



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 2-го курсу
групи КП-93
Фещенко Єгор Олександрович

Перевірив
“” “вересня” 2020р.
викладач
Петрашенко Андрій Васильович

Київ 2020

Варіант 24

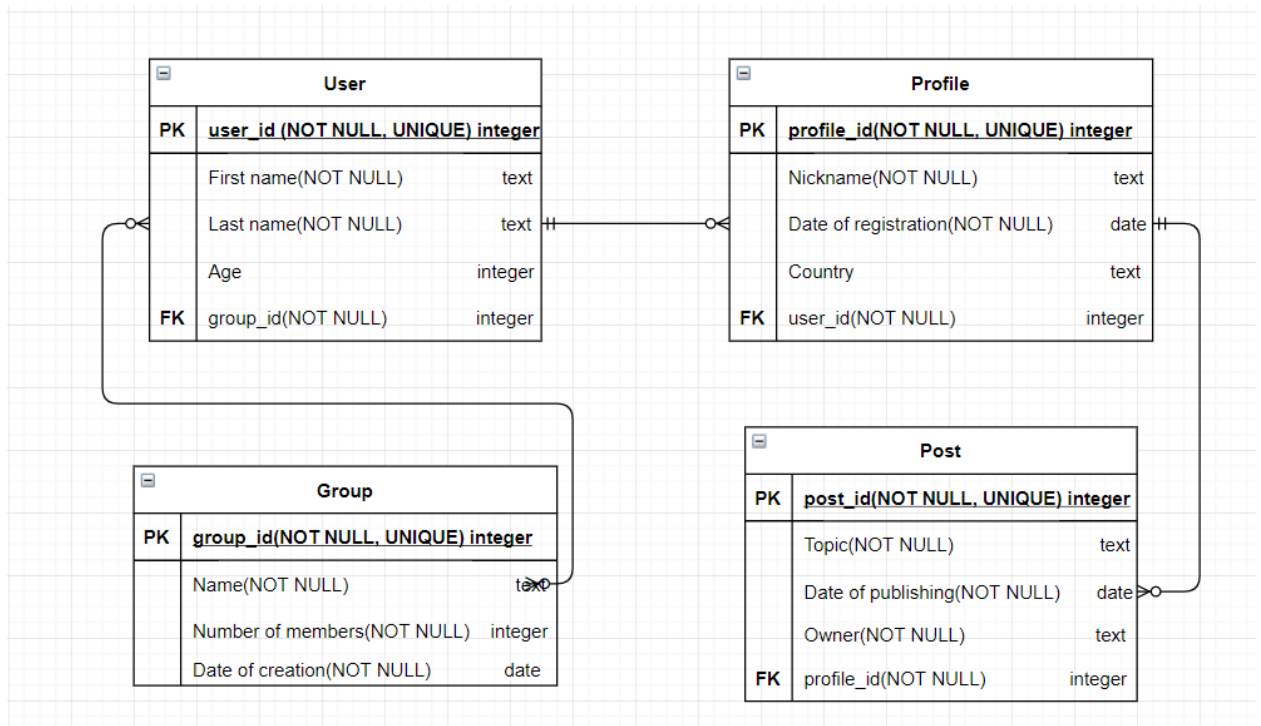
Загальне завдання роботи полягає у наступному:

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.

URL репозиторія (код):

URL репозиторія (звіт):

1) Схеми БД



Відповідні ORM класи

User

```

class User(Base):
    __tablename__ = "users"
    user_id = Column(Integer, primary_key=True, nullable=False)
    first_name = Column(Text, nullable=False)
    last_name = Column(Text, nullable=False)
    age = Column(Integer)
    children = relationship("Profile")

    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

```

Group

```

class Group(Base):
    __tablename__ = "groups"
    group_id = Column(Integer, primary_key=True, nullable=False)
    name = Column(Text, nullable=False)
    date_of_creation = Column(Date, nullable=False)
    number_of_members = Column(Integer, nullable=False)
    users = relationship("User", secondary="user_groups")

    def __init__(self, name, date_of_creation, number_of_members):
        self.name = name
        self.date_of_creation = date_of_creation
        self.number_of_members = number_of_members

```

Profile

```

class Profile(Base):
    __tablename__ = "profiles"
    profile_id = Column(Integer, primary_key=True)
    nickname = Column(Text, nullable=False)
    date_of_registration = Column(Date, nullable=False)
    country = Column(Text)
    user_id = Column(Integer, ForeignKey('users.user_id'), nullable=False)
    children = relationship("Post")

    def __init__(self, nickname, date_of_registration, country, user_id):
        self.nickname = nickname
        self.date_of_registration = date_of_registration
        self.country = country
        self.user_id = user_id

```

Post

```
class Post(Base):
    __tablename__ = "posts"
    post_id = Column(Integer, primary_key=True)
    topic = Column(Text, nullable=False)
    date_of_publishing = Column(Date, nullable=False)
    owner = Column(Text, nullable=False)
    profile_id = Column(Integer, ForeignKey('profiles.profile_id'), nullable=False)

    def __init__(self, topic, date_of_publishing, owner, profile_id):
        self.topic = topic
        self.date_of_publishing = date_of_publishing
        self.owner = owner
        self.profile_id = profile_id
```

Приклади ORM запитів:

```
def add_entity(self, new_entity):
    try:
        self.session.add(new_entity)
        self.session.commit()
    except (Exception, exc.DatabaseError, exc.InvalidRequestError) as error:
        self.session.execute('ROLLBACK')
        print(error)
```

```
def get_entity(self, entity_id):
    try:
        user = self.session.query(User).get(entity_id)
        self.session.commit()
    except (Exception, exc.DatabaseError, exc.InvalidRequestError) as error:
        print("Error occurred: " + error)
        self.session.execute("ROLLBACK")
    return user
```

```
def delete_entity(self, entity_id):
    try:
        self.delete_links(entity_id)
        self.session.query(User).filter_by(user_id=entity_id).delete()
        self.session.commit()
    except (Exception, exc.DatabaseError, exc.InvalidRequestError) as error:
        print("Error occurred: " + error)
        self.session.execute("ROLLBACK")
```

2)

Команда створення GIN індексу

(Оскільки дані в моїй БД не підходять під використання даного індексу було створено таблицю з потрібними полями)

Query EditorQuery History

```
1 create index song_indx on songs using gin (lyrics)
```

Query EditorQuery History

```
1 select * from songs where lyrics = '{b80a6c59b4d7974568a15052a6baca3b,be1bbb0ec15b6ce24d8521f96b34700b,6cdc8995424dec
2
```

Data OutputExplainMessagesNotifications

	name text	lyrics text[]	
1	8263bad...	{b80a6c59b4d7974568a15052a6baca3b,b...	
2	e6ed2e6...	{b80a6c59b4d7974568a15052a6baca3b,b...	
3	beab3d5...	{b80a6c59b4d7974568a15052a6baca3b,b...	
4	656f3026...	{b80a6c59b4d7974568a15052a6baca3b,b...	
5	abb997f0...	{b80a6c59b4d7974568a15052a6baca3b,b...	
6	a40bce0f...	{b80a6c59b4d7974568a15052a6baca3b,b...	
7	d092e45...	{b80a6c59b4d7974568a15052a6baca3b,b...	
8	11b75fcb...	{b80a6c59b4d7974568a15052a6baca3b,b...	
9	262c057...	{b80a6c59b4d7974568a15052a6baca3b,b...	

Data OutputExplainMessagesNotifications

	QUERY PLAN text	
1	Seq Scan on songs (cost=0.00..681.00 rows=10000 width=417)	
2	Filter: (lyrics = '{b80a6c59b4d7974568a15052a6baca3b,be1bbb0ec15b6...	

```
1 select * from songs where lyrics[3] < 'BB' and lyrics[4] < 'SS' order by name
```

Data Output Explain Messages Notifications

	name text	lyrics text[]
1	0005849...	{b80a6c59b4d7974568a15052a6baca3b,b...
2	000665e...	{b80a6c59b4d7974568a15052a6baca3b,b...
3	002270a...	{b80a6c59b4d7974568a15052a6baca3b,b...
4	0022cc7...	{b80a6c59b4d7974568a15052a6baca3b,b...
5	00239d9...	{b80a6c59b4d7974568a15052a6baca3b,b...
6	00265ac...	{b80a6c59b4d7974568a15052a6baca3b,b...
7	002a43f9...	{b80a6c59b4d7974568a15052a6baca3b,b...
8	003dabd...	{b80a6c59b4d7974568a15052a6baca3b,b...
9	00425bb...	{b80a6c59b4d7974568a15052a6baca3b,b...
10	0048aec...	{b80a6c59b4d7974568a15052a6baca3b,b...

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Sort (cost=762.20..764.98 rows=1111 width=417)
2	Sort Key: name
3	-> Seq Scan on songs (cost=0.00..706.00 rows=1111 width=417)
4	Filter: ((lyrics[3] < 'BB'::text) AND (lyrics[4] < 'SS'::text))

Команда створення BRIN індексу(Оскільки дані в моїх таблицях не підходять під раціональне використання даного індексу, була створена таблиця із 20000 записами поточного часу та рандомного цілого числа)

Query Editor Query History

```
1 create index prod_indx on products using brin("time")
```

```
1 select * from products where "time" < '20:00' group by "time", products
```

	<div>time</div> <div>time with time zone</div>	<div>products</div> <div>integer</div>	
1	00:02:55.614444+02:00	138	
2	03:48:58.778635+02:00	155	
3	05:37:29.531675+02:00	0	
4	01:17:53.226754+02:00	1	
5	04:38:32.629130+02:00	283	
6	03:28:59.969865+02:00	1	
7	02:24:33.186336+02:00	0	
8	05:55:27.643481+02:00	285	
9	00:51:52.088668+02:00	0	
10	05:53:00.785055+02:00	1	

Query EditorQuery History

1

`explain select * from products where "time" < '20:00' group by "time", products`

Data OutputExplainMessagesNotifications

	<div>QUERY PLAN</div> <div>text</div> <div></div>	
1	HashAggregate (cost=435.07..557.05 rows=12198 width=16)	
2	Group Key: "time", products	
3	-> Bitmap Heap Scan on products (cost=15.08..374.08 rows=12198 width=16)	
4	Recheck Cond: ("time" < '20:00:00+02':time with time zone)	
5	-> Bitmap Index Scan on prod_indx (cost=0.00..12.03 rows=20000 width=0)	
6	Index Cond: ("time" < '20:00:00+02':time with time zone)	

Query Editor

Query History

1

```
select * from products where "time" < '21:00' and "time" > '20:45' order by products
```

Data Output

Explain

Messages

Notifications

	<div>time</div> <div>time with time zone</div>	<div>products</div> <div>integer</div>	
1	20:52:54.984471+02:00	0	
2	20:52:44.413048+02:00	0	
3	20:55:48.149510+02:00	0	
4	20:46:59.732959+02:00	0	
5	20:59:20.623602+02:00	0	
6	20:52:40.039299+02:00	0	
7	20:52:08.211415+02:00	0	
8	20:46:19.534862+02:00	0	
9	20:48:13.900163+02:00	0	
10	20:51:27.144719+02:00	0	
11	20:50:28.395792+02:00	0	
12	20:47:19.400390+02:00	0	
13	20:56:49.859882+02:00	0	
14	20:51:54.107835+02:00	0	
15	20:48:06.074739+02:00	0	
16	20:57:43.910063+02:00	0	
17	20:56:36.452407+02:00	0	
18	20:51:23.841627+02:00	0	
19	20:46:28.146710+02:00	0	

Query Editor

Query History

1 explain select * from products where "time" < '21:00' and "time" > '20:45' order by products

Data Output

Explain

Messages

Notifications

▲

QUERY PLAN

text

🔒

1

Sort (cost=442.32..443.51 rows=476 width=16)

2

Sort Key: products

3

-> Bitmap Heap Scan on products (cost=12.15..421.15 rows=476 width=...

4

Recheck Cond: (("time" < '21:00:00+02':time with time zone) AND ("ti...

5

-> Bitmap Index Scan on prod_indx (cost=0.00..12.03 rows=20000 wi...

6

Index Cond: (("time" < '21:00:00+02':time with time zone) AND ("ti...

У даних випадках індекси прискорюють виконання запиту, оскільки були обрані раціональні поля для індексування.

GIN індексує не атомарні дані, а ті, які складаються із декількох елементів. Індексуються окремі елементи. Доречно використовувати його, наприклад у повнотекстовому пошуку.

Натомість BRIN створює «сторінки» значень де помічає мінімальне та максимальне значення на сторінці, що пришвидчує пошук відсортованих даних, та даних де можна чітко виділити діапазон на мін-макс. Недоречно використовувати якщо дані неможливо укласти у сторінки(розкиданий діапазон значень).

3) Текст тригерної функції after update:

```
declare
    c cursor for
    select first_name, last_name, age from users;
BEGIN
    IF new.age > 150 or new.age < 0 then
        raise notice 'invalid age';
        new.age = 18;
    end if;
    for record in c loop
        if record.first_name = record.last_name then
            raise notice 'Notice: first and last names are the same';
        end if;
    end loop;
    return new;
end;
```

Інформація до виклику тригера:

	user_id [PK] integer	first_name text	last_name text	age integer
1	1	Jack	Sparrow	30

Зміни:

```
Data Output  Explain  Messages  Notifications
ЗАМЕЧАНИЕ: invalid age
UPDATE 1

Query returned successfully in 141 msec.
```

	user_id [PK] integer	first_name text	last_name text	age integer
1	1	Jack	Sparrow	18

Текст тригерної функції insert:

```

declare
  c cursor for
    select first_name, last_name, age from users;
BEGIN
  for record in c loop
    IF new.age < 18 then
      raise notice 'User is not allowed! Too young';
      return null;
    end if;
    if new.first_name = 'Banned name' OR new.last_name = 'Banned name' then
      raise notice 'Error: first or last names are inadmissible!';
      return null;
    end if;
  end loop;
  return new;
end;

```

Інформація до виклику тригера:

```

1 insert into users (first_name, last_name, age) values('John', 'Brown', 10),
2 ('Banned name', 'White', 30)

```

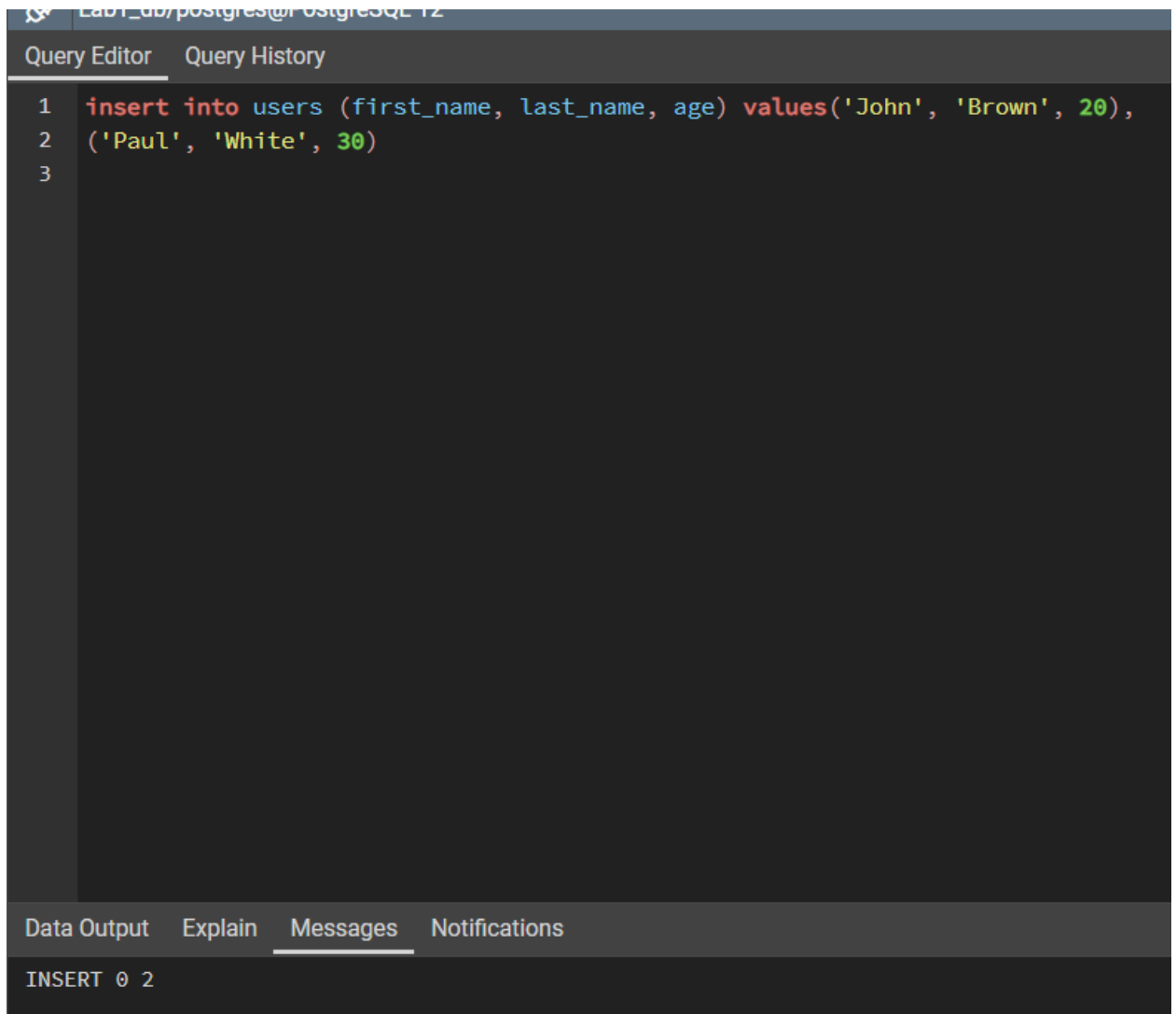
Зміни:

```

ЗАМЕЧАНИЕ: User is not allowed! Too young
ЗАМЕЧАНИЕ: Error: first or last names are inadmissible!
INSERT 0 0

```

Правильний запит:



The screenshot shows a PostgreSQL Query Editor window. The title bar indicates the connection is to 'Lab1_db/postgres@postgres12'. The 'Query Editor' tab is active, displaying an SQL insert statement across three lines: '1 insert into users (first_name, last_name, age) values('John', 'Brown', 20),', '2 ('Paul', 'White', 30)', and '3'. The bottom status bar shows 'Data Output', 'Explain', 'Messages' (which is selected), and 'Notifications'. Below this, it displays 'INSERT 0 2'.

```
1 insert into users (first_name, last_name, age) values('John', 'Brown', 20),
2 ('Paul', 'White', 30)
3
```

Messages

INSERT 0 2

Контрольні питання:

- 1) **ORM** – це технологія програмування, яка створює віртуальну базу даних, яка є відображенням таблиць баз даних у формі класів. Виконує функцію перетворення таблиць у об'єкти класів та навпаки, для зручного використання у ОО мовах програмування. Також спрощує використання методів для роботи із БД.
- 2) Для **Btree** раціонально використовувати поля з великим «розкидом» значень для зручної побудови бінарного дерева, особливістю якого є константний час пошуку елемента у гілках. Недоречно його використовувати його при малої кількості даних та упорядкованих даних.
Натомість **BRIN** створює «сторінки» значень де помічає мінімальне та максимальне значення на сторінці, що пришвидчує пошук відсортованих даних, та даних де можна чітко виділити

діапазон на мін-макс. Недоречно використовувати якщо дані неможливо укласти у сторінки(розкиданий діапазон значень). **GIN** індексує не атомарні дані, а ті, які складаються із декількох елементів. Індесуються окремі елементи. Доречно використовувати його, наприклад у повнотекстовому пошуку **Hash** – хеш-таблиця , яка використовує функції хешування для створення індексу, який відповідає індексованому значенню. Доречно використовувати при пошуку та порівнянні великих значень

- 3) **Тригери** викликаються БД при певних обставинах (before update , after delete тощо) для обробки виключних ситуацій, виконання необхідних дій після\до дії, для запису та створення нових таблиць для подальшого аналізу даних та надання додаткової інформації від БД. Тригери прив'язують функції до виконання певних запитів