Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 3

з дисципліни «Бази даних і засоби управління»
«Засоби оптимізації роботи СУБД PostgreSQL»

Виконав:
студент групи КП-81
Бухаленков Дмитро Олександрович
Перевірив: Радченко К.О.

Київ 2020

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

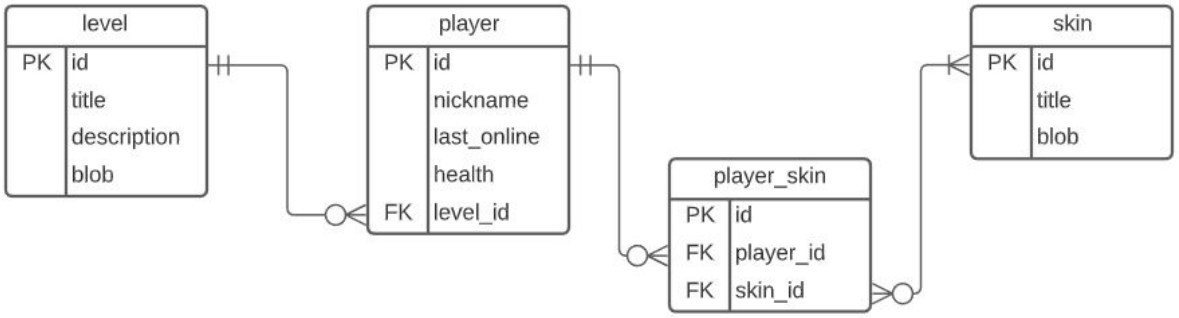
Загальне завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи No2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

Варіант 2:

2	*Hash, BRIN*	*after insert, update*

Графічна ER модель



Сутності БД

Сутність	Атрибут	Тип (Розмір)
Level (інформація про рівень)гри	id – унікальний ID рівня в БД title – назва рівня description – опис рівня blob – містить строкове посилання на файл бінарного набору даних, що містить інформацію про текстури, скрипти рівня	Числовий Текстовий(20) Текстовий(200) Текстовий(50)
Player (інформація про гравця)	id – унікальний ID гравця в БД nickname – ім'я гравця last_online – коли гравець в останнє заходив в гру health – кількість очок здоров'я гравця level_id – ID рівня, на якому перебуває гравець	Числовий Текстовий(20) Дата Числовий Числовий
Skin (інформація про скін)	id – унікальний ID скіну в БД title – назва скіну blob – посилання на файл даних з текстурами, скриптами і тд.	Числовий Текстовий(20) Текстовий(50)
Player_Skin (інформація про володіння скінами)	id – унікальний ID зв'язку «гравець-скін» player_id – ID гравця skin_id – ID скіна	Числовий Числовий Числовий

Меню додатку

```
Menu:


  1 - level
  2 - player
  3 - skin
  4 - player_skin
  5 - Fill table "level" by random data
  6 - Commit
  7 - Exit
```

```
GET skin
Enter search criteria:
id , title , blob


Table :  `skin`

<Skin>{'id': 1, 'blob': './blob/dragonlore', 'title': 'Dragon Lore'})
<Skin>{'id': 2, 'blob': './blob/goldenabyss', 'title': 'Golden Abyss'})
<Skin>{'id': 3, 'blob': './blob/reddawn', 'title': 'Red Dawn'})
```

Класси

```
class Level(Base, Repr):
    __tablename__ = 'level'

    id = Column(Integer, primary_key=True)
    title = Column(String)
    description = Column(String)
    blob = Column(String)

    players = relationship('Player')

    def __init__(self, title=None, description=None,
blob=None):
        self.title = title
        self.description = description
        self.blob = blob
```

```
class Player(Base, Repr):
    __tablename__ = 'player'

    id = Column(Integer, primary_key=True)
    nickname = Column(String)
    last_online = Column(Date)
    health = Column(Integer)
    level_id = Column(Integer,
ForeignKey('level.id'))

    player_skins = relationship("PlayerSkin")

    def __init__(self, nickname=None,
last_online=None, health=None, level_id=None):
        self.nickname = nickname
        self.last_online = last_online
        self.health = health
        self.level_id = level_id
```

```
class Skin(Base, Repr):
    __tablename__ = 'skin'

    id = Column(Integer, primary_key=True)
    title = Column(String)
    blob = Column(String)

    player_skins = relationship("PlayerSkin")

    def __init__(self, title=None, blob=None):
        self.title = title
        self.blob = blob
```

```
class PlayerSkin(Base, Repr):
    __tablename__ = 'player_skin'

    id = Column(Integer, primary_key=True)
    player_id = Column(Integer,
ForeignKey('player.id'))
    skin_id = Column(Integer, ForeignKey('skin.id'))

    def __init__(self, player_id=None,
skin_id=None):
        self.player_id = player_id
        self.skin_id = skin_id
```

Запити ORM

```
def get(self, table_name, condition):

    object_class = TABLES[table_name]
    objects = session.query(object_class)

    if condition:
        try:
            pairs = self.pairs_from_str(condition)
        except Exception as err:
            raise Exception('Incorrect input')
        objects = self.filter_by_pairs(objects,
pairs, object_class)

    return list(objects)
```

```
def insert(self, table_name, columns, values):
    columns = [c.strip() for c in
columns.split(',')]
    values = [v.strip() for v in values.split(',')]

    pairs = dict(zip(columns, values))
    object_class = TABLES[table_name]
    obj = object_class(**pairs)

    session.add(obj)
```

```
def delete(self, table_name, condition):
    try:
        pairs = self.pairs_from_str(condition)
    except Exception as err:
        raise Exception('Incorrect input')
    object_class = TABLES[table_name]

    objects = session.query(object_class)
    objects = self.filter_by_pairs(objects, pairs,
object_class)

    objects.delete()
```

```
def update(self, table_name, condition, statement):
    try:
        pairs = self.pairs_from_str(condition)
        new_values = self.pairs_from_str(statement)
    except Exception as err:
        raise Exception('Incorrect input')

    object_class = TABLES[table_name]

    objects = session.query(object_class)
    objects = self.filter_by_pairs(objects, pairs,
object_class)

    for obj in objects:
        for field_name, value in new_values.items():
            setattr(obj, field_name, value)
```

Індекси

Пошук з фільтрацією (без індексів)

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using level_pkey on level (cost=0.29..1908.09 rows=50217 width=47) (actual time=0.019..9.504 rows=49992 loops=1)
2	Index Cond: (id < 50000)
3	Planning Time: 0.086 ms
4	Execution Time: 11.305 ms

Запит з фільтрацією та сортуванням

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000 order by id
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using level_pkey on level (cost=0.29..1908.09 rows=50217 width=47) (actual time=0.032..8.559 rows=49992 loops=1)
2	Index Cond: (id < 50000)
3	Planning Time: 0.134 ms
4	Execution Time: 10.135 ms

Створення Hash індексу

```
CREATE INDEX hash1 ON level using hash(id)
```

Запит з фільтрацією (hash)

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using level_pkey on level (cost=0.29..1908.09 rows=50217 width=47) (actual time=0.035..16.269 rows=49992 lo...
2	Index Cond: (id < 50000)
3	Planning Time: 0.145 ms
4	Execution Time: 19.188 ms

Запит з фільтрацією та сортуванням

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000 order by id
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using level_pkey on level (cost=0.29..1908.09 rows=50217 width=47) (actual time=0.031..14.844 rows=49992 lo...
2	Index Cond: (id < 50000)
3	Planning Time: 0.152 ms
4	Execution Time: 17.509 ms

Створення BRIN індексу

```
CREATE INDEX brin1 ON level using brin(id)
```

Запит з фільтрацією (brin)

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on level (cost=24.68..1509.67 rows=50217 width=47) (actual time=0.200..6.054 rows=49992 loops=1)
2	Recheck Cond: (id < 50000)
3	Rows Removed by Index Recheck: 4784
4	Heap Blocks: lossy=512
5	-> Bitmap Index Scan on brin1 (cost=0.00..12.13 rows=51439 width=0) (actual time=0.185..0.185 rows=5120 loops=1)
6	Index Cond: (id < 50000)
7	Planning Time: 0.121 ms
8	Execution Time: 7.311 ms

Запит з фільтрацією та сортуванням

```
14  EXPLAIN ANALYZE SELECT * FROM level WHERE id<50000 order by id
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using level_pkey on level (cost=0.29..1908.09 rows=50217 width=47) (actual time=0.023..9.345 rows=49992 loo...
2	Index Cond: (id < 50000)
3	Planning Time: 0.141 ms
4	Execution Time: 10.911 ms

Висновки: BRIN покращує швидкодію порівняно з запитами без індексів, а Hash оперує навіть гірше ніж без індексів. BRIN індекси добре

використовувати коли в нас велика кількість даних, отже в нашому випадку з 90 000 записами він виграє у швидкодії.

- B-Tree - For most datatypes and queries
- GIN - For JSONB/hstore/arrays
- GiST - For full text search and geospatial datatypes
- SP-GiST - For larger datasets with natural but uneven clustering
- BRIN - For really large datasets that line up sequentially
- Hash - For equality operations, and generally B-Tree still what you want here

Тригери

Для тригера, що спрацює після вставки, створимо функцію, що перевірить чи є вже гравець з таким нікнеймом, якщо ж є, то вивести попередження та дописати до нікнейму в кінець унікальній id гравця, що зробить нікнейм унікальним. Також функція перевіряє здоров'я гравця на мінімальне значення (100) і якщо здоров'я нижче, то викидає про це помилку.

after_insert()

General Definition Code Options Parameters Security SQL

```
1  DECLARE
2      nick text;
3      nick_id int;
4  BEGIN
5      FOR nick, nick_id IN
6          SELECT nickname, id from player
7      LOOP
8          IF NEW.nickname = nick AND NEW.id != nick_id THEN
9              RAISE INFO 'Username already exists';
10             UPDATE player SET nickname=NEW.nickname || NEW.id WHERE id= NEW.id;
11             EXIT;
12         END IF;
13     END LOOP;
14     IF NEW.health <100 THEN
15         RAISE EXCEPTION 'Health can`t be lower than 100';
16     END IF;
17     RETURN NEW;
18 END;
19
```

Приклади запитів:

Додаємо нового гравця з унікальним нікнеймом, але здоров’ям нижче за 100, отримуємо відповідну помилку.

```
12  INSERT INTO public.player(
13       nickname, last_online, health, level_id)
14      VALUES ( 'Failure Player', '12.11.2020', 70, 7);
```

Data Output Explain Messages Notifications

```
ERROR: ОШИБКА:  Health can`t be lower than 100
CONTEXT:  функция PL/pgSQL after_insert(), строка 16, оператор RAISE


SQL state: P0001
```

Якщо нікнейм не унікальний, і неправильна кількість очок здоров’я, то бачимо попередження і помилку одночасно.

```
12  INSERT INTO public.player(
13       nickname, last_online, health, level_id)
14      VALUES ( 'NewGuy', '12.11.2020', 70, 7);
```

Data Output Explain Messages Notifications

```
ИНФОРМАЦИЯ:  Username already exists

ERROR: ОШИБКА:  Health can`t be lower than 100
CONTEXT:  функция PL/pgSQL after_insert(), строка 16, оператор RAISE


SQL state: P0001
```

Тепер додамо гравця з існуючим нікнеймом але з задовільним здоров’ям, отримаємо попередження, користувача додано зі зміненим тепер вже унікальним нікнеймом.

```sql
INSERT INTO public.player(
     nickname, last_online, health, level_id)
    VALUES ( 'NewGuy', '12.11.2020', 101, 7);
```

Data Output | Explain | Messages | Notifications

```
ИНФОРМАЦИЯ:  Username already exists
INSERT 0 1

Query returned successfully in 348 msec.
```

Як бачимо в таблиці з’явився NewGuy24 з унікальним нікнеймом, в кінці якого id гравця.

8	12	test_player	2018-08-08	110	5
9	13	NewGuy	2019-12-12	100	6
10	24	NewGuy24	2020-11-12	101	7

Для тригера після оновлення зробимо схожу логіку: він перевіряє унікальність нового нікнейму і здоров’я, якщо значення незадовільне то викидає помилку.

after_update()

General | Definition | Code | Options | Parameters | Security | SQL

```sql
DECLARE
    nick text;
    nick_id int;
BEGIN
    FOR nick, nick_id IN
        SELECT nickname, id from player
    LOOP
        IF NEW.nickname = nick AND NEW.id != nick_id THEN
            RAISE EXCEPTION 'Username already exists';
            EXIT;
        END IF;
    END LOOP;
    IF NEW.health <100 THEN
        RAISE EXCEPTION 'Health can`t be lower than 100';
    END IF;
    RETURN NEW;
END;

```

Як бачимо, неправильні запити не проходять – викидається помилка

```
12  UPDATE public.player
13      SET  nickname='NewGuy', health=-7
14      WHERE id=24;
```

Data Output Explain Messages Notifications

```
ERROR: ОШИБКА:  Username already exists
CONTEXT:  функция PL/pgSQL after_update(), строка 10, оператор RAISE


SQL state: P0001
```

```
12  UPDATE public.player
13      SET  nickname='RadomName', health=-7
14      WHERE id=24;
```

Data Output Explain Messages Notifications

```
ERROR: ОШИБКА:  Health can`t be lower than 100
CONTEXT:  функция PL/pgSQL after_update(), строка 15, оператор RAISE


SQL state: P0001
```

Коректні запити проходять і дані про гравця оновлюються.

```
12  UPDATE public.player
13      SET  nickname='RadomName', health=102
14      WHERE id=24;
```

Data Output Explain Messages Notifications

```
UPDATE 1

Query returned successfully in 94 msec.
```

9	13	NewGuy	2019-12-12	100	6
10	24	RadomName	2020-11-12	102	7

Висновки: в ході виконання лабораторної роботи було опановано способи оптимізації додатку для роботи з базою даних за допомогою ORM, досліджено вплив використання індексів, а також створено і протестовано тригери.