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»**

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-82

Мороз А.С.

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2020

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

*Загальне завдання роботи полягає у наступному*:

1. Перетворити модуль “Модель” з шаблону MVC лабораторній роботі №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

Варіант 17

У другому завданні проаналізувати індекси GIN, BRIN.

Умова для тригера – before update, delete.

Навігація по звіту

1. [Завдання 1.](#Завдання1)
2. [Завдання 2.](#Завдання2)
3. [Завдання 3.](#Завдання3)
4. [Програмний код.](#Ілюстрації)

**Завдання 1**

Нижче (Рис. 1) наведено логічну модель бази даних:

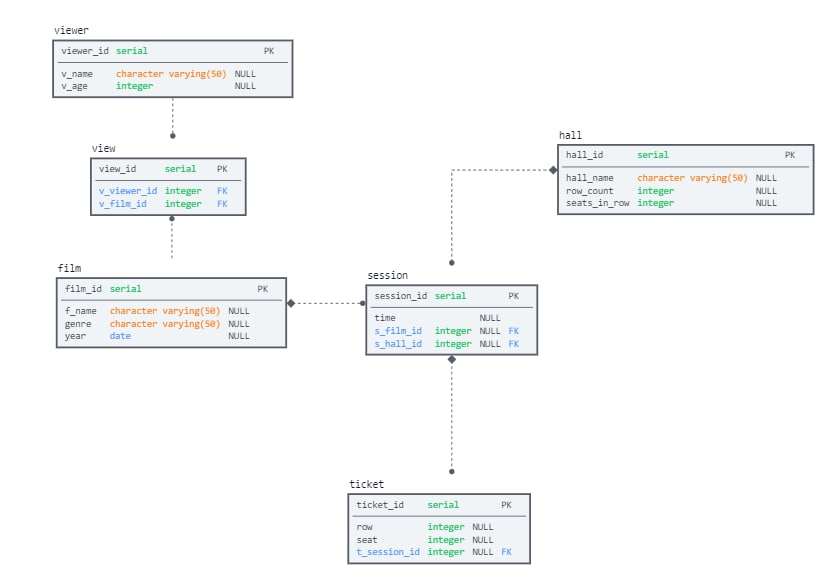
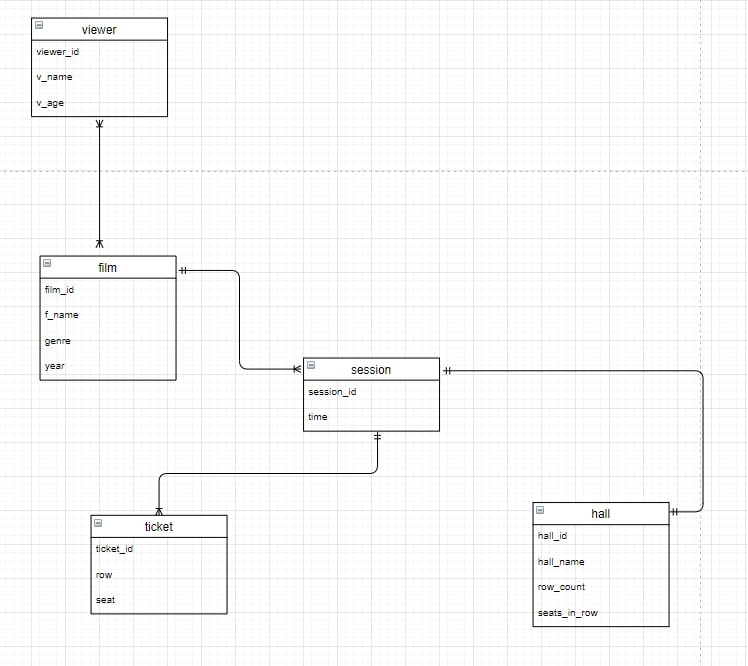


Рис. 1 – Логічна модель бази даних

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

 Зобразимо сутнісні класи програми:

Продемонструємо код лише для одного класу Film:

Класс Film, де за допомогою декораторів бібліотеки TypeORM описується зв’язок класу із таблицею бази даних.

import { Column, Entity, OneToMany, PrimaryGeneratedColumn } from "typeorm";

import { Session } from "./z\_session.js"

import { \_View } from "./z\_view.js"

@Entity({ name: 'film' })

export class Film {

    @PrimaryGeneratedColumn()

    film\_id: number;

    @Column({ type: 'character varying', length: 50, nullable: false})

    f\_name: string;

    @Column({ type: 'character varying', length: 50, nullable: false})

    genre: string;

    @Column({ type: 'date', nullable: false})

    year: string;

    @OneToMany(() => Session, session => session.s\_film\_id)

    session: Session[];

    @OneToMany(() => \_View, view => view.v\_film\_id)

    view: \_View[];

}

Кожний клас таблиці «обслуговує» клас «сервіс», що містить методи для роботи з відповідною таблицею.

Метод для вставки у таблицю:

async add\_data\_film() {

        const film: Film = Reader.prepare\_data\_film();

        try {

            await this.connection.manager.save(film);

            console.log(`Film with id ${film.film\_id} has been saved`);

        } catch (err) {

            console.log(err);

        }

    }

Метод для редагування запису:

async edit\_data\_film() {

        const id: number = +question('Film id: ');

        try {

            const filmRepo: Repository<Film> = getRepository(Film);

            let filmEdit: Film = await filmRepo.findOne({

                where: { film\_id: id }

            });

            if(!filmEdit) {

                console.log(`There is no film with id ${id}`);

            } else {

                const film: Film = Reader.prepare\_data\_film();

                await this.connection

                    .createQueryBuilder()

                    .update(Film)

                    .set({ ...film })

                    .where('film\_id = :id', {id})

                    .execute();

                console.log(`Film with id ${id} has been updated`);

            }

        } catch (err) {

            console.log(err);

        }

    }

Метод для видалення запису:

async delete\_data\_film(){

        const id: number = +question('film id: ');

        try {

            const filmRepo: Repository<Film> = getRepository(Film);

            let filmEdit: Film = await filmRepo.findOne({

                where: { film\_id: id }

            });

            if (!filmEdit) {

                console.log(`There is no film with id ${id}`);

            } else {

                await this.connection

                    .createQueryBuilder()

                    .delete()

                    .from(Film)

                    .where('film\_id = :id', {id})

                    .execute();

                console.log(`Film with id ${id} has been deleted`);

            }

        } catch(err) {

            console.log(err);

        }

    }

Метод для виведення вмісту таблиці:

async show\_film\_data() {

        try {

            const film: Array<Film> = await this.connection.manager.find(Film);

            Printer.print\_film(film);

        } catch (err) {

            console.log(err);

        }

    }

Окрім сервісів, також було створено допоміжні класи, використання методів частини з них(Printer, Reader) можна помітити у сервісах. Продемонструємо методи цих класів для роботи з поточним класом – Film:

Метод класу Reader для зчитування інформації з консолі та формування об’єкту:

 static prepare\_data\_film(): Film {

        const film: Film = new Film();

        film.f\_name = question('film name: ');

        film.genre = question('film genre: ');

        film.year = Format.toDate(new Date(Date.now()));

        return film;

    }

Метод класу Printer для форматування виводу даних із таблиці:

static print\_film(film: Array<Film>) {

        console.log('film\_Id |        f\_name        |     genre     |   year');

        console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

        film.forEach((item: Film) => {

            let filmName: string = '';

            let genre: string = '';

            if (item.f\_name.length > 22) {

                filmName = item.f\_name.substr(0, 19) + '...';

            } else {

                filmName = Format.toField(22, item.f\_name);

            }

            if (item.genre.length > 15) {

                genre = item.genre.substr(0, 12) + '...';

            } else {

                genre = Format.toField(15, item.genre);

            }

            console.log(`${Format.toField(8, item.film\_id.toString())}|${filmName}|${genre}|${Format.toField(12, Format.toDate(new Date(item.year)))}`);

            console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

        });

    }

Клас FormatData для «зручного» форматування окремих полів:

export class Format {

    static toDate(date: Date): string {

        let dd: string = date.getDate().toString();

        let mm: string = (date.getMonth() + 1).toString();

        let yy: string = (date.getFullYear() % 100).toString();

        if (+dd < 10) {

            dd = '0' + dd;

        }

        if (+mm < 10) {

            mm = '0' + mm;

        }

        if (+yy < 10) {

           yy = '0' + yy;

        }

        return dd + '.' + mm + '.' + yy;

    }

    static toField(len: number, str: string): string {

        let newStr: string = str;

        while (newStr.length < len) {

            newStr += ' ';

        }

        return newStr;

    }

}

Інші класи та сервіси реалізовані по аналогії.

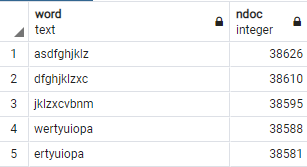
**Завдання 2**

GIN

Для дослідження індексу була створена таблиця test\_gin, яка має дві колонки типу text та tsvector. Колонка з типом tsvector проіндексована як GIN. У таблицю було занесено 1000000 записів.

Знайдемо, які слова зустрічаються найчастіше:

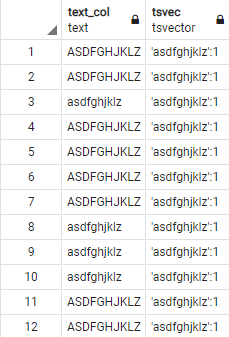




Проведемо пошук по першому слову:



Результат:



Для неіндексованої таблиці:



Для проіндексованої таблиці:



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.

Виконуємо запит для пошуку:



У неіндексованій таблиці:

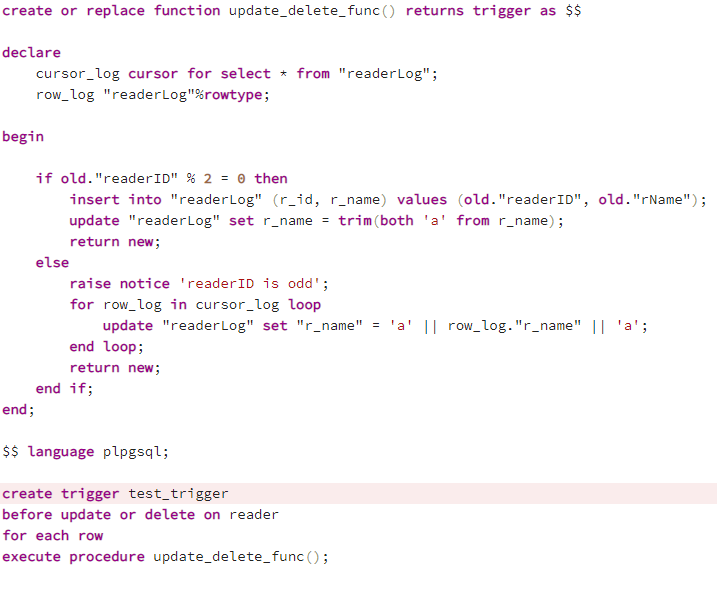


У індексованій таблиці:



**Завдання 3**

Тригер:



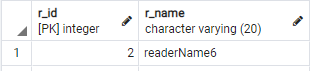
Принцип роботи

Тригер спрацьовує після видалення з таблиці чи при оновлені у таблиці reader. Якщо значення ідентифікатора запису, який видаляється або оновлюється, парне, то цей запис заноситься у додаткову таблицю readerLog. Також, з кожного значення r\_name видаляються символи “х” на початку і кінці. Якщо значення ідентифікатора непарне, то до кожного значення r\_name у таблиці readerLog додається “x” на початку і кінці.

Видаляємо запис із reader:

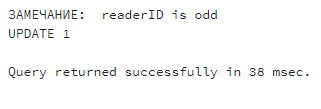


readerLog



Оновлюємо запис:

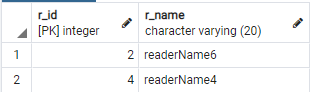




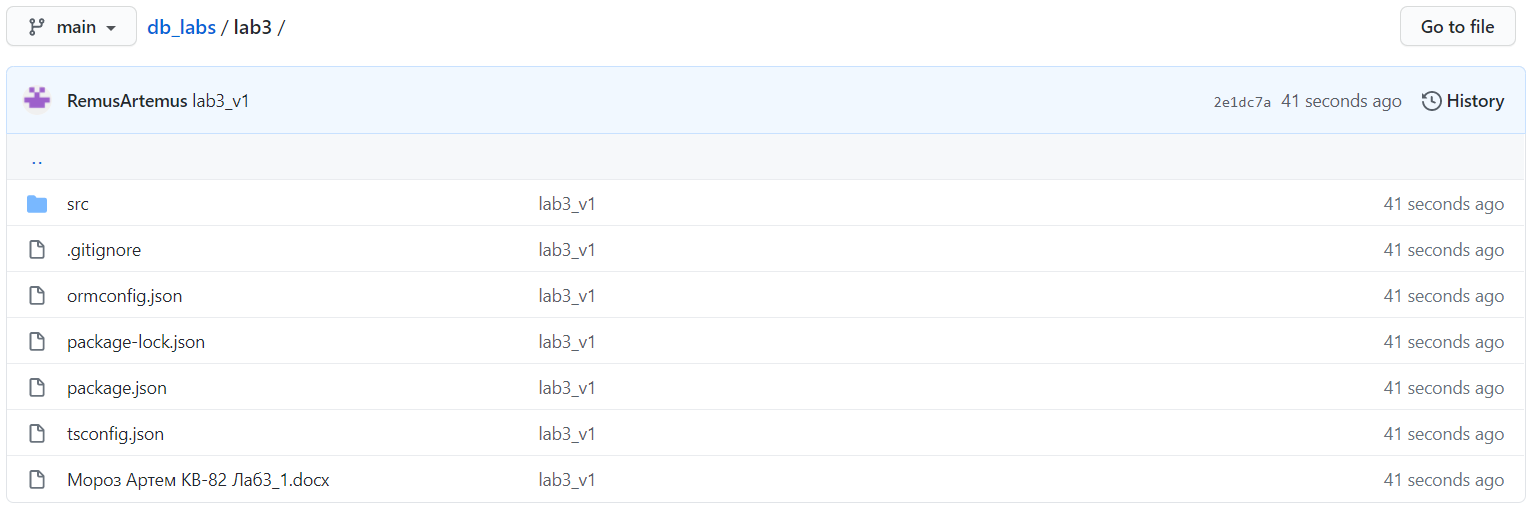
Повторно видаляємо запис:



readerLog



**Ілюстрації програмного коду на Github**



Посилання на Github:

<https://github.com/RemusArtemus/db_labs>