



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота №3
з дисципліни “Бази даних”
тема “Засоби оптимізації роботи СУБД PostgreSQL”

Виконав
студент II курсу
групи КП-93
Варіант 9
Зверев Костянтин Васильович

Перевірів
“--” “вересня” 2020р.
викладач
Петрашенко Андрій Васильович

Київ 2020

Мета роботи

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Постановка завдання

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

Посилання на репозиторій: <https://github.com/zver-came/lab3>

Варіант: База даних Університет. Заліковка КП-9309

№ варіанта	Види індексів	Умови для тригера
9	<i>BTree, BRIN</i>	<i>before delete, update</i>

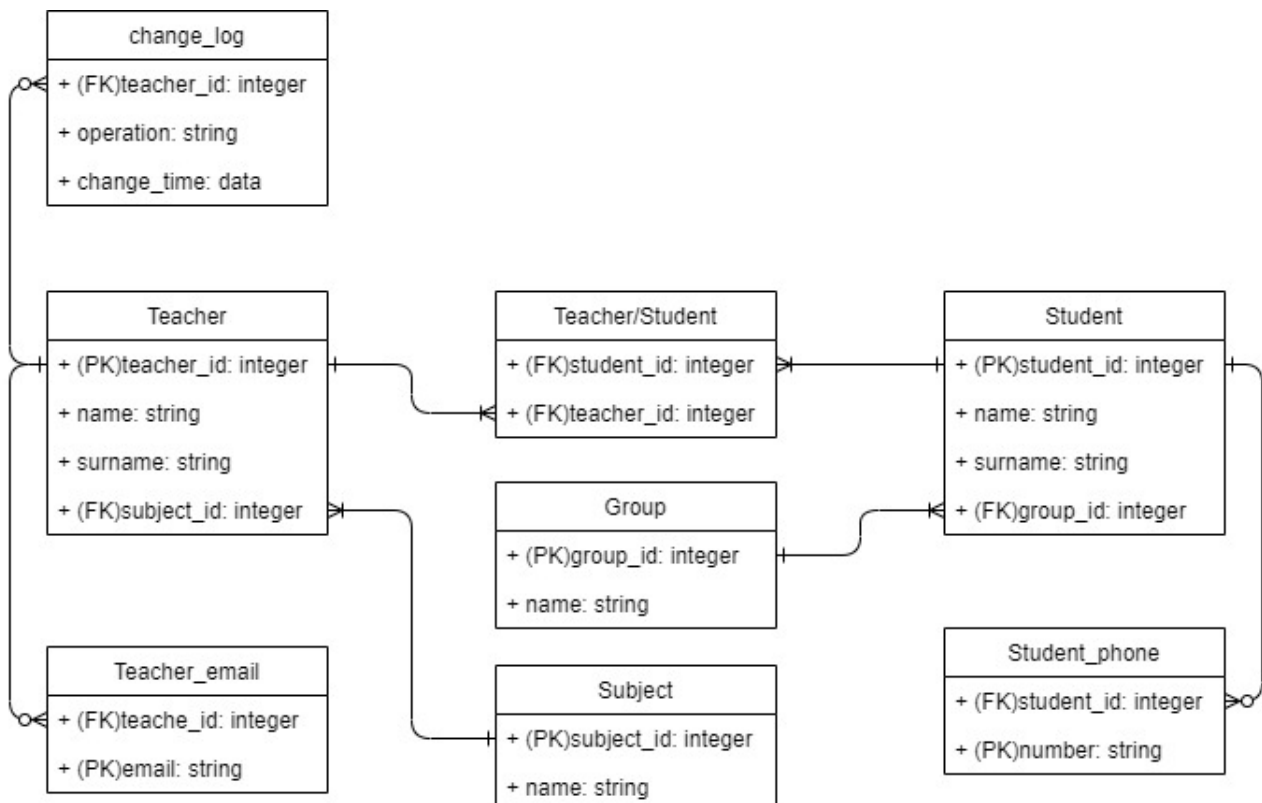


Схема бази даних у вигляді таблиць

Класи ORM, що відповідають таблицям бази даних

Group

```
class Group(Base):
    __tablename__ = 'groups'
    group_id = Column(Integer, autoincrement=True, primary_key=True)
    name = Column('name', String(32))
    students = relationship("Student", back_populates='group', cascade='all, delete, delete-orphan')

    def __init__(self, name):
        self.name = name
```

Student_phone

```
class Student_phone(Base):
    __tablename__ = 'student_phone'
    phone_number = Column('phone_number', String(50), primary_key=True)
    student_id = Column(Integer, ForeignKey('students.student_id'), primary_key=True)
    student = relationship('Student', back_populates='phones')

    def __init__(self, phone_number, student_id):
        self.phone_number = phone_number
        self.student_id = student_id
```

Student

```
class Student(Base):
    __tablename__ = 'students'
    student_id = Column(Integer, autoincrement=True, primary_key=True)
    name = Column('name', String(50))
    surname = Column('surname', String(50))
    group_id = Column(Integer, ForeignKey('groups.group_id'))
    teachers = relationship("Teacher", secondary=teacher_studen_association)
    phones = relationship('Student_phone', back_populates='student', cascade='all, delete, delete-orphan')
    group = relationship('Group', back_populates='students')

    def __init__(self, name, surname, group_id):
        self.name = name
        self.surname = surname
        self.group_id = group_id
```

Subject

```
class Subject(Base):
    __tablename__ = 'subjects'
    name = Column('name', String(50))
    subject_id = Column(Integer, autoincrement=True, primary_key=True)
    teachers = relationship("Teacher", back_populates='subject', cascade='all, delete, delete-orphan')

    def __init__(self, name):
        self.name = name
```

Teacher_student

```

teacher_studen_association = Table(
    'teacher_student', Base.metadata,
    Column('student_id', Integer, ForeignKey('students.student_id')),
    Column('teacher_id', Integer, ForeignKey('teachers.teacher_id'))
)

```

Teacher_email

```

class Teacher_email(Base):
    __tablename__ = 'teacher_email'
    email = Column('email', String(50), primary_key=True)
    teacher_id = Column(Integer, ForeignKey('teachers.teacher_id'), primary_key=True)
    teacher = relationship('Teacher', back_populates='emails')

    def __init__(self, email, teacher_id):
        self.email = email
        self.teacher_id = teacher_id

```

Teacher

```

class Teacher(Base):
    __tablename__ = 'teachers'
    teacher_id = Column(Integer, autoincrement=True, primary_key=True)
    name = Column('name', String(50))
    surname = Column('surname', String(50))
    subject_id = Column(Integer, ForeignKey('subjects.subject_id'))
    students = relationship("Student", secondary=teacher_studen_association)
    emails = relationship("Teacher_email", back_populates='teacher', cascade='all, delete, delete-orphan')
    subject = relationship("Subject", back_populates='teachers')

    def __init__(self, name, surname, subject_id):
        self.name = name
        self.surname = surname
        self.subject_id = subject_id

```

Приклади ORM запитів

```

def select_item(self, type, value):
    try:
        return self.session.query(type).get(value)
    except Exception as exp:
        print('You have search problem. Detail info: %s' % exp)

def add_new_item(self, new_item):
    try:
        self.session.add(new_item)
        self.session.commit()
        return new_item
    except Exception as exp:
        print('You have problem with adding item. Detail info: %s' % exp)

```

```

def delete_item(self, item):
    try:
        self.session.delete(item)
        self.session.commit()
    except Exception as exp:
        print('You have problem with delete item. Detail info: %s'%exp)

def update_item(self):
    try:
        self.session.commit()
    except Exception as exp:
        print('You have problem with update item. Detail info: %s'%exp)

```

```

def add_new_link(self, student_id, teacher_id):
    teacher = self.select_item(orm.Teacher, teacher_id)
    student = self.select_item(orm.Student, student_id)
    try:
        teacher.students.append(student)
        self.session.commit()
        return True
    except: return False

def delete_link(self, student_id, teacher_id):
    teacher = self.select_item(orm.Teacher, teacher_id)
    student = self.select_item(orm.Student, student_id)
    try:
        i = teacher.students.index(student)
        del teacher.students[i]
        self.session.commit()
        return True
    except: return False

```

Та їх використання в програмі

1. Створення нової сутності Вчителя

```

1. Find teacher
2. Add new teacher
3. Delete teacher
4. Update teacher
5. Work with teacher email
6. Get all teacher students
7. Work with subject menu
8. Work with student menu
9. Open main menu

```

```

Enter command: 2
Enter teacher params:
Name: Oleg
Surname: Olegovich

```

```

-----
Subject params
-> name
-> id
-> skip
Enter params: id
Enter subject id: 124

```

```

-----
Subject params
-> name
-> id
-> skip
Enter params: skip

```

```

-----
(124, 'd')

```

```

-----
Enter id: 124

```

```

-----
Teacher successfully added with id -> 5320
-----

```

Додавання зв'язку

1. Find student
2. Add new student
3. Delete student
4. Update student
5. Get all student subjects
6. Get all student teachers
7. Add student teacher
8. Delete student teacher
9. Work with student phone numbers
10. Work with group menu
11. Work with teachers menu
12. Open main menu

Enter command: 7

Student params:

-> name

-> surname

-> id

-> phone

-> group

-> skip

Enter id: 285405

Teacher params

-> name

-> surname

-> id

-> email

-> subject

-> skip

Enter params: id

Enter teacher id: 5320

Enter id: 5320

Teacher <-> Student link successfully added with teacher id -> 5320 and student id -> 285405

Завдання 2

Приклади запитів до та після застосування індексів

Команди створення індексів:

Btree

```
create index btree_student_index on students using btree(name,student_id)
```

Brin

```
create index brin_student_index on students using brin(student_id)
```

Без використання btree index

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 explain analyze select * from students where name = 'FYQMRK'
2 and student_id > 90000
3
```

4 **Результат** План выполнения Сообщения

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..4660.20 rows=1 width=22) (actual time=22.511..45.199 rows=1 loops=1)	
2	Workers Planned: 1	
3	Workers Launched: 1	
4	-> Parallel Seq Scan on students (cost=0.00..3660.10 rows=1 width=22) (actual time=10.894..10.895 rows=1 loops=2)	
5	Filter: ((student_id > 90000) AND ((name)::text = 'FYQMRK'::text))	
6	Rows Removed by Filter: 117985	
7	Planning Time: 0.096 ms	
8	Execution Time: 45.216 ms	

Використовуючи btree index

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 explain analyze select * from students where name='FYQMRK'
2 and student_id > 90000
3
```

4 **Результат** План выполнения Сообщения

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on students (cost=4.43..8.45 rows=1 width=22) (actual time=0.045..0.046 rows=1 loops=1)	
2	Recheck Cond: (((name)::text = 'FYQMRK'::text) AND (student_id > 90000))	
3	Heap Blocks: exact=1	
4	-> Bitmap Index Scan on btree_student_index (cost=0.00..4.43 rows=1 width=0) (actual time=0.037..0.037 rows=1 loops=1)	
5	Index Cond: (((name)::text = 'FYQMRK'::text) AND (student_id > 90000))	
6	Planning Time: 0.173 ms	
7	Execution Time: 0.078 ms	

Без використання btree index

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```

1  explain analyze select * from students order by name
2

```

Результат

План выполнения

Сообщения

	<div> <div>QUERY PLAN</div> <div>text</div> </div>
1	Sort (cost=29836.56..30426.48 rows=235971 width=22) (actual time=1041.020..1291.124 rows=235971 loops=1)
2	Sort Key: name
3	Sort Method: external merge Disk: 7992kB
4	-> Seq Scan on students (cost=0.00..3937.71 rows=235971 width=22) (actual time=0.018..14.483 rows=235971 loops=1)
5	Planning Time: 0.095 ms
6	Execution Time: 1298.477 ms

Використовуючи btree index

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```

1  explain analyze select * from students order by name
2

```

Результат

План выполнения

Сообщения

	<div> <div>QUERY PLAN</div> <div>text</div> </div>
1	Index Scan using btree_student_index on students (cost=0.42..13419.79 rows=235971 width=22) (actual time=0.011..99.242 rows=235971 lo...
2	Planning Time: 0.059 ms
3	Execution Time: 104.725 ms

Без використання brin index

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```

1  explain analyze select name,student_id from students where student_id<55000
2  group by name, student_id
3

```

Результат

План выполнения

Сообщения

	<div> <div>QUERY PLAN</div> <div>text</div> </div>
1	Group (cost=0.42..178.96 rows=4327 width=11) (actual time=0.026..1.827 rows=4484 loops=1)
2	Group Key: student_id
3	-> Index Scan using students_pkey on students (cost=0.42..168.14 rows=4327 width=11) (actual time=0.021..0.941 rows=4484 loops=1)
4	Index Cond: (student_id < 55000)
5	Planning Time: 0.238 ms
6	Execution Time: 1.978 ms

Використовуючи brin index

Query Editor

История запросов

1

explain analyze select name,student_id from students where student_id<55000

2

group by name, student_id

3

4

Результат

План выполнения

Сообщения

5

6

QUERY PLAN

text

1

HashAggregate (cost=1828.83..1872.10 rows=4327 width=11) (actual time=2.142..2.614 rows=4484 loops=1)

2

Group Key: student_id

3

-> Bitmap Heap Scan on students (cost=13.11..1818.01 rows=4327 width=11) (actual time=0.020..1.394 rows=4484 loops=1)

4

Recheck Cond: (student_id < 55000)

5

Rows Removed by Index Recheck: 14673

6

Heap Blocks: lossy=128

7

-> Bitmap Index Scan on brin_student_index (cost=0.00..12.03 rows=18152 width=0) (actual time=0.013..0.013 rows=1280 loops=1)

8

Index Cond: (student_id < 55000)

9

Planning Time: 0.101 ms

10

Execution Time: 2.855 ms

У даних випадках індекси прискорюють виконання запиту, оскільки були обрані раціональні поля для індексування. Для Btree раціонально використовувати поля з великим «розкидом» значень для зручної побудови бінарного дерева, особливістю якого є константний час пошуку елемента у гілках. Недоречно його використовувати його при малої кількості даних та упорядкованих даних. Натомість BRIN створює «сторінки» значень де помічає мінімальне та максимальне значення на сторінці, що пришвидчує пошук відсортованих даних, та даних де можна чітко виділити діапазон на мін-макс. Недоречно використовувати якщо дані мають розкиданий діапазон значень.

Завдання 3

Текст тригерної функції before delete/update

```

database_lab1/postgres@PostgreSQL 11

Query Editor  История запросов

1 create or replace function check_teacher_info() returns trigger as $mys$
2 declare
3 temail varchar(50);
4 tlink int;
5 begin
6     if(TG_OP='UPDATE') then
7         if new.name='' or new.surname='' then
8             raise exception 'teacher with id % have trable with personal info, empty value',old.teacher_id;
9         else
10            insert into teacher_change_log(teacher_id, change_date,operation) values(old.teacher_id,now(),'update');
11        end if;
12        return new;
13    else
14        for temail in select email from teacher_email where teacher_id=old.teacher_id
15        loop
16            delete from teacher_email where teacher_id=old.teacher_id and email=temail;
17            raise notice 'email %s for teacher with teacher id %s successfully deleted',temail,old.teacher_id;
18        end loop;
19        for tlink in select student_id from teacher_student where teacher_id=old.teacher_id
20        loop
21            delete from teacher_student where teacher_id=old.teacher_id and student_id=tlink;
22            raise notice 'link for teacher with id % and student with id % successfully deleted',old.teacher_id,tlink;
23        end loop;
24        insert into teacher_change_log(teacher_id, change_date,operation) values(old.teacher_id,now(),'delete');
25    end if;
26    return old;
27 end;
28 $mys$ language plpgsql;

```

Даний тригер при виконанні операції оновлення перевіряє чи випадково не були задані пусті поля та повідомляє про це, а при видаленні підчищає зовнішні ключі що унеможливорює виникнення помилок при видаленні об'єкта з таблиці вчителі, також даний тригер заносить інформацію про зміни в спеціальну таблицю так званий журнал змін. Далі наведено приклади виведення результатів після спрацювання тригера.

Початкові дані

database_lab1/postgres@PostgreSQL 11				database_lab1/postgres@PostgreSQL 11			
Query Editor История запросов				Query Editor История запросов			
<pre>1 select * from teacher_email where teacher_id=5319</pre>				<pre>1 select * from teacher_student where teacher_id=5319</pre>			
Результат	План выполнения	Сообщения		Результат	План выполнения	Сообщения	
email [PK] character varying (50)	teacher_id [PK] integer			teacher_id [PK] integer	student_id [PK] integer		
1 kostya_email	5319			1	5319	112000	
				2	5319	112001	
				3	5319	112002	
				4	5319	112003	
				5	5319	112004	
				6	5319	112005	

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 select * from teachers where teacher_id=5319
2
3
```

Результат План выполнения Сообщения

	teacher_id [PK] integer	name character varying (50)	surname character varying (50)	subject_id integer
1	5319	kostya	zverev	124

Виконаємо операцію оновлення

Введемо одне з полів пустим для спрацювання виняткової ситуації

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 update teachers set name='',surname='Zverev' where teacher_id=5319
2
3
```

Результат План выполнения Сообщения

ERROR: ОШИБКА: teacher with id 5319 have trable with personal info, empty value
CONTEXT: функция PL/pgSQL check_teacher_info(), строка 8, оператор RAISE

Введемо правильні поля

database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 update teachers set name='Kostya',surname='Zverev' where teacher_id=5319
2
3
```

Результат План выполнения Сообщения

UPDATE 1

Покажемо зміни в основній таблиці

Query Editor История запросов

```
1 select * from teachers where teacher_id=5319
```

2

3

4 Результат План выполнения Сообщения

	teacher_id integer	name character varying (50)	surname character varying (50)	subject_id integer
1	5319	Kostya	Zverev	124

Відповідний запис в журналі змін

Query Editor История запросов

```
1 update teachers set name='Kostya',surname='Zverev' where teacher_id=5319
2 select * from teacher_change_log where teacher_id=5319
3
```

Результат План выполнения Сообщения

	change_date timestamp with time zone	operation character varying (50)	teacher_id integer
1	2020-11-29 20:45:24.119129+00	update	5319

Виконаємо операцію видалення

Query Editor История запросов

```
1 delete from teachers where teacher_id=5319
2 select * from teacher_change_log where teacher_id=5319
3
```

Результат План выполнения Сообщения

ЗАМЕЧАНИЕ: email kostya_emails for teacher with teacher id 5319s successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112000 successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112001 successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112002 successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112003 successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112004 successfully deleted
 ЗАМЕЧАНИЕ: link for teacher with id 5319 and student with id 112005 successfully deleted
 DELETE 1

Запрос завершён успешно, время выполнения: 56 msec.


Відсутність даного запису в таблицях

Query Editor История запросов

```
1 select * from teachers where teacher_id=5319
2
```

Результат План выполнения Сообщения

	teacher_id [PK] integer	name character varying (50)	surname character varying (50)	subject_id integer


 database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 select * from teacher_email where teacher_id=5319
2
3
```

Результат План выполнения Сообщения

	email [PK] character varying (50)	teacher_id [PK] integer

 database_lab1/postgres@PostgreSQL 11

Query Editor История запросов

```
1 select * from teacher_student where teacher_id=5319
2
3
```

Результат План выполнения Сообщения

	teacher_id [PK] integer	student_id [PK] integer

Відповідний запис в журналі змін

Query Editor История запросов

```
1 delete from teachers where teacher_id=5319
2 select * from teacher_change_log where teacher_id=5319
3
```

Результат План выполнения Сообщения

	change_date timestamp with time zone	operation character varying (50)	teacher_id integer
1	2020-11-29 20:45:24.119129+00	update	5319
2	2020-11-29 20:47:26.095113+00	delete	5319

Контрольні питання:

- 1) ORM – технологія, призначена для взаємодії з БД на об'єктно-орієнтованому рівні, за допомогою якої елементи БД та взаємодію з ними можна описати у вигляді спеціальних класів та об'єктів. Також допомагає уникнути написання SQL-коду на стороні клієнтської програми.
- 2) Для Btree раціонально використовувати поля з великим «розкидом» значень для зручної побудови бінарного дерева, особливістю якого є константний час пошуку елемента у гілках. Недоречно його використовувати його при малій кількості даних та упорядкованих даних. Натомість BRIN створює «сторінки» значень де помічає мінімальне та максимальне значення на сторінці, що пришвидчує пошук відсортованих даних, та даних де можна чітко виділити діапазон на мін-макс. Недоречно використовувати якщо дані неможливо укласти у сторінки(розкиданий діапазон значень).
GIN індексує не атомарні дані, а ті, які складаються із декількох елементів. Індексуються окремі елементи. Доречно використовувати його, наприклад у повнотекстовому пошуку
Nash – хеш-таблиця , яка використовує функції хешування для створення індексу, який відповідає індексованому значенню.
Доречно використовувати при пошуку та порівнянні великих значень
- 3) Функції допомагають винести певний функціонал у об'єкт, для подальшого використання у запитах. Тригери допомагають перекласти обов'язки з контролю даних, виконання додаткових маніпуляцій з даними та обробки виняткових ситуацій при виконанні CRUD операцій на сторону SQL-сервера.

Висновки:

У даній лабораторній роботі було виконано ознайомлення із технологією ORM, створення БД на основі неї, та виконання CRUD операцій. Відбулося ознайомлення із індексуванням БД, та реалізованими індексами у PostgreSQL таких як BRIN, BTree тощо. Також були створені тригери, та функції мовою SQL для обробки виконання запитів.