МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №** 3

з дисципліни «Бази даних і засоби управління»

«**Засоби оптимізації роботи СУБД PostgreSQL**»

Виконав: Скиба Ярослав

Студент групи КВ-93

Перевірив: Павловський В.І.

**Київ 2021**

*Метою роботи* є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

*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.

|  |  |  |
| --- | --- | --- |
| *21* | *Btree, Hash* | *before delete, update* |

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: https://github.com/yaroslavskybadev/DB\_Lab3

**Завдання 1**

ER-діаграма, предметної області “Бібліотека”.

Модель «сутність-зв’язок» намальована у draw.io з використанням нотації Crow’s foot.

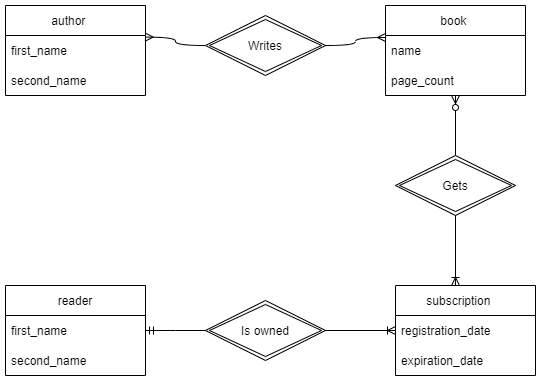


Рисунок 1 - Модель «сутність-зв’язок»

**Опис зв’язків**

У автора може бути багато книг, але обов’язково повинна бути одна. В однієї книги може бути багато авторів, але обов’язково повинен бути один. Книга може використовуватись в багатьох абонементах або може бути відсутня у всіх абонементах. Абонемент може містити багато книг, але принаймні повинна бути одна. Абонемент повинен належати виключно одному читачеві. В одного читача може бути багато абонементів, але обов’язково повинен бути хоча б один.

**Перетворення моделі у схему бази даних**

Таблиця book – відповідає за сутність книги. Таблиця author – відповідає за сутність автора. Ці дві таблиці пов’язані між собою відношення many-to-many, тому потрібно ввести ще одну таблицю book\_author. Таблиця subscription – відповідає за сутність підписки. Таблиці book & subscription пов’язані між собою відношення many-to-many, тому потрібно ввести ще одну таблицю book\_subscription. Таблиця reader – відповідає за сутність читача. Таблиці subscription & reader пов’язані між собою відношення many-to-one, тому потрібно ввести foreign key у таблиці subscription.

Схема «сутність-зв’язок» намальована у draw.io з використанням нотації Crow’s foot.

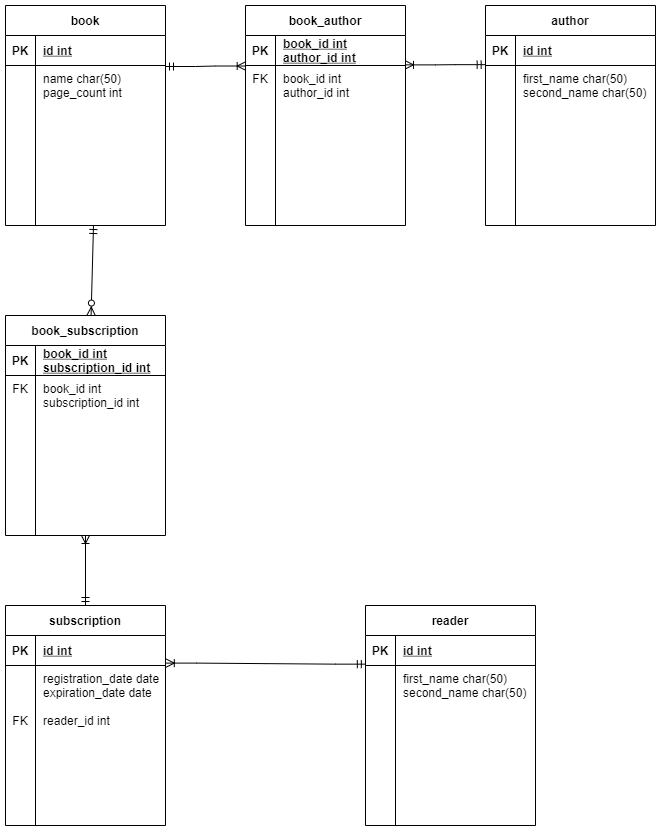


Рисунок 2 - Схема «сутність-зв’язок»

*Author.java*

package com.github.yaroslavskybadev.model;import javax.persistence.CascadeType;import javax.persistence.Column;import javax.persistence.Entity;import javax.persistence.GeneratedValue;import javax.persistence.GenerationType;import javax.persistence.Id;import javax.persistence.JoinColumn;import javax.persistence.JoinTable;import javax.persistence.ManyToMany;import javax.persistence.Table;import javax.validation.constraints.NotNull;import javax.validation.constraints.Size;import java.util.ArrayList;import java.util.List;import java.util.Objects;@Entity@Table(name = "author")public class Author { @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*) private Long id; @NotNull @Size(min = 1, max = 50) @Column(name = "first\_name", nullable = false, length = 50) private String firstName; @NotNull @Size(min = 1, max = 50) @Column(name = "second\_name", nullable = false, length = 50) private String secondName; @ManyToMany(cascade = CascadeType.*ALL*) @JoinTable( name = "book\_author", joinColumns = @JoinColumn(name = "author\_id"), inverseJoinColumns = @JoinColumn(name = "book\_id") ) private final List<Book> bookList = new ArrayList<>(); public Long getId() { return id; } public void setId(Long id) { this.id = id; } public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } public String getSecondName() { return secondName; } public void setSecondName(String secondName) { this.secondName = secondName; } public List<Book> getBookList() { return bookList; } public void addBook(Book book) { bookList.add(book); } public void removeBook(Book book) { bookList.remove(book); } @Override public boolean equals(Object o) { if (this == o) { return true; } if (o == null || getClass() != o.getClass()) { return false; } final Author author = (Author) o; return id.equals(author.id) && firstName.equals(author.firstName) && secondName.equals(author.secondName) && bookList.equals(author.bookList); } @Override public int hashCode() { return Objects.*hash*(id, firstName, secondName, bookList); } @Override public String toString() { return "Author{" + "id=" + id + ", firstName='" + firstName + '\'' + ", secondName='" + secondName + '\'' + ", bookList=" + bookList + '}'; }}

*Book.java*

package com.github.yaroslavskybadev.model;import javax.persistence.CascadeType;import javax.persistence.Column;import javax.persistence.Entity;import javax.persistence.GeneratedValue;import javax.persistence.GenerationType;import javax.persistence.Id;import javax.persistence.JoinColumn;import javax.persistence.JoinTable;import javax.persistence.ManyToMany;import javax.persistence.SequenceGenerator;import javax.persistence.Table;import javax.validation.constraints.Min;import javax.validation.constraints.NotNull;import javax.validation.constraints.Size;import java.util.ArrayList;import java.util.List;import java.util.Objects;@Entity@Table(name = "book")public class Book { @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*) private Long id; @NotNull @Size(min = 1, max = 50) @Column(nullable = false, length = 50) private String name; @Min(1) @Column(name = "page\_count") private Integer pageCount; @ManyToMany(cascade = CascadeType.*ALL*) @JoinTable( name = "book\_author", joinColumns = @JoinColumn(name = "book\_id"), inverseJoinColumns = @JoinColumn(name = "author\_id") ) private final List<Author> authorList = new ArrayList<>(); @ManyToMany @JoinTable( name = "book\_subscription", joinColumns = @JoinColumn(name = "book\_id"), inverseJoinColumns = @JoinColumn(name = "subscription\_id") ) private final List<Subscription> subscriptionList = new ArrayList<>(); public Long getId() { return id; } public void setId(Long id) { this.id = id; } public String getName() { return name; } public void setName(String name) { this.name = name; } public Integer getPageCount() { return pageCount; } public void setPageCount(Integer pageCount) { this.pageCount = pageCount; } public List<Author> getAuthorList() { return authorList; } public void addAuthor(Author author) { authorList.add(author); } public void removeAuthor(Author author) { authorList.remove(author); } public List<Subscription> getSubscriptionList() { return subscriptionList; } public void addSubscription(Subscription subscription) { subscriptionList.add(subscription); } public void removeSubscription(Subscription subscription) { subscriptionList.remove(subscription); } @Override public boolean equals(Object o) { if (this == o) { return true; } if (o == null || getClass() != o.getClass()) { return false; } final Book book = (Book) o; return id.equals(book.id) && name.equals(book.name) && Objects.*equals*(pageCount, book.pageCount) && authorList.equals(book.authorList) && subscriptionList.equals(book.subscriptionList); } @Override public int hashCode() { return Objects.*hash*(id, name, pageCount, authorList, subscriptionList); } @Override public String toString() { return "Book{" + "id=" + id + ", name='" + name + '\'' + ", pageCount=" + pageCount + ", authorList=" + authorList + ", subscriptionList=" + subscriptionList + '}'; }}

*Reader.java*

package com.github.yaroslavskybadev.model;import javax.persistence.CascadeType;import javax.persistence.Column;import javax.persistence.Entity;import javax.persistence.GeneratedValue;import javax.persistence.GenerationType;import javax.persistence.Id;import javax.persistence.OneToMany;import javax.persistence.Table;import javax.validation.constraints.NotNull;import javax.validation.constraints.Size;import java.util.ArrayList;import java.util.List;import java.util.Objects;@Entity@Table(name = "reader")public class Reader { @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*) private Long id; @NotNull @Size(min = 1, max = 50) @Column(name = "first\_name", nullable = false, length = 50) private String firstName; @NotNull @Size(min = 1, max = 50) @Column(name = "second\_name", nullable = false, length = 50) private String secondName; @OneToMany(mappedBy = "reader", cascade = CascadeType.*ALL*, orphanRemoval = true) private final List<Subscription> subscriptionList = new ArrayList<>(); public Long getId() { return id; } public void setId(Long id) { this.id = id; } public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } public String getSecondName() { return secondName; } public void setSecondName(String secondName) { this.secondName = secondName; } public List<Subscription> getSubscriptionList() { return subscriptionList; } public void addSubscription(Subscription subscription) { subscriptionList.add(subscription); } public void removeSubscription(Subscription subscription) { subscriptionList.remove(subscription); } @Override public boolean equals(Object o) { if (this == o) { return true; } if (o == null || getClass() != o.getClass()) { return false; } final Reader reader = (Reader) o; return id.equals(reader.id) && firstName.equals(reader.firstName) && secondName.equals(reader.secondName) && subscriptionList.equals(reader.subscriptionList); } @Override public int hashCode() { return Objects.*hash*(id, firstName, secondName, subscriptionList); } @Override public String toString() { return "Reader{" + "id=" + id + ", firstName='" + firstName + '\'' + ", secondName='" + secondName + '\'' + ", subscriptionList=" + subscriptionList + '}'; }}

*Subscription.java*

package com.github.yaroslavskybadev.model;import javax.persistence.CascadeType;import javax.persistence.Column;import javax.persistence.Entity;import javax.persistence.FetchType;import javax.persistence.GeneratedValue;import javax.persistence.GenerationType;import javax.persistence.Id;import javax.persistence.JoinColumn;import javax.persistence.JoinTable;import javax.persistence.ManyToMany;import javax.persistence.ManyToOne;import javax.persistence.Table;import javax.validation.constraints.NotNull;import java.sql.Date;import java.util.ArrayList;import java.util.List;import java.util.Objects;@Entity@Table(name = "subscription")public class Subscription { @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*) private Long id; @NotNull @ManyToOne(fetch = FetchType.*LAZY*, cascade = CascadeType.*ALL*) @JoinColumn(name = "reader\_id", nullable = false) private Reader reader; @NotNull @Column(name = "registration\_date", nullable = false) private Date registrationDate; @NotNull @Column(name = "expiration\_date", nullable = false) private Date expirationDate; @ManyToMany(cascade = CascadeType.*ALL*) @JoinTable( name = "book\_subscription", joinColumns = @JoinColumn(name = "subscription\_id"), inverseJoinColumns = @JoinColumn(name = "book\_id") ) private final List<Book> bookList = new ArrayList<>(); public Long getId() { return id; } public void setId(Long id) { this.id = id; } public Reader getReader() { return reader; } public void setReader(Reader reader) { this.reader = reader; } public Date getRegistrationDate() { return registrationDate; } public void setRegistrationDate(Date registrationDate) { this.registrationDate = registrationDate; } public Date getExpirationDate() { return expirationDate; } public void setExpirationDate(Date expirationDate) { this.expirationDate = expirationDate; } public List<Book> getBookList() { return bookList; } public void addBook(Book book) { bookList.add(book); } public void removeBook(Book book) { bookList.remove(book); } @Override public boolean equals(Object o) { if (this == o) return true; if (o == null || getClass() != o.getClass()) return false; Subscription that = (Subscription) o; return id.equals(that.id) && reader.equals(that.reader) && registrationDate.equals(that.registrationDate) && expirationDate.equals(that.expirationDate) && bookList.equals(that.bookList); } @Override public int hashCode() { return Objects.*hash*(id, reader, registrationDate, expirationDate, bookList); } @Override public String toString() { return "Subscription{" + "id=" + id + ", reader=" + reader + ", registrationDate=" + registrationDate + ", expirationDate=" + expirationDate + ", bookList=" + bookList + '}'; }}

*Запити у вигляді ORM*

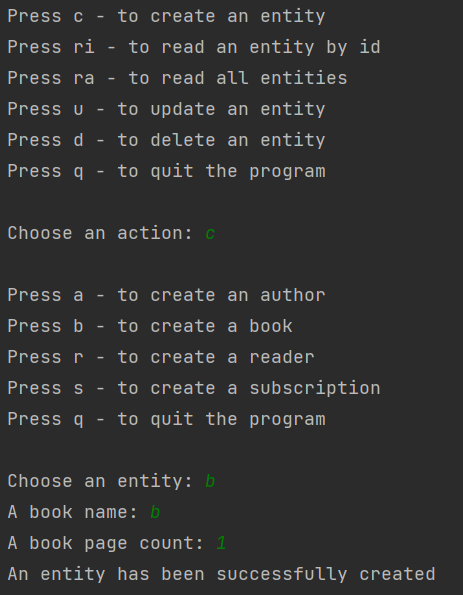


Рисунок 3 - Створення запису

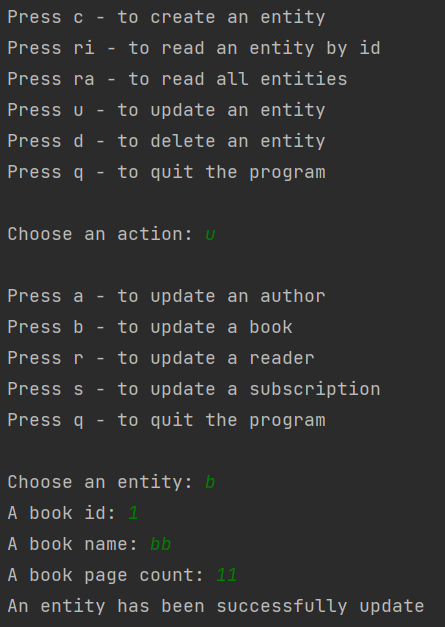


Рисунок 4 - Модифікування запису

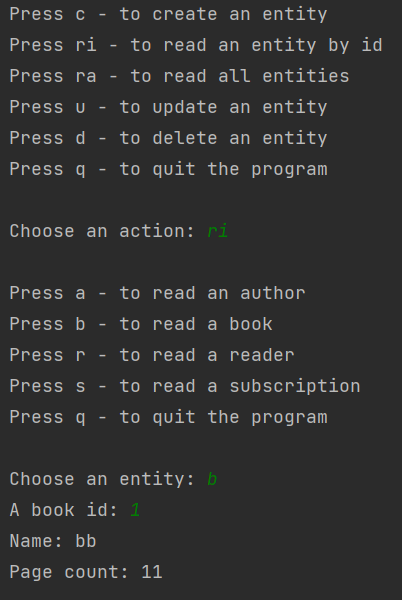


Рисунок 5 - Зчитування запису

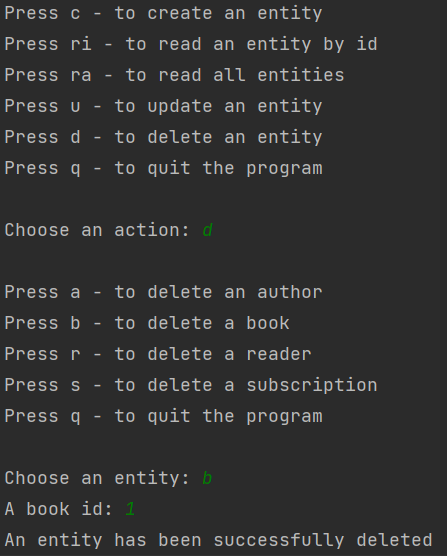


Рисунок 6 - Видалення запису

**Завдання 2**

*Команда створення bree індексу для таблиці book*

create index book\_btree\_index on book using btree (page\_count);

*Команда створення hash індексу для таблиці subscription*

create index subscription\_hash\_index on subscription using hash (reader\_id);

*Команди використання bree індексу для таблиці book*

select \* from book where page\_count = 1;

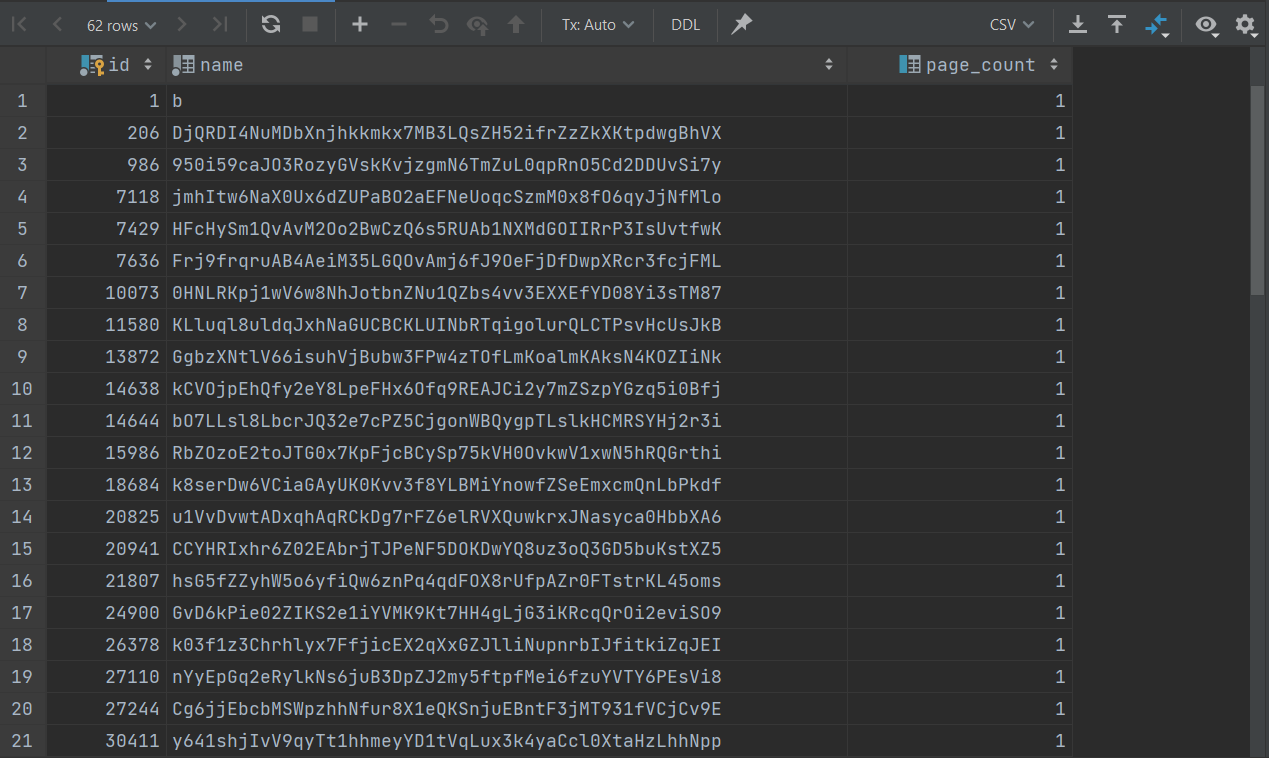
Рисунок 7 — таблиця book

Рисунок 8 — час виконання запиту

select avg(page\_count) from book;

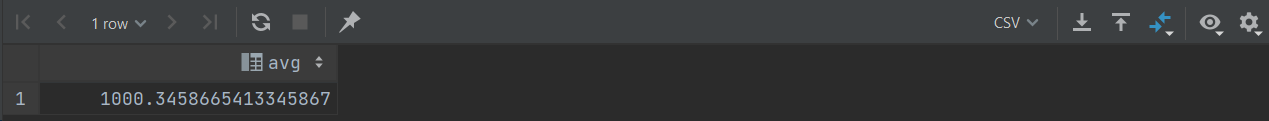
Рисунок 9 — таблиця book

Рисунок 10— час виконання запиту

select page\_count from book group by page\_count;

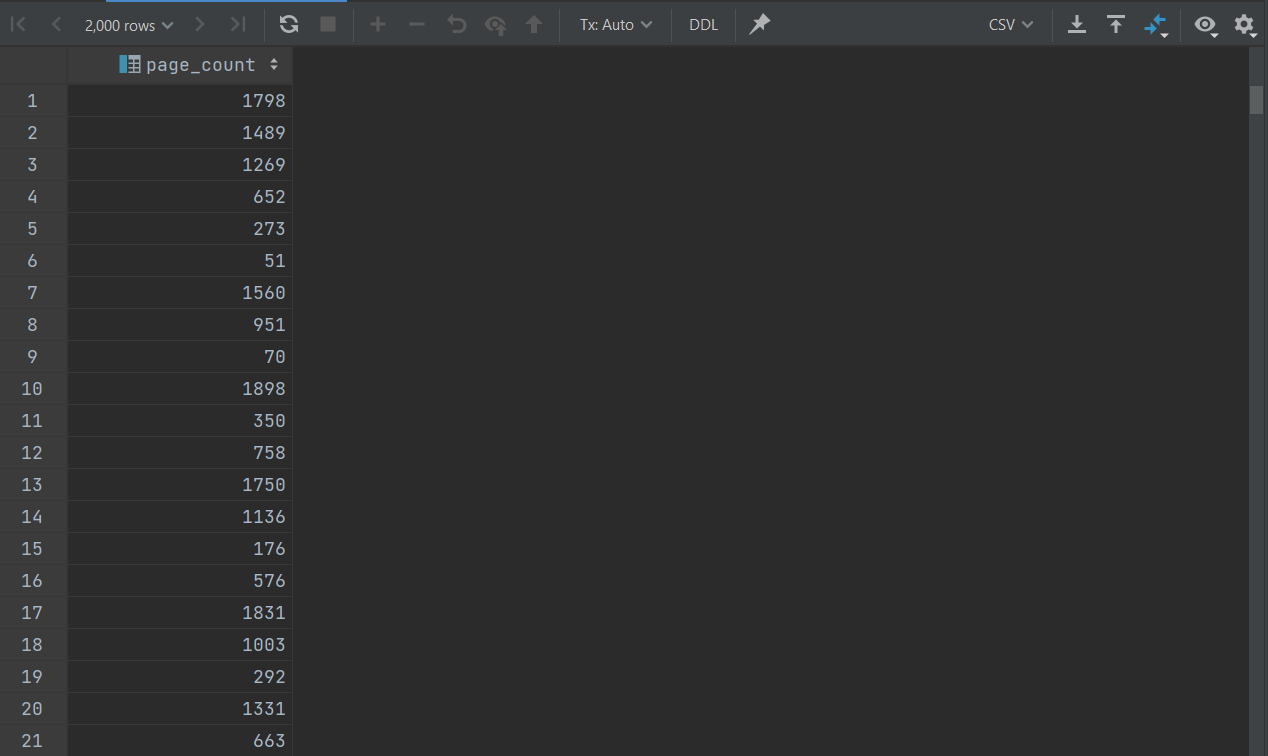
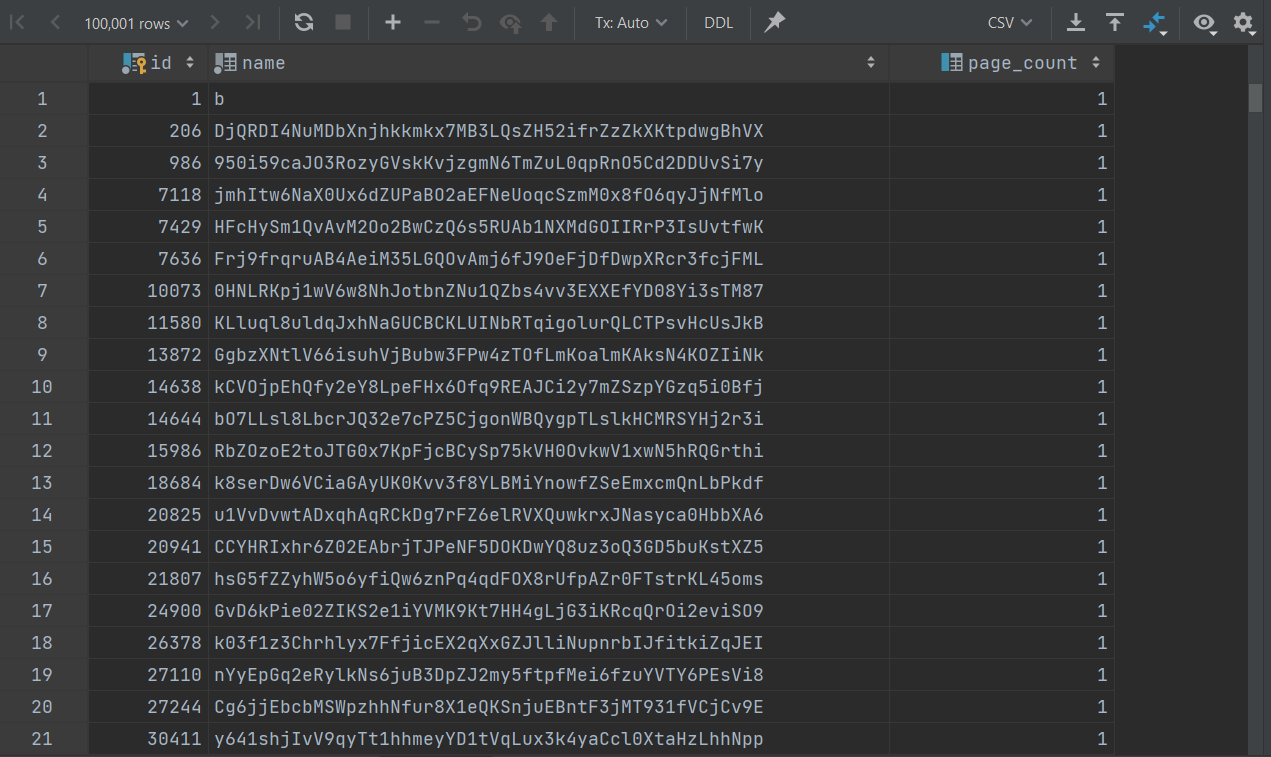
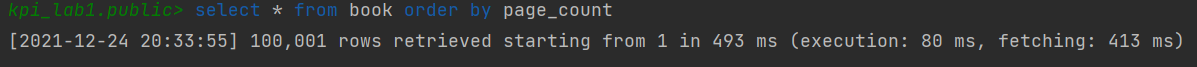
Рисунок 11 — таблиця book

Рисунок 12 — час виконання запиту

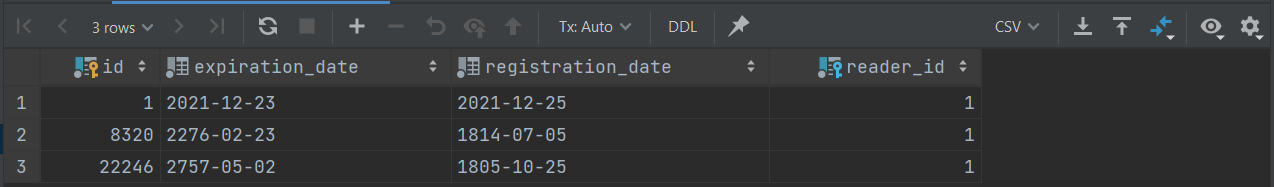
select \* from book order by page\_count;

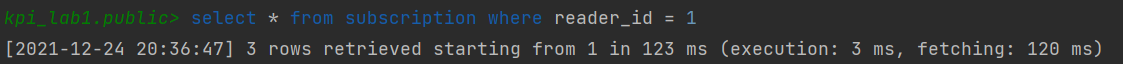
Рисунок 13 — таблиця book

Рисунок 14 — час виконання запиту

*Команди використання hash індексу для таблиці subscription*

select \* from subscription where reader\_id = 1;

Рисунок 15 — таблиця reader

Рисунок 16 — час виконання запиту

select avg(reader\_id) from subscription where reader\_id = 1;

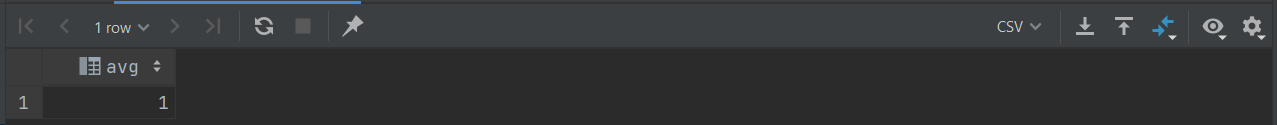
Рисунок 17 — таблиця reader



Рисунок 18 — час виконання запиту

select reader\_id from subscription where reader\_id = 1 group by reader\_id;

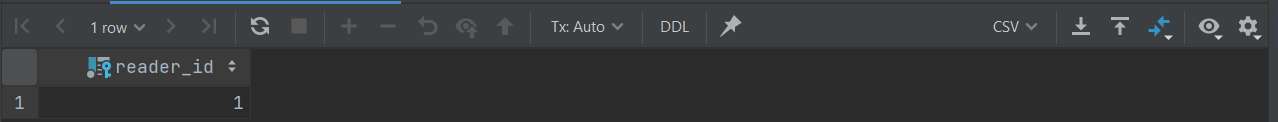
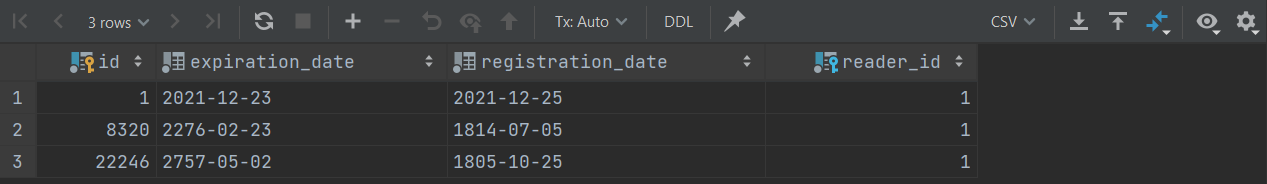
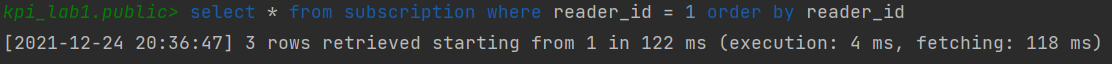
Рисунок 19 — таблиця reader

Рисунок 20 — час виконання запиту

select \* from subscription where reader\_id = 1 order by reader\_id;

Рисунок 21 — таблиця reader

Рисунок 22 — час виконання запиту

**Завдання 3**

*Створення допоміжних таблиць*

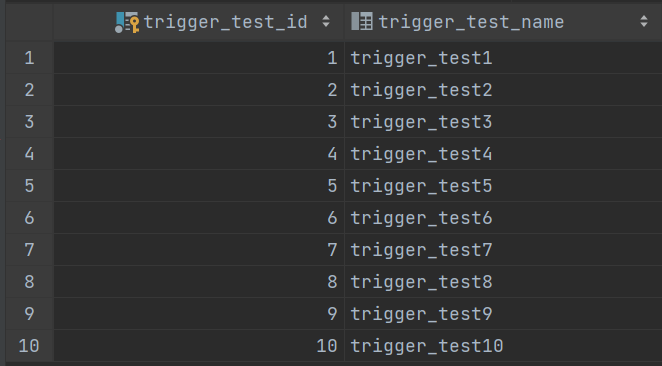
DROP TABLE IF EXISTS trigger\_test;CREATE TABLE trigger\_test(trigger\_test\_id bigserial PRIMARY KEY, trigger\_test\_name text);DROP TABLE IF EXISTS trigger\_test\_log;CREATE TABLE trigger\_test\_log(id bigserial PRIMARY KEY, trigger\_test\_log\_id bigint, trigger\_test\_log\_name text);

*Створення тригера*

CREATE OR REPLACE FUNCTION *before\_delete\_update\_func*() RETURNS TRIGGER as $trigger$DECLARE CURSOR\_LOG CURSOR FOR SELECT \* FROM trigger\_test\_log; row\_ trigger\_test\_log%ROWTYPE;BEGIN IF old.trigger\_test\_id % 2 = 0 THEN IF old.trigger\_test\_id % 3 = 0 THEN RAISE NOTICE 'trigger\_testID is multiple of 2 and 3'; FOR row\_ IN CURSOR\_LOG LOOP UPDATE trigger\_test\_log SET trigger\_test\_log\_name = '\_' || row\_.trigger\_test\_log\_name || '\_log' WHERE id = row\_.id; END LOOP; RETURN OLD; ELSE RAISE NOTICE 'trigger\_testID is even'; INSERT INTO trigger\_test\_log(trigger\_test\_log\_id, trigger\_test\_log\_name) VALUES (old.trigger\_test\_id, old.trigger\_test\_name); UPDATE trigger\_test\_log SET trigger\_test\_log\_name = trim(BOTH '\_log' FROM trigger\_test\_log\_name); RETURN NEW; END IF; ELSE RAISE NOTICE 'trigger\_testID is odd'; FOR row\_ IN CURSOR\_LOG LOOP UPDATE trigger\_test\_log SET trigger\_test\_log\_name = '\_' || row\_.trigger\_test\_log\_name || '\_log' WHERE id = row\_.id; END LOOP; RETURN OLD; END IF;END$trigger$ LANGUAGE plpgsql;CREATE TRIGGER before\_delete\_update\_trigger BEFORE DELETE OR UPDATE ON trigger\_test FOR EACH ROWEXECUTE procedure *before\_delete\_update\_func*();

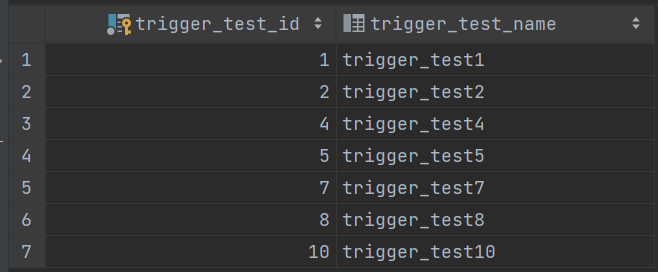
*Занесення тестових даних*

INSERT INTO trigger\_test(trigger\_test\_name) VALUES ('trigger\_test1'),('trigger\_test2'), ('trigger\_test3'), ('trigger\_test4'), ('trigger\_test5'), ('trigger\_test6'), ('trigger\_test7'), ('trigger\_test8'), ('trigger\_test9'), ('trigger\_test10');

Рисунок 23 — Таблиця trigger\_test

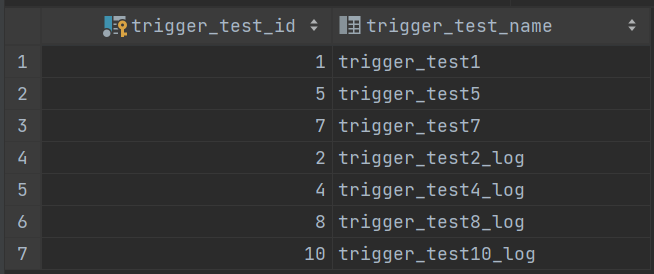
*Видалення кожного третього елемента з таблиці trigger\_test*

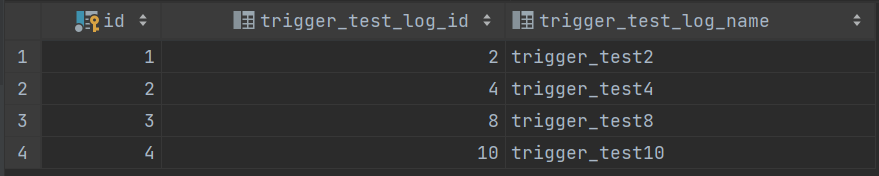
DELETE FROM trigger\_test WHERE trigger\_test\_id % 3 = 0;

Рисунок 24 — Таблиця trigger\_test після видалення

*Оновлення рядків таблиці trigger\_test*

UPDATE trigger\_test SET trigger\_test\_name = trigger\_test\_name || '\_log' WHERE trigger\_test\_id % 2 = 0;

Рисунок 25 — Таблиця trigger\_test після оновлення

Рисунок 26 — Таблиця trigger\_test\_log після оновлення