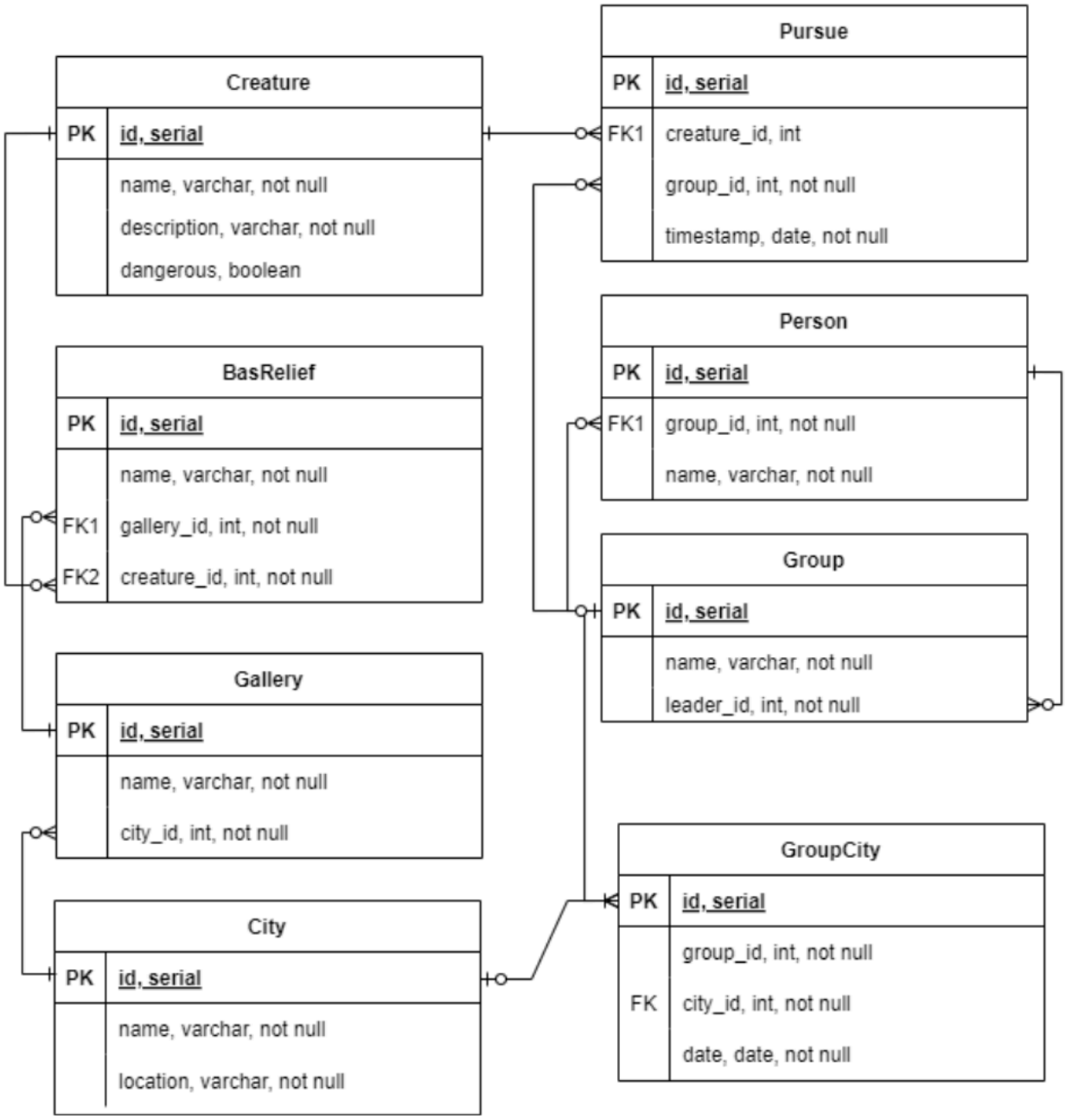
ЗАДАНИЕ

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.



РЕШЕНИЕ

Функциональные зависимости

1. Creature

- `id_serial → name, description, dangerous`

---

2. City

- `id_serial → name, location`

---

3. Gallery

- `id_serial → name, city_id`

---

4. BasRelief

- `id_serial → name, gallery_id, creature_id`

---

5. “Group”

- `id_serial → name, leader_id`
- `leader_id → id_serial`

---

6. Person

- `id_serial → group_id, name`

---

7. GroupCity

- `id → group_id, city_id, date`

---

8. Pursue

Атрибуты: `(creature_id, group_id, timestamp)`
Ключ: составной `(creature_id, group_id, timestamp)`
FD:

- `id → group_id, name, timestamp)`

Нормализация

1НФ:

Отношение находится в 1НФ, если все его атрибуты являются атомарными, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Все таблицы имеют первичные ключи (`id_serial`), и все атрибуты используют скалярные типы: `int` , `varchar` , `boolean` , `date` . Массивов, списков или составных атрибутов не наблюдается.

Вывод: модель удовлетворяет 1НФ.

2НФ:

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от первичного ключа.

Все таблицы используют surrogate keys (`id_serial`) как первичные ключи — это скалярные значения. В таблицах, где ключ составной (например, `GroupCity` , `Pursue`), все неключевые атрибуты (например, `date` , `timestamp`) функционально зависят от всего составного ключа:

- В `Pursue` — `timestamp` зависит от `(creature_id, group_id)` ;
- В `GroupCity` — `date` зависит от `(group_id, city_id)` .

Вывод: модель удовлетворяет 2НФ.

3НФ:

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут не зависит транзитивно от первичного ключа.

**вывод: модель удовлетворяет 3НФ, т. к. она находится в 2НФ и в ней нет транзитивных отношений;;

BCNF:

Отношение находится в BCNF, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.

1. **Есть фз, детерминант которой не ключ**:
   `leader_id→id
2. **BCNF требует**, чтобы для любой фз X → Y, X было суперключом.
   `leader_id` не суперключ (в исходной схеме только `id` — PK).

Вывод: `Group` в исходном виде не удовлетворяет BCNF.

Декомпозиция

1. **Оставляем в `Group` только то, что детерминируется ключом `id_serial`:**

```
    Group (
    id  SERIAL    PRIMARY KEY,
    name       VARCHAR   NOT NULL );
    ```

```nginx
id → name
```

```
— детерминант (`id`) – суперключ.
```

2. **Выносим связь «группа ↔ её лидер» в отдельную таблицу `GroupLeader`:**

```
GroupLeader (
group_id   INT       NOT NULL  REFERENCES Group(id),
leader_id  INT       NOT NULL  REFERENCES Person(id),
PRIMARY KEY (group_id),   UNIQUE (leader_id)
```

Денормализация

Полезным внедрением денормализации будет создание представлений, которые помогут фильтровать объединения таблиц

```
CREATE VIEW GroupWithLeader AS
SELECT
  g.id AS group_id,
  g.name AS group_name,
  p.name AS leader_name
FROM "Group" g
JOIN GroupLeader ON g.id = GroupLeader.group_id
```

```
JOIN Person       ON Person.id = GroupLeader.leader_id;
```

представление `GroupWithLeader` — группа + имя лидера обеспечивает читабельность для работы с группами и их лидерами, а также упрощает работу с запросами.

```
 group_id |      group_name        |         leader_name
----------+------------------------+-----------------------------
        1 | Археологи Мискатоника | Профессор Дайер
        2 | Группа выживших        | Неизвестный преследователь
        3 | Эскадра наблюдения     | Офицер Лейк
```

Триггеры и функции

Для созданной в рамках лабораторной работы №1 предметной области были добавлены триггер trg_check_creature_dangerous со связанная с ним функция check_creature_dangerous().

```
CREATE OR REPLACE FUNCTION check_creature_dangerous()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM creature WHERE id_serial = NEW.creature_id
AND dangerous = true) THEN
        UPDATE creature
        SET dangerous = true
        WHERE id_serial = NEW.creature_id;
        RAISE NOTICE 'Существо (ID=%) признано опасным группой (ID=%).
Статус обновлён.', NEW.creature_id, NEW.group_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_creature_dangerous
BEFORE INSERT ON pursue
FOR EACH ROW
EXECUTE FUNCTION check_creature_dangerous();
```