



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих
комп'ютерних систем**

Лабораторна робота №2

з дисципліни **Бази даних і засоби управління**
*на тему: "Проектування бази даних та ознайомлення з базовими
операціями СУБД PostgreSQL"*

Виконав:

студент III курсу
групи KB-13 Шандиба А. А.
Telegram: <https://t.me/andriic0>
Github: <https://github.com/andreas778/bd>

Київ – 2023

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

Завдання роботи полягає у наступному:

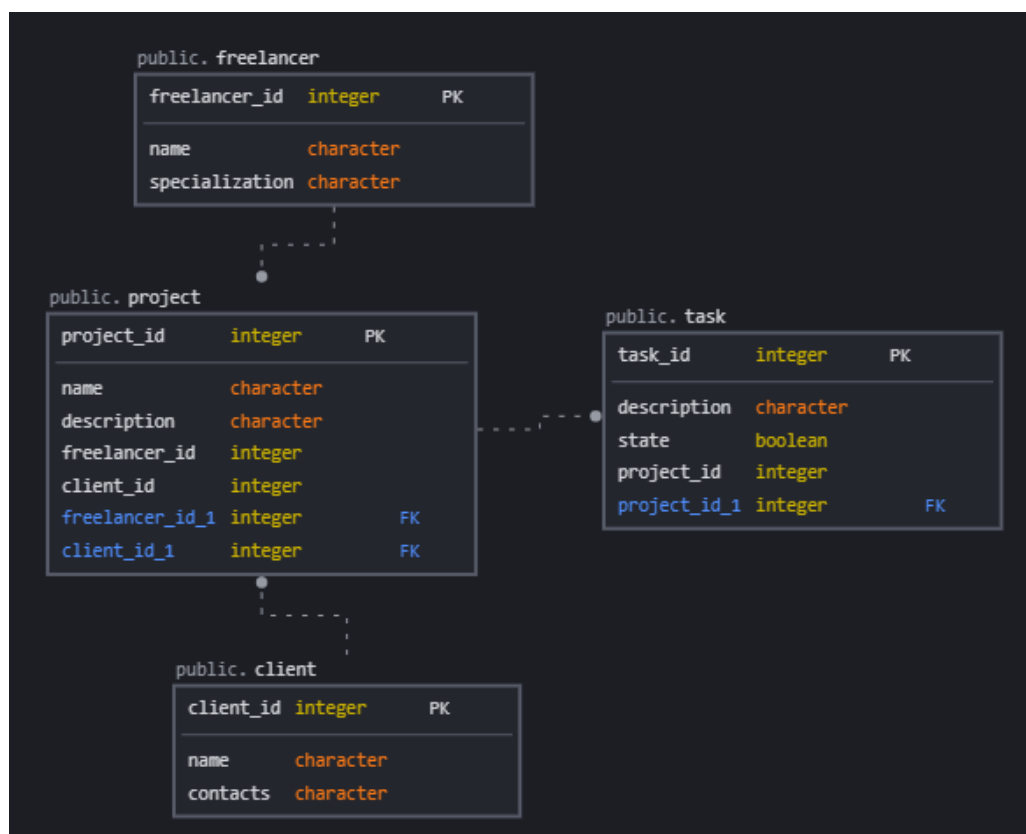
1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проєкції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Варіант 22

22	Hash, BRIN	after delete, insert
----	------------	----------------------

Завдання 1

Логічна схема бази даних «Система управління завданнями та проектами для фрілансерів»



Моделі ORM

```
class Freelancer(DeclarativeBase):
```

```
    __tablename__ = 'freelancer'
```

```
    freelancer_id = Column('freelancer_id', Integer, primary_key=True)
```

```
    name = Column('name', String(30))
```

```
    specialization = Column('specialization', String(30))
```

```
class Client(DeclarativeBase):
```

```
    __tablename__ = 'client'
```

```
    client_id = Column('client_id', Integer, primary_key=True)
```

```
    name = Column('name', String(30))
```

```
    contacts = Column('contacts', String(30))
```

```
class Project(DeclarativeBase):
```

```
    __tablename__ = "project"
```

```
    project_id = Column('project_id', Integer, primary_key=True)
```

```
    name = Column('name', String(30))
```

```
    description = Column('description', String)
```

```
    freelancer_id = Column('freelancer_id', Integer,  
ForeignKey("freelancer.freelancer_id"))
```

```
    client_id = Column('client_id', Integer, ForeignKey("client.client_id"))
```

```
class Task(DeclarativeBase):  
    __tablename__ = "task"  
    task_id = Column('task_id', Integer, primary_key=True)  
    description = Column('description', String)  
    state = Column('state', Boolean)  
    project_id = Column('project_id', Integer, ForeignKey("project.project_id"))
```

Приклади запитів у вигляді ORM

Код запиту на видалення даних:

```
def delete_request(self, table, where):  
    temp = Table(table, self.metadata, autoload=True, autoload_with=self.engine)  
    query = delete(temp).where(text(str(where)))  
    results = self.session.execute(query)  
    results = self.session.execute(select([temp])).fetchall()  
    output('output.txt', results)  
    self.session.commit()
```

Код запиту на оновлення даних:

```
def update_request(self, table, condition):  
    temp = Table(table, self.metadata, autoload=True, autoload_with=self.engine)  
    where, what = condition.split(',')  
    res = eval('dict(' + what + ')')  
    query = update(temp).values(res).where(text(where))  
    results = self.session.execute(query)  
    results = self.session.execute(select([temp])).fetchall()  
    output('output.txt', results)  
    self.session.commit()
```

Код запиту на вставку даних:

```
def insert_request(self, table, condition):
```

```
    temp = Table(table, self.metadata, autoload=True, autoload_with=self.engine)
```

```
    res = eval('dict(' + condition + ')')
```

```
    query = insert(temp)
```

```
    ResultProxy = self.session.execute(query, res)
```

```
    results = self.session.execute(select([temp])).fetchall()
```

```
    output('output.txt', results)
```

```
    self.session.commit()
```

Завдання 2

Для дослідження індексу була створена таблиця test_table, яка має дві колонки типу integer та character varying. У таблицю було занесено 100000 записів.

```
1 CREATE TABLE test_table (  
2     test_value NUMERIC,  
3     test_string VARCHAR  
4 );  
5 INSERT INTO test_table (test_value, test_string)  
6 SELECT random() * 1000,  
7     chr(trunc(65 + random() * 50)::int) ||  
8     chr(trunc(65 + random() * 25)::int) ||  
9     chr(trunc(65 + random() * 25)::int) ||  
10    chr(trunc(65 + random() * 25)::int)  
11 FROM generate_series(1, 100000);
```

Створимо HASH індекс:

```
Query  Query History  
1 CREATE INDEX idx_hash ON test_table USING HASH (test_value);
```

Виконаємо тестовий запит на пошук записів за значенням поля test_value.

```
Query  Query History
1  EXPLAIN ANALYZE SELECT * FROM test_table WHERE test_value = 500;
```

Без використання індексу:

✓ Successfully run. Total query runtime: 427 msec. 5 rows affected. ✕

З індексом HASH:

✓ Successfully run. Total query runtime: 171 msec. 4 rows affected. ✕

Як можна побачити, пошук значення з індексом відбувається набагато швидше, оскільки HASH індекс використовується для точного співпадіння значень. Він надає високу продуктивність при пошуку конкретних значень. Використання HASH індексу дозволило швидко знайти відповідні рядки без необхідності сканувати всю таблицю,

Створимо BRIN індекс:

```
Query  Query History
1  CREATE INDEX idx_brin ON test_table USING BRIN (test_value);
```

Виконаємо тестовий запит на пошук записів за значенням поля test_value.

```
Query  Query History
1  EXPLAIN ANALYZE SELECT * FROM test_table WHERE test_value BETWEEN 490 AND 510;
```

Без використання індексу:

✓ Successfully run. Total query runtime: 457 msec. 5 rows affected. ✕

З індексом BRIN:

✓ Successfully run. Total query runtime: 334 msec. 5 rows affected. ✕

Як можна побачити, пошук значення з індексом BRIN працює швидше, ніж без нього. Оскільки BRIN індекс працює так: всі дані діляться на

секції, і кожного разу, коли ми шукаємо мінімальне число, ми дивимось на метадані кожної секції. Зазвичай там зберігається мінімальне і максимальне число секції, але може бути й по іншому. Це дозволяє не проглядати зайвий раз деякі ділянки пам'яті.

Завдання 3

Для тестування тригерів створимо таблицю, для зберігання повідомлень від тригеру.

```
1 -- Table: public.messages
2
3 -- DROP TABLE public.messages;
4
5 CREATE TABLE public.messages
6 (
7     id integer NOT NULL DEFAULT nextval('messages_id_seq'::regclass),
8     date_time timestamp with time zone,
9     message character varying COLLATE pg_catalog."default"
10 )
11
12 TABLESPACE pg_default;
13
14 ALTER TABLE public.messages
15     OWNER to postgres;
```

Команда створення тригеру:

```

1 CREATE OR REPLACE FUNCTION my_trigger_func() RETURNS trigger AS $$
2 DECLARE
3     curs CURSOR FOR SELECT * FROM users;
4     row users%ROWTYPE;
5 BEGIN
6     IF (TG_OP = 'DELETE') THEN
7         INSERT INTO messages (message, date_time) VALUES ('After Delete operation from users table', NOW());
8         RAISE NOTICE 'Successful delete';
9         RETURN OLD;
10    ELSEIF (TG_OP = 'INSERT') THEN
11        IF NEW.user_id < 2000 THEN
12            RAISE NOTICE 'Id can't be less than 2000';
13            RETURN NULL;
14        END IF;
15
16        FOR row IN curs LOOP
17            IF NEW.user_name like row.user_name THEN
18                NEW.user_name = NEW.user_name || "_upd";
19            END IF;
20        END LOOP;
21
22        INSERT INTO messages (message, date_time) VALUES ('After Delete operation from users table', NOW());
23        RAISE NOTICE 'Successful insert';
24        RETURN NEW;
25    END IF;
26 END;
27 $$ language plpgsql;
28
29 DROP TRIGGER IF EXISTS my_trigger ON users;
30
31 CREATE TRIGGER my_trigger
32 AFTER DELETE OR INSERT
33 ON users
34 FOR EACH ROW EXECUTE PROCEDURE my_trigger_func();

```

Data Output Notifications Explain Messages Query History

CREATE TRIGGER

Query returned successfully in 64 msec.

Виконаємо запит видалення з таблиці:

Query Editor

```
1 DELETE FROM users WHERE user_id = 58503 OR user_id = 85912
```

Data Output Notifications Explain Messages Query History

```

NOTICE: Successful delete
NOTICE: Successful delete
DELETE 2

```

Query returned successfully in 68 msec.

Бачимо повідомлення від тригера. Зміст таблиці Messages також був змінений:

Query Editor

1

SELECT * FROM public.messages

2

Data Output

Notifications

Explain

Messages

Query History

	id integer	date_time timestamp with time zone	message character varying
1	1	2020-12-05 03:05:29.754673+03	After Delete operation from users table
2	2	2020-12-05 03:05:58.225874+03	After Delete operation from users table
3	3	2020-12-05 03:05:58.225874+03	After Delete operation from users table

Виконаємо запит додавання даних:

Query Editor	
1	<code>INSERT INTO users VALUES (1234, 'DDDD')</code>
<div><div>Data Output</div><div>Notifications</div><div>Explain</div><div><u>Messages</u></div><div>Query History</div></div>	
<pre>NOTICE: Id can't be less than 2000 INSERT 0 1 Query returned successfully in 76 msec.</pre>	

Тригер повернув «Помилку» через те, що ми не виконали його умову додавання даних. Виконаємо запит, з збільшенням індексів:

```
Query Editor

1  INSERT INTO users VALUES (123456, 'DDDD');
2  INSERT INTO users VALUES (123457, 'DDDD');
```

Data Output Notifications Explain Messages Query History

```
NOTICE:  Successful insert
NOTICE:  Successful insert
INSERT 0 1
```

Бачимо повідомлення від тригера. Зміст таблиці users був змінений наступним чином:

```
1  SELECT * from users WHERE user_name LIKE '%DDDD%'
```





	user_id [PK] integer	user_name character varying
1	123456	DDDD
2	123457	DDDD_upd

Зміст таблиці Logs також був змінений:

Query Editor

```
1 SELECT * FROM public.messages
2
```

[Data Output](#) [Notifications](#) [Explain](#) [Messages](#) [Query History](#)

	 id integer	 date_time timestamp with time zone	 message character varying	
1	1	2020-12-05 03:05:29.754673+03	After Delete operation from users table	
2	2	2020-12-05 03:05:58.225874+03	After Delete operation from users table	
3	3	2020-12-05 03:05:58.225874+03	After Delete operation from users table	
4	4	2020-12-05 03:13:10.783992+03	After Delete operation from users table	
5	5	2020-12-05 03:13:37.307308+03	After Delete operation from users table	
6	6	2020-12-05 03:13:37.307308+03	After Delete operation from users table	
7	7	2020-12-05 03:15:15.630531+03	After Delete operation from users table	
8	8	2020-12-05 03:15:15.630531+03	After Delete operation from users table	
9	9	2020-12-05 03:17:10.451649+03	After Delete operation from users table	
10	10	2020-12-05 03:17:10.451649+03	After Delete operation from users table	
11	11	2020-12-05 03:17:11.861673+03	Insert operation from users table	
12	12	2020-12-05 03:17:11.861673+03	Insert operation from users table	
13	13	2020-12-05 03:17:43.12545+03	After Delete operation from users table	
14	14	2020-12-05 03:17:43.12545+03	After Delete operation from users table	
15	15	2020-12-05 03:17:46.188994+03	Insert operation from users table	
16	16	2020-12-05 03:17:46.188994+03	Insert operation from users table	