



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(ла)
студент(ка) 2 курсу
групи КП-91

Лопаткіна Катерина Олегівна
(прізвище, ім'я, по батькові)

варіант Школа

(предмети, вчителі, оцінки, учні,
група)

Перевірів
“ ____ ” “ ____ ” 20__ р.
викладач

Петрашенко Андрій Васильович
(прізвище, ім'я, по батькові)

Київ 2020

Мета

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL

Загальні вимоги до завдання

Загальне 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.

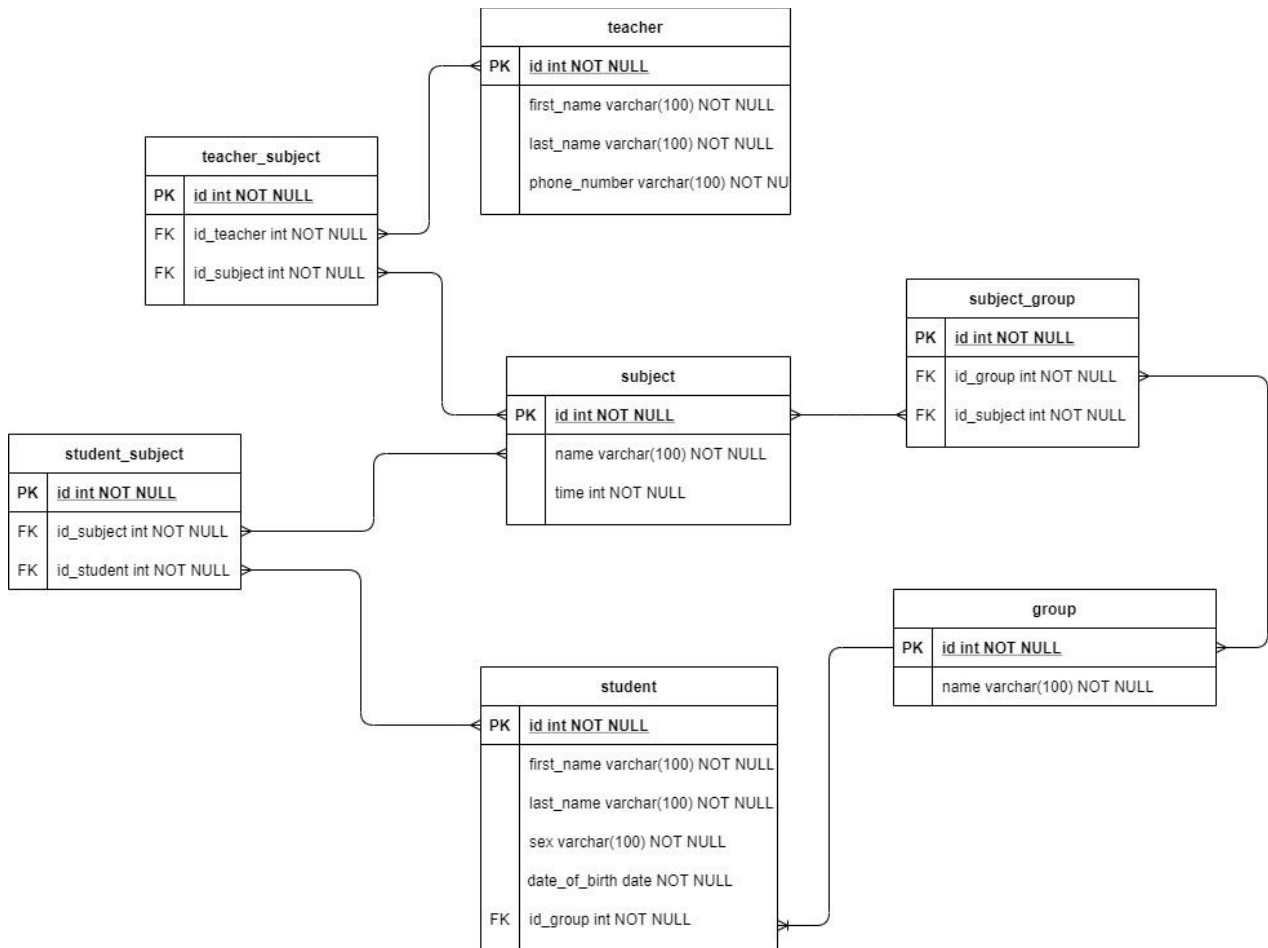
Варіант №15

Репозиторій :

<https://github.com/katerynaLopatkina/Database/tree/main/lab3>

Пункт 1

Схема бази даних:



Класи ORM:

```
class Group(Base):
    __tablename__ = 'group'
    id = Column(Integer, primary_key=True)
    name = Column(String)

    students = relationship("Student", cascade="all, delete", passive_deletes=True)

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

    def __repr__(self):
        return "<Group(id='{id}', name='{name}')>" \
            .format(self.id, self.name)
```

```

student_subject_links = Table('student_subject', Base.metadata,
                               Column('id_student', Integer, ForeignKey('student.id', ondelete="CASCADE")),
                               Column('id_subject', Integer, ForeignKey('subject.id', ondelete="CASCADE")),
                               UniqueConstraint('id_student', 'id_subject', name='unique_student_subject')
                               )

class Student(Base):
    __tablename__ = 'student'
    id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    sex = Column(String)
    date_of_birth = Column(Date)
    id_group = Column(Integer, ForeignKey('group.id', ondelete="CASCADE"))

    def __init__(self, id, first_name, last_name, sex, date_of_birth, id_group):
        self.id = id
        self.first_name = first_name
        self.last_name = last_name
        self.sex = sex
        self.date_of_birth = date_of_birth
        self.id_group = id_group

    def __repr__(self):
        return "<Student(id={}, first_name='{}', last_name='{}', sex={}, date_of_birth='{}', id_group={})>" \
            .format(self.id, self.first_name, self.last_name, self.sex, self.date_of_birth, self.id_group)

```

```

subject_group_links = Table('subject_group', Base.metadata,
                              Column('id_group', Integer, ForeignKey('group.id', ondelete="CASCADE")),
                              Column('id_subject', Integer, ForeignKey('subject.id', ondelete="CASCADE")),
                              UniqueConstraint('id_group', 'id_subject', name='unique_group_subject')
                              )

class Subject(Base):
    __tablename__ = 'subject'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    time = Column(Integer)

    students = relationship("Student", secondary=student_subject_links, backref="subjects")
    teachers = relationship("Teacher", secondary=teacher_subject_links, backref="subjects")
    groups = relationship("Group", secondary=subject_group_links, backref="subjects")

    def __init__(self, id, name, time):
        self.id = id
        self.name = name
        self.time = time

    def __repr__(self):
        return "<Subject(id={}, name='{}', time='{}')>" \
            .format(self.id, self.name, self.time)

```

```

teacher_subject_links = Table('teacher_subject', Base.metadata,
                               Column('id_teacher', Integer, ForeignKey('teacher.id', ondelete="CASCADE")),
                               Column('id_subject', Integer, ForeignKey('subject.id', ondelete="CASCADE")),
                               UniqueConstraint('id_teacher', 'id_subject', name='unique_teacher_subject')
                               )

class Teacher(Base):
    __tablename__ = 'teacher'
    id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    phone_number = Column(String)

    def __init__(self, id, first_name, last_name, phone_number):
        self.id = id
        self.first_name = first_name
        self.last_name = last_name
        self.phone_number = phone_number

    def __repr__(self):
        return "<Teacher(id={}, first_name='{}', last_name='{}', phone_number='{}')>" \
            .format(self.id, self.first_name, self.last_name, self.phone_number)

```

ORM запити:

```

def get_one(self, id_q, table_name):
    return self.session.query(get_table(table_name)).filter_by(id=id_q).first()

def get_one_by_ids(self, id1, id2, table_name1):

    if table_name1 == SUBJECT_GROUP_TABLE:
    elif table_name1 == STUDENT_SUBJECT_TABLE:

    elif table_name1 == TEACHER_SUBJECT_TABLE:
        res = self.session.query(teacher_subject_links) \
            .filter(and_(teacher_subject_links.c.id_teacher == id2,
                        teacher_subject_links.c.id_subject == id1)).first()
        if res is not None:
            s = Subject(res.id_subject, None, None)
            subjects = [s]
            teacher = Teacher(res.id_teacher, None, None, None)
            teacher.subjects = subjects
            return [teacher]
        else:
            return [None]

```

```

def get_many(self, table_name):
    return self.session.query(get_table(table_name)).all()

# deleting data from tables
def delete_one(self, id, table_name):
    req = self.session.query(get_table(table_name)).get(id)
    self.session.delete(req)
    self.session.commit()
    return req

```

```

def delete_by_ids(self, id1, id2, table_name):
    if table_name == TEACHER_SUBJECT_TABLE:
        req = teacher_subject_links.delete() \
            .where(and_(teacher_subject_links.c.id_subject == id1,
                        teacher_subject_links.c.id_teacher == id2))
        res = self.session.execute(req)
        self.session.commit()
        return res
    elif table_name == STUDENT_SUBJECT_TABLE:
        req = student_subject_links.delete() \
            .where(and_(student_subject_links.c.id_subject == id1,
                        student_subject_links.c.id_student == id2))
        res = self.session.execute(req)
        self.session.commit()
        return res
    elif table_name == SUBJECT_GROUP_TABLE:
        req = subject_group_links.delete() \
            .where(and_(subject_group_links.c.id_subject == id1,
                        subject_group_links.c.id_group == id2))
        res = self.session.execute(req)
        self.session.commit()
        return res

# inserting data to table
def add_row(self, data_to_add, table):
    data = self.array_to_model_data(data_to_add, table)
    if data is not None:
        self.session.add(data)
        self.session.commit()
    return data

```



```
def update_row(self, data, table):

    if table == SUBJECT_GROUP_TABLE:
        req = subject_group_links.update() \
            .where(and_(subject_group_links.c.id_subject == data[1],
                        subject_group_links.c.id_group == data[0])) \
            .values(id_subject=data[3], id_group=data[2])
        self.session.execute(req)
        self.session.commit()
        return self.get_one_by_ids(data[3], data[2], table)[0]
    elif table == TEACHER_SUBJECT_TABLE:
        req = teacher_subject_links.update() \
            .where(and_(teacher_subject_links.c.id_subject == data[1],
                        teacher_subject_links.c.id_teacher == data[0])) \
            .values(id_subject=data[3], id_teacher=data[2])
        self.session.execute(req)
        self.session.commit()
        return self.get_one_by_ids(data[3], data[2], table)[0]
    elif table == STUDENT_SUBJECT_TABLE:
        req = student_subject_links.update() \
            .where(and_(student_subject_links.c.id_subject == data[1],
                        student_subject_links.c.id_student == data[0])) \
            .values(id_subject=data[3], id_student=data[2])
        self.session.execute(req)
        self.session.commit()
        return self.get_one_by_ids(data[3], data[2], table)[0]
    else:
        self.session.query(get_table(table)) \
            .filter(get_table(table).id == data.id) \
            .update(
                get_data_table(table, data)
            )
        self.session.commit()
        item = self.get_one(data.id, table)
        return item
```

Пунт 2

BRIN

Візьмемо таблицю student. Перед використанням BRIN потрібно дізнатися для якого стовпця буде доречніше його створити. Для цього ми зробимо запит до pg_stats по нашій таблиці. Нам виведе ім'я стовпців і їх кореляцію. Обираємо стовпець значення якого найблище до одиниці.

```
select attname, correlation from pg_stats where tablename='student'
order by correlation desc nulls last
```

Data Output

	attname name	correlation real
1	id	0.9999921
2	sex	0.49789345
3	last_name	0.27797768
4	first_name	0.031363364
5	id_group	0.0010819797
6	date_of_birth	-0.009991625

Найкращим варіантом буде вибрати стовпець last_name, стовпець sex не будемо брати до уваги, тому що в нього може бути лише два значення Male/Female.

Індекс створюємо наступною командою :

Query Editor

```
1 create index brin_index on student using brin(last_name)
```


1. Вибрати всіх студентів в прізвищі яких є 'ah' і відсортувати їх за спаданням по стовпцю id.

```
1 select * from student where last_name like '%ah%'
2 order by student.id asc
```

Data Output

	id [PK] integer	first_name character varying (100)	last_name character varying (100)	sex character varying (100)	id_group integer	date_of_birth date
1	4	Katya1.756977911094836	Lahno	Female	4813	2002-03-17
2	6	Andrey3.6006241743323244	Lahno	Female	4613	2018-01-09
3	7	Andrey3.1720410799247922	Lahno	Male	2680	2016-05-05
4	8	Katya1.9122739955014794	Lahno	Female	3439	2004-11-20
5	12	Andrey6.3999090075375165	Lahno	Female	4933	2009-01-27
6	19	Olya5	Lahno	Female	789	2006-08-24
7	20	Andrev7	Lahno	Female	4808	2019-09-05

```
1 explain analyze
2 select * from student where last_name like '%ah%'
3 order by student.id asc
```

Data Output

	QUERY PLAN text
1	Index Scan using student_pkey on student (cost=0.42..5576.30 rows=50587 width=35) (actual time=0.015..42.235 rows=50413 loops=1)
2	Filter: ((last_name)::text ~~ '%ah%':text)
3	Rows Removed by Filter: 100608
4	Planning Time: 0.082 ms
5	Execution Time: 43.336 ms

Час виконання запиту до створення індексу BIN був майже такий самий.

2. Тепер виведемо кількість студентів і їхні прізвища тих у кого є 'ah'.

Query Error

```
1 select last_name, count(*) from student where last_name like '%ah%'
2 group by last_name
```

Data Output

	last_name character varying (100)	count bigint
1	Lahno	50413

Query Editor

```
1 explain analyze
2 select last_name, count(*) from student where last_name like '%ah%'
3 group by last_name
4
5 |
```

Data Output

	QUERY PLAN
	text
1	HashAggregate (cost=3402.70..3402.74 rows=4 width=15) (actual time=28.230..28.231 rows=1 loops=1)
2	Group Key: last_name
3	-> Seq Scan on student (cost=0.00..3149.76 rows=50587 width=7) (actual time=0.030..19.918 rows=50413 loops=1)
4	Filter: ((last_name)::text ~~ '%ah%':text)
5	Rows Removed by Filter: 100608
6	Planning Time: 0.129 ms
7	Execution Time: 28.261 ms

В даному випадку час виконання запиту скоротився на 4 мілісекунди.

В нашому випадку використання BIN не дуже й допомогло нам. Можливо через те, що для індекса був обраний стовпець не з дуже високим значенням кореляції. Отже можна зробити висновок що GIN не завжди доцільно використовувати, його більше доцільно використовувати на великих базах даних і на стовпцях які більше менш відсортовані, наприклад на стовпці який зберігає дату. Хоча даний індекс не дуже тяжкий, тому його використання не повинно замедлити виконання запитів, BRIN це той випадок коли індексів можна створити багато “на всякий випадок”.

Hash

Hash індексування найбільш ефективно при використанні оператора “=” при створенні запиту, тому ми візьмемо таблицю teacher і будемо робити індекс для first_name стовпця.

Створюємо індекс:

```
create index first_name_index on teacher using hash(first_name)
```

Messages

CREATE INDEX

Query returned successfully in 259 msec.

Зробимо простий запит з використанням оператора “=”

```
select * from teacher where first_name = 'Dmytro0'
```

Data Output

	id [PK] integer	first_name character varying (100)	last_name character varying (100)	phone_number character varying (13)
1	1	Dmytro0	Melnik	380384244213
2	18	Dmytro0	Malyutin	380958270881
3	2680	Dmytro0	Nazarenko	380860660644
4	51477	Dmytro0	Melnik	380357251643

```
1 explain analyze
2 select * from teacher where first_name = 'Dmytro0'
```

Data Output

	QUERY PLAN text
1	Seq Scan on teacher (cost=0.00..2103.99 rows=1 width=35) (actual time=0.013..10.313 rows=4 loops=1)
2	Filter: ((first_name)::text = 'Dmytro0'::text)
3	Rows Removed by Filter: 99996
4	Planning Time: 0.074 ms
5	Execution Time: 10.325 ms

Після створення індексу


```
select * from teacher where first_name = 'Dmytro0'
```

Data Output

	id [PK] integer	first_name character varying (100)	last_name character varying (100)	phone_number character varying (13)
1	51477	Dmytro0	Melnik	380357251643
2	2680	Dmytro0	Nazarenko	380860660644
3	18	Dmytro0	Malyutin	380958270881
4	1	Dmytro0	Melnik	380384244213

```
explain analyze
select * from teacher where first_name = 'Dmytro0'
```

Data Output

	QUERY PLAN	
	text	
1	Index Scan using first_name_index on teacher (cost=0.00..8.02 rows=1 width=35) (actual time=0.010..0.014 rows=4 loops=1)	
2	Index Cond: ((first_name)::text = 'Dmytro0'::text)	
3	Planning Time: 0.068 ms	
4	Execution Time: 0.024 ms	

Як бачимо швидкодія запиту збільшилась майже на 99%.

Але для запиту з використанням оператора like Hash індекс не використовується:

```
explain analyze
select * from teacher where first_name like 'Dmytro0'
```

Data Output

	QUERY PLAN	
	text	
1	Seq Scan on teacher (cost=0.00..2092.00 rows=1 width=35) (actual time=0.053..12.248 rows=4 loops=1)	
2	Filter: ((first_name)::text ~~ 'Dmytro0'::text)	
3	Rows Removed by Filter: 99996	
4	Planning Time: 0.068 ms	
5	Execution Time: 12.270 ms	

Тому даний індекс доцільніше використовувати для стовпців які найчастіше будуть бути використанні з оператором “=” - стовпці з числовими записами і датами.

Пункт 3

Створюємо тригер і функцію. Кожного разу коли у нас буде операція Оновлення та видалення (тригер спрацює перед видаленням) на таблиці, буде спрацьовувати тригер teacher_tr який в свою чергу буде викликати функцію std_func(). Ця функція в свою чергу, дивлячись на тип операції, або , якщо це оновлення, перевіряє нові значення полів для оновлення і, якщо вони валідні то в таблицю teacher_audit вставляємо запис з потрібними полями, або , якщо це видалення , то просто вставляємо потрібні значення в teacher_audit.

```
CREATE OR REPLACE FUNCTION std_func() RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        IF NEW.first_name is not NULL and length(NEW.first_name) = 0 THEN
            RAISE EXCEPTION 'first_name is empty';
        END IF;

        IF NEW.last_name is not NULL and length(NEW.last_name) = 0 THEN
            RAISE EXCEPTION 'first_name is empty';
        END IF;

        IF NEW.phone_number is not NULL and (length(NEW.phone_number) > 12 or length(NEW.phone_number) < 12) THEN
            RAISE EXCEPTION 'phone number is wrong or empty';
        END IF;

        INSERT INTO teacher_audit VALUES('UPDATE', user, NEW.first_name, New.last_name, New.phone_number, now());
        RETURN NEW;

    ELSEIF (TG_OP = 'DELETE') THEN
        INSERT INTO teacher_audit VALUES('DELETE', user, OLD.first_name, OLD.last_name, OLD.phone_number, now());
        RETURN OLD;

    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
1 CREATE TRIGGER teacher_tr
2 BEFORE DELETE OR UPDATE ON teacher
3 FOR EACH ROW EXECUTE PROCEDURE std_func();
```

Таким чином, будь то оновлення або видалення запису з таблиці teacher, в новій таблиці teacher_audit ми будемо бачити хто, коли і що змінив.

Приклади запитів і змін в таблиці teacher_audit.

1) Видалення.

Teacher_audit до операції видалення.

Query Editor

```
1 SELECT * FROM public.teacher_audit
```

Data Output

	action character varying	user character varying	first_name character varying	last_name character varying	phone_number character varying	action_date timestamp with time zone
1	DELETE	postgres	Pavel1	Melnik	380864703366	2020-12-26 00:00:00-03
2	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
3	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03

Запит на видалення

Query Editor

```
1 DELETE FROM teacher WHERE ID = 4
```

Messages

DELETE 1

Teacher_audit після операції видалення.

Query Editor

```
1 SELECT * FROM public.teacher_audit
```

Data Output

	action character varying	user character varying	first_name character varying	last_name character varying	phone_number character varying	action_date timestamp with time zone
1	DELETE	postgres	Pavel1	Melnik	380864703366	2020-12-26 00:00:00-03
2	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
3	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
4	DELETE	postgres	Oleg0	Melnik	380678668724	2020-12-26 15:18:10.649708-03

2) Оновлення .

Teacher_audit до операції оновлення .

Query Editor

```
1 SELECT * FROM public.teacher_audit
```

Data Output

	action character varying	user character varying	first_name character varying	last_name character varying	phone_number character varying	action_date timestamp with time zone
1	DELETE	postgres	Pavel1	Melnik	380864703366	2020-12-26 00:00:00-03
2	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
3	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
4	DELETE	postgres	Oleg0	Melnik	380678668724	2020-12-26 15:18:10.649708-03

Запит на оновлення

```
1 UPDATE teacher SET first_name = 'Katya' where teacher.id = 1
```

Teacher_audit після операції оновлення.

Query Editor

```
1 SELECT * FROM public.teacher_audit
```

Data Output

	action character varying	user character varying	first_name character varying	last_name character varying	phone_number character varying	action_date timestamp with time zone
1	DELETE	postgres	Pavel1	Melnik	380864703366	2020-12-26 00:00:00-03
2	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
3	UPDATE	postgres	Dima	Melnik	380384244213	2020-12-26 00:00:00-03
4	DELETE	postgres	Oleg0	Melnik	380678668724	2020-12-26 15:18:10.649708-03
5	UPDATE	postgres	Katya	Melnik	380384244213	2020-12-26 15:24:05.702007-03

Оброблення невірних даних.

Query Editor

```
1 UPDATE teacher SET phone_number = 'Katya' where teacher.id = 1
```

Messages

ERROR: ОШИБКА: phone number is wrong or empty
CONTEXT: функция PL/pgSQL std_func(), строка 13, оператор RAISE

SQL state: P0001