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

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

Студент:
Кулагин Вячеслав, P3109
Преподаватель:
Райла Мартин

Санкт-Петербург
2024

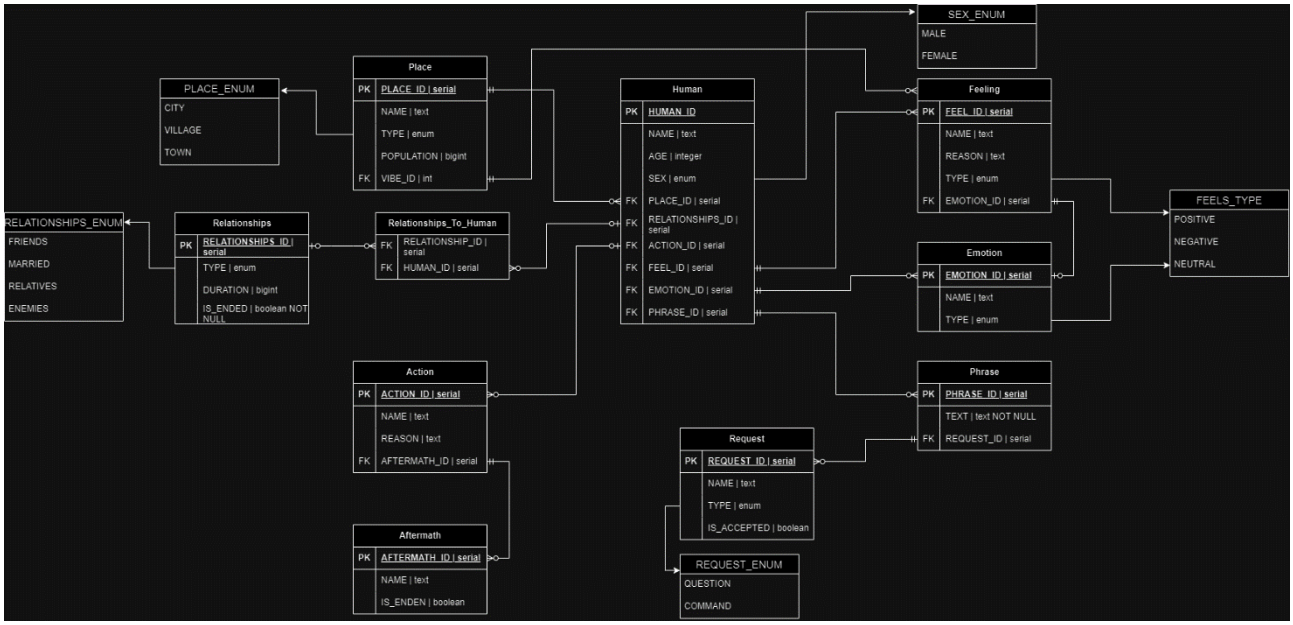
Оглавление

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.

Функциональный зависимости



emotions: emotion_id → (name, type)
feeling: feel_id → (name, reason, type, emotion_id)
place: place_id → (name, type, population, vibe_id)
request: request_id → (type, is_accepted)
phrase: phrase_id → (text, request_id)
aftermath: aftermath_id → (name, is_ended)

action: action_id → (name, reason, aftermath_id)
relationships: relationships_id → (type, duration, is_ended)
relationships_to_human: (relationships_id, human_id) → ()
human: human_id → (name, age, sex, place_id, action_id, feel_id, emotion_id, phrase_id)

Нормальные формы

1NF: Отношение уже находится в 1NF, потому что все атрибуты во всех таблицах являются скалярными величинами

2NF: Отношение находится в 1NF, все его неключевые атрибуты полностью функционально зависят от первичного ключа → отношение в 2NF

3NF: Отношение находится в 2NF, все неключевые атрибуты полностью не транзитивно зависят только от первичных ключей → отношение в 3NF

BCNF: Составные ключи не используются, при этом отношение в 3NF → отношение в BCNF

Обоснование каждой таблицы

Для каждой таблицы мы проверим функциональные зависимости и убедимся, что каждая неключевая атрибуция полностью зависит от первичного ключа, и что нет транзитивных зависимостей.

- **Emotions**
  emotion_id → (name, type)
  emotion_id однозначно идентифицирует каждую эмоцию
  Name (название) и type (тип) эмоции однозначно идентифицируют каждую эмоцию. У них у всех есть уникально, относящиеся конкретно к этой эмоции название и тип. При этом чувства (feeling) выделена в отдельную таблицу, т. к. не является логическим продолжением и дополнением эмоций.
  Следовательно, таблица в 3NF и BCNF

- **Feeling**
  feel_id → (name, reason, type, emotion_id)
  feel_id однозначно идентифицирует каждое чувство
  Name (название), type (тип) однозначно идентифицируют каждое чувство, при этом каждое отдельное чувство может вызывать определенную эмоцию (но эмоция может существовать отдельно от чувств, поэтому эмоция существует отдельно от чувств)
  Следовательно, таблица в 3NF и BCNF

- **Place**
  place_id → (name, type, population, vibe_id)

place_id однозначно идентифицирует каждое конкретное место
Name (имя), type (тип), population (население) однозначно идентифицируют каждое конкретное место (город). При этом vibe_id отражает чувства, которые существуют отдельно от города. Другими словами, каждое чувство, которое навевает город, может существовать отдельно от этого города, его может испытывать также каждый отдельный человек (не обязательно проживающий в данном городе)
Следовательно, таблица в 3NF и BCNF

- **Request**
  request_id → (type, is_accepted)
  request_id однозначно идентифицирует каждый конкретный запрос
  Type (тип) и is_accepted (статус принятия) однозначно идентифицируют каждый конкретный запрос, при этом атрибуты напрямую зависят только от первичного ключа (request_id)
  Следовательно, таблица в 3NF и BCNF

- **Phrase**
  phrase_id → (text, request_id)
  phrase_id однозначно идентифицирует каждую конкретную фразу
  Text (текст) однозначно идентифицируют каждую существующую фразу, при этом фраза может содержать request_id и нужно учитывать, что запрос может существовать независимо от существования фразы. Запрос может быть создан с помощью иных невербальных способов общения, а значит, необязательно должен быть неразрывно соединен с произнесенной фразой.
  Следовательно, таблица в 3NF и BCNF

- **Aftermath**
  aftermath_id → (name, is_ended)
  aftermath_id однозначно идентифицирует каждое конкретное последствие
  Name (название) и is_ended (статус окончания) однозначно идентифицируют каждое конкретное последствие, при этом атрибуты напрямую зависят только от первичного ключа (aftermath_id)
  Следовательно, таблица в 3NF и BCNF

- **Action**
  action_id → (name, reason, aftermath_id)
  action_id однозначно идентифицирует каждое конкретное действие
  Name (имя), reason (причина) однозначно идентифицируют каждое конкретное действие, при этом последствие (aftermath_id) выделено отдельно, потому что действие может не нести никаких последствий, а последствие может возникнуть как результат не только лишь одного действия, но и многих других (и это последствие может быть одно для них всех)
  Следовательно, таблица в 3NF и BCNF

- **Relationships**
  relationships_id → (type, duration, is_ended)
  relationships_id однозначно идентифицирует каждое взаимоотношение
  eType (тип), duration (длительность), is_ended (статус завершённости) однозначно идентифицируют каждые конкретные отношения, при этом атрибуты напрямую зависят только от первичного ключа (relationships_id), при этом они напрямую не соотносятся с людьми, чтобы была возможность состоять в нескольких отношениях одновременно.
  Следовательно, таблица в 3NF и BCNF

- **Relationships_to_human**
  (relationships_id, human_id) → ()
  Таблица существует только для создание связей между людьми и отношениями. Она не имеет атрибутов и содержат только первичные ключи. Таким образом, таблица уже находится в 3NF и BCNF

- **Human**
  human_id → (name, age, sex, place_id, action_id, feel_id, emotion_id, phrase_id)
  human_id однозначно идентифицирует каждого человек
  Name (имя), age (возраст), sex (пол) однозначно идентифицировать каждого конкретного человека. При этом отдельно человек связан с place (местом/городом) (в городе может проживать много людей, город может существовать в отрыве от человека, он также имеет ряд атрибутов, свойственных только ему), action (действием) (одно и то же действие может свершать несколько людей одновременно, последствия действия могут влиять не только на одного человека), feel (чувством) (одно и то же чувство может испытывать несколько людей, оно не должно быть неразрывно связано в каждым конкретным человеком), emotion (эмоцией) (одна и та же эмоция может быть свойственна сразу нескольким людям), phrase (фраза) (одна и та же фраза может быть произнесена разными людьми, ровно как она может быть адресована разным людям)
  Следовательно, таблица в 3NF и BCNF

Возможная денормализация

Возможно допустить следующие денормализации для более удобной работы с данными:

- Объединить таблицы action и aftermath, чтобы было удобнее понимать, чем кончилось конкретное действие. Это нарушит нормализацию, потому что атрибуты aftermath будут в транзитивной зависимости от action_id, что нарушает 3NF. Однако это может быть удобно для анализа действий в таблице для наблюдателя

- Также возможно по такой же логике объединить phrase и request

Триггер

Требуется создать триггер, который будет высчитывать рейтинг города (по умолчанию равному 5), выраженный дробным числом. Формируется он по следующим правилам (в зависимости от количества взаимоотношений между людьми):

- Если в городе появляются человек, который стал врагом (ENEMIES), то рейтинг снижается на 1.5
- Если в городе появляются человек, который подружился (FRIENDS), то рейтинг города повышается на 0.5
- Если в городе появляется стал партнёром (MARRIED), то рейтинг города повышается на 1
- Появление в городе родственников (RELATIVES) никак не влияет на рейтинг

Рейтинг города влияет на следующие:

- Если рейтинг выше 10, то все в городе становятся счастливее (все чувства и эмоции людей увеличиваются на 1 ступень: NEGATIVE → NEUTRAL, NEUTRAL → POSITIVE)
- Если рейтинг города в диапазоне от 0 до 10 включительно, рейтинг города не оказывает влияния на жителей
- Если рейтинг города становится отрицательным, то все в городе грустят (все чувства и эмоции людей становятся NEGATIVE)

Было создано 2 триггера, которые концептуально сходных по смыслу. Один срабатывает при добавлении нового элемента в таблицу relationships_to_human (которая как раз отражает существующие взаимоотношения между людьми), а второй срабатывает при удалении элементов и возвращает рейтинг города на изначальные значения (а также редактирует эмоции и чувства всех людей)

Добавление

```
CREATE OR REPLACE FUNCTION update_city_rating() RETURNS TRIGGER AS $$
DECLARE
   city_id INT;
   relationship_type RELATIONSHIPS_TYPE;
   city_rating FLOAT;
BEGIN
   SELECT h.PLACE_ID, r.TYPE INTO city_id, relationship_type
   FROM human h
   JOIN relationships_to_human rh ON h.HUMAN_ID = rh.HUMAN_ID
   JOIN relationships r ON rh.RELATIONSHIPS_ID = r.RELATIONSHIPS_ID
   WHERE rh.HUMAN_ID = NEW.HUMAN_ID AND rh.RELATIONSHIPS_ID = NEW.RELATIONSHIPS_ID;

   SELECT RATING INTO city_rating FROM place WHERE PLACE_ID = city_id;

   IF relationship_type = 'ENEMIES' THEN
      city_rating := city_rating - 1.5;
```

```
  ELSIF relationship_type = 'FRIENDS' THEN
    city_rating := city_rating + 0.5;
  ELSIF relationship_type = 'MARRIED' THEN
    city_rating := city_rating + 1.0;
  END IF;

  UPDATE place SET RATING = city_rating WHERE PLACE_ID = city_id;

    IF city_rating > 10 THEN
      UPDATE emotion
      SET TYPE = CASE
        WHEN TYPE = 'NEGATIVE' THEN 'NEUTRAL'
        WHEN TYPE = 'NEUTRAL' THEN 'POSITIVE'
        ELSE TYPE
      END
      WHERE EMOTION_ID IN (
        SELECT EMOTION_ID
        FROM human
        WHERE PLACE_ID = city_id
      );

      UPDATE felling
      SET TYPE = CASE
        WHEN TYPE = 'NEGATIVE' THEN 'NEUTRAL'
        WHEN TYPE = 'NEUTRAL' THEN 'POSITIVE'
        ELSE TYPE
      END
      WHERE FEEL_ID IN (
        SELECT FEEL_ID
        FROM human
        WHERE PLACE_ID = city_id
      );

    ELSIF city_rating < 0 THEN
      UPDATE felling
      SET TYPE = 'NEGATIVE'
      WHERE FEEL_ID IN (
        SELECT FEEL_ID
        FROM human
        WHERE PLACE_ID = city_id
      );

      UPDATE emotion
      SET TYPE = 'NEGATIVE'
      WHERE EMOTION_ID IN (
        SELECT EMOTION_ID
        FROM human
        WHERE PLACE_ID = city_id
      );

    ELSE
    END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;



CREATE TRIGGER trg_update_city_rating
AFTER INSERT ON relationships_to_human
FOR EACH ROW
EXECUTE FUNCTION update_city_rating();
```

Удаление

```
CREATE OR REPLACE FUNCTION update_city_rating_on_delete() RETURNS TRIGGER AS $$
DECLARE
  city_id INT;
  relationship_type TEXT;
  city_rating FLOAT;
BEGIN
  SELECT h.PLACE_ID, r.TYPE INTO city_id, relationship_type
  FROM human h
  JOIN relationships r ON r.RELATIONSHIPS_ID = OLD.RELATIONSHIPS_ID
  WHERE h.HUMAN_ID = OLD.HUMAN_ID;

  SELECT RATING INTO city_rating FROM place WHERE PLACE_ID = city_id;

  IF relationship_type = 'ENEMIES' THEN
    city_rating := city_rating + 1.5;
  ELSIF relationship_type = 'FRIENDS' THEN
    city_rating := city_rating - 0.5;
  ELSIF relationship_type = 'MARRIED' THEN
    city_rating := city_rating - 1.0;
  END IF;

  UPDATE place SET RATING = city_rating WHERE PLACE_ID = city_id;

  IF city_rating > 10 THEN
    UPDATE emotion
    SET TYPE = CASE
      WHEN TYPE = 'NEGATIVE' THEN 'NEUTRAL'
      WHEN TYPE = 'NEUTRAL' THEN 'POSITIVE'
      ELSE TYPE
    END
    WHERE EMOTION_ID IN (
      SELECT EMOTION_ID
      FROM human
      WHERE PLACE_ID = city_id
    );

    UPDATE felling
    SET TYPE = CASE
      WHEN TYPE = 'NEGATIVE' THEN 'NEUTRAL'
      WHEN TYPE = 'NEUTRAL' THEN 'POSITIVE'
      ELSE TYPE
    END
    WHERE FEEL_ID IN (
      SELECT FEEL_ID
      FROM human
      WHERE PLACE_ID = city_id
    );

  ELSIF city_rating < 0 THEN
    UPDATE felling
    SET TYPE = 'NEGATIVE'
    WHERE FEEL_ID IN (
      SELECT FEEL_ID
      FROM human
      WHERE PLACE_ID = city_id
    );

    UPDATE emotion
    SET TYPE = 'NEGATIVE'
    WHERE EMOTION_ID IN (
```

```
        SELECT EMOTION_ID
        FROM human
        WHERE PLACE_ID = city_id
      );

  END IF;

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;



CREATE TRIGGER trigger_update_city_rating_on_delete
AFTER DELETE ON relationships_to_human
FOR EACH ROW
EXECUTE FUNCTION update_city_rating_on_delete();
```

Вывод

Проведя эту работу, я понял, что такое нормализация, какие существуют нормальные форму и как создавать триггеры.