1. ***Проанализировать бизнес-логику приложения и создать структуру базы данных, которая может использоваться для хранения данных этого приложения. В базе данных должно быть минимум десять таблиц. Если таблиц получается более двадцати то рекомендуется ограничиться частью функционала приложения и не превышать это количество. В качестве отчета по этой части проекта необходимо приложить команды создания таблиц***.

// Устанавливаем Postgresql на LINUX

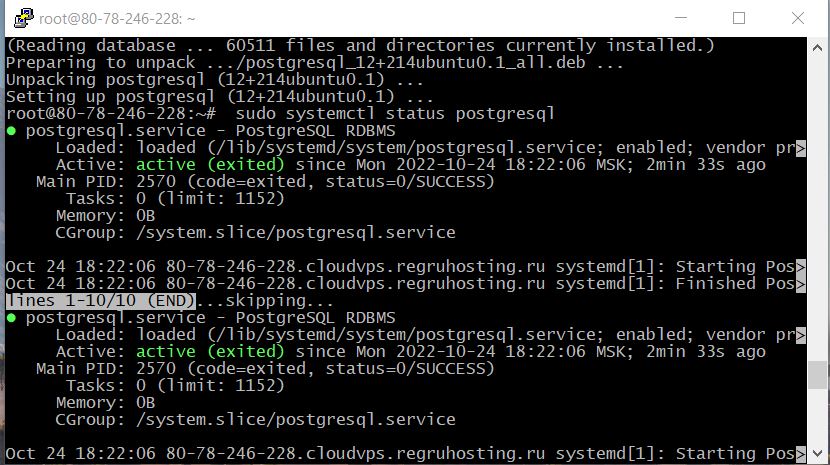
$ sudo apt update

$ sudo apt upgrade

$ sudo apt install postgresql

// Проверяем статус Postgresql

$ sudo systemctl status postgresql



// Переключаемся на пользователя postgres

$ sudo su - postgres

// Проверяем версию консольного клиента

$ psql --version

// Запускаем консольного клиента

$ psql

// Создаем пользователя

postgres=# CREATE USER my\_admin WITH PASSWORD 'adminsecurity';

// Создаем базу данных

postgres=# CREATE DATABASE mybase;

// Предоставляем все права пользователю my\_admin на базу mybase

postgres=# GRANT ALL PRIVILEGES ON DATABASE mybase to my\_admin;

// Проверка подключения к новой базе новым пользователем

$ psql -Umy\_admin mybase -h127.0.0.1

// Создаем таблицы базы данных

CREATE TABLE users(

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(120) NOT NULL UNIQUE,

phone VARCHAR(15) UNIQUE,

main\_photo\_id INT,

created\_at TIMESTAMP );

CREATE TABLE messages (

id SERIAL PRIMARY KEY,

from\_user\_id INT NOT NULL,

to\_user\_id INT NOT NULL,

body TEXT,

is\_important BOOLEAN,

is\_delivered BOOLEAN,

created\_at TIMESTAMP );

CREATE TABLE friendship (

id SERIAL PRIMARY KEY,

requested\_by\_user\_id INT NOT NULL,

requested\_to\_user\_id INT NOT NULL,

status\_id INT NOT NULL,

requested\_at TIMESTAMP,

confirmed\_at TIMESTAMP );

CREATE TABLE friendship\_statuses (

id SERIAL PRIMARY KEY,

name VARCHAR(30) UNIQUE );

CREATE TABLE photo (

id SERIAL PRIMARY KEY,

url VARCHAR(250) NOT NULL UNIQUE,

owner\_id INT NOT NULL,

uploaded\_at TIMESTAMP NOT NULL,

size INT NOT NULL );

CREATE TABLE bots\_type (

id SERIAL PRIMARY KEY,

name VARCHAR(30) UNIQUE );

CREATE TABLE bots (

id SERIAL PRIMARY KEY,

name VARCHAR(30) UNIQUE ,

type\_id INT NOT NULL,

owner\_id INT NOT NULL,

created\_at TIMESTAMP);

CREATE TABLE one\_sensor\_bots\_data (

id SERIAL PRIMARY KEY,

bot\_id INT NOT NULL,

temp INT ,

press INT,

created\_at TIMESTAMP);

CREATE TABLE two\_sensor\_bots\_data (

id SERIAL PRIMARY KEY,

bot\_id INT NOT NULL,

temp\_out INT ,

press INT,

temp\_in INT ,

humid\_air INT,

created\_at TIMESTAMP);

CREATE TABLE three\_sensor\_bots\_data (

id SERIAL PRIMARY KEY,

bot\_id INT NOT NULL,

temp\_out INT ,

press INT,

temp\_in INT ,

humid\_air INT,

humid\_soil INT,

photo\_id INT,

created\_at TIMESTAMP);

1. ***Используя генератор тестовых данных, заполнить созданную БД данными в количестве минимум сто строк для тех таблиц, где это имеет смысл. Доработать данные запросами если это необходимо. В качестве отчёта приложить дамп БД с данными.***

Дамп БД по ссылке:

*https://github.com/Kuldyaev/postgresql\_GB/blob/main/mybase.dump.sql*

1. ***Создать внешние ключи, если они не были созданы на шаге 1 в командах создания таблиц. В качестве отчета приложить команды создания внешних ключей.***

ALTER TABLE users

ADD CONSTRAINT users\_main\_photo\_id\_fk

FOREIGN KEY (main\_photo\_id)

REFERENCES photo (id);

ALTER TABLE messages

ADD CONSTRAINT messages\_from\_user\_id\_fk

FOREIGN KEY (from\_user\_id)

REFERENCES users (id);

ALTER TABLE messages

ADD CONSTRAINT messages\_to\_user\_id\_fk

FOREIGN KEY (to\_user\_id)

REFERENCES users (id)

ON DELETE CASCADE;

ALTER TABLE photo

ADD CONSTRAINT owner\_id\_user\_id\_fk

FOREIGN KEY (owner\_id)

REFERENCES users (id);

ALTER TABLE friendship

ADD CONSTRAINT requested\_by\_user\_id\_fk

FOREIGN KEY (requested\_by\_user\_id)

REFERENCES users (id);

ALTER TABLE friendship

ADD CONSTRAINT requested\_to\_user\_id\_fk

FOREIGN KEY (requested\_to\_user\_id)

REFERENCES users (id);

ALTER TABLE friendship

ADD CONSTRAINT status\_id\_friendship\_statuses\_id\_fk

FOREIGN KEY (status\_id)

REFERENCES friendship\_statuses (id);

ALTER TABLE bots

ADD CONSTRAINT bots\_type\_id\_fk

FOREIGN KEY (type\_id)

REFERENCES bots\_type (id);

ALTER TABLE bots

ADD CONSTRAINT owner\_id\_user\_id\_fk

FOREIGN KEY (owner\_id)

REFERENCES users (id);

ALTER TABLE one\_sensor\_bots\_data

ADD CONSTRAINT one\_sens\_bot\_id\_bots\_id\_fk

FOREIGN KEY (bot\_id)

REFERENCES bots (id);

ALTER TABLE two\_sensor\_bots\_data

ADD CONSTRAINT two\_sens\_bot\_id\_bots\_id\_fk

FOREIGN KEY (bot\_id)

REFERENCES bots (id);

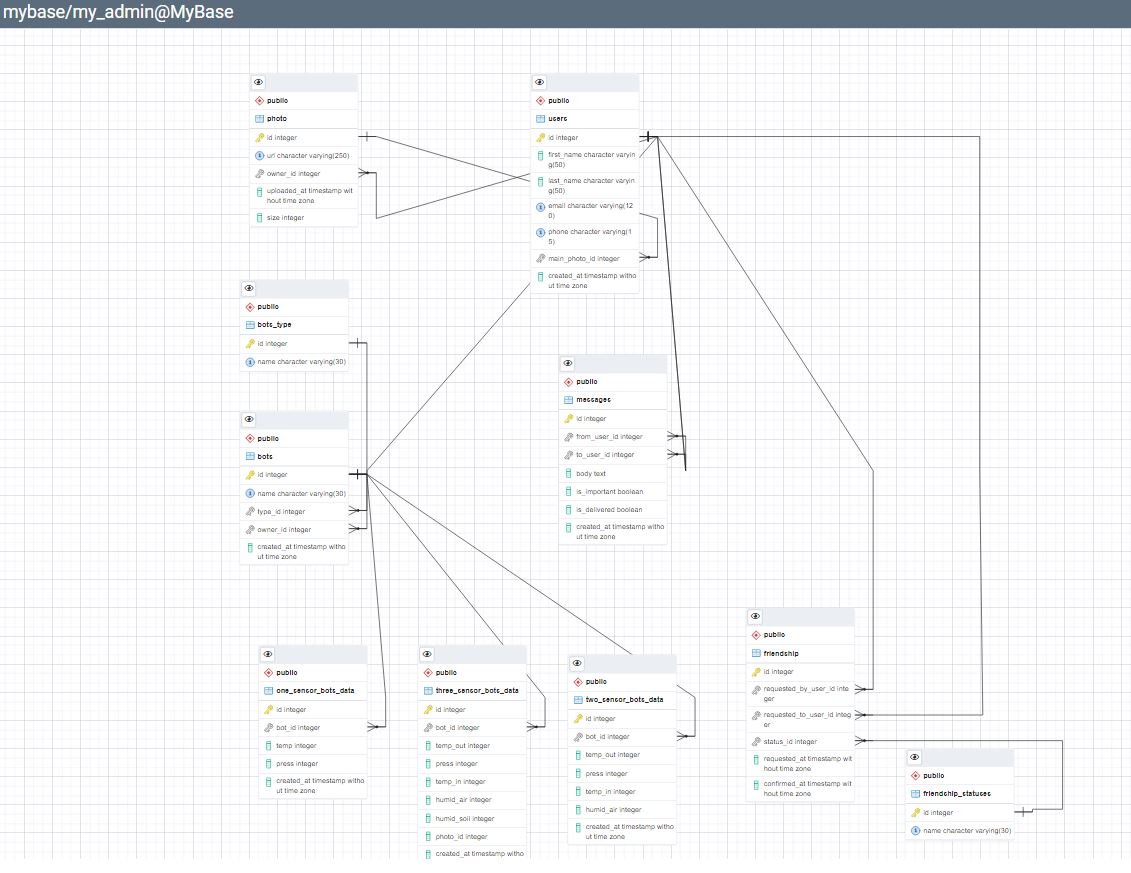
ALTER TABLE three\_sensor\_bots\_data

ADD CONSTRAINT three\_sens\_bot\_id\_bots\_id\_fk

FOREIGN KEY (bot\_id)

REFERENCES bots (id);

1. ***Создать диаграмму отношений. В качестве отчета приложить файл изображения диаграммы отношений***



https://github.com/Kuldyaev/postgresql\_GB/blob/main/ERD.JPG

1. ***Создать два сложных (многотабличных) запроса с использованием подзапросов***.

SELECT last\_name,

first\_name,

(SELECT COUNT(messages.id) FROM messages WHERE messages.from\_user\_id = users.id) AS messages\_number

FROM users

WHERE (SELECT COUNT(messages.id) FROM messages WHERE messages.from\_user\_id = users.id) = (SELECT

(SELECT COUNT(messages.id) FROM messages WHERE messages.from\_user\_id = users.id) AS messages\_number

FROM users

ORDER BY messages\_number DESC LIMIT 1);

SELECT

photo.id,

url

FROM photo

WHERE owner\_id = (SELECT id FROM users WHERE first\_name = 'Viacheslav' AND last\_name ='Kuldyaev');

1. ***Создать два сложных запроса с использованием объединения JOIN и без использования подзапросов***

SELECT

messages.id,

users.first\_name AS first\_name,

users.last\_name AS last\_name,

messages.body AS messages\_text

FROM messages

JOIN users

ON users.id = messages.from\_user\_id

ORDER BY last\_name;

SELECT

users.last\_name AS last\_name,

users.first\_name AS first\_name,

photo.url AS photo\_url,

photo.size AS photo\_size

FROM users

JOIN photo

ON users.id = photo.owner\_id

ORDER BY last\_name;

1. ***Создать два представления, в основе которых лежат сложные запросы***
2. ***Создать пользовательскую функцию.***

CREATE FUNCTION max\_quantity\_messages ()

RETURNS BIGINT AS

$$

SELECT

(SELECT COUNT(messages.id) FROM messages WHERE messages.from\_user\_id = users.id)

AS messages\_number

FROM users

ORDER BY messages\_number DESC

LIMIT 1;

$$

LANGUAGE SQL;

1. ***Создать триггер.***

##### CREATE OR REPLACE FUNCTION check\_new\_user\_name\_func()

##### RETURNS TRIGGER AS

##### $$

##### BEGIN

##### IF EXISTS (SELECT id FROM users WHERE last\_name = NEW.last\_name AND first\_name = NEW.first\_name) THEN

##### RAISE EXCEPTION 'USER IS NOT PHOTO OWNER';

##### END IF;

##### RETURN NEW;

##### END

##### $$

##### LANGUAGE PLPGSQL;

##### CREATE TRIGGER check\_new\_user\_name\_trigger BEFORE INSERT ON users

##### FOR EACH ROW

##### EXECUTE FUNCTION check\_new\_user\_name\_func();

1. ***Для одного из запросов, созданных в пункте 6, провести оптимизацию. В качестве отчета приложить планы выполнения запроса, ваш анализ и показать действия, которые улучшили эффективность запроса***