

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО"

Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №3

із дисципліни «Бази даних»

на тему «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав:

студент 3 курсу ФПМ групи КП-82

Новохацький Владислав Андрійович

Прийняв: Радченко К.О.

	Бали
Якість виконання	
Термін здачі	
Сумарний бал	

Мета роботи

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

Постановка завдання

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

Варіант 15

<i>15</i>	<i>Hash, BRIN</i>	<i>before delete, update</i>
-----------	-------------------	------------------------------

Опис програми

Програма допомагає взаємодіяти з базою даних завдяки консольному інтерфейсу, є можливість додавати, змінювати видаляти елементи бази даних.

Видалення елементів відбувається каскадно, тобто видаляються всі елементи, що втрачають своє значення в базі даних (реалізовано програмно)

Програма побудована на згідно патерну MVC, весь код розділений на три складові: model, view, controller, код розподілений по окремим файлам.

INSERT, UPDATE, DELETE реалізовано за допомогою orm від sqlalchemy

При випадковому генеруванні даних також генеруються випадки між різними таблицями.

Створені тригери для подій UPDATE та BEFORE DELETE.

Результати виконання роботи

Завдання 1

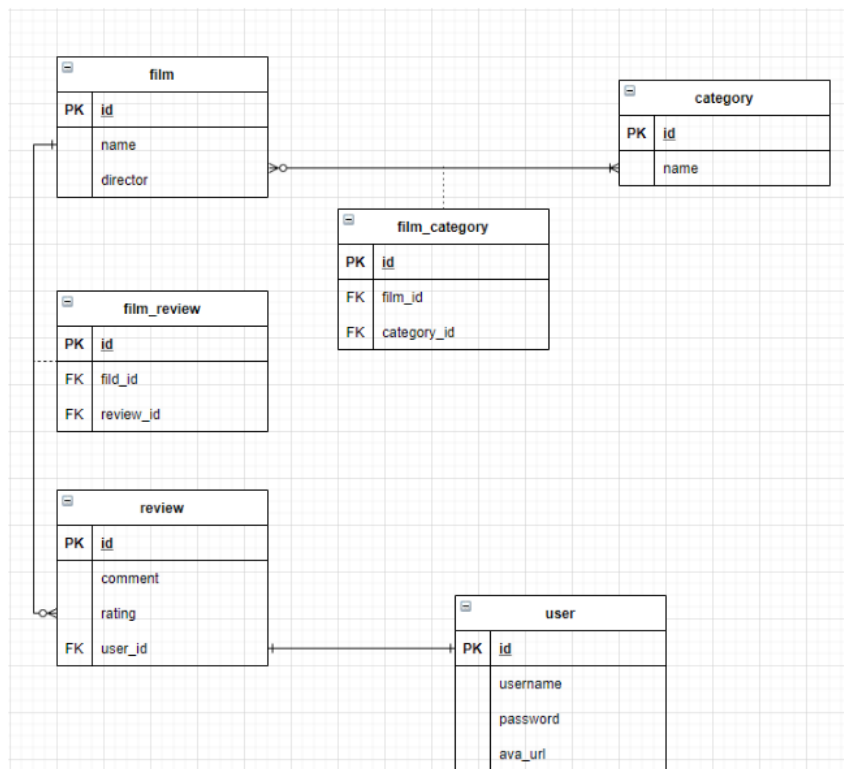
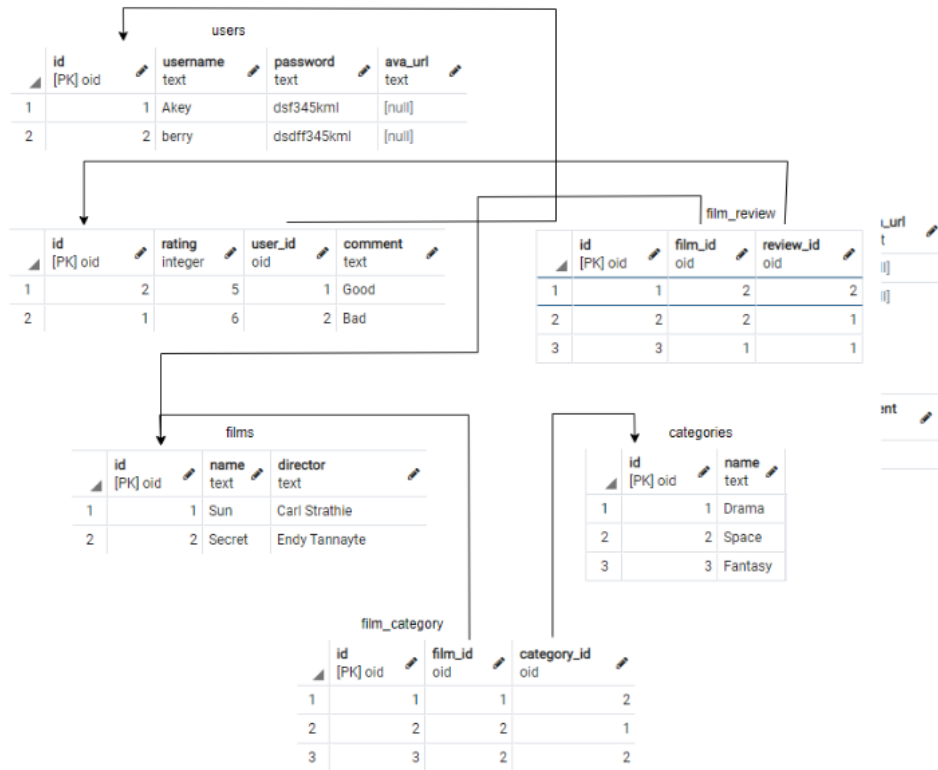


Рис.1. Схема бази даних у вигляді таблиць і зв'язки між ними.

```

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    username = Column(String)
    password = Column(String)
    ava_url = Column(String)

    def __init__(self, username, password, ava_url):
        self.username = username
        self.password = password
        self.ava_url = ava_url

class Category(Base):
    __tablename__ = "categories"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    films = relationship("Film", secondary=films_categories_association, back_populates="categories")

    def __init__(self, name):
        self.name = name

```

Рис.2. Класи User та Category

```

class Film(Base):
    __tablename__ = "films"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    director = Column(String)
    categories = relationship("Category", secondary=films_categories_association, back_populates="films")
    reviews = relationship("Review", secondary=films_review_association)

    def __init__(self, name, director):
        self.name = name
        self.director = director

class Review(Base):
    __tablename__ = "reviews"
    comment = Column(String)
    id = Column(Integer, primary_key=True)
    rating = Column(Integer)
    user_id = Column(Integer, ForeignKey('users.id'))
    user = relationship("User")

    def __init__(self, comment, rating, user_id):
        self.comment = comment
        self.rating = rating
        self.user_id = user_id

```

Рис.3. Класи Film та Review

```

INFO:sqlalchemy.engine.base.Engine:BEGIN (implicit)
INFO:sqlalchemy.engine.base.Engine:INSERT INTO users (username, password, ava_url) VALUES (%(username)s, %(password)s, %(ava_url)s) RETURNING users.id
INFO:sqlalchemy.engine.base.Engine:{'username': 'vlad', 'password': '1234', 'ava_url': '1234'}
INFO:sqlalchemy.engine.base.Engine:COMMIT

INFO:sqlalchemy.engine.base.Engine:BEGIN (implicit)
INFO:sqlalchemy.engine.base.Engine:SELECT users.id AS users_id, users.username AS users_username, users.password AS users_password, users.ava_url AS users_ava_url
FROM users
WHERE users.id = %(param_1)s
INFO:sqlalchemy.engine.base.Engine:{'param_1': 55}
INFO:sqlalchemy.engine.base.Engine:UPDATE users SET username=%(username)s, password=%(password)s, ava_url=%(ava_url)s WHERE users.id = %(users_id)s
INFO:sqlalchemy.engine.base.Engine:{'username': 'ua', 'password': '45', 'ava_url': '45', 'users_id': 55}
INFO:sqlalchemy.engine.base.Engine:COMMIT

```

Рис.4. Результати logger'у (можемо прослідити query)

Завдання 2

```
CREATE INDEX hash ON public.films USING hash(id)
```

Рис. 5. Створення HASH індексу

```
1 EXPLAIN SELECT * FROM public.films
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 Seq Scan on films (cost=0.00..36.40 rows=2340 width=12)

```
1 EXPLAIN SELECT MIN(name) FROM public.films
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 Aggregate (cost=42.25..42.26 rows=1 width=32)

2 -> Seq Scan on films (cost=0.00..36.40 rows=2340 width=4)

```
1 EXPLAIN SELECT * FROM public.films ORDER BY name
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 Sort (cost=167.35..173.20 rows=2340 width=12)

2 Sort Key: name

3 -> Seq Scan on films (cost=0.00..36.40 rows=2340 width=12)

```
1 EXPLAIN SELECT * FROM public.films WHERE id>1000 ORDER BY name
```

Data Output Explain Messages Notifications

QUERY PLAN
text

1 Sort (cost=105.90..109.00 rows=1239 width=12)

2 Sort Key: name

3 -> Seq Scan on films (cost=0.00..42.25 rows=1239 width=12)

4 Filter: (id > 1000)

Рис. 6. Результати отримання даних з таблиці з HASH індексуванням

```
CREATE INDEX BRIN ON public.films USING BRIN(id)
```

Рис. 7. Створення BRIN індексу

1

EXPLAIN SELECT * FROM public.films

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Seq Scan on films (cost=0.00..35.80 rows=2280 width=12)

1

EXPLAIN SELECT MIN(name) FROM public.films

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Aggregate (cost=41.50..41.51 rows=1 width=32)

2

-> Seq Scan on films (cost=0.00..35.80 rows=2280 width=4)

1

EXPLAIN SELECT * FROM public.films ORDER BY name

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Sort (cost=162.96..168.66 rows=2280 width=12)

2

Sort Key: name

3

-> Seq Scan on films (cost=0.00..35.80 rows=2280 width=12)

1

EXPLAIN SELECT * FROM public.films WHERE id>1000 ORDER BY name

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Sort (cost=103.28..106.30 rows=1207 width=12)

2

Sort Key: name

3

-> Seq Scan on films (cost=0.00..41.50 rows=1207 width=12)

4

Filter: (id > 1000)

Рис. 8. Результати отримання даних з таблиці з BRIN індексуванням

Завдання 3

BEFORE DELETE

```
CREATE OR REPLACE function func1()
    RETURNS trigger AS
$$
DECLARE
    cur_film CURSOR
    IS
        SELECT
            *
        FROM
            public.films
            WHERE id<100;
BEGIN
    FOR film IN cur_film
    LOOP
        IF film.id>50 THEN
            UPDATE films SET name=OLD.name WHERE id=film.id;
        ELSE
            UPDATE films SET name=OLD.id WHERE id=film.id;
        END IF;
    END LOOP;
    return OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER bfr_delete
    BEFORE DELETE
    ON public.films FOR EACH ROW
EXECUTE PROCEDURE func1()
```

Рис.9. Текст тригера Before Delete

Не має семантичного сенсу. До видалення елементу змінює name всіх елементів з id<100 на name елементу, що видаляємо при id елементу більше 50, та на id елементу при за інших обставин.



74	XWQ	YJR
73	XWQ	PVB
72	XWQ	BQC
71	XWQ	GBP
70	XWQ	YFS
9	789	SQ
8	789	BT
7	789	Iam
1	789	Carl Strathie

Рис. 10. Результати роботи тригера при видаленні елементу з ID 789 та name YWQ

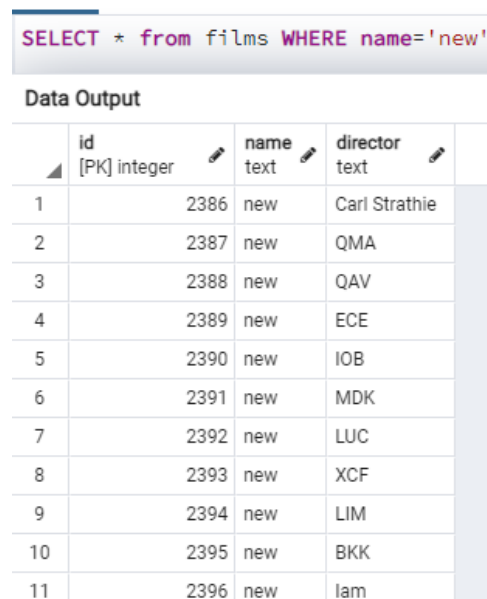
UPDATE

```
CREATE OR REPLACE function func2()
    RETURNS trigger AS
$$
DECLARE
    cur_film2 CURSOR
    IS
        SELECT
            *
        FROM
            public.films
            WHERE id<100;
BEGIN
    FOR film IN cur_film
    LOOP
        INSERT INTO films (name, director) VALUES (OLD.name, film.director);
    END LOOP;
    return OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER updateTrigger
    AFTER UPDATE
    ON public.films FOR EACH ROW
EXECUTE PROCEDURE func2();
```

Рис.11. Текст тригера Update

Не має семантичного сенсу. При модифікуванні елементу додає (COUNT(id<100)) елементів зі старим name модифікованого елементу та director зі списку (id<100).



The screenshot shows a SQL query: `SELECT * from films WHERE name='new'`. Below the query is a table titled "Data Output" with 4 columns: `id` (integer, PK), `name` (text), and `director` (text). The table contains 11 rows of data, all with the name 'new'.

	id [PK] integer	name text	director text
1	2386	new	Carl Strathie
2	2387	new	QMA
3	2388	new	QAV
4	2389	new	ECE
5	2390	new	IOB
6	2391	new	MDK
7	2392	new	LUC
8	2393	new	XCF
9	2394	new	LIM
10	2395	new	BKK
11	2396	new	Iam

Рис.12. Результати роботи тригера при модифікуванні елементу зі старим name *new*