Dokumentacja Projektu Bazy Danych I

Kacper Tracz

18 stycznia 2024

Spis treści

1 Wprowadzenie **3**
1.1 Temat projektu . . . 3
1.2 Analiza wymagań użytkownika . . . 3
1.3 Zaprojektowanie funkcji . . . 3

2 Zdefiniowanie encji (obiektów) oraz ich atrybutów **4**
2.1 Tabela: `oddzial_glowny` . . . 4
2.2 Tabela: `oddzial` . . . 4
2.3 Tabela: `straznik` . . . 5
2.4 Tabela: `licencja` . . . 5
2.5 Tabela: `rybak` . . . 6
2.6 Tabela: `zbiornik` . . . 6
2.7 Tabela: `zwierze` . . . 7
2.8 Tabela: `zwierze_zbiornik` . . . 7
2.9 Tabela: `lista` . . . 8
2.10 Tabela: `rynek` . . . 8
2.11 Tabela: `rynek_zwierze` . . . 9

3 Zaprojektowanie relacji pomiędzy encjami (ERD) **10**
3.1 Diagram ERD . . . 10
3.2 Opis relacji . . . 10

4 Wyzwalacze i funkcje w Bazie Danych **11**
4.1 Wyzwalacz badający wprowadzane dane do tabeli `licencja` . . . 11
4.1.1 `sprawdz_licencje()` . . . 11
4.1.2 `licence_checker` . . . 11
4.2 Wyzwalacz generujący nową licencję . . . 12
4.2.1 `create_licence()` . . . 12
4.2.2 `licence_creator` . . . 12
4.3 Wyzwalacz ustalający legalność zbiornika . . . 12
4.3.1 `legal_setter()` . . . 12
4.3.2 `legal_setter` . . . 13
4.4 Wyzwalacz dodający zwierzęta do zbiornika . . . 13
4.4.1 `add_animals()` . . . 13
4.4.2 `animal_adder` . . . 13
4.5 Wyzwalacz usuwający rybaka . . . 13

4.5.1 `rybak_delete()` . . . 13
4.5.2 `rybak_delete` . . . 14
4.6 Wyzwalacz tworzący listę ryb dla rybaka . . . 14
4.6.1 `create_list()` . . . 14
4.6.2 `list_creator` . . . 14
4.7 Wyzwalacz dodający elementy do marketu . . . 15
4.7.1 `add_to_market()` . . . 15
4.7.2 `market_price_adder` . . . 15
4.8 Wyzwalacz usuwający rynek . . . 15
4.8.1 `delete_rynek()` . . . 15
4.8.2 `rynek_zwierze_deleter` . . . 15
4.9 Wyzwalacz usuwający zwierzę . . . 16
4.9.1 `delete_zwierze()` . . . 16
4.9.2 `zwierze_deleter` . . . 16
4.10 Wyzwalacz usuwający zbiornik . . . 16
4.10.1 `delete_zbiornik()` . . . 16
4.10.2 `zbiornik_deleter` . . . 16
4.11 Wyzwalacz usuwający oddział . . . 17
4.11.1 `delete_oddzial()` . . . 17
4.11.2 `oddzial_deleter` . . . 17
4.12 Wyzwalacz usuwający oddział główny . . . 17
4.12.1 `delete_oddzial_glowny()` . . . 17
4.12.2 `oddzial_glowny_deleter` . . . 17
4.13 Funkcja do generowania liczb losowych z przedziału [min, max] . . . 18
4.13.1 `random_generator()` . . . 18
4.14 Funkcja do sprzedawania ryb danego rybaka w danym sklepie . . . 18
4.14.1 `sell_animals()` . . . 18
4.15 Funkcja do sprawdzania poprawności licencji rybaka . . . 19
4.15.1 `check_rybak_licence()` . . . 19
4.15.2 `rewrite_db()` . . . 20

5 Aplikacja klienta **21**
5.1 Struktura Projektu . . . 22
5.2 Działanie projektu w Django . . . 22
5.3 utils.py . . . 22
5.4 urls.py . . . 23
5.5 forms.py . . . 23
5.6 views.py . . . 23
5.7 Formularze . . . 24
5.8 Obsługa aplikacji . . . 25

6 Wykaz literatury **26**

1 Wprowadzenie

1.1 Temat projektu

Tematem projektu jest wędkarstwo i treści z nim powiązane, takie jak struktura administracyjna czy rynek. Ma on na celu sprawdzenie umiejętności tworzenia baz danych w PostgreSQL jak również graficznej prezentacji informacji.

1.2 Analiza wymagań użytkownika

Projektowana baza danych ma służyć obszarowi zarządzania rybołówstwem/myślictwem oraz handlem produktami zwierzęcymi. Poniżej przedstawiono główne wymagania użytkownika:

1. **Zarządzanie oddziałami:** Możliwość dodawania, usuwania i modyfikowania informacji o oddziałach oraz ich relacji hierarchicznych.
2. **Rejestracja strażników:** System powinien umożliwiać dodawanie nowych strażników do konkretnych oddziałów. Informacje dotyczące strażników obejmują imię, nazwisko, wiek i przypisany oddział.
3. **Licencje dla rybaków:** Możliwość generowania automatycznych licencji dla rybaków wraz z kontrolą ich ważności. Licencje powinny być przypisywane do konkretnych rybaków.
4. **Zarządzanie rybakami:** System powinien umożliwiać dodawanie, usuwanie i modyfikowanie informacji o rybakach, takich jak imię, nazwisko, wiek, stan konta oraz przypisana licencja.
5. **Zarządzanie zbiornikami:** Możliwość dodawania, usuwania i modyfikowania informacji o zbiornikach wraz z ich przypisanymi oddziałami. Informacje o zbiornikach obejmują nazwę, pojemność, legalność, przypisany oddział.
6. **Zarządzanie zwierzętami:** System powinien umożliwiać dodawanie, usuwanie i modyfikowanie informacji o zwierzętach. Każde zwierzę powinno być przypisane do konkretnego zbiornika.
7. **Generowanie listy rybaków:** Automatyczne tworzenie listy zwierząt, które każdy rybak może złowić w określonym czasie.
8. **Zarządzanie rynkami:** Możliwość dodawania, usuwania i modyfikowania informacji o rynkach, wraz z przypisanym głównym oddziałem.
9. **Handel na rynku:** System powinien umożliwiać dodawanie ofert sprzedaży zwierząt na rynku wraz z ustalaniem cen.

1.3 Zaprojektowanie funkcji

W oparciu o analizę wymagań użytkownika, poniżej przedstawiono podstawowe funkcje realizowane w bazie danych:

1. **Dodaj, usuń, edytuj oddział główny**

2. **Dodaj, usuń, edytuj oddział**
3. **Dodaj, usuń, edytuj strażnika**
4. **Dodaj, usuń, edytuj zbiornik**
5. **Generuj licencję**
6. **Dodaj, usuń, edytuj licencję**
7. **Dodaj, usuń, edytuj rybaka**
8. **Dodaj, usuń, edytuj zwierzę**
9. **Generuj listy dla rybaków**
10. **Generuj zwierzęta w zbiorniku**
11. **Dodaj, usuń, edytuj rynek**
12. **Generuj oferty sprzedaży zwierzęcia na rynku**
13. **Dodaj, usuń, edytuj ofertę sprzedaży zwierzęcia na rynku**

2 Zdefiniowanie encji (obiektów) oraz ich atrybutów

2.1 Tabela: `oddzial_glowny`

Tabela `oddzial_glowny` przechowuje informacje o głównych oddziałach.

Nazwa - Unikalna nazwa oddziału głównego.

```
CREATE TABLE projekt.oddzial_glowny
(
    nazwa VARCHAR NOT NULL,
    CONSTRAINT oddzial_glowny_pk PRIMARY KEY (nazwa)
);
```

2.2 Tabela: `oddzial`

Tabela `oddzial` przechowuje informacje o oddziałach podległych głównym oddziałom.

Nazwa - Unikalna nazwa oddziału.

Oddział Nadrędny - Nazwa głównego oddziału, do którego podlega dany oddział.

```
CREATE TABLE projekt.oddzial
(
    nazwa VARCHAR NOT NULL,
    oddzial_nadrzedny VARCHAR NOT NULL,
    CONSTRAINT oddzial_pk PRIMARY KEY (nazwa),
    CONSTRAINT oddzial_fk FOREIGN KEY (oddzial_nadrzedny)
    REFERENCES projekt.oddzial_glowny(nazwa) ON DELETE CASCADE
);
```

2.3 Tabela: `straznik`

Tabela `straznik` przechowuje informacje o strażnikach.

Straznik_ID - Unikalny identyfikator strażnika.

Imie - Imię strażnika.

Nazwisko - Nazwisko strażnika.

Wiek - Wiek strażnika.

Oddzial_ID - Nazwa oddziału, do którego przypisany jest strażnik.

```
CREATE TABLE projekt.straznik
(
    straznik_id SERIAL,
    imie VARCHAR NOT NULL,
    nazwisko VARCHAR NOT NULL,
    wiek INTEGER NOT NULL CHECK(wiek>18),
    oddzial_id VARCHAR NOT NULL,
    CONSTRAINT straznik_pk PRIMARY KEY (straznik_id),
    CONSTRAINT straznik_fk FOREIGN KEY (oddzial_id)
    REFERENCES projekt.oddzial(nazwa) ON DELETE CASCADE
);
```

2.4 Tabela: `licencja`

Tabela `licencja` przechowuje informacje o licencjach.

Licencja_ID - Unikalny identyfikator licencji.

Data_Startu - Data rozpoczęcia ważności licencji.

Data_Konca - Data zakończenia ważności licencji.

```
CREATE TABLE projekt.licencja
(
    licencja_id SERIAL,
    data_startu DATE NOT NULL,
    data_konca DATE NOT NULL,
    CONSTRAINT licencja_pk PRIMARY KEY (licencja_id)
);
```

2.5 Tabela: `rybak`

Tabela `rybak` przechowuje informacje o rybakach.

Rybak_ID - Unikalny identyfikator rybaka.

Imie - Imię rybaka.

Nazwisko - Nazwisko rybaka.

Stan_Konta - Stan konta rybaka.

Wiek - Wiek rybaka.

Licencja_ID - Identyfikator powiązanej licencji.

```
CREATE TABLE projekt.rybak
(
    rybak_id SERIAL,
    imie VARCHAR NOT NULL,
    nazwisko VARCHAR NOT NULL,
    stan_konta NUMERIC(10,2) NOT NULL CHECK(stan_konta>=0),
    wiek INTEGER NOT NULL CHECK(wiek>18),
    licencja_id INTEGER UNIQUE,
    CONSTRAINT rybak_pk PRIMARY KEY (rybak_id),
    CONSTRAINT rybak_fk FOREIGN KEY (licencja_id)
    REFERENCES projekt.licencja(licencja_id) ON DELETE CASCADE
);
```

2.6 Tabela: `zbiornik`

Tabela `zbiornik` przechowuje informacje o zbiornikach.

Nazwa - Unikalna nazwa zbiornika.

Objetosc - Pojemność zbiornika.

Legalny - Flaga określająca, czy zbiornik jest legalny.

Oddzial - Nazwa oddziału, do którego przypisany jest zbiornik.

```
CREATE TABLE projekt.zbiornik
(
    nazwa VARCHAR NOT NULL,
    objetosc INTEGER NOT NULL CHECK(objetosc >18),
    legalny BOOLEAN NOT NULL,
    oddzial VARCHAR NOT NULL,
    CONSTRAINT zbiornik_pk PRIMARY KEY (nazwa),
    CONSTRAINT zbiornik_fk FOREIGN KEY (oddzial)
    REFERENCES projekt.oddzial(nazwa) ON DELETE CASCADE
);
```

2.7 Tabela: `zwierze`

Tabela `zwierze` przechowuje informacje o zwierzętach.

Nazwa - Unikalna nazwa zwierzęcia.

Gatunek - Gatunek zwierzęcia.

Legalna - Flaga określająca, czy zwierzę jest legalne.

```
CREATE TABLE projekt.zwierze
(
    nazwa VARCHAR NOT NULL,
    gatunek VARCHAR NOT NULL,
    legalna BOOLEAN NOT NULL,
    CONSTRAINT zwierze_pk PRIMARY KEY (nazwa)
);
```

2.8 Tabela: `zwierze_zbiornik`

Tabela `zwierze_zbiornik` przechowuje informacje o przypisaniach zwierząt do zbiorników.

Zwierze_Zbiornik_ID - Unikalny identyfikator przypisania.

Zwierze - Nazwa zwierzęcia.

Zbiornik - Nazwa zbiornika.

```
CREATE TABLE projekt.zwierze_zbiornik
(
    zwierze_zbiornik_id SERIAL,
    zwierze VARCHAR NOT NULL,
    zbiornik VARCHAR NOT NULL,
    CONSTRAINT zwierze_zbiornik_pk PRIMARY KEY(zwierze_zbiornik_id),
    CONSTRAINT zwierze_fk FOREIGN KEY (zwierze)
```

```
    REFERENCES projekt.zwierze(nazwa) ON DELETE CASCADE,
    CONSTRAINT zbiornik_fk FOREIGN KEY (zbiornik)
    REFERENCES projekt.zbiornik(nazwa) ON DELETE CASCADE
);
```

2.9 Tabela: `lista`

Tabela `lista` przechowuje informacje o listach rybaków.

Lista_ID - Unikalny identyfikator listy.

Zwierze - Nazwa zwierzęcia.

Rybak - Identyfikator powiązanego rybaka.

Ilosc - Ilość zwierząt na liście.

```
CREATE TABLE projekt.lista
(
    lista_id SERIAL,
    zwierze VARCHAR NOT NULL,
    rybak INTEGER NOT NULL,
    ilosc INTEGER NOT NULL CHECK(ilosc>0),
    CONSTRAINT lista_pk PRIMARY KEY(lista_id),
    CONSTRAINT zwierze_fk FOREIGN KEY (zwierze)
    REFERENCES projekt.zwierze(nazwa) ON DELETE CASCADE,
    CONSTRAINT rybak_fk FOREIGN KEY (rybak)
    REFERENCES projekt.rybak(rybak_id) ON DELETE CASCADE
);
```

2.10 Tabela: `rynek`

Tabela `rynek` przechowuje informacje o rynkach.

Nazwa - Unikalna nazwa rynku.

Oddzial_Glowny - Nazwa głównego oddziału przypisanego do rynku.

```
CREATE TABLE projekt.rynek
(
    nazwa VARCHAR NOT NULL,
    oddzial_glowny VARCHAR NOT NULL,
    CONSTRAINT rynek_pk PRIMARY KEY (nazwa),
    CONSTRAINT oddzial_fk FOREIGN KEY (oddzial_glowny)
    REFERENCES projekt.oddzial_glowny(nazwa) ON DELETE CASCADE
);
```

2.11 Tabela: `rynek_zwierze`

Tabela `rynek_zwierze` przechowuje informacje o przypisaniach zwierząt do rynków.

Rynek_Zwierze_ID - Unikalny identyfikator przypisania.

Rynek - Nazwa rynku.

Zwierze - Nazwa zwierzęcia.

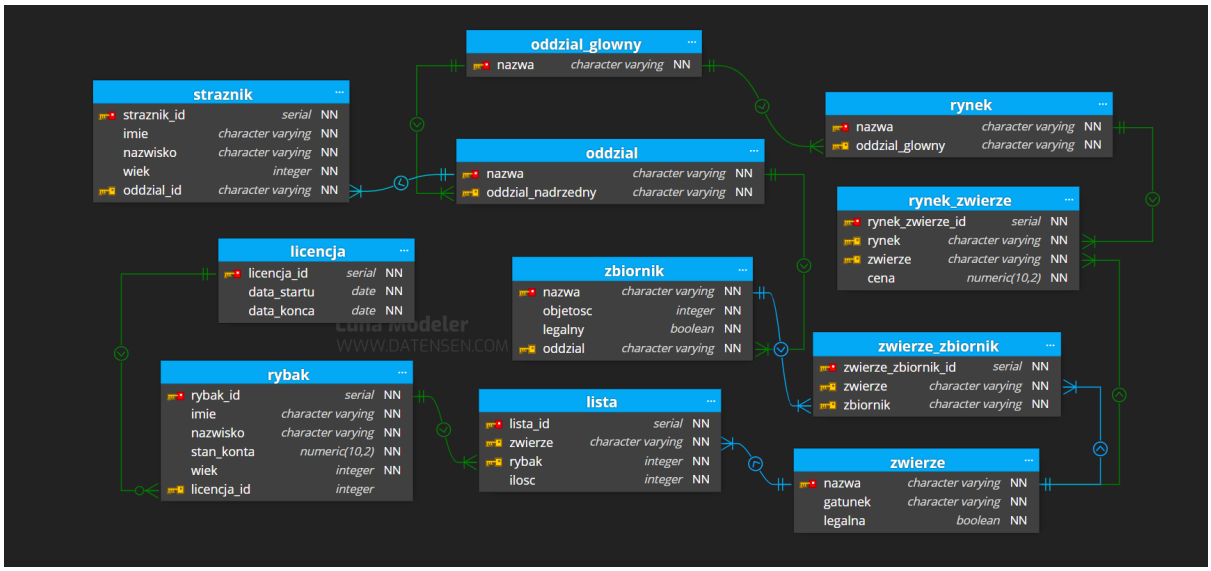
Cena - Cena zwierzęcia na rynku.

```
CREATE TABLE projekt.rynek_zwierze
(
    rynek_zwierze_id SERIAL,
    rynek VARCHAR NOT NULL,
    zwierze VARCHAR NOT NULL,
    cena NUMERIC(10, 2) NOT NULL CHECK(cena>0),
    CONSTRAINT rynek_zwierze_pk PRIMARY KEY (rynek_zwierze_id),
    CONSTRAINT rynek_fk FOREIGN KEY (rynek)
    REFERENCES projekt.rynek(nazwa),
    CONSTRAINT zwierze_fk FOREIGN KEY (zwierze)
    REFERENCES projekt.zwierze(nazwa) ON DELETE CASCADE
);
```

3 Zaprojektowanie relacji pomiędzy encjami (ERD)

W tej sekcji przedstawione są relacje pomiędzy poszczególnymi encjami w formie diagramu ERD oraz opisu relacji.

3.1 Diagram ERD



Rysunek 1: ERD

3.2 Opis relacji

Relacja 1: Tabela `oddzial` jest powiązana z tabelą `oddzial_glowny` poprzez klucz obcy `oddzial_nadrzedny`, wskazujący na główny oddział, do którego podlega dany oddział.

Relacja 2: Tabela `straznik` jest powiązana z tabelą `oddzial` poprzez klucz obcy `oddzial_id`, wskazujący na oddział, do którego przypisany jest strażnik.

Relacja 3: Tabela `rybak` jest powiązana z tabelą `licencja` poprzez klucz obcy `licencja_id`, wskazujący na licencję przypisaną danemu rybakowi.

Relacja 4: Tabela `rybak` jest powiązana z tabelą `lista` poprzez klucz główny `rybak_id`, wskazujący na identyfikator rybaka przypisanego do listy.

Relacja 5: Tabela `lista` jest powiązana z tabelą `zwierze` poprzez klucz obcy `zwierze`, wskazujący na zwierzę znajdujące się na liście.

Relacja 6: Tabela `lista` jest powiązana z tabelą `rybak` poprzez klucz obcy `rybak`, wskazujący na rybaka związane z listą.

Relacja 7: Tabela `zbiornik` jest powiązana z tabelą `oddzial` poprzez klucz obcy `oddzial`, wskazujący na oddział, do którego przypisany jest zbiornik.

Relacja 8: Tabela `zbiornik` jest powiązana z tabelą `zwierze_zbiornik` poprzez klucz główny `nazwa`, wskazujący na zbiornik, do którego przypisane są zwierzęta.

Relacja 9: Tabela `zwierze` jest powiązana z tabelą `zwierze_zbiornik` poprzez klucz główny `nazwa`, wskazujący na zwierzę przypisane do zbiornika.

Relacja 10: Tabela `zwierze` jest powiązana z tabelą `lista` poprzez klucz obcy `zwierze`, wskazujący na zwierzę znajdujące się na liście rybaka.

Relacja 11: Tabela `rynek` jest powiązana z tabelą `oddzial_glowny` poprzez klucz obcy `oddzial_glowny`, wskazujący na główny oddział przypisany do rynku.

Relacja 12: Tabela `rynek` jest powiązana z tabelą `rynek_zwierze` poprzez klucz główny `nazwa`, wskazujący na rynek, do którego przypisane są zwierzęta.

Relacja 13: Tabela `rynek_zwierze` jest powiązana z tabelą `zwierze` poprzez klucz główny `zwierze`, wskazujący na zwierzę przypisane do rynku.

4 Wyzwalacze i funkcje w Bazie Danych

4.1 Wyzwalacz badający wprowadzane dane do tabeli `licencja`

4.1.1 `sprawdz_licencje()`

Wyzwalacz ten sprawdza, czy data rozpoczęcia (`data_startu`) licencji nie jest późniejsza niż data zakończenia (`data_konca`). Jeśli warunek nie jest spełniony, wyzwalacz zgłasza wyjątek.

```
create or replace function sprawdz_licencje() returns trigger as
$$
begin
  if new.data_startu >= new.data_konca then
    raise exception 'Data startu nie może być później niż data zakończenia!';
    return null;
  else
    return new;
  end if;
end
$$
language plpgsql;
```

4.1.2 `licence_checker`

Wyzwalacz `licence_checker` jest dodany do tabeli `licencja` i wykonuje funkcję `sprawdz_licencje()` przed operacjami `insert` lub `update`.

```
create trigger licence_checker
before insert or update on projekt.licencja
for each row execute procedure sprawdz_licencje();
```

4.2 Wyzwalacz generujący nową licencję

4.2.1 create_licence()

Wyzwalacz `create_licence()` decyduje losowo, czy należy utworzyć nową licencję dla rybaka. Jeśli tak, losuje liczbę lat ważności i dodaje nową licencję.

```
create or replace function create_licence()
returns trigger as
$$
declare
czy_tworzyc_licencje integer;
ile_lat_wazna integer;
begin
  czy_tworzyc_licencje := random_generator(0, 3);
  if czy_tworzyc_licencje = 0 then
    return new;
  else
    ile_lat_wazna := random_generator(1, 10000);
    insert into projekt.licencja (data_startu, data_konca)
      values (CURRENT_DATE, CURRENT_DATE + ile_lat_wazna);
    new.licencja_id := (select max(l.licencja_id) from projekt.licencja l);
    return new;
  end if;
end
$$
language plpgsql;
```

4.2.2 licence_creator

Wyzwalacz `licence_creator` jest dodany do tabeli `rybak` i wykonuje funkcję `create_licence()` przed operacją `insert`.

```
create trigger licence_creator
before insert on projekt.rybak
for each row execute procedure create_licence();
```

4.3 Wyzwalacz ustalający legalność zbiornika

4.3.1 legal_setter()

Wyzwalacz `legal_setter()` ustawia losowo flagę `legalny` dla nowo dodanego zbiornika.

```
create or replace function legal_setter() returns trigger as
$$
begin
  if random_generator(0, 1) then
    new.legalny := true;
  else
    new.legalny = false;
  end if;
```

```
  return new;
end
$$
language plpgsql;
```

4.3.2 legal_setter

Wyzwalacz `legal_setter` jest dodany do tabeli `zbiornik` i wykonuje funkcję `legal_setter()` przed operacją `insert`.

```
create trigger legal_setter
before insert on projekt.zbiornik
for each row execute procedure legal_setter();
```

4.4 Wyzwalacz dodający zwierzęta do zbiornika

4.4.1 add_animals()

Wyzwalacz `add_animals()` dodaje losową ilość losowych zwierząt do nowo utworzonego zbiornika.

```
create or replace function add_animals() returns trigger as
$$
begin
  insert into projekt.zwierze_zbiornik (zwierze, zbiornik)
    select z.nazwa, new.nazwa from projekt.zwierze z
    order by random() limit random_generator(10, 30);
  return new;
end
$$
language plpgsql;
```

4.4.2 animal_adder

Wyzwalacz `animal_adder` jest dodany do tabeli `zbiornik` i wykonuje funkcję `add_animals()` po operacji `insert`.

```
create trigger animal_adder
after insert on projekt.zbiornik
for each row execute procedure add_animals();
```

4.5 Wyzwalacz usuwający rybaka

4.5.1 rybak_delete()

Wyzwalacz `rybak_delete()` usuwa powiązaną licencję i wpisy w tabeli `lista` po usunięciu rybaka.

```
create or replace function rybak_delete() returns trigger as
$$
begin
```

```
  if old.licencja_id in (select l.licencja_id from projekt.licencja l
                         where l.licencja_id = old.licencja_id) then
    delete from projekt.licencja where old.licencja_id = licencja_id;
  end if;
  delete from projekt.lista  where old.rybak_id = rybak;
  return old;
end
$$
language plpgsql;
```

4.5.2 rybak_delete

Wyzwalacz `rybak_delete` jest dodany do tabeli `rybak` i wykonuje funkcję `rybak_delete()` po operacji `delete`.

```
create trigger rybak_delete
after delete on projekt.rybak
for each row execute procedure rybak_delete();
```

4.6 Wyzwalacz tworzący listę ryb dla rybaka

4.6.1 create_list()

Wyzwalacz `create_list()` dodaje wpisy do tabeli `lista`, tworząc listę zwierząt dla nowo dodanego rybaka.

```
create or replace function create_list() returns trigger as
$$
begin
  insert into projekt.lista (zwierze, rybak, ilosc)
    select z.nazwa, cast(new.rybak_id as integer),
           random_generator(1, 5) from projekt.zwierze z
    where z.nazwa in (select z2.nazwa from projekt.zwierze z2
                      order by random() limit random_generator(0, 20));
  return new;
end
$$
language plpgsql;
```

4.6.2 list_creator

Wyzwalacz `list_creator` jest dodany do tabeli `rybak` i wykonuje funkcję `create_list()` po operacji `insert`.

```
create trigger list_creator
after insert on projekt.rybak
for each row execute procedure create_list();
```

4.7 Wyzwalacz dodający elementy do marketu

4.7.1 add_to_market()

Wyzwalacz add_to_market() dodaje zwierzęta o losowej cenie do rynku po jego dodaniu.

```
create or replace function add_to_market() returns trigger as
$$
declare
rec record;
begin
  insert into projekt.rynek_zwierze (rynek, zwierze, cena)
    select new.nazwa, z.nazwa,
           cast(random_generator(1, 10000) as float)/100
    from projekt.zwierze z;
  return new;
end
$$
language plpgsql;
```

4.7.2 market_price_adder

Wyzwalacz market_price_adder jest dodany do tabeli rynek i wykonuje funkcję add_to_market() po operacji insert.

```
create trigger market_price_adder
after insert on projekt.rynek
for each row execute procedure add_to_market();
```

4.8 Wyzwalacz usuwający rynek

4.8.1 delete_rynek()

Wyzwalacz delete_rynek() usuwa powiązane rekordy z tabeli rynek_zwierze po usunięciu rekordu z tabeli rynek.

```
create or replace function delete_rynek() returns trigger as
$$
begin
  delete from projekt.rynek_zwierze where rynek = old.nazwa;
  return old;
end
$$
language plpgsql;
```

4.8.2 rynek_zwierze_deleter

Wyzwalacz rynek_zwierze_deleter jest dodany do tabeli rynek i wykonuje funkcję delete_rynek() przed operacją delete.

```
create trigger rynek_zwierze_deleter
before delete on projekt.rynek
for each row execute procedure delete_rynek();
```

4.9 Wyzwalacz usuwający zwierzę

4.9.1 delete_zwierze()

Wyzwalacz delete_zwierze() usuwa powiązane rekordy z tabeli lista, zwierze_zbiornik i rynek_zwierze po usunięciu rekordu z tabeli zwierze.

```
create or replace function delete_zwierze() returns trigger as
$$
begin
  delete from projekt.lista where zwierze = old.nazwa;
  delete from projekt.zwierze_zbiornik where zwierze = old.nazwa;
  delete from projekt.rynek_zwierze where zwierze = old.nazwa;
  return old;
end
$$
language plpgsql;
```

4.9.2 zwierze_deleter

Wyzwalacz zwierze_deleter jest dodany do tabeli zwierze i wykonuje funkcję delete_zwierze() przed operacją delete.

```
create trigger zwierze_deleter
before delete on projekt.zwierze
for each row execute procedure delete_zwierze();
```

4.10 Wyzwalacz usuwający zbiornik

4.10.1 delete_zbiornik()

Wyzwalacz delete_zbiornik() usuwa powiązane rekordy z tabeli zwierze_zbiornik po usunięciu rekordu z tabeli zbiornik.

```
create or replace function delete_zbiornik() returns trigger as
$$
begin
  delete from projekt.zwierze_zbiornik where old.nazwa = zbiornik;
  return old;
end
$$
language plpgsql;
```

4.10.2 zbiornik_deleter

Wyzwalacz zbiornik_deleter jest dodany do tabeli zbiornik i wykonuje funkcję delete_zbiornik() przed operacją delete.

```
create trigger zbiornik_deleter
before delete on projekt.zbiornik
for each row execute procedure delete_zbiornik();
```

4.11 Wyzwalacz usuwający oddział

4.11.1 delete_oddzial()

Wyzwalacz delete_oddzial() usuwa powiązane rekordy z tabeli zbiornik i straznik po usunięciu rekordu z tabeli oddzial.

```
create or replace function delete_oddzial() returns trigger as
$$
begin
  delete from projekt.zbiornik where old.nazwa = oddzial;
  delete from projekt.straznik where old.nazwa = oddzial_id;
  return old;
end
$$
language plpgsql;
```

4.11.2 oddzial_deleter

Wyzwalacz oddzial_deleter jest dodany do tabeli oddzial i wykonuje funkcję delete_oddzial() przed operacją delete.

```
create trigger oddzial_deleter
before delete on projekt.oddzial
for each row execute procedure delete_oddzial();
```

4.12 Wyzwalacz usuwający oddział główny

4.12.1 delete_oddzial_glowny()

Wyzwalacz delete_oddzial_glowny() usuwa rekordy z tabeli oddzial po usunięciu rekordu z tabeli oddzial_glowny.

```
create or replace function delete_oddzial_glowny() returns trigger as
$$
begin
  delete from projekt.oddzial where old.nazwa = oddzial_nadrzedny;
  return old;
end
$$
language plpgsql;
```

4.12.2 oddzial_glowny_deleter

Wyzwalacz oddzial_glowny_deleter jest dodany do tabeli oddzial_glowny i wykonuje funkcję delete_oddzial_glowny() przed operacją delete.

```
create trigger oddzial_glowny_deleter
before delete on projekt.oddzial_glowny
for each row execute procedure delete_oddzial_glowny();
```

4.13 Funkcja do generowania liczb losowych z przedziału [min, max]

4.13.1 random_generator()

Funkcja `random_generator()` generuje liczbę losową z przedziału [min, max].

```
create or replace function random_generator(minimum integer, maximum integer) returns
$$
begin
  return FLOOR(RANDOM() * (maximum - minimum + 1) + minimum);
end
$$
language plpgsql;
```

4.14 Funkcja do sprzedawania ryb danego rybaka w danym sklepie

4.14.1 sell_animals()

Funkcja `sell_animals()` sprzedaje zwierzęta danego rybaka w danym sklepie.

```
create or replace function sell_animals(id_rybak int, id_sklep varchar)
returns void as
$$
declare
record RECORD;
id_zwierze varchar;
cena_val numeric(10,2);
ilosc_val int;
sprzedane varchar;
begin
  if not exists (select 1 from projekt.rybak r where r.rybak_id = id_rybak) then
    raise exception 'Rybaka o id = % nie ma w bazie!!!', id_rybak;
    return;
  end if;

  if not exists (select 1 from projekt.rynek r where r.nazwa = id_sklep) then
    raise exception 'Rynku o nazwie = % nie ma w bazie!!!', id_sklep;
    return;
  end if;

  sprzedane := 'Sprzedane zwierzęta: ';

  for record in select zwierze from projekt.lista where rybak = id_rybak loop
    if exists (select 1 from projekt.rynek_zwierze where
    record.zwierze = zwierze and rynek = id_sklep) then
      select cena, zwierze into cena_val, id_zwierze from projekt.rynek_zwierze rz
      where record.zwierze = zwierze and rynek = id_sklep;
      select ilosc into ilosc_val from projekt.lista l
```

```
      where zwierze = record.zwierze and rybak = id_rybak;
      delete from projekt.lista where rybak = id_rybak and zwierze = record.zwierze;
      update projekt.rybak set stan_konta = stan_konta + cena_val * ilosc_val
      where rybak_id = id_rybak;
      sprzedane := sprzedane || record.zwierze || ', ';
    end if;
  end loop;

  -- raise exception '%', SUBSTRING(sprzedane FROM 1 FOR LENGTH(sprzedane) - 2);
  return;
end
$$
language plpgsql;
```

4.15 Funkcja do sprawdzania poprawności licencji rybaka

4.15.1 `check_rybak_licence()`

Funkcja `check_rybak_licence()` sprawdza poprawność licencji danego rybaka przez strażnika.

```
create or replace function check_rybak_licence(id_rybak int, id_straznik int)
returns void as
$$
declare
straznik_nazwa varchar;
rybak_nazwa varchar;
val varchar;
begin
  select r.imie || ' ' || r.nazwisko, s.imie || ' ' || s.nazwisko
  into rybak_nazwa, straznik_nazwa
  from projekt.straznik s, projekt.rybak r
  where s.straznik_id = id_straznik and r.rybak_id = id_rybak;

  select 'Strażnik ' || straznik_nazwa || ' sprawdza rybaka ' ||
  rybak_nazwa || ': ' into val;

  if (select licencja_id from projekt.rybak where rybak_id = id_rybak)
  is null then
    raise exception  '%Nie posiada on ważnej licencji!!!!', val;
  elsif (select data_startu from projekt.rybak r join projekt.licencja l
  using(licencja_id) where r.rybak_id = id_rybak) > CURRENT_DATE then
    raise exception '% Posiada licencję, ale jeszcze nie rozpoczęła się
    jej ważność!!!', val;
  elsif (select data_konca from projekt.rybak r join projekt.licencja l
  using(licencja_id) where r.rybak_id = id_rybak) < CURRENT_DATE then
    raise exception '%Licencja przeterminowana!!!', val;
  else
    raise exception '%Rybak posiada ważne dokumenty', val;
```

```
  end if;
  return;
end
$$
language plpgsql;
```

4.15.2 `rewrite_db()`

Funkcja `rewrite_db()` resetuje bazę do stanu początkowego w celu przywrócenia danych. Tablice asocjacyjne i licencja będą posiadały inne dane, ze względu na użycie funkcji `random_generator()` przy ich tworzeniu.

```
create or replace function rewrite_db() returns void as
$$
begin
-----usuniecie rekordow z bazy danych-----
delete from projekt.rybak;
delete from projekt.zwierze;
delete from projekt.straznik;
delete from projekt.lista;
delete from projekt.licencja;
delete from projekt.oddzial;
delete from projekt.oddzial_glowny;
delete from projekt.zbiornik;
delete from projekt.zwierze_zbiornik;
delete from projekt.rynek;
delete from projekt.rynek_zwierze;

-----wlaczenie wyzwalaczy uzytych do budowania bazy danych-----
alter table projekt.rybak enable trigger list_creator;
alter table projekt.zbiornik enable trigger legal_setter;
alter table projekt.rybak enable trigger licence_creator;
alter table projekt.zbiornik  enable trigger animal_adder

---... dodanie danych do tablicy

-----wylaczenie wyzwalaczy uzytych do budowania bazy danych-----
alter table projekt.rybak disable trigger list_creator;
alter table projekt.zbiornik disable trigger legal_setter;
alter table projekt.rybak disable trigger licence_creator;
alter table projekt.zbiornik  disable trigger animal_adder;
end
$$
language plpgsql;
```

5 Aplikacja klienta

Aplikacja została stworzona w Django. Do aplikacji mamy różne poziomy dostępu:

1. ADMIN - możliwość dodania do wszystkich tabel w bazie danych, wiersz poleceń umożliwiający wykonanie jakiegokolwiek polecenia
    - login: admin
    - hasło admin
2. RYBAK - możliwość dodania do niektórych tablic, wybrania danych przy użyciu odpowiednich poleceń, usunięcia niektórych danych, SPECIAL - specialna funkcja tylko dla tego użytkownika, pozwalająca na sprzedaż wszystkich ryb w danym sklepie
    - login: rybak
    - hasło kochampieski2
3. STRAZNIK - tak samo jak RYBAK, SPECIAL - możliwość sprawdzenia danego rybaka przez danego strażnika pod kątem posiadania i ważności licencji
    - login: rybak
    - hasło kochampieski2

System autoryzacji zapewnia różny dostęp do bazy danych.

5.1 Struktura Projektu

```
bazy-projekt-1/
    |-- bazy_danych/
        |-- bazy_danych/
        |-- bd1/
            |-- templates/
               |-- db/
                   |-- admin_bar.html
                   |-- delete.html
                   |-- insert.html
                   |-- show.html
                   |-- special.html
                   |-- select.html
                |-- main
                    |-- base.html
                    |-- dokumentacja.html
                    |-- erd.html
                    |-- home.html
                    |-- script.html
                |-- registration
                    |-- login.html
                    |-- sign_up.html
            |-- admin.py
            |-- apps.py
            |-- forms.py
            |-- urls.py
            |-- utils.py
            |-- views.py
        |-- manage.py
    |-- django/
    |-- README.md
    |-- script.sql
```

5.2 Działanie projektu w Django

5.3 utils.py

W tym pliku znajdują się 2 funkcje:

1. execute_raw_sql_query(query, params) - wykonuje query
2. group_checker(request, name) - sprawdza przynależność użytkownika do grupy

```
from django.db import connection
from django.contrib.auth.models import User, Group

def execute_raw_sql_query(query, params=None):
    with connection.cursor() as cursor:
        if params:
            cursor.execute(query, params)
        else:
            cursor.execute(query)

        # Fetch the results
        columns = [col[0] for col in cursor.description]
        results = [dict(zip(columns, row)) for row in cursor.fetchall()]

    return results

def group_checker(request, name):
    return request.user.groups.filter(name=name).exists()
```

5.4 urls.py

Specyfikacja adresów url dla projektu.

```
from django.urls import path
from . import views
from django.contrib.auth.views import LogoutView

urlpatterns =[
    path('', views.home, name='home'),
    path('home', views.home, name='home'),
    path('skrypt/', views.skrypt, name='skrypt'),
    path('own_query/', views.display_data, name='own_query'),
    path('dokumentacja/', views.dokumentacja, name='dokumentacja'),
    path('erd/', views.erd, name='erd'),
    path('sign-up/', views.sign_up, name='sign_up'),
    path('logout/', views.custom_logout, name='logout'),
    path('select/', views.select, name='select'),
    path('delete/', views.delete, name='delete'),
    path('insert/', views.insert, name='insert'),
    path('update/', views.update, name='update'),
    path('special/', views.special, name='special'),
    path('rewrite/', views.rewrite, name='rewrite'),
]
```

5.5 forms.py

Formularz Django, użyty przy rejestracji użytkownika

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from .models import *

class RegisterForm(UserCreationForm):
    email = forms.EmailField(required = False)

    class Meta:
        model = User
        fields = ["username", "email", "password1", "password2"]
```

5.6 views.py

Funkcje renderujące widok dla użytkownika, przykładowa funkcja generująca widok służący do usuwania danych z bazy:

```
def delete(request):
    if request.method == 'POST':
        if 'delete_rybak_imie' in request.POST:
            imie = request.POST['rybak_imie']
            query = f"delete from projekt.rybak where imie like '{str(imie)}%'"

        elif 'delete_rybak_ryby' in request.POST:
            ilosc = request.POST['rybak_ryby']
            query = f"delete from projekt.rybak where rybak_id in (select l.rybak from
            projekt.lista l group by l.rybak having sum(l.ilosc) > {str(ilosc)})"

        elif 'delete_rybak_wiek' in request.POST:
            wiek = request.POST['rybak_wiek']
            query = f"delete from projekt.rybak where wiek > {str(wiek)}"

        elif 'delete_lista_legalne' in request.POST:
            legalne = request.POST.get('lista_legalne', False)
            if legalne == 'on':
                legalne = True
            else:
                legalne = False
            query = f"delete from projekt.lista where zwierze in (select z.nazwa from projekt.zwierze
            z where z.legalna = {str(legalne)})"

        else:
            query = ''

        try:
            # Attempt to execute the raw SQL query using the function
            execute_raw_sql_query(query)

            # Pass the query and results to the template
            return render(request, 'db/delete.html', {'query': query, 'is_rybak':
            group_checker(request, 'rybak'), 'is_straznik': group_checker(request, 'straznik')})
        except Exception as e:
            # Handle exceptions (e.g., invalid SQL syntax)
            error_message = f"{str(e).split('CONTEXT')[0]}"
            if error_message == "'NoneType' object is not iterable":
                return render(request, 'db/delete.html', {'query': query, 'is_rybak':
                group_checker(request, 'rybak'),'is_straznik': group_checker(request, 'straznik')})
            return render(request, 'db/delete.html', {'query': query, 'error_message': error_message,
        'is_rybak': group_checker(request, 'rybak'), 'is_straznik': group_checker(request, 'straznik')}
    return render(request, 'db/delete.html', {'is_rybak': group_checker(request, 'rybak'),
                                              'is_straznik': group_checker(request, 'straznik')})
```

5.7 Formularze

Formularze mają postać:

```
<form method="post" action="{% url 'delete' %}">
    <div class="form-group">
        {% csrf_token %}
        <label for="rybak_imie">S owo:</label>
        <input type="text" class="form-control" name="rybak_imie" required>
        <br>
        <button type="submit" name="delete_rybak_imie" class="btn btn-primary">
            Usu  rybaka/rybak w
        </button>
    </div>
</form>
```

Wartość action jest polem 'name' zdefiniowanym w urls.py, dzięki czemu możemy generować dany widok zdefiniowany w views.py. Token csrf_token zabezpiecza przed atakami CSRF. Kiedy formularz zostanie wysłany, csrf_token wstawia ukryte pole ¡input do formularza, zawierające właściwy token CSRF. Gdy żądanie POST zostanie przesłane, Django sprawdza, czy token w zapytaniu jest zgodny z tokenem przypisanym do sesji

użytkownika. Jeśli tokeny się zgadzają, żądanie jest uznawane za autoryzowane. Przekazane parametry są przetwarzane w widoku, który zwraca render strony podanej jako drupi parametr funkcji render. Na przykładzie show.html:

```
{% if error_message %}
    <p class="h4" style="color:red">{{ error_message }}</p>
{% elif results %}
    <p class="h4" style="color:red">{{ query }}</p>
    <br>
    <table class="table table-dark">
        <thead>
            <tr>
                {% for key in results.0.keys %}
                    <th scope="col">{{ key }}</th>
                {% endfor %}
            </tr>
        </thead>
        <tbody>
            {% for row in results %}
                <tr>
                    {% for value in row.values %}
                        <td>{{ value }}</td>
                    {% endfor %}
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endif %}
```

Widzimy, że jeżeli polecenie SQL zakończyło się powodzeniem to renderowana jest część results, natomiast gdy wystąpił błąd na stronie zostanie on wyświetlony w error_message. Przetwarzanie formularzy w widoku polega na wyciągnięciu wartości z request i wykonaniu odpowiedniego polecenia SQL:

```
if request.method == 'POST':
    if 'delete_rybak_imie' in request.POST:
        imie = request.POST['rybak_imie']
        query = f"delete from projekt.rybak where imie like '{str(imie)}%'"
```

Przypadek wyżej pokazany służy do usuwania rybaka pod kątem matchingu jego imienia ze stringiem podanym w polu 'rybak_imie'.
Dane do bazy wprowadzamy w sposób ręczny, wpisując poszczególne dane w odpowiednie miejsca formularzy.

5.8 Obsługa aplikacji

Pod **tym linkiem** znajduje się repozytorium z zadaniem. Po zainstalowaniu wszystkich modułów zdefiniowanych w aplikacji i uruchomieniu jej przechodzimy pod adres podany w terminalu.
Poziomy dostępu udostepniają użytkownikom różne możliwośći:

1. Wspólne widoki (użytkownik niezalogowany):
    - Strona domowa
    - Podgląd skryptu SQL
    - Podgląd ERD
    - Dokumentacja
2. Wspólne widoki (użytkownik zalogowany):

- Strona domowa
- Podgląd skryptu SQL
- Podgląd ERD
- Dokumentacja
- INSERT
- Zresetuj dane w bazie - korzysta z funkcji `rewrite_db()`

3. Dodatkowe widoki dla ADMIN:
   - Polecenia własne - pełny dostęp do bazy danych
4. Dodatkowe widoki dla RYBAK:
   - SELECT
   - DELETE
   - SPECIAL - korzysta z funkcji `sell_animals()`
5. Dodatkowe widoki dla STRAZNIK
   - SELECT
   - DELETE
   - SPECIAL - korzysta z funkcji `check_rybak_licence()`

Przechodząc pod odpowiedni widok jesteśmy w stanie wykonywać różne operacje na bazie danych, wprowadzając dane do formularzy.

6 Wykaz literatury

W projekcie korzystałem z następujących źródeł:

1. Dokumentacja Postgresql
2. Dokumentacja Django