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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №3

По Базам Данных

Выполнил:

Ларионов Владислав Васильевич

Группа P3109

Практик:

Воронина Дарья Сергеевна

Санкт-Петербург 2024

**Содержание**

[**Задание** 3](#_Toc196922895)

[**Выполнение задания** 4](#_Toc196922896)

[**2.1 Функциональные зависимости** 4](#_Toc196922897)

[**2.2 Приведение к 3NF** 4](#_Toc196922898)

[**2.3** **Приведение к BCNF** 5](#_Toc196922899)

[**2.4** **Полезные денормализации** 5](#_Toc196922900)

[**2.5** **Триггер** 6](#_Toc196922901)

[**Исходная и нормализованная (одинаковые) формы:** 7](#_Toc196922902)

[**Вывод:** 12](#_Toc196922903)

# **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 основе NF (как минимум).
* Опишите изменения в функциональных зависимостях, произошедшие после преобразования в 3NF (как минимум). Постройте схему на основе NF;
* 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.

# **Выполнение задания**

# **2.1 Функциональные зависимости**

**Таблица planet:**

* planet\_id –> name

**Таблица natural\_scene:**

* natural\_scene\_id –> planet\_id, timestamp

**Таблица phenomena:**

* phenomen\_id –> name, description

**Таблица consequences:**

* consequence\_id –> consequence

**Таблица phenomena\_consequences:**

* (phenomen\_id, consequence\_id) –> (нет атрибутов, только ключ)
* phenomen\_id –> (частичная зависимость)
* consequence\_id –> (частичная зависимость)

**Таблица phenomena\_in\_scene:**

* (natural\_scene\_id, phenomen\_id) –> (нет атрибутов, только ключ)
* natural\_scene\_id –> (частичная зависимость)
* phenomen\_id –> (частичная зависимость)

## **2.2 Приведение к 3NF**

**Исходная схема уже находится в 3NF по двум причинам:**

1. Все таблицы находятся в 2NF (нет частичных зависимостей от составных ключей)
2. Нет транзитивных зависимостей (все неключевые атрибуты зависят только от первичного ключа)

## **Приведение к BCNF**

1. Все таблицы имеют простые первичные ключи (не составные)
2. В таблицах с составными ключами нет других функциональных зависимостей

## **Полезные денормализации**

Создание отдельной таблицы для отчетов, которая будет содержать в себе данные об экшн-сцене.

CREATE TABLE IF NOT EXISTS scene\_reports (

    scene\_report\_id SERIAL PRIMARY KEY,

    FOREIGN KEY (natural\_scene\_id) REFERENCES natural\_scene(natural\_scene\_id) ON DELETE CASCADE,

    scene\_timestamp TIMESTAMP,

    FOREIGN KEY (planet\_id) REFERENCES planet(planet\_id) ON DELETE CASCADE,

    planet\_name VARCHAR(6),

    phenomena\_list TEXT,

    consequences\_list TEXT

);

Поможет отслеживать экшн сцены, так как есть мгновенный доступ к данным, однако может быть избыточность данных и высокие затраты на обновление.

## **Триггер**

Триггер проверяет, существует ли уже какое-то явление в экшн-сцене. Если существует, то печатается ошибка, в противном же случае все добавляется.

CREATE OR REPLACE FUNCTION check\_phenomenon\_uniqueness()

RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (

        SELECT 1

        FROM phenomena\_in\_scene

        WHERE natural\_scene\_id = NEW.natural\_scene\_id

        AND phenomen\_id = NEW.phenomen\_id

    ) THEN

        RAISE EXCEPTION 'Природное явление (ID: %) уже присутствует в сцене (ID: %)',

        NEW.phenomen\_id, NEW.natural\_scene\_id;

    END IF;

    RETURN NEW;

END;

$$

CREATE TRIGGER trg\_check\_phenomenon\_uniqueness

BEFORE INSERT OR UPDATE ON phenomena\_in\_scene

FOR EACH ROW

EXECUTE FUNCTION check\_phenomenon\_uniqueness();

# **Исходная и нормализованная (одинаковые) формы:**

DROP TABLE IF EXISTS phenomena\_in\_scene;

DROP TABLE IF EXISTS phenomena\_consequences;

DROP TABLE IF EXISTS natural\_scene;

DROP TABLE IF EXISTS phenomena;

DROP TABLE IF EXISTS consequences;

DROP TABLE IF EXISTS planet;

CREATE TABLE IF NOT EXISTS planet (

    planet\_id SERIAL PRIMARY KEY,

    name VARCHAR(6) NOT NULL

);

CREATE TABLE IF NOT EXISTS natural\_scene (

    natural\_scene\_id SERIAL PRIMARY KEY,

    planet\_id INT NOT NULL,

    timestamp TIMESTAMP NOT NULL,

    FOREIGN KEY (planet\_id) REFERENCES planet(planet\_id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS phenomena (

    phenomen\_id SERIAL PRIMARY KEY,

    name VARCHAR(12) NOT NULL,

    description VARCHAR(10)

);

CREATE TABLE IF NOT EXISTS consequences (

    consequence\_id SERIAL PRIMARY KEY,

    consequence VARCHAR(22) NOT NULL

);

CREATE TABLE IF NOT EXISTS phenomena\_consequences (

    phenomen\_id INT NOT NULL,

    consequence\_id INT NOT NULL,

    PRIMARY KEY (phenomen\_id, consequence\_id),

    FOREIGN KEY (phenomen\_id) REFERENCES phenomena(phenomen\_id) ON DELETE CASCADE,

    FOREIGN KEY (consequence\_id) REFERENCES consequences(consequence\_id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS phenomena\_in\_scene (

    natural\_scene\_id INT NOT NULL,

    phenomen\_id INT NOT NULL,

    FOREIGN KEY (natural\_scene\_id) REFERENCES natural\_scene(natural\_scene\_id) ON DELETE CASCADE,

    FOREIGN KEY (phenomen\_id) REFERENCES phenomena(phenomen\_id) ON DELETE CASCADE

);

CREATE OR REPLACE FUNCTION check\_phenomenon\_uniqueness()

RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (

        SELECT 1

        FROM phenomena\_in\_scene

        WHERE natural\_scene\_id = NEW.natural\_scene\_id

        AND phenomen\_id = NEW.phenomen\_id

    ) THEN

        RAISE EXCEPTION 'Природное явление (ID: %) уже присутствует в сцене (ID: %)',

        NEW.phenomen\_id, NEW.natural\_scene\_id;

    ELSE

        PERFORM 1 FROM phenomena\_in\_scene;

    END IF;

    RETURN NEW;

END; $$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_phenomenon\_uniqueness

BEFORE INSERT OR UPDATE ON phenomena\_in\_scene

FOR EACH ROW

EXECUTE FUNCTION check\_phenomenon\_uniqueness();

INSERT INTO planet (name) VALUES

('Юпитер');

INSERT INTO phenomena (name, description) VALUES

('вихри', NULL),

('порывы ветра', 'ураганные'),

('потоки газа', 'восходящие'),

('воронка', 'гигантская');

INSERT INTO consequences (consequence) VALUES

('нарушать строй облаков'),

('раскрывать пелену'),

('открывать вид'),

('низвергаться в глубины');

INSERT INTO natural\_scene (planet\_id, timestamp) VALUES

(1, '2025-03-03 12:00:00'),

(1, '2025-03-03 14:00:00');

INSERT INTO phenomena\_consequences (phenomen\_id, consequence\_id) VALUES

(1, 1),

(2, 2),

(3, 3),

(4, 4);

INSERT INTO phenomena\_in\_scene (natural\_scene\_id, phenomen\_id) VALUES

(1, 1),

(1, 2),

(2, 3),

(2, 4);

SELECT \* from natural\_scene;

SELECT \* from planet;

SELECT \* from phenomena\_in\_scene;

SELECT \* from phenomena;

SELECT \* from phenomena\_consequences;

SELECT \* from consequences;

# **Вывод:**

В ходе выполнения данной лабораторной работы я научился работать с функциональными зависимостями, узнал про их виды, разобрался с нормальными формами, научился приводить к ним базу данных. Также я узнал, что такое денормализация, попробовал денормализовать базу данных с целью увеличения оптимизации запросов.