

**Національний технічний університет України “Київський  
політехнічний інститут ім. Ігоря Сікорського”**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих  
комп’ютерних систем**

**Лабораторна робота 2**

**“Засоби оптимізації СУБД PostgreSQL”**

**Група: KB-11**

**Виконав: Овчинніков Д.С.**

**Телеграм @dmytro\_ovchinnikov**

**Оцінка:**

**Київ – 2023**

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проєкції (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 18

18	BTree, GIN	after update, insert
----	------------	----------------------

ER модель «Шкільна система управління навчанням.»

Перелік сутностей і опис їх призначення:

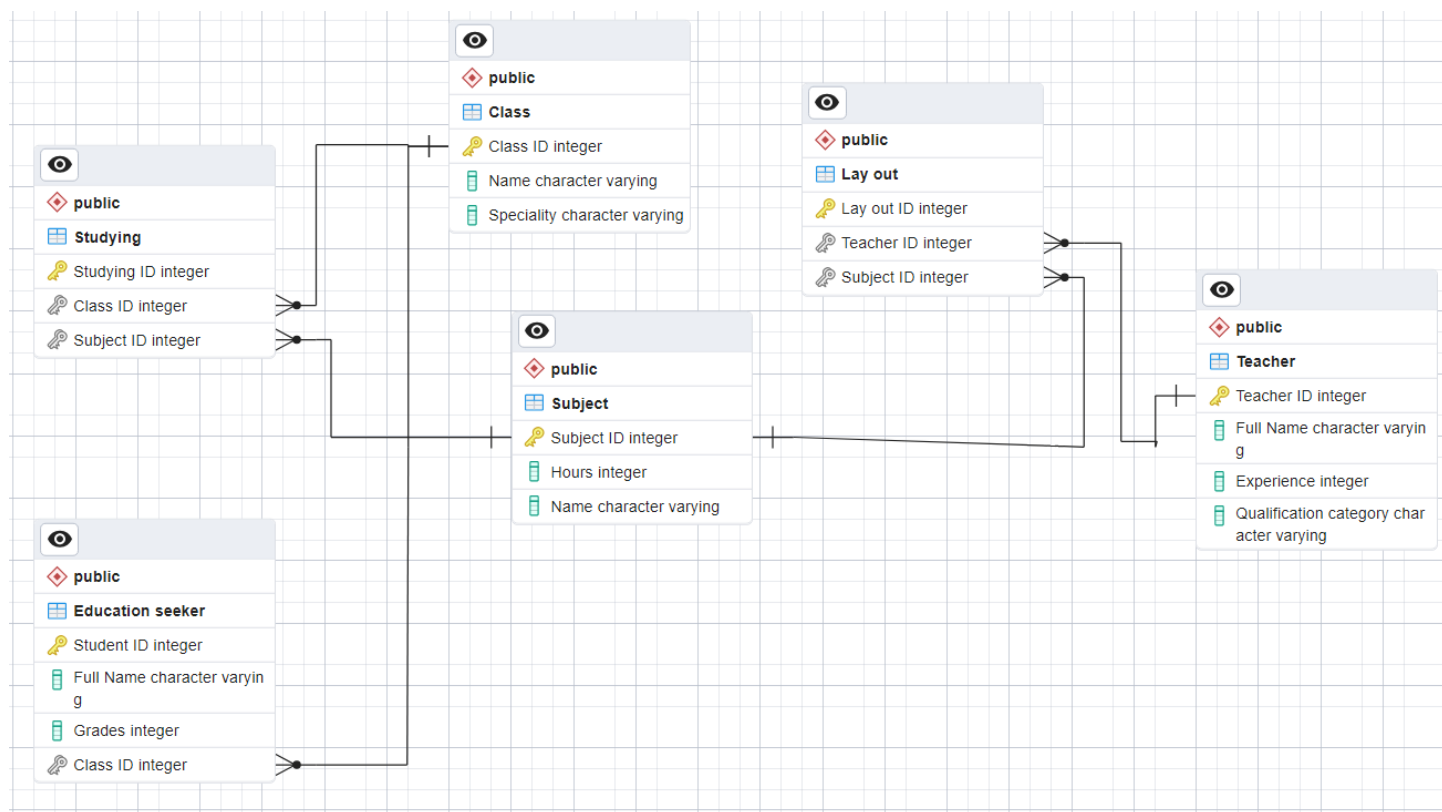
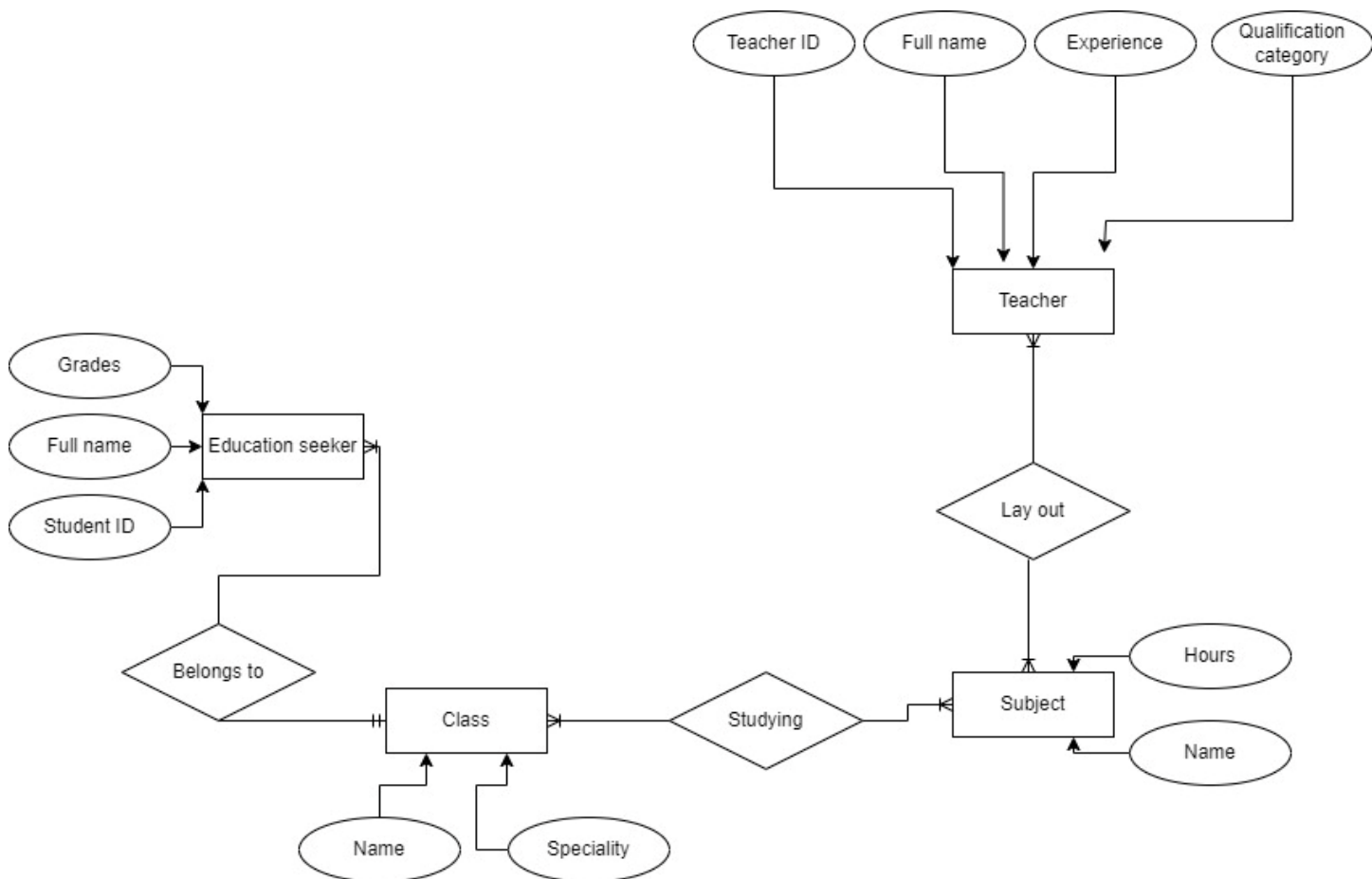
**Здобувач освіти (Education seeker)**, сутність призначена для збереження даних про здобувача освіти – ПІБ, середня оцінка, та ID учнівського, ID класу.

**Клас (Class)**, сутність призначена для збереження даних про клас здобувачів освіти для відслідковування року їх навчання та предмету їх поглибленого вивчення.

**Предмет (Subject)**, сутність призначена для збереження даних про предмети що вивчаються в школі – назву предмету, годин вивчення предмета.

**Вчитель (Teacher)**, сутність призначена для збереження даних про вчителя – ПІБ, Стаж, та Кваліфікаційна категорія.

Здобувач може навчатися тільки в одному класі. Для одного класу викладається декілька предметів і предмети викладаються для декількох класів. Вчитель може викладати декілька предметів і викладати ці ж предмети може і інший вчитель.



## Пункт №1

### Класи-сутності таблиць

```
class Class(Base):
    __tablename__ = 'Class'

    ClassID = Column('Class ID', Integer, primary_key=True, autoincrement=True)
    Name = Column(String, nullable=False)
    Speciality = Column(String, nullable=False)
    students = relationship('EducationSeeker', back_populates='clas')

class EducationSeeker(Base):
    __tablename__ = 'Education seeker'
    StudentID = Column('Student ID', Integer, primary_key=True, autoincrement=True)
    Full_name = Column('Full Name', String, nullable=False)
    Grades = Column(Integer, nullable=False)
    ClassID = Column('Class ID', Integer, ForeignKey('Class.Class ID',
onupdate='CASCADE', ondelete='CASCADE'),
        nullable=False)
    clas = relationship('Class', back_populates='students')

class Subject(Base):
    __tablename__ = 'Subject'
    SubjectID = Column('Subject ID', Integer, primary_key=True, autoincrement=True)
    Hours = Column(Integer, nullable=False)
    Name = Column(String, nullable=False)

class LayOut(Base):
    __tablename__ = 'Lay out'
    Lay_outID = Column('Lay out ID', Integer, primary_key=True, autoincrement=True)
    teacher_id = Column('Teacher ID', Integer, ForeignKey('Teacher.Teacher ID',
onupdate='CASCADE', ondelete='CASCADE'),
        nullable=False)
    SubjectID = Column('Subject ID', Integer, ForeignKey('Subject.Subject ID',
onupdate='CASCADE', ondelete='CASCADE'),
        nullable=False)
    teacher = relationship('Teacher', foreign_keys=[teacher_id])
    subject = relationship('Subject')

class Teacher(Base):
    __tablename__ = 'Teacher'
    teacher_id = Column('Teacher ID', Integer, primary_key=True, autoincrement=True)
    Full_name = Column('Full Name', String, nullable=False)
    Experience = Column(Integer, nullable=False)
    Qualification_category = Column('Qualification category', String, nullable=False)
    lay = relationship('LayOut', foreign_keys=[LayOut.teacher_id], overlaps="teacher")

class Studying(Base):
    __tablename__ = 'Studying'
    StudyingID = Column('Studying ID', Integer, primary_key=True, autoincrement=True)
    ClassID = Column('Class ID', Integer, ForeignKey('Class.Class ID', onupdate='CASCADE',
ondelete='CASCADE'), nullable=False)
    SubjectID = Column('Subject ID', Integer, ForeignKey('Subject.Subject ID',
onupdate='CASCADE', ondelete='CASCADE'), nullable=False)
    class_ = relationship('Class')
    subject = relationship('Subject')
```

## Вигляд функцій керування базою даних

```
class Model:
    def create_all(self):
        Base.metadata.create_all(engine)

    def add_class(self, name, spec):
        class_ = Class(Name=name, Speciality=spec)
        session.add(class_)
        session.commit()

    def update_class(self, class_id, name, spec):
        class_ = session.query(Class).get(class_id)
        if class_:
            class_.Name = name
            class_.Speciality = spec
            session.commit()
            return 1
        else:
            return 0

    def delete_class(self, class_id):
        class_ = session.query(Class).get(class_id)
        studying = session.query(Studying).filter_by(ClassID=class_id).first()
        if class_ and not studying:
            session.delete(class_)
            session.commit()
            return "Class deleted succesfully"
        else:
            return "There is no such class id"

    def add_education_seeker(self, name, grades, class_id):
        class_ = session.query(Class).get(class_id)
        if class_:
            education_seeker = EducationSeeker(Full_name = name, Grades = grades, ClassID
= class_id)
            session.add(education_seeker)
            session.commit()
            return 1
        else:
            return 0

    def update_education_seeker(self, student_id, name, grade, class_id):
        class_ = session.query(Class).get(class_id)
        education_seeker = session.query(EducationSeeker).get(student_id)
        if class_ and education_seeker:
            education_seeker.Full_name = name
            education_seeker.Grades = grade
            education_seeker.ClassID = class_id
            session.commit()
            return 1
        else:
            return 0

    def delete_education_seeker(self, student_id):
        education_seeker = session.query(EducationSeeker).get(student_id)
        if education_seeker:
            session.delete(education_seeker)
            session.commit()
            return 1
        else:
            return 0

    def add_subject(self, hours, name):
```

```

subject = Subject(Hours=hours, Name=name)
session.add(subject)
session.commit()

def update_subject(self, subject_id, hours, name):
    subject = session.query(Subject).get(subject_id)
    if subject:
        subject.Hours = hours
        subject.Name = name
        session.commit()
        return 1
    else:
        return 0

def delete_subject(self, subject_id):
    subject = session.query(Subject).get(subject_id)
    laying = session.query(LayOut).filter_by(SubjectID=subject_id).first()
    studying = session.query(Studying).filter_by(ClassID=subject_id).first()
    if subject and not laying and not studying:
        session.delete(subject)
        session.commit()
        return 1
    else:
        return 0

def add_teacher(self, name, exp, qual):
    teacher = Teacher(Full_name=name, Experience=exp, Qualification_category=qual)
    session.add(teacher)
    session.commit()

def update_teacher(self, teacher_id, name, exp, qual):
    teacher = session.query(Teacher).get(teacher_id)
    if teacher:
        teacher.Full_name = name
        teacher.Experience = exp
        teacher.Qualification_category = qual
        session.commit()
        return 1
    else:
        return 0

def delete_teacher(self, teacher_id):
    teacher = session.query(Teacher).get(teacher_id)
    laying = session.query(LayOut).filter_by(TeacherID=teacher_id).first()
    if teacher and not laying:
        session.delete(teacher)
        session.commit()
        return 1
    else:
        return 0

def add_laying(self, teach_id, subject_id):
    teacher = session.query(Teacher).get(teach_id)
    subject = session.query(Subject).get(subject_id)
    if teacher and subject:
        lay_out = LayOut(TeacherID=teach_id, SubjectID=subject_id)
        session.add(lay_out)
        session.commit()
        return 1
    else:
        return 0

def update_laying(self, laying_id, teach_id, subject_id):
    laying = session.query(LayOut).get(laying_id)

```

```

teacher = session.query(Teacher).get(teach_id)
subject = session.query(Subject).get(subject_id)
if laying and teacher and subject:
    laying.TeacherID = teach_id
    laying.SubjectID = subject_id
    session.commit()
    return 1
else:
    return 0

def delete_laying(self, laying_id):
    laying = session.query(LayOut).get(laying_id)
    if laying:
        session.delete(laying)
        session.commit()
        return 1
    else:
        return 0

def add_studying(self, class_id, subject_id):
    class_ = session.query(Class).get(class_id)
    subject = session.query(Subject).get(subject_id)
    if class_ and subject:
        studying = LayOut(ClassID=class_id, SubjectID=subject_id)
        session.add(studying)
        session.commit()
        return 1
    else:
        return 0

def update_studying(self, studying_id, class_id, subject_id):
    studying = session.query(LayOut).get(studying_id)
    class_ = session.query(Class).get(class_id)
    subject = session.query(Subject).get(subject_id)
    if studying and class_ and subject:
        studying.ClassID = class_id
        studying.SubjectID = subject_id
        session.commit()
        return 1
    else:
        return 0

def delete_studying(self, studying_id):
    studying = session.query(Studying).get(studying_id)
    if studying:
        session.delete(studying)
        session.commit()
        return 1
    else:
        return 0


```

Model.py було повністю переписано для sqlalchemy. При цьому вхідні аргументи залишились без змін ,отже користувач змін не відчує.

## Пункт №2


Профільтруємо таблицю Teacher ,і виберемо вчителів з певною кваліфікаційною категорією яку ми заповнили випадковим даними в ргр

```
EXPLAIN ANALYZE SELECT * FROM public."Teacher" WHERE "Qualification category" = 'First'
```

	QUERY PLAN	
	text	
1	Seq Scan on "Teacher" (cost=0.00..68.08 rows=838 width=27) (actual time=0.023..0.800 rows=824 loops=1)	
2	Filter: (("Qualification category")::text = 'First'::text)	
3	Rows Removed by Filter: 2472	
4	Planning Time: 2.670 ms	
5	Execution Time: 0.849 ms	

Як ми бачимо без індексу знадобилась майже мікросекунда,подивимось скільки знадобиться якщо використати індекс Btree

```
CREATE INDEX idx_full_name_btree ON public."Teacher" USING btree ("Qualification category");  
EXPLAIN ANALYZE SELECT * FROM public."Teacher" WHERE "Qualification category" = 'First'
```

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on "Teacher" (cost=14.67..50.97 rows=824 width=27) (actual time=0.081..0.180 rows=824 loops=1)	
2	Recheck Cond: (("Qualification category")::text = 'First'::text)	
3	Heap Blocks: exact=26	
4	-> Bitmap Index Scan on idx_full_name_btree (cost=0.00..14.46 rows=824 width=0) (actual time=0.067..0.067 rows=824 loop=1)	
5	Index Cond: (("Qualification category")::text = 'First'::text)	
6	Planning Time: 3.144 ms	
7	Execution Time: 0.225 ms	

Як результат використання індексу пришвидшило виконання запиту у 4 рази. ВТree забезпечує швидкий доступ до даних, оскільки він дозволяє швидко знаходити потрібні значення через структуру дерева та розподіл даних по різних рівнях.



Тепер підрахуємо кількість вчителів певної категорії, спочатку без індексу

```
EXPLAIN ANALYZE SELECT "Qualification category", COUNT(*) AS total FROM public."Teacher" GROUP BY "Qualification category";
```

	QUERY PLAN text	🔒
1	HashAggregate (cost=75.44..75.48 rows=4 width=14) (actual time=1.758..1.759 rows=4 loops=1)	
2	Group Key: "Qualification category"	
3	Batches: 1 Memory Usage: 24kB	
4	-> Seq Scan on "Teacher" (cost=0.00..58.96 rows=3296 width=6) (actual time=0.018..0.456 rows=3296 loops=1)	
5	Planning Time: 0.238 ms	
6	Execution Time: 1.804 ms	

Тепер з індексом Btree

```
CREATE INDEX idx_btree ON public."Teacher" USING btree ("Qualification category");  
EXPLAIN ANALYZE SELECT "Qualification category", COUNT(*) AS total FROM public."Teacher" GROUP BY "Qualification category";
```


	QUERY PLAN text	🔒
1	HashAggregate (cost=75.44..75.48 rows=4 width=14) (actual time=0.997..0.998 rows=4 loops=1)	
2	Group Key: "Qualification category"	
3	Batches: 1 Memory Usage: 24kB	
4	-> Seq Scan on "Teacher" (cost=0.00..58.96 rows=3296 width=6) (actual time=0.015..0.222 rows=3296 loops=1)	
5	Planning Time: 1.168 ms	
6	Execution Time: 1.062 ms	

Різниця майже в 2 рази. Індекс BTree допомагає швидко знаходити та організовувати дані відповідно до значень у відповідному стовпці. Під час виконання операції Group By, індекс BTree допомагає швидко відокремити рядки за значеннями цього стовпця для подальшого групування та може зменшити кількість ресурсів, необхідних для виконання операцій групування за рахунок оптимізації доступу до даних та їхнього відбору.

Відсортуюмо всіх вчителів за стажем


Без індексу

```
EXPLAIN ANALYZE SELECT * FROM public."Teacher" WHERE "Experience" > 20;
```

	QUERY PLAN	
	text	
1	Seq Scan on "Teacher" (cost=0.00..67.20 rows=2665 width=27) (actual time=0.022..0.556 rows=2665 loops...	
2	Filter: ("Experience" > 20)	
3	Rows Removed by Filter: 631	
4	Planning Time: 0.174 ms	
5	Execution Time: 0.701 ms	

З індексом Btree

```
CREATE INDEX idx_btre ON public."Teacher" USING btree ("Experience");  
EXPLAIN ANALYZE SELECT * FROM public."Teacher" WHERE "Experience" > 20;
```

	QUERY PLAN	
	text	
1	Seq Scan on "Teacher" (cost=0.00..67.20 rows=2665 width=27) (actual time=0.012..0.525 rows=2665 loops...	
2	Filter: ("Experience" > 20)	
3	Rows Removed by Filter: 631	
4	Planning Time: 3.756 ms	
5	Execution Time: 0.615 ms	

Результат майже не змінився через те що дані нерівномірно розподілені, та велика частина значень відпала через цю умову що сповільнило пошук за допомогою Btree


Так як у моїй базі даних немає таблиць з типами з якими працюють GIN то створимо тестову таблицю.

```
CREATE TABLE test(
  id serial PRIMARY KEY,
  vector tsvector,
  num integer,
  txt TEXT
);

INSERT INTO test (vector, num, txt)
SELECT
  to_tsvector('english', md5(random()::text)),
  floor(random() * 1000),
  md5(random()::text)
FROM generate_series(1, 10000);
```


Відсортуємо за txt без індексації

```
Explain ANALYZE SELECT * FROM test ORDER BY vector
```

	QUERY PLAN	
	text	
1	Sort (cost=17267.93..17542.93 rows=110000 width=86) (actual time=144.591..180.723 rows=110000 loops=1)	
2	Sort Key: vector	
3	Sort Method: external merge Disk: 10712kB	
4	-> Seq Scan on test (cost=0.00..2793.00 rows=110000 width=86) (actual time=0.017..6.724 rows=110000 loops...	
5	Planning Time: 0.415 ms	
6	Execution Time: 185.975 ms	

А тепер з індексом GIN

```
CREATE INDEX ON test using GIN(vector);
Explain ANALYZE SELECT * FROM test ORDER BY vector
```

	QUERY PLAN	
	text	
1	Sort (cost=17267.93..17542.93 rows=110000 width=86) (actual time=122.113..154.456 rows=110000 loops=1)	
2	Sort Key: vector	
3	Sort Method: external merge Disk: 10712kB	
4	-> Seq Scan on test (cost=0.00..2793.00 rows=110000 width=86) (actual time=0.006..5.430 rows=110000 loops...	
5	Planning Time: 3.301 ms	
6	Execution Time: 159.404 ms	

Як ми можемо помітити Gin пришвидшивши виконання в півтора рази. Даний індекс створений для текстового пошуку і спеціалізується на великих рядках, тож при малою

кількості вибірок він може себе погано показувати. GIN індексує не значення а окремі елементи.