



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”

Виконав
студент III курсу
групи KB-84
Антонюк А І

Київ 2020

Засоби оптимізації роботи СУБД PostgreSQL

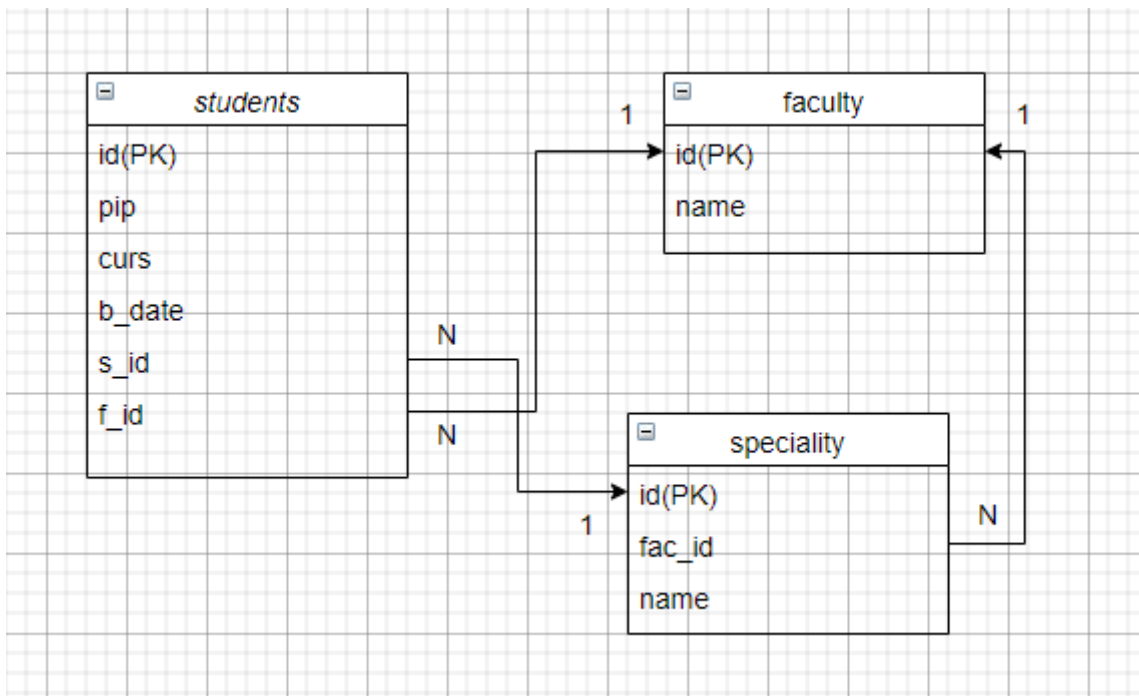
Варіант 2: Університет (Студенти , спеціальності , факультети)

№ варіанта	Види індексів	Умови для тригера
2	Hash, BRIN	after insert, update

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.



Завдання 1

Розглянемо реалізацію даної бази даних за допомогою платформи Node.js і бібліотеки Sequelize. Далі на прикладі розберемо таблицю faculty та її зв'язки. Її сутність в окремому файлі.

```
const { DataTypes } = require('sequelize');

const db = require('../db');

const Faculty = db.define('faculty', {
  name: {
    type: DataTypes.STRING,
  }
}, {
  underscored: true,
  freezeTableName: true,
  timestamps: false,
});

Faculty.associate = models => {
  Faculty.hasMany(models.Students, {
    foreignKey: 'f_id'
  });
  Faculty.hasMany(models.Speciality, {
    foreignKey: 'fac_id'
  });
};

module.exports = Faculty;
```

В цьому файлі ми створили Об'єкт/Клас який описує поля цієї таблиці та її зв'язки. А саме – факультет має багато спеціальностей і студентів, а їх зв'язок зберігає Об'єкт/Клас faculty в іншому файлі.

Тепер, щоб отримати усі факультети потрібно написати:

```
router.get('/faculty', async (req, res) => {
  try {
    const facultys = await Faculty.findAll();
    res.json(facultys);
  } catch € {
    res.status(400).json€;
  }
});
```

Для пошуку за коментаріями:

```
router.post('/faculty/search', async (req, res) => {
  try {
```

```

    const {name} = req.body;
    const facultys = await Faculty.findAll({
      where: {
        name: {
          [Op.iLike]: `%${name}%`
        }
      }
    });
    res.json(facultys);
  } catch € {
    res.status(400).json€;
  }
});

```

Для видалення:

```

router.delete('/faculty/:id', async (req, res) => {
  try {
    const response = await Faculty.destroy({
      where: {
        id: req.params.id
      }
    });
    res.json(response);
  } catch € {
    res.status(400).json€;
  }
});

```

Для створення:

```

router.post('/faculty/new', async (req, res) => {
  try {
    const {name} = req.body;
    const newFaculty = await Faculty.create({
      name
    });
    res.json(newFaculty);
  } catch € {
    res.status(400).json€;
  }
});

```

Для змінення за id:

```

router.put('/faculty/:id', async (req, res) => {
  try {
    console.log(req.params.id, 'body', req.body);

    const {name} = req.body;

```

```

let updatedFaculty = await Faculty.update({
  name: name,
}, {
  where: {
    id: req.params.id
  }
});
if(updatedFaculty) {
  updatedFaculty = await Faculty.findOne({
    where: {
      id: req.params.id
    }
  });
}
res.json(updatedFaculty);
} catch € {
  res.status(400).json€;
}
});



```

Завдання 2

№ варіанта	Види індексів
2	Hash, BRIN

Hash

Для цього завдання створимо 100000+ випадкових рядків в таблиці faculty(id, name).

1	select count(*) from faculty		
<div>Data OutputExplainMessagesNotifications</div>			
	count bigint		
1	110014		

1 `select * from faculty limit 5 offset 200;`

Data Output

Explain

Messages

Notifications

	<div>id</div> <div>[PK] integer</div>	<div>name</div> <div>character varying (255)</div>	
1	221	92l2Sr2bGmfM4Od	
2	222	N92VA3vM3h33xX	
3	223	ZqsTsle5Wru0uxt	
4	224	5dBectq5V8l2fYh	
5	225	OHdweOgMPMTVL3a	

Вибере одне з випадково створених – N92VA3vM3h33xX. Зробимо пошук по цьому імені.

1 explain select * from faculty where "name" = 'N92VA3vM3h33xX';

Data Output

Explain

Messages

Notifications

▲

QUERY PLAN

text

🔒

1

Seq Scan on faculty (cost=0.00..2076.18 rows=1 width=19)

2

Filter: ((name)::text = 'N92VA3vM3h33xX'::text)

Тепер створимо індекс.

1	drop index if exists "name_idx";
2	create index "name_idx" on "faculty" using hash("name");

Data Output

Explain

Messages

Notifications

CREATE INDEX

Query returned successfully in 441 msec.

Виконаємо той же запит. І отримуємо Cost = 8 при Index Scan.

```
1 explain analyze select * from faculty where "name" = 'N92VA3vM3h33xX';
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Index Scan using name_idx on faculty (cost=0.00..8.02 rows=1 width=19) (actual time=0....	
2	Index Cond: ((name)::text = 'N92VA3vM3h33xX'::text)	
3	Planning Time: 1.480 ms	
4	Execution Time: 0.058 ms	

Тепер бачимо, що використання індексу дало значне прискорення при пошуку. Тобто, застосування індексу HASH покращило роботу повнотекстового пошуку.

BRIN

Для цього завдання створимо 100000 випадкових рядків в таблиці students(id, pip, curs, b_date, s_id, f_id).

```
13 select count(*) from students|
```

Data Output Explain Messages Notifications

	count	
	bigint	
1	100000	

```
13 select * from students limit 5 offset 10000
```

Data Output Explain Messages Notifications

	id [PK] integer	pip text	curs integer	b_date timestamp without time zone	s_id integer	f_id integer
1	10001	d89f3a...	4	2020-12-23 12:34:26.283991	3	5
2	10002	9103c8...	2	2020-12-23 12:35:26.283991	4	2
3	10003	f5dffc1...	5	2020-12-23 12:36:26.283991	4	4
4	10004	d78382...	3	2020-12-23 12:37:26.283991	4	5
5	10005	6eb887...	4	2020-12-23 12:38:26.283991	3	1

Виконаємо запит без індексу по стовпчику b_date, оскільки тут данні корелюють із їх фізичним положенням в таблиці. А це є тим випадком, коли BRIN працює ефективно

```
15 explain analyze select * from public.students where b_date between '2020-12-23 12:34:26.283991' and '2020-12-23 12:38:26.283991';
```

Data Output	Explain	Messages	Notifications
QUERY PLAN			
text			
1	Seq Scan on students (cost=0.00..2637.00 rows=5 width=57) (actual time=0.737..8.678 rows=5 loops=1)		
2	Filter: ((b_date >= '2020-12-23 12:34:26.283991'::timestamp without time zone) AND (b_date <= '2020-12-23 12:38:26.283991'::timestamp without time zone))		
3	Rows Removed by Filter: 99995		
4	Planning Time: 0.057 ms		
5	Execution Time: 8.689 ms		

Тепер створимо індекс students_date_idx

```
17 create index students_date_idx on students(b_date);
```

Data Output	Explain	Messages	Notifications
CREATE INDEX			
Query returned successfully in 170 msec.			

Зробимо запит з індексом, отримаємо:

```
15 explain analyze select * from public.students where b_date between '2020-12-23 12:34:26.283991' and '2020-12-23 12:38:26.283991';
```

```
16
```

Data Output	Explain	Messages	Notifications
QUERY PLAN			
text			
1	Index Scan using "students_date_idx" on students (cost=0.29..8.39 rows=5 width=57) (actual time=0.013..0.014 rows=5 loops=1)		
2	Index Cond: ((b_date >= '2020-12-23 12:34:26.283991'::timestamp without time zone) AND (b_date <= '2020-12-23 12:38:26.283991'::timestamp without time zone))		
3	Planning Time: 0.070 ms		
4	Execution Time: 0.025 ms		

Бачим, що пошук з індексом працює набагато швидше.

Завдання 3

№ варіанта	Умови для тригера
2	<i>after insert, update</i>

Команди створення тригера та підключення до таблиці

```
1 create or replace function faculty_trigger() returns trigger as $$
2 begin
3     if (tg_op = 'INSERT') then
4         if (char_length(new.name)<4)or(char_length(new.name)>60) then
5             raise exception 'Wrong string';
6             return null;
7         end if;
8         insert into logs(log, some_time) values(concat('inserted faculty ', new.name), now()::timestamp);
9         raise notice 'Successfull inserted';
10        return new;
11    elsif tg_op = 'UPDATE' then
12        insert into logs(log, some_time) values('upadted row in table faculty', now()::timestamp);
13        raise notice 'Successfull updated';
14        return new;
15    else return null;
16    end if;
17 end;
18 $$language plpgsql;
19
20 create trigger faculty_trigger() after insert or update on public.faculty
21 for each row execute procedure faculty_trigger();
```

Створимо таблицю logs в яку будуть вставлятись записи про те, що відбувся insert or update. Також будемо фіксувати час цього запису.

```
24 create table logs(
25     id serial primary key,
26     log text,
27     some_time timestamp
28 )
```

Виконаємо insert в таблицю faculty

```
30 insert into faculty(name) values ('abcdefghjdjdjdjjdj');
31
32
33
34
```

Data Output Explain Messages Notifications

ПОВІДОМЛЕННЯ: Successfull inserted
INSERT 0 1

Query returned successfully in 52 msec.

Виконаємо update в таблиці faculty

```
34 update faculty set name = 'andriyantoiyk' where id = 21;  
35  
36  
37  
38  
39
```

Data Output Explain Messages Notifications

ПОВІДОМЛЕННЯ: Successfull updated
UPDATE 1

Query returned successfully in 55 msec.

Введемо закоротке ім'я для вставки, отримаємо помилку:

```
38 insert into faculty(name) values ('ab');  
39
```

Data Output Explain Messages Notifications

ERROR: ПОМИЛКА: Wrong string
CONTEXT: Функція PL/pgSQL faculty_trigger() рядок 5 в RAISE

SQL state: P0001

Перевіримо таблицю logs

```
36 select * from logs;  
37
```

Data Output Explain Messages Notifications

	id [PK] integer		log text		some_time timestamp without time zone	
1		1	inserted faculty aaaaaaaaaaaaaa		2020-12-15 21:11:43.146607	
2		2	upadted row in table faculty		2020-12-15 21:14:43.33115	
3		3	inserted faculty hifftttttttttttttt		2020-12-16 11:48:12.9688	
4		4	inserted faculty abcdefghjdjdjdjdj	←	2020-12-16 12:02:20.848281	
5		5	upadted row in table faculty	←	2020-12-16 12:04:38.987514	

Бачимо, що появились відповідні записи після запитів insert та update. Також бачимо, що при вводі некоректного імені запису в таблиці немає. Це говорить про те, що тригер працює правильно.