

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

Факультет программной инженерии и компьютерной техники

Дисциплина: Базы Данных

Лабораторная работа №5

Выполнил:
Серебренникова В. В.

Группа:
P33202

Проверил:
Нечкасова О. А.

Санкт-Петербург
2023

Оглавление

Оглавление.....	2
Текст задания.....	3
Описание предметной области	4
Триггеры	7
Процедуры	16
Функции.....	17
Бизнес-процессы	20
Бизнес-процесс “Сражение”	20
Бизнес-процесс “Повышение уровня”	28
Индексы	34
Сценарии	34
EXPLAIN ANALYZE Ситуация до добавления индексов	35
EXPLAIN ANALYZE Ситуация после добавления индексов	37
Выводы.....	39

Текст задания

Для выполнения лабораторной работы №5 необходимо:

- Добавить в ранее созданную базу данных триггеры для обеспечения комплексных ограничений целостности.
- Реализовать функции и процедуры на основе описания бизнес-процессов, определенных при описании предметной области. Должна быть обеспечена проверка корректности вводимых данных для созданных функций и процедур.
- Необходимо произвести анализ использования созданной базы данных, выявить наиболее часто используемые объекты базы данных, виды запросов к ним. Результаты должны быть представлены в виде текстового описания.
- На основании полученного описания требуется создать подходящие индексы и доказать, что они будут полезны для представленных в описании случаев использования базы данных.

Отчёт по лабораторной работе должен содержать:

- Титульный лист.
- Текст задания.
- Код триггеров, функций, процедур.
- Описание наиболее часто используемых сценариев при работе с базой данных.
- Описание индексов и обоснование их использования.
- Выводы по работе.

Описание предметной области

Genshin Impact - компьютерная игра в жанре action-adventure от третьего лица. В игре присутствуют элементы РПГ и открытый мир. Персонажи и то, как с ними работать, какое давать им оружие и артефакты - основной интерес игроков Genshin Impact, потому что игровой процесс целиком зависит от персонажей, которыми обладает игрок, и от того, как он их развивает, и по этой причине база данных, которую я бы хотела создать на предмете “Базы Данных”, сосредоточена вокруг персонажей.

Центральное место в базе данных займут персонажи и их снаряжение и характеристики. Персонажи смогут сражаться со врагами при помощи специального бизнес-процесса - сражения.

Персонажи

Персонажи будут содержаться в стержневой сущности “Персонажи”, у которой есть характеристическая сущность “Характеристики персонажа”.

Враги

Для хранения врагов есть характеристическая сущность “Список врагов”, выступающая в роли энциклопедии по всем существующим врагам. Для перечисления врагов, в данный момент существующих в мире игрока, создана сущность “Враги”, ссылающаяся на список врагов и содержащая значение здоровья данного врага.

Снаряжение

Существуют стержневые сущности “Артефакты” и “Оружие”, содержащие информацию о типах и видах артефактов и оружия. Характеристическая сущность “Снаряжение” связывает персонажа и все предметы, которыми он пользуется (одно оружие и от 0 до 5 артефактов разных видов).

Материалы

Существует характеристическая сущность “Материалы”, описывающая существующие в игре материалы. С помощью материалов можно повышать уровень персонажей (связь между материалами и персонажами указана в ассоциативной сущности “возвышение”). Материалы добываются при уничтожении врагов (связь между врагами и материалами указана в ассоциативной сущности “Предметы, падающие со врага”). Для учёта того, какие предметы находятся в инвентаре игрока, существует сущность “Инвентарь”, содержащая записи обо всех добавленных в инвентарь материалах по их айди.

Сражения

Для реализации процесса сражений добавляются ещё две сущности - ассоциативная “Сражения”, в которой содержатся сами сражения, и связанная с ней характеристическая “Ход сражения”, в которой описывается ход сражения.

Список выделенных ограничений:

1. CONSTRAINT ch_check для сущности “Персонажи” проверяет следующее:
 - a. Значения характеристики ch_element должны быть равны одному из следующих: ‘anemo’, ‘geo’, ‘electro’, ‘dendro’, ‘hydro’, ‘pyro’, ‘cryo’.
 - b. Значения характеристики weapon_type должны быть равны одному из следующих: ‘bow’, ‘claymore’, ‘polearm’, ‘sword’, ‘catalyst’.
 - c. Значения характеристики rarity должно быть равно одному из следующих: 4, 5.

2. CONSTRAINT ch_ch_check для сущности “Характеристики персонажей” проверяет следующее:
 - a. Значения характеристики ch_rank должны быть равны одному из следующих: 1, 2, 3, 4, 5, 6.
 - b. Значения характеристики target должны быть равны одному из следующих: 0, 579100, 854125, 1195925, 1611875, 3423125.
 - c. Значения характеристики status должны быть равны одному из следующих: ‘alive’, ‘defeated’.
 - d. Значение характеристики energy должно быть между 0 и 110 или равно им.
 - e. Значение характеристики hp должно быть меньше или равно значению характеристики max_hp.
3. CONSTRAINT en_list_check для сущности “Список врагов” проверяет следующее:
 - a. Значения характеристики en_element должны быть равны одному из следующих: ‘anemo’, ‘geo’, ‘electro’, ‘dendro’, ‘hydro’, ‘pyro’, ‘cryo’, ‘base’.
 - b. Значения характеристики en_class должны быть равны одному из следующих: ‘common’, ‘elite’, ‘normal boss’, ‘weekly boss’.
4. CONSTRAINT wear_check для сущности “Оружие” проверяет следующее:
 - a. Значения характеристики weapon_type должны быть равны одному из следующих: ‘bow’, ‘claymore’, ‘polearm’, ‘sword’, ‘catalyst’.
 - b. Значения характеристики rarity должны быть равны одному из следующих: 1, 2, 3, 4, 5.
5. CONSTRAINT art_check для сущности “Артефакты” проверяет следующее:
 - a. Значения характеристики art_type должны быть равны одному из следующих: ‘flower of life’, ‘plume of death’, ‘sands of eon’, ‘goblet of eonothem’, ‘circlet of logos’.
 - b. Значения характеристики rarity должны быть равны одному из следующих: 1, 2, 3, 4, 5.

Реализуемые бизнес-процессы

- 1) Сражение. При активации сражения между персонажем и врагом персонаж и враг по очереди ходят (персонаж первый).
 - a. Ход персонажа, у врага снимается здоровье в размере: округленное (атака персонажа * случайное значение (в диапазоне от 0 до 1) * 10 + атака персонажа * 3.5).
 - i. Если показатель “энергия” персонажа < 0 , наносится обычная атака. К показателю “энергия” добавляется 10.
 - ii. Если показатель “энергия” персонажа ≥ 100 , то персонаж наносит дополнительную атаку. После этого “энергия” = 0.
 - b. Ход врага, у персонажа снимается здоровье в размере: округленное (атака врага * случайное значение (в диапазоне от 0 до 1)).

Этот шаг повторяется, пока здоровье кого-то из них не станет равно 0. Если хп персонажа = 0, за него больше нельзя сражаться, и его статус меняется, но он остаётся в базе данных. Если хп врага = 0, враг умирает, переходим к добыче опыта, выводится статистика.

- 2) Добыча опыта. При победе над врагом количество врагов в мире снижается (удаляется враг из таблицы), а в инвентарь добавляются предметы с него (предметы из графы loot) и опыт. К каждому врагу привязано несколько предметов, и каждый из них может выпасть с вероятностью 30% - то есть, может выпасть несколько предметов, а может не выпасть ни одного. Уровень персонажа можно увеличить только по достижении цели - конкретного количества опыта, и игрок должен сражаться со врагами пока его персонаж не получит достаточно опыта.
 - a. К опыту персонажа добавляется количество опыта, получаемого со врага.
 - b. Враг умирает.
 - c. Если опыт персонажа $>$ цели, опыт персонажа = цели.
 - d. Если опыт персонажа = цели, увеличивается уровень персонажа:
 - i. Из инвентаря пропадают предметы, нужные персонажу для увеличения уровня.
 - ii. Атака, особая атака и макс. здоровье персонажа увеличиваются на 1%.
 - iii. Уровень персонажа увеличивается на 1.
 - iv. Опыт персонажа становится равным 0.
 - v. Здоровье персонажа = Макс. здоровье персонажа

Триггеры

1. INSERT

a. Триггер before_insert_on_ch_characteristics

Перед вставкой новой записи в таблицу CH_CHARACTERISTICS триггер проверяет, что персонаж существует в таблице CHARACTERS:

```
CREATE OR REPLACE FUNCTION check_character_exists()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM CHARACTERS WHERE ID = NEW.id) THEN
RAISE EXCEPTION 'Персонаж с ID % не существует', NEW.id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_ch_characteristics
BEFORE INSERT ON CH_CHARACTERISTICS
FOR EACH ROW
EXECUTE FUNCTION check_character_exists();
```

Пример:

```
insert into ch_characteristics values (100, 5, 15, 1611875, 800, 8000, 80000, 0, 'alive');
ERROR: Персонаж с ID 100 не существует
CONTEXT: функция PL/pgSQL check_character_exists(), строка 4, оператор RAISE
```

b. Триггер before_insert_on_enemies

Перед вставкой новой записи в таблицу ENEMIES триггер проверяет, что враг существует в таблице EN_LIST:

```
CREATE OR REPLACE FUNCTION check_enemie_exists()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM EN_LIST WHERE ID = NEW.en_id) THEN
RAISE EXCEPTION 'Тип врага с ID % не существует', NEW.en_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_enemies
BEFORE INSERT ON ENEMIES
FOR EACH ROW
EXECUTE FUNCTION check_enemie_exists();
```

Пример:

```
insert into enemies(en_id, hp) values(100, 100);
```

```
ERROR: Тип врага с ID 100 не существует
```

```
CONTEXT: функция PL/pgSQL check_enemie_exists(), строка 4, оператор RAISE
```

c. Триггер before_insert_on_fight_recordings

Перед вставкой новой записи в таблицу FIGHT_RECORDINGS триггер проверяет, что битва существует в таблице FIGHTS:

```
CREATE OR REPLACE FUNCTION check_fight_exists()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM FIGHTS WHERE ID = NEW.Fight_ID) THEN
RAISE EXCEPTION 'Битва с ID % не существует', NEW.Fight_ID;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_fights
BEFORE INSERT ON FIGHT_RECORDINGS
FOR EACH ROW
EXECUTE FUNCTION check_fight_exists();
```

Пример:

```
insert into fight_recordings(fight_id, character_hp, character_dmg, enemy_hp, enemy_dmg)
values (1000, 100, 100, 100, 100)
```

```
ERROR: Битвы с ID 1000 не существует
```

```
CONTEXT: функция PL/pgSQL check_fight_exists(), строка 4, оператор RAISE
```

d. Триггер before_insert_on_ascention

Перед вставкой новой записи в таблицу ASCENTION триггер проверяет, что материал существует в таблице MATERIALS, а персонаж существует в таблице CHARACTERS:

```
CREATE OR REPLACE FUNCTION check_material_and_character_exist()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM MATERIALS WHERE ID = NEW.item_id) THEN
RAISE EXCEPTION 'Материала с ID % не существует', NEW.item_id;
ELSEIF NOT EXISTS (SELECT 1 FROM CHARACTERS WHERE ID =
NEW.character_id) THEN
RAISE EXCEPTION 'Персонажа с ID % не существует', NEW.character_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_ascention
```



```
BEFORE INSERT ON ASCENTION
FOR EACH ROW
EXECUTE FUNCTION check_material_and_character_exist();
```

Примеры:

```
insert into ascention(character_id, item_id)
values(200, 5);
ERROR: Персонажа с ID 200 не существует
CONTEXT: функция PL/pgSQL check_material_and_character_exist(), строка 6, оператор RAISE
```

```
insert into ascention(character_id, item_id)
values(5, 200);
ERROR: Материала с ID 200 не существует
CONTEXT: функция PL/pgSQL check_material_and_character_exist(), строка 4, оператор RAISE
```

е. Триггер before_insert_on_loot

Перед вставкой новой записи в таблицу LOOT триггер проверяет, что материал существует в таблице MATERIALS, а враг существует в таблице EN_LIST:

```
CREATE OR REPLACE FUNCTION check_material_and_enemie_exist()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM MATERIALS WHERE ID = NEW.item_id) THEN
RAISE EXCEPTION 'Материала с ID % не существует', NEW.item_id);
ELSEIF NOT EXISTS (SELECT 1 FROM EN_LIST WHERE ID = NEW.enemie_id) THEN
RAISE EXCEPTION 'Врага с ID % не существует', NEW.enemie_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_loot
BEFORE INSERT ON LOOT
FOR EACH ROW
EXECUTE FUNCTION check_material_and_enemie_exist();
```

Пример:

```
insert into loot(enemie_id, item_id)
values(200, 5);
ERROR: Врага с ID 200 не существует
CONTEXT: функция PL/pgSQL check_material_and_enemie_exist(), строка 6, оператор RAISE
```

```
insert into loot(enemie_id, item_id)
values(5, 200);
ERROR: Материала с ID 200 не существует
CONTEXT: функция PL/pgSQL check_material_and_enemie_exist(), строка 4, оператор RAISE
```

ф. Триггер before_insert_on_fights

Перед вставкой новой записи в таблицу FIGHTS триггер проверяет, что персонаж

существует в таблице CHARACTERS, а враг существует в таблице ENEMIES:

```
CREATE OR REPLACE FUNCTION check_character_and_enemie_exist()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM CHARACTERS WHERE ID = NEW.character_id)
THEN
RAISE EXCEPTION 'Персонажа с ID % не существует', NEW.character_id;
ELSEIF NOT EXISTS (SELECT 1 FROM EN_LIST WHERE ID = NEW.enemie_id) THEN
RAISE EXCEPTION 'Врага с ID % не существует', NEW.enemie_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_fights
BEFORE INSERT ON FIGHTS
FOR EACH ROW
EXECUTE FUNCTION check_character_and_enemie_exist();
```

Пример:

```
insert into fights(character_id, enemie_id, fight_time)
values(200, 5, NOW());
```

```
ERROR: Персонажа с ID 200 не существует
CONTEXT: функция PL/pgSQL check_character_and_enemie_exist(), строка 4, оператор RAISE
```

```
insert into fights(character_id, enemie_id, fight_time)
values(5, 500, NOW());
```

```
ERROR: Врага с ID 500 не существует
CONTEXT: функция PL/pgSQL check_character_and_enemie_exist(), строка 6, оператор RAISE
```

g. Триггер before_insert_on_inventory

Перед вставкой новой записи в таблицу INVENTORY триггер проверяет, что материал существует в таблице MATERIALS:

```
CREATE OR REPLACE FUNCTION check_material_exists()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM MATERIALS WHERE ID = NEW.material_id) THEN
RAISE EXCEPTION 'Материала с ID % не существует', NEW.material_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_inventory
BEFORE INSERT ON INVENTORY
FOR EACH ROW
EXECUTE FUNCTION check_material_exists();
```

Пример:

```
insert into inventory(material_id) values(500);
```

ERROR: Материала с ID 500 не существует

CONTEXT: функция PL/pgSQL check_material_exists(), строка 4, оператор RAISE

h. Триггер before_insert_on_equipment

Перед вставкой новой записи в таблицу EQUIPMENT триггер проверяет, что персонаж существует в таблице CHARACTERS или равен NULL, оружие существует в таблице WEAPONS или равно NULL, артефакт существует в таблице ARTIFACTS или равен NULL:

```
CREATE OR REPLACE FUNCTION check_for_equipment()
RETURNS TRIGGER AS $$
BEGIN
IF NOT EXISTS (SELECT 1 FROM CHARACTERS WHERE ID = NEW.character_id)
AND NEW.character_id IS NOT NULL THEN
RAISE EXCEPTION 'Персонажа с ID % не существует', NEW.character_id;
ELSEIF NOT EXISTS (SELECT 1 FROM ARTIFACTS WHERE ID = NEW.flower_id)
AND NEW.flower_id IS NOT NULL THEN
RAISE EXCEPTION 'Артефакта с ID % не существует', NEW.flower_id;
ELSEIF NOT EXISTS (SELECT 1 FROM ARTIFACTS WHERE ID = NEW.plume_id)
AND NEW.plume_id IS NOT NULL THEN
RAISE EXCEPTION 'Артефакта с ID % не существует', NEW.plume_id;
ELSEIF NOT EXISTS (SELECT 1 FROM ARTIFACTS WHERE ID = NEW.sands_id)
AND NEW.sands_id IS NOT NULL THEN
RAISE EXCEPTION 'Артефакта с ID % не существует', NEW.sands_id;
ELSEIF NOT EXISTS (SELECT 1 FROM ARTIFACTS WHERE ID = NEW.goblet_id)
AND NEW.goblet_id IS NOT NULL THEN
RAISE EXCEPTION 'Артефакта с ID % не существует', NEW.goblet_id;
ELSEIF NOT EXISTS (SELECT 1 FROM ARTIFACTS WHERE ID = NEW.circlet_id)
AND NEW.circlet_id IS NOT NULL THEN
RAISE EXCEPTION 'Артефакта с ID % не существует', NEW.circlet_id;
ELSEIF NOT EXISTS (SELECT 1 FROM WEAPONS WHERE ID = NEW.weapon_id)
AND NEW.weapon_id IS NOT NULL THEN
RAISE EXCEPTION 'Оружия с ID % не существует', NEW.weapon_id;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_insert_on_equipment
BEFORE INSERT ON EQUIPMENT
FOR EACH ROW
EXECUTE FUNCTION check_for_equipment();
```

Пример:

```
INSERT INTO EQUIPMENT(character_id, weapon_id, flower_id, plume_id, sands_id,
goblet_id, circlet_id)
VALUES(200, 1, 1, 1, 1, 1, 1);
```

ERROR: Персонажа с ID 200 не существует
CONTEXT: функция PL/pgSQL check_for_equipment(), строка 4, оператор RAISE

```
INSERT INTO EQUIPMENT(character_id, weapon_id, flower_id, plume_id, sands_id,  
goblet_id, circlet_id)  
VALUES(1, 250, 1, 1, 1, 1, 1);
```

ERROR: Оружия с ID 250 не существует
CONTEXT: функция PL/pgSQL check_for_equipment(), строка 16, оператор RAISE

```
INSERT INTO EQUIPMENT(character_id, weapon_id, flower_id, plume_id, sands_id,  
goblet_id, circlet_id)  
VALUES(1, 1, 500, 1, 1, 1, 1);
```

ERROR: Артефакта с ID 500 не существует
CONTEXT: функция PL/pgSQL check_for_equipment(), строка 6, оператор RAISE

i. Триггер after_insert_on_characters

После вставки новой записи (нового персонажа) в таблицу CHARACTERS триггер вставляет новую соответствующую запись в таблицу CH_CHARACTERISTICS:

```
CREATE OR REPLACE FUNCTION add_to_ch_characteristics()  
RETURNS TRIGGER AS $$  
BEGIN  
INSERT INTO CH_CHARACTERISTICS(id, ch_rank, experience, target, hp, max_hp, atk,  
energy, status)  
VALUES(NEW.id, 1, 0, 579100, 20000, 20000, 1500, 0, 'alive');  
INSERT INTO EQUIPMENT(character_id)  
VALUES(NEW.id);  
RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER after_insert_on_characters  
AFTER INSERT ON CHARACTERS  
FOR EACH ROW  
EXECUTE FUNCTION add_to_ch_characteristics();
```

Пример:

```
insert into characters (name, ch_element, rarity, weapon_type, constellation, birthday)  
values ('Navia', 'geo', 5, 'claymore', 'Rosa Multiflora', 'december 20th');
```

2. UPDATE

a. Триггеры before_update_on_characters, before_update_on_en_list, before_update_on_materials

- i. Перед обновлением записи из таблицы CHARACTERS триггер выводит сообщение о том, что менять данные из этой таблицы запрещено:

```
CREATE OR REPLACE FUNCTION restrict_update()  
RETURNS TRIGGER AS $$
```

```

BEGIN
RAISE EXCEPTION 'Менять данные в таблице запрещено.';
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER before_update_on_characters
BEFORE UPDATE ON CHARACTERS
FOR EACH ROW
EXECUTE FUNCTION restrict_update();

```

Пример:

```

update characters
set name = 'no name'
where id = 50

```

ERROR: Менять данные в таблице запрещено.

CONTEXT: функция PL/pgSQL restrict_update(), строка 3, оператор RAISE

- ii. Перед обновлением записи из таблицы EN_LIST триггер выводит сообщение о том, что менять данные из этой таблицы запрещено:

```

CREATE TRIGGER before_update_on_en_list
BEFORE UPDATE ON EN_LIST
FOR EACH ROW
EXECUTE FUNCTION restrict_update();

```

Пример:

```

update en_list
set name = 'no name'
where id = 15

```

ERROR: Менять данные в таблице запрещено.

CONTEXT: функция PL/pgSQL restrict_update(), строка 3, оператор RAISE

- iii. Перед обновлением записи из таблицы MATERIALS триггер выводит сообщение о том, что менять данные из этой таблицы запрещено:

```

CREATE TRIGGER before_update_on_materials
BEFORE UPDATE ON MATERIALS
FOR EACH ROW
EXECUTE FUNCTION restrict_update();

```

Пример:

```

update materials
set name = 'no name'
where id = 15

```

ERROR: Менять данные в таблице запрещено.

CONTEXT: функция PL/pgSQL restrict_update(), строка 3, оператор RAISE

3. DELETE

a. Триггер before_delete

- i. Перед удалением записи из таблицы CHARACTERS триггер выводит информацию о том, что так делать нельзя:

```
CREATE OR REPLACE FUNCTION restrict_delete()
RETURNS TRIGGER AS $$
BEGIN
RAISE EXCEPTION 'Невозможно удалить данные из этой таблицы.';
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_delete_on_characters
BEFORE DELETE ON CHARACTERS
FOR EACH ROW
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from characters where id = 50
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

- ii. Перед удалением записи из таблицы EN_LIST триггер выводит информацию о том, что так делать нельзя:

```
CREATE TRIGGER before_delete_on_en_list
BEFORE DELETE ON EN_LIST
FOR EACH ROW
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from en_list where id = 30
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

- iii. Перед удалением записи из таблицы CH_CHARACTERISTICS триггер выводит информацию о том, что так делать нельзя:

```
CREATE TRIGGER before_delete_on_ch_characteristics
BEFORE DELETE ON CH_CHARACTERISTICS
FOR EACH ROW
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from ch_characteristics where id = 50
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

- iv. Перед удалением записи из таблицы ASCENTION триггер выводит информацию о том, что так делать нельзя:

```
CREATE TRIGGER before_delete_on_ascention  
BEFORE DELETE ON ASCENTION  
FOR EACH ROW  
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from ascention where id = 50
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

- v. Перед удалением записи из таблицы LOOT триггер выводит информацию о том, что так делать нельзя:

```
CREATE TRIGGER before_delete_on_loot  
BEFORE DELETE ON LOOT  
FOR EACH ROW  
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from loot where id = 50
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

- vi. Перед удалением записи из таблицы MATERIALS триггер выводит информацию о том, что так делать нельзя:

```
CREATE TRIGGER before_delete_on_materials  
BEFORE DELETE ON MATERIALS  
FOR EACH ROW  
EXECUTE FUNCTION restrict_delete();
```

Пример:

```
delete from materials where id = 50
```

ERROR: Невозможно удалить данные из этой таблицы.

CONTEXT: функция PL/pgSQL restrict_delete(), строка 3, оператор RAISE

Процедуры

1. Процедура “**turn_alive**” возвращает всех персонажей к жизни с максимальным здоровьем. Подобная процедура существует в самой видеоигре, по которой делалась база данных.

```
CREATE OR REPLACE PROCEDURE turn_alive()  
AS $$  
BEGIN  
UPDATE CH_CHARACTERISTICS  
SET status = 'alive', hp = max_hp;  
END;  
$$ LANGUAGE plpgsql;
```

Часть процедур находится в составе бизнес-процессов, поэтому приведена в отчете далее.

Функции

1. Функция “**get_characters_by_element**” для получения списка персонажей с заданным элементом:

```
CREATE OR REPLACE function get_characters_by_element(  
p_element text  
)  
returns table  
(  
id integer,  
name text,  
element text,  
weapon_type text  
)  
AS $$  
BEGIN  
return query(  
SELECT characters.id, characters.name, characters.ch_element, characters.weapon_type  
FROM CHARACTERS WHERE ch_element = p_element  
);  
END;  
  
$$ LANGUAGE plpgsql;
```

Пример:

```
select * from get_characters_by_element('geo')
```

	id integer	name text	element text	weapon_type text
1	1	Albedo	geo	sword
2	3	Arataki Itto	geo	claymore
3	27	Noelle	geo	claymore
4	28	Ningguang	geo	catalyst
5	44	Yun Jin	geo	polearm
6	45	Zhongli	geo	polearm
7	50	Navia	geo	claymore
8	51	Navia	geo	claymore

2. Функция “**get_characters_by_weapon**” для получения списка персонажей с заданным типом оружия:

```
CREATE OR REPLACE function get_characters_by_weapon(  
p_type text  
) returns table  
(
```

```

id integer,
name text,
element text,
weapon text
)
AS $$
BEGIN
RETURN QUERY(
SELECT characters.id, characters.name, characters.ch_element, characters.weapon_type
FROM CHARACTERS WHERE weapon_type = p_type
);
END;

$$ LANGUAGE plpgsql;

```

Пример:

```
select * from get_characters_by_weapon('claymore')
```

	id integer	name text	element text	weapon text
1	3	Arataki Itto	geo	claymore
2	5	Beidou	electro	claymore
3	9	Chongyun	cryo	claymore
4	12	Dory	electro	claymore
5	15	Freminet	cryo	claymore
6	17	Kaveh	dendro	claymore
7	27	Noelle	geo	claymore
8	30	Razor	electro	claymore
9	32	Sayu	anemo	claymore
10	40	Xinyan	pyro	claymore
11	50	Navia	geo	claymore
12	51	Navia	geo	claymore

3. Функция “get_characteristics” для получения списка характеристик персонажа:

```

CREATE OR REPLACE function get_characteristics(
p_name text
) returns table (
rank int,
exp int,
tg int,
health int,
max_health int,
dmg int,
eng int,

```









```

stat text
)
AS $$
BEGIN
RETURN QUERY(
SELECT ch_rank, experience, target, hp, max_hp, atk, energy, status FROM CH_CHARACTERISTICS
WHERE id = (SELECT id FROM CHARACTERS WHERE name = p_name)
);
END;
$$ LANGUAGE plpgsql;

```

Пример:

```
select * from get_characteristics('Albedo')
```

	rank integer 	exp integer 	tg integer 	health integer 	max_health integer 	dmg integer 	eng integer 	stat text 
1	6	0	854125	7103	7103	156	70	alive

Бизнес-процессы

Бизнес-процесс “Сражение”

1. **Функция “insert_into_fight_recordings”** добавляет в таблицу `fight_recordings` записи о ходе проведения сражения. В таблицу заносит айди сражения, базовые значения атак врага и персонажа и изменяющиеся по ходу сражения значения здоровья персонажа и врага.

```
create function insert_into_fight_recordings(  
f_ch_id INTEGER,  
f_en_id INTEGER  
)  
returns void  
language plpgsql  
as  
$$  
BEGIN  
  
    with res as  
    (  
    select  
    enemies.id as en_id,  
    hp as en_hp,  
    atk as en_atk  
    from enemies left join en_list  
    on enemies.en_id = en_list.id  
    where enemies.id = f_en_id  
    ),  
    res2 as  
    (  
    select  
    ch_characteristics.id as ch_id,  
    ch_characteristics.hp as ch_hp,  
    ch_characteristics.atk as ch_atk  
    from ch_characteristics  
    where ch_characteristics.id = f_ch_id  
    ),  
    f_id as  
    (  
    select id as f_id  
    from fights  
    order by id desc  
    limit 1  
    ),  
    selected as  
    (  
    select *
```

```

        from res
        cross join res2
        cross join f_id
    )

    insert
    into fight_recordings(fight_id, character_hp, character_dmg, enemy_hp, enemy_dmg)
    select f_id, ch_hp, ch_atk, en_hp, en_atk from selected;
END;
$$;

```

```

alter function insert_into_fight_recordings(integer, integer) owner to postgres;

```

2. Функция **“start_a_fight”** начинает битву, добавляет запись о ней в таблицу **“fights”** и первую запись о ходе битвы с начальными значениями здоровья врага и персонажа в таблицу **“fight_recordings.”**

```

create function start_a_fight(
    f_ch_id INTEGER,
    f_en_id INTEGER
)
returns integer
language plpgsql
as
$$
BEGIN
    INSERT INTO FIGHTS(character_id, enemy_id, fight_time)
    VALUES (f_ch_id, f_en_id, NOW());
    PERFORM insert_into_fight_recordings(f_ch_id, f_en_id);
    RETURN (SELECT id FROM FIGHTS ORDER BY ID DESC LIMIT 1);
END;
$$;

```

```

alter function start_a_fight(integer, integer) owner to postgres;

```

3. Функция **“get_en_hp”** нужна для получения информации о текущем значении здоровья врага в начале битвы.

```

create function get_en_hp(
f_en_id int
)
returns int
language plpgsql
as
$$
BEGIN
return (select hp from enemies where id = f_en_id);
END;
$$;

alter function get_en_hp(integer) owner to postgres;

```

4. Функция “**get_ch_hp**” нужна для получения информации о текущем значении здоровья персонажа в начале битвы.

```

create function get_ch_hp(
f_ch_id int
)
returns int
language plpgsql
as
$$
BEGIN
return (select hp from ch_characteristics where id = f_ch_id);
END;
$$;

alter function get_ch_hp(integer) owner to postgres;

```

5. Функция “en_attack” проводит атаку по персонажу со стороны врага. При этом снижается показатель здоровья персонажа. Атака умножается на функцию random(), дающую случайное значение между 0 и 1, чтобы уравновесить сражения между врагами и персонажами.

```
create function en_attack(  
    f_ch_id int,  
    f_en_id int,  
    f_ch_health int  
)  
returns int  
language plpgsql  
as  
$$  
BEGIN  
    f_ch_health := f_ch_health - (  
        select  
            floor(atk * random()) as en_atk  
        from enemies left join en_list on enemies.en_id = en_list.id  
where enemies.id = f_en_id  
    );  
  
    if (f_ch_health < 0) THEN  
        f_ch_health := 0;  
    end if;  
  
    UPDATE CH_CHARACTERISTICS  
    SET hp = f_ch_health  
    WHERE id = f_ch_id;  
    PERFORM insert_into_fight_recordings(f_ch_id, f_en_id);  
    return f_ch_health;  
END;  
$$;  
  
alter function en_attack(integer, integer, integer) owner to postgres;
```

6. **Функция “ch_attack”** проводит атаку по врагу со стороны персонажа. При этом снижается показатель здоровья врага. Атака умножается на функцию random(), дающую случайное значение между 0 и 1, умножается на 10, и после складывается со значением атаки, умноженным на 3.5, чтобы уравновесить сражения между врагами и персонажами. Так же персонажу доступна бонусная атака раз в 10 ходов.

```
create function ch_attack(  
    f_ch_id int,  
    f_en_id int,  
    f_en_health int  
)  
returns int  
language plpgsql  
as  
$$  
DECLARE  
f_energy INTEGER := (select energy from ch_characteristics where id = f_ch_id);  
BEGIN  
    if (select energy  
        from ch_characteristics  
        where id = f_ch_id) > 100  
    OR  
    (select energy  
        from ch_characteristics  
        where id = f_ch_id) = 100  
    THEN  
        f_en_health := f_en_health - (  
            select  
            floor(atk * random() * 10 + atk * 2.5) as ch_atk  
            from ch_characteristics where id = f_ch_id  
        );  
    update ch_characteristics  
    set energy = 0
```



```

where id = f_ch_id;

end if;

f_en_health := f_en_health - (
    select
        floor(atk * random() * 10 + atk * 2.5) as ch_atk
    from ch_characteristics where id = f_ch_id
);

if (f_en_health < 0) THEN
    f_en_health := 0;
end if;

f_energy := (select energy + 10 from ch_characteristics where id = f_ch_id);

UPDATE CH_CHARACTERISTICS
SET energy = f_energy
WHERE id = f_ch_id;

UPDATE ENEMIES
SET hp = f_en_health
WHERE
        id
        =
        f_en_id;
PERFORM insert_into_fight_recordings(f_ch_id, f_en_id);

return f_en_health;
END;
$$;

alter function ch_attack(integer, integer) owner to postgres;

```

7. **Функция “get_loot”** при победе персонажа над врагом добавляет предметы с него в инвентарь (inventory). При этом добавляется наугад от одного до трех предметов со врага.

```

create function get_loot(

```

```

    f_en_id int
)
returns void
language plpgsql
as
$$
BEGIN
insert into inventory (material_id)
select item_id from loot where enemy_id = f_en_id limit floor(random() * 3 + 1);
END;
$$;

alter function get_loot(integer) owner to postgres;

```

8. Функция “**simulate_a_fight**” запускает процесс сражения.

```

create function simulate_a_fight(
    f_ch_id INTEGER,
    f_en_id INTEGER
)
returns text
language plpgsql
as
$$
DECLARE
    f_en_health INTEGER := get_en_hp(f_en_id);
    f_ch_health INTEGER := get_ch_hp(f_ch_id);
    message text = "";

BEGIN

```

```

IF f_en_health <= 0 OR f_ch_health <= 0 THEN
message := 'One of the opponents can"t fight';
RETURN message;
END IF;

PERFORM start_a_fight(f_en_id, f_ch_id);

WHILE f_en_health != 0 AND f_ch_health != 0 LOOP
f_en_health := ch_attack(f_ch_id, f_en_id, f_en_health);
IF f_en_health > 0 THEN
f_ch_health := en_attack(f_ch_id, f_en_id, f_ch_health);
END IF;
END LOOP;

IF f_ch_health > 0 AND f_en_health <= 0 THEN
PERFORM get_loot(f_en_id);
PERFORM get_exp(f_ch_id, f_en_id);
DELETE FROM enemies where id = f_en_id;
message := 'Character won';
ELSEIF f_ch_health <= 0 AND f_en_health > 0 THEN
UPDATE CH_CHARACTERISTICS
SET status = 'defeated'
WHERE id = f_ch_id;
message := 'Enemie won';
ELSE
UPDATE CH_CHARACTERISTICS
SET status = 'defeated'
WHERE id = f_ch_id;
message := 'Both opponents were defeated';
END IF;
return message;
END;

```

```
$$;
```

```
alter function simulate_a_fight(integer, integer) owner to postgres;
```

Бизнес-процесс “Повышение уровня”

1. **Функция “get_max_hp”** нужна для получения информации о текущем значении максимального здоровья персонажа для его последующего увеличения.

```
create function get_max_hp(  
f_ch_id int  
)  
returns int  
language plpgsql  
as  
$$  
BEGIN  
return (select max_hp from ch_characteristics where id = f_ch_id);  
END;  
$$;
```

```
alter function get_max_hp(integer) owner to postgres;
```

2. **Функция “get_ch_atk”** нужна для получения информации о текущем значении атаки персонажа для его последующего увеличения.

```
create function get_ch_atk(  
f_ch_id int  
)  
returns int  
language plpgsql  
as  
$$  
BEGIN  
return (select atk from ch_characteristics where id = f_ch_id);  
END;
```

```
$$;
```

```
alter function get_ch_atk(integer) owner to postgres;
```

3. Процедура **“update_ch”** обновляет значения параметров “максимальное здоровье”, “здоровье”, “атака”, “цель”, “опыт” при достижении персонажем нового уровня. Параметры “максимальное здоровье” и “атака” увеличиваются соответственно в 1.3 и 1.25 раз, параметр “здоровье” становится равным обновленному параметру “максимальное здоровье”, опыт обнуляется, а “цель” становится равна переданному в процедуру новому значению цели.

```
create or replace procedure update_ch(
```

```
f_ch_id int,
```

```
new_target int
```

```
)
```

```
language plpgsql
```

```
as
```

```
$$
```

```
DECLARE
```

```
cur_max int := get_max_hp(f_ch_id);
```

```
cur_atk int := get_ch_atk(f_ch_id);
```

```
BEGIN
```

```
cur_max := floor(cur_max * 1.3);
```

```
cur_atk := floor(cur_atk * 1.25);
```

```
UPDATE CH_CHARACTERISTICS
```

```
SET max_hp = cur_max,
```

```
hp = cur_max,
```

```
target = new_target,
```

```
atk = cur_atk,
```

```
experience = 0;
```

```
END;
```

```
$$;
```

4. Функция **“level_up”** вызывается в ходе бизнес процесса при достижении целевого значения опыта. В функции обновляется текущее значение параметра “ранг” (уровень персонажа), и если оно не максимальное, происходит вызов процедуры update_ch, дающей персонажу новые характеристики в соответствии с новым рангом.

```

create or replace function level_up(
f_ch_id int
) returns void
language plpgsql
as
$$
DECLARE
cur_rank integer := (select ch_rank
from ch_characteristics
where id = f_ch_id);
BEGIN
IF cur_rank < 6 THEN
CALL delete_from_inventory(f_ch_id);
cur_rank := cur_rank + 1;
UPDATE CH_CHARACTERISTICS
SET ch_rank = cur_rank
WHERE id = f_ch_id;
END IF;
IF cur_rank = 2 THEN
CALL update_ch(f_ch_id, 854125);
ELSEIF cur_rank = 3 THEN
CALL update_ch(f_ch_id, 1195925);
ELSEIF cur_rank = 4 THEN
CALL update_ch(f_ch_id, 1611875);
ELSEIF cur_rank = 5 THEN
CALL update_ch(f_ch_id, 3423125);
ELSEIF cur_rank = 6 THEN
CALL update_ch(f_ch_id, 0);
END IF;
END;
$$;

alter function level_up(integer) owner to postgres;

```

5. Функция “**get_exp**” вызывается в ходе бизнес-процесса “сражение” в случае выигрыша персонажа в битве. В функции к текущему значению опыта персонажа добавляется количество опыта, получаемого со врага. В результате, если опыта стало больше, чем нужная цель, вызывается функция, повышающая уровень персонажа.

```
create or replace function get_exp(  
f_ch_id int,  
f_en_id int  
)  
returns void  
language plpgsql  
as  
$$  
DECLARE  
f_ch_exp int := (SELECT experience FROM CH_CHARACTERISTICS WHERE id = f_ch_id);  
f_target int := (SELECT target FROM CH_CHARACTERISTICS WHERE id = f_ch_id);  
f_en_exp int := (SELECT experience FROM EN_LIST WHERE id = f_en_id);  
is_found boolean := get_asc(f_ch_id);  
BEGIN  
f_ch_exp := f_ch_exp + f_en_exp;  
IF f_ch_exp >= f_target AND is_found THEN  
PERFORM level_up(f_ch_id);  
ELSE  
UPDATE CH_CHARACTERISTICS  
SET experience = f_ch_exp  
WHERE id = f_ch_id;  
END IF;  
END;  
$$;  
  
alter function get_exp(integer, integer) owner to postgres;
```

6. Функция “**get_asc**” нужна для поиска предметов, необходимых для увеличения уровня персонажа в инвентаре. Возвращает булевое значение.

```

create or replace function get_asc(
    f_ch_id integer
) returns boolean
language plpgsql
AS
$$
DECLARE
is_found boolean = FALSE;
BEGIN
IF(select count(*) from inventory where material_id in(select item_id from ascension
    where character_id = f_ch_id limit 1 offset 0)) > 0
AND
(select count(*) from inventory where material_id in(select item_id from ascension
    where character_id = f_ch_id limit 1 offset 1)) > 0
AND
(select count(*) from inventory where material_id in(select item_id from ascension
    where character_id = f_ch_id limit 1 offset 2)) > 0
THEN
is_found := TRUE;
END IF;
RETURN is_found;
END;
$$;

alter function get_asc(integer) owner to postgres;

```

7. Процедура “delete from inventory” удаляет из инвентаря игрока предметы, необходимые для повышения уровня персонажа во время исполнения бизнес-процесса “Повышение уровня”.

```

create or replace procedure delete_from_inventory(
    f_ch_id integer
)
AS

```



```

$$
BEGIN
with first_q as (
select id from inventory where material_id in(select item_id from ascension
      where character_id = f_ch_id limit 1 offset 0) limit 1
), second_q as (
select id from inventory where material_id in(select item_id from ascension
      where character_id = f_ch_id limit 1 offset 1) limit 1
), third_q as (
select id from inventory where material_id in(select item_id from ascension
      where character_id = f_ch_id limit 1 offset 2) limit 1
), first_c as (
select coalesce(first_q.id, second_q.id) as id from first_q full join second_q on first_q.id = second_q.id
), second_c as (
select coalesce(first_c.id, third_q.id) as id from first_c full join third_q on first_c.id = third_q.id
)
delete from inventory where id in(select id from second_c);
END;
$$ language plpgsql;

```

Индексы

Сценарии

Наиболее часто используемые сценарии при работе с таблицами:

1. CHARACTERS

- a. Вывод айди, имен и уровней всех персонажей с определенным элементом
Для облегчения вывода была создана функция `get_characters_by_element`
- b. Вывод айди, имен и уровней всех персонажей с определенным типом оружия
Для облегчения вывода была создана функция `get_characters_by_weapon`

2. CH_CHARACTERISTICS

- a. Получение характеристик персонажа по айди (здоровье, макс. здоровье, атака, опыт, участвует в бизнес-процессах “Сражение” и “Повышение уровня”)

3. ENEMIES

- a. Получение характеристик врага (здоровье и тип врага из таблицы `en_id`, входит в бизнес-процесс “Сражение”)

4. LOOT

- a. Просмотр предметов, которые падают со врагов, для дальнейшего добавления в инвентарь (входит в бизнес-процесс “Сражение”)

5. ASCENTION

- a. Получение списка материалов, необходимых конкретному персонажу для повышения уровня (входит в бизнес-процесс “Повышение уровня”)

6. INVENTORY

- a. Удаление из инвентаря материалов при повышении уровня персонажа (входит в бизнес-процесс “Повышение уровня”)
- b. Добавление в инвентарь материалов при победе над врагом (входит в бизнес-процесс “Сражение”)

7. FIGHTS

- a. Начало нового сражения, проведение сражения (входит в бизнес-процесс “Сражение”)

8. FIGHT_RECORDINGS

- a. Добавление записей о ходе сражения (входит в бизнес процесс “Сражение”)

EXPLAIN ANALYZE

Ситуация до добавления индексов

EXPLAIN и EXPLAIN ANALYZE похожие команды, однако EXPLAIN только составляет план запроса, а EXPLAIN ANALYZE исполняет функцию, к которой просится провести анализ, и выводит время ее выполнения. Для того, чтобы после добавления индексов провести абсолютно идентичный анализ на таблице в том же состоянии, как до проведения анализа, мы будем пользоваться операцией ROLLBACK (откат).

Пример транзакции для проведения анализа:

```
BEGIN;
```

```
EXPLAIN ANALYZE
```

```
SELECT simulate_a_fight(1, 14);
```

```
ROLLBACK;
```

Проверим производительность на самой массивной функции в моей базе данных. Это функция `simulate_a_fight`. Так как в функции в битве между двумя врагами повсеместно используется рандом, и результат битвы не определен точно, то попробуем посчитать среднее между всеми полученными результатами: 7.365, 8.911, 3.063, 4.994, 5.942. Среднее время выполнения команды - 6.055 мс.

Data Output		Messages
	QUERY PLAN	
	text	
1	Result (cost=0.00..0.26 rows=1 width=32) (actual time=5.924..5.925 rows=1 loops...	
2	Planning Time: 0.027 ms	
3	Execution Time: 5.942 ms	

Посчитаем производительность на функции `get_characters_by_element`, где нет случайных значений, а значит, анализ будет точнее. До добавления индексов функция выполнялась примерно за 0.307 мс.

	QUERY PLAN	
	text	
1	ProjectSet (cost=0.00..5.27 rows=1000 width=32) (actual time=0.275..0.279 rows=8 loops...	
2	-> Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1)	
3	Planning Time: 0.034 ms	
4	Execution Time: 0.307 ms	

Посчитаем производительность на функции `get_ch_hp`, которая часто используется в бизнес-процессах. До добавления индексов функция выполнялась в среднем за 0.124 мс.

Data Output		Messages
	QUERY PLAN	
	text	
1	Result (cost=0.00..0.26 rows=1 width=4) (actual time=0.103..0.103 rows=1 loops...	
2	Planning Time: 0.034 ms	
3	Execution Time: 0.124 ms	

В ходе лабораторной работы было выявлено, что наиболее часто в базе данных обращаются ко следующим параметрам:

1. `id` из таблицы `CH_CHARACTERISTICS` (в ходе бизнес процессов “Сражение” и “Повышение уровня”)
2. `hp` из таблицы `CH_CHARACTERISTICS` (в ходе бизнес процессов “Сражение” и “Повышение уровня”)
3. `id` из таблицы `ENEMIES` (в ходе бизнес-процесса “Сражение”)
4. `hp` из таблицы `ENEMIES` (в ходе бизнес процессов “Сражение” и “Повышение уровня”)
5. `fight_id` из таблицы `FIGHTS`(в ходе бизнес процессов “Сражение” и “Повышение уровня”)

Для ускорения работы бизнес-процессов и поиска персонажей по имени были созданы следующие индексы:

```
CREATE INDEX ch_id_idx ON CH_CHARACTERISTICS(id);
CREATE INDEX en_id_idx ON ENEMIES(id);
CREATE INDEX ch_hp_idx ON CH_CHARACTERISTICS(hp);
CREATE INDEX en_hp_idx ON ENEMIES(hp);
CREATE INDEX ch_atk_idx ON CH_CHARACTERISTICS(atk);
CREATE INDEX en_atk_idx ON EN_LIST(atk);
CREATE INDEX f_id_idx ON FIGHTS(id);
CREATE INDEX ch_n_idx ON CHARACTERS(name);
CREATE INDEX ch_el_idx ON CHARACTERS(ch_element);
CREATE INDEX ch_w_idx ON CHARACTERS(weapon_type);
```

По умолчанию создается индекс вида B-Tree (Balanced Tree).

EXPLAIN ANALYZE

Ситуация после добавления индексов

Посчитаем среднее время исполнения команды `simulate_a_fight` после добавления индексов: 6.939, 4.880, 3.676, 3.833, 3.621. Среднее время выполнения - 4.5898 мс. Это на 24% быстрее, чем без индексов.

	QUERY PLAN
	text
1	Result (cost=0.00..0.26 rows=1 width=32) (actual time=3.812..3.813 rows=1 loops=...
2	Planning Time: 0.034 ms
3	Execution Time: 3.833 ms

Для исследования работы индексов заменим В-Tree индексы для атрибутов “айди” таблиц “Характеристики персонажей” и “Враги” на Хэш-индексы. Для этого выполним следующие операции:

```
DROP INDEX ch_id_idx;
```

```
DROP INDEX en_id_idx;
```


```
CREATE INDEX ch_id_idx ON CH_CHARACTERISTICS USING HASH(id);
```

```
CREATE INDEX en_id_idx ON ENEMIES USING HASH(id);
```


В плане построения запроса мы можем увидеть, как производится обращение к созданному нами индексу (HASH COND, 2 строка). План построения запроса:

	QUERY PLAN
	text
1	Hash Right Join (cost=3.61..5.19 rows=1 width=16) (actual time=0.052..0.063 rows=1 loops=1)
2	Hash Cond: (en_list.id = enemies.en_id)
3	-> Seq Scan on en_list (cost=0.00..1.45 rows=45 width=8) (actual time=0.007..0.010 rows=45 loops=1)
4	-> Hash (cost=3.60..3.60 rows=1 width=16) (actual time=0.035..0.035 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on enemies (cost=0.00..3.60 rows=1 width=16) (actual time=0.023..0.026 rows=1 loops=1)
7	Filter: (id = 14)
8	Rows Removed by Filter: 127
9	Planning Time: 0.184 ms
10	Execution Time: 0.085 ms


После замены индекса по умолчанию на Хэш-индекс для атрибутов “айди” таблиц “Характеристики персонажей” и “Враги” среднее время выполнения функции стабильно стало попадать в следующий диапазон: [4.000, 4.200]. Это значительно меньше, чем с В-Tree индексом. Это доказывает, что в операциях прямого сравнения “=”, часто появляющихся в запросах, эффективнее себя показывают Хэш-индексы.

	QUERY PLAN	
	text	
1	Result (cost=0.00..0.26 rows=1 width=32) (actual time=4.063..4.063 rows=1 loops...	
2	Planning Time: 0.028 ms	
3	Execution Time: 4.081 ms	

После добавления индексов на имя и элемент персонажей функция стала выполняться в среднем за 0.17 мс, что быстрее почти на 50%.

	QUERY PLAN	
	text	
1	ProjectSet (cost=0.00..5.27 rows=1000 width=32) (actual time=0.112..0.122 rows=8 loops...	
2	-> Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.002..0.002 rows=1 loops=1)	
3	Planning Time: 0.049 ms	
4	Execution Time: 0.159 ms	

После добавления индексов функция стала исполняться за примерно 0.107 мс, что на 20% быстрее.

	QUERY PLAN	
	text	
1	Result (cost=0.00..0.26 rows=1 width=4) (actual time=0.088..0.088 rows=1 loops...	
2	Planning Time: 0.031 ms	
3	Execution Time: 0.107 ms	

В целом, основываясь на времени исполнения функций до и после создания индексов, можно сказать, что добавление индексов повышает скорость исполнения команд примерно на 25%.

Выводы

По результатам лабораторной работы я изучила триггеры, функции и процедуры PostgreSQL, научилась работать с ними и реализовала бизнес-процессы по базе данных по компьютерной игре Genshin Impact, работу с которой я вела на протяжении всего семестра. Я научилась анализировать эффективность исполнения команд в базе данных при помощи EXPLAIN и EXPLAIN ANALYZE, и для повышения производительности обращений к таблицам научилась создавать индексы.

В целом все пять лабораторных работ, сделанных мною по предмету “Базы данных”, сильно расширили мои знания по работе с таблицами SQL. Я провела большую работу и узнала много нового.