Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

Вариант №1800

Выполнил

Макогон Ярослав Вадимович

Номер группы: P3118

Проверила

Бострикова Д. К.

г. Санкт-Петербург 2025г.

Содержание

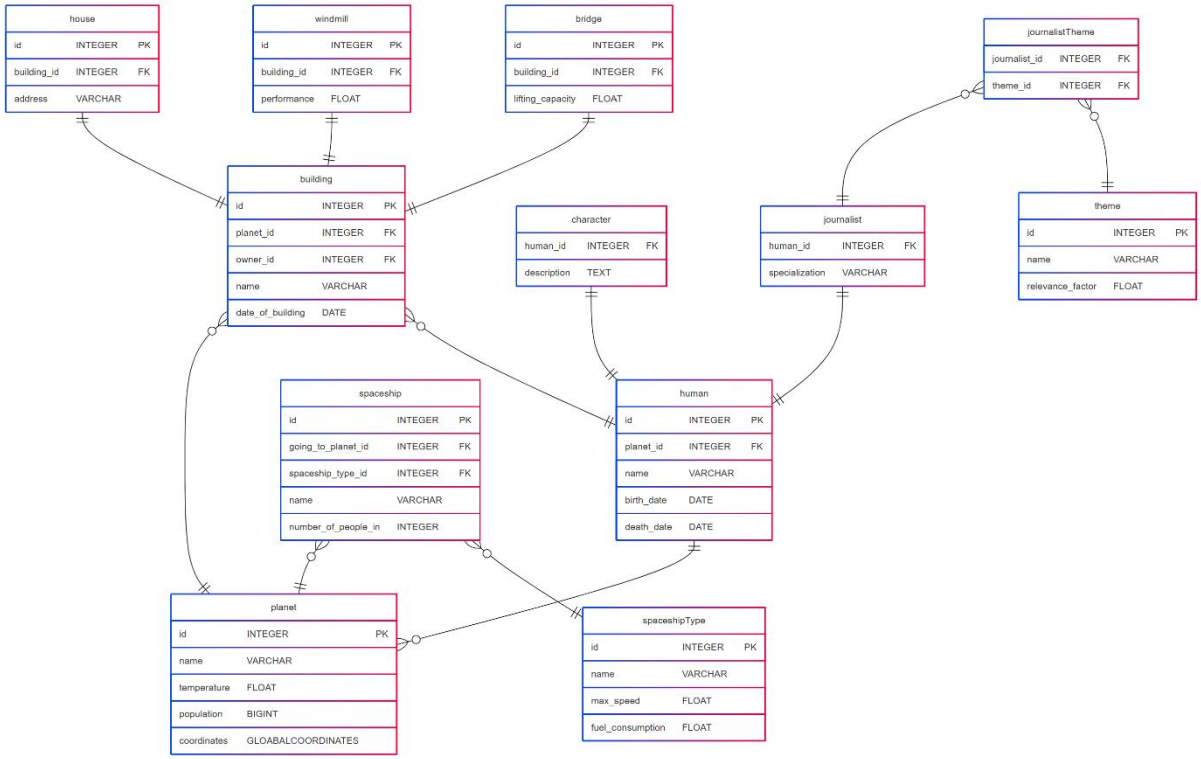
Задание

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.

Даталогическая модель



Отношение	**Функциональные зависимости**
planet	id → name *(полная)* id → temperature *(полная)* id → population *(полная)*
human	id → name *(полная)* id → birth_date *(полная)* id → planet_id *(полная)*
building	id → name *(полная)* id → date_of_building *(полная)* id → planet_id *(полная)* id → owner_id *(полная)*
house	building_id → address *(полная)*
windmill	building_id → performance *(полная)*
bridge	building_id → lifting_capacity *(полная)*
spaceshipType	id → name *(полная)* id → max_speed *(полная)* id → fuel_consumption *(полная)*
spaceship	id → name *(полная)* id → number_of_people_in *(полная)* id → going_to_planet_id *(полная)* id → spaceship_type_id *(полная)*
character	human_id → description *(полная)*
journalist	human_id → specialization *(полная)*
journalistTheme	(journalist_id, theme_id) → (…)
theme	id → name *(полная)* id → relevance_factor *(полная)*

Нормальные формы

1NF:

1. Отношение, на пересечении каждой строки и столбца — одно значение.

2NF:

1. Отношение в 1NF
2. Атрибуты, не входящие в первичный ключ, в полной функциональной зависимости от первичного ключа отношения.

3NF:

1. Отношение 2NF
2. Все атрибуты, которые не входят в первичный ключ, не находятся в транзитивной функциональной зависимости от первичного ключа.

BCNF:

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

Из описания нормальных форм, представленных выше, можно сделать вывод, что отношения в построенной мной схеме уже полностью находятся в BCNF, потому что:

- Отношение, на пересечении каждой строки и столбца — одно значение.
- Атрибуты, не входящие в первичный ключ, в полной функциональной зависимости от первичного ключа отношения.
- Все атрибуты, которые не входят в первичный ключ, не находятся в транзитивной функциональной зависимости от первичного ключа.
- Во всех отношениях никакая часть составного ключа не зависит от не ключевого атрибута

  □

  Все отношения, кроме *journalistTheme* имеют несоставной первичный ключ.

  В *journalistTheme* нет атрибутов, не входящих в первичный ключ.

  ⇨ Не может возникнуть ситуации, когда не ключевой атрибут зависит от части составного ключа.

  □

Денормализация

Можно объединить таблицы *spaceShip* и *spaceShipType*. Это позволит уменьшить количество объединений таблиц в запросах, связанных с космическими кораблями, так как скорее всего в большинстве из них нужно будет получать данные о типе корабля и его характеристиках.
Таким образом, в отношении *spaceShip* появится 3 новых атрибута: type_name, max_speed, fuel_consumption. Тогда отношения окажутся в 2НФ, так как появятся функциональные зависимости:

spaceship	**id → name *(полная)***
	id → number_of_people_in *(полная)*
	id → going_to_planet_id *(полная)*
	id → spaceship_type_id *(полная)*
	***id* → type_name (полная)**
	type_name → max_speed *(транзитивная)*
	type_name → fuel_consumption*(транзитивная)*

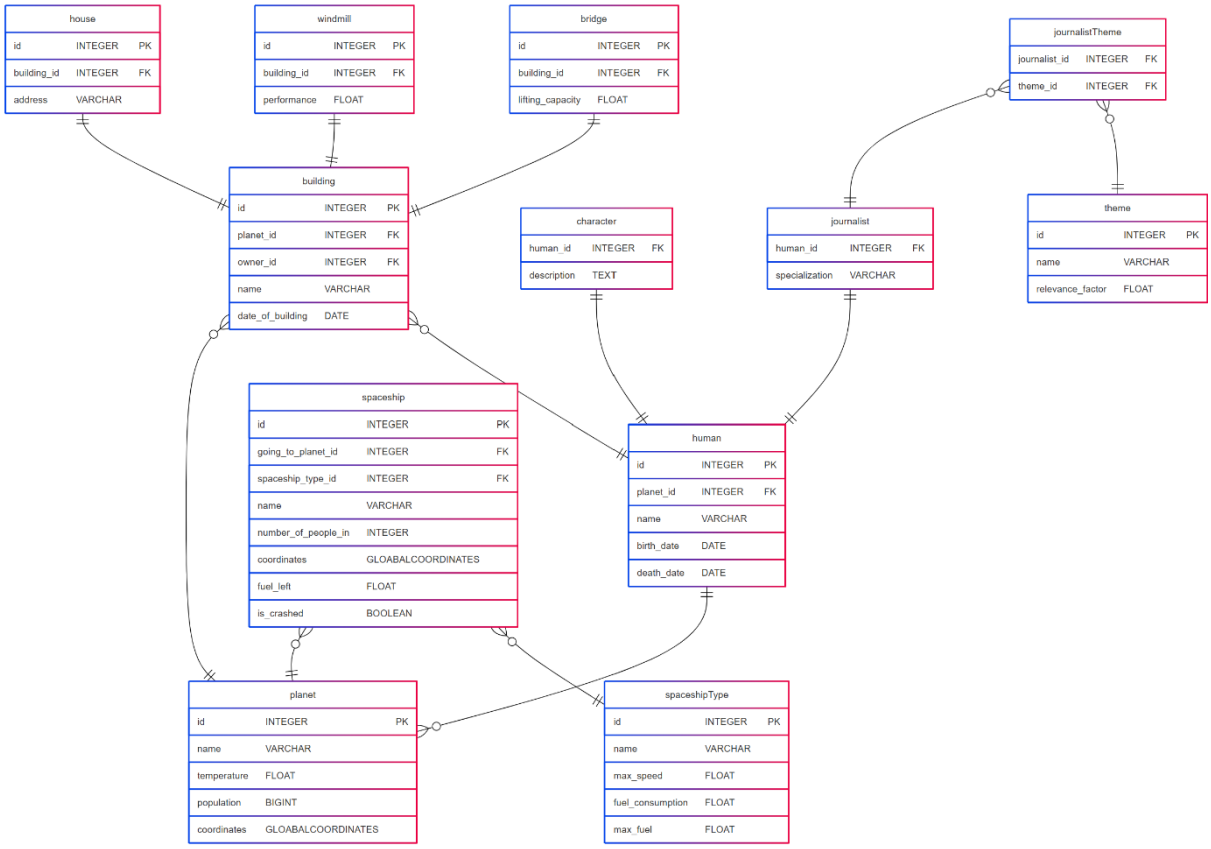
Значит, не все атрибуты вне РК не транзитивно зависят от РК => 2НФ.

Остальные денормализации будут не оправданы в силу особенностей архитектуры построенной схемы:

- ⇨ денормализация связи many-to-many => значимая просадка по памяти и, возможно, логике.
- ⇨ денормализация остальных отношений => абсолютное нарушение логики построения схемы из-за того, что оставшиеся отношения представляют из себя конкретизацию более общих объектов: human > journalist; human > character; building > windmill; building > house; building > bridge.

Триггеры и plpgsql -_-

Немного модифицировал отношения, чтобы триггер получился поинтереснее. Сейчас даталогическая модель выглядит примерно так:



Реализованы триггеры

- ⇨ Для проверки того, что оставшееся топливо у космического корабля не превышает максимально возможного количества у данного типа корабля.

- ⇨ Для передвижения космического корабля на какую-то точку. Если топлива кораблю хватает, чтобы пролететь такое расстояние, то он его пролетает и топливо у него уменьшается в соответствии с расходом для конкретной модели корабля. Если же, топлива не хватает, что корабль разбивается и больше не может двигаться.

Итоговый скрипт с небольшими комментариями:

```
-- Create

CREATE TYPE GloabalCoordinates AS (
    X FLOAT8,
    Y FLOAT8,
    Z FLOAT8
);

CREATE TABLE IF NOT EXISTS planet (
     id SERIAL PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     temperature FLOAT4 CHECK(temperature > -273.15 AND temperature <
10000000),
     population BIGINT NOT NULL CHECK(population >= 0),
    coordinates GloabalCoordinates NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS human (
     id SERIAL PRIMARY KEY,
     planet_id INTEGER REFERENCES planet(id) ON DELETE SET NULL,
     name VARCHAR(50) NOT NULL,
     birth_date DATE NOT NULL CHECK(birth_date >= '1950-01-01' AND
birth_date <= CURRENT_DATE),
    death_date DATE CHECK(birth_date <= death_date AND death_date <=
CURRENT_DATE)
);

CREATE TABLE IF NOT EXISTS building (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    date_of_building DATE CHECK(date_of_building <= CURRENT_DATE),
    planet_id INTEGER NOT NULL,
    owner_id INTEGER,
    CONSTRAINT fk_planet FOREIGN KEY (planet_id) REFERENCES planet(id)
ON DELETE CASCADE,
    CONSTRAINT fk_owner FOREIGN KEY (owner_id) REFERENCES human(id) ON
DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS house (
     id SERIAL PRIMARY KEY,
     building_id INTEGER REFERENCES building(id) NOT NULL,
     address VARCHAR(100)
);

CREATE TABLE IF NOT EXISTS windmill (
     id SERIAL PRIMARY KEY,
     building_id INTEGER REFERENCES building(id) NOT NULL,
     performance FLOAT4 CHECK(performance >= 0 AND performance <= 1)
);

CREATE TABLE IF NOT EXISTS bridge (
     id SERIAL PRIMARY KEY,
     building_id INTEGER REFERENCES building(id) NOT NULL,
     lifting_capacity FLOAT4 CHECK(lifting_capacity >= 0)
);

CREATE TABLE IF NOT EXISTS spaceshipType (
     id SERIAL PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     max_speed FLOAT8 NOT NULL CHECK(max_speed >= 0),
     fuel_consumption FLOAT8 NOT NULL CHECK(fuel_consumption >= 0),
    max_fuel FLOAT8 NOT NULL
```

```
);

CREATE TABLE IF NOT EXISTS spaceship (
      id SERIAL PRIMARY KEY,
      name VARCHAR(50) NOT NULL,
      number_of_people_in INTEGER NOT NULL CHECK(number_of_people_in >=
0),
      going_to_planet_id INTEGER REFERENCES planet(id),
      spaceship_type_id INTEGER REFERENCES spaceshipType(id) NOT NULL,
     coordinates GloabalCoordinates NOT NULL UNIQUE,
     fuel_left FLOAT8 NOT NULL,
     is_crashed BOOLEAN NOT NULL,
     CONSTRAINT not_neg_fuel_left CHECK(fuel_left >= 0)
);


CREATE TABLE IF NOT EXISTS character (
      human_id INTEGER REFERENCES human(id) NOT NULL UNIQUE,
      description TEXT
);

CREATE TABLE IF NOT EXISTS journalist (
      human_id INTEGER REFERENCES human(id) NOT NULL UNIQUE,
      specialization VARCHAR(30)
);

CREATE TABLE IF NOT EXISTS theme (
      id SERIAL PRIMARY KEY,
      name VARCHAR(50) NOT NULL,
      relevance_factor FLOAT4 NOT NULL CHECK(relevance_factor >= 0 AND
relevance_factor <= 100)
);

CREATE TABLE IF NOT EXISTS journalistTheme (
      journalist_id INTEGER REFERENCES journalist(human_id) NOT NULL,
      theme_id INTEGER REFERENCES theme(id) NOT NULL
);


-- Fill
INSERT INTO planet (name, temperature, population, coordinates) VALUES
('Earth', 15.0, 8000000000, ROW(0, 0, 0)),
('Mars', -63.0, 0, ROW(1, 2, 3)),
('Venus', 460.0, 0, ROW(10, 10, 10)),
('Jupiter', -110.0, 0, ROW(20, 20, 20)),
('Mercury', 167.0, 0, ROW(30, 30, 30));

INSERT INTO human (planet_id, name, birth_date, death_date) VALUES
(1, 'Alice', '1970-01-01', '2020-01-01'),
(2, 'Bob', '1985-05-15', NULL),
(3, 'Charlie', '1990-10-10', NULL),
(4, 'Diana', '2000-12-31', NULL),
(5, 'Eve', '1955-06-01', '2023-07-01');

INSERT INTO building (name, date_of_building, planet_id, owner_id)
VALUES
('Base Alpha', '2000-01-01', 1, 1),
('Mars Dome', '2010-05-05', 2, 2),
('Venus Tower', '2015-07-07', 3, NULL),
('Jupiter Station', '2020-08-08', 4, 3),
('Mercury Lab', '1999-12-31', 5, 4);

INSERT INTO house (building_id, address, coordinates) VALUES
(1, 'Street 1', ROW(100, 200, 300)),
(2, 'Avenue 2', ROW(110, 210, 310)),
(3, 'Road 3', ROW(120, 220, 320)),
```

```
(4, 'Boulevard 4', ROW(130, 230, 330)),
(5, 'Path 5', ROW(140, 240, 340));

INSERT INTO windmill (building_id, performance) VALUES
(1, 0.8),
(2, 0.5),
(3, 0.9),
(4, 0.6),
(5, 0.75);

INSERT INTO bridge (building_id, lifting_capacity) VALUES
(1, 1000.0),
(2, 1500.0),
(3, 1200.0),
(4, 1300.0),
(5, 1100.0);

INSERT INTO spaceshipType (name, max_speed, fuel_consumption,
max_fuel) VALUES
('Scout', 50000, 10, 1000),
('Cruiser', 30000, 20, 2000),
('Freighter', 20000, 50, 5000),
('Shuttle', 10000, 5, 300),
('Interceptor', 70000, 25, 1500);

INSERT INTO spaceship (name, number_of_people_in, going_to_planet_id,
spaceship_type_id, coordinates, fuel_left, is_crashed) VALUES
('Explorer One', 5, 2, 1, ROW(200, 200, 200), 900, false),
('Red Falcon', 3, 3, 2, ROW(210, 210, 210), 1500, false),
('Heavy Load', 10, 4, 3, ROW(220, 220, 220), 4000, false),
('Quick Jump', 2, 5, 4, ROW(230, 230, 230), 200, false),
('Interceptor Z', 1, 1, 5, ROW(240, 240, 240), 1200, true);

INSERT INTO character (human_id, description) VALUES
(1, 'Main character'),
(2, 'Sidekick'),
(3, 'Scientist'),
(4, 'Pilot'),
(5, 'Historian');

INSERT INTO journalist (human_id, specialization) VALUES
(1, 'Politics'),
(2, 'Science'),
(3, 'Technology'),
(4, 'Culture'),
(5, 'Economics');

INSERT INTO theme (name, relevance_factor) VALUES
('Colonization', 80),
('Energy', 90),
('AI Ethics', 70),
('Interplanetary Travel', 95),
('Terraforming', 85);

INSERT INTO journalistTheme (journalist_id, theme_id) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);



-- Kill function
```

```
CREATE OR REPLACE FUNCTION kill(IN human_id INTEGER)
RETURNS void
AS $$
BEGIN
    IF
        NOT EXISTS(SELECT *
        FROM human
        WHERE id = human_id)
    THEN
        RAISE EXCEPTION 'Human with id=% not found', human_id;
        RETURN;
    END IF;

    IF
        (SELECT death_date
        FROM human
        WHERE id = human_id)
        IS NOT NULL
    THEN
        RAISE EXCEPTION 'Human with id=% has alredy dead', human_id;
        RETURN;
    END IF;

    UPDATE human
    SET death_date = CURRENT_DATE
    WHERE id = human_id;
END
$$
LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION check_max_fuel()
RETURNS trigger
AS $$
DECLARE
    max_fuel FLOAT8;
BEGIN
    max_fuel = (
        SELECT spaceshipType.max_fuel
        FROM spaceshipType
        WHERE spaceshipType.id = NEW.spaceship_type_id
    );
    IF
        max_fuel
        <
        NEW.fuel_left
    THEN
        NEW.fuel_left = max_fuel;
        RAISE NOTICE 'Fuel left value for spaceship with id=% was
changed to %, becouse of max fuel constriant.', NEW.id, max_fuel;
    END IF;

    RETURN NEW;
END
$$
LANGUAGE plpgsql;


CREATE TRIGGER check_max_fuel_trigger
BEFORE INSERT OR UPDATE
ON spaceship
FOR EACH ROW EXECUTE FUNCTION check_max_fuel();
```

```
CREATE OR REPLACE FUNCTION get_distance(IN fromCoordinates
GloabalCoordinates, IN toCoordinates GloabalCoordinates)
RETURNS FLOAT8
AS $$
DECLARE
    result FLOAT8;
BEGIN
    result = sqrt((fromCoordinates.X - toCoordinates.X)^2 +
(fromCoordinates.Y - toCoordinates.Y)^2 + (fromCoordinates.Z -
toCoordinates.Z)^2);
    RETURN result;
END
$$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get_new_point(
    IN start_point GloabalCoordinates,
    IN end_point GloabalCoordinates,
    IN len FLOAT8
)
RETURNS GloabalCoordinates
AS $$
DECLARE
    v GloabalCoordinates;
    res GloabalCoordinates;
    vlen FLOAT8;
BEGIN
    v = ROW(end_point.X - start_point.X, end_point.Y - start_point.Y,
end_point.Z - start_point.Z);
    vlen = abs(get_distance(start_point, end_point));
    v = ROW(v.X / vlen, v.Y / vlen, v.Z / vlen);
    res = ROW(
        start_point.X + v.X * len,
        start_point.Y + v.Y * len,
        start_point.Z + v.Z * len
    );
    RETURN res;
END;
$$
LANGUAGE plpgsql;


-- Trigger function that changes spaceship params (fuel_left,
is_crashed) when it moves somewhere
CREATE OR REPLACE FUNCTION spaceship_check()
RETURNS trigger
AS $$
DECLARE
    distance FLOAT8;
    fuel_consumption FLOAT8;
    need_fuel FLOAT8;
BEGIN
    IF OLD.is_crashed AND NEW.coordinates != OLD.coordinates
    THEN
        RAISE EXCEPTION 'Spaceship with id=% can''t move because it
was crashed!', OLD.id;
    END IF;

    IF NEW.coordinates = OLD.coordinates
    THEN
        RETURN NEW;
    END IF;

    fuel_consumption = (
        SELECT spaceshipType.fuel_consumption
        FROM spaceshipType
```

```
        WHERE spaceshipType.id = NEW.spaceship_type_id
    );
    distance = get_distance(OLD.coordinates, NEW.coordinates);
    need_fuel = distance * fuel_consumption;

    IF need_fuel <= OLD.fuel_left
    THEN
        NEW.fuel_left = OLD.fuel_left - need_fuel;
    ELSE
        NEW.fuel_left = 0;
        NEW.is_crashed = TRUE;
        NEW.coordinates = get_new_point(OLD.coordinates,
NEW.coordinates, OLD.fuel_left/fuel_consumption);
        RAISE NOTICE 'Spaceship with id=% was crashed!', NEW.id;
    END IF;
    RETURN NEW;

END
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_spaceship_changes
BEFORE UPDATE
ON spaceship
FOR EACH ROW
EXECUTE FUNCTION spaceship_check();


-- VIEW to see spaceship and its fuel_consumption
CREATE VIEW spaceships_with_fuel_consumptions AS (
    SELECT spaceship.id, spaceship.name,
spaceship.fuel_left,spaceship.is_crashed ,
spaceshipType.fuel_consumption, spaceshipType.max_fuel
    FROM spaceship
    JOIN spaceshipType
    ON spaceshipType.id = spaceship.spaceship_type_id
);

-- Auxiliary function: it shows how mush fuel will use spaceship with
id = spaceship_id to move for distance
CREATE OR REPLACE FUNCTION used_fuel_for_distance(IN spaceship_id
BIGINT, IN distance FLOAT8)
RETURNS FLOAT8
AS $$
DECLARE
    fuel_consumption FLOAT8;
    is_crashed BOOLEAN;
    spaceship_type_id BIGINT;
BEGIN
    SELECT spaceship.is_crashed, spaceship.spaceship_type_id
    INTO is_crashed, spaceship_type_id
    FROM spaceship
    WHERE id = spaceship_id;

    IF is_crashed
    THEN
        RETURN -1;
    END IF;

    fuel_consumption = (
        SELECT spaceshipType.fuel_consumption
        FROM spaceshipType
        WHERE spaceshipType.id = spaceship_type_id
    );
    RETURN (fuel_consumption * distance);
END
```

```
$$
LANGUAGE plpgsql;

-- I want to see a table that shows how much fuel spaceship will use
to move for some distance
CREATE OR REPLACE FUNCTION fuel_used_by_spaceships(IN distance FLOAT8)
RETURNS SETOF RECORD
AS $$
    SELECT spaceship.id, spaceship.name, spaceship.is_crashed,
used_fuel_for_distance(id, distance)
    FROM spaceship;
$$
LANGUAGE SQL;

/*
SELECT * FROM fuel_used_by_spaceships(100)
AS (id integer, name varchar, is_crashed boolean, fuel_used float8);

*/
```

Выводы

- Познакомился с принципами нормализации баз данных
- Научился писать функции и триггеры в PostgreSQL, используя plpgsql