



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”  
Варіант “Блог”

Виконала  
студентка 2 курсу  
групи КП-92

Семьонова Поліна Дмитрівна

Перевірів  
“ \_\_\_\_\_ ” “ \_\_\_\_\_ ” 2020 р.  
викладач

Замекула Олексій Ігорович  
(прізвище, ім’я, по батькові)

Київ 2020

# План

1. Завдання
2. Структура розробленої моделі у схемі бази даних (таблиці) PostgreSQL
3. Класи ORM, що відповідають таблицям бази даних, приклади запитів у вигляді ORM.
4. Команди створення індексів, тексти, результати і час виконання запитів SQL.
5. Команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних
6. Ілюстрації програмного коду з репозиторію Git.

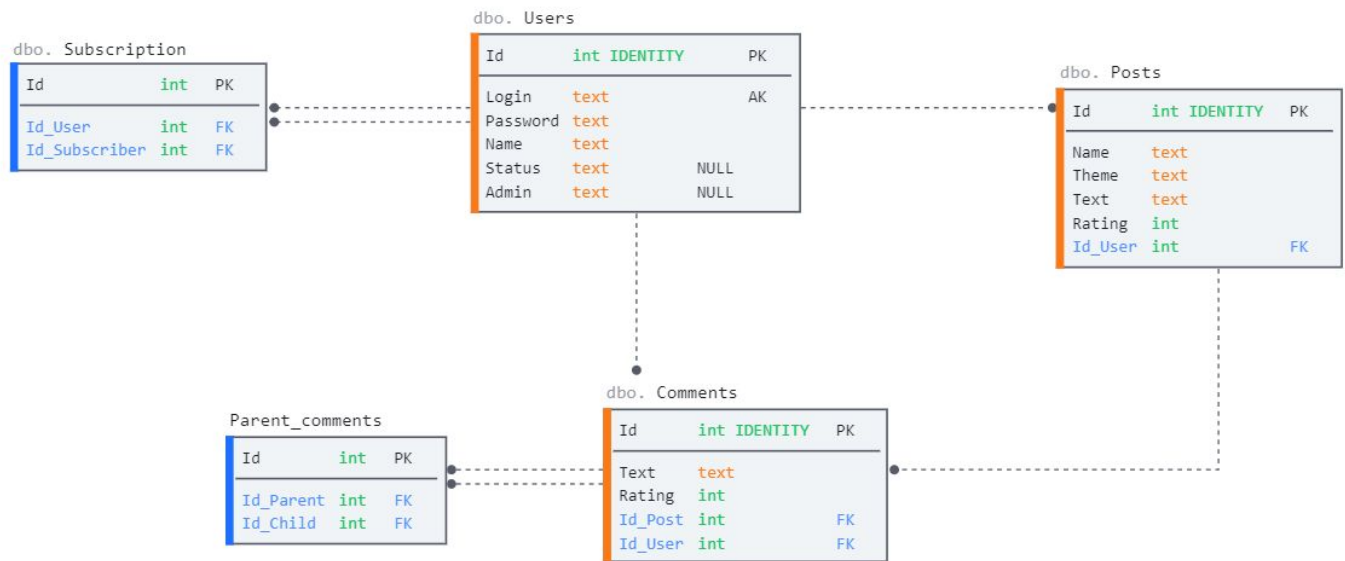
### Завдання

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.

### B-21

21	<i>Btree, Hash</i>	<i>before delete, update</i>
----	--------------------	------------------------------

# Структура розробленої моделі у схему бази даних (таблиці) PostgreSQL



## Класи ORM, що відповідають таблицям бази даних, приклади запитів у вигляді ORM

### users

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    login = Column(Text)
    password = Column(Text)
    name = Column(Text)
    status = Column(Text)
    admin = Column(Text)
    comments = relationship("Comment", cascade="all, delete", backref="users")
    posts = relationship("Post", passive_deletes=True, backref="users")
    id_sub = relationship('User',
        secondary=subscription,
        primaryjoin=(subscription.c.id_user == id),
        secondaryjoin=(subscription.c.id_subscriber == id),
        backref=backref('subscriber', lazy='dynamic'),
        lazy='dynamic')

    def __repr__(self):
        return "<User(id={}, login={}, password={}, name={}, status={}, admin={})>" \
            .format(self.id, self.login, self.password, self.name, self.status, self.admin)
```

### posts

```
class Post(Base):
    __tablename__ = 'posts'
    id = Column(Integer, primary_key=True)
    name = Column(Text)
    theme = Column(Text)
    text = Column(Text)
    rating = Column(Integer)
    id_user = Column(Integer, ForeignKey('users.id'))
    comments = relationship("Comment", cascade="all, delete", backref="posts")
    def __repr__(self):
        return "<Post(id={}, name={}, theme={}, text={}, rating={}, id_user={})>" \
            .format(self.id, self.name, self.theme, self.text, self.rating, self.id_user)
```

### comments

```
class Comment(Base):
    __tablename__ = 'comments'
    id = Column(Integer, primary_key=True)
    text = Column(Text)
    rating = Column(Integer)
    id_user = Column(Integer, ForeignKey('users.id'))
    id_post = Column(Integer, ForeignKey('posts.id'))
    parent = relationship('Comment', cascade="all, delete",
        secondary=parent_comments,
        primaryjoin=(parent_comments.c.id_parent == id),
```

```

secondaryjoin = (parent_comments.c.id_child == id),
                 backref=backref('child', lazy='dynamic',),
                 lazy='dynamic')
def __repr__(self):
return "<Comment(id={}, text='{}', rating={}, id_user={}, id_post={})>" \
.format(self.id, self.text, self.rating, self.id_user, self.id_post)

```

## parent\_comments

```

parent_comments=Table('parent_comments', Base.metadata,
    Column('id_parent', Integer, ForeignKey('comments.id')),
    Column('id_child', Integer, ForeignKey('comments.id'))
)

```

## subscription

```

subscription =Table('subscription',Base.metadata,
    Column('id_user', Integer, ForeignKey('users.id')),
    Column('id_subscriber', Integer, ForeignKey('users.id'))
)

```

## Examples

### find\_post

```

try:
    posts=self.s.query(Post).filter_by(name=name).all()
    print(posts)
    if posts:
        return posts
    else:
        return "Can't find posts by name"
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in find_post():", error)

```

### find\_post\_by\_id

```

try:
    post=self.s.query(Post).get(id)
    self.s.commit()
    if post:
        return post
    else:
        return "Can't find posts by id"
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in find_post_by_id():", error)

```

## find\_all\_posts

```
try:
    posts = self.s.query(Post).all()
    self.s.commit()
    if posts:
        return posts
    else:
        return "Can`t find posts"
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in find_all_posts():", error)
```

## add\_post

```
try:
    post=Post(
        name=name,
        theme = theme,
        text = text,
        rating = 0,
        id_user = id_user
    )
    self.s.add(post)
    self.s.commit()
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in add_post():", error)
return "Add is successful"
```

## update\_post

```
try:
    p = self.s.query(Post).filter_by(id=id).update(
        {Post.text: text, Post.rating:
rating, Post.name:name, Post.theme:theme})
    self.s.commit()
    if p:
        return "Update post is successful"
    else:
        return "Can`t update post"
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in update_post():", error)
```

## remove\_post

```
try:
    post=self.find_post_by_id(id)
    if type(post) is not Post:
        return "Can`t find id"
    c = self.s.delete(post)
```

```
        self.s.commit()
    except (Exception, exc.SQLAlchemyError) as error:
        self.s.rollback()
        print("Error in remove_post():", error)
    return "Delete is successful"
```

## find\_subscribers

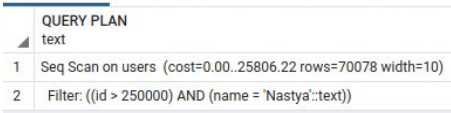

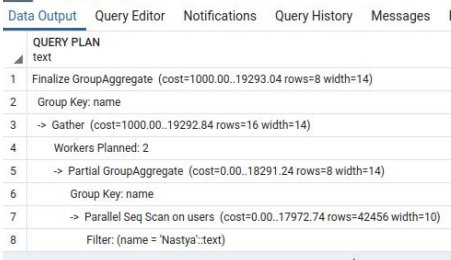
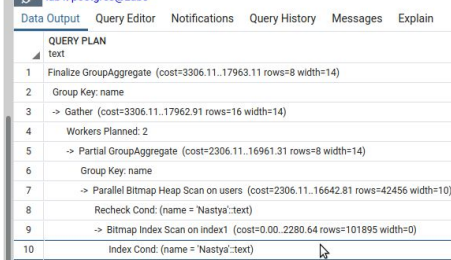
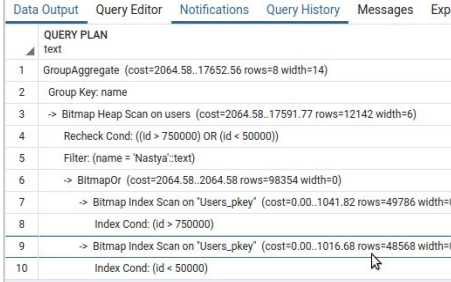
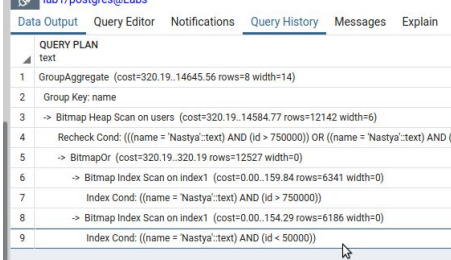
```
try:
    subscribers=self.s.query(User).join(subscription,
subscription.c.id_subscriber==User.id).filter_by(id_user=id_user).all()
    self.s.commit()
    if subscribers:
        return subscribers
    else:
        return "Can`t find subscribers by id_user"
except (Exception, exc.SQLAlchemyError) as error:
    self.s.rollback()
    print("Error in find_subscriber():", error)
```



# Команди створення індексів, тексти, результати і час виконання запитів SQL

## Btree

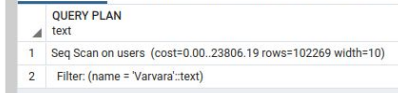
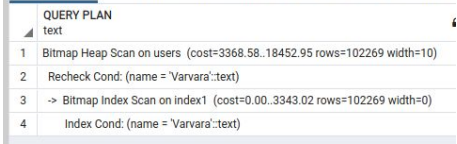
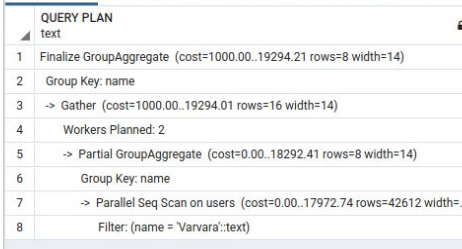
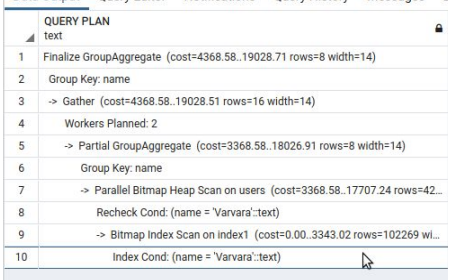
```
1 create index index1 on users(name,id)
```

Запити	before creating	after creating
select * from users where name='Nastya' and id>250000	 <pre> QUERY PLAN text 1  Seq Scan on users  (cost=0.00..25806.22 rows=70078 width=10) 2  Filter: ((id &gt; 250000) AND (name = 'Nastya'::text))           </pre>	 <pre> QUERY PLAN text 1  Bitmap Heap Scan on users  (cost=1762.72..16619.89 rows=70078 width=10) 2  Recheck Cond: ((name = 'Nastya'::text) AND (id &gt; 250000)) 3  -&gt; Bitmap Index Scan on index1  (cost=0.00..1745.20 rows=70078 width=0) 4  Index Cond: ((name = 'Nastya'::text) AND (id &gt; 250000))           </pre>
select max(id),min(id), name where name='Nastya' group by name	 <pre> Data Output Query Editor Notifications Query History Messages Explain QUERY PLAN text 1  Finalize GroupAggregate  (cost=1000.00..19293.04 rows=8 width=14) 2  Group Key: name 3  -&gt; Gather  (cost=1000.00..19292.84 rows=16 width=14) 4  Workers Planned: 2 5  -&gt; Partial GroupAggregate  (cost=0.00..18291.24 rows=8 width=14) 6  Group Key: name 7  -&gt; Parallel Seq Scan on users  (cost=0.00..17972.74 rows=42456 width=10) 8  Filter: (name = 'Nastya'::text)           </pre>	 <pre> Data Output Query Editor Notifications Query History Messages Explain QUERY PLAN text 1  Finalize GroupAggregate  (cost=3306.11..17963.11 rows=8 width=14) 2  Group Key: name 3  -&gt; Gather  (cost=3306.11..17962.91 rows=16 width=14) 4  Workers Planned: 2 5  -&gt; Partial GroupAggregate  (cost=2306.11..16961.31 rows=8 width=14) 6  Group Key: name 7  -&gt; Parallel Bitmap Heap Scan on users  (cost=2306.11..16642.81 rows=42456 width=10) 8  Recheck Cond: (name = 'Nastya'::text) 9  -&gt; Bitmap Index Scan on index1  (cost=0.00..2280.64 rows=101895 width=0) 10 Index Cond: (name = 'Nastya'::text)           </pre>
select name, count(*) from users where name='Nastya' and (id>750000 or id<50000) group by name	 <pre> Data Output Query Editor Notifications Query History Messages Explain QUERY PLAN text 1  GroupAggregate  (cost=2064.58..17652.56 rows=8 width=14) 2  Group Key: name 3  -&gt; Bitmap Heap Scan on users  (cost=2064.58..17591.77 rows=12142 width=6) 4  Recheck Cond: ((id &gt; 750000) OR (id &lt; 50000)) 5  Filter: (name = 'Nastya'::text) 6  -&gt; BitmapOr  (cost=2064.58..2064.58 rows=98354 width=0) 7  -&gt; Bitmap Index Scan on "Users_pkey"  (cost=0.00..1041.82 rows=49786 width=0) 8  Index Cond: (id &gt; 750000) 9  -&gt; Bitmap Index Scan on "Users_pkey"  (cost=0.00..1016.68 rows=48568 width=0) 10 Index Cond: (id &lt; 50000)           </pre>	 <pre> lab1/postgres@Labs Data Output Query Editor Notifications Query History Messages Explain QUERY PLAN text 1  GroupAggregate  (cost=320.19..14645.56 rows=8 width=14) 2  Group Key: name 3  -&gt; Bitmap Heap Scan on users  (cost=320.19..14584.77 rows=12142 width=6) 4  Recheck Cond: (((name = 'Nastya'::text) AND (id &gt; 750000)) OR ((name = 'Nastya'::text) AND (id &lt; 50000))) 5  -&gt; BitmapOr  (cost=320.19..320.19 rows=12527 width=0) 6  -&gt; Bitmap Index Scan on index1  (cost=0.00..159.84 rows=6341 width=0) 7  Index Cond: ((name = 'Nastya'::text) AND (id &gt; 750000)) 8  -&gt; Bitmap Index Scan on index1  (cost=0.00..154.29 rows=6186 width=0) 9  Index Cond: ((name = 'Nastya'::text) AND (id &lt; 50000))           </pre>

В усіх прикладах команда з індексом виконувалась швидше, оскільки використання бінарного дерева допомагає нам виконувати бінарний пошук, який має швидкодію  $O(\log(n))$  на елементах замість лінійного, який має швидкодію  $O(n)$ . Отже, знаючи діапазон шуканих елементів, ми на кожному кроці можемо відкинути половину можливих значень, тому бінарний пошук значно швидший, ніж лінійний.

## Hash

```
1 create index index1 on users using hash(name);
```

Запити	before creating	after creating
select name,id from users where name ='Nastya'	 <pre> QUERY PLAN text 1 Seq Scan on users (cost=0.00..23806.19 rows=102269 width=10) 2 Filter: (name = 'Nastya'::text) </pre>	 <pre> QUERY PLAN text 1 Bitmap Heap Scan on users (cost=3368.58..18452.95 rows=102269 width=10) 2 Recheck Cond: (name = 'Nastya'::text) 3 -&gt; Bitmap Index Scan on index1 (cost=0.00..3343.02 rows=102269 width=0) 4 Index Cond: (name = 'Nastya'::text) </pre>
select name, count(*) from users where name='Nastya' ' and (id>750000 or id<50000) group by name	 <pre> QUERY PLAN text 1 Finalize GroupAggregate (cost=1000.00..19294.21 rows=8 width=14) 2 Group Key: name 3 -&gt; Gather (cost=1000.00..19294.01 rows=16 width=14) 4 Workers Planned: 2 5 -&gt; Partial GroupAggregate (cost=0.00..18292.41 rows=8 width=14) 6 Group Key: name 7 -&gt; Parallel Seq Scan on users (cost=0.00..17972.74 rows=42612 width=10) 8 Filter: (name = 'Nastya'::text) </pre>	 <pre> QUERY PLAN text 1 Finalize GroupAggregate (cost=4368.58..19028.71 rows=8 width=14) 2 Group Key: name 3 -&gt; Gather (cost=4368.58..19028.51 rows=16 width=14) 4 Workers Planned: 2 5 -&gt; Partial GroupAggregate (cost=3368.58..18026.91 rows=8 width=14) 6 Group Key: name 7 -&gt; Parallel Bitmap Heap Scan on users (cost=3368.58..17707.24 rows=42612 width=10) 8 Recheck Cond: (name = 'Nastya'::text) 9 -&gt; Bitmap Index Scan on index1 (cost=0.00..3343.02 rows=102269 width=0) 10 Index Cond: (name = 'Nastya'::text) </pre>

В усіх прикладах команда з індексом виконувалась швидше, оскільки використання хеш-таблиць дозволяє нам отримувати необхідне значення, перетворивши поле в ключ через спеціальну хеш-функцію. Звичайно, ця хеш-функція може перетворити різні значення в однаковий ключ і тоді доведеться проходити всі можливі значення з цим ключем. Проте воно все-таки швидше, оскільки дозволяє відкинути всі значення, які не перетворились в шуканий ключ. Від правильно підібраної функції для хешування залежить наскільки пришвидшився пошук елементів. Також досить важливе значення має як саме хеш-таблиця вирішує колізії. Проте, оскільки для отримання результатів, нам необхідно знати повне значення, щоб перетворити в ключ, ми можемо виконувати пошук лише за оператором "=", що є досить незручним, тому я б не рекомендувала використовувати цей тип індексації.

## Команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних

### before delete

```
create or replace function func_for_before_delete()
Returns trigger as $$
declare
    rec record;
begin
    for rec in select * from parent_comments
    loop
        if rec.id_child=OLD.id THEN
            DELETE FROM parent_comments where id=rec.id;
        end if;
        if rec.id_parent=OLD.id THEN
            DELETE FROM parent_comments where id=rec.id;
        end if;
    end loop;
Return OLD;
end;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER before_comment_delete
BEFORE DELETE
ON comments for each row
EXECUTE PROCEDURE func_for_before_delete();
```

### Дані до тригера

public.comments/lab1/postgres@Labs

	id [PK] integer	text text	rating integer	id_post integer	id_user integer
1	200123	Scfae9...	-359	4	5
2	200122	d5eef0...	-359	8	5
3	200120	fb3fec...	-359	7	7
4	200119	c38a2f...	-359	21	9
5	200118	d8e7fd...	-359	13	2
6	200117	c4fc72...	-359	9	4
7	200116	d2809...	-359	10	5

public.parent\_comments/lab1/postgres@Labs

	id [PK] integer	id_parent integer	id_child integer
1	2	6	3
2	16	200123	200122
3	19	200119	200120

## Дані після першого видалення

```
1 delete from comments where id=200122
```

public.comments/lab1/postgres@Labs

Data Output Query Editor Notifications Query History Message

	id [PK] integer	text text	rating integer	id_post integer	id_user integer
1	200123	Scfae9...	-359	4	5
2	200120	fb3fec...	-359	7	7
3	200119	c38a2f...	-359	21	9
4	200118	d8e7fd...	-359	13	2
5	200117	c4fc72...	-359	9	4
6	200116	d2809...	-359	10	5


public.parent\_comments/lab1/postgres@La

Data Output Query Editor Notifications Qu

	id [PK] integer	id_parent integer	id_child integer
1	2	6	3
2	19	200119	200120

## Дані після другого видалення

1 delete from comments where id=200119

 public.comments/lab1/postgres@Labs

Data Output


Query Editor

Notifications

Query History

Messages

	id [PK] integer	text text	rating integer	id_post integer	id_user integer
1	200123	5cfae9...	-359	4	5
2	200120	fb3fec...	-359	7	7
3	200118	d8e7fd...	-359	13	2
4	200117	c4fc72...	-359	9	4
5	200116	d2809...	-359	10	5

 public.parent\_comments/lab1/postgres@La

Data Output

Query Editor

Notifications

Queries

	id [PK] integer	id_parent integer	id_child integer
1	2	6	3

## before update

```
create or replace function func_before_update()
Returns trigger as $$
begin
    if(New.text='') then
        raise exception 'you can't update with empty text';
```

```

end if;
Return NEW;
end;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER before_update_comment
before update
ON comments for each row
EXECUTE PROCEDURE func_before_update();

```

## Дані до тригера

public.comments/lab1/postgres@Labs

Data Output Query Editor Notifications Query History Messa

	id [PK] integer	text text	rating integer	id_post integer	id_user integer
1	200123	iyut	-359	4	5
2	200120	fb3fec...	-359	7	7
3	200118	d8e7fd...	-359	13	2
4	200117	c4fc72...	-359	9	4
5	200116	d2809...	-359	10	5

## Дані після першого оновлення

Query Editor

```
1 update comments set text='' where id=200123
```

---

lab1/postgres@Labs

Data Output Notifications Query History Messages Explain

ERROR: you can't update with empty text  
 CONTEXT: PL/pgSQL function func\_before\_update() line 4 at RAISE  
 SQL state: P0001

## Дані після другого оновлення

Query Editor

```
1 update comments set text='hello' where id=200123
```

---

public.comments/lab1/postgres@Labs

Data Output Query Editor Notifications Query History Messa

	id [PK] integer	text text	rating integer	id_post integer	id_user integer
1	200123	hello	-359	4	5
2	200120	fb3fec...	-359	7	7
3	200118	d8e7fd...	-359	13	2
4	200117	c4fc72...	-359	9	4
5	200116	d2809...	-359	10	5

## Ілюстрації програмного коду з репозиторію Git

Pokaori lab is done 3		f6595ba 2 hours ago	History
..			
📁 .idea	lab is done 3		2 hours ago
📄 View.py	lab is done 3		2 hours ago
📄 database.py	lab is done 3		2 hours ago
📄 main.py	lab is done 3		2 hours ago
📄 models.py	lab is done 3		2 hours ago

<https://github.com/Pokaori/db/tree/master/lab3>