



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих
комп'ютерних систем**

Лабораторна робота №3

з дисципліни

«Бази даних і засоби управління»

Виконав: студент III курсу

ФПМ групи КВ-83

Пащенко Антон

Київ – 2020

Засоби оптимізації роботи СУБД PostgreSQL

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.

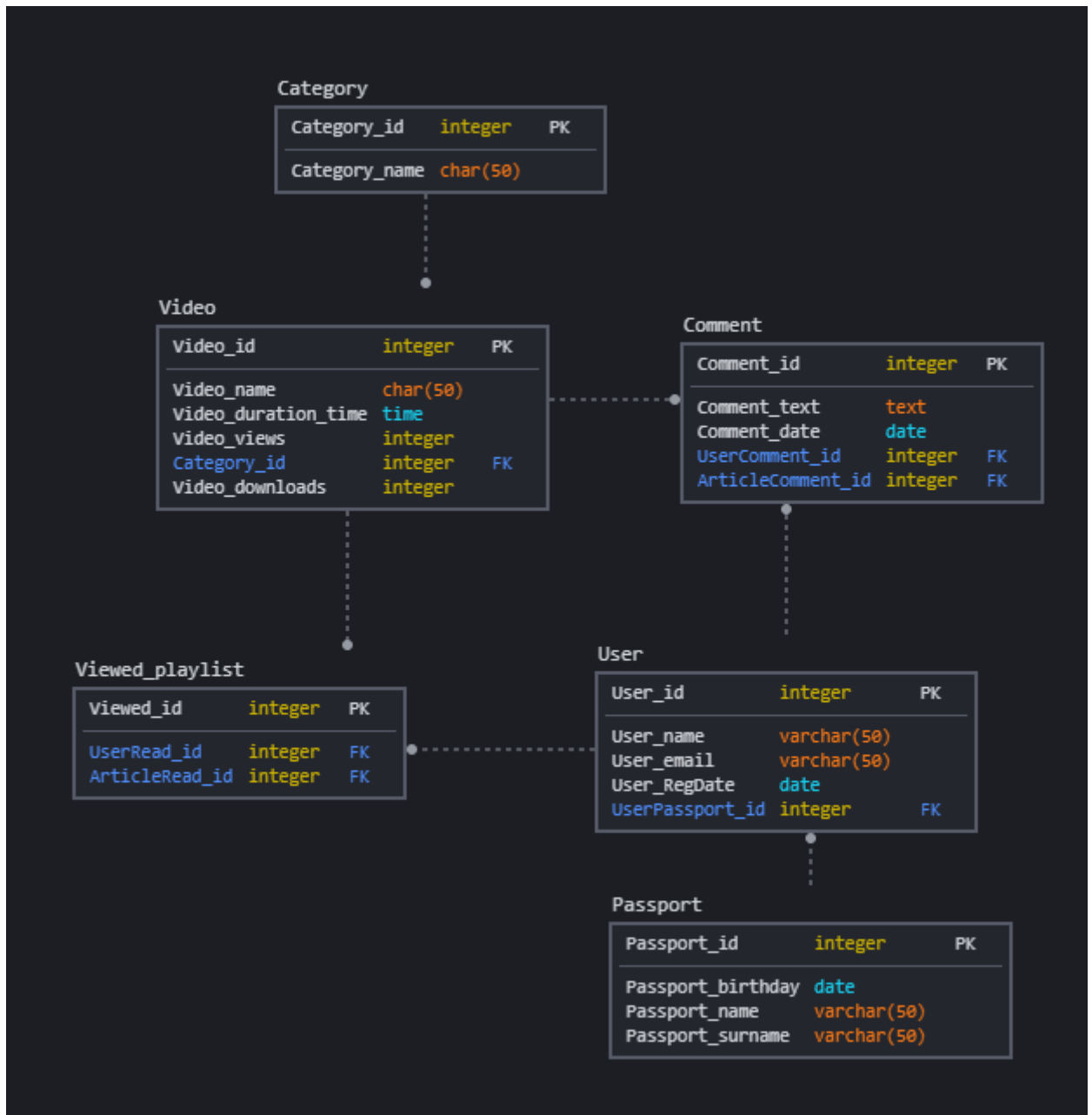
Варіант 19

У другому завданні проаналізувати індекси BTREE, BRIN.

Умова для тригера – before delete, insert

Завдання 1

Логічна схема бази даних “Відеохостінг”



Для перетворення модулю “Model” програми, створеної в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі використовую бібліотеку sqlalchemy.

Зобразимо сутнісні класи програми:

```
from sqlalchemy.ext.declarative import declarative_base
```

```
from sqlalchemy import Column, Integer, String, Date, Text, DateTime, ForeignKey
```

```
from sqlalchemy.orm import relationship
```

```
import datetime
```

```
Base = declarative_base()
```

```
class Category(Base):
```

```
    __tablename__ = 'category'
```

```
    id = Column('category_id', Integer, primary_key=True)
```

```
    category_name = Column(String(50))
```

```
    def __init__(self, data):
```

```
        self.category_name = data['category_name']
```

```
    def change_data(self, data):
```

```
        for key, value in data.items():
```

```
            setattr(self, key, value)
```

```
    def to_tuple(self):
```

```
        return (self.id, self.category_name)
```

```
    def get_column_names():
```

```
        return ['category_id', 'category_name']
```

```
class Comentary(Base):
```

```
    __tablename__ = 'commentaries'
```

```
    id = Column('comment_id', Integer, primary_key=True)
```

```
    comment_text = Column(Text)
```

```
    comment_to_video = Column(Integer, ForeignKey('video.video_id'))
```

```
    comment_from_user = Column(Integer, ForeignKey('users.user_id'))
```

```
    video = relationship("Video", backref="commentaries")
```

```
    user = relationship("User", backref="commentaries")
```

```
    def __init__(self, data):
```

```
self.comment_text = data['comment_text']
self.comment_to_video = int(data['comment_to_video'])
self.comment_from_user = int(data['comment_from_user'])
```

```
def change_data(self,data):
```

```
    for key,value in data.items():
```

```
        if key == 'comment_to_video':
```

```
            self.comment_to_video = int(value)
```

```
        elif key == 'comment_from_user':
```

```
            self.comment_from_user = int(value)
```

```
        else:
```

```
            setattr(self,key,value)
```

```
def to_tuple(self):
```

```
    return (self.id,self.comment_text,self.comment_to_video,self.comment_from_user)
```

```
def get_column_names():
```

```
    return ['comment_id','comment_text','comment_to_video','comment_from_user']
```

```
class Passport(Base):
```

```
    __tablename__ = 'passport'
```

```
    id = Column('passport_id',Integer, primary_key=True)
```

```
    passport_name = Column(String(50))
```

```
    passport_surname = Column(String(50))
```

```
    passport_date = Column(Date)
```

```
    def __init__(self,data):
```

```
        self.passport_name = data['passport_name']
```

```
        self.passport_surname = data['passport_surname']
```

```
        self.passport_date = datetime.datetime.strptime(data['passport_date'], '%Y-%m-%d').date()
```

```

def change_data(self,data):

    for key,value in data.items():

        if key == 'passport_date':
            self.passport_date = datetime.datetime.strptime(value, '%Y-%m-%d').date()
        else:
            setattr(self,key,value)

def to_tuple(self):
    return (self.id,self.passport_name,self.passport_surname,self.passport_date)

def get_column_names():
    return ['passport_id','passport_name','passport_surname','passport_date']

class User(Base):
    __tablename__ = 'users'
    id = Column('user_id',Integer, primary_key=True)
    user_name = Column(String(50))
    user_email = Column(String(50))
    user_regdate = Column(Date)
    user_passport_id = Column(Integer, ForeignKey('passport.passport_id',
ondelete='CASCADE'))
    passport =
relationship("Passport",cascade="all,delete",backref="user",passive_deletes=True)
    def __init__(self,data):
        self.user_name = data['user_name']
        self.user_name = data['user_name']
        self.user_email = data['user_email']
        self.user_regdate = datetime.datetime.strptime(data['user_regdate'], '%Y-%m-%d').date()
        self.user_passport_id = int(data['user_passport_id'])

    def change_data(self,data):

        for key,value in data.items():

```

```

        if key == 'user_regdate':
            self.user_regdate = datetime.datetime.strptime(value, '%Y-%m-%d').date()
        elif key == 'user_passport_id':
            self.user_passport_id = int(value)
        else:
            setattr(self, key, value)

    def to_tuple(self):
        return
        (self.id, self.user_name, self.user_email, self.user_regdate, self.user_passport_id)

    def get_column_names():
        return ['user_id', 'user_name', 'user_email', 'user_regdate', 'user_passport_id']

class Video(Base):
    __tablename__ = 'video'

    id = Column('video_id', Integer, primary_key=True)
    video_name = Column(String(50))
    video_duration_time = Column(DateTime(timezone=False))
    video_views = Column(Integer)
    video_downloads = Column(Integer)
    video_category = Column(Integer, ForeignKey('category.category_id'))
    category = relationship("Category", backref="video")

    def __init__(self, data):
        self.video_name = data['video_name']
        self.video_duration_time = datetime.datetime.strptime(data['video_duration_time'],
'%H:%M:%S').time()
        self.video_views = int(data['video_views'])
        self.video_downloads = int(data['video_downloads'])
        self.video_category = int(data['video_category'])

    def change_data(self, data):

```

```

for key,value in data.items():

    if key == 'video_duration_time':
        self.video_duration_time = datetime.datetime.strptime(value,
'%H:%M:%S').time()
    elif key == 'video_views':
        self.video_views = int(value)
    elif key == 'video_downloads':
        self.video_downloads = int(value)
    elif key == 'video_category':
        self.video_category = int(value)
    else:
        setattr(self,key,value)

def to_tuple(self):
    return
(self.id,self.video_name,self.video_duration_time,self.video_views,self.video_downloads,self.vide
o_category)

def get_column_names():
    return
['video_id','video_name','video_duration_time','video_views','video_downloads','video_category']

```

Завдання 2

Створення та аналіз індекса BТREE

Для дослідження індексу була використана таблиця video та поле . Вони проіндексовані як BТREE. У таблицю було занесено 100000 записів. Виконаний запит.

SELECT * FROM video WHERE video_views = 50

До індексування.

	<div>QUERY PLAN</div> <div>text</div> <div></div>
1	Seq Scan on video (actual time=0.023..0.993 rows=114 loops...
2	Filter: (video_views = 50)
3	Rows Removed by Filter: 9893
4	Planning Time: 0.104 ms
5	Execution Time: 1.013 ms

Після індексування.

	<div>QUERY PLAN</div> <div>text</div> <div></div>
1	Bitmap Heap Scan on video (actual time=0.054..0.111 rows=1...
2	Recheck Cond: (video_views = 50)
3	Heap Blocks: exact=60
4	-> Bitmap Index Scan on btree (actual time=0.045..0.045 row...
5	Index Cond: (video_views = 50)
6	Planning Time: 1.308 ms
7	Execution Time: 0.142 ms

Створення та аналіз індекса BRIN

Індекс BRIN використовують на даних значення яких відповідає їх розташуванню в пам'яті. Була використана таблиця що має поле log_timestamp що зберігає час отримання даних. В таблицю занесено 31536001 кортежей. Виконаний запит:

```
SELECT AVG(temperature) FROM temperature_log
WHERE log_timestamp>='2016-04-04' AND log_timestamp<'2016-04-05';
```

До індексування

	<div> <div>QUERY PLAN</div> <div>text</div> </div> <div></div>
1	Finalize Aggregate (cost=399055.45..399055.46 rows=1 width=32) (...)
2	-> Gather (cost=399055.23..399055.44 rows=2 width=32) (actual ti...
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Partial Aggregate (cost=398055.23..398055.24 rows=1 width=...
6	-> Parallel Seq Scan on temperature_log (cost=0.00..397967....)
7	Filter: ((log_timestamp >= '2016-04-04 00:00:00'::timestam...
8	Rows Removed by Filter: 10483200
9	Planning Time: 0.156 ms
10	Execution Time: 2058.969 ms

Після індексування.

	<div> <div>QUERY PLAN</div> <div>text</div> </div> <div></div>
1	Finalize Aggregate (cost=287054.86..287054.87 rows=1 width=32) (actual time=83.14...
2	-> Gather (cost=287054.64..287054.85 rows=2 width=32) (actual time=22.746..86.71...
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Partial Aggregate (cost=286054.64..286054.65 rows=1 width=32) (actual time=...
6	-> Parallel Bitmap Heap Scan on temperature_log (cost=54.33..285966.41 row...
7	Recheck Cond: ((log_timestamp >= '2016-04-04 00:00:00'::timestamp withou...
8	Rows Removed by Index Recheck: 4693
9	Heap Blocks: lossy=640
10	-> Bitmap Index Scan on idx_temperature_log_log_timestamp (cost=0.00..3...
11	Index Cond: ((log_timestamp >= '2016-04-04 00:00:00'::timestamp withou...
12	Planning Time: 1.703 ms
13	Execution Time: 86.795 ms

Завдання 3

Тригер:

```
create or replace function save_sum() returns trigger as $$
declare
  s integer := 0;
  viewed record;
begin
  for viewed in select video_views as count from video loop
    s := s + viewed.count;
  end loop;
  assert s > 0, 'sum of views cannot be lesser than zero';
  insert into view_sum(before_video,sum_views) values (new.video_name,s);
  return new;
end;
$$ language plpgsql;

create trigger save_views before insert on video for each row execute procedure save_sum()
```

Принцип роботи:

Тригер спрацьовує перед вставкою в таблицю video та зберігає суму переглядів всіх відео до вставки та назву відео в таблицю view_sum.

Вставляємо запис у video:



```
INSERT INTO video VALUES (DEFAULT,'TEST','21:31:43',1000,10,2)
```

view_sum:

id [PK] integer	before_video character varying (50)	sum_views integer
1	TEST	498414

Перевіримо суму:

```
1 SELECT sum(video_views) FROM video
```

Data Output		Explain	Messages	Notifications
	sum bigint 			
1	499414			

$$499414 - 1000 = 498414 \Rightarrow \text{Все вірно}$$