ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №10

Тема: «Транзакції в СКБД PostgreSQL».

Виконала:
Ст. *Пелещак Вероніка*
ПМІ-35с

2025

Тема: Транзакції в СКБД PostgreSQL.

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

Завдання:

1. Розробити кілька транзакцій відповідно до власних потреб роботи зі створюваною базою даних. У транзакціях необхідно продемонструвати:
   a. команди початку та фіксації транзакції;
   b. використання точки збереження SAVEPOINT;
   c. відкат виконання транзакції ROLLBACK TO;
2. На прикладі паралельного виконання двох транзакцій заданого рівня ізоляції пояснити їх роботу з погляду видимості змін під час виконання та після фіксації.

Контрольні питання:

1. **Поясніть у чому суть поняття транзакції.**

   Транзакція — це логічна одиниця роботи з базою даних, яка складається з однієї або кількох операцій (запитів). Вона виконується як єдине ціле: або всі її дії застосовуються, або жодна.

2. **Назвіть властивості транзакційю.**

   Атомарність (Atomicity)

   Узгодженість (Consistency)

   Ізоляція (Isolation)

   Довговічність (Durability)

3. **Що означає атомарність транзакції?**

   Атомарність означає неподільність транзакції: усі її дії виконуються повністю, або жодна не виконується. У разі помилки транзакція відміняється й база даних повертається до попереднього стану.

4. **Що означає узгодженість транзакції?**

   Узгодженість гарантує, що після завершення транзакції база даних залишається у правильному стані, який відповідає всім правилам і обмеженням цілісності.

5. **Що означає ізоляція транзакції?**

   Ізоляція забезпечує незалежність виконання транзакцій: проміжні результати однієї транзакції не видно іншим, доки вона не завершиться.

6. **Що означає довговічність транзакції?**

   Довговічність гарантує, що результати успішно завершеної транзакції збережуться навіть у випадку збоїв системи.

7. **Що таке точка збереження транзакції?**

   Точка збереження (savepoint) — це проміжна позначка всередині транзакції, до якої можна повернутися, виконавши частковий відкат без скасування всієї транзакції.

8. **Що таке відкат виконання транзакції?**

   Відкат (rollback) — це операція, яка скасовує всі зміни, виконані транзакцією (або частково — до точки збереження), і відновлює базу даних у стан, що був до її початку.

9. **Які є рівні ізоляції транзакцій?**

   Read Uncommitted — читання непідтверджених даних.

   Read Committed — читання лише підтверджених даних.

   Repeatable Read — повторюваність читання.

   Serializable — повна ізоляція транзакцій, еквівалентна їх послідовному виконанню.

10. **На основі якого механізму реалізовано рівні ізоляції транзакцій?**

    Рівні ізоляції реалізуються за допомогою блокувань (locks) та механізмів контролю конкурентності, які регулюють доступ транзакцій до одних і тих самих даних

11. **Що означає поняття «брудне читання»?**

Ситуація, коли транзакція читає дані, змінені іншою транзакцією, яка ще не завершена. Якщо ця транзакція буде відкатана, прочитані дані виявляться недійсними.

12. Що означає поняття «неповторюване читання»?

Виникає, коли транзакція повторно читає ті самі дані і отримує інші значення, бо інша транзакція їх змінила та підтвердила.

13. Що означає поняття «фантомне читання»?

Трапляється, коли одна транзакція двічі виконує той самий запит із умовою відбору, але між цими операціями інша транзакція додає нові рядки, які також задовольняють умову — у результаті з'являються «фантомні» записи.

14. Що означає поняття «аномалія серіалізації»?

Це порушення логічної коректності, яке виникає тоді, коли результат паралельного виконання транзакцій відрізняється від результату їх послідовного виконання.

Хід роботи:

1. Я створюю новий баг, але перед цим перевіряю, чи існує девелопер, якому хочу призначити. Якщо девелопера нема — усе відкотиться. Якщо модуль обрано неправильно, можна відкотитись тільки до SAVEPOINT.

```
1  BEGIN;
2
3  DO $$
4  BEGIN
5      IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = 2 AND LOWER(user_role) = 'developer') THEN
6          RAISE EXCEPTION 'Developer with ID=2 does not exist or is not a Developer';
7      END IF;
8  END $$;
9
10 INSERT INTO bugs(software_id, created_user_id, assigned_user_id, bug_name, priority, status, bug_module, description)
11 VALUES  (1, 2, 3, 'Некоректне відображення таблиці', 'High', 'New', 'Reports',
12         'При великій кількості записів таблиця не відображається повністю.');
13
14 SAVEPOINT fix_module;
15
16 UPDATE bugs
17 SET bug_module = 'UI'
18 WHERE bug_name = 'Некоректне відображення таблиці';
19
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 48 msec.
```

```
20
21  ROLLBACK TO fix_module;
22
23
24  SELECT * FROM bugs;
25
26
```

Data Output | Messages | Notifications

```
ROLLBACK

Query returned successfully in 56 msec.
```

	assigned_user_id integer	bug_name character varying (100)	priority character varying (20)	status character varying (50)	bug_module character varying (100)	date date	description text
1	1	Нерівний розмір шрифта	Low	New	UI	2025-02-21	Шрифт не масштабується належним чином на мобільних пристро…
2	5	Помилка при запуску програми	Critical	In Progress	Core	2025-03-01	Програма завершується аварійно під час запуску.
3	4	Помилка в системі аутентифікації	High	New	Authentication	2025-01-26	Невірно працює логін через Google.
4	4	Система не зберігає налаштування	Medium	Fixed	Settings	2024-08-27	Після перезапуску програми налаштування не зберігаються.
5	2	Обрізаний експорт аудіо	High	In Progress	Export	2025-03-14	Під час експорту аудіо фінальна частина треку обрізається.
6	9	Некоректний пошук	Medium	Fixed	Search	2025-01-13	Пошук за ключовим словом не повертає результатів.
7	7	Помилка збереження	High	In Progress	Export	2025-04-12	Після збереження проєкт не відкривається, файл пошкоджено.
8	4	Проблема з пензлем	Medium	New	UI	2025-04-11	Пензель не реагує на зміну розміру під час малювання.
9	3	Новий баг	Low	New	UI	2025-03-18	Перевірка оновлення.
10	9	Несправність інтерфейсу	High	New	UI	2024-11-02	Інтерфейс не реагує на кліки кнопок.
11	3	Некоректне відображення таблиці	High	New	Reports	2025-09-21	При великій кількості записів таблиця не відображається повністю.

2. Я змінюю статус конкретного багу на «Fixed», але перед цим перевіряю, чи він дійсно «In Progress». Якщо статус інший, транзакція скасовується.

```
1   BEGIN;
2
3   DO $$
4   BEGIN
5       IF NOT EXISTS (SELECT 1 FROM bugs WHERE bug_id = 2 AND LOWER(status) = 'in progress') THEN
6           RAISE EXCEPTION 'Bug with ID=2 in not in "In Progress" status';
7       END IF;
8   END $$;
9
10
11  UPDATE bugs
12  SET status = 'Fixed'
13  WHERE bug_id = 2;
14
15  SAVEPOINT add_comment;
16
17  INSERT INTO history_corrections(bug_id, user_id, corrected_coment)
18  VALUES (2, 2, 'Баг виправлено: додано обробку винятків при запуску.');
19
20
21  COMMIT;
22
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 57 msec.
```

	corrected_id [PK] integer	bug_id integer	user_id integer	corrected_coment text	date date
1	1	5	4	Додано збереження налаштувань після перезапуску програми	2024-10-02
2	2	2	2	Змінено дизайн кнопок на головній сторінці.	2024-11-15
3	3	1	5	Система аутентифікації була вдосконалена для кращої безпеки.	2025-03-03
4	4	4	2	Додано можливість скидання пароля при аутентифікації через Google.	2025-02-09
5	5	3	3	Шрифт було адаптовано для різних розмірів екранів.	2025-03-01
6	6	6	2	Додано перевірку коректності збереження даних перед експортом.	2025-03-19
7	7	8	7	Зміни на сервері дозволяють зберігати файли без пошкоджень, додано перевірку цілісності.	2025-04-15
8	8	9	4	Покращено обробку пензля для зміни розміру, тепер пензель адаптується до розмірів екрану.	2025-04-12
9	9	7	6	Виправлено помилку пошуку, тепер результати повертаються коректно.	2025-01-22
10	10	2	2	Баг виправлено: додано обробку винятків при запуску.	2025-09-21

	bug_id [PK] integer	software_id integer	created_user_id integer	assigned_user_id integer	bug_name character varying (100)	priority character varying (20)	status character varying (50)	bug_module character varying (100)	date date	description text
1	1	5	3	1	Нерівний розмір шрифта	Low	New	UI	2025-02-21	Шрифт не масшт
2	4	2	2	4	Помилка в системі аутентифікації	High	New	Authentication	2025-01-26	Невірно працює
3	5	4	5	4	Система не зберігає налаштування	Medium	Fixed	Settings	2024-08-27	Після перезапуск
4	6	8	8	2	Обрізаний експорт аудіо	High	In Progress	Export	2025-03-14	Під час експорту
5	7	7	3	9	Некоректний пошук	Medium	Fixed	Search	2025-01-13	Пошук за ключов
6	8	6	7	7	Помилка збереження	High	In Progress	Export	2025-04-12	Після збереженн
7	9	9	4	4	Проблема з пензлем	Medium	New	UI	2025-04-11	Пензель не реагу
8	11	1	2	3	Некоректне відображення таблиці	High	New	Reports	2025-09-21	При великій кіль
9	3	1	5	9	Несправність інтерфейсу	High	New	UI	2024-11-02	Інтерфейс не реа
10	2	3	3	5	Помилка при запуску програми	Critical	Fixed	Core	2025-03-01	Програма завери
11	10	1	2	3	Новий баг	Low	New	UI	2025-03-18	Перевірка оновл

3. **Read Committed:** Цей рівень дозволяє уникнути брудного читання, але можливі неповторювані читання.
   1. **Приклад 1:**

Транзакція 1:

```
1  BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
2
3  SELECT bug_id, status FROM bugs WHERE bug_id = 4;
4
```

Data Output Messages Notifications

	bug_id [PK] integer	status character varying (50)
1	4	In Progress

Транзакція 2:

Query | Query History

```
1  BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
2
3  UPDATE bugs SET status = 'Fixed' WHERE bug_id = 4;
4  COMMIT;
```

Data Output | Messages | Notifications

```
COMMIT
```

Транзакція 1:

```
6
7  SELECT bug_id, status FROM bugs WHERE bug_id = 4;
8  COMMIT;
```

Data Output | Messages | Notifications

	bug_id [PK] integer	status character varying (50)
1	4	Fixed

```
6
7  SELECT bug_id, status FROM bugs WHERE bug_id = 4;
8  COMMIT;
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 137 msec.
```

Результат: у першому запиті транзакція T1 отримала початкове значення статусу багу. Після того як транзакція T2 оновила статус на «Fixed» і зафіксувала зміни, повторний SELECT у T1 вже повернув нове значення. Це демонструє явище **неповторюваного читання**.

2. **Приклад 2:**

Транзакція 1:

```
34
35  BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
36
37  SELECT bug_id, priority FROM bugs WHERE bug_id = 3;
38
39
40
41
```

Data Output | Messages | Notifications

	bug_id [PK] integer	priority character varying (20)
1	3	Medium

Транзакція 2:

```
28
29  BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
30  SELECT bug_id, priority FROM bugs WHERE bug_id = 3;
31  UPDATE bugs SET priority = 'Critical' WHERE bug_id = 3;
32  COMMIT;
33
```

Data Output | Messages | Notifications

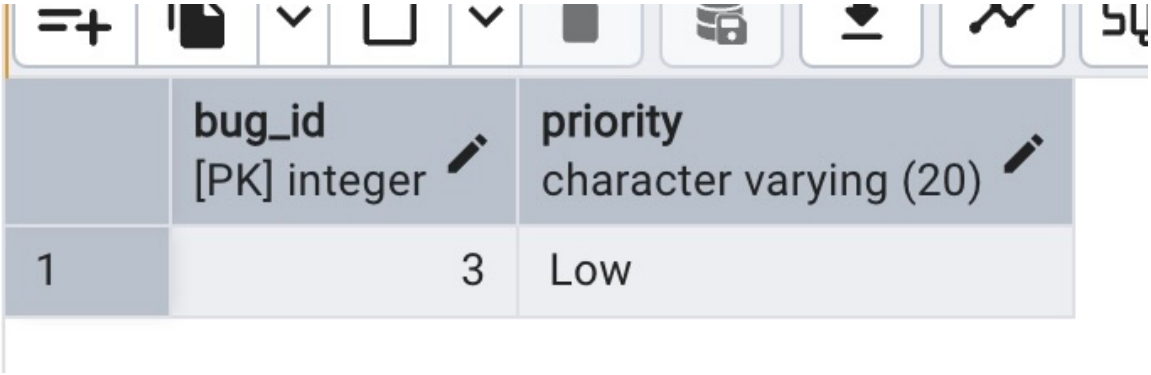
```
COMMIT

Query returned successfully in 122 msec.
```

Транзакція 1:

```
39
40  UPDATE bugs SET priority = 'Low' WHERE bug_id = 3;
41  COMMIT;
42
43
44
```

Data Output | Messages | Notifications

```
COMMIT
```



	bug_id [PK] integer	priority character varying (20)
1	3	Low

Результат: спочатку T1 прочитала початковий пріоритет бага. Паралельно T2 також прочитала те саме значення, змінила його на «Critical» і зафіксувала зміни. Після цього T1 виконала власне оновлення й встановила пріоритет у «Low». У підсумку результат змін T2 було втрачено. Це демонструє проблему **втраченого оновлення.**

4. **Repeatable Read:** Гарантує відсутність брудних та неповторюваних читань, але залишаються можливі фантомні читання.
    1. **Приклад 1:**

Транзакція 1:

```
10
11  BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
12
13  SELECT bug_id, priority FROM bugs WHERE priority = 'High';
14
```

	bug_id [PK] integer	priority character varying (20)
1	11	High
2	6	High
3	8	High

Транзакція 2:

```
7
8   BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
9
10  INSERT INTO bugs(software_id, created_user_id, assigned_user_id, bug_name, priority, status, bug_module, description)
11  VALUES (1, 2, 3, 'Новий баг з високим пріоритетом', 'High', 'New', 'UI', 'Перевірка фантомного читання');
12  COMMIT;
13
14
```

Data Output | Messages | Notifications

```
COMMIT
```

Транзакція 1:

```
16  SELECT bug_id, priority FROM bugs WHERE priority = 'High';
17  COMMIT;
18
19
20
21
22
```

Data Output | Messages | Notifications

```
COMMIT
```

	bug_id [PK] integer	priority character varying (20)
1	11	High
2	6	High
3	8	High

Результат: T1 спочатку прочитала всі баги з пріоритетом «High». Паралельно T2 додала новий баг із пріоритетом «High» і зафіксувала зміни. При повторному SELECT у T1 новий рядок не відобразився, що демонструє блокування **фантомного читання.**

2. **Приклад 2:**

Транзакція 1:

```
45
46  BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
47  SELECT COUNT(*) FROM bugs WHERE priority = 'High';
```

Data Output | Messages | Notifications

	count bigint
1	4

Транзакція 2:

```
35
36  BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
37  SELECT COUNT(*) FROM bugs WHERE priority = 'High';
38  UPDATE bugs SET priority = 'Low' WHERE bug_id = 3;
39  COMMIT;
40
```

Data Output | Messages | Notifications

```
COMMIT
```

Транзакція 1:

```
48  UPDATE bugs SET priority = 'Low' WHERE bug_id = 1;
49  COMMIT;
50
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 67 msec.
```

	count bigint
1	4

Результат: T1 спочатку порахувала кількість багів з пріоритетом «High». Паралельно T2 змінила пріоритет одного з багів на «Critical» і зафіксувала зміни. Після цього T1 оновила пріоритет іншого бага на «Low» і комітнулася. При цьому початковий підрахунок COUNT у T1 не змінився, що демонструє стабільність читаних даних та захист від **фантомного читання**.

5. **Serializable:** Повністю блокує: брудні, неповторювані, фантомні читання. Але можливі аномалії серіалізації → одна з транзакцій може бути відхилена.
   1. **Приклад 1:**

Транзакція 1:

```
23
24  BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
25
26  SELECT COUNT(*) FROM bugs WHERE status = 'New';
27
28  UPDATE bugs SET status = 'In Progress' WHERE bug_id = 1;
29
30
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 78 msec.
```

Транзакція 2:

```
18  BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
19
20  SELECT COUNT(*) FROM bugs WHERE status = 'New';
21
22  UPDATE bugs SET status = 'In Progress' WHERE bug_id = 3;
23  COMMIT;
24
25
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 43 msec.
```

Транзакція 1:

```
31  COMMIT;
32
33
```

Data Output | Messages | Notifications

```
ERROR:  could not serialize access due to read/write dependencies among transactions
```

Результат: При спробі одночасного читання COUNT та оновлення статусів виник конфлікт доступу, через що одна з транзакцій отримала помилку «**could not serialize access due to read/write dependencies among transactions**». Це показує, що при SERIALIZABLE база даних гарантує повну ізоляцію, але при конкурентних змінах деякі транзакції можуть бути відхилені і потребують повторного виконання.

2. Приклад 2:

Транзакція 1:

```
52
53  BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
54  SELECT COUNT(*) FROM bugs WHERE created_user_id = 2;
55  INSERT INTO bugs(software_id, created_user_id, assigned_user_id, bug_name, priority, status, bug_module, description)
56  VALUES (1, 2, 3, 'Bug from T1', 'Medium', 'New', 'Core', 'T1 insert');
57
```

Data Output | Messages | Notifications

```
INSERT 0 1
```

Транзакція 2:

```
44  BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
45  SELECT COUNT(*) FROM bugs WHERE created_user_id = 2;
46  INSERT INTO bugs (software_id, created_user_id, assigned_user_id, bug_name, priority, status, bug_module, description)
47  VALUES (
48      1, 2, 3,
49      'Bug from T2', 'High', 'New', 'Login', 'Example bug inserted in T2'
50  );
51
52
```

Data Output | Messages | Notifications

```
INSERT 0 1
```

Транзакція 1:

```
58
59  COMMIT;
60
61
```

Data Output | Messages | Notifications

```
COMMIT
```

Транзакція 2:

```
52
53  COMMIT;
54
55
56
57
58
```

Data Output | Messages | Notifications

```
ERROR:  could not serialize access due to read/write dependencies among transactions
Reason code: Canceled on identification as a pivot, during commit attempt
```

Результат: Дві транзакції T1 і T2 працювали на рівні SERIALIZABLE і одночасно читали кількість багів для created_user_id = 2, а потім вставляли нові записи. Через високий рівень ізоляції виник конфлікт доступу, оскільки обидві транзакції намагалися одночасно змінювати дані так, щоб виглядати послідовно. У результаті одна з транзакцій могла отримати помилку «**could not serialize access due to read/write dependencies among transactions**» і потребувала повторного виконання, що демонструє строгість ізоляції SERIALIZABLE.

Висновок: Виконані транзакції демонструють основні принципи роботи з базою даних: початок та фіксація транзакції, використання точки збереження (SAVEPOINT) і відкат змін (ROLLBACK TO). Паралельне виконання двох транзакцій показало, як рівень ізоляції визначає видимість змін: одні транзакції бачать лише підтверджені дані інших, а незавершені або відкотані зміни залишаються прихованими до коміту. Це забезпечує узгодженість та контроль над станом бази даних під час одночасної роботи декількох користувачів.