

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»  
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ  
**Кафедра системного програмування та спеціалізованих комп'ютерних  
систем**

**Лабораторна робота №3**  
з дисципліни  
**«Бази даних і засоби управління»**  
Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу  
ФПМ групи КВ-94  
Жила Д.М  
Перевірів: доц. Петрашенко А. В.

Київ – 2021

*Мета роботи:* здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

*Загальне завдання роботи полягає у наступному:*

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (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 4

У другому завданні проаналізувати індекси *GIN*, *BRIN*.

Умова для тригера – *after delete, insert*

### Завдання 1

#### Інформація про модель та структуру бази даних

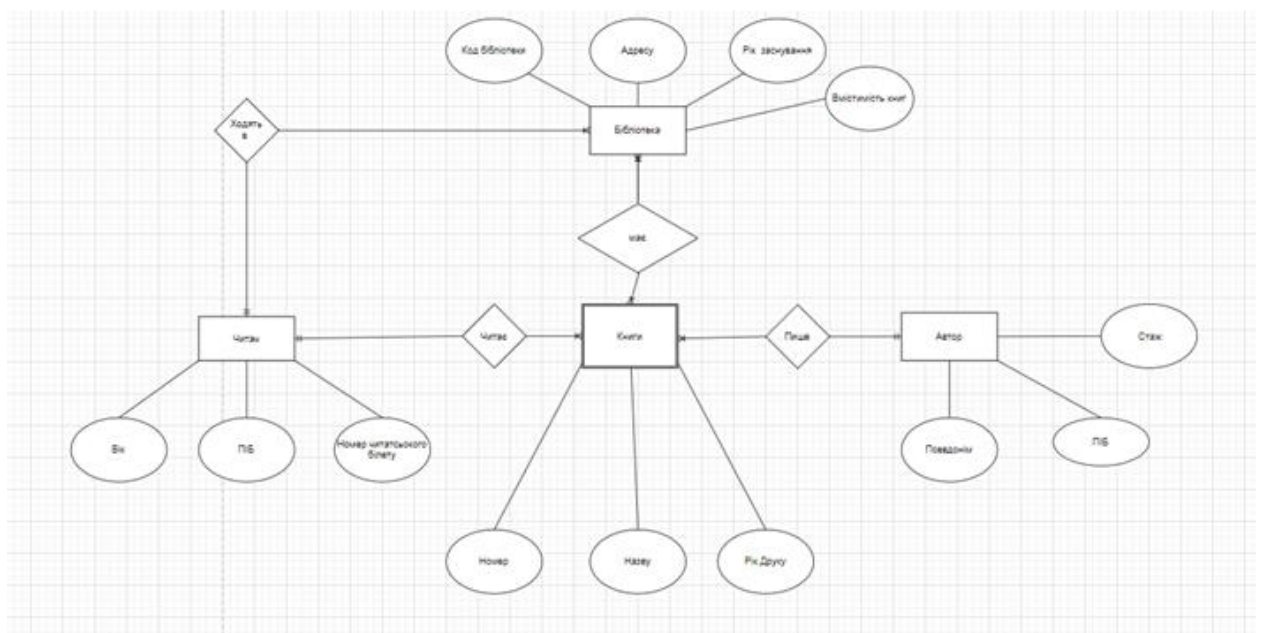


Рис. 1 - Концептуальна модель предметної області “Облік книгозбірні”

Нижче (Рис. 2) наведено логічну модель бази даних:

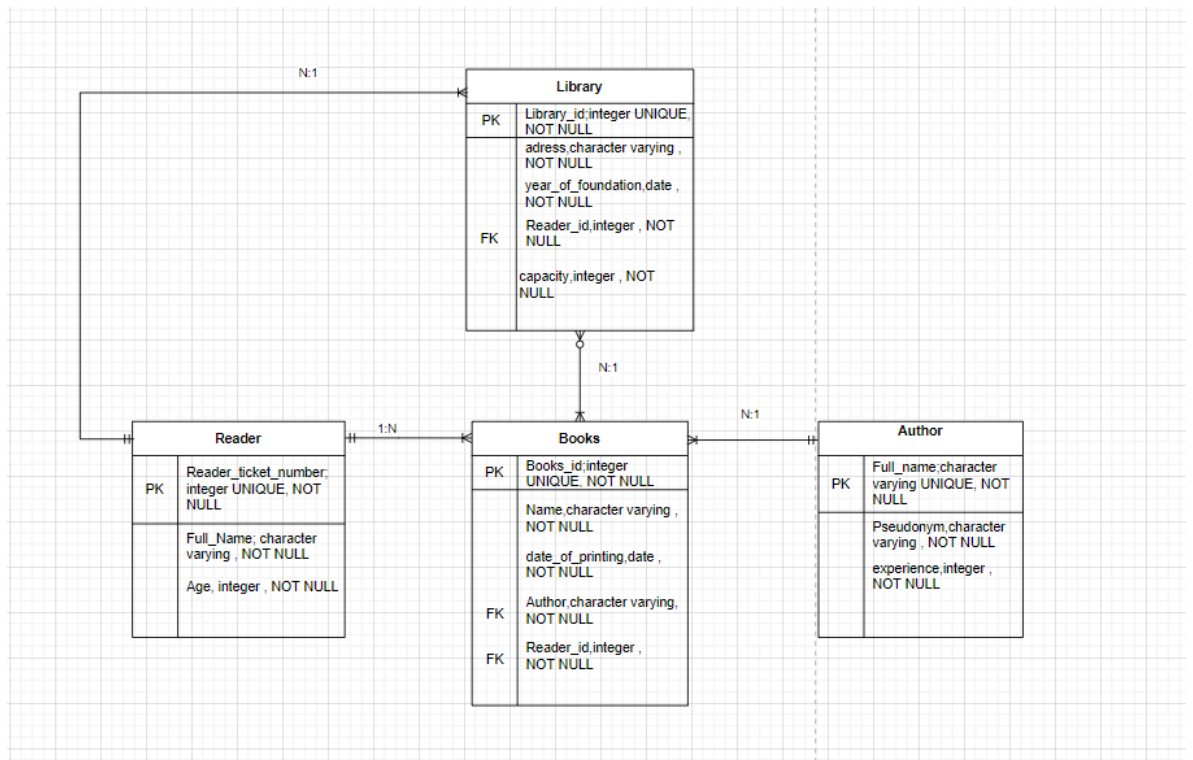


Рис. 2 – Логічна модель бази даних

Для перетворення модуля “Model” програми, створеного в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі було використано бібліотеку “peewee”

Код сутносних класів програми:

```

database_proxy = peewee.DatabaseProxy()

class Library_table(peewee.Model):
    class Meta(object):
        database = database_proxy
        schema = f"Library_loan"

class Author(Library_table):
    Full_name = peewee.CharField(max_length=255, null=False)
    Pseudonym = peewee.CharField(max_length=255, null=False)
    experience = peewee.BigIntegerField(null=False)

class Reader(Library_table):
    Full_Name = peewee.CharField(max_length=255, null=False)
    Age = peewee.BigIntegerField(null=False)

class Books(Library_table):
    date_of_printing = peewee.DateTimeField(null=False)
    Name = peewee.CharField(max_length=255, null=False)
    Author = peewee.ForeignKeyField(Author, backref="books")
    Reader_id = peewee.ForeignKeyField(Reader, backref="readed")
  
```

```

class Library(Library_table):
    year_of_foundation = peewee.DateTimeField(null=False)
    address = peewee.CharField(max_length=255, null=False)
    capacity = peewee.BigIntegerField(null=False)
    Reader_id = peewee.ForeignKeyField(Reader, backref="libraries")

class LibraryBooks(Library_table):
    Library_id = peewee.ForeignKeyField(Library, backref="books")
    Book_id = peewee.ForeignKeyField(Books, backref="libraries")

```

Програма працює ідентично програмі з лабораторної роботи 2, за виключенням незначних текстових змін. Інтерфес модуля «model» не було змінено.

Приклад отримання усіх даних з таблиці «Reader».

```
Reader.select()
```

## Завдання 2

### GIN

Для дослідження індексу була створена таблиця, яка має дві колонки: числову і текстову. Вони проіндексовані як GIN. У таблицю було занесено 1000000 записів.

Створення таблиці та її заповнення:

```

DROP TABLE IF EXISTS "test_gin";

CREATE TABLE "test_gin" (
    "id" bigserial PRIMARY KEY,
    "test_text" varchar(255)
);

INSERT INTO "test_gin" ("test_text")
SELECT
    substr(characters, (random() * length(characters) + 1)::integer, 10)
FROM
    (VALUES ('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000000) as q;

```

Вибір даних без індексу:

```

lab3=# SELECT COUNT(*) FROM "test_gin" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "test_gin" WHERE "id" % 2 = 0 OR "test_text"::text LIKE 'b%';
SELECT COUNT(*), SUM("id") FROM "test_gin" WHERE "test_text"::text LIKE 'b%' GROUP BY "id" % 2;
count
-----
500000
(1 row)

Time: 79,765 ms
count
-----
500000
(1 row)

Time: 137,058 ms
count | sum
-----+-----
(0 rows)

Time: 150,075 ms
lab3=#

```

Сворюємо індекс:

```

DROP INDEX IF EXISTS "test_gin_test_text_index";

CREATE INDEX "test_gin_test_text_index" ON "test_gin" USING gin ("test_text");

```

Вибір даних з створеним індексом:

```
lab3=# SELECT COUNT(*) FROM "test_gin" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "test_gin" WHERE "id" % 2 = 0 OR "test_text"::text LIKE 'b%';
SELECT COUNT(*), SUM("id") FROM "test_gin" WHERE "test_text"::text LIKE 'b%' GROUP BY "id" % 2;
count
-----
500000
(1 row)

Time: 79,966 ms
count
-----
500000
(1 row)

Time: 135,493 ms
count | sum
-----+-----
(0 rows)

Time: 150,690 ms
lab3=#
```

## BRIN

Для дослідження індексу була створена таблиця, яка має дві колонки: t\_data типу timestamp without time zone (дата та час (без часового поясу)) і t\_number типу integer. Колонка t\_data проіндексована як BRIN. У таблицю занесено 1000000 записів.

Створення таблиці та її заповнення:

```
DROP TABLE IF EXISTS "test_brin";

CREATE TABLE "test_brin" (
    "id" bigserial PRIMARY KEY,
    "test_time" timestamp
);

INSERT INTO "test_brin" ("test_time")
SELECT
    (timestamp '2021-01-01' + random() * (timestamp '2020-01-01' - timestamp '2022-01-01'))
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000000) as q;
```

Вибір даних без індексу:

```
SELECT COUNT(*) FROM "test_brin" WHERE "test_time" >= '20200505' AND "test_time" <= '20210505';
SELECT COUNT(*), SUM("id") FROM "test_brin" WHERE "test_time" >= '20200505' AND "test_time" <= '20210505' GROUP BY "id" % 2;
ERROR: syntax error at or near "-"
LINE 1: ~
Time: 0,571 ms
CREATE TABLE
Time: 19,356 ms
INSERT 0 1000000
Time: 6180,769 ms (00:06,181)
count
-----
500000
(1 row)
Time: 204,326 ms
count
-----
329849
(1 row)
Time: 150,296 ms
```

Сворюємо індекс:

```
DROP INDEX IF EXISTS "test_btree_test_text_index";

CREATE INDEX "test_btree_test_text_index" ON "test_btree" USING btree ("test_text");
```

Вибір даних з створеним індексом:

```
count
-----
500000
(1 row)

Time: 166,650 ms
count
-----
329849
(1 row)

Time: 151,440 ms
count |      sum
-----+-----
164862 | 82469276344
164987 | 82496117854
(2 rows)

Time: 211,163 ms
dedmon=#
```

### Завдання 3

Розробити тригер бази даних PostgreSQL.

Умова для тригера – after delete, insert.

Таблиці:

```
DROP TABLE IF EXISTS "reader";
CREATE TABLE "reader" (
    "readerID" bigserial PRIMARY KEY,
    "readerName" varchar(255)
);
```

```
DROP TABLE IF EXISTS "readerLog";
CREATE TABLE "readerLog" (
    "id" bigserial PRIMARY KEY,
    "readerLogID" bigint,
    "readerLogName" varchar(255)
);
```

Тригер:

```
CREATE OR REPLACE FUNCTION update_insert_func() RETURNS TRIGGER as $$

DECLARE
    CURSOR_LOG CURSOR FOR SELECT * FROM "readerLog";
    row_Log "readerLog"%ROWTYPE;

begin
    IF NEW."readerID" % 2 = 0 THEN
        INSERT INTO "readerLog" ("readerLogID", "readerLogName") VALUES (new."readerID",
new."readerName");
        UPDATE "readerLog" SET "readerLogName" = trim(BOTH 'x' FROM "readerLogName");
        RETURN NEW;
    ELSE
        RAISE NOTICE 'readerID is odd';
        FOR row_log IN cursor_log LOOP
            UPDATE "readerLog" SET "readerLogName" = 'y' || row_Log."readerLogName"
|| 'y' WHERE "id" = row_log."id";
        END LOOP;
        RETURN NEW;
    END IF;
end;
```



```

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER "test_trigger"
AFTER UPDATE OR INSERT ON "reader"
FOR EACH ROW
EXECUTE procedure update_insert_func();

```

### Принцип роботи:

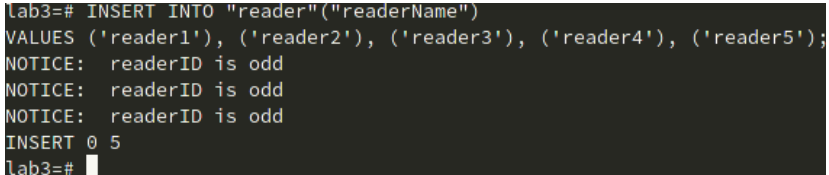
Тригер спрацьовує після оновлення у таблиці чи при додаванні нових рядків у таблицю reader. Якщо значення ідентифікатора запису, який дододється або оновлюється, парне, то цей запис заноситься у додаткову таблицю readerLog. Також, з кожного значення «readerName» видаляються символи «х» на початку і кінці. Якщо значення ідентифікатора непарне, то до кожного значення «readerLogName» у таблиці readerLog додається “у” на початку і кінці.

Занесемо тестові дані до таблиці:

```

INSERT INTO "reader" ("readerName")
VALUES ('reader1'), ('reader2'), ('reader3'), ('reader4'), ('reader5');

```



```

lab3=# INSERT INTO "reader" ("readerName")
VALUES ('reader1'), ('reader2'), ('reader3'), ('reader4'), ('reader5');
NOTICE: readerID is odd
NOTICE: readerID is odd
NOTICE: readerID is odd
INSERT 0 5
lab3=#

```

```

lab3=# SELECT * FROM "reader";
SELECT * FROM "readerLog";

```

readerID	readerName
1	reader1
2	reader2
3	reader3
4	reader4
5	reader5

(5 rows)

id	readerLogID	readerLogName
1	2	yyreader2yy
2	4	yreader4y

(2 rows)

```

lab3=#

```

Оновимо дані в одному з рядків:

```
lab3=# UPDATE "reader" SET "readerName" = "readerName" || 'Lx' WHERE "readerID" = 5;
NOTICE: readerID is odd
UPDATE 1
lab3=# SELECT * FROM "reader";
SELECT * FROM "readerLog";
 readerID | readerName
-----+-----
          1 | reader1
          2 | reader2
          3 | reader3
          4 | reader4
          5 | reader5Lx
(5 rows)

 id | readerLogID | readerLogName
-----+-----
  1 |           2 | yyyreader2yyy
  2 |           4 | yyreader4yy
(2 rows)

lab3=#
```

Оскільки id рядку який було оновлено є непарним числом, то це призвело до додавання до кожного значення «readerLogName» у таблиці readerLog строки “y” на початку і кінці.

Змінемо значення парного рядка:

```
lab3=# UPDATE "reader" SET "readerName" = "readerName" || 'Lx' WHERE "readerID" = 4;
UPDATE 1
lab3=# SELECT * FROM "reader";
SELECT * FROM "readerLog";
 readerID | readerName
-----+-----
          1 | reader1
          2 | reader2
          3 | reader3
          5 | reader5Lx
          4 | reader4Lx
(5 rows)

 id | readerLogID | readerLogName
-----+-----
  1 |           2 | yyyreader2yyy
  2 |           4 | yyreader4yy
  3 |           4 | reader4L
(3 rows)

lab3=#
```

Як бачимо, при оновленні парного рядка його значення буде занесено у таблицю "readerLog" з прибраними символами «x» на початку і кінці.

## Завдання 4

Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Самі транзакції особливих пояснень не вимагають, транзакція — це  $N$  ( $N \geq 1$ ) запитів до БД, які успішно виконуються всі разом або зовсім не виконуються. Ізольованість транзакції показує те, наскільки сильно вони впливають одне на одного паралельно виконуються транзакції.

Вибираючи рівень транзакції, ми намагаємося дійти консенсусу у виборі між високою узгодженістю даних між транзакціями та швидкістю виконання цих транзакцій.

Варто зазначити, що найвищу швидкість виконання та найнижчу узгодженість має рівень `read uncommitted`. Найнижчу швидкість виконання та найвищу узгодженість — `serializable`.

При паралельному виконанні транзакцій можливі виникнення таких проблем:

1. **Втрачене оновлення**

Ситуація, коли при одночасній зміні одного блоку даних різними транзакціями, одна зі змін втрачається.

2. **«Брудне» читання**

Читання даних, які додані чи змінені транзакцією, яка згодом не підтвердиться (відкотиться).

3. **Неповторюване читання**

Ситуація, коли при повторному читанні в рамках однієї транзакції, раніше прочитані дані виявляються зміненими.

4. **Фантомне читання**

Ситуація, коли при повторному читанні в рамках однієї транзакції одна і та ж вибірка дає різні множини рядків.

Стандарт SQL-92 визначає наступні рівні ізоляції:

1. **Serializable (впорядкованість)**

Найбільш високий рівень ізолюваності; транзакції повністю ізолюються одна від одної. На цьому рівні результати паралельного виконання транзакцій для бази даних у більшості випадків можна вважати такими, що збігаються з послідовним виконанням тих же транзакцій (по черзі в будь-якому порядку).

Як бачимо, дані у транзакціях ізолювано.

<pre>START TRANSACTION lab3=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE; SET lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   100   ABC 2   200   BCA 3   300   CAB (3 rows)</pre>	<pre>SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE; START TRANSACTION SET lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   100   ABC 2   200   BCA 3   300   CAB (3 rows)</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<pre>3   300   CAB (3 rows)  lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   100   ABC 2   200   BCA 3   300   CAB (3 rows)  lab3=# █</pre>	<pre>lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# UPDATE "task4" SET "num" = "num" + 1; UPDATE 3 lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   101   ABC 2   201   BCA 3   301   CAB (3 rows)  lab3=# □</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Тепер при оновленні даних в T2(частина фото зправа) бачимо, що T2 блокується поки T1 не зафіксує зміни або не відмінить їх.

<pre>lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   100   ABC 2   200   BCA 3   300   CAB (3 rows)  lab3=# UPDATE "task4" SET "num" = "num" + 1; ERROR: could not serialize access due to concurrent update lab3=# ROLLBACK lab3-!# ; ROLLBACK lab3=# □</pre>	<pre>lab3=# lab3=# UPDATE "task4" SET "num" = "num" + 1; UPDATE 3 lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   101   ABC 2   201   BCA 3   301   CAB (3 rows)  lab3=# COMMIT; COMMIT lab3=# □</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 2. Repeatable read (повторюваність читання)

Рівень, при якому читання одного і того ж рядку чи рядків в транзакції дає однаковий результат. (Поки транзакція не закінчена, ніякі інші транзакції не можуть змінити ці дані).

<pre>lab3=# START TRANSACTION; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE; START TRANSACTION SET lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   101   ABC 2   201   BCA 3   301   CAB (3 rows)  lab3=# UPDATE "task4" SET "num" = "num" + 1; UPDATE 3 lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# lab3=# □</pre>	<pre>lab3=# START TRANSACTION; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE; START TRANSACTION SET lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   101   ABC 2   201   BCA 3   301   CAB (3 rows)  lab3=# SELECT * FROM "task4"; id   num   char -----+----- 1   101   ABC 2   201   BCA 3   301   CAB (3 rows)  lab3=# □</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Тепер транзакція T2(зправа) буде чекати поки T1 не зафіксує зміни або не відмінить їх.



```

lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=#
lab3=# SELECT * FROM "task4";
id | num | char
----+-----+-----
 1 | 104 | ABC
 2 | 204 | BCA
 3 | 304 | CAB
(3 rows)

lab3=# START TRANSACTION;
START TRANSACTION
lab3=*# SELECT * FROM "task4";
id | num | char
----+-----+-----
 1 | 104 | ABC
 2 | 204 | BCA
 3 | 304 | CAB
(3 rows)

lab3=*# SELECT * FROM "task4";
id | num | char
----+-----+-----
 1 | 100 | ABC
 2 | 200 | BCA
 3 | 300 | CAB
(3 rows)

lab3=*#

```

#### 4. Read uncommitted (читання незафіксованих даних)

Найнижчий рівень ізоляції, який відповідає рівню 0. Він гарантує тільки відсутність втрачених оновлень. Якщо декілька транзакцій одночасно намагались змінювати один і той же рядок, то в кінцевому варіанті рядок буде мати значення, визначений останньою успішно виконаною транзакцією. У PostgreSQL READ UNCOMMITTED розглядається як READ COMMITTED.

# Ілюстрації програмного коду на Github

The screenshot shows the GitHub repository page for **DmytroZhyla/BD\_Lab3**. The repository is public and has 3 commits. The file list includes:

- Gin: Add files via upload (now)
- Lab: Add files via upload (now)
- README.md: Update README.md (2 minutes ago)
- lab3.py3: Add files via upload (now)
- postgres.ini.secure: Add files via upload (now)
- schema.png: Add files via upload (now)

The **README.md** file is open, showing the following content:

## BD\_Lab3

Лабораторна робота №3 з дисципліни «Бази даних і засоби управління» Тема: «Засоби оптимізації роботи СУБД PostgreSQL» Студент групи КВ-94 Жила Дмитро

On the right side of the repository page, the **About** section contains the following text:

Лабораторна робота №3 з дисципліни «Бази даних і засоби управління» Тема: «Засоби оптимізації роботи СУБД PostgreSQL» Студент групи КВ-94 Жила Дмитро

The **Releases** section shows: No releases published. [Create a new release](#)

The **Packages** section shows: No packages published. [Publish your first package](#)

The **Languages** section shows a bar chart with the following data:

Language	Percentage
Python	87.0%
PostgreSQL	13.0%

Посилання на репозиторій: [https://github.com/DmytroZhyla/BD\\_Lab3](https://github.com/DmytroZhyla/BD_Lab3)