

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»

Тема: «**Засоби оптимізації роботи СУБД PostgreSQL**»

Виконав: студент III курсу

ФПМ групи КВ-81

Поляков Є.А.

Викладач: Петрашенко А. В.

Київ 2020

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.

Вимоги до пункту завдання №1

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlalchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:M, M:M та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

Вимоги до пункту завдання №2

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результируючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

Вимоги до пункту завдання №3

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується

(вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

Варіант 15:

Види індексів - Hash, BRIN

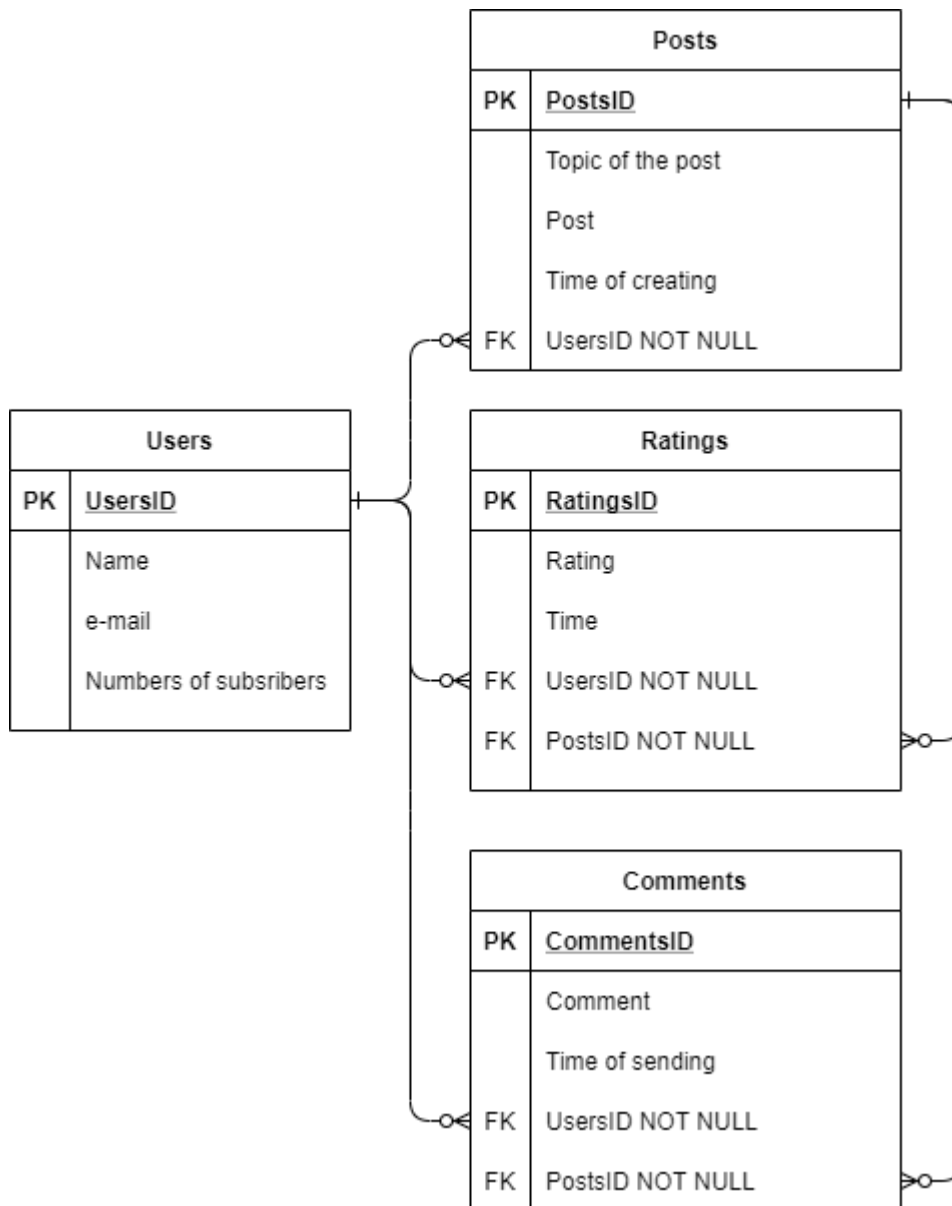
Умови для тригера - before delete, update

Посилання на Git-репозиторій:

https://github.com/Gregor-an/Database_lab

Пункт №1:

Зв'язки між таблицями:



Опис таблиці "Users":

```
class Users(Base):
    __tablename__ = 'Users'
    UserID = Column(Integer, Sequence('Users_UserID_seq'), primary_key=True)
    Name = Column(String, nullable=False)
    Email = Column(String, nullable=False)
    N_sub = Column(Integer, nullable=False)

    posts=relationship("Posts", back_populates="users", cascade="all, delete-orphan")
    comments=relationship("Comments", back_populates="users", cascade="all, delete-orphan")
    ratings=relationship("Ratings", back_populates="users", cascade="all, delete-orphan")
    def __repr__(self):
        return "<User(name='%s', email='%s', n_sub='%s')>" % (
            self.Name, self.Email, self.N_sub)
```

Опис таблиці “Posts”:

```
class Posts(Base):
    __tablename__ = 'Posts'
    PostID = Column(Integer, Sequence('Posts_PostID_seq'), primary_key=True)
    Topic = Column(String(60), nullable=False)
    Post = Column(Text, nullable=False)
    Time_create = Column(Date, nullable=False)

    UserIDFK = Column('UserIDFK', Integer, ForeignKey('Users.UserID', ondelete="CASCADE", onupdate="CASCADE"), nullable=False)
    users=relationship("Users", back_populates="posts")
    comments=relationship("Comments", back_populates="posts", cascade="all, delete-orphan")
    ratings=relationship("Ratings", back_populates="posts", cascade="all, delete-orphan")
    def __repr__(self):
        return "<Post(Topic='%s', Post='%s', Time_create='%s')>" % (
            self.Topic, self.Post, self.Time_create)
```

Опис таблиці “Comments”:

```
class Comments(Base):
    __tablename__ = 'Comments'
    CommentID = Column(Integer, Sequence('Comments_CommentID_seq'), primary_key=True)
    Comment = Column(Text, nullable=False)
    Time_create = Column(Date, nullable=False)

    UserIDFK = Column('UserIDFK', Integer, ForeignKey('Users.UserID', ondelete="CASCADE", onupdate="CASCADE"), nullable=False)
    users=relationship("Users", back_populates="comments")
    PostIDFK = Column('PostIDFK', Integer, ForeignKey('Posts.PostID', ondelete="CASCADE", onupdate="CASCADE"), nullable=False)
    posts=relationship("Posts", back_populates="comments")

    def __repr__(self):
        return "<Comments(Comment='%s', Time_create='%s')>" % (
            self.Comment, self.Time_create)
```

Опис таблиці “Ratings”:

```
class Ratings(Base):
    __tablename__ = 'Ratings'
    RatingID = Column(Integer, Sequence('Ratings_RatingID_seq'), primary_key=True)
    Rating = Column(Text, nullable=False)
    Time_create = Column(Date, nullable=False)

    UserIDFK = Column('UserIDFK', Integer, ForeignKey('Users.UserID', ondelete="CASCADE", onupdate="CASCADE"), nullable=False)
    users=relationship("Users", back_populates="ratings")
    PostIDFK = Column('PostIDFK', Integer, ForeignKey('Posts.PostID', ondelete="CASCADE", onupdate="CASCADE"), nullable=False)
    posts=relationship("Posts", back_populates="ratings")

    def __repr__(self):
        return "<Ratings(Rating='%s', Time_create='%s')>" % (
            self.Rating, self.Time_create)
```

```
#####INSERT#####

def insert_into(table, columns, values, ind):
    try:
        add_list=[]
        col_length=len(columns)
        count=0
        for i in range(ind):
            obj=eval(table)()
            for j in range(col_length):
                setattr(obj,columns[j],values[count])
                count+=1
            add_list.append(obj)
        session.add_all(add_list)
        session.commit()
        return True
    except Exception as err:
        print("Error {}".format(err))
        session.rollback();
        return False
```

```
#####UPDATE Users#####

def update_table(samples,columns,values):
    col_length=len(columns)
    for i in range(col_length):
        for j in samples:
            setattr(j,columns[i],values[i])
    return samples
```

```
#####DELETE#####

def delete(table,where_cond):
    try:
        if len(where_cond)=='t':
            samples=session.query(eval(table)).delete()
        else:
            samples=session.query(eval(table)).filter(getattr(eval(table),where_cond[0])==where_cond[1]).delete()
        session.commit()
        return True
    except Exception as err:
        print("Error {}".format(err))
        return False
#####
```

```
♀Choose category:
```

```
1.Insert  
2.Update  
3.Delete  
4.Exit  
_
```

Insert:

```
♀Insert:(Write table to insert or 'Exit'):  
Users  
♀Insert:(Write columns to insert into):  
Name  
Email  
N_sub_  
_
```

```
Insert:  
Write values to insert 1:('string')  
Name_1  
Name_1@gmail.com  
.15  
_
```

```
Insert in 'Users' table in '['Name', 'Email', 'N_sub']'  
columns this ['Name_1', 'Name_1@gmail.com', 15] values
```

	UserID [PK] integer	Name name	Email character varying	N_sub integer
1	100020	Name_1	Name_1@gmail.com	15
2	100019	'Yehor'	'sdasd'	1234
3	100018	ed	LLLLLLLLLS	2225
4	100017	0f38a	da9a7	94

Update:

```
♀Update:(Write table to update or 'Exit'):  
Users  
♀Update:(Write columns and values):  
Name=Changed_name  
  
♀Update:(Write condition):("table_name",'string')  
Name=Name_1_
```

```
Update in 'Users' table and set ['Changed_name']  
by Name=Name_1 condition
```

Data Output				
	UserID [PK] integer	Name name	Email character varying	N_sub integer
1	100020	Changed_name	Name_1@gmail.com	15
2	100019	Yehor	'sdasd'	1234
3	100018	ed	LLLLLLLLS	2225
4	100017	0f38a	da9a7	94
5	100016	33377	fd380	99
6	100015	e079b	ed097	36

Delete:

Для перевірки Delete створимо для останнього користувача відповідні колонки в таблицях “Posts”, “Comments”, “Ratings”.


```

❏Insert:(Write table to insert or 'Exit'):
Posts
❏Insert:(Write columns to insert into):
PostID
Topic
Post
Time_create
UserIDFK

Insert:
Write values to insert 1:('string')
1
Some topic
Some post
2020-12-12
100020

```

	PostID [PK] integer	Topic character varying (50)	Post text	Time_create date	UserIDFK integer
1	1	Some topic	Some p...	2020-12-12	100020
2	20026	60eba	77f2fa...	2020-03-20	1
3	20027	74335	a85984...	2014-07-19	1
4	20028	9befe	af601d...	2014-10-17	1
5	20029	6451e	a77c1b...	2016-03-30	3
6	20030	286e3	fd0dbf...	2014-12-30	6
7	20031	4853c	f5e6a2...	2020-09-20	10
8	20032	77be4	bdd180...	2018-06-22	4

	RatingID [PK] integer	Rating integer	Time_create date	UserIDFK integer	PostIDFK integer
1	1	5	2020-12-12	100020	1
2	4	4	2020-01-05	4	20029
3	5	5	2020-01-06	4	20030
4	6	1	2020-01-08	5	20031

	CommentID [PK] integer	Comment text	Time_create date	UserIDFK integer	PostIDFK integer
1	1	Some comment	2020-12-12	100020	1
2	4	Com_4	2020-01-04	3	20029

```

?Delete:(Write table to delete from or 'Exit'):
Users
?Delete:
Write condition or press 't' to delete all
Press 'Enter' to return back:
UserID=100020
Delete in item(s) 'Users' table by UserID=100020 condition

```

1

select * from "Users" where "UserID" = 100020;

Data Output

	UserID [PK] integer	Name name	Email character varying	N_sub integer	

Query Editor						Expla
1	select * from "Posts" where "UserIDFK" = 100020;					Succ 0 ro
Data Output						
	PostID [PK] integer	Topic character varying (50)	Post text	Time_create date	UserIDFK integer	

Query Editor

1 select * from "Comments" where "UserIDFK" = 100020;

Data Output

	CommentID [PK] integer	Comment text	Time_create date	UserIDFK integer	PostIDFK integer

Query Editor

```
1 select * from "Ratings" where "UserIDFK" = 100020;
```

Data Output

	RatingID [PK] integer	Rating integer	Time_create date	UserIDFK integer	PostIDFK integer

2 пункт:

Види індексів - Hash, BRIN

Розглянемо *Hash* індекс:

Hash індекс може виконувати тільки просте порівняння (=).

Індекс не створено:

Query Editor

1 explain analyze select * from "Users";

Data Output

	QUERY PLAN	
	text	🔒
1	Seq Scan on "Users" (cost=0.00..2468.17 rows=100017 width=78) (...)	
2	Planning Time: 0.126 ms	
3	Execution Time: 30.623 ms	

При пошуку числа:

Query Editor		E
1	<code>drop index ind_hash;</code>	S
2	<code>create index ind_hash on "Users" using hash("N_sub");</code>	7
3	<code>explain analyze select * from "Users" where "N_sub"=100;</code>	
Data Output		
	QUERY PLAN text	🔒
1	Bitmap Heap Scan on "Users" (cost=19.74..1023.82 rows=483 width=78) (actual time=19.956....	
2	Recheck Cond: ("N_sub" = 100)	
3	Heap Blocks: exact=440	
4	-> Bitmap Index Scan on ind_hash (cost=0.00..19.62 rows=483 width=0) (actual time=19.891...	
5	Index Cond: ("N_sub" = 100)	
6	Planning Time: 1.304 ms	
7	Execution Time: 20.273 ms	

При пошуку рядка:

Query Editor	
1	<code>drop index if exists ind_hash;</code>
2	<code>create index ind_hash on "Users" using hash("Name");</code>
3	<code>explain analyze select * from "Users" where "Name" = 'Yehor';</code>
Data Output	
	QUERY PLAN text
1	Seq Scan on "Users" (cost=0.00..260.00 rows=1 width=78) (actual time=1.064..1.064 rows=0 lo...
2	Filter: ("Name" = 'Yehor'::name)
3	Rows Removed by Filter: 10000
4	Planning Time: 53.910 ms
5	Execution Time: 1.098 ms

При використанні іншої умови окрім «=» індекс не використовується.

Query Editor	
1	<code>drop index ind_hash;</code>
2	<code>create index ind_hash on "Users" using hash("N_sub");</code>
3	<code>explain analyze select * from "Users" where "N_sub"<100;</code>
Data Output	
	QUERY PLAN
	text
1	Seq Scan on "Users" (cost=0.00..2718.21 rows=99534 width=78) (actual time=0.034..14.045...
2	Filter: ("N_sub" < 100)
3	Rows Removed by Filter: 516
4	Planning Time: 1.717 ms
5	Execution Time: 16.273 ms

Розглянемо *Brin* індекс:

Brin індекс використовується для великих баз даних.

Навіть для найбільшої таблиці із завдання на 100000 рядків *Brin* індекс не використовується.

Query Editor	
1	<code>drop index if exists ind_hash;</code>
2	<code>create index ind_hash on "Posts" using Brin ("Time_create");</code>
3	<code>explain analyze select * from "Posts"</code>
4	<code>where "Time_create" between '2019-12-01' and '2020-01-01';</code>
Data Output	
	QUERY PLAN
	text
1	Seq Scan on "Posts" (cost=0.00..2236.00 rows=1353 width=28) (actual time=0.059..17.911 rows=...
2	Filter: (("Time_create" >= '2019-12-01'::date) AND ("Time_create" <= '2020-01-01'::date))
3	Rows Removed by Filter: 98742
4	Planning Time: 20.530 ms
5	Execution Time: 18.031 ms

Тому було створено таблицю на 8 000 000 рядків для перевірки роботи індекса.

Робота без індекса:

```
4 explain analyze select * from public.testtab
5 where date between '2019-08-07 ' and '2019-08-08';
6
```

Data Output	
	QUERY PLAN
	text
1	Gather (cost=1000.00..133474.28 rows=1 width=49) (actual time=676.222..688.481 rows=0 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on testtab (cost=0.00..132474.18 rows=1 width=49) (actual time=607.319..607.319 rows=...
5	Filter: ((date >= '2019-08-07 00:00:00'::timestamp without time zone) AND (date <= '2019-08-08 00:00:00'::ti...
6	Rows Removed by Filter: 2666667
7	Planning Time: 0.674 ms
8	Execution Time: 688.614 ms

Робота з індексом:

```
4 create index testtab_idx on testtab using brin(date);
5 explain analyze select * from public.testtab
6 where date between '2019-08-07 ' and '2019-08-08';
```

Data Output	
	QUERY PLAN
	text
1	Bitmap Heap Scan on testtab (cost=20.03..33406.84 rows=1 width=49) (actual time=0.118..0.118 rows=0 loops=1)
2	Recheck Cond: ((date >= '2019-08-07 00:00:00'::timestamp without time zone) AND (date <= '2019-08-08 00:00:00'::timestamp witho...
3	-> Bitmap Index Scan on testtab_idx (cost=0.00..20.03 rows=12403 width=0) (actual time=0.117..0.117 rows=0 loops=1)
4	Index Cond: ((date >= '2019-08-07 00:00:00'::timestamp without time zone) AND (date <= '2019-08-08 00:00:00'::timestamp with...
5	Planning Time: 1.053 ms
6	Execution Time: 0.136 ms

Brin повільніше в роботі, ніж btree індекс, але займає набагато менше місця.



Порівняння з btree:

4	<code>drop index if exists testtab_idx;</code>
5	<code>create index testtab_idx_btree on testtab using btree(date);</code>
6	<code>explain analyze select * from public.testtab</code>
Data Output	
	QUERY PLAN text
1	Index Scan using testtab_idx_btree on testtab (cost=0.43..8.45 rows=1 width=49) (actual time=0.006..0.006 rows=0 loops=1)
2	Index Cond: ((date >= '2019-08-07 00:00:00':timestamp without time zone) AND (date <= '2019-08-08 00:00:00':timestamp without tim.
3	Planning Time: 1.481 ms
4	Execution Time: 0.029 ms

Використання пам'яті:

```
7 select pg_size_pretty(pg_total_relation_size('testtab_idx')) as brin,
8 pg_size_pretty(pg_total_relation_size('testtab_idx_btree')) as btree;
9
```

Data Output

	 brin text	 btree text	
1	64 kB	171 MB	

Brin займає набагато менше пам'яті, при цьому, все одно достатньо сильно пришвидшує пошук в таблиці.

3 Пункт:

Умови для тригера - before delete, update

Тригер обновлює дані про кількість видалених полів “Posts” для кожного користувача.

```
create or replace function del_post()
returns trigger as
$del_post$
declare
    curs CURSOR FOR SELECT * FROM "Posts";
    row_1 RECORD;
begin
open curs;
loop
    FETCH from curs INTO row_1;
    EXIT WHEN NOT FOUND;
    if old."UserIDFK" = row_1."UserIDFK" then
        update "Users" set "post_del" = "post_del"+1 where row_1."UserIDFK" = "Users"."UserID";
        IF NOT FOUND THEN
            RAISE EXCEPTION 'CANNOT FIND USER WHERE % = %' ,row_1."UserIDFK","Users"."UserID" ;
        end if;
        EXIT;
    end if;
end loop;
close curs;
return old;
end;
$del_post$ language plpgsql;

drop trigger if exists d_post on "Posts";

create trigger d_post
before delete on "Posts"
for each row
execute procedure del_post();
```

Для перевірки роботи розглянемо кількість полів таблиці “Posts”, які посилаються на першого користувача.

```
33 select "UserID", "PostID" from "Users","Posts"
34 where "UserID" = 100021 and "UserID"="UserIDFK";
35
```




Data Output			
	UserID integer	PostID integer	
1	100021	135466	
2	100021	155888	
3	100021	156892	
4	100021	157002	
5	100021	162364	
6	100021	178260	
7	100021	186892	
8	100021	202851	
9	100021	206553	
10	100021	223900	
11	100021	130050	
12	100021	130051	
13	100021	130052	
14	100021	130053	
15	100021	130054	

При видаленні даних полів таблиці «Posts» поле "post_del" в таблиці "Users" повинно збільшитися на кількість видалених полів.

Видалимо всі поля таблиці "Posts", які посилаються на даного користувача.

```
17 delete from "Posts" where "UserIDFK" = 100021;
18 select "UserID", "PostID" from "Users","Posts"
19 where "UserID" = 100021 and "UserID"="UserIDFK";
```

Data Output

	UserID integer 	PostID integer 	

До виклику:

Data Output					
	UserID [PK] integer	Name name	Email character varying	N_sub integer	post_del integer
1	100021	da153	dffe7	57	0
2	100022	c8f69	3ac76	31	0
3	100023	a70f9	8a601	47	0
4	100024	f8f97	4af91	93	0

Після:

	UserID [PK] integer	Name name	Email character varying	N_sub integer	post_del integer
1	100021	da153	dffe7	57	15
2	100022	c8f69	3ac76	31	0
3	100023	a70f9	8a601	47	0
4	100024	f8f97	4af91	93	0
5	100025	78c14	4f864	85	0

Спробуємо видалити всі поля з “Posts”

	UserID [PK] integer	Name name	Email character varying	N_sub integer	post_del integer
1	100021	da153	dffe7	57	15
2	100022	c8f69	3ac76	31	7
3	100023	a70f9	8a601	47	8
4	100024	f8f97	4af91	93	14
5	100025	73a1e	4f994	95	11
6	100026	f3852	cccd9	75	11
7	100027	f198d	25b19	31	20
8	100028	e60fb	934d5	49	23
9	100029	94610	404ed	58	25
10	100030	3ba4d	dfb2b	49	19
11	100031	537d8	b0b6b	88	21

Всі значення в полі “post_del” змінилися, окрім першого користувача, бо ми вже видалили всі поля з “Posts”, які посилалися на нього.