Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по дисциплине

«Базы данных»

Вариант №1377

Выполнил:

Студент группы P3113

Султанов А.Р.

Проверил:

Горбунов М.В.

г. Санкт-Петербург

2023г.

Оглавление

Задание

Задание.

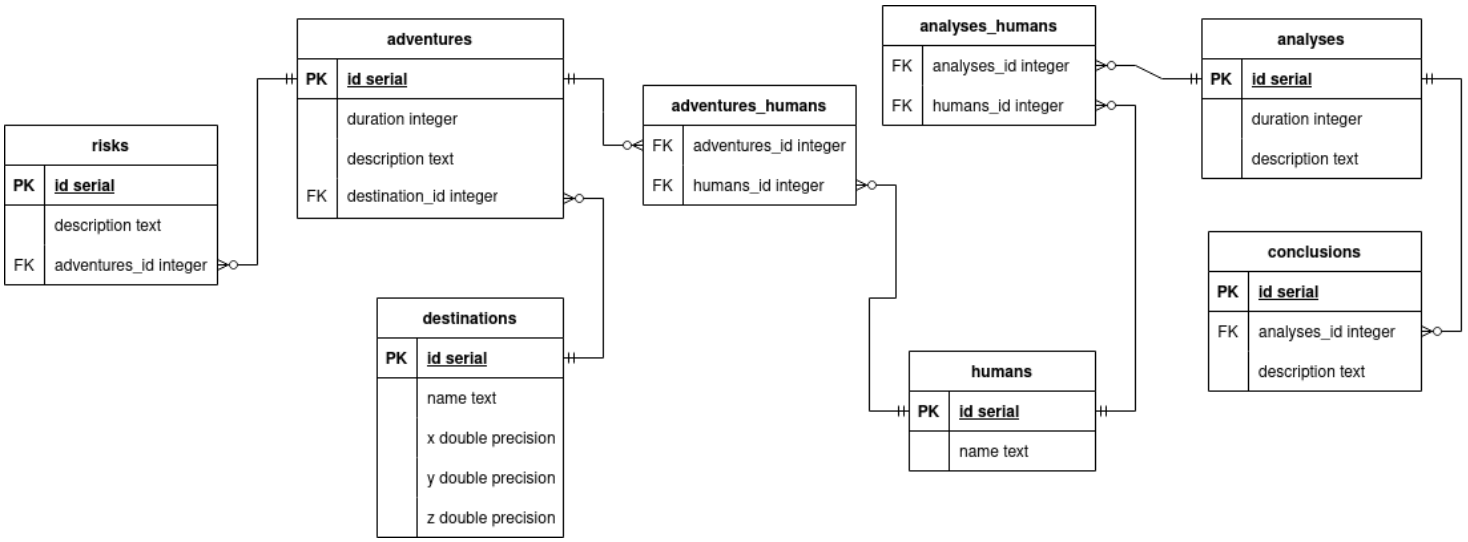
Для отношений, полученных при построении предметной области из лабораторной работы №1, выполните следующие действия:

- 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.

Отчёт по лабораторной работе должен содержать:

1. Текст задания.
2. Исходная, нормализованная и денормализованная модели.
3. Ответы на вопросы, представленные в задании.
4. Функция и триггер на языке PL/pgSQL
5. Выводы по работе.


adventures
PK id serial
duration integer
description text
FK destination_id integer
risks
PK id serial
description text
FK adventures_id integer
destinations
PK id serial
name text
x double precision
y double precision
z double precision
adventures_humans
FK adventures_id integer
FK humans_id integer
analyses_humans
FK analyses_id integer
FK humans_id integer
humans
PK id serial
name text
analyses
PK id serial
duration integer
description text
conclusions
PK id serial
FK analyses_id integer
description text

Функциональные зависимости

Во всех отношениях, в которых есть PK (id), от них зависят все остальные атрибуты. Помимо этого:

destinations: (x, y, z) -> name

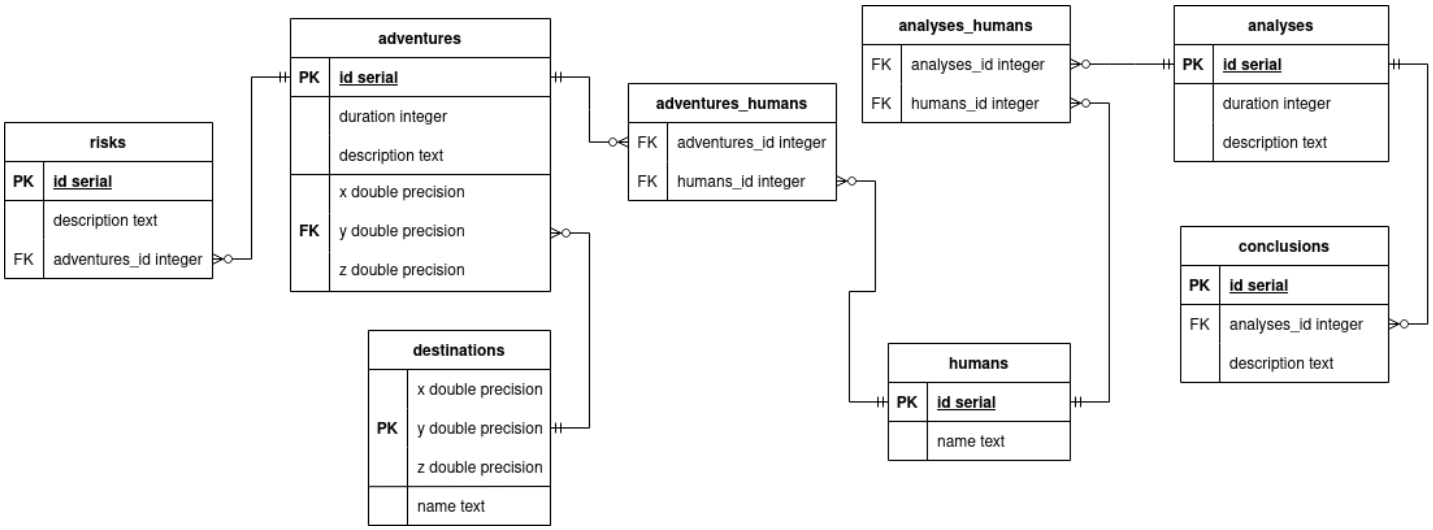
Вид 1NF

Модель уже соответствует 1NF - у каждой таблицы есть первичный ключ. благодаря чему все строки уникальны. Все столбцы не пытаются хранить в себе составные значения и значения атрибутов однородны (один и тот же тип). Также, в рамках отдельно взятой таблицы, имена столбцов уникальны и нет самодельных “массивов”. На этом шаге у модели не видно недостатков (по NF).

Вид 2NF

Модель уже соответствует 2NF, так как удовлетворяет 1NF и все атрибуты зависят от ключа полностью (нет составных ключей). На этом шаге у модели не видно недостатков (по NF).

Вид 3NF



Для удовлетворения 3NF стоит убрать **id** из таблицы destinations. Вместо этого, стоит сделать первичным ключом составной ключ **(x, y, z)** и заменить соответствующие обращения к этой таблице. Причина этого изменения - наличие в таблице destinations транзитивной функциональной зависимости: **id** -> **(x, y, z)** -> **name**.

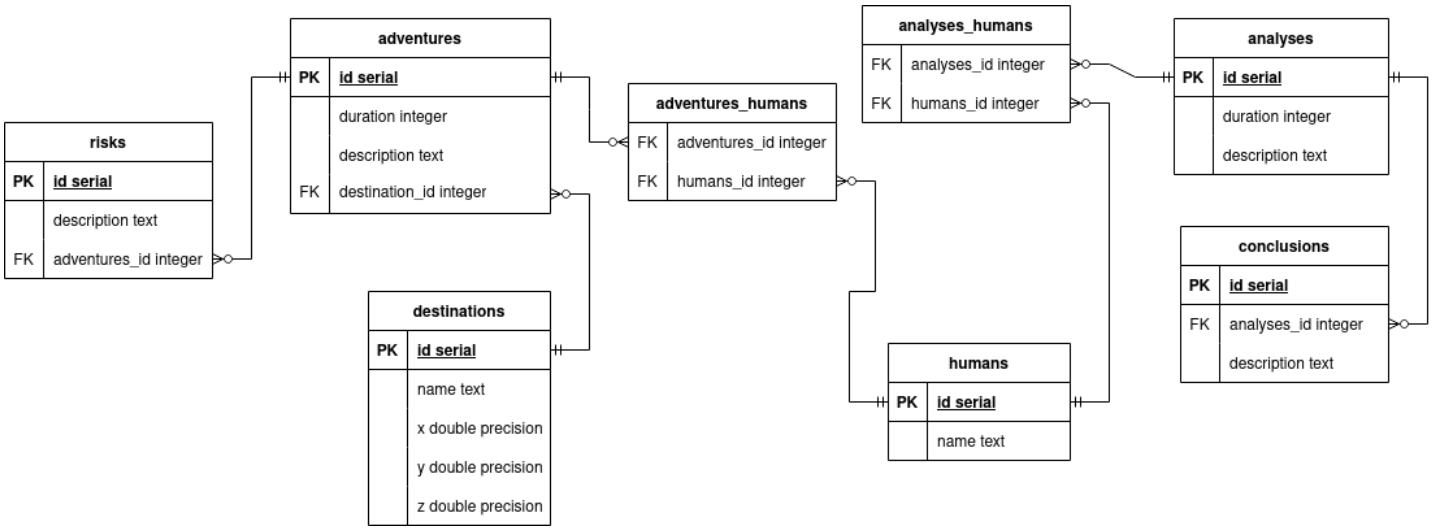
В результате, после приведения отношений к 3 нормальной форме (2NF и так соблюдалось), у **destinations** был убран неуместный id, и PK стал составным - координатами. Плюсом является то, что в таблице исчезла транзитивность и обращение к этой таблице (foreign key) стало чуть более явным - задающимся через координаты (x, y, z). Минус - то же самое обращение к таблице - для обращения к destinations теперь нужно иметь сразу три поля x, y, z и делать их составным указателем, что увеличивает размер таблиц, ссылающихся на destinations и делает обращение менее удобным.

Вид BCNF

После изменений в предыдущем шаге, модель удовлетворяет BCNF, так как удовлетворяется 3NF и нет такого составного ключа, что какая-то его часть зависит от неключевого столбца. На этом шаге у модели не видно недостатков (по NF).

Полезные денормализации

Как было сказано ранее, пусть становление **(x, y, z)** первичным составным ключом и позволило удовлетворить 3NF, но это изменение повлекло ряд неприятных последствий: Все обращения к **destinations** теперь требуют наличия 3 столбцов **(x, y, z)**, что, довольно прозаично, выглядит как не совсем оправданный эксцесс. Таким образом, вполне оптимальным изменением будет вернуться к изначальному варианту и сделать PK **id**, а **(x, y, z)** - простыми атрибутами. Так, обращение к **destinations** будет требовать всего одного столбца в таблице и теперь у этой таблицы несоставной ключ, что также является позитивным изменением.



Функция

```
CREATE
OR REPLACE FUNCTION distance(
    x1 adventures.x % TYPE,
    y1 adventures.y % TYPE,
    z1 adventures.z % TYPE,
    x2 adventures.x % TYPE,
    y2 adventures.y % TYPE,
    z2 adventures.z % TYPE
) returns DOUBLE PRECISION LANGUAGE plpgsql as
$$
BEGIN
RETURN ABS(
    SQRT(POWER(x1, 2) + POWER(y1, 2) + POWER(z1, 2)) - SQRT(POWER(x2, 2) + POWER(y2, 2) +
POWER(z2, 2))
);
END;
$$;

CREATE OR REPLACE FUNCTION min_distance_between_human_adventure_destinations(human1
humans.id % TYPE, human2 humans.id % TYPE) returns SETOF DOUBLE PRECISION LANGUAGE
plpgsql as
$func$
BEGIN
RETURN QUERY
WITH adventure_fectch1 AS (
    SELECT
          adventures.x x1,
          adventures.y y1,
          adventures.z z1
    FROM
          humans
          INNER JOIN adventures_humans ON adventures_humans.humans_id = humans.id
          INNER JOIN adventures ON adventures.id = adventures_humans.adventures_id
    WHERE
          humans.id = human1
```

```
), adventure_fectch2 AS (
    SELECT
          adventures.x x2,
          adventures.y y2,
          adventures.z z2
    FROM
          humans
          INNER JOIN adventures_humans ON adventures_humans.humans_id = humans.id
          INNER JOIN adventures ON adventures.id = adventures_humans.adventures_id
    WHERE
          humans.id = human2
), distances AS (
    SELECT
          distance(x1, y1, z1, x2, y2, z2) distance
    FROM
          adventure_fectch1
          CROSS JOIN adventure_fectch2
) SELECT MIN(distance) FROM distances;
END;
$func$;
```

Триггер

```
CREATE OR REPLACE FUNCTION delete_random_adventures_humans() RETURNS TRIGGER AS
$$
BEGIN

DELETE FROM
    adventures_humans ah1
WHERE
    ah1.adventures_id = NEW.adventures_id
    AND ah1.humans_id = (
          SELECT
          ah2.humans_id hid
          FROM
          adventures
          INNER JOIN adventures_humans ah2 ON ah2.adventures_id = adventures.id
          WHERE
          adventures.id = NEW.adventures_id
          ORDER BY
          random()
          LIMIT
          1
    );
    RETURN NEW;

END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER risk_trigger
AFTER INSERT ON risks FOR EACH ROW
EXECUTE PROCEDURE delete_random_adventures_humans();
```

Заключение

В рамках данной лабораторной работы я познакомился с нормализацией, нормальными формами и функциональными связями. Также на практике изучил функции, триггеры в PostgreSQL.