

Міністерство освіти і науки України  
Львівський національний університет імені Івана Франка  
Факультет прикладної математики та інформатики  
Кафедра програмування

Лабораторна робота №2  
**“Транзакції в СКБД PostgreSQL”**

Підготував:  
студент групи ПМІ-31  
Процьків Назарій

Львів 2023

**Тема:** Вивчення понять транзакції та управління конкурентним доступом в СКБД PostgreSQL.

**Мета роботи:** Ознайомлення з використанням транзакцій, їх розробкою та застосуванням, рівнями ізоляцій та механізмом управління конкурентним доступом в СКБД PostgreSQL.

Завдання: (Варіант №17)

Розробити базу даних для сайту соціальної мережі. Соціальна мережа підтримує реєстрацію користувачів зі збереженням усіх їхніх деталей (ім'я, прізвище, дата народження, місце проживання, телефони, сайти/e-mail/skype іт.д.), місця і періоди перебування, місця і періоди навчання, роботи, служби, приєднані файли (зображення, фільми, аудіо), які можна пов'язувати із місцями з деталей. Крім того, кожен користувач має можливість розміщувати свої повідомлення на власній сторінці, отримувати на повідомлення «лайки» та коментарі, а також додавати інших користувачів в друзі, або в «чорний список». Додатково користувач повинен мати змогу шукати нових друзів за довільними критеріями.

### Хід роботи

1. Опрацював теоретичні відомості про транзакції, атомарні транзакції, команди, скасування змін, точки збереження, повернення до точок збереження, рівні ізоляції, явні блокування, перевірки цілісності даних, обмеження, блокування та індекси.
2. Написав елементарну транзакцію з двома операціями, одна з яких скасована з допомогою SAVEPOINT та операції повернення до конкретного моменту ROLLBACK TO (Рівень ізоляції - Read Committed).

Операція зі зміни імені користувача з id 124. Початкові дані:

Query

Query History

1

select \* from users where id = 124;

Messages

Data Output

Notifications

</

Транзакція:

Змінив ім'я та прізвище користувача на Pablo Protskiv, зберіг це за допомогою SAVEPOINT та змінив дані ще раз. Зробив відкат до попередньої зміни за допомогою ROLLBACK..

```
3  BEGIN;
4
5  UPDATE users
6  SET firstname = 'Pablo'
7  WHERE id = 124;
8
9  UPDATE users
10 SET lastname = 'Protskiv'
11 WHERE id = 124;
12
13 SAVEPOINT savedFirstLastName;
14
15 UPDATE users
16 SET firstname = 'Petro'
17 WHERE id = 124;
18
19 UPDATE users
20 SET lastname = 'Oleksiyovych'
21 WHERE id = 124;
22
23 rollback to savedFirstLastName;
24
25 COMMIT;
```

Messages   **Data Output**   Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date
1	124	Pablo	Protskiv	2008-03-19

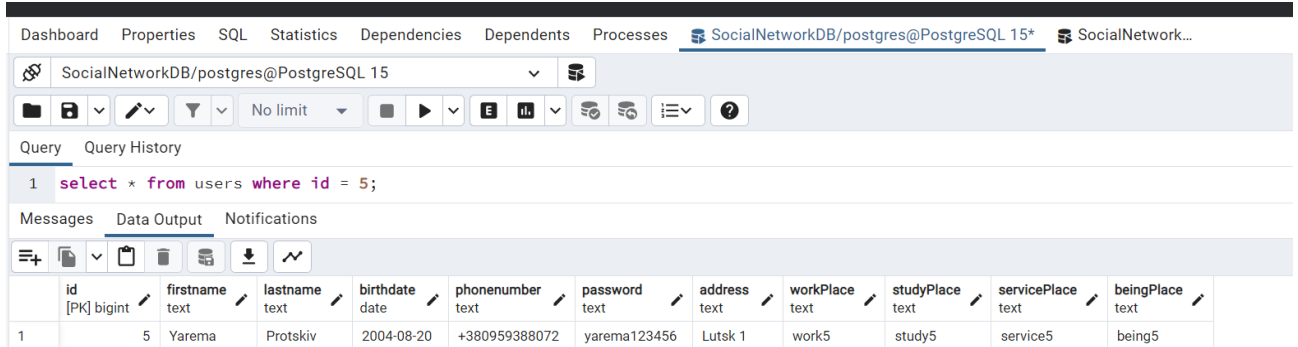
## Проблеми паралельних запитів

Написав транзакції, щоб показати, як вони допомагають вирішити проблеми під час виконання запитів паралельно.

1. Dirty read - може виникнути, коли перша транзакція зчитує дані, які були змінені, але ще не зафіксовані іншою транзакцією:

Приклад 1:

вибір юзера з id = 5:

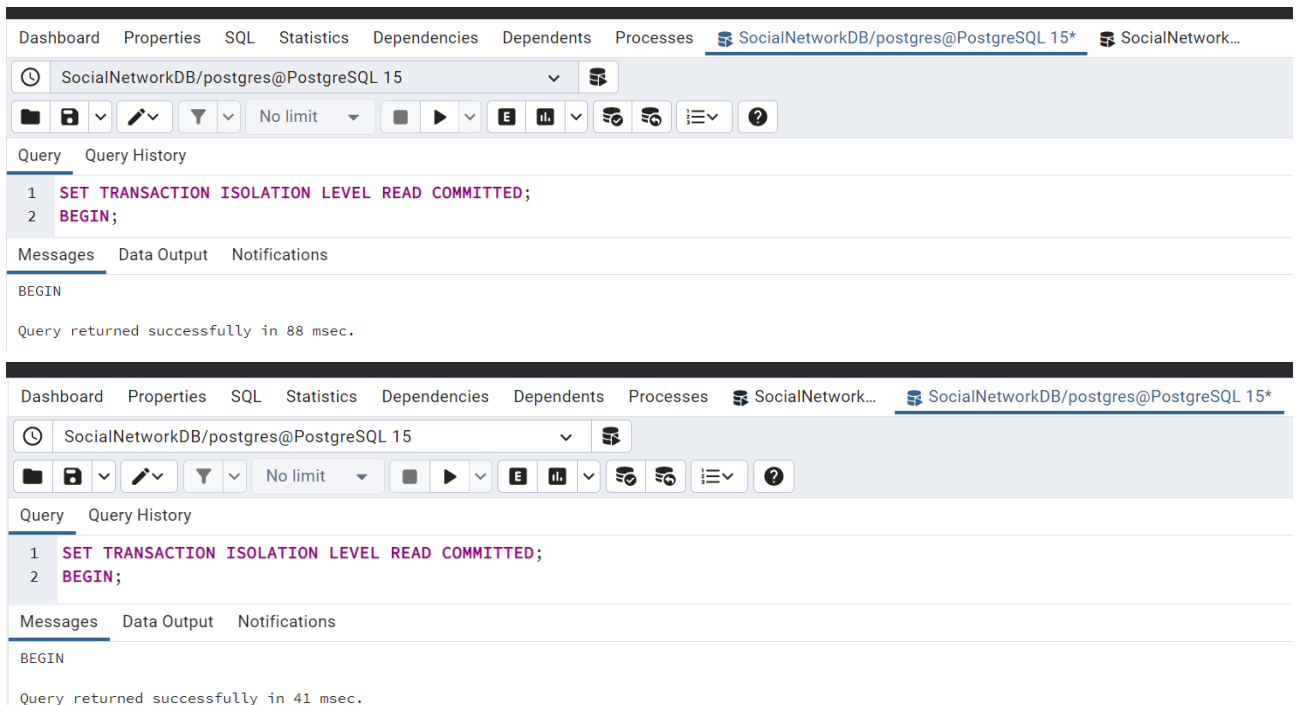


The screenshot shows the pgAdmin interface with the 'Query' tab selected. The query executed is `select * from users where id = 5;`. The 'Data Output' tab shows a single row of data for the user with id 5.

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenumber text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	5	Yarema	Protskiv	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Зараз його прізвище "Protskiv".

Відкрив дві вкладки в середовищі pgAdmin, в них встановив рівень ізоляції "Read Committed":



The first screenshot shows the pgAdmin interface with the 'Query' tab selected. The query executed is `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` followed by `BEGIN;`. The 'Messages' tab shows the message `BEGIN` and `Query returned successfully in 88 msec.`

The second screenshot shows the pgAdmin interface with the 'Query' tab selected. The query executed is `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;` followed by `BEGIN;`. The 'Messages' tab shows the message `BEGIN` and `Query returned successfully in 41 msec.`

Змінив прізвище цього юзера в першій вкладці без коміту:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 UPDATE users
2 SET lastname = 'Tymchyshyn'
3 WHERE users.id = 5;
```

Messages Data Output Notifications

UPDATE 1

Query returned successfully in 77 msec.

В другій вкладці написав SELECT для вибору даних:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetwork... SocialNetworkDB/postgres@PostgreSQL 15\*

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 select * from users where id = 5;
```

Messages Data Output Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenum text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	5	Yarema	Protskiv	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Бачимо, що прізвище користувача з  $id = 5$  не змінилось. Це означає, що друга вкладка не побачила змін. Але якщо дописати COMMIT до першої вкладки

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 UPDATE users
2 SET lastname = 'Tymchyshyn'
3 WHERE users.id = 5;
4 COMMIT;
```

Messages Data Output Notifications

COMMIT

Query returned successfully in 47 msec.

і запустити цей же запит в другій, то зміни вже буде видно:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetwork... SocialNetworkDB/postgres@PostgreSQL 15\*

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 select * from users where id = 5;
```

Messages Data Output Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenum text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	5	Yarema	Tymchyshyn	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Бачимо, що тепер в користувача з  $id = 5$  прізвище Tymchyshyn.

2. Nonrepeatable Read - коли одна і та ж транзакція отримує різні результати:

Встановив в обох вкладках рівень ізолюваності “Repeatable Read”:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 BEGIN;
```

Messages Data Output Notifications

BEGIN

Query returned successfully in 46 msec.

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 BEGIN;
```

Messages Data Output Notifications

BEGIN

Query returned successfully in 39 msec.

В першій вкладці знову написав SELECT:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 select * from users where id = 5;
```

Messages Data Output Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenum text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	5	Yarema	Tymchyshyn	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Бачимо, що на початку прізвище користувача - Тимчишин.

Далі в другій вкладці змінив його на Процьків і зберіг дані комітом.

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 UPDATE users
2 SET lastname = 'Protskiv'
3 WHERE users.id = 5;
4 COMMIT;
```

Messages Data Output Notifications

COMMIT

Query returned successfully in 47 msec.

Тепер знову пишу той же SELECT в першій вкладці

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 select * from users where id = 5;
```

Messages Data Output Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenumber text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	5	Yarema	Tymchyshyn	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Побачив, що прізвище користувача так і не змінилось, це означає, що рівень ізоляції “Repeatable Read” повертає той самий результат, навіть якщо якісь інші транзакції його змінили, тобто він запобігає “Non-repeatable Read”.

3. Phantom read - виникає, коли транзакція читає рядки, які задовольняють умову, а інша транзакція в цей же момент вставляє або видаляє рядки, які також задовольняють цю умову.

В обох вкладках встановив рівень ізоляції “Serializable”:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 BEGIN;
```

Messages Data Output Notifications

BEGIN

Query returned successfully in 38 msec.

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 BEGIN;
```

Messages Data Output Notifications

BEGIN

Query returned successfully in 40 msec.

В першій вкладці виконав SELECT з такою умовою, яку задовольняє кілька рядків:

Dashboard Properties SQL Statistics Dependencies Dependents Processes SocialNetworkDB/postgres@PostgreSQL 15\* SocialNetwork...

SocialNetworkDB/postgres@PostgreSQL 15

Query Query History

```
1 select * from users where lastname = 'Protskiv';
```

Messages Data Output Notifications

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenumber text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1	1	PABLO	Protskiv	2003-12-29	+380963566929	qwerty123456	Lviv, Chervonoi Kalyny 58B	work1	study1	service1	being1
2	124	Pablo	Protskiv	2008-03-19	+380446126493	password124	address124	work124	study124	service124	being124
3	5	Yarema	Protskiv	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Троє користувачів з прізвищем “Процьків”.

В другій вкладці додав в таблицю рядок, який описує ще одного користувача з прізвищем “Процьків”:

The screenshot shows the DBeaver SQL editor interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and SocialNetworkDB/postgres@PostgreSQL 15\*. The SQL editor contains the following query:

```
1 INSERT INTO users VALUES (1001, 'Oleksii', 'Protskiv', '26/12/2003', '+380963566999', 'address1001');
```

Below the query, the Messages tab shows the execution result:

```
INSERT 0 1
```

Query returned successfully in 57 msec.

Знову пишу той же SELECT в першій вкладці:

The screenshot shows the DBeaver SQL editor interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and SocialNetworkDB/postgres@PostgreSQL 15\*. The SQL editor contains the following query:

```
1 select * from users where lastname = 'Protskiv';
```

Below the query, the Data Output tab shows the result set:

	id [PK] bigint	firstname text	lastname text	birthdate date	phonenum text	password text	address text	workPlace text	studyPlace text	servicePlace text	beingPlace text
1		PABLO	Protskiv	2003-12-29	+380963566929	qwerty123456	Lviv, Chervonoi Kalyny 58B	work1	study1	service1	being1
2	124	Pablo	Protskiv	2008-03-19	+380446126493	password124	address124	work124	study124	service124	being124
3	5	Yarema	Protskiv	2004-08-20	+380959388072	yarema123456	Lutsk 1	work5	study5	service5	being5

Як було три кортежі даних, так і лишилося, бо рівень ізольованості “Serializable” ніколи не допускає “Phantom read”.

4. Serialization anomaly - коли транзакції, які виглядають серіалізованими, повертають результати, неможливі під час послідовного виконання тих самих транзакцій:

В першій вкладці оновлюю кортеж користувача з id = 5:

The screenshot shows the DBeaver SQL editor interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, Processes, and SocialNetworkDB/postgres@PostgreSQL 15\*. The SQL editor contains the following query:

```
1 UPDATE users  
2 SET lastname = 'Tymchyshyn'  
3 where id = 5;
```

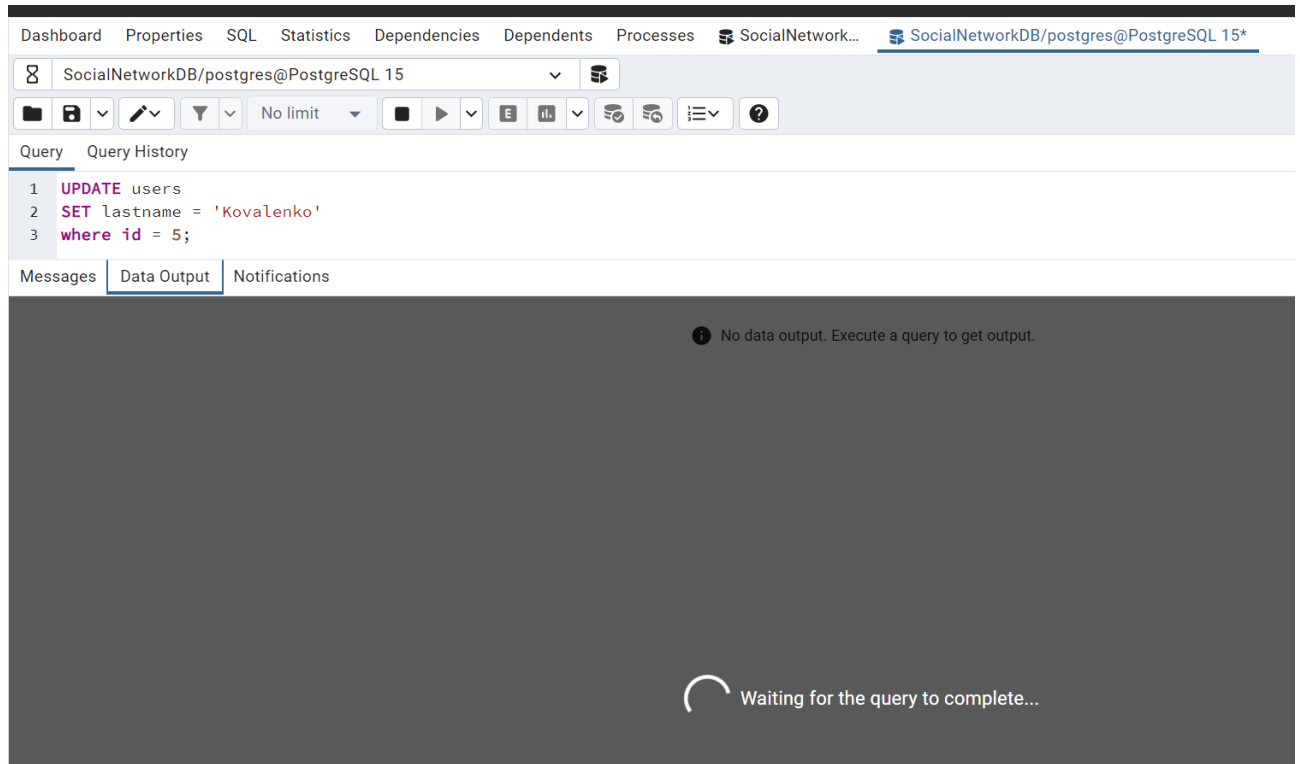
Below the query, the Messages tab shows the execution result:

```
UPDATE 1
```

Query returned successfully in 69 msec.

В другій вкладці написав схожий запит, який змінює цей же кортеж:





Цей запит не виконався, бо рівень ізоляції “Serializable” не допускає “Serialization anomaly”.

**Висновок:** під час виконання лабораторної роботи ознайомився з використанням транзакцій, їх розробкою та застосуванням, рівнями ізоляцій та механізмом управління конкурентним доступом в СКБД PostgreSQL.