НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

**Мегафакультет:** Компьютерных технологий и управления

**Направление:** 09.03.04 «Программная инженерия»

**Курсовая работа. Этап 3**

По дисциплине:

«Информационные системы и базы данных»

На тему:

«Реализация ER-модели Базы Данных»

**Выполнили:**

студентки группы **P33112**

Маньшина Елена

Витальевна

Корнишова Евгения

Александровна

**Преподаватель:**

**Харитонова Анастасия**

**Евгеньевна**

г. Санкт-Петербург

2020 г.

**Текст задания**

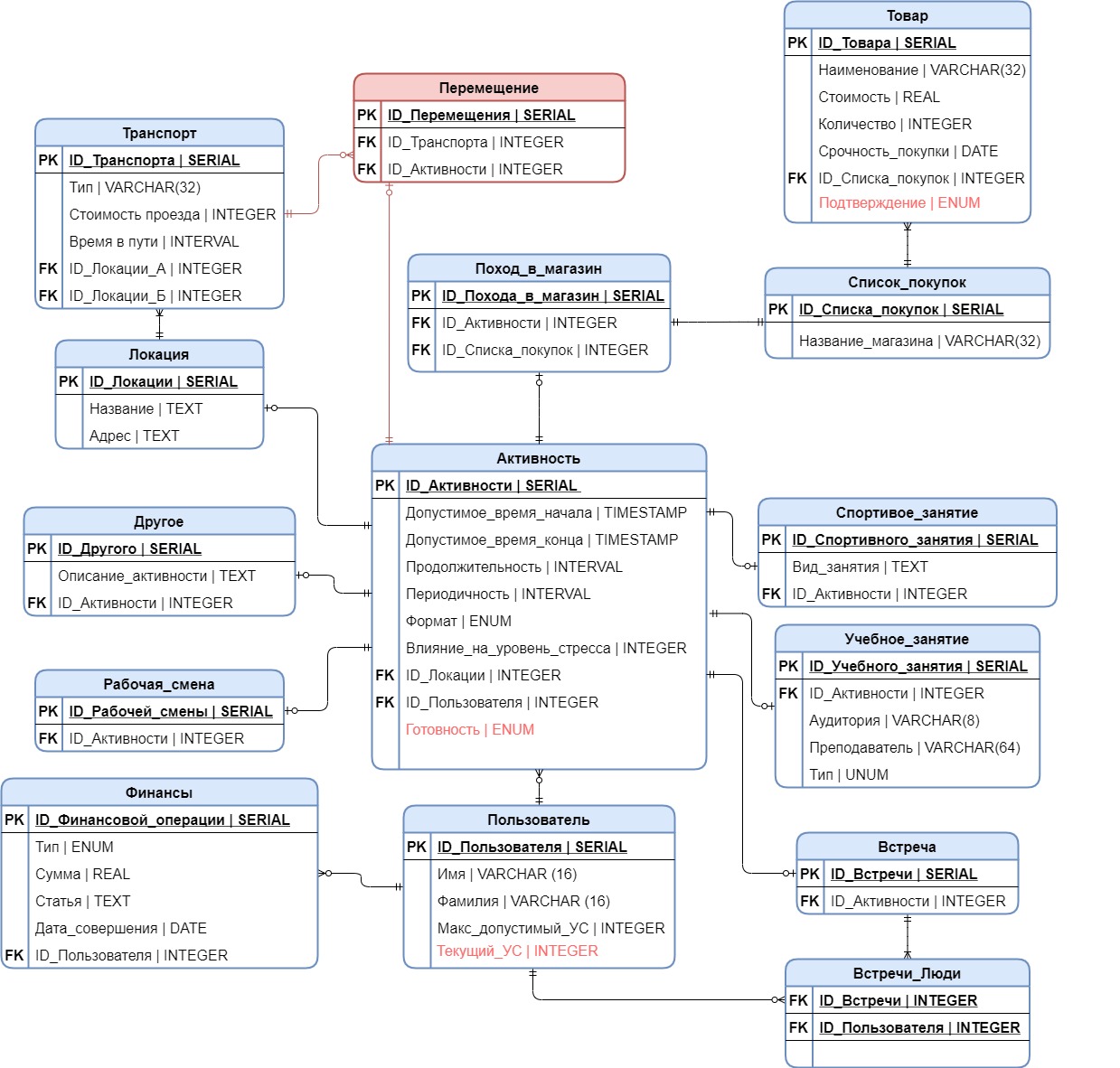
●Реализовать ER-модель в реляционной СУБД PostgreSQL с использованием языков SQL и PL/pgSQL.

● Разработать скрипт для заполнения таблиц наборами данных.

● Обеспечить целостность данных в базе.

● Реализовать процедуры, функции для изменения данных в БД, которые могут быть вызваны из командной строки.

● Реализовать средства для оптимизации времени выполнения наиболее часто используемых запросов к разработанной базе данных.



**Реализация ER-модель с обеспечением целостности данных в БД.**

create type format\_enum as enum ('очный', 'дистанционный');

create type isDone\_enum as enum ('выполнено', 'не выполнено');

create type lesson\_type\_enum as enum ('лекция', 'практика');

create type isConfirmed\_enum as enum ('подтвержден', 'не подтвержден');

create type finance\_type\_enum as enum ('доход', 'расход');

CREATE TABLE пользователь

(

id\_пользователя SERIAL PRIMARY KEY,

имя varchar(16) NOT NULL,

фамилия varchar(16) NOT NULL,

макс\_допустимый\_ус INTEGER NOT NULL CHECK (макс\_допустимый\_ус >= 0),

текущий\_ус INTEGER NOT NULL CHECK (текущий\_ус >= 0)

);

CREATE TABLE список\_покупок

(

id\_списка\_покупок SERIAL PRIMARY KEY,

название\_магазина varchar(32) NOT NULL

);

CREATE TABLE локация

(

id\_локации SERIAL PRIMARY KEY,

название TEXT NOT NULL,

адрес TEXT NOT NULL

);

CREATE TABLE активность

(

id\_активности SERIAL PRIMARY KEY,

допустимое\_время\_начала TIMESTAMP NOT NULL,

допустимое\_время\_конца TIMESTAMP NOT NULL,

продолжительность INTERVAL NOT NULL,

периодичность INTERVAL,

формат format\_enum,

влияние\_на\_уровень\_стресса INTEGER NOT NULL,

готовность isDone\_enum,

id\_локации INTEGER NOT NULL REFERENCES локация ON DELETE SET NULL,

id\_пользователя INTEGER NOT NULL REFERENCES пользователь ON DELETE CASCADE

CONSTRAINT checkInterval

**CHECK(**

**периодичность IS NULL**

**AND допустимое\_время\_конца - продолжительность >= CURRENT\_TIMESTAMP**

**AND допустимое\_время\_конца > допустимое\_время\_начала**

**AND допустимое\_время\_конца - допустимое\_время\_начала >= продолжительность**

**OR (периодичность IS NOT NULL**

**AND допустимое\_время\_конца::time > допустимое\_время\_начала::time**

**AND допустимое\_время\_конца::time - допустимое\_время\_начала::time < периодичность**

**AND допустимое\_время\_конца::time - допустимое\_время\_начала::time >= продолжительность**

**)**

**)**

);

CREATE TABLE учебное\_занятие

(

id\_учебного\_занятия SERIAL PRIMARY KEY,

аудитория VARCHAR(8),

преподаватель VARCHAR(64) NOT NULL,

тип lesson\_type\_enum,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE поход\_в\_магазин

(

id\_похода\_в\_магазин SERIAL PRIMARY KEY,

id\_списка\_покупок INTEGER NOT NULL REFERENCES список\_покупок ON DELETE CASCADE,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE рабочая\_смена

(

id\_рабочей\_смены SERIAL PRIMARY KEY,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE спортивное\_занятие

(

id\_спортивного\_занятия SERIAL PRIMARY KEY,

вид\_занятия TEXT NOT NULL,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE встреча

(

id\_встречи SERIAL PRIMARY KEY,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE встречи\_люди

(

id\_встречи INTEGER NOT NULL REFERENCES встреча ON DELETE CASCADE,

id\_пользователя INTEGER NOT NULL REFERENCES пользователь ON DELETE CASCADE,

PRIMARY KEY (id\_встречи, id\_пользователя)

);

CREATE TABLE другое

(

id\_другого SERIAL PRIMARY KEY,

описание\_активности TEXT NOT NULL,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

CREATE TABLE товар

(

id\_товара SERIAL PRIMARY KEY,

наименование VARCHAR(32) NOT NULL,

стоимость REAL CHECK (стоимость > 0) NOT NULL,

количество INTEGER CHECK (количество > 0) NOT NULL,

срочность\_покупки DATE NOT NULL,

подтверждение isConfirmed\_enum,

id\_списка\_покупок INTEGER NOT NULL REFERENCES список\_покупок ON DELETE CASCADE

);

CREATE TABLE финансы

(

id\_финансовой\_операции SERIAL PRIMARY KEY,

тип finance\_type\_enum,

сумма REAL CHECK (сумма > 0) NOT NULL,

статья TEXT NOT NULL,

дата\_совершения DATE NOT NULL,

id\_пользователя INTEGER NOT NULL REFERENCES пользователь ON DELETE CASCADE

);

CREATE TABLE транспорт

(

id\_транспорта SERIAL PRIMARY KEY,

тип VARCHAR(32) NOT NULL,

стоимость\_проезда INTEGER CHECK (стоимость\_проезда > 0) NOT NULL,

время\_в\_пути INTERVAL NOT NULL,

id\_локации\_а INTEGER NOT NULL REFERENCES локация ON DELETE CASCADE,

id\_локации\_б INTEGER NOT NULL REFERENCES локация ON DELETE CASCADE

);

CREATE TABLE перемещение

(

id\_перемещения SERIAL PRIMARY KEY,

id\_транспорта INTEGER NOT NULL REFERENCES транспорт ON DELETE CASCADE,

id\_активности INTEGER NOT NULL REFERENCES активность ON DELETE CASCADE

);

DROP TABLE транспорт CASCADE;

DROP TABLE финансы CASCADE;

DROP TABLE учебное\_занятие CASCADE;

DROP TABLE поход\_в\_магазин CASCADE;

DROP TABLE рабочая\_смена CASCADE;

DROP TABLE спортивное\_занятие CASCADE;

DROP TABLE встречи\_люди CASCADE;

DROP TABLE встреча CASCADE;

DROP TABLE другое CASCADE;

DROP TABLE товар CASCADE;

DROP TABLE список\_покупок CASCADE;

DROP TABLE активность CASCADE;

DROP TABLE пользователь CASCADE;

DROP TABLE локация CASCADE;

DROP TABLE перемещение CASCADE;

**Триггеры**

**-- триггер чтобы нельзя было запланировать встречу с самим собой**

CREATE OR REPLACE FUNCTION isMeetingCorrect() RETURNS TRIGGER AS

$$

DECLARE организатор\_встречи SMALLINT;

BEGIN

организатор\_встречи = (SELECT пользователь.id\_пользователя from встречи\_люди

join встреча on встреча.id\_встречи = NEW.id\_встречи

join активность on активность.id\_активности = встреча.id\_активности

join пользователь on пользователь.id\_пользователя = активность.id\_пользователя

GROUP BY пользователь.id\_пользователя);

IF (NEW.id\_пользователя = организатор\_встречи) THEN

RAISE EXCEPTION 'Невозможно запланировать встречу с самим собой';

ELSE

RETURN NEW;

END IF;

END;

$$ LANGUAGE plpgsql;

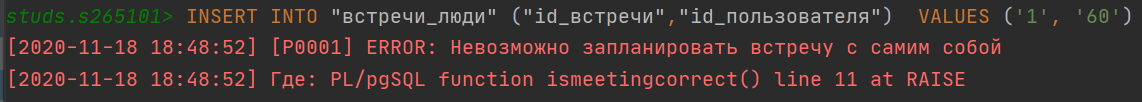
CREATE TRIGGER isMeetingCorrect

BEFORE INSERT OR UPDATE

ON встречи\_люди

FOR EACH ROW

EXECUTE PROCEDURE isMeetingCorrect();



**-- если занятие проходит очно, то обязательно должна быть указана аудитория**

CREATE OR REPLACE FUNCTION hasLessonRoom() RETURNS TRIGGER AS

$$

DECLARE формат format\_enum;

BEGIN

формат = (SELECT активность.формат from активность

join учебное\_занятие уз on активность.id\_активности = NEW.id\_активности

group by активность.формат);

IF (NEW.аудитория IS NULL and формат='очный') THEN

RAISE EXCEPTION 'Очное занятие можно добавить лишь с указанием аудитории';

ELSE

RETURN NEW;

END IF;

END;

$$ LANGUAGE plpgsql;

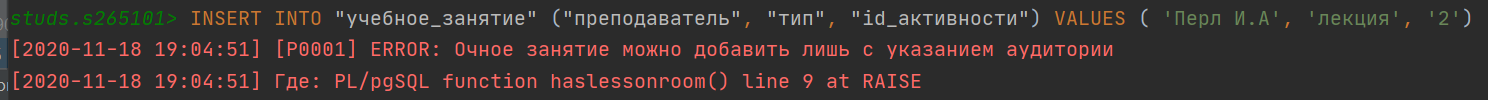
CREATE TRIGGER hasLessonRoom

BEFORE INSERT OR UPDATE

ON учебное\_занятие

FOR EACH ROW

EXECUTE PROCEDURE hasLessonRoom();



**-- при подтверждении покупки создается новая запись в таблице финансы**

CREATE TRIGGER make\_purchase

AFTER UPDATE OF подтверждение ON товар

FOR EACH ROW

EXECUTE PROCEDURE confirm\_purchase();

CREATE OR REPLACE FUNCTION confirm\_purchase() RETURNS trigger AS

$$

DECLARE user\_id INTEGER;

BEGIN

user\_id = (SELECT id\_пользователя FROM товар JOIN поход\_в\_магазин USING (id\_списка\_покупок) JOIN активность USING (id\_активности) WHERE id\_товара = NEW.id\_товара);

INSERT INTO финансы (тип, сумма, статья, дата\_совершения, id\_пользователя) VALUES ('расход', NEW.стоимость \* New.количество, NEW.наименование, CURRENT\_DATE, user\_id);

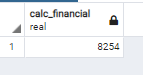
RETURN NULL;

END

$$ LANGUAGE plpgsql;

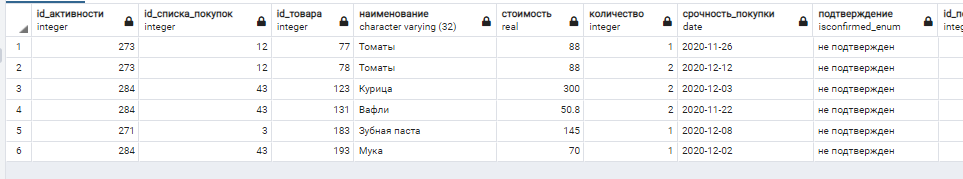
Проверяем сумму расходов пользователя 1 до покупки товаров:

select calc\_financial(1, 'расход');



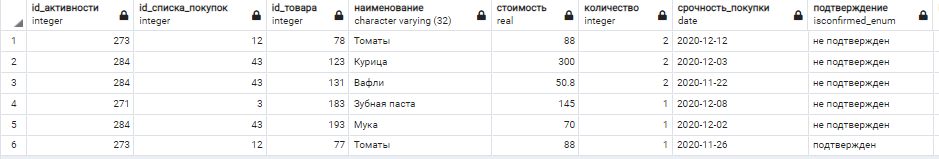
Смотрим, какие товары хочет купить пользователь:

select \* from товар join поход\_в\_магазин using (id\_списка\_покупок) join активность using (id\_активности) where id\_пользователя=1;



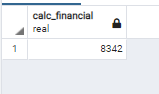
Подтверждаем покупку товара 77:

UPDATE товар SET "подтверждение" = 'подтвержден' WHERE id\_товара = 77;



Проверяем, что сумма расходов пользователя изменилась:

select calc\_financial(1, 'расход');



**-- при подтверждении выполнения активности изменяется текущий уровень стресса**

CREATE TRIGGER perform\_act

AFTER UPDATE OF готовность ON активность

FOR EACH ROW

EXECUTE PROCEDURE confirm\_action();

CREATE OR REPLACE FUNCTION confirm\_action() RETURNS trigger AS

$$

BEGIN

IF NEW.готовность = 'не выполнено' THEN

RETURN NULL;

END IF;

UPDATE пользователь SET текущий\_ус = текущий\_ус+NEW.влияние\_на\_уровень\_стресса WHERE id\_пользователя = NEW.id\_пользователя;

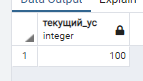
RETURN NULL;

END

$$ LANGUAGE plpgsql;

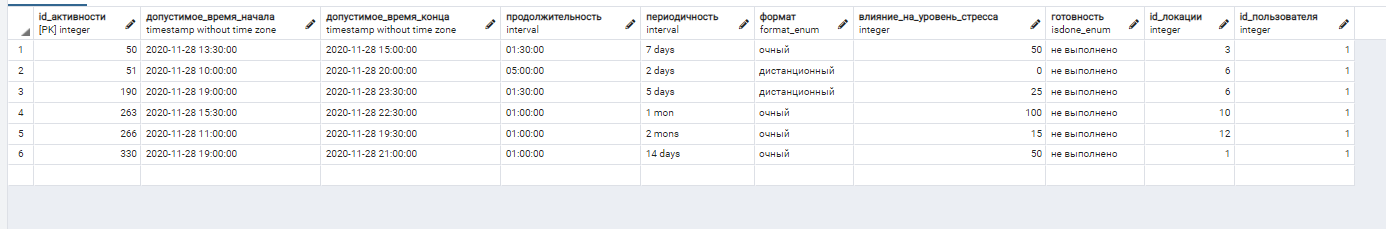
Проверяем текущий уровень стресса пользователя 1:

select текущий\_ус from пользователь where id\_пользователя = 1;



Просматриваем список активностей пользователя 1:

select \* from активность where id\_пользователя = 1;

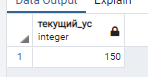


Подтверждаем выполнение активности пользователем 1:

UPDATE активность SET готовность = 'выполнено' where id\_активности=50;

Убеждаемся, что уровень стресса пользователя 1 изменился:

select текущий\_ус from пользователь where id\_пользователя = 1;



**Функции**

**-- подсчитывает сумму расходов/доходов пользователя за определенный период**

CREATE OR REPLACE FUNCTION calc\_financial(user\_id INTEGER, type finance\_type\_enum, from\_date Date = null, to\_date Date = null) RETURNS REAL AS

$$

BEGIN

IF (from\_date IS null AND to\_date IS null) THEN

RETURN (SELECT sum(сумма) from финансы where id\_пользователя = user\_id AND тип = type);

ELSIF (from\_date IS null AND to\_date IS NOT null) THEN

RETURN (SELECT sum(сумма) from финансы where id\_пользователя = user\_id AND тип = type AND дата\_совершения <= to\_date);

ELSIF (from\_date IS NOT null AND to\_date IS null) THEN

RETURN (SELECT sum(сумма) from финансы where id\_пользователя = user\_id AND тип = type AND дата\_совершения >= from\_date);

ELSE

RETURN (SELECT sum(сумма) from финансы where id\_пользователя = user\_id AND тип = type AND дата\_совершения >= from\_date AND дата\_совершения <= to\_date);

END IF;

END

$$ LANGUAGE plpgsql;

Пример работы в примере работы триггера make\_purchase

**-- рассчитывает предполагаемую стоимость похода в магазин на основе списка  
-- покупок**

CREATE OR REPLACE FUNCTION shopping\_costs(shopping\_id INTEGER) RETURNS REAL AS

$$

BEGIN

RETURN (SELECT sum(товар.количество \* товар.стоимость) FROM товар JOIN поход\_в\_магазин USING (id\_списка\_покупок) WHERE id\_похода\_в\_магазин = shopping\_id);

END

$$ LANGUAGE plpgsql;





**-- выводит баланс пользователя**

CREATE OR REPLACE FUNCTION check\_balance(user\_id INTEGER) RETURNS REAL AS

$$

DECLARE income REAL;

DECLARE expenses REAL;

BEGIN

income = (SELECT calc\_financial(user\_id, 'доход'));

expenses = (SELECT calc\_financial(user\_id, 'расход'));

RETURN (income - expenses);

END

$$ LANGUAGE plpgsql;





**-- выводит список дел, которые пользователь может выполнить сегодня, начиная -- с наиболее срочных.**

CREATE OR REPLACE FUNCTION daily\_activity\_list(user\_id INTEGER, day DATE) RETURNS TABLE(допустимое\_время\_начала timestamp, допустимое\_время\_конца timestamp, продолжительность interval, периодичность interval, формат format\_enum, влияние\_на\_уровень\_стресса integer, локация text) AS

$$

BEGIN

RETURN QUERY SELECT активность.допустимое\_время\_начала, активность.допустимое\_время\_конца, активность.продолжительность, активность.периодичность, активность.формат, активность.влияние\_на\_уровень\_стресса, локация.название

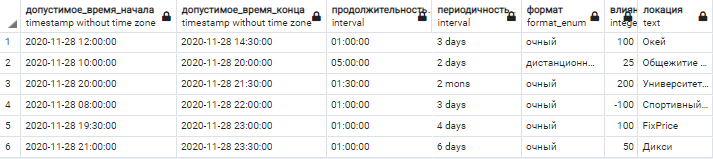
FROM активность JOIN локация USING (id\_локации)

WHERE активность.id\_пользователя = user\_id   
AND DATE(активность.допустимое\_время\_начала) <= day   
AND DATE(активность.допустимое\_время\_конца) >= day ORDER BY активность.допустимое\_время\_конца;

END

$$ LANGUAGE plpgsql;





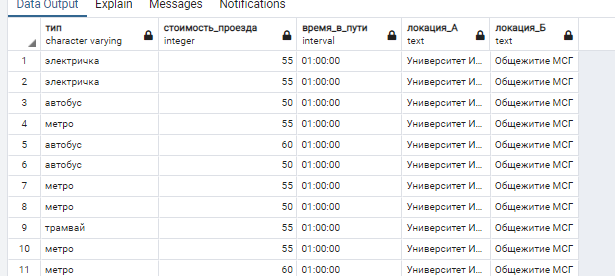
**-- показывает транспорт необходимый для перемещения от активности А до   
-- активности Б**CREATE OR REPLACE FUNCTION transport\_for\_activities(id\_activity\_A INTEGER, id\_activity\_B INTEGER) RETURNS TABLE (тип VARCHAR(32), стоимость\_проезда INTEGER, время\_в\_пути INTERVAL, локация\_А TEXT, локация\_Б TEXT) AS   
$$  
DECLARE id\_location\_A INTEGER;  
DECLARE id\_location\_B INTEGER;   
 BEGIN  
 id\_location\_A = (SELECT id\_локации FROM активность where id\_активности = id\_activity\_A);  
 id\_location\_B = (SELECT id\_локации FROM активность where id\_активности = id\_activity\_B);

IF id\_location\_A = id\_location\_B THEN  
 RETURN;  
 END IF;

RETURN QUERY (  
 SELECT транспорт.тип, транспорт.стоимость\_проезда, транспорт.время\_в\_пути, locationA.название AS локация\_А, locationB.название AS локация\_Б  
 FROM транспорт   
 JOIN локация AS locationA ON транспорт.id\_локации\_а = locationA.id\_локации  
 JOIN локация AS locationB ON транспорт.id\_локации\_б = locationB.id\_локации  
 WHERE транспорт.id\_локации\_а = id\_location\_A AND транспорт.id\_локации\_б = id\_location\_B);  
 END

$$ LANGUAGE plpgsql;





**Индексы**

**CREATE INDEX** "ИД\_пользователя" **ON** "пользователь" **USING BTREE** ("id\_пользователя");

**CREATE INDEX** "ИД\_пользователя\_активности" **ON** "активность" **USING BTREE** ("id\_пользователя");

**CREATE INDEX** "ИД\_списка\_покупок\_магазина" **ON** "поход\_в\_магазин" **USING BTREE** ("id\_списка\_покупок");

**CREATE INDEX** "ИД\_списка\_покупок\_товара" **ON** "товар" **USING BTREE** ("id\_списка\_покупок");

**CREATE INDEX** "Тип\_финансов" **ON** "финансы" **USING HASH** ("тип");

**CREATE INDEX** "ИД\_локации\_А" **ON** "транспорт" **USING BTREE** ("id\_локации\_а");

**CREATE INDEX** "ИД\_локации\_Б" **ON** "транспорт" **USING BTREE** ("id\_локации\_б");

**CREATE INDEX** "допустимое\_время\_начала\_активности" **ON** "активность" **USING BTREE** ("допустимое\_время\_начала");

**CREATE INDEX** "допустимое\_время\_конца\_активности" **ON** "активность" **USING BTREE** ("допустимое\_время\_конца");

**Вывод**: в ходе выполнения третьего этапа курсовой работы мы реализовали ER-модель в реляционной СУБД PostgreSQL, разработали скрипт для заполнения таблиц наборами данных, обеспечили целостность данных в базе, а так же реализовать процедуры и функции для изменения данных в БД, которые могут быть вызваны из командной строки, оптимизировали временя выполнения наиболее часто используемых запросов.