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

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)

2.3 Приведение к BCNF

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

2.4 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.

2.5 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.