

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

Долгов Олексій Юрійович
(прізвище, ім'я, по батькові)

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

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

Київ 2020

Постановка завдання

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.

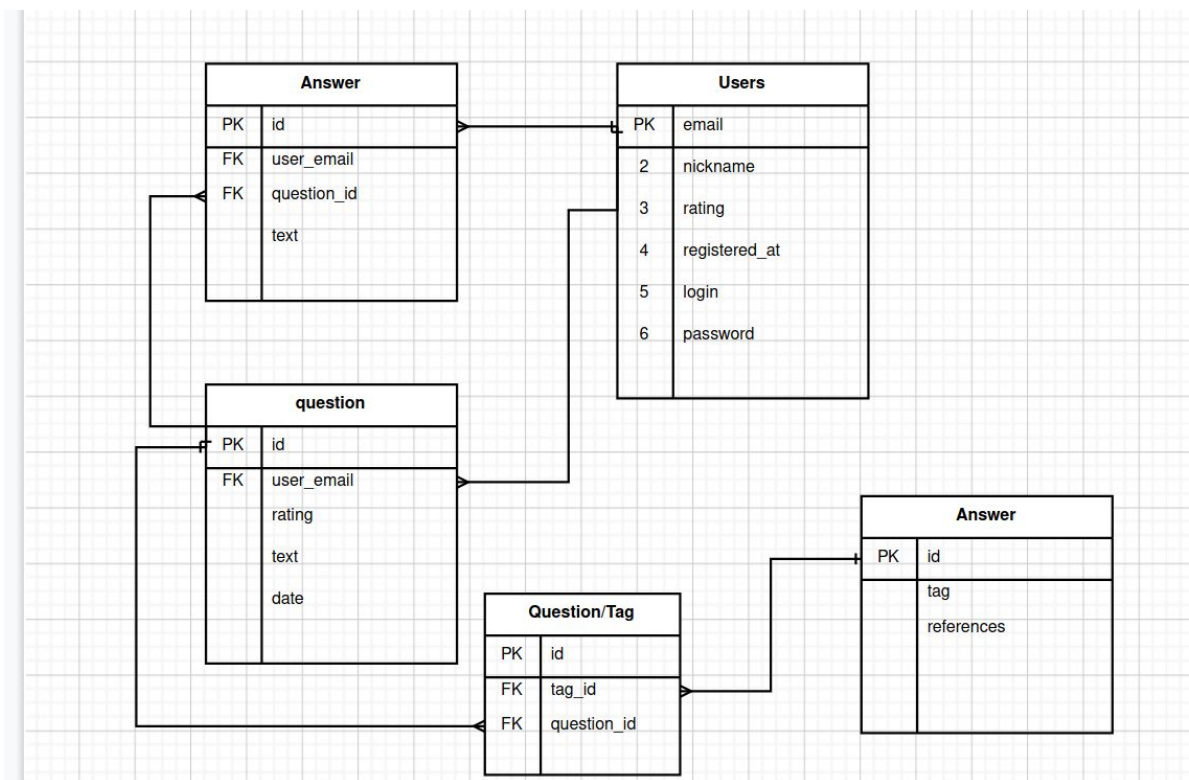
Варіант 8

Індекси: Btree, Gin

Тригер: After insert, update

Відповіді на вимоги

1) Схема бази даних



Класи ORM:

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

    id = Column(Integer, primary_key=True)
    text = Column(Text)
    rating = Column(Integer)
    user_email = Column(Text, ForeignKey('Users.email'))
    question_id = Column(Integer, ForeignKey('Question.id'))
    question = relationship("Question")
    user = relationship("User")
```

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

    id = Column(Integer, primary_key=True)
    text = Column(Text)
    rating = Column(Integer)
    user_email = Column(Text, ForeignKey("Users.email"))
    date = Column(Date)
    answer = relationship("Answer", cascade="all, delete-orphan")
```

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

    id = Column(Integer, primary_key=True)
    references = Column(Integer)
    tag = Column(Text)
```

```
class User(Base):
    __tablename__ = 'Users'

    email = Column(Text, primary_key=True)
    nickname = Column(Text)
    rating = Column(Integer)
    registered_at = Column(Date)
    role = Column(Text)
    login = Column(Text)
    password = Column(Text)
    answer = relationship("Answer", cascade="all, delete-orphan")
    question = relationship("Question", cascade="all, delete-orphan")
```

```
Question_Tag = Table('Question/Tag', Base.metadata,
    Column('id', Integer, primary_key=True),
    Column('question_id', Integer, ForeignKey('Question.id')),
    Column('tag_id', Integer, ForeignKey('Tag.id'))
)
```

Приклади запитів у вигляді ORM:

```

def add_new_instance(self, instance: all_tables):
    self._session.add(instance)

def delete_instance(self, instance, id):
    res = self._session.query(instance).filter(instance.id == id).all()
    if res:
        self._session.query(instance).filter(instance.id == id).delete()
        return True
    else:
        return False

def delete_user(self, email):
    user = self._session.query(all_tables.User).filter(all_tables.User.email == email).all()
    if user:
        self._session.query(all_tables.User).filter(all_tables.User.email == email).delete()
        return True
    else:
        return False

```

2) Індокси:

Створення:

1. Btree

```

1 CREATE INDEX btree_rating
2     ON public."Question" USING btree
3     (rating ASC NULLS LAST)
4 ;

```

2. Gin

```

1 CREATE INDEX text_gin_idx
2 ON "Question"
3 USING gin (to_tsvector('english', "Question".text));



```

Выполнение:

Бінарне дерево:

Без индекса (71 мс)

```
1 explain analyze select * from "Question" as q
2 where q.rating > 98
3 order by text;
4
```

	Data Output	Explain	Messages	Notifications
	 QUERY PLAN text 			
3	Workers Launched: 2			
4	-> Sort (cost=13350.95..13359.85 rows=3559 width=51) (actual time=53.322..5...			
5	Sort Key: text			
6	Sort Method: quicksort Memory: 667kB			
7	Worker 0: Sort Method: quicksort Memory: 940kB			
8	Worker 1: Sort Method: quicksort Memory: 552kB			
9	-> Parallel Seq Scan on "Question" q (cost=0.00..13141.02 rows=3559 width...			
10	Filter: (rating > 98)			
11	Rows Removed by Filter: 275798			
12	Planning Time: 0.076 ms			
13	Execution Time: 71.137 ms			

С индексом:(36.5 мс)

pgAdmin 4 interface showing a query execution plan for a query with an index. The query is: `explain analyze select * from "Question" as q where q.rating > 98 order by text;`. The execution plan shows a Bitmap Index Scan on btree_rating, which is highlighted in green. The total query runtime is 168 msec, and 10 rows are affected.

Без индекса (301 мс):

```
1 explain analyze select * from "Question" as q group by q.rating, q.id
2 having q.rating > (select count(*) % 100 from "Question" as qu where qu.rating = 25)
3 order by q.rating;
4
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
12	Rows Removed by Filter: 277172	
13	-> Group (cost=0.42..44217.46 rows=280001 width=51) (actual time=51.564..235.260 rows=11295...	
14	Group Key: q.id	
15	-> Index Scan using "Question_pkey" on "Question" q (cost=0.42..43517.46 rows=280001 width=...	
16	Filter: (rating > \$1)	
17	Rows Removed by Filter: 727048	
18	Planning Time: 0.200 ms	
19	Execution Time: 301.990 ms	

С индексом (249 мс):

```
1 explain analyze select * from "Question" as q group by q.rating, q.id
2 having q.rating > (select count(*) % 100 from "Question" as qu where qu.rating = 25)
3 order by q.rating;
4
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
13	-> Index Scan using "Question_pkey" on "Question" q (cost=0.42..43517.46 rows=280001 width=...	
14	Filter: (rating > \$0)	
15	Rows Removed by Filter: 727048	
16	Planning Time: 0.128 ms	
17	Execution Time: 249.607 ms	

Без индекса (957 мс):

```
1 explain analyze select * from "Question" as q group by q.rating, q.id
2 having q.rating > (select min(rating) from "Question")
3 order by q.rating;
4
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
10	-> Parallel Seq Scan on "Question" (cost=0.00..12266.01 rows=350001 width=4) (actu...	
11	-> Group (cost=0.42..44217.46 rows=280001 width=51) (actual time=55.177..424.164 rows=835...	
12	Group Key: q.id	
13	-> Index Scan using "Question_pkey" on "Question" q (cost=0.42..43517.46 rows=280001 width=...	
14	Filter: (rating > \$1)	
15	Rows Removed by Filter: 4186	
16	Planning Time: 0.119 ms	
17	Execution Time: 957.891 ms	

С индексом (898 мс)

```
1 explain analyze select * from "Question" as q group by q.rating, q.id
2 having q.rating > (select min(rating) from "Question")
3 order by q.rating;
4
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
10	Heap Fetches: 0	
11	-> Group (cost=0.42..44217.46 rows=280001 width=51) (actual time=0.025..367.370 rows=8358...	
12	Group Key: q.id	
13	-> Index Scan using "Question_pkey" on "Question" q (cost=0.42..43517.46 rows=280001 width=...	
14	Filter: (rating > \$1)	
15	Rows Removed by Filter: 4186	
16	Planning Time: 0.144 ms	
17	Execution Time: 898.774 ms	

Індекс Gin:

Без індексу (341 мс):

```
1 explain analyze select * from "Question" as q group by q.id
2 having q.id > (select avg(id) from "Question")
3 and q.id % 2 = 0
4 order by q.text desc
5 limit 25;
```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
20	-> Sort (cost=15792.81..15794.26 rows=583 width=51) (ac...	
21	Sort Key: q.id	
22	Sort Method: external merge Disk: 4792kB	
23	Worker 0: Sort Method: external merge Disk: 4744kB	
24	Worker 1: Sort Method: external merge Disk: 6328kB	
25	-> Parallel Seq Scan on "Question" q (cost=0.00..15766....	
26	Filter: (((id)::numeric > \$1) AND ((id % 2) = 0))	
27	Rows Removed by Filter: 198449	
28	Planning Time: 0.183 ms	
29	Execution Time: 341.368 ms	

З індексом (365 мс):

1	explain analyze select * from "Question" as q group by q.id
2	having q.id > (select avg(id) from "Question")
3	and q.id % 2 = 0
4	order by q.text desc
5	limit 25;

Data Output	Explain	Messages	Notifications
	QUERY PLAN text		
26		Filter: (((id)::numeric > \$1) AND ((id % 2) = 0))	
27		Rows Removed by Filter: 198449	
28		Planning Time: 0.164 ms	
29		Execution Time: 365.324 ms	

Висновки щодо результатів:

Індекс Btree зміг прискорити час виконання запитів, оскільки дозволив шукати записи за їх рейтингом зі складністю $\log(n)$ а не за n , як було без індексів. Пришвидшення запитів стає особливо помітним, коли ми шукаємо відносно невелику кількість записів, і чим більшу кількість записів ми шукаємо, тем менш ефективним стає індекс.

На відміну від Btree, індекс GIN не зміг прискорити виконання запитів. Це є наслідком того, що тексти в таблиці згенеровані випадково, через що ми маємо велику кількість унікальних елементів, для яких використання цього індексу є недоцільним.

3) Тригер

Текст тригерної функції:

```
4 BEGIN
5     if OLD.text = NEW.text then
6         raise EXCEPTION 'Old text cannot be same as new';
7     end if;
8     FOR r IN
9         SELECT * FROM "Question" WHERE user_email = NEW.user_email
10    LOOP
11        if r.date = CURRENT_DATE then
12            qty = qty + 1;
13        end if;
14
15    END LOOP;
16    if qty > 5 then
17        raise NOTICE 'You will not receive rating boost because you have already posted 5 questions';
18    elsif qty > 2 then
19        update "Users" set rating = (rating + 10)
20        where email = NEW.user_email;
21        raise NOTICE 'You received 10 rating!';
22    end if;
23    -- code
24    RETURN NEW;
```

При додаванні нового питання користувачем тригер додає 10 рейтингу, але додає не більш ніж за 5-ть заданих користувачем питань за день.

При апдейті видає помилку, якщо новий текст питання повністю збігається зі старим текстом

Перед додаванням:

	email [PK] text	nickname text	rating integer	registered_at date	role text	login text	password text
1	0	new	0	2020-11-02	User	orm	e2e42a07550...
2	111@gmail...	Usss	0	2020-09-20	User	Usss	1111
3	123121@gm...	Badfdf	0	2020-09-20	User	Foo	2222
4	444@gmail...	444	0	2020-11-02	User	444	<md5 HASH o...
5	orm@gmail...	fdsa	100	2020-11-18	usefdsar	fdsa	fdsa
6	ormser@gm...	orms	0	2020-11-18	User	orms	698d51a19d8...
7	sfdsfdsads...	Bashar	0	2020-09-20	Admin	Bar	3333

Текст додавання:

```
1 insert into "Question" (text, rating, user_email, date)
2 values ('ffsfdsfdfsdas', 12, 'ormser@gmail.com', CURRENT_DATE)
```

Після додавання:

	email [PK] text	nickname text	rating integer	registered_at date	role text	login text	password text
1	0	new	0	2020-11-02	User	orm	e2e42a07550...
2	111@gmail....	Usss	0	2020-09-20	User	Usss	1111
3	123121@gm...	Badfdf	0	2020-09-20	User	Foo	2222
4	444@gmail....	444	0	2020-11-02	User	444	<md5 HASH o...
5	orm@gmail....	fdsa	100	2020-11-18	usefdsar	fdsa	fdsa
6	ormser@gm...	orms	10	2020-11-18	User	orms	698d51a19d8...
7	sfdsfdsads...	Bashar	0	2020-09-20	Admin	Bar	3333

Оновлення (доданого в попередньому запиті запису):

```
1 update "Question" set text = 'ffsfdsfdfsdas'
2 where id = 1355953
3
```

Data Output Explain Messages Notifications

ERROR: old text cannot be same as new

Контрольні запитання

1. Сформулювати призначення та задачі об'єктно-реляційної проекції (ORM).
2. Проаналізувати основні види індексів у PostgreSQL (*BTree*, *BRIN*, *GIN*, *Hash*): призначення, сфера застосування, переваги та недоліки.
3. Пояснити призначення тригерів та функцій у базах даних.

1) Призначенням ORM є полегшення праці з базою даних і пришвидшення розробки модулів, які працюють за базами даних шляхом абстрагування від мови sql і наданням інтерфейсу, який дозволяє позбутися від написання аналогічних sql запитів для різних класів.

2) Призначення

Btree - прискорення роботи з даними, які можна відсортувати

GIn - для полнотекстового пошуку

Hash - створення хеш-таблиць для даних

Brin - прискорення пошуку строк, шляхом відкидання завідомо не підходящі строки

Сфера застосування і недоліки:

Btree - Будь-які дані, які можна відсортувати

Недолік - займає деяку кількість пам'яті

Gin - текстові дані, серед яких відносна невелика кількість унікальних лексем

Недолік - повільне оновлення індексу, при додаванні нового документу

Hash - Для пришвидшення доступу до даних за ключем

Недолік - великий об'ємом займає пам'яті

Brin - Для пошуку строк в надвеликих текстових даних

Недолік - працює повільніше ніж B-дерево, але й займає менше місця. Через цю особливість для невеликих таблиць, використання B-дерева буде доцільнішим.