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
Гераймович Д. Ю.
Перевірив:

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.

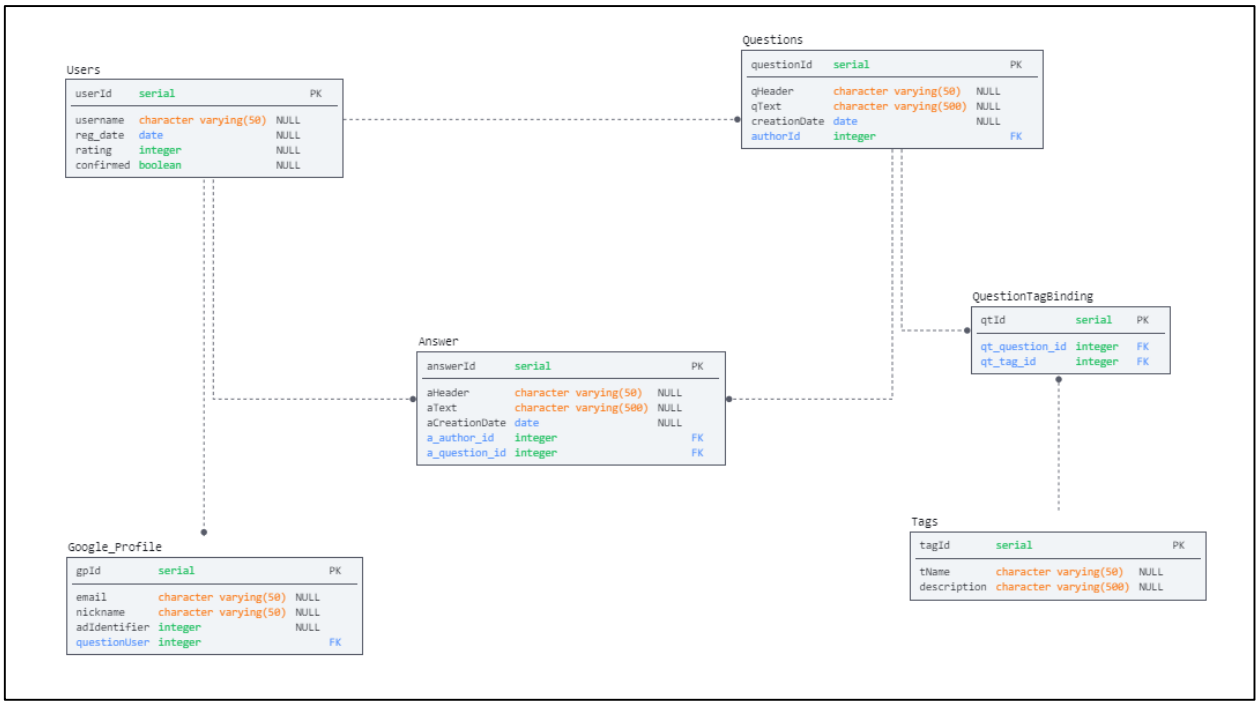
Варіант 3

У другому завданні проаналізувати індекси GIN, Hash.

Умова для тригера – before delete, update.

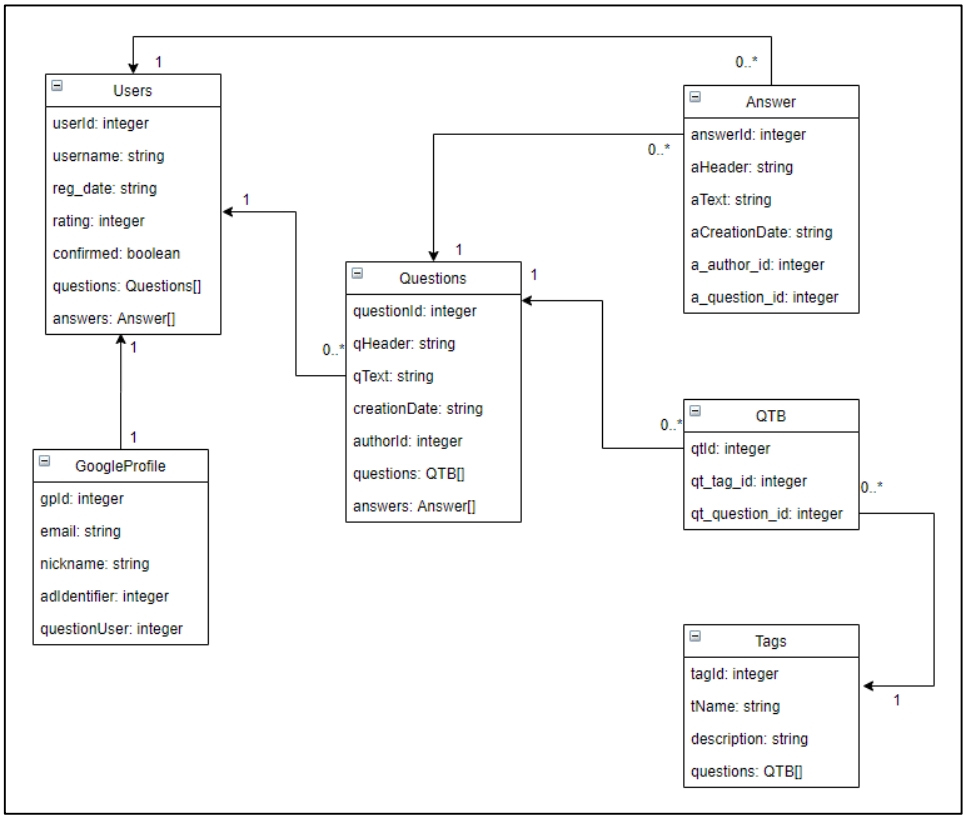
Завдання 1

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



Для перетворення модулів моделей програми, створених в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі було використано бібліотеку TypeORM.

Зобразимо сутнісні класи програми:



Продемонструємо код лише для одного класу Answer:

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

```
import { Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn } from "typeorm";
import { Questions } from "./Questions.js";
import { Users } from "./Users.js";


@Entity({ name: 'Answer' })
export class Answer {

    @PrimaryGeneratedColumn()
    answerId: number;

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

    @Column({ type: 'character varying', length: 500, nullable: false })
    aText: string;

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

    @Column({ type: 'integer', nullable: false })
    @ManyToOne(() => Users, (users: Users) => users.userId)
    @JoinColumn({ name: 'a_author_id' })
    a_author_id: number;

    @Column({ type: 'integer', nullable: false })
    @ManyToOne(() => Questions, (questions: Questions) => questions.questionId)
    @JoinColumn({ name: 'a_question_id' })
    a_question_id: number;
}
```

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

Метод для вставки у таблицю:

```
async addDataAnswer() {
    const answer: Answer = Reader.prepareDataAnswer();

    try {
        await this.connection.manager.save(answer);

        console.log(`Answer with id ${answer.answerId} has been saved`);
    } catch (err) {
        console.log(err);
    }
}
```

Метод для редагування запису:

```
async editDataAnswer() {
    const id: number = +question('Answer id: ');

    try {
        const ansRepo: Repository<Answer> = getRepository(Answer);
        let ansEdit: Answer = await ansRepo.findOne({
            where: {
                answerId: id
            }
        });

        if (!ansEdit) {
            console.log(`There is no answer with id ${id}`);
        } else {
            const answer: Answer = Reader.prepareDataAnswer();
            const userRepo: Repository<Users> = getRepository(Users);
            const questionRepo: Repository<Questions> = getRepository(Questions);

            const userRow: Users = await userRepo.findOne({
                where: { userI: answer.a_author_id }
            });
            const questRow: Questions = await questionRepo.findOne({
                where: { questionId: answer.a_question_id }
            });

            if (!userRow || !questRow) {
                console.log(`There is no user with id ${answer.a_author_id} or question with id ${answer.a_question_id}`);
            } else {
                await this.connection
                    .createQueryBuilder()
                    .update(Answer)
                    .set({ ...answer })
                    .where('answerId = :id', { id })
                    .execute();

                console.log(`Answer with id ${id} has been updated`);
            }
        }
    } catch (err) {
        console.log(err);
    }
}
```

Метод для видалення запису:

```
async deleteDataAnswer() {
    const id: number = +question('Answer id: ');

    try {
        const ansRepo: Repository<Answer> = getRepository(Answer);
        const answer: Answer = await ansRepo.findOne({
            where: {
                answerId: id
            }
        });

        if (!answer) {
            console.log(`There is no answer with id ${id}`);
        } else {
            await this.connection
                .createQueryBuilder()
                .delete()
                .from(Answer)
                .where('answerId = :id', { id })
                .execute();

            console.log(`Answer with id ${id} has been deleted`);
        }
    } catch (err) {
        console.log(err);
    }
}
```

Метод для виведення вмісту таблиці:

```
async showDataAnswer() {
    try {
        const answers: Array<Answer> = await this.connection.manager.find(Answer);

        Printer.printAnswers(answers);
    } 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 – Answer:

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

```
static prepareDataAnswer(): Answer {
    const answer: Answer = new Answer();

    answer.aHeader = question('answer header: ');
    answer.aText = question('answer text: ');
    answer.aCreationDate = Format.toDate(new Date(Date.now()));
    answer.a_author_id = +question('author id: ');
    answer.a_question_id = +question('question id: ');

    return answer;
}
```

Метод класу Printer для форматування виводу даних із таблиці:

```
static printAnswers(answers: Array<Answer>) {
    console.log('answerId |     aHeader     |       aText       | aCreationDate | a_author_id | a_question_id');
    console.log('_____');

    answers.forEach((item: Answer) => {
        let modHeader: string = '';
        let modText: string = '';

        if (item.aHeader.length > 16) {
            modHeader = item.aHeader.substr(0, 13) + '...';
        } else {
            modHeader = Format.toField(16, item.aHeader);
        }

        if (item.aText.length > 19) {
            modText = item.aText.substr(0, 16) + '...';
        } else {
            modText = Format.toField(19, item.aText);
        }

        console.log(`${Format.toField(9, item.answerId.toString())}|${modHeader}|${modText}|${Format.toField(15, Format.toDate(new Date(item.aCreationDate)))}|${Format.toField(13, item.a_author_id.toString())}|${item.a_question_id}`);
        console.log('_____');
    });
}
```

Інші класи та сервіси реалізовані по аналогії.

Завдання 2

GIN

Для дослідження індексу була створена таблиця, яка має дві колонки типу text та одну колонку типу tsvector. У таблицю було занесено 1000000 записів.

Знайдемо слова, які зустрічаються найрідше:

```
select word, ndoc from ts_stat('select text3 from "test_gin"') order by ndoc asc limit 5
```

Результат:

	word text	ndoc integer
1	vbnmqwertyuiopa	18998
2	xcvbnmqwertyuio	19039
3	hjklzxcvbnmqwer	19041
4	xcvbnm	19063
5	dfghjklzxcvbnmq	19083

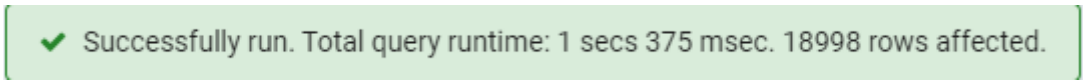
Здійснимо пошук по цих словах.

```
select * from test_gin where text3 @@ to_tsquery('vbnmqwertyuiopa')
```

Результат:

	text1 text	text2 text	text3 tsvector
1	'1':2 'tyui...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
2	'1':2 'rtyu...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
3	'1':2 'dfg...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
4	'1':2 'cvb...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
5	'1':2 'pas...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
6	'1':2 'pas...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
7	'1':2 'ghjk...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
8	'1':2 'ghjk...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
9	'1':2 'ghjk...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
10	'1':2 'xcv...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
11	'1':2 'mq...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...
12	'1':2 'fghj...	'1':1,2 '2':...	'1':1,2 '2':3,4 ...

У неіндексованій таблиці:



У індексованій таблиці:

Hash

Багато сучасних мов програмування включають хеш-таблиці в якості базового типу даних. Зовні це виглядає, як звичайний масив, але в якості індексу використовується не ціле число, а будь-який тип даних (наприклад, рядок). Хеш-індекс в PostgreSQL влаштований схожим чином.

Для дослідження індексу була створена таблиця records, яка має дві колонки: record_id типу integer та rec_text текстового типу. Вони проіндексовані як Hash. У таблицю було занесено 1000000 записів.

Виконаємо запити для пошуку:

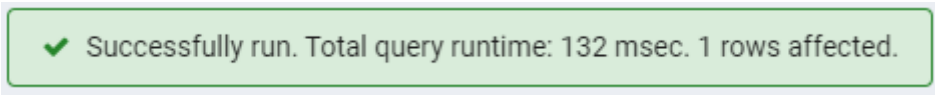
```
select count(*) from records where rec_text like '%qwert%';
select count(*) from records where rec_text like '%qwert%' or rec_text like '%asdf%';
```

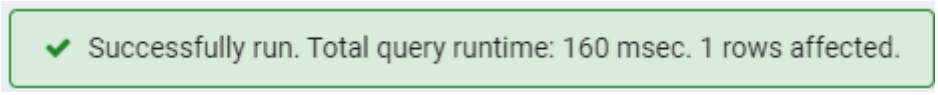
Результат:

	count bigint
1	250704

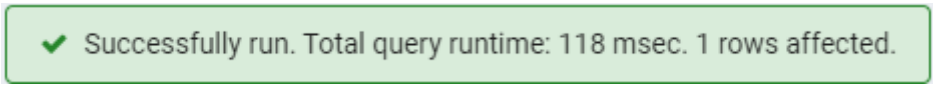
	count bigint
1	162829

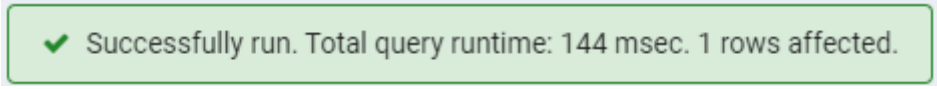
У неіндексованій таблиці:





У індексованій таблиці:





Завдання 3

Тригер:

```
create or replace function delete_update_func() returns trigger as $$

declare
    cursor_log cursor for select * from "authorLog";
    row_log "authorLog"%rowtype;

begin

    if old."authorID" % 2 = 0 then
        insert into "authorLog" ("auName") values (old."aName");
        update "authorLog" set "auName" = trim(both 'A' from "auName");
        return new;
    else
        raise notice 'authorID is odd';
        for row_log in cursor_log loop
            update "authorLog" set "auName" = 'A' || row_log."auName" || 'A';
        end loop;
        return new;
    end if;
end;

$$ language plpgsql;

create trigger test_trigger
before delete or update on author
for each row
execute procedure delete_update_func();
```

Принцип роботи

Тригер спрацьовує перед видаленням з таблиці чи при оновлені у таблиці author. Якщо значення ідентифікатора запису, який видаляється або оновлюється, парне, то цей запис заноситься у додаткову таблицю authorLog. Також, з кожного значення auName видаляються символи “A” на початку і кінці. Якщо значення ідентифікатора непарне, то до кожного значення auName у таблиці authorLog додається “A” на початку і кінці.

Видаляємо запис із author:

```
delete from author where "authorID" = 2;
```

authorLog:

	aId [PK] integer	auName character varying (50)
1	1	authorName2

Оновлюємо запис:

```
update author set "aName" = 'asdasdasd' where "authorID" = 3;
```

```
ЗАМЕЧАНИЕ:  authorID is odd
UPDATE 1
```

Повторно видаляємо запис:

```
delete from author where "authorID" = 4;
```

authorLog:

	aId [PK] integer	auName character varying (50)
1	1	authorName2
2	2	authorName4