

## Лабораторна робота 10. Транзакції та механізми блокування

Мета: Ознайомити студентів із принципами роботи транзакцій у стандарті SQL та в СУБД PostgreSQL, їхніми основними властивостями, а також з механізмами блокувань, які забезпечують цілісність та узгодженість даних.

Завдання:

1. Ознайомитися з основними властивостями транзакцій (ACID);
2. Виконати SQL-оператори для управління транзакціями та блокуваннями;
3. Налаштувати рівні ізоляції транзакцій та виявити аномалії, які можуть виникати;
4. Навчитися аналізувати та вирішувати конфлікти транзакцій та уникати deadlock'ів.

Результат:

Студенти повинні подати SQL-скрипти, що відображають управління транзакціями, налаштування рівнів ізоляції транзакцій та блокуваннями згідно завдання та предметної області, їх опис, а також звіт з результатами тестування.

### Теоретичні відомості до виконання лабораторної роботи

**Транзакцією** називається множина операцій, що виконується застосунком (додатком), яка переводить базу даних з одного коректного стану в інший коректний стан (узгодженість) за умови, що транзакція виконана повністю (атомарність) і без перешкод з боку інших транзакцій (ізоляція).

Основні властивості транзакцій:

Атомарність — транзакція або виконується повністю, або не виконується взагалі.

Узгодженість — після виконання транзакції дані повинні залишатися в коректному стані.

Ізоляція — результати виконання однієї транзакції не повинні впливати на інші паралельні транзакції.

Довговічність — після завершення транзакції її зміни мають залишатися збереженими.

START TRANSACTION вказує системі, що програма починає нову транзакцію. Таку саму дію виконує оператор BEGIN. У PostgreSQL цей оператор видає повідомлення про помилку, якщо програма вже виконує транзакцію. Така поведінка відповідає стандарту SQL, проте деякі інші СУБД у цьому випадку починають вкладену підтранзакцію.

Транзакції мають явний кінець, який відмічається пропозиціями COMMIT (для успішної транзакції – зафіксувати зміни) або ROLLBACK (при виникненні будь-якої помилки в базі даних, транзакція неуспішна та анулюється – відкат транзакції).

Виконання транзакції можна налаштувати, вказавши, наприклад, рівень ізоляції

SET TRANSACTION *режим\_транзакції* [, ...]

де *режим\_транзакції* може бути наступним:

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }

Пояснення рівнів ізоляції див. нижче. Рівень ізоляції Read Committed використовується в PostgreSQL за замовчуванням і, вочевидь, саме цей рівень використовується в абсолютній більшості застосунків.

На наступні транзакції встановлення режиму не впливає. Але можна встановити рівень ізоляції за замовчуванням:

```
ALTER SYSTEM SET default_transaction_isolation = 'режим_транзакції';
```

Нехай існує проста база даних, яка моделює предметну область "Навчальний процес" з наступними таблицями та даними.

```
-- Створення таблиці студентів
```

```
CREATE TABLE students (  
    student_id SERIAL PRIMARY KEY, -- код студента  
    student_name VARCHAR(100) NOT NULL, -- прізвище, ім'я студента  
    enrollment_date DATE NOT NULL -- дата зарахування  
);
```

```
-- Створення таблиці дисциплін
```

```
CREATE TABLE courses (  
    course_id SERIAL PRIMARY KEY, -- код дисципліни  
    course_name VARCHAR(100) NOT NULL, -- назва дисципліни  
    credits INT NOT NULL -- кількість кредитів  
);
```

```
-- Створення таблиці реєстрації на дисципліни
```

```
CREATE TABLE enrollments (  
    student_id INT REFERENCES students(student_id), -- код студента  
    course_id INT REFERENCES courses(course_id), -- код дисципліни  
    grade int, -- оцінка  
    PRIMARY KEY (student_id, course_id) -- складений первинний ключ  
);
```

```
-- Додавання даних до таблиць
```

```
INSERT INTO students (student_name, enrollment_date)  
VALUES  
    ('Alice', '2023-09-01'),  
    ('Bob', '2023-09-01'),  
    ('Charlie', '2023-09-01');
```

```
INSERT INTO courses (course_name, credits)  
VALUES  
    ('Database Systems', 4),  
    ('Operating Systems', 3),  
    ('Algorithms', 4);
```

```
INSERT INTO enrollments (student_id, course_id, grade)  
VALUES  
    (1, 1, 65),  
    (1, 2, 80),  
    (2, 1, 55),  
    (3, 3, 70);
```

### *Приклади транзакцій*

**Проста транзакція** – оновлення оцінок студентів:

```
BEGIN;  
UPDATE enrollments SET grade = 80 WHERE student_id = 1 AND course_id = 1;  
COMMIT;
```

**Транзакція з відкатом** – коригування оцінки, але через помилку транзакція відкачується:

```
BEGIN;  
UPDATE enrollments SET grade = 30 WHERE student_id = 2 AND course_id = 1;  
ROLLBACK;
```

### ***Рівні ізоляції транзакцій та їх особливості***

Стандарт SQL описує чотири рівні ізоляції. Ці рівні визначаються перерахуванням аномалій, які допускаються або не допускаються при одночасному виконанні транзакцій на цьому рівні.

#### ***READ UNCOMMITTED (Читання непідтверджених даних)***

На цьому рівні ізоляції транзакції можуть читати незавершені зміни інших транзакцій. Це найнижчий рівень ізоляції і дозволяє найвищий рівень паралельності, але також є вразливим до всіх видів аномалій.

Для того, щоб повторити наведені команди, треба відкрити два термінали і в кожному запустити psql. Якщо ви працюєте з PgAdmin, достатньо відкрити два різних Query Tools (інструменти запитів). У першому можна вводити команди однієї транзакції, а в другому – команди іншого.

Демонстрація "брудного" читання (dirty read):

```
-- Сеанс 1  
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
UPDATE enrollments SET grade = 70 WHERE student_id = 1 AND course_id = 1;
```

```
-- Сеанс 2  
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 1;  
-- Можливе читання значення 70, хоча транзакція ще не зафіксована
```

```
-- Сеанс 1  
ROLLBACK;
```

```
-- Сеанс 2  
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 1;  
COMMIT;
```

*Цей рівень не підтримується в PostgreSQL, тому найнижчим рівнем для цієї СУБД є READ COMMITTED.*

#### ***READ COMMITTED (Читання підтверджених даних)***

На рівні READ COMMITTED транзакція бачить тільки підтверджені зміни. Це означає, що транзакція не може побачити непідтверджені дані інших транзакцій. Це зменшує ризик "брудного читання" (Dirty Read), але все ще дозволяє "незалежне" та "фантомне читання".

Демонстрація запобігання "брудному" читанню:

```
-- Сеанс 1  
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
UPDATE enrollments SET grade = 75 WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 2
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;
-- Значення буде видно тільки після фіксації транзакції в сеансі 1
```

```
-- Сеанс 1
COMMIT;
```

```
-- Сеанс 2
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;
COMMIT;
```

На цьому рівні можлива аномалія неповторного читання, де один і той же рядок може бути змінений іншою транзакцією між двома запитами SELECT в одній транзакції.

#### *REPEATABLE READ (Повторюване читання)*

Цей рівень гарантує, що дані, прочитані транзакцією, не будуть змінені іншими транзакціями. Однак фантоми можуть з'явитися, коли нові рядки додаються у відповідь на умови запиту.

Демонстрація запобігання "незаблокованому" читанню (non-repeatable read):

```
-- Сеанс 1
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 2;
```

```
-- Сеанс 2
BEGIN;
UPDATE enrollments SET grade = 70 WHERE student_id = 1 AND course_id = 2;
COMMIT;
```

```
-- Сеанс 1: повторне читання покаже те саме значення, яке було на початку транзакції
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 2;
COMMIT;
```

#### *SERIALIZABLE (Серіалізований)*

Рівень SERIALIZABLE гарантує, що транзакції будуть виконуватись, ніби вони йдуть одна за одною, забезпечуючи максимальний захист від аномалій. Це може викликати конфлікти через можливе блокування і вимагати перезапуску транзакцій.

Демонстрація найвищого рівня ізоляції SERIALIZABLE:

```
-- Сеанс 1
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE enrollments SET grade = 75 WHERE student_id = 3 AND course_id = 3;
```

```
-- Сеанс 2
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE enrollments SET grade = 60 WHERE student_id = 3 AND course_id = 3;
-- Транзакція другого сеансу буде відхилена через конфлікт блокувань
```

```
-- Сеанс 1  
COMMIT;
```

### *Аномалії на різних рівнях ізоляції Брудне читання (Dirty Read)*

Відбувається, коли транзакція читає незавершені зміни іншої транзакції. Наприклад, транзакція може прочитати оцінку студента, яку ще не було підтверджено.

```
-- Сеанс 1  
BEGIN;  
UPDATE enrollments SET grade = 73 WHERE student_id = 1 AND course_id = 2;
```

```
-- Сеанс 2  
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 2;  
-- Можливо прочитати нове значення 73 до фіксації транзакції в сеансі 1
```

```
-- Сеанс 1  
ROLLBACK;
```

```
-- Сеанс 2  
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 2;  
COMMIT;
```

Брудне читання допускається стандартом на рівні Read Uncommitted.

### *Неповторюване читання (Non-Repeatable Read)*

Це аномалія, коли одна й та сама транзакція читає значення, яке змінюється іншою транзакцією між читаннями. Наприклад, транзакція двічі виконує SELECT, і між цими двома запитами інша транзакція змінює значення.

```
-- Сеанс 1  
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 2  
BEGIN;  
UPDATE enrollments SET grade = 78 WHERE student_id = 2 AND course_id = 1;  
COMMIT;
```

```
-- Сеанс 1: при повторному читанні значення зміниться  
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;  
COMMIT;
```

Неповторюване читання допускається стандартом на рівнях Read Uncommitted і Read Committed. А ось брудне читання Read Committed не допускає.

### *Фантомне читання (Phantom Read)*

Фантоми з'являються на рівнях ізоляції REPEATABLE READ та нижче, коли в результаті запиту SELECT додаються нові рядки, які відповідають критеріям запиту, створені іншою транзакцією.

```
-- Сеанс 1
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT * FROM enrollments WHERE grade > 60;
```

```
-- Сеанс 2
BEGIN;
INSERT INTO enrollments (student_id, course_id, grade) VALUES (2, 2, 62);
COMMIT;
```

```
-- Сеанс 1: повторний запит не покаже новий рядок, оскільки REPEATABLE READ блокує
фантоми
SELECT * FROM enrollments WHERE grade > 60;
COMMIT;
```

Фантомне читання допускається стандартом на рівнях Read Uncommitted, Read Committed і Repeatable Read. Але на рівні Repeatable Read не допускається неповторюване читання.

### *Втрачена зміна (Lost Update)*

Така аномалія трапляється, коли дві транзакції читають одне й те ж значення, обидві його змінюють, не враховуючи змін, зроблених іншою транзакцією і підтверджують результат. Наприклад, два викладачі збираються оновити оцінку студента одночасно. Кожний з викладачів збирається збільшити сумарну поточну оцінку на 10 балів (реєструють доздачу робіт).

Перша транзакція читає поточну оцінку (50 балів), потім друга транзакція читає те ж саме значення. Перша транзакція збільшує оцінку на 10 балів (виходить 60 б.) і записує це значення. Друга транзакція робить так само – отримує ті ж 60 б. і записує їх. В результаті студент втратив 10 балів.

```
-- Сеанс 1
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 2
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT grade FROM enrollments WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 1
UPDATE enrollments SET grade=grade+10 WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 2
UPDATE enrollments SET grade=grade+10 WHERE student_id = 2 AND course_id = 1;
```

--Виникає блокування транзакції (рис. 1).

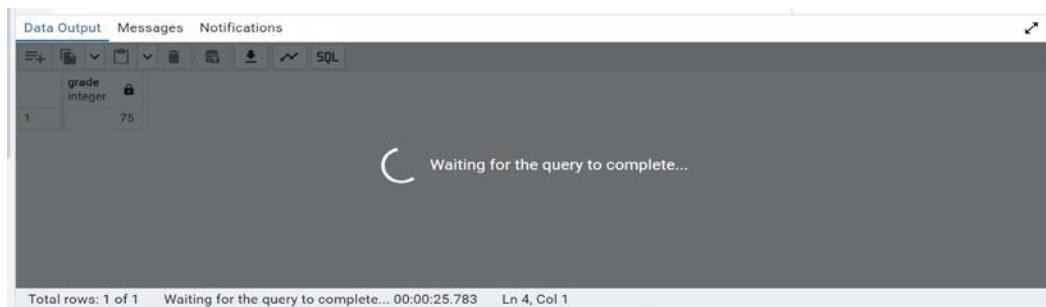


Рисунок 1 – Блокування рядка

```
-- Сеанс 1
COMMIT;
```

Втрачене оновлення не допускається стандартом на жодному рівні ізоляції.

### **Режими блокування та Deadlock**

Механізми блокування в базах даних використовуються для уникнення конфліктів при паралельному виконанні транзакцій. Основні види блокувань:

Shared (спільне) — дозволяє читати дані іншим транзакціям, але не змінювати;

Exclusive (ексклюзивне) — блокує як читання, так і запис даних іншими транзакціями.

Для блокування відношень визначено 8 різних режимів. Така кількість необхідна для того, щоб якомога більша кількість команд, що відносяться до однієї таблиці, могли виконуватись одночасно.

Немає ніякого сенсу вчити ці режими напам'ять, головне – мати в потрібний момент перед очима матрицю, яка показує, які блокування конфліктують один з одним. Матриця наведена в таблиці 1 з прикладами команд, які вимагають відповідні рівні блокування.

Перші 4 режими допускають одночасну зміну даних в таблиці, а наступні 4 – ні.

Перший режим (Access Share) – найслабший, він сумісний з будь-яким іншим, крім останнього (Access Exclusive). Цей останній режим – монопольний, він не сумісний з жодним режимом.

Команда ALTER TABLE має багато варіантів, різні з яких вимагають різних рівнів блокування. Тому в матриці ця команда з'являється в різних рядках і позначена зірочкою.

Таблиця 1 – Матриця блокувань

Режим блокування	AS	RS	RE	SUE	S	SRE	E	AE	приклад SQL-команд
Access Share (AS)								X	SELECT
Row Share (RS)							X	X	SELECT FOR UPDATE/SHARE
Row Exclusive (RE)					X	X	X	X	INSERT, UPDATE, DELETE
Share Update Exclusive (SUE)				X	X	X	X	X	VACUUM, ALTER TABLE*, CREATE INDEX CONCURRENTLY
Share (S)			X	X		X	X	X	CREATE INDEX
Share Row Exclusive (SRE)			X	X	X	X	X	X	CREATE TRIGGER, ALTER TABLE*
Exclusive (E)		X	X	X	X	X	X	X	REFRESH MAT. VIEW CONCURRENTLY
Access Exclusive (AE)	X	X	X	X	X	X	X	X	DROP, TRUNCATE, VACUUM FULL, LOCK TABLE, ALTER TABLE*, REFRESH MAT. VIEW

Наприклад, що станеться, якщо виконати команду CREATE INDEX:

Знаходимо в документації, що ця команда встановлює блокування в режимі Share. По матриці визначаємо, що команда сумісна сама з собою (тобто можна одночасно створювати кілька індексів) і з командами читання. Таким чином, команди SELECT продовжать роботу, а ось команди UPDATE, DELETE, INSERT будуть заблоковані.

І навпаки – незавершені транзакції, що змінюють дані в таблиці, будуть блокувати роботу команди CREATE INDEX. Тому й існує варіант команди – CREATE INDEX CONCURRENTLY. Він працює довше (і може навіть впасти з помилкою), зате допускає одночасну зміну даних.

**Блокування рядків** – при оновленні конкретного рядка.

```
-- Сеанс 1
BEGIN;
UPDATE enrollments SET grade = 74 WHERE student_id = 1 AND course_id = 1;
```

```
-- Сеанс 2
BEGIN;
UPDATE enrollments SET grade = 73 WHERE student_id = 1 AND course_id = 1;
-- Сеанс 2 очікує на звільнення блокування в сеансі 1
```

```
-- Сеанс 1
COMMIT;
```

```
-- Сеанс 2
SELECT grade FROM enrollments WHERE student_id = 1 AND course_id = 1;
COMMIT;
```

**Deadlock (взаємоблокування)** виникає, коли дві транзакції очікують одна на одну, і жодна не може продовжити роботу.

```
-- Сеанс 1
BEGIN;
UPDATE enrollments SET grade = 72 WHERE student_id = 1 AND course_id = 1;
```

```
-- Сеанс 2
BEGIN;
UPDATE enrollments SET grade = 71 WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 1
UPDATE enrollments SET grade = 75 WHERE student_id = 2 AND course_id = 1;
```

```
-- Сеанс 2
UPDATE enrollments SET grade = 69 WHERE student_id = 1 AND course_id = 1;
-- Deadlock: обидва сеанси очікують на блокування іншого
```

### ***Моніторинг блокувань у PostgreSQL***

PostgreSQL надає можливість перевірити блокування та конфлікти між транзакціями через системні представлення.

#### *Перегляд блокувань*

Для перевірки наявності блокувань використовується представлення pg\_locks для виявлення активних блокувань. Цей запит показує всі блокування, які очікують на дозвіл.

```
SELECT * FROM pg_locks WHERE NOT granted;
```

#### *Перевірка транзакцій, які чекають блокування*

Цей запит повертає ідентифікатори процесів (pid), типи блокувань, таблиці та режими, в яких запитуються блокування, що очікують на дозвіл.

```
SELECT pid, locktype, relation::regclass, mode, granted  
FROM pg_locks  
WHERE granted = false;
```

Список типів блокувань (або типів об'єктів), назви приведені у відповідність зі стовпчиком locktype представлення pg\_locks:

- Relation – блокування відношень.
- transactionid і virtualxid – блокування номеру транзакції (справжнього чи віртуального). Кожна транзакція сама утримує виняткове блокування свого власного номера, тому такі блокування зручно використовувати, коли потрібно дочекатися закінчення іншої транзакції.
- Tuple – блокування версії рядка. Використовується іноді для встановлення пріоритету серед кількох транзакцій, які очікують блокування одного і того ж рядка.
- Extend – використовується при додаванні сторінок до файлу будь-якого відношення.
- Object – блокування об'єктів, які не є відношеннями (баз даних, схем, підписок тощо).
- Page – блокування сторінки, використовується іноді і тільки деякими типами індексів.
- Advisory – рекомендаційне блокування, встановлюється користувачем вручну.

Знаходити номер блокуючого процесу, а в загальному випадку – кілька номерів, зручно за допомогою функції pg\_blocking\_pids(номер):

```
SELECT pg_blocking_pids(4782);
```

#### *Виявлення deadlock'ів (взаємоблокувань)*

Для виявлення deadlock'ів PostgreSQL автоматично розриває один із процесів, але, якщо потрібно, можна використовувати розширені засоби моніторингу, такі як pg\_stat\_activity. Представлення pg\_stat\_activity використовують для моніторингу всіх активних транзакцій:

```
SELECT pid, username, state, query  
FROM pg_stat_activity  
WHERE state = 'idle in transaction';
```

Цей запит покаже всі транзакції, що перебувають у стані очікування. Це дозволяє ідентифікувати процеси, які можуть бути причиною deadlock'ів.

#### *Завершення процесу, який викликав блокування*

Вказавши pid процесу, який блокує інші транзакції, можна завершити його, щоб уникнути deadlock.

```
SELECT pg_terminate_backend(<pid>);
```

#### **Висновки**

Розглянуті рівні ізоляції та аномалії дозволяють забезпечити баланс між продуктивністю та цілісністю даних у СУБД. PostgreSQL підтримує потужний набір засобів для роботи з транзакціями і блокуваннями, а також інструменти для моніторингу та усунення проблем.

*Структура звіту до лабораторної роботи*

**Для кожного з запитів представити:**

- 1) словесна постановка задачі, що вирішується;
- 2) SQL-код рішення;
- 3) скріншот отриманого результату.
- 4) опис рівня ізоляції, аномалії, які виникли під час виконання.

1. Виконати транзакції з різними рівнями ізоляції:

1.1) READ COMMITTED. Напишіть дві транзакції, де одна читає дані до завершення змін іншою. Спробуйте виявити аномалію неповторного читання.

1.2) REPEATABLE READ. Запустіть дві транзакції на повторюване читання. Перевірте, чи з'являються нові рядки у вибірці.

1.3) SERIALIZABLE. Виконайте дві транзакції, які оновлюють дані одночасно.

2. Виконати запити, які використовують блокування:

2.1) напишіть транзакцію з ексклюзивним блокуванням та перевірте, як інша транзакція не зможе звернутися до тих же рядків;

3. Демонстрація взаємоблокувань (Deadlock):

3.1) Запустіть дві транзакції, де одна блокує ресурс, який потрібно іншій, і навпаки.

4. Моніторинг блокувань:

4.1) використовуйте представлення pg\_locks для виявлення активних блокувань;

4.2) перегляньте представлення pg\_stat\_activity для моніторингу всіх активних транзакцій

4.3) в ситуації, коли транзакція блокує інші, завершіть блокуючий процес.

5. Пояснити результати виконання кожного запиту.

Звіт до лабораторної роботи 10 можна здати онлайн на сайті ДО [edu.op.edu.ua](http://edu.op.edu.ua) до початку вашого заняття.