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** 3](#_Toc134913375)

[**Вариант** 3](#_Toc134913376)

[**Список сущностей** 4](#_Toc134913377)

[**Инфологическая модель** 5](#_Toc134913378)

[**Даталогическая модель** 6](#_Toc134913379)

[**Функциональные зависимости** 7](#_Toc134913380)

[**Нормализация** 8](#_Toc134913381)

[**Денормализация** 10](#_Toc134913382)

[**Триггер и функция** 11](#_Toc134913383)

[**Вывод** 21](#_Toc134913384)

# **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

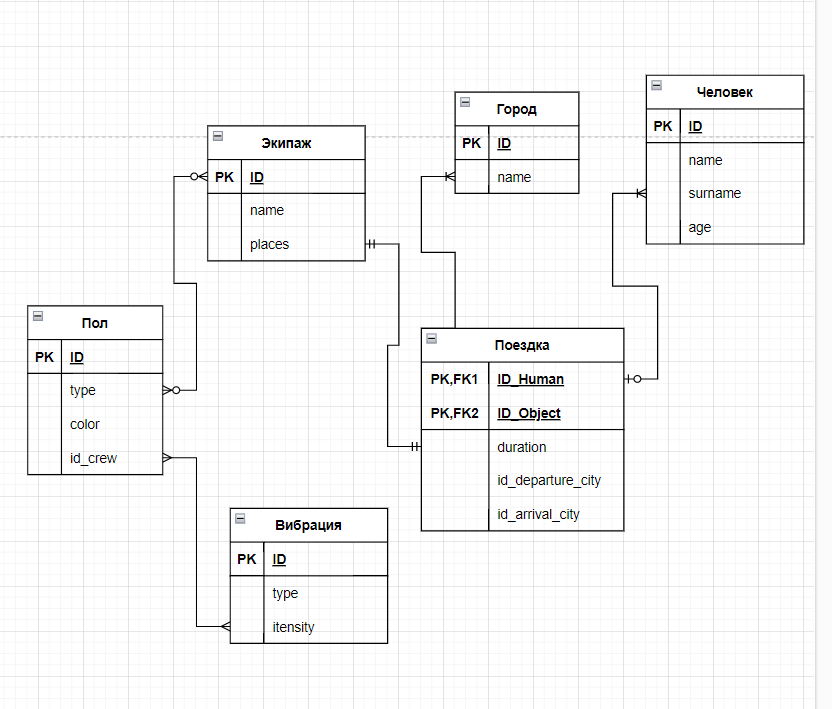
Характеристические

* Пол – тип, к какому экипажу относится

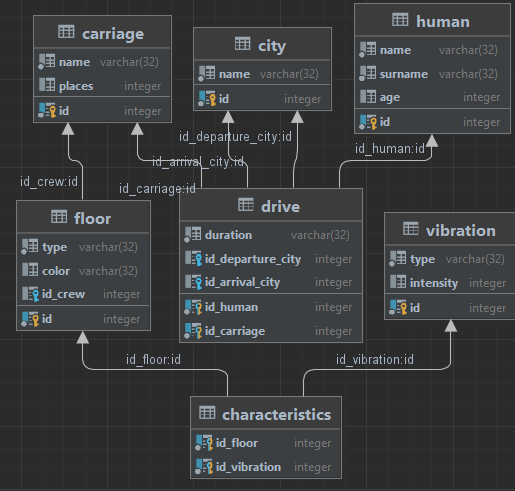
Ассоциативные

* 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.

1. carriage:

* ID -> Name,
* ID -> Places.

1. city:

* ID -> Name.

1. floor:

* ID -> Type,
* ID -> Color,
* ID -> ID\_carriage.

1. vibration:

* ID -> type,
* ID -> intensity.

1. 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.*

1. 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).*

1. 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.

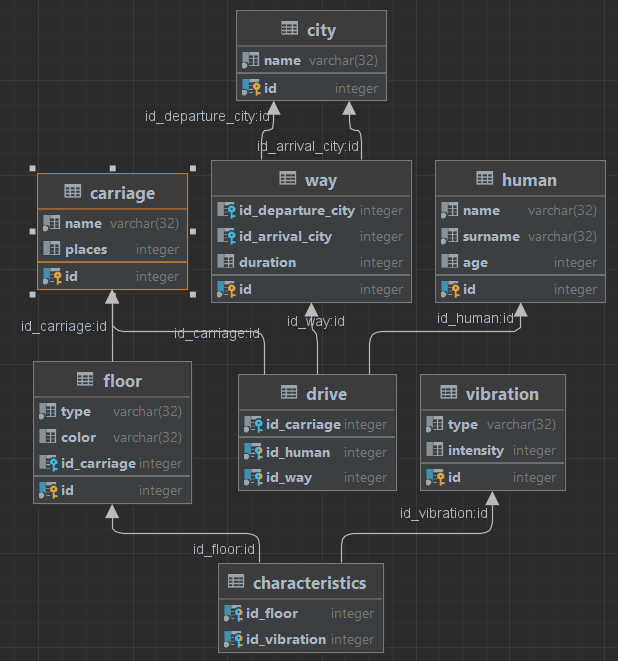
Разделим таблицу на две таблицы, чтобы устранить транзитивность.

1. 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.

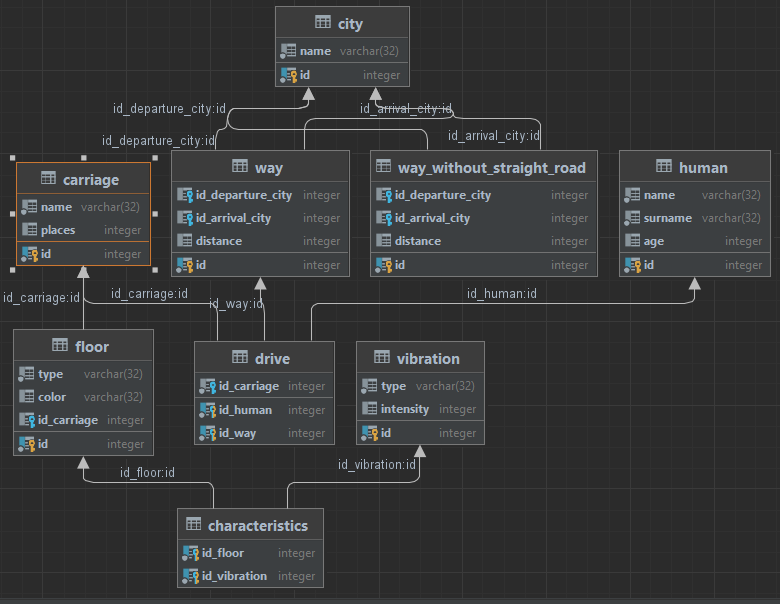
# **Денормализация**

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.