

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №2

Транзакції в СКБД PostgreSQL

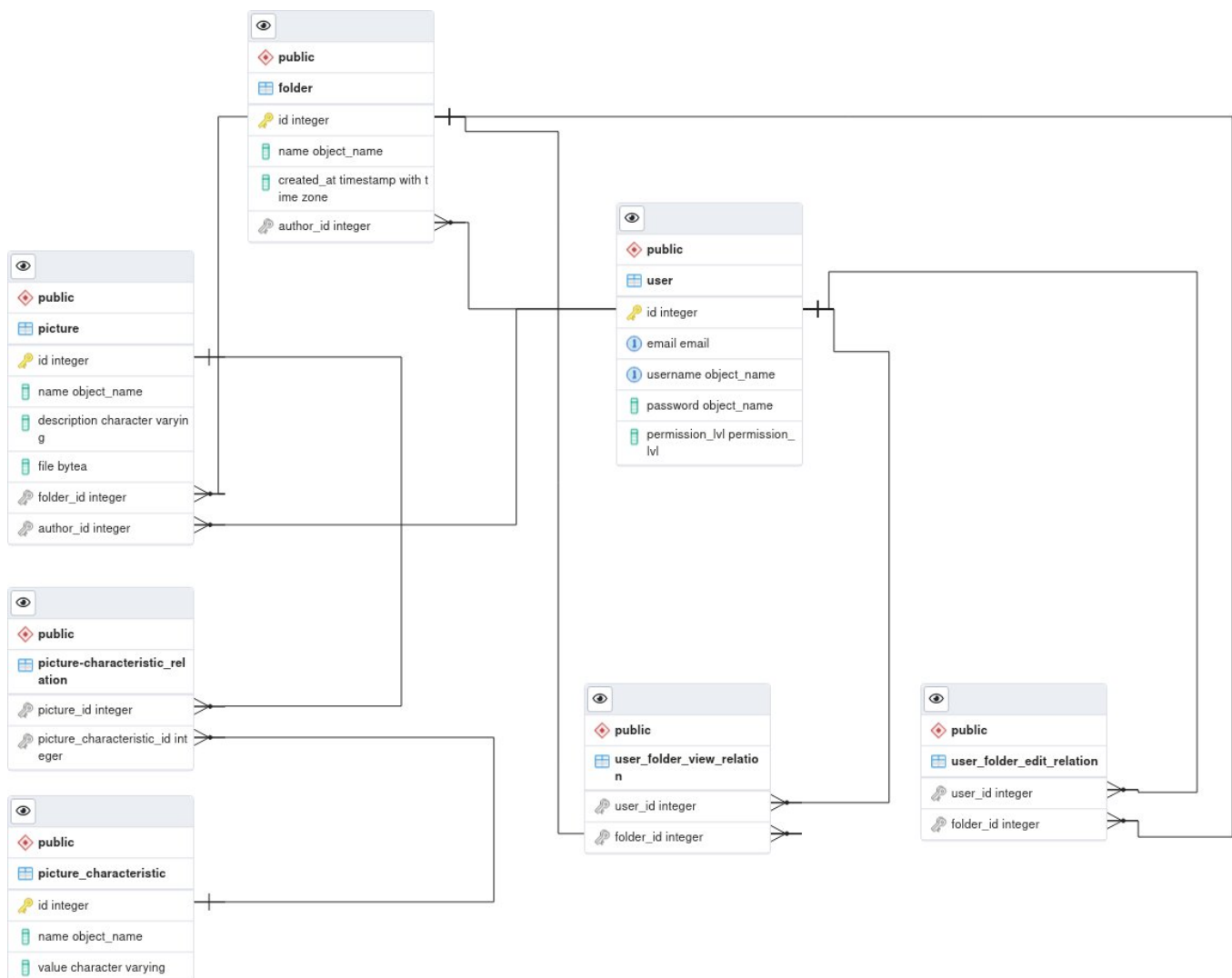
Оцінка

Прийняв:
ас. Жировецький В.В.

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.

ER Діаграма:



Хід роботи

Початкові дані

user:

	id	email	username	password	permission_lvl
1	1	author1@mail.com	first_author	passw	1
2	2	author_2@mail.ua	second_author	passw	1
3	3	author_3@mail.ua	third_author	passw	1
4	4	user@mail.ua	common_user	passw	0
5	5	admin@mail.com	admin_1	passw	2

folder

	id	name	created_at	author_id
1	1	winter	2022-09-25 14:08:54.917928 +00:00	1
2	2	summer	2022-09-25 14:08:54.917928 +00:00	1
3	3	park	2022-09-25 14:08:54.917928 +00:00	3
4	4	screenshots	2022-09-25 14:08:54.917928 +00:00	2
5	5	university	2022-09-25 14:08:54.917928 +00:00	5

picture

	id	name	description	file	folder_id	author_id
17	17	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
18	18	park in the night		/mnt/1/Downloads/index.jpg	3	3
19	19	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
20	20	river	summer river	/mnt/1/Downloads/index.jpg	2	2
21	21	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
22	22	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
23	23	park in the night		/mnt/1/Downloads/index.jpg	3	3
24	24	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
25	25	river	summer river	/mnt/1/Downloads/index.jpg	2	2
26	26	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
27	27	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
28	28	park in the night		/mnt/1/Downloads/index.jpg	3	3
29	29	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
30	30	river	summer river	/mnt/1/Downloads/index.jpg	2	2
31	31	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
32	32	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
33	33	park in the night		/mnt/1/Downloads/index.jpg	3	3
34	34	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
35	35	river	summer river	/mnt/1/Downloads/index.jpg	2	2
36	36	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
37	37	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
38	38	park in the night		/mnt/1/Downloads/index.jpg	3	3
39	39	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
40	40	river	summer river	/mnt/1/Downloads/index.jpg	2	2
41	41	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
42	42	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
43	43	park in the night		/mnt/1/Downloads/index.jpg	3	3
44	44	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
45	45	river	summer river	/mnt/1/Downloads/index.jpg	2	2

user_folder_edit_relation

	user_id	folder_id
1	3	1
2	1	4
3	2	2

user_folder_view_relation

	user_id	folder_id
1	4	1
2	2	1
3	4	4
4	3	3
5	1	3

демонстрація commit і rollback

Для демонстрації коміта і ролбека використаю попередню процедуру, тільки додам умову: "якщо кількість фото що буде змінена (тобто видалена, або змінить автора) > 10, значить ми нічого не робимо з фото. Але повернути ці фото". Це може не сти і певний практичний характер, бо якщо автор зробив багато фото, то краще їх видаляти після прямого зв'язку адміністратора з користувачем.

Код

```
CREATE OR REPLACE PROCEDURE ban(in user_id int, out deleted_photo text[], out
author_changed_photo text[])
AS $$
DECLARE
user_to_ban RECORD;
current_user_id ALIAS FOR user_id;
folders_to_delete int[];
to_append text[];

folders CURSOR FOR SELECT DISTINCT folder.* FROM folder
LEFT JOIN picture ON folder.id = picture.folder_id
WHERE folder.author_id = user_id OR picture.author_id = user_id;

BEGIN
EXECUTE 'SELECT * FROM "user" WHERE id = $1' INTO STRICT user_to_ban
USING user_id;

raise info 'Username of user to ban: %', user_to_ban.username;

IF user_to_ban.permission_lvl != 0 THEN
raise info 'User role changed to reader';
UPDATE "user" SET permission_lvl = 0 WHERE id = user_id;
ELSE
-- додав exception, якого не вистачало в минулій лабораторній
raise exception 'User is already common reader';
END IF;

-- комічу зміни, застосовую операцію зміни ролі користувача до бд
commit;

raise info 'Delete user folders and change authors';
```


та у консоль було виведено

```
Username of user to ban: third_author
User role changed to reader
Delete user folders and change authors
Rollback changes
Delete permissions
[2022-09-25 17:22:11] 1 row retrieved starting from 1 in 20 ms (execution: 6 ms, fetching: 14 ms)
```

Роль користувача з 3 ід змінилась

	id	email	username	password	permission_lvl
1	1	author1@mail.com	first_author	passw	1
2	2	author_2@mail.ua	second_author	passw	1
3	4	user@mail.ua	common_user	passw	0
4	5	admin@mail.com	admin_1	passw	2
5	3	author_3@mail.ua	third_author	passw	0

Так само як і видалились його дозволи

	user_id	folder_id
1	4	1
2	2	1
3	4	4
4	1	3

	user_id	folder_id
1	1	4
2	2	2

Але таблиці **picture** та **folder** залишились без змін

	id	name	description	file	folder_id	author_id
17	17	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
18	18	park in the night		/mnt/1/Downloads/index.jpg	3	3
19	19	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
20	20	river	summer river	/mnt/1/Downloads/index.jpg	2	2
21	21	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
22	22	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
23	23	park in the night		/mnt/1/Downloads/index.jpg	3	3
24	24	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
25	25	river	summer river	/mnt/1/Downloads/index.jpg	2	2
26	26	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
27	27	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
28	28	park in the night		/mnt/1/Downloads/index.jpg	3	3
29	29	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
30	30	river	summer river	/mnt/1/Downloads/index.jpg	2	2
31	31	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
32	32	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
33	33	park in the night		/mnt/1/Downloads/index.jpg	3	3
34	34	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
35	35	river	summer river	/mnt/1/Downloads/index.jpg	2	2
36	36	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
37	37	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
38	38	park in the night		/mnt/1/Downloads/index.jpg	3	3
39	39	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
40	40	river	summer river	/mnt/1/Downloads/index.jpg	2	2
41	41	snow	it is white	/mnt/1/Downloads/index.jpg	1	1
42	42	screen_2020-0-0		/mnt/1/Downloads/index.jpg	4	1
43	43	park in the night		/mnt/1/Downloads/index.jpg	3	3
44	44	snow man	it also white	/mnt/1/Downloads/index.jpg	1	3
45	45	river	summer river	/mnt/1/Downloads/index.jpg	2	2

	id	name	created_at	author_id
1	1	winter	2022-09-25 14:08:54.917928 +00:00	1
2	2	summer	2022-09-25 14:08:54.917928 +00:00	1
3	3	park	2022-09-25 14:08:54.917928 +00:00	3
4	4	screenshots	2022-09-25 14:08:54.917928 +00:00	2
5	5	university	2022-09-25 14:08:54.917928 +00:00	5

демонстрація блоку begin, end і savepoint

чому цього не було в попередній процедурі? Процедури самі по собі вже огорнуті транзакціями, а транзакцію всередині транзакції створити не можна. Також, вкінці транзакції автоматично викликається коміт, а raise exception викликає rollback

```
-- початок транзакції
begin;
    -- зміна яка має відбутись
    UPDATE "user" SET permission_lvl = 2 WHERE id = 1;
    -- точка, до якої ми будемо виконувати ролбек
    savepoint save_point;
    -- зміна, що буде відменена
    UPDATE "user" SET permission_lvl = 1 WHERE id = 1;
    -- робимо ролбек до нашої точки
    rollback to save_point;
end;
```

як ми бачимо, permission_lvl у користувача з ід 1, після виконання транзакції рівний 2

	id	email	username	password	permission_lvl
1	2	author_2@mail.ua	second_author	passw	1
2	4	user@mail.ua	common_user	passw	0
3	5	admin@mail.com	admin_1	passw	2
4	3	author_3@mail.ua	third_author	passw	0
5	1	author1@mail.com	first_author	passw	2

read committed

Рівень ізоляції за замовчуванням. Полягає в тому, що SELECT бачити тільки дані що були передані на початок запиту, тобто тільки закомічені зміни. Тобто SELECT бачить "знімок" бази даних на початок виконання запиту. Але, це стосується незакомічених даних інших сесій, в межах однієї транзакції незакомічені зміни буде видно. В цьому типі ізоляції можливі всі рівні аномалії окрім грязного читання

П.С. - на скріншотах буде результат виконання тільки виділених рядків, а не всього коду

Спочатку запустимо транзакцію, в якій змінимо ім'я користувача

```
73  
74 begin;  
75     UPDATE "user" SET username = 'new_username' WHERE id = 2;  
76     SELECT username FROM "user" WHERE id = 2;  
77 end;
```

Output postgres.public.user

username
1 new_username

після чого запустимо у іншій сесії select

```
1 SELECT username FROM "user" WHERE id = 2
```

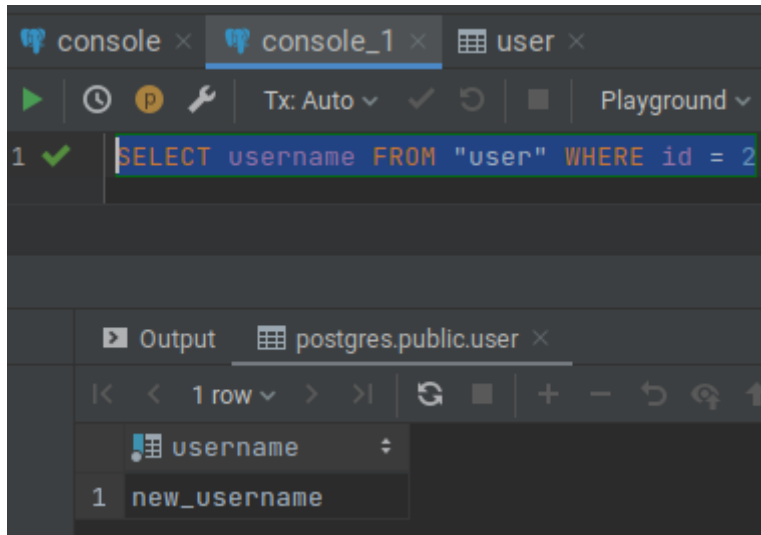
Output postgres.public.user

username
1 second_author

Як можна помітити, у 2 сесії ми не бачимо закомічених змін. Тепер завершимо транзакцію в 1 сесії

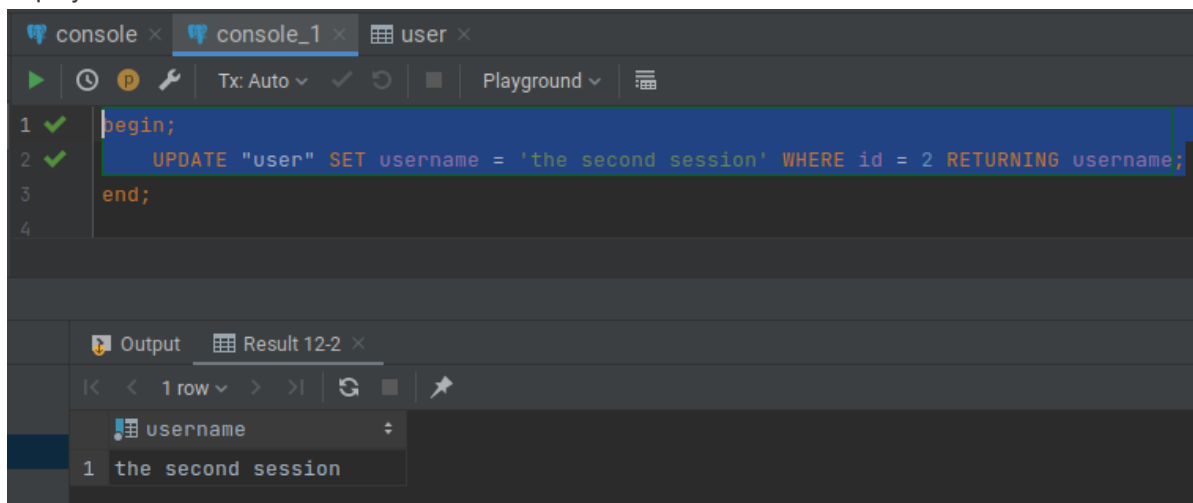
```
73  
74 begin;  
75     UPDATE "user" SET username = 'new_username' WHERE id = 2;  
76     SELECT username FROM "user" WHERE id = 2;  
77 end;
```


І бачимо що тепер, в 2 сесії ми бачимо всі зміни

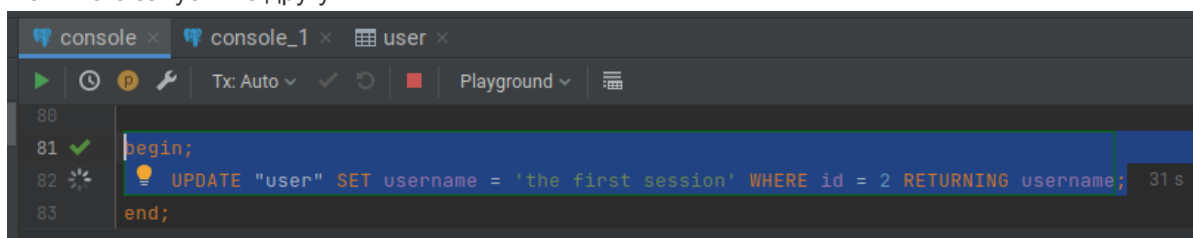


Щодо команд зміни (UPDATE, DELETE, тд), пошук рядків до зміни відбувається так само як і в SELECT. але, може бути таке, що запис, який має бути змінений в цій транзакції(далі друга), буде змінений при закритті іншої(далі перша) паралельної. Тоді друга транзакція буде очікувати на закриття, або відкот першої транзакції. Якщо перша транзакція відкотилась, друга без проблем внесе свої зміни, якщо ж перша транзакція була закомічена, і якщо в ній відбулось видалення, то друга транзакція проігнорує оновлення видалених рядків, інакше спробує внести свої зміни. також WHERE у методах зміни в 2 транзакції буде запущений йще раз.

Тепер спробуємо це на практиці. Створимо дві майже ідентичні транзакції, і спочатку запустимо першу



після чого запустимо другу



як ми бачимо, пройшло 30 секунд, а UPDATE так і не спрацював, бо він очікує на завершення першої транзакції. Завершимо її

```
console × console_1 × user ×
Tx: Auto ✓ Playground
1 begin;
2 UPDATE "user" SET username = 'the second session' WHERE id = 2 RETURNING username;
3 ✓ end;
```

і зразу бачимо що UPDATE у другій транзакції спрацював

```
console × console_1 × user ×
Tx: Auto ✓ Playground
80
81 ✓ begin;
82 ✓ UPDATE "user" SET username = 'the first session' WHERE id = 2 RETURNING username;
83 end;
```

Output Result 11-2

username
1 the first session

Тепер спробуємо видалити елемент в 1 транзакції

Наприклад, в 1 транзакції видалемо папку з ід == 1

```
console × console_1 × user ×
Tx: Auto ✓ Playground
1 ✓ begin;
2 ✓ DELETE FROM "folder" WHERE id = 1 RETURNING name;
3 end;
```

Output Result 21-2

name
1 winter

після чого ми завершуємо першу транзакцію, і бачимо що в другій в нас змінилось рівно 0 елементів

```
console × console_1 × user ×
Tx: Auto ✓ Playground
81 ✓ begin;
82 ✓ UPDATE folder SET name = 'really unique username' WHERE id = 1 RETURNING name;
83 end;
```

Output Result 19-2

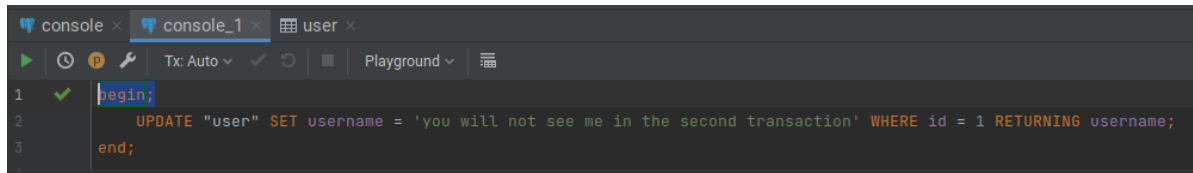
name
0 rows

repeatable read

Рівень ізоляції `repeatable read` бачить лише дані передані на початку транзакції. Тобто, на відміну від попереднього рівня ізоляції, він не бачить зміни що відбулись вже під час виконання ізоляції. Але, так само як в `read committed` буде бачити незакомічені зміни в межах цієї транзакції. Загалом, `read committed` та `repeatable read` відрізняються тим, що у `read committed` "знімок" бази даних береться на початку будь якого окремого запиту, а у `repeatable read` - на початку транзакції

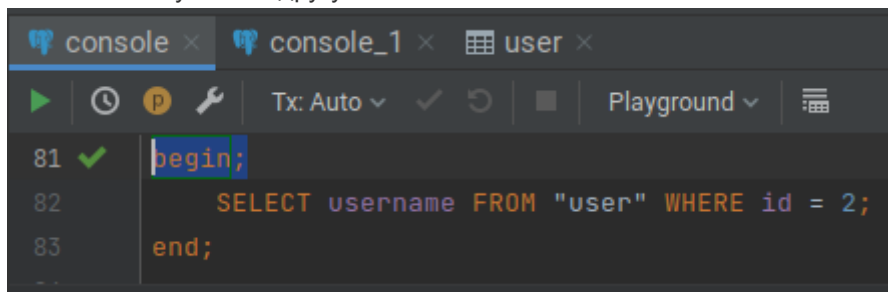
Також, так само як і `read committed`, методи змін бачать очікують і бачать зафіксовані зміни іншими транзакціями.

Перевіримо тези нарахунок `SELECT`. Спочатку просто запустимо першу транзакцію



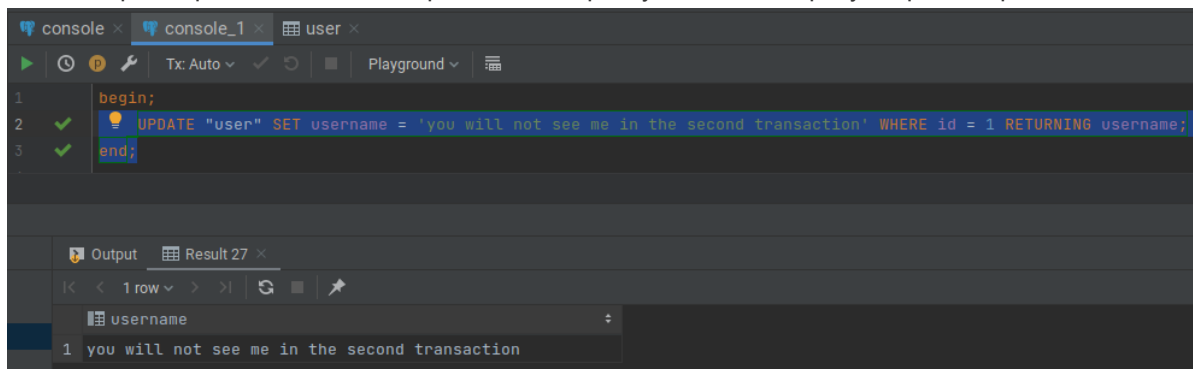
```
console x console_1 x user x
Tx: Auto ✓ Tx: Auto ✓ Tx: Auto ✓
Playground v
1 ✓ begin;
2 UPDATE "user" SET username = 'you will not see me in the second transaction' WHERE id = 1 RETURNING username;
3 end;
```

після чого запустимо і другу



```
console x console_1 x user x
Tx: Auto ✓ Tx: Auto ✓ Tx: Auto ✓
Playground v
81 ✓ begin;
82 SELECT username FROM "user" WHERE id = 2;
83 end;
```

далі, в першій транзакції зміни юзернейм для користувача з ід 1 і зразу закриємо транзакцію

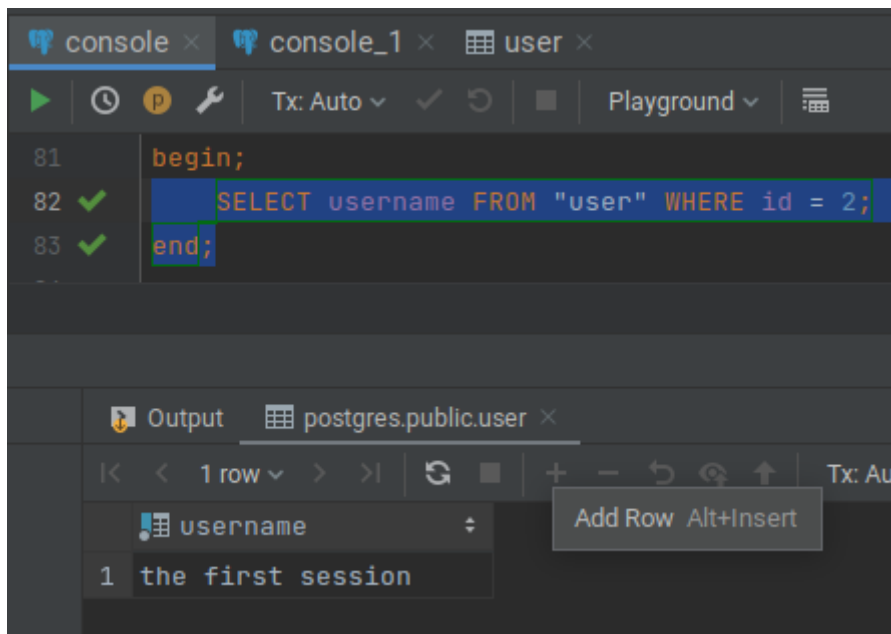


```
console x console_1 x user x
Tx: Auto ✓ Tx: Auto ✓ Tx: Auto ✓
Playground v
1 ✓ begin;
2 ✓ UPDATE "user" SET username = 'you will not see me in the second transaction' WHERE id = 1 RETURNING username;
3 ✓ end;
```



```
Output Result 27 x
1 row v
username
1 you will not see me in the second transaction
```

і виведемо юзернейм цього ж користувача в 1 транзакції



Зазначу, що порядок запуску транзакцій тут ніякої ролі не грає. При скріншотах я помилився з виділенням, і вийшло так що першу транзакцію вже запустив після запуску другої. Але як ми бачимо, зміни в другу транзакції так і не потрапили