УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.01 Информатика и вычислительная техника

Дисциплина «Базы данных»

Лабораторная работа №3

Вариант 3404

Студент

Баянов Р. Д.

P3134

Преподаватель

Перцев Т.

Санкт-Петербург

2023 г.

Оглавление

Задание

Для отношений, полученных при построении предметной области из лабораторной работы №1, выполните следующие действия:

- Опишите функциональные зависимости для отношений полученной схемы (минимальное множество);
- Приведите отношения в 3NF (как минимум). Постройте схему на основеNF (как минимум).
- Опишите изменения в функциональных зависимостях, произошедшие после преобразования в 3NF (как минимум). Постройте схему на основеNF;
- Преобразуйте отношения в BCNF. Докажите, что полученные отношения представлены в BCNF. Если ваша схема находится уже в BCNF, докажите это;
- Какие денормализации будут полезны для вашей схемы? Приведите подробное описание.

Придумайте триггер и связанную с ним функцию, относящиеся к вашей предметной области, согласуйте их с преподавателем и реализуйте на языке PL/pgSQL.

Вариант

Внезапно вибрация пола приобрела совершенно иной характер. Странный экипаж замедлял движение -- это было несомненно! Время, видимо, бежало быстрее, чем казалось Олвину. Он глянул на табло и несколько удивился -- надпись гласила: <Лиз. 23 минуты>.

Описание предметной области

Вероятно, человек в подземном туннеле на экипаже с вибрирующим полом едет в город Лиз. И на табло видит время, оставшееся до прибытия.

Список сущностей

Стержневые

- Человек – имя, фамилия, возраст
- Город - имя
- Экипаж – имя, количество мест

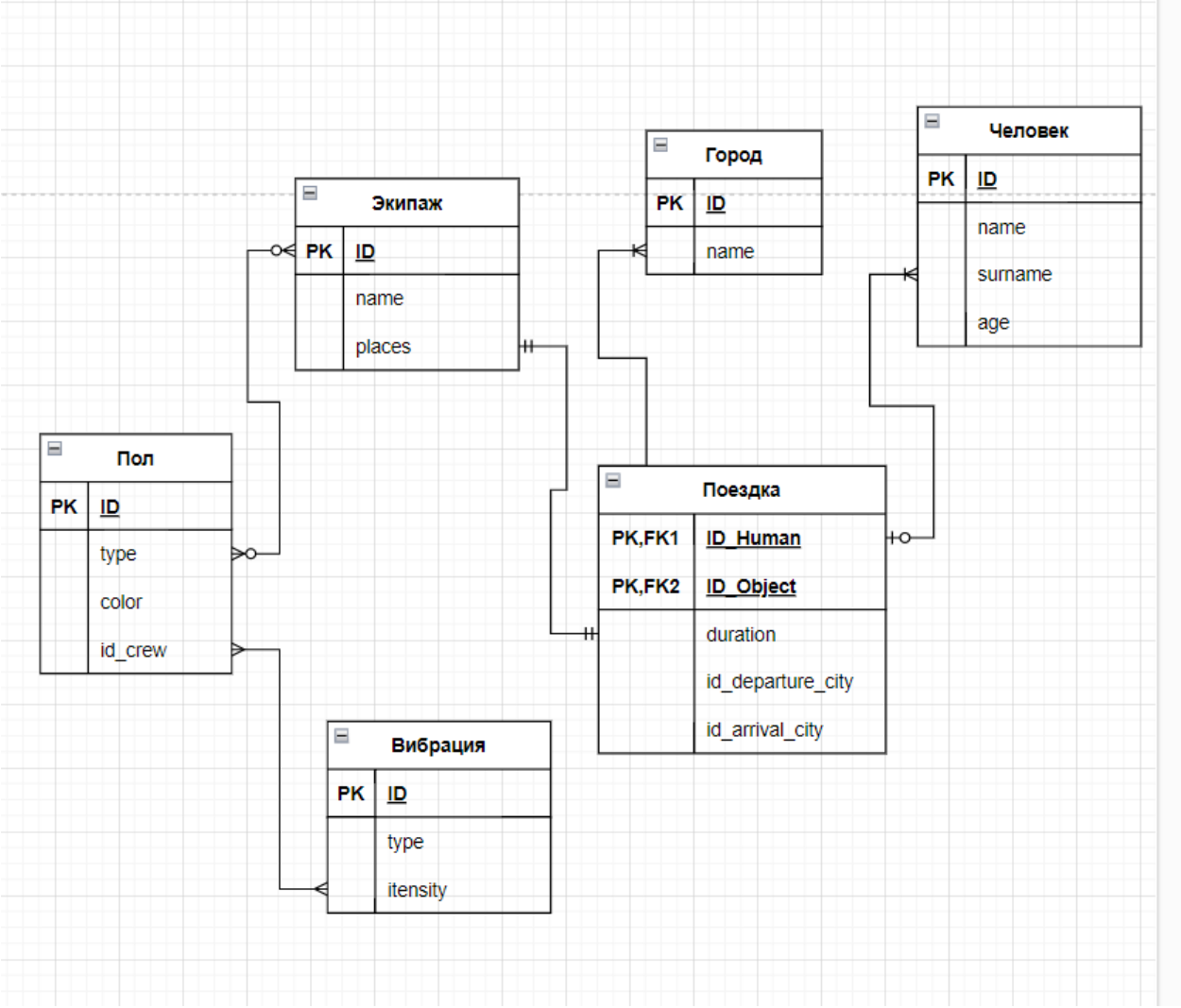
Характеристические

- Пол – тип, к какому экипажу относится

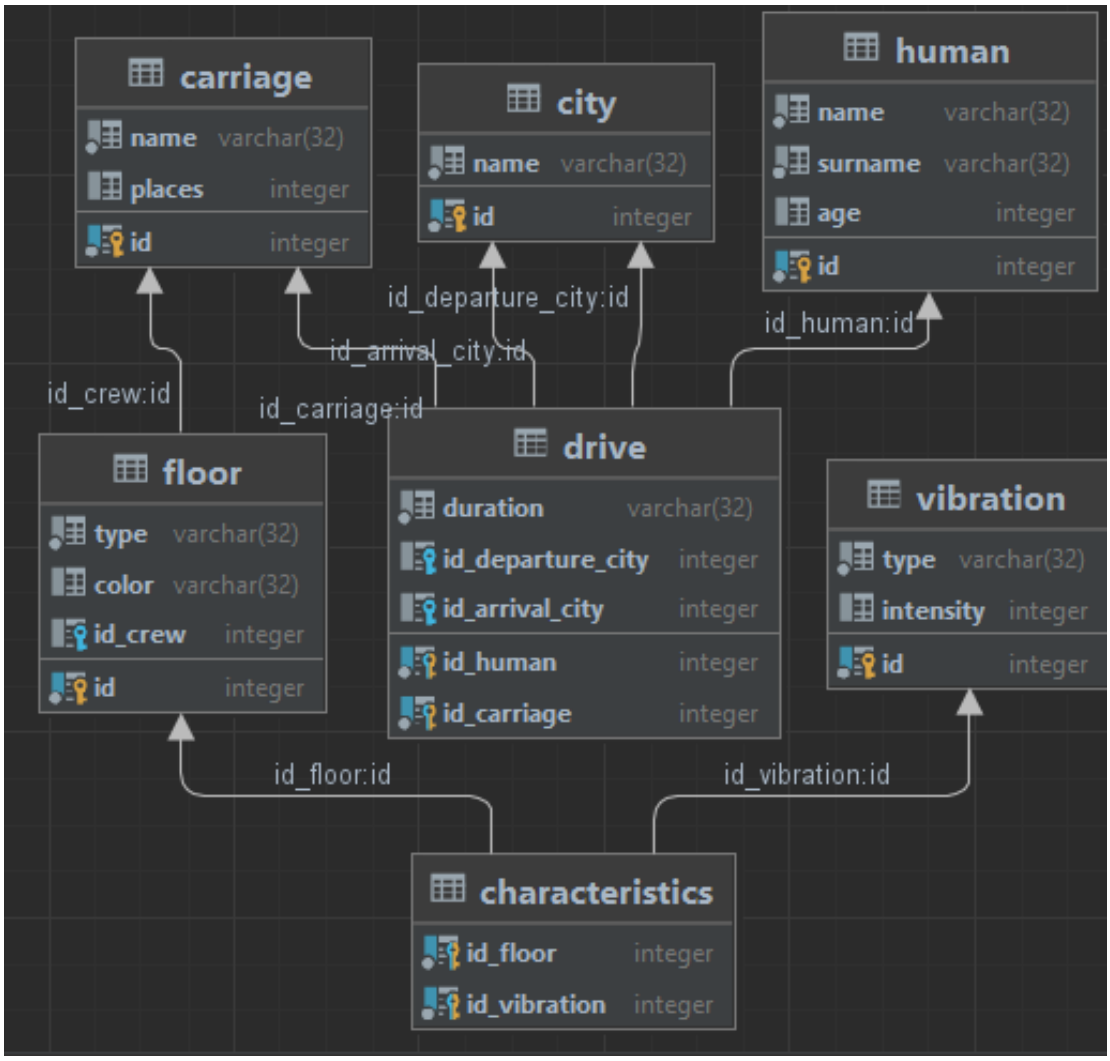
Ассоциативные

- Поездка – кто ехал, на чём ехал, сколько длилась поездка, к какому городу приехал, от какого горда приехал

Инфологическая модель



Даталогическая модель



Функциональные зависимости

1. human:
- ID -> Name,
- ID -> Surname,
- ID -> Age.
2. carriage:
- ID -> Name,
- ID -> Places.
3. city:
- ID -> Name.
4. floor:
- ID -> Type,
- ID -> Color,
- ID -> ID_carriage.
5. vibration:
- ID -> type,
- ID -> intensity.
6. drive:
- id_departure_city, id_arrival_city -> duration
- id_human, id_carriage -> id_departure_city,
- id_human, id_carriage -> id_arrival_city.

Нормализация

1) 1NF:

Отношение, у которых на пересечении каждой строки и каждого столбца – одно значение.

Моя схема уже находится в 1NF, так как в моей БД на пересечении только одно значение.

2) 2NF:

Отношение находится во 2NF, если оно находится в 1NF и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа (ПК). Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Моя схема уже находится в 2NF, так как в моей БД в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно вывести какую-то функциональную зависимость из представленных (то есть нет частичных зависимостей от первичного ключа).

3) 3NF:

Отношение находится в 3NF, когда находится во 2NF и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Моя схема не находится в 3NF, так как есть атрибут транзитивно зависящий от первичного ключа

- id_departure_city, id_arrival_city -> duration
- id_human, id_object -> id_departure_city,
- id_human, id_object -> id_arrival_city.

То есть id_human, id_object -> duration.

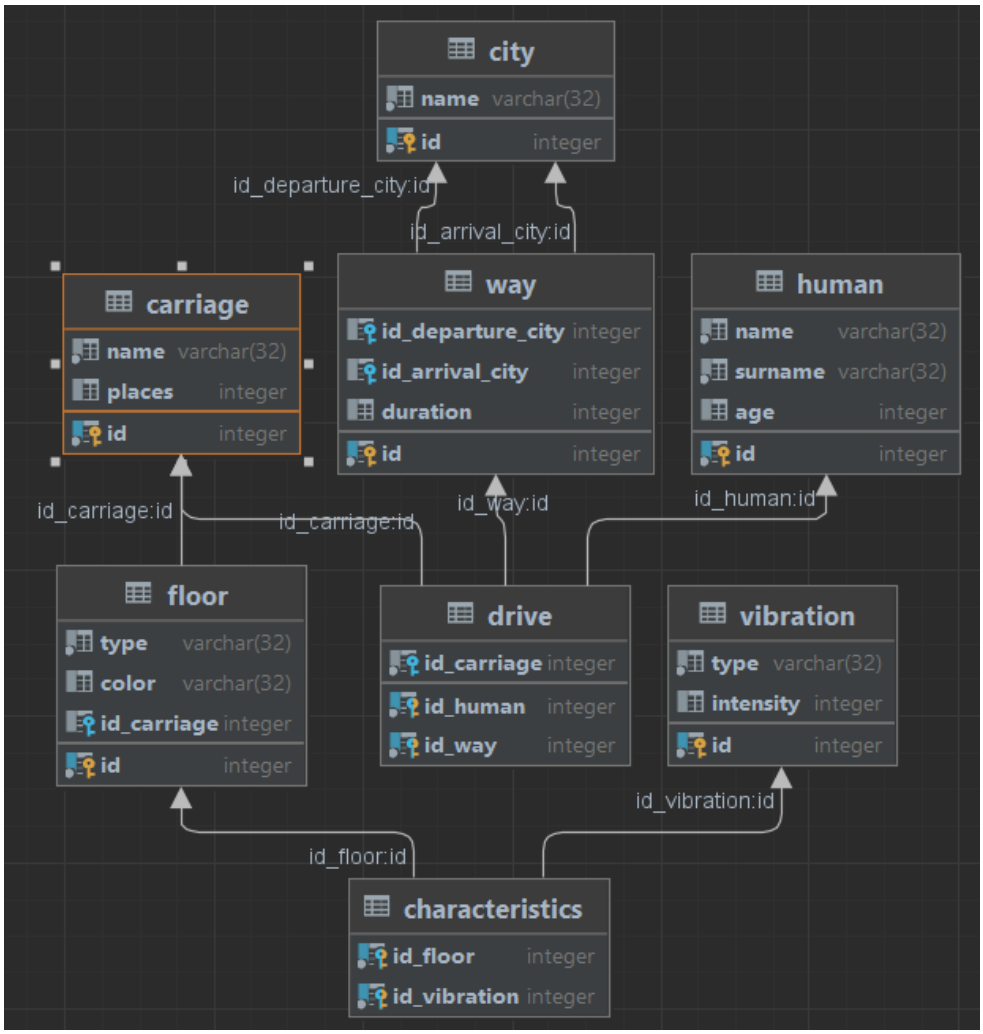
Разделим таблицу на две таблицы, чтобы устранить транзитивность.

4) BCNF:

Отношение в NFBC, когда для всех функциональных зависимостей отношения выполняется условие: детерминант — потенциальный ключ.

Моя схема не находится в NFBC, так как такие поля как id_departure_city и id_arrival_city в таблице drive не являются первичными ключами, но при этом определяют поле duration.

Исправленная база данных:



Теперь функциональные зависимости выглядят так:

drive: id_human, id_way -> id_carriage;

way: id -> id_departure_city,

id -> id_arrival_city,

id -> duration;

city: id -> name;

Такая модель соответствует правилам 3NF и BCNF.

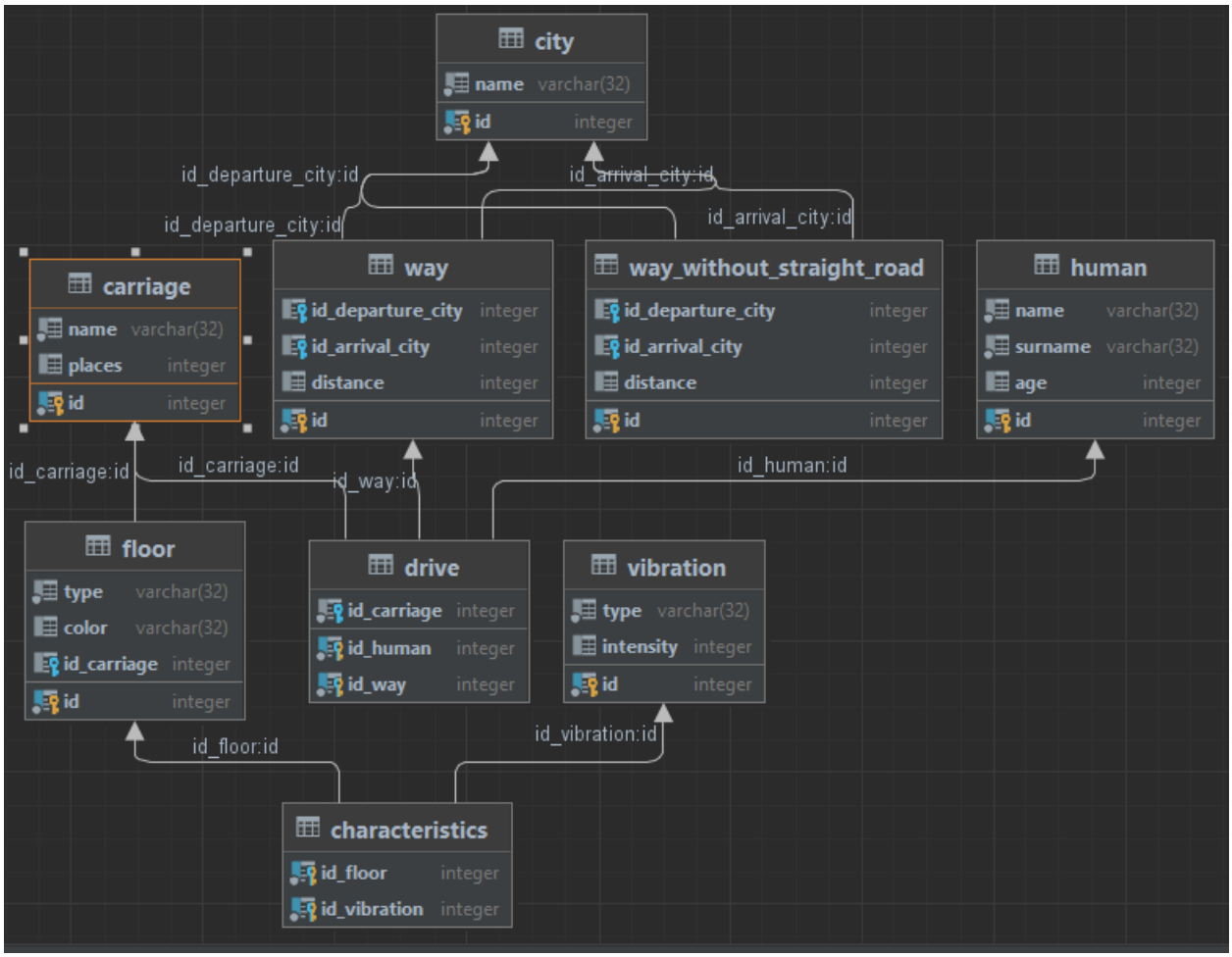
Денормализация

Для того, чтобы облегчить написание запросов и увеличить их скорость нужна денормализация. В моей схеме самым очевидным вариантом было бы объединить таблицы *floor* и *vibration* в одну.

Триггер и функция

Нужно создать триггер и связанную с ней функцию, которая находит минимальное расстояние между двумя городами, не имеющими прямой дороги. Для этого создадим дополнительную таблицу под названием way_without_straight_road. Будем добавлять новую строку в таблицу way, если там уже есть такая запись, то не будем ничего делать, если такой записи нет, то будет срабатывать триггер, который сначала добавит новую строку в таблицу way (на что сработает триггер) и затем найдёт минимальное расстояние в графе между двумя этими городами, и если же такое расстояние найдётся, запишем эти два города и минимальное расстояние между ними в новую таблицу way_without_straight_road.

Вот как будет выглядеть новая даталогическая модель:



Вот так выглядит сама функция на языке PostgreSQL:

CREATE OR REPLACE FUNCTION find_min_distance(start_city_id INTEGER, end_city_id INTEGER) RETURNS INTEGER AS

$$

```
DECLARE
   distances         INTEGER[]; --для хранения расстояний между вершинами графа.
   visited           BOOLEAN[]; --для отметки посещеённых вершин в процессе обхода графа полностью.
   previous          INTEGER[]; --для хранения информации о том, из какой вершины была достигнута текущая вершина при поиске кратчайшего пути.
   i                 INTEGER; --итератор по всем городам.
   current           INTEGER; --для хранения индекса текущей вершины при обходе графа.
   min_distance      INTEGER; --для хранения минимального расстояния до ближайшей непосещённой вершины при обходе графа.
   neighbor_id       INTEGER; --для хранения ID соседней вершины при обновлении расстояний до соседних городов.
   neighbor_distance INTEGER; --для хранения расстояния до соседней вершины при обновлении расстояний до сосоедних городов.
   n CONSTANT        INTEGER := 10; --размерность массивов.
BEGIN
   -- Инициализация массивов
   SELECT COUNT(*) INTO i FROM city;
   FOR i IN 1..n
      LOOP
         distances[i] := 1000000;
         visited[i] := FALSE;
         previous[i] := NULL;
      END LOOP;

   -- Начальная точка
   distances[start_city_id] := 0;

   -- Цикл по всем городам
```

```
    FOR i IN 1..(SELECT COUNT(*) FROM city)
        LOOP
            -- Найти ближайший непосещенный город
            min_distance := 1000000;
            FOR current IN 1..(SELECT COUNT(*) FROM city)
                LOOP
                    IF visited[current] = FALSE AND distances[current] < min_distance
THEN
                        min_distance := distances[current];
                        i := current;
                    END IF;
                END LOOP;

            IF min_distance = 1000000 THEN
                -- Все оставшиеся города недостижимы
                RETURN NULL;
            END IF;

            visited[i] := TRUE;

            -- Обновить расстояния до соседних городов
            FOR neighbor_id, neighbor_distance IN SELECT ID_arrival_city, distance
FROM way WHERE ID_departure_city = i
                LOOP
                    IF distances[i] + neighbor_distance < distances[neighbor_id] THEN
                        distances[neighbor_id] := distances[i] + neighbor_distance;
                        previous[neighbor_id] := i;
                    END IF;
                END LOOP;
```

```
        END LOOP;

    -- Восстановить путь
    current := end_city_id;
    WHILE previous[current] IS NOT NULL
        LOOP
            current := previous[current];
        END LOOP;

    RETURN distances[end_city_id];
END;
$$ LANGUAGE plpgsql;

--триггерная функция
CREATE OR REPLACE FUNCTION add_new_way() RETURNS TRIGGER AS
$$
BEGIN
    INSERT INTO way_without_straight_road(ID_DEPARTURE_CITY,
ID_ARRIVAL_CITY, DISTANCE)
    VALUES (NEW.id_departure_city, NEW.id_arrival_city, NEW.distance);

    return NEW;
END;
$$ LANGUAGE plpgsql;

--триггер
CREATE OR REPLACE TRIGGER new_way
    AFTER INSERT
    ON way
```

```
   FOR EACH ROW
EXECUTE PROCEDURE add_new_way();


--главная функция
CREATE OR REPLACE FUNCTION add_way(city1 int, city2 int) RETURNS int
as
$$
BEGIN
   IF EXISTS(SELECT id_departure_city, id_arrival_city
            FROM way
            WHERE id_arrival_city = city2
              AND id_departure_city = city1) THEN
      PERFORM distance FROM way WHERE id_arrival_city = city2 AND
id_departure_city = city1;
      RETURN NULL;
   ELSE
      INSERT INTO way(id_departure_city, id_arrival_city, distance) VALUES
(city1, city2, find_min_distance(city1, city2)); --действие на которое
срабатывает триггер.
      DELETE FROM way WHERE id_arrival_city = city2 AND
id_departure_city = city1;
      RETURN find_min_distance(city1, city2);
   END IF;
END;
$$ LANGUAGE plpgsql;


--вызов главной функции
SELECT add_way(1, 5);
--SELECT find_min_distance(1, 8);
```

Также реализация самой БД на языке PostgreSQL:

```
--Drop TABLE human CASCADE;
--DROP TABLE crew CASCADE;
--DROP TABLE Characteristics CASCADE;
--DROP TABLE city CASCADE;
--DROP TABLE floor CASCADE;
--DROP TABLE drive CASCADE;
--DROP TABLE vibration CASCADE;
CREATE TABLE human
(
   ID      SERIAL PRIMARY KEY,
   Name    VARCHAR(32) NOT NULL,
   Surname VARCHAR(32) NOT NULL,
   Age     INTEGER DEFAULT 0
);
CREATE TABLE carriage
(
   ID     SERIAL PRIMARY KEY,
   Name   VARCHAR(32) NOT NULL,
   Places INTEGER DEFAULT 1
);
CREATE TABLE floor
(
   ID      SERIAL PRIMARY KEY,
   Type    VARCHAR(32) NOT NULL,
   Color   VARCHAR(32),
   ID_carriage INTEGER REFERENCES carriage
);
CREATE TABLE city
```

```
(
   ID   SERIAL PRIMARY KEY,
   Name VARCHAR(32) NOT NULL
);
CREATE TABLE way
(
   ID              SERIAL PRIMARY KEY,
   ID_departure_city INTEGER REFERENCES city,
   ID_arrival_city   INTEGER REFERENCES city,
   distance          INTEGER DEFAULT NULL
);
CREATE TABLE way_without_straight_road
(
   ID SERIAL PRIMARY KEY,
   ID_departure_city INTEGER REFERENCES city,
   ID_arrival_city INTEGER REFERENCES city,
   distance INTEGER DEFAULT NULL
);
CREATE TABLE drive
(
   ID_Human  INTEGER REFERENCES human,
   ID_Carriage INTEGER NOT NULL REFERENCES carriage,
   ID_way    INTEGER REFERENCES way,
   PRIMARY KEY (ID_Human, ID_way)
);
CREATE TABLE vibration
(
   ID       SERIAL PRIMARY KEY,
   Type     VARCHAR(32) NOT NULL,
```

```
    Intensity INTEGER
);
CREATE TABLE characteristics
(
    ID_Floor     INTEGER REFERENCES floor,
    ID_Vibration INTEGER REFERENCES vibration,
    PRIMARY KEY (ID_Floor, ID_Vibration)
);
INSERT INTO human(name, surname, age)
VALUES ('Mark', 'Bulochka', 19),
       ('Julia', 'Oseledko', 20),
       ('Gosha', 'Smirnov', 18),
       ('Ravil', 'Keks', 17),
       ('Ivan', 'Fedotov', 18),
       ('Lesa', 'Oseledko', 1),
       ('Albert', 'Vafauulin', 23);
INSERT INTO carriage(name, places)
VALUES ('Pegas', 4),
       ('Unicorn', 2),
       ('Centaur', 2),
       ('Boom', 10);
INSERT INTO floor(type, color, ID_carriage)
VALUES ('laminat', 'red', 2),
       ('linoleum', 'blue', 2),
       ('parket', 'yellow', 1),
       ('self-leveling', 'blue', 3);
INSERT INTO city(Name)
VALUES ('Saint-Petersburg'),
       ('Krasnodar'),
```

```
    ('Ufa'),
    ('Ekaterinburg'),
    ('Cheliabinsk'),
    ('New-York'),
    ('Anapa'),
    ('Birsk'),
    ('Sochi');
INSERT INTO way(id_departure_city, id_arrival_city, distance)
VALUES (1, 2, 500),
    (2, 7, 600),
    (7, 9, 300),
    (9, 6, 350),
    (1, 8, 200),
    (8, 3, 400),
    (3, 5, 250),
    (5, 4, 100),
    (4, 6, 250);
INSERT INTO drive(ID_Human, ID_Carriage, ID_way)
VALUES (2, 3, 1),
    (3, 1, 2),
    (4, 2, 3);
INSERT INTO vibration(type, intensity)
VALUES ('strong', 10),
    ('middle', 5),
    ('low', 2);
INSERT INTO characteristics(ID_Floor, ID_Vibration)
VALUES (1, 2),
    (2, 3),
    (3, 1),
```

(4, 2);

В результате выполнения функции и триггера, в новую таблицу добавится строчка с информацией о двух городах, не имеющих между собой прямой дороги и с минимальным расстоянием между ними.

Вывод

При выполнении лабораторной работы я познакомился с нормализацией, денормализацией и функциональными зависимостями. Узнал о способах нормализировать базу данных и о разных нормальных формах моделей данных. Также научился создавать триггеры и функции на языке PostgreSQL.