

```



```
grant select on dostawca, kupujacy to owner;  
grant all on zaid_seq, doid_seq, wlid_seq, tpid_seq, prid_seq to  
    ↪ owner;  
grant select on wlasciciel to owner;
```

### 3.5.3 Dostawca

```
drop role if exists provider;  
create role provider;  
revoke all on produkt, zamowienie, dostarcza, typ_produkту,  
    ↪ dostawca, kupujacy, wlasciciel from provider cascade;  
grant insert, update on typ_produkту, dostarcza to provider;  
grant select on dostarcza, typ_produkту, produkt to provider;  
grant select on wlasciciel to provider;  
grant delete on dostarcza, typ_produkту to provider;  
grant all on zaid_seq, doid_seq, wlid_seq, tpid_seq, prid_seq to  
    ↪ provider;
```