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.

**Содержание**

[Задание 3](#_Toc197112465)

[Даталогическая модель 4](#_Toc197112466)

[Нормальные формы 5](#_Toc197112467)

[Денормализация 6](#_Toc197112488)

[Триггеры и plpgsql -\_- 7](#_Toc197112494)

[Выводы 15](#_Toc197112501)

### 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:

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

### 2NF:

### Отношение в 1NF

### Атрибуты, не входящие в первичный ключ, в полной функциональной зависимости от первичного ключа отношения.

### 3NF:

### Отношение 2NF

### Все атрибуты, которые не входят в первичный ключ, не находятся в транзитивной функциональной зависимости от первичного ключа.

### BCNF:

### 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 PK не транзитивно зависят от PK => 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