Федеральное государственное автономное образовательное учреждение высшего образования Университет ИТМО

Факультет программной инженерии и компьютерных технологий

«Информационные системы и базы данных»

Отчёт по Курсовой работе

Этап 3

Управление умным домом

Студенты:

Ляо Ихун

Скакун Артем Андреевич

Гр. P33131

Преподаватель:

Байрамова Хумай Бахруз Кызы

Этап 3

Текст задачи:

Реализовать даталогическую модель в реляционной СУБД PostgreSQL:

• Создать необходимые объекты базы данных.

• Заполнить созданные таблицы тестовыми данными.

• Сделать скрипты для:

◦ создания/удаления объектов базы данных;

◦ заполнения/удаления созданных таблиц.

• Обеспечить целостность данных при помощи средств языка DDL.

• Добавить в базу данных триггеры для обеспечения комплексных ограничений целостности.

• Реализовать функции и процедуры на основе описания бизнес-процессов (из этапа

№1).

• Произвести анализ использования созданной базы данных:

◦ выявить наиболее часто используемые запросы к объектам базы данных;

◦ результаты представить в виде текстового описания.

• Создать индексы и доказать, что они полезны для вашей базы данных:

◦ доказательство должно быть приведено в виде текстового описания.

Бизнес-Процессы

1. Регистрация пользователя.

```
create or replace function register(name_u varchar(64),password_u varchar(256), age_u int,
gender_u gender,phone_u varchar(64),email_u varchar(128)) returns setof "user" as $$
    DECLARE
        new_id int;
    BEGIN
        insert into "user"(password, gender, name, age)  values (password_u, gender_u,
name_u, age_u);
        select id from "user" order by id desc limit 1 into new_id;
        insert into contact(user_id, email, phone) values (new_id, email_u, phone_u);
        return query select * from "user" order by id desc limit 1;
end;
$$ language plpgsql;
```

2. Авторизация пользователя.

```
create or replace function login_by_phone(phone_u varchar(64),password_u varchar(256))
returns bool as $$
BEGIN
    if (select count(*) from "user" inner join contact c on "user".id = c.user_id where
"user".password = password_u and phone = phone_u) = 1 then
        return true;
    else
     return false;
    end if;
end;
$$ language plpgsql;
create or replace function login_by_email(email_u varchar(128),password_u varchar(256))
returns bool as $$
BEGIN
    if (select count(*) from "user" inner join contact c on "user".id = c.user_id where
"user".password = password_u and email = email_u) = 1 then
        return true;
    else
        return false;
    end if;
end;
$$ language plpgsql;
select login_by_email('       0@gmail.com', '481.6490150315751');
```

3. Пользователь добавляет себя в дом в приложение.
Пользователь просто присоединяется к другому дому.

```
create or replace function add_user_to_home(user_id_u int, home_id int) returns setof
list_user_house as $$
BEGIN
   insert into list_user_house(user_id, house_id) VALUES (user_id_u, home_id);
   return query select * from list_user_house order by id desc limit 1;
end;
$$ language plpgsql;
```

4. Добавление комнат в дом. Пользователь добавляет комнаты в квартиру, и данные о них.

```
create or replace function add_room_to_home(house_id_u int, square_u float,height_u float, t
room_type) returns setof room as $$
BEGIN
   insert into room(house_id, square, height, type) VALUES (house_id_u, square_u, height_u,
t);
   return query select * from room order by id desc limit 1;
end;
$$ language plpgsql;
```

5. Добавление умных вещей в помещение. Пользователь добавляет умные вещи в комнату.

```
create or replace function add_furniture_to_room(room int, manu varchar(64), avai bool,ft
furniture_type) returns setof furniture as $$
BEGIN
   insert into furniture(room_id, manufacture, Available, type) VALUES (room, manu, avai, ft);
   return query select * from furniture order by id desc limit 1;
end;
$$ language plpgsql;
```

6. Создание сценария по условию - умная вещь начинает работать, если выполняется какое-то условие (например, увлажнитель начинает работать, если влажность воздуха ниже нормы). Сценарий создан пользователем или программистом.

```
create or replace function add_condition_script(con text, name varchar(256)) returns
setof script as $$
   DECLARE
      new_id int = 0;
   BEGIN
   insert into script(creator_name, type) VALUES (name, 'CONDITIONAL');
   select script.id from script order by id desc limit 1 into new_id;
   insert into condition_script(script_id, condition) VALUES (new_id, con);
   return query select * from script order by id desc limit 1;
end;
$$ language plpgsql;
```

7. Создание сценария по расписанию - работа умных вещей по заданному расписанию. Сценарий создан пользователем или программистом.

```
create or replace function add_schedule_script(st time,et time, name varchar(256))
returns setof script as $$
DECLARE
   new_id int = 0;
BEGIN
   insert into script(creator_name, type) VALUES (name, 'SCHEDULE');
   select script.id from script order by id desc limit 1 into new_id;
   insert into schedule_script(script_id, end_time,start_time) VALUES (new_id, et,st);
   return query select * from script order by id desc limit 1;
end;
$$ language plpgsql;
```

8. Пользователь может поделиться сценариями с другими пользователями.

```
create or replace function add_user_script(ui int , si int) returns setof list_script_user as $$
BEGIN
   insert into list_script_user(script_id, user_id) VALUES (si,ui);
   return query select * from list_script_user order by id desc limit 1;
end;
$$ language plpgsql;
```

9. Пользователь управляет состоянием вещей при помощи сценариев.
10. Пользователь управляет состоянием вещей вручную.
11. Пользователь сообщают о проблемах, служба поддержки их рашают

```
create or replace function add_report(ui int , si int, des text, type_p problem_type)
returns setof list_script_user as $$
BEGIN
   insert into problem(user_id, support_man_id, description, type) VALUES
(ui,si,des,type_p);
   return query select * from problem order by id desc limit 1;
end;
$$ language plpgsql;
```

Создание БД

```
CREATE TYPE PROBLEM_TYPE AS ENUM ('UI', 'BUGS', 'SCRIPT');
CREATE TYPE HOUSE_TYPE AS ENUM ('APARTMENTS', 'VILLAS', 'HIGH-
END','ORDINARY');
CREATE TYPE FURNITURE_TYPE AS ENUM ('AIR_CONDITION','LIGHT', 'HUMIDIFIER',
'BATHTUB', 'OUTLET','CURTAINS', 'FAN', 'CAMERA', 'WATER_HEATER');
```

```
CREATE TYPE ACTION_TYPE AS ENUM
('CLOSE','OPEN','SWITCH_OFF','SWITCH_ON','ADJUST_VALUE','TURN_ON','TURN_OFF')
;
CREATE TYPE SCRIPT_TYPE AS ENUM ('CONDITIONAL','SCHEDULE');
CREATE TYPE COUNTRY AS ENUM('US','UK','RUSSIAN','CHINA','FRANCE');
create type city as enum('Shanghai', 'Beijing', 'Shenzhen', 'Guangzhou', 'Chengdu','Paris',
'Marseille', 'Lyon', 'Toulouse','Cambridge', 'Edinburgh',  'London', 'Liverpool','New York', 'Los
Angeles', 'Chicago', 'Boston');
create type room_type as enum('KITCHEN','BEDROOM','BATHROOM','LIVING');
create type gender as enum ('MAN','WOMAN');
CREATE TABLE if not exists "user"(
   id serial primary key not null ,
   password varchar(256) not null ,
   gender gender not null ,
   name varchar(64) not null,
   age int not null check ( age > 0 )
);
CREATE TABLE if not exists Support_man(
   id serial primary key not null ,
   password varchar(256) not null,
   name varchar(64) not null ,
   is_free boolean not null DEFAULT True
);
create table if not exists Problem(
   id serial primary key not null ,
   user_id int not null REFERENCES  "user"(id),
   support_man_id int not null REFERENCES Support_man(id),
   is_finished boolean not null DEFAULT false,
   description text not null,
   type problem_type not null,
   Data date default current_date
);
create table if not exists address(
   id serial not null primary key ,
   country country not null ,
   city city not null ,
   street varchar(128) not null
);
create table if not exists House(
   id serial primary key not null ,
   address_id int references address(id) not null ,
   type house_type not null
);
create table if not exists Room(
   id serial primary key not null ,
   house_id int not null references House(id),
   square float not null,
   height float not null,
   type room_type not null,
   is_filled boolean not null default false
);
create table if not exists furniture(
   id serial primary key not null ,
   room_id int references Room(id) not null ,
   manufacture varchar(258) not null ,
   Available boolean not null default True,
   type furniture_type not null
);
create table if not exists action(
   id serial primary key not null ,
   type_furniture furniture_type not null ,
   type action_type not null ,
   description text
);
create table if not exists script(
   id serial primary key not null ,
   creator_name varchar(256) not null,
   type script_type not null
);
create table if not exists schedule_script(
   script_id int references script(id) not null ,
   start_time time not null ,
   end_time time not null
);
create table if not exists Condition_script(
   script_id int references script(id) not null ,
   condition text not null
);
create table if not exists contact(
   user_id int references "user"(id) not null unique,
   email varchar(128) not null unique ,
   phone varchar(64) not null unique
);
create table if not exists List_Action_Script(
   id serial not null primary key ,
   script_id int references script(id),
   action_id int references action(id)
);
create table if not exists list_script_user(
   id serial not null primary key ,
   script_id int references script(id),
   user_id int references "user"(id),
   unique (script_id,user_id)
);
create table if not exists list_user_house(
   id serial not null primary key ,
   user_id int references "user"(id),
   house_id int references House(id),
   unique (user_id,house_id)
);
```

Заполнение БД

```
create or replace function fill_user() returns setof "user" as $$
DECLARE
   i int = 0;
   names varchar(64)[] = array ['Peter','Bob','John','Tomas','Alex','Anna'];
BEGIN
   while i < 1000000 loop
         insert into "user"(name,gender,password,age) values
(names[i%6+1],'MAN',random()*1000,28);
         i = i + 1;
      end loop;
   return query select * from "user" limit 500;
end;
$$ language plpgsql;
select fill_user();

create or replace function fill_support() returns setof Support_man as $$
   DECLARE
      i int = 0;
```

```
        names varchar(64)[] = array ['Peter','Bob','John','Tomas','Alex','Anna'];
    BEGIN
        while i < 1000 loop
            insert into Support_man(password, name) VALUES
(floor(random()*1000000),names[i%6+1]);
            i = i + 1;
            end loop;
        return query select * from Support_man;
    end;
    $$ language plpgsql;
select fill_support();
select count(*) from support_man;

create or replace function fill_problem() returns setof problem as $$
DECLARE
    i int = 0;
BEGIN
    while i < 300000 loop
            insert into problem(user_id, support_man_id, description,is_finished, type) values
(i%1000000 + 1,i%1000 + 1,'test test','true','BUGS');
            i = i + 1;
        end loop;
    return query select * from problem limit 500;
end;
$$ language plpgsql;
select fill_problem();

create or replace function fill_address() returns setof address as $$
DECLARE
    i int = 0;
    countries country[] = array ['US','UK','RUSSIAN','CHINA','FRANCE'];
    cities city[] = array ['Shanghai', 'Beijing', 'Shenzhen', 'Guangzhou', 'Chengdu','Paris', 'Marseille',
'Lyon', 'Toulouse','Cambridge', 'Edinburgh',  'London', 'Liverpool','New York', 'Los Angeles',
'Chicago', 'Boston'];
BEGIN
    while i < 1000000 loop
            insert into address(country, city, street) VALUES (countries[i%5+1],cities[i%17+1],'xx
streest');
            i = i + 1;
        end loop;
    return query select * from address limit 500;
end;
$$ language plpgsql;
select fill_address();

create or replace function fill_house() returns setof address as $$
DECLARE
    i int = 0;
    houses house_type[] = array ['APARTMENTS', 'VILLAS', 'HIGH-END','ORDINARY'];
BEGIN
    while i < 1000000 loop
            insert into house(address_id, type) VALUES (i+1,houses[i%4+1]);
            i = i + 1;
        end loop;
    return query select * from address limit 500;
end;
$$ language plpgsql;
select fill_house();

create or replace function fill_room() returns setof address as $$
DECLARE
    i int = 0;
    rooms room_type[] = array ['KITCHEN','BEDROOM','BATHROOM','LIVING'];
```

```
BEGIN
    while i < 1000000 loop
            insert into room(house_id, square, height,type) VALUES (i+1,19,3,rooms[i%4+1]),
                                                                   (i+1,22,3,rooms[i%4+1]),
                                                                   (i+1,22,3,rooms[i%4+1]);
            i = i + 1;
        end loop;
    return query select * from address limit 500;
end;
$$ language plpgsql;
select fill_room();

create or replace function fill_furniture() returns setof furniture as $$
DECLARE
    i int = 0;
    ft furniture_type[] = array ['AIR_CONDITION','LIGHT', 'HUMIDIFIER', 'BATHTUB',
'OUTLET','CURTAINS', 'FAN', 'CAMERA', 'WATER_HEATER'];
BEGIN
    while i < 3000000 loop
            insert into furniture(room_id, manufacture, type) values
(i+1,'xiaomi',ft[i%9+1]),(i+1,'xiaomi',ft[i%9+1]);
            i = i + 1;
        end loop;
    return query select * from furniture limit 500;
end;
$$ language plpgsql;
select fill_furniture();

create or replace function fill_action() returns setof action as $$
DECLARE
    i int = 0;
    ft furniture_type[] = array ['AIR_CONDITION','LIGHT', 'HUMIDIFIER', 'BATHTUB',
'OUTLET','CURTAINS', 'FAN', 'CAMERA', 'WATER_HEATER'];
BEGIN
    while i < 9 loop
            insert into action(type_furniture, type, description) values (ft[i%9+1],'TURN_ON','Test'),
(ft[i%9+1],'TURN_OFF','Test')
                                                                        ,  (ft[i%9+1],'SWITCH_OFF','Test'),
(ft[i%9+1],'SWITCH_OFF','Test');
            i = i + 1;
        end loop;
    return query select * from action limit 500;
end;
$$ language plpgsql;
select fill_action();

create or replace function fill_contact() returns setof contact as $$
DECLARE
    i int = 0;
BEGIN
    while i < 1000000 loop
            insert into contact(user_id, email, phone)  VALUES
(i+1,to_char(i,'9999999')||'@gmail.com',to_char(i,'99999999')) ;
            i = i + 1;
        end loop;
    return query select * from contact limit 500;
end;
$$ language plpgsql;
select fill_contact();

create or replace function fill_script() returns setof script as $$
DECLARE
    i int = 0;
```

```
   st script_type [] = array ['CONDITIONAL','SCHEDULE'];
BEGIN
   while i < 10000000 loop
         if i < 5000000 then
            insert into script(creator_name, type)  VALUES ('Tom',st[1]) ;
         elsif i < 10000000 then
            insert into script(creator_name, type)  VALUES ('Tom',st[2]) ;
         end if;
         i = i + 1;
      end loop;
   return query select * from script limit 500;
end;
$$ language plpgsql;
select fill_script();

create or replace function fill_condition_script() returns setof condition_script as $$
DECLARE
   i int = 0;
BEGIN
   while i < 5000000 loop
         insert into condition_script(script_id, condition)  VALUES (i+1,'click button') ;
         i = i + 1;
      end loop;
   return query select * from condition_script limit 500;
end;
$$ language plpgsql;
select fill_condition_script();

create or replace function fill_schedule_script() returns setof schedule_script as $$
DECLARE
   i int = 0;
BEGIN
   while i < 5000000 loop
         insert into schedule_script(script_id, start_time,end_time)  VALUES
(i+5000001,'8:00','9:00') ;
         i = i + 1;
      end loop;
   return query select * from schedule_script limit 500;
end;
$$ language plpgsql;
select fill_schedule_script();

create or replace function fill_list_action_script() returns setof list_action_script as $$
DECLARE
   i int = 0;
BEGIN
   while i < 10000000 loop
         insert into list_action_script(script_id, action_id) values (i+1,1),(i+1,2),(i+1,3);
         i = i + 1;
      end loop;
   return query select * from list_action_script limit 500;
end;
$$ language plpgsql;
select  fill_list_action_script();

create or replace function fill_list_user_house() returns setof list_user_house as $$
DECLARE
   i int = 0;
BEGIN
   while i < 1000000 loop
         insert into list_user_house(user_id, house_id) values (i + 1,i+1);
         i = i + 1;
      end loop;
```

```
    return query select * from list_user_house limit 500;
end;
$$ language plpgsql;
select  fill_list_user_house();

create or replace function fill_list_script_user() returns setof list_script_user as $$
DECLARE
    i int = 0;
BEGIN
    while i < 1000000 loop
            insert into list_script_user(user_id, script_id) values (i + 1,i*10+1),(i + 1,i*10+2),(i +
1,i*10+3),(i + 1,i*10+4),(i + 1,i*10+5),(i + 1,i*10+6),(i + 1,i*10+7),(i + 1,i*10+8),(i + 1,i*10+9),(i +
1,i*10+10);
            i = i + 1;
        end loop;
    return query select * from list_script_user limit 500;
end;
$$ language plpgsql;
select  fill_list_script_user();
```

Создание триггеров

```
CREATE OR REPLACE FUNCTION new_problem() RETURNS TRIGGER AS
$before_insert_problem_trigger$
BEGIN
    if ((select count(*) from problem where support_man_id = NEW.support_man_id and
Problem.is_finished = false) >= 5) THEN
        raise exception 'You cannot dispatch this problem to a busy support man';
    ELSIF ((select count(*) from problem where support_man_id = NEW.support_man_id and
Problem.is_finished = false) >= 4) THEN
        update Support_man set is_free = false where Support_man.id = NEW.support_man_id;
    end if;
    Return NEW;
end;
$before_insert_problem_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER before_insert_problem_tri BEFORE INSERT ON Problem FOR EACH
ROW EXECUTE PROCEDURE new_problem();

CREATE OR REPLACE FUNCTION after_update_problem() RETURNS TRIGGER AS
$after_update_problem_trigger$
BEGIN
    if((select count(*) from problem where support_man_id = new.support_man_id and
Problem.is_finished = false)>=5) THEN
        update Support_man set is_free = false where Support_man.id = NEW.support_man_id;
    ELSIF((select count(*) from problem where support_man_id = new.support_man_id and
Problem.is_finished = false)<5) then
        update Support_man set is_free = TRUE where Support_man.id =
NEW.support_man_id;
    end if;
    if((select count(*) from problem where support_man_id = old.support_man_id and
Problem.is_finished = false)>=5) THEN
        update Support_man set is_free = false where Support_man.id = old.support_man_id;
    ELSIF((select count(*) from problem where support_man_id = old.support_man_id and
Problem.is_finished = false)<5) then
        update Support_man set is_free = TRUE where Support_man.id = old.support_man_id;
    end if;
    return new;
end;
```

```
$after_update_problem_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER after_update_problem_tri AFTER UPDATE ON Problem FOR EACH
ROW EXECUTE PROCEDURE after_update_problem();

CREATE OR REPLACE FUNCTION before_update_problem() RETURNS TRIGGER AS
$before_update_problem_trigger$
BEGIN
   if((select count(*) from problem where support_man_id = new.support_man_id and
Problem.is_finished = false)>=5  and new.is_finished = false) THEN
      raise exception 'Dispatch a problem to a busy support man is forbidden';
   end if;
   return new;
end;
$before_update_problem_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER before_update_problem_tri BEFORE UPDATE ON Problem FOR EACH
ROW EXECUTE PROCEDURE before_update_problem();


CREATE OR REPLACE FUNCTION new_furniture() returnS TRIGGER AS
$insert_furniture_trigger$
BEGIN
   IF ((select count(*) from furniture where room_id = new.room_id) >= 10) THEN
      raise exception 'Room is filled.';
   ELSIF ((select count(*) from furniture where room_id = new.room_id) >= 9) THEN
      update room set is_filled = true where room.id = new.room_id;
   end if;
   Return new;
end ;
$insert_furniture_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER insert_furniture_tri BEFORE INSERT ON furniture FOR EACH ROW
EXECUTE PROCEDURE new_furniture();
```

Удаление таблиц

```
drop table list_script_user cascade ;
drop table list_user_house cascade ;
drop table List_Action_Script cascade ;
drop table contact cascade ;
drop table schedule_script cascade ;
drop table Condition_script cascade ;
drop table script cascade ;
drop table action cascade ;
drop table furniture cascade;
drop table Room cascade ;
drop table House cascade ;
drop table address cascade ;
drop table Problem cascade ;
drop table Support_man cascade ;
drop table "user" cascade ;
drop type room_type;
DROP TYPE action_type;
DROP TYPE script_type;
DROP TYPE problem_type;
DROP TYPE house_type;
DROP TYPE furniture_type;
Drop Type city;
drop type COUNTRY;
```

Удаление объектов

```
truncate table list_script_user cascade ;
truncate table list_user_house cascade ;
truncate table List_Action_Script cascade ;
truncate table contact cascade ;
truncate table schedule_script cascade ;
truncate table Condition_script cascade ;
truncate table script cascade ;
truncate table action cascade ;
truncate table furniture cascade;
truncate table Room cascade ;
truncate table House cascade ;
truncate table address cascade ;
truncate table Problem cascade ;
truncate table Support_man cascade ;
truncate table "user" cascade ;
```

Удаление индексов

```
Drop index list_action_script_index;
Drop index schedule_script_index;
Drop index condition_script_index;
DROP INDEX furniture_index;
drop index problem_index;
```

Удаление триггеров

```
drop trigger before_insert_problem_tri on problem;
drop trigger after_update_problem_tri on problem;
drop trigger before_update_problem_tri on problem;
drop trigger insert_furniture_tri on furniture;
```

Создание идексов

```
CREATE INDEX problem_index ON problem (support_man_id);
Create Index furniture_index on furniture(room_id);
create index schedule_script_index on schedule_script(script_id);
create index condition_script_index on condition_script(script_id);
create index list_action_script_index on list_action_script(script_id);
```

Наиболее часто используемые запросы к объектам базы данных

1. Insert into script, insert into list_script_user, insert into list_action_script, insert into condition_script, insert into schedule_script,select * from script.
   Скрипт – это главная функция нашего приложения. Пользователи или программисты создают скрипт и пользователи их используют.Высшие запросы все для создания скриптов

2. Insert into problem, update problem set, select count(*) from problem where problem.support_man_id = ?? and problem.is_finished = false .

   Здесь собирает все информации о проблемах и требованиях пользователей.

3. select script_id,condition from script inner join Condition_script Cs on script.id = Cs.script_id where id = ???

   Найти информацию какого-то скрипта с условией.

4. select * from script inner join schedule_script ss on script.id = ss.script_id where id = ????

   Найти информацию какого-то скрипта расписания.

5. select * from script inner join List_Action_Script LAS on script.id = LAS.script_id where script.id = ???;

   Найти все действии какого-то скрипта

Доказание полезности индесков

1. problem_index
выполнять запрос 'Select * from problem where support_man_id = 123’ 3 раза и сравнять время с и без индеска

С	89 ms	98 ms	81 ms
Без	110 ms	118 ms	111 ms

выполнять запрос 'Insert into problem’ 3 раза и сравнять время с и без индеска

С	52 ms	52 ms	50 ms
Без	106 ms	101 ms	89s

2. furniture_index
выполнять запрос Insert into furniture(room_id,manufacture,available,type) values(‘123’,’smart life’,’t’,’LIGHT’); 3 раза и сравнять время с и без индекса.

С	55 ms	55 ms	65 ms
Без	517 ms	520ms	522 ms

3. condition_script_index
   select script_id,condition from script inner join Condition_script Cs on script.id = Cs.script_id where id = 1212321

С	51 ms	44 ms	42 ms
Без	433 ms	428 ms	417ms

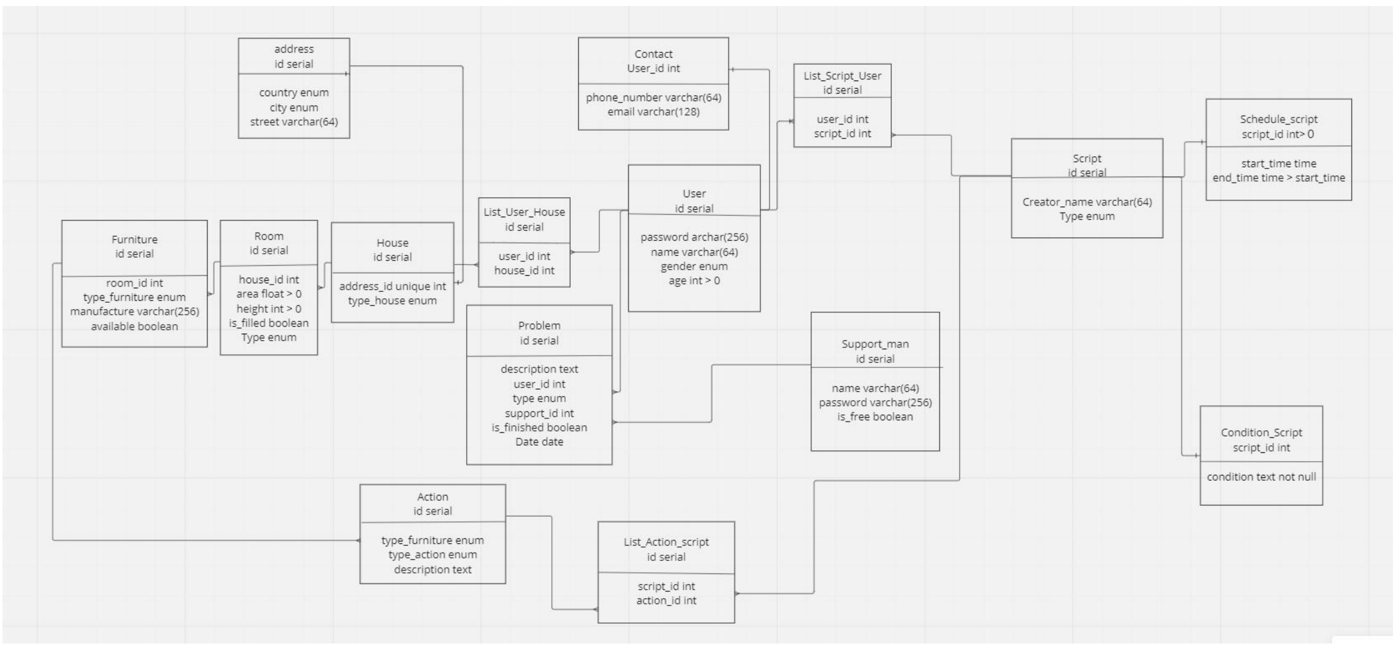
4. schedule_script_index
   select * from script inner join schedule_script ss on script.id = ss.script_id where id = 1234227;

С	37 ms	34 ms	43 ms
Без	418 ms	441 ms	420 ms

5. List_action_script_index
   select * from script inner join List_Action_Script LAS on script.id = LAS.script_id where script.id = 199999;

С	37 ms	34 ms	35 ms
Без	3 s 487 ms	3 s 458 ms	3 s 467 ms

Даталогическую модель



Вывод:

Все таблицы созданы при помощи разных органичений, чтобы они выполняют наши требования. Четыре триггера создано для завершения целостность данных. Пять идексов создано для оптимизации скорости работы БД. Они все помагают использовать эту БД и соответстуют тому, что описан в этапе 1.