**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;

1. 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');

1. 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;

1. 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;

1. 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;

1. 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;

1. 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;

1. 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;

1. Пользователь управляет состоянием вещей при помощи сценариев.
2. Пользователь управляет состоянием вещей вручную.
3. Пользователь сообщают о проблемах, служба поддержки их рашают
4. create or replace function *add\_report*(ui int , si int, des text, type\_p problem\_type) returns setof list\_script\_user as $$
5. BEGIN
6. insert into problem(user\_id, support\_man\_id, description, type) VALUES (ui,si,des,type\_p);
7. return query select \* from problem order by id desc limit 1;
8. end;
9. $$ 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

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

* + - 1. 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.

* + - 1. 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.

* + - 1. select \* from script inner join schedule\_script ss on script.id = ss.script\_id where id = ????

Найти информацию какого-то скрипта расписания.

* + - 1. 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 | 1. s |

1. 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 |

1. 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 |

1. schedule\_script\_index

select \* from script inner join schedule\_script ss on script.id = ss.script\_id where id = 1234227;

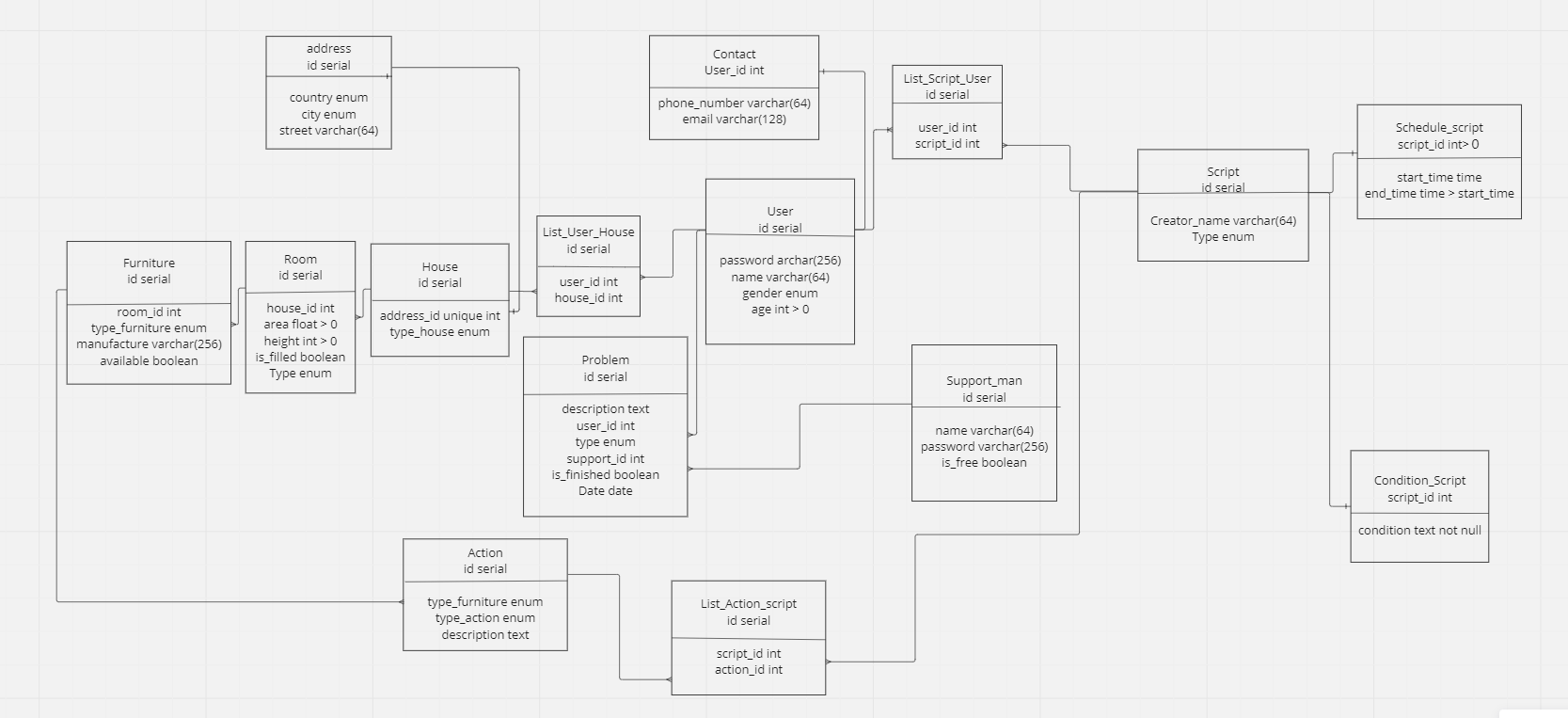
|  |  |  |  |
| --- | --- | --- | --- |
| С | 37 ms | 34 ms | 43 ms |
| Без | 418 ms | 441 ms | 420 ms |

1. 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.