МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота № 2

з дисципліни “ Бази даних і засоби управління”

Засоби оптимізації роботи СУБД PostgreSQL

Виконав
студент III курсу
групи КВ-84
Кривко Євген Олегович

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.

Завдання 1

Представлення таблиць у вигляді класів

```
class Film(Base):
    __tablename__ = 'Films'
    id = Column(Integer, primary_key=True)
    name_f = Column(String(30))
    year_f = Column(Integer)
    genre_f = Column(String(30))
    duration_f = Column(Integer)

    def __repr__(self):
        return "<Films('%s', '%s', '%s', '%s')>" % (self.name_f, self.year_f, self.genre_f, self.duration_f)
```

```
class Cinema(Base):
    __tablename__ = 'Cinemas'
    id = Column(Integer, primary_key=True)
    name_c = Column(String(30))
    street = Column(String(30))

    def __repr__(self):
        return "<Cinemas('%s', '%s')>" % (self.name_c, self.street)
```

```
schedule = Table(
    'Schedule', Base.metadata,
    Column('id', Integer, primary_key=True),
    Column('id_s', Integer, ForeignKey('Sessions.id')),
    Column('id_c', Integer, ForeignKey('Cinemas.id'))
)


class Session(Base):
    __tablename__ = 'Sessions'
    id = Column(Integer, primary_key=True)
    id_f = Column(Integer, ForeignKey('Films.id'))
    start_date = Column(Date)
    hall_name = Column(String(30))
    film = relationship("Film", uselist=False)
    cinemas = relationship("Cinema", secondary=schedule)
```

Функції для створення одного/декількох записів

```
    # Створити запис
    def create_item(self, table_name, columns, item):

        obj = self._tables[table_name]()
        for i in range(len(columns)):
            obj.__dict__[columns[i]] = item[i]
        self._session.add(obj)
        self._session.commit()

    # Створити декілька записів
    def create_items(self, table_name, columns, items):

        for j in range(len(items)):
            obj = self._tables[table_name]()
            for i in range(len(columns)):
                obj.__dict__[columns[i]] = items[j][i]
            self._session.add(obj)
        self._session.commit()
```

Функції для зчитування записів

```
# Взяти дані про запис з бази
def read_item(self, table_name, columns, item_id):
    col_names = []
    tbl_entity = self._tables[table_name]
    for i in range(len(columns)):
        col_names.append(tbl_entity.__dict__[columns[i]])

    query = self._session.query(*col_names).filter(tbl_entity.id == item_id)
    return query.all()

# Прочитати дані таблиці з бази
def read_items(self, table_name, columns):
    tbl_entity = self._tables[table_name]
    if columns is not None:
        col_names = []
        for i in range(len(columns)):
            col_names.append(tbl_entity.__dict__[columns[i]])
        query = self._session.query(*col_names)
    else:
        query = self._session.query(tbl_entity)
    return query.all()
```

Функції оновлення та видалення записів

```python
# Оновити запис
def update_item(self, table_name, columns, item, item_id):

    tbl_entity = self._tables[table_name]

    update_values = dict(zip(columns, item))
    self._session.query(tbl_entity) \
        .filter(tbl_entity.id == item_id) \
        .update(update_values)

    self._session.commit()

# Видалити запис за ключем
def delete_item(self, table_name, item_id):
    tbl_entity = self._tables[table_name]
    self._session.query(tbl_entity).filter(tbl_entity.id == item_id).delete()
    self._session.commit()
```

Завдання 2

№ варіанта	*Види індексів*
14	*Btree, Hash*

Btree

Створимо 100000 випадкових рядків в таблиці cinemas(id_c, name_c,street).

```
1  select count(*) from cinemas
```

Data Output | Explain | Messages | Notifications

	count bigint
1	100000

```
9  select * from cinemas limit 5 offset 10;
```

Data Output | Explain | Messages | Notifications

	id_c [PK] integer	name_c character varying (30)	street character varying (30)
1	11	iz90BcMCQgwnfV	h3IhJ7MoOvqEhGk
2	12	xOp053KFDk4gdCM	QMXAOsZxFb42GfG
3	13	PVty19Fq2yKm6ik	HHWBpVELvJEamOW
4	14	XgclVPSbziaZRlp	SpwbitWivzryq4j
5	15	vtEdEb21eiLpuBj	H7Ts1QcrH9duChm

Виконаємо запит без індексу по стовпчику name_c запису 'iz90BcMCQgwnfV'

```
11  explain select * from cinemas where "name_c" = 'iz90BcMCQgwnfV'
12
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on cinemas (cost=0.00..2084.00 rows=1 width=34)
2	Filter: ((name_c)::text = 'iz90BcMCQgwnfV'::text)

Створимо індекс “cinemas_btree” для таблиці “cinemas” по стовпчику “name_c”:

```
13  drop index if exists "cinemas_btree";
14  create index "cinemas_btree" on "cinemas" using btree("name_c");
15
16
17
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 1 secs 173 msec.
```

Виконаємо пошук 'iz90BcMCQgwnfV' знову.

```
11  explain select * from cinemas where "name_c" = 'iz90BcMCQgwnfV'
12
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using cinemas_btree on cinemas (cost=0.42..8.44 rows=1 width=34)
2	Index Cond: ((name_c)::text = 'iz90BcMCQgwnfV'::text)

Бачимо, що пошук з індексом працює набагато швидше.

Hash

Для цього завдання знову використаємо таблицю cinemas(id_c, name_c,street). Слід зазначити, що індекс hash найкраще підходить для пошуку з використанням порівняння на “=”. Візьмем стовпчик street.

```
1  select count(*) from cinemas
```

Data Output | Explain | Messages | Notifications

	count bigint
1	100000

```
1  select * from cinemas limit 4 offset 30000
```

Data Output | Explain | Messages | Notifications

	id_c [PK] integer	name_c character varying (30)	street character varying (30)
1	30001	S6PFcV5Br05PCZI	pJcDMTd5FUcNxum
2	30002	3Zny2PaewQjuVMe	avKUaT98jEg1tZa
3	30003	VYdWI1TxsJ3exPB	7smGNlQzsHZwdtg
4	30004	oRzHhQ1jYIkakuU	PVpB2IZ5umpbrvt

Виберемо одне з значень – "pJcDMTd5FUcNxum". Зробимо пошук по цьому імені.

```
1  explain select * from cinemas where "street" = 'pJcDMTd5FUcNxum'
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on cinemas (cost=0.00..2084.00 rows=1 width=34)
2	Filter: ((street)::text = 'pJcDMTd5FUcNxum'::text)

Створимо індекс “cinemas_hash” для таблиці “cinemas” по стовпчику “street”:

```
6  create index "cinemas_hash" on "cinemas" using hash("street");
7
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 340 msec.
```

Виконаємо пошук "pJcDMTd5FUcNxum" знову.

```
1  explain select * from cinemas where "street" = 'pJcDMTd5FUcNxum'
2
3
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using cinemas_hash on cinemas (cost=0.00..8.02 rows=1 width=34)
2	Index Cond: ((street)::text = 'pJcDMTd5FUcNxum'::text)

Подивившись на результати пошуку, можна сказати , що використання індексу дало досить значне підвищення швидкодії.

Завдання 3

№ варіанта	*Умови для тригера*
14	*after insert, update*

Створимо таблицю cinemas_logs в яку будуть вставлятись записи про те, що відбувся insert or update.

```
create table cinemas_logs(
id serial primary key,
log_text text)
```

Команди створення тригеру та підключення до таблиці cinemas

```
create or replace function trigger_for_cinemas() returns trigger as $$
begin
  if (tg_op = 'INSERT') then
    if (char_length(new.name_c)<2)or(char_length(new.name_c)>30) then
    raise exception 'Yours date is wrong';
    return null;
    end if;
    insert into cinemas_logs(log_text) values(concat('inserted in cinemas: name_c = ', new.name_c ,' street = ', new.street));
    raise notice 'Successfull inserted';
    return new;
  elsif tg_op = 'UPDATE' then
    insert into cinemas_logs(log_text) values('updated row in table cinemas');
  raise notice 'Successfull updated';
    return new;
  else return null;
  end if;
end;
$$language plpgsql;

create trigger trigger_for_cinemas after insert or update on public.cinemas
for each row execute procedure trigger_for_cinemas()
```

Виконаємо insert в таблицю cinemas

```
insert into cinemas(name_c,street) values('aaaaaaaaaaaaaaaa','bbbbbbbbbbbbbbbbb')
```

Data Output | Explain | Messages | Notifications

```
ПОВІДОМЛЕННЯ:  Successfull inserted
INSERT 0 1

Query returned successfully in 113 msec.
```

Виконаємо update в таблиці cinemas

```
update cinemas set name_c = 'ccccccccccccccccc' where name_c = 'aaaaaaaaaaaaaaaa'
```

Data Output | Explain | Messages | Notifications

```
ПОВІДОМЛЕННЯ:  Successfull updated
UPDATE 1

Query returned successfully in 115 msec.
```

Введемо закоротке ім’я для вставки, отримаємо помилку:

```
31  insert into cinemas(name_c,street) values('a','bbbbbbbbbbbbbbbbbb')
32
```

Data Output | Explain | Messages | Notifications

```
ERROR: ПОМИЛКА:  Yours date is wrong
CONTEXT:  Функція PL/pgSQL trigger_for_cinemas() рядок 5 в RAISE


SQL state: P0001
```

Перевіримо таблицю logs

```
39  select* from cinemas_logs
40
41
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	log_text text
1	14	inserted in cinemas: name_c = aaaaaaaaaaaaaaaa street = bbbbbbbbbbbbbbbbb
2	15	updated row in table cinemas

Бачимо, що появились відповідні записи після запитів insert та update. Також бачимо, що при вводі некоректного імені запису в таблиці немає. Це говорить про те, що тригер працює правильно.