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»

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав:
студент групи КВ-81
Далечин Владислав
Перевірив:
Петрашенко А.В.

Київ 2020

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL

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.

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 результуючих 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 17

1 7	*GIN, BRIN*	*before update, delete*

Завдання 1

Query Editor | Query History

```
1  SELECT * FROM public."Cinemas"
2  ORDER BY id ASC
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	c_name character varying (20)	address character varying (40)
1	1	Kiev	Velyka Vasylkivska Street, 19
2	2	October	Konstantinovskaya Street, 26
3	3	Torch	Mykola Bazhana Avenue, 3

```
1  SELECT * FROM public."Cinema_Session"
2  ORDER BY id ASC
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	session_id integer	cinema_id integer
1	1	1	1
2	2	3	3
3	3	2	2

```
1  SELECT * FROM public."Films"
2  ORDER BY id ASC
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	f_name character varying (20)	release_year integer	f_genre character varying (10)	f_duration integer
1	1	InterStellar	2014	Fantastic	168
2	2	Joker	2019	Drama	116
3	3	Gentlemen	2019	Criminal	113
4	7	film1	2020	Comedy	[null]
5	8	film4	2020	[null]	[null]
6	9	film5	2020	[null]	[null]

```
1  SELECT * FROM public."Sessions"
2  ORDER BY id ASC
```

Data Output | Explain | Messages | Notifications

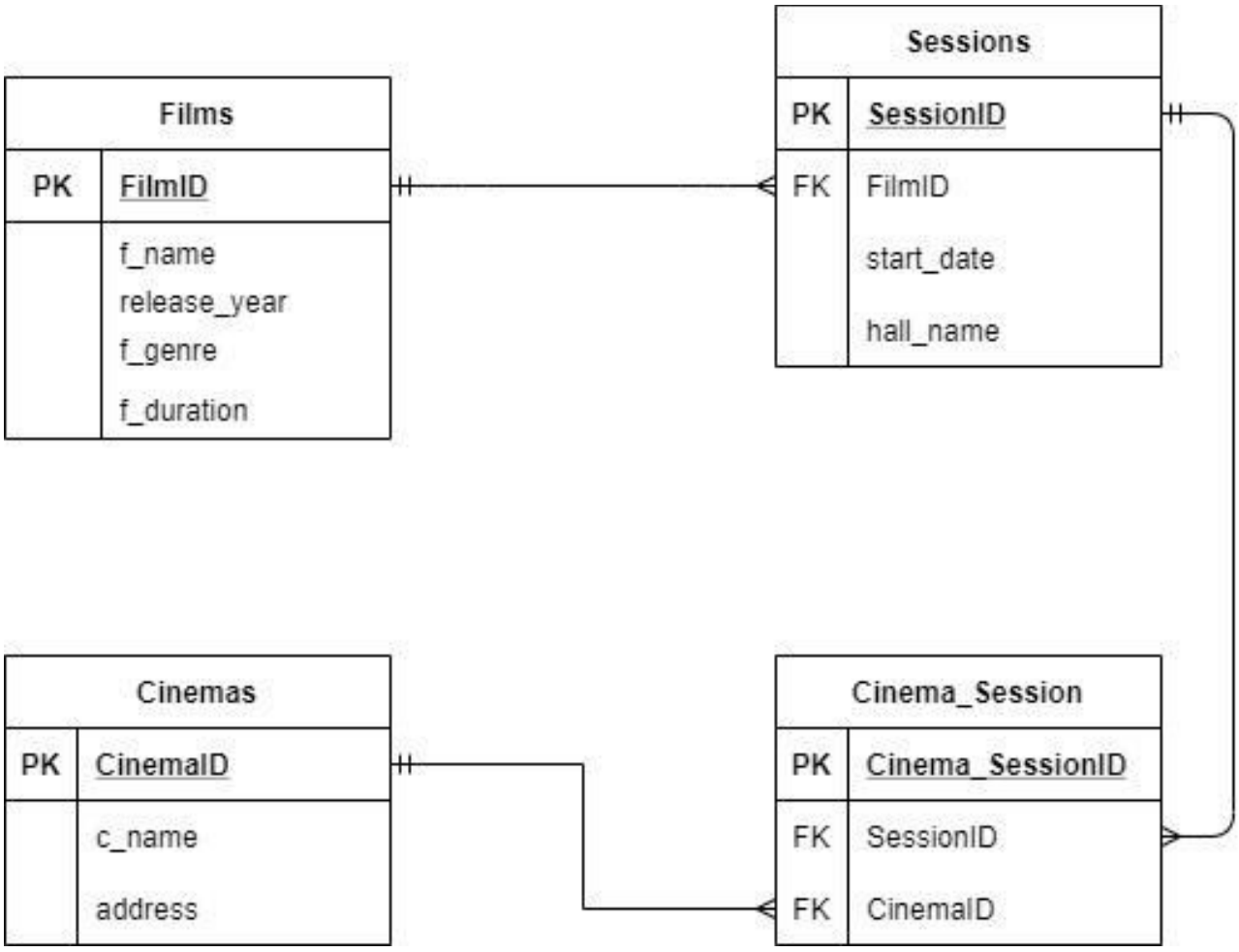
	id [PK] integer	start_date date	hall_name character varying (20)	film_id integer
1	1	2020-09-17	Almandine	1
2	2	2020-09-17	Ultramarine	2
3	3	2020-09-18	Terracotta	3
4	10	2020-11-25	Emerald	8

Query Editor | Query History

```
1  SELECT * FROM public.logs
2
```

Data Output | Explain | Messages | Notificatio

	text text	date_added timestamp without time zone
1	UPDAT...	2020-11-26 03:27:29.555228
2	DELET...	2020-11-26 03:27:45.342952



Зміст файлу base.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine =
create_engine('postgresql://postgres:root@localhost:5432/t
ickets')
Session = sessionmaker(bind=engine)

Base = declarative_base()
```

Перетворений модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM). Функції, що виконують вставку, вилучення, модифікації та отримання необхідних даних.

Деякі основні функції з файлу orm_model.py

```
import datetime
import sqlalchemy
from sqlalchemy import and_

import base
from cinema import Cinema
from film import Film
from session import Session


  class ORMModelPostgre(object):

      def __init__(self, conn_parameters=None):

          self._tables = {}
          self._foreign_keys = []
          self._session = base.Session()

      # Додати таблиці до моделі
      def add_tables(self, tables):
          for table in tables:
```

```
22.             self._tables[table.__tablename__] = table
23.
24.     # В л а с т и в і с т ь , щ о м і с т и т ь н а з в и
   т а б л и ц ь
25.     @property
26.     def tables(self):
27.         return list(self._tables.keys())
28.
29.     # Д о д а т и д а н і п р о з о в н і ш н і й к л ю ч
30.     def add_foreign_key(self, key_parameters):
31.         if len(key_parameters) != 4:
32.             return
33.         if (key_parameters.get('fk_table') is not None
34.                 and key_parameters.get('fk_column') is not None
35.                 and key_parameters.get('ref_table') is not None
36.                 and key_parameters.get('ref_column')):
37.             self._foreign_keys.append(key_parameters)
38.
39.     # В л а с т и в і с т ь , щ о м і с т и т ь д а н і п р о
   з о в н і ш н і к л ю ч і
40.     @property
41.     def foreign_keys(self):
42.         return self._foreign_keys
43.
44.     # В з я т и н а з в и с т о в п і в з т а б л и ц і
45.     def get_columns(self, table_name):
46.         return self._tables[table_name].__table__.columns.keys()
47.
48.     # В з я т и т и п с т о в п ц я з т а б л и ц і
49.     def get_column_type(self, table, column_name):
50.         for c in self._tables[table].__table__.columns:
51.             if column_name == c.name:
52.                 return c.type.python_type
53.
54.     # В з я т и т и п и с т о в п ц і в з т а б л и ц і
55.     def get_column_types(self, table):
56.         # return self._tables[table].__table__.columns.keys()
57.         return [c.type.python_type for c in
   self._tables[table].__table__.columns]
58.
59.     # Г е н е р у в а т и р я д к о в і д а н і
60.     def generate_str(self, quantity, str_len):
61.
62.         if str_len <= 0:
```

```
63.             return ''
64.         op = 'chr(trunc(65 + random()*25)::int)'
65.         parameters = op
66.         for i in range(1, str_len):
67.             parameters += ' || ' + op
68.
69.         query = 'SELECT {0} from
  generate_series({1},{2})'.format(parameters, 1, quantity)
70.         print(query)
71.         cursor = self._conn.cursor()
72.         cursor.execute(query)
73.         str_res = [str0[0] for str0 in cursor.fetchall()]
74.         return str_res
75.
76.     # Генерувати числові дані
77.     def generate_numbers(self, quantity, max_value):
78.
79.         query = 'SELECT trunc(random()*{0})::int from
  generate_series({1},{2})'.format(max_value, 1, quantity)
80.         print(query)
81.         cursor = self._conn.cursor()
82.         cursor.execute(query)
83.         numbers = [num[0] for num in cursor.fetchall()]
84.         return numbers
85.
86.
87.     # Генерувати дані, що містять дату
88.     def generate_date(self, quantity, days=90, shift=0,
  start_date=None):
89.
90.         if start_date is None:
91.             start_date = "NOW()"
92.         query = "select to_char(NOW() + (random() * (NOW() + '{0}
  days' - NOW())) + '{1} days', 'DD/MM/YYYY') " \
93.                 "from generate_series({2},{3})" \
94.             .format(days, shift, 1, quantity)
95.         cursor = self._conn.cursor()
96.         cursor.execute(query)
97.         dates = [date0[0] for date0 in cursor.fetchall()]
98.         return dates
99.
100.    # Фільми, що будуть показувати після
  дати та тривалістю у рамках заданого
```

```
    def search_query2(self, after_date, min_duration, max_duration):
        date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
        query = self._session.query(Film.f_name, Film.f_duration, Cinema.c_name, Session.start_date) \
            .join(Session.film).join(Session.cinemas) \
            .filter(Film.f_duration.between(min_duration, max_duration)) \
            .filter(Session.start_date > date_obj)
        return query.all()

    # Фільми, що будуть показувати після дати у кінотеатрі
    def search_query1(self, after_date, cinema):
        date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
        query = self._session.query(Film.f_name, Film.f_genre, Session.start_date, Session.hall_name) \
            .join(Session.film).join(Session.cinemas) \
            .filter(Cinema.c_name == cinema) \
            .filter(Session.start_date > date_obj)
        return query.all()

    # Фільми, назви яких містять словосполучення та відповідного жанру
    def search_query3(self, str_seq, str_genre):

        query = self._session.query(Film.f_name, Film.f_genre, Film.f_duration, Film.release_year)\
            .filter(and_(Film.f_name.ilike('%'+str_seq+'%'),
                         Film.f_genre.ilike('%'+str_genre+'%')))
        return query.all()

    def roll_back(self):
        self._session.rollback()

    # 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)
```

```
136.         self._session.commit()
137.
138.     # С т в о р и т и  д е к і л ь к а  з а п и с і в
139.     def create_items(self, table_name, columns, items):
140.
141.         for j in range(len(items)):
142.             obj = self._tables[table_name]()
143.             for i in range(len(columns)):
144.                 obj.__dict__[columns[i]] = items[j][i]
145.             self._session.add(obj)
146.         self._session.commit()
147.
148.     # В з я т и  д а н і  п р о  з а п и с  з  б а з и
149.     def read_item(self, table_name, columns, item_id):
150.         col_names = []
151.         tbl_entity = self._tables[table_name]
152.         for i in range(len(columns)):
153.             col_names.append(tbl_entity.__dict__[columns[i]])
154.
155.         query =
   self._session.query(*col_names).filter(tbl_entity.id == item_id)
156.         return query.all()
157.
158.     # П р о ч и т а т и  д а н і  т а б л и ц і  з  б а з и
159.     def read_items(self, table_name, columns):
160.         tbl_entity = self._tables[table_name]
161.         if columns is not None:
162.             col_names = []
163.             for i in range(len(columns)):
164.                 col_names.append(tbl_entity.__dict__[columns[i]])
165.             query = self._session.query(*col_names)
166.         else:
167.             query = self._session.query(tbl_entity)
168.         return query.all()
169.
170.     # О н о в и т и  з а п и с
171.     def update_item(self, table_name, columns, item, item_id):
172.
173.         tbl_entity = self._tables[table_name]
174.
175.         update_values = dict(zip(columns, item))
176.         self._session.query(tbl_entity) \
177.             .filter(tbl_entity.id == item_id) \
178.             .update(update_values)
```

```
179.
180.         self._session.commit()
181.
182.     # Видалити запис за ключем
183.     def delete_item(self, table_name, item_id):
184.         tbl_entity = self._tables[table_name]
185.         self._session.query(tbl_entity).filter(tbl_entity.id ==
  item_id).delete()
186.         self._session.commit()
187.
188.     # Закрити підключення до бази даних
189.     def __del__(self):
190.         self._session.close()
191.
192.
193. # Тестування модуля
194. if __name__ == '__main__':
195.     model = ORMModelPostgre()
196.     model.add_tables([Film, Cinema, Session])
197.
198.     model.update_item('Films', ['f_genre'], ['Comedy'], 7)
199.
200.     print(Film.__table__.c['f_name'].type)
201.     ttt = Film.__table__.c['f_name'].type.python_type
202.     if ttt is str:
203.         print('Yes it is')
204.
205.     print(convert(Film.__table__.c['f_name'].type))
206.     print(Film.__table__.c['f_name'].type.__class__)
207.     ttt2 = Session.__table__.c['film_id'].type.__class__
208.     print(str(ttt2))
209.     if ttt2 is sqlalchemy.Integer:
210.         print('Yes')
211.     print(ttt2)
212.
213.     print(model.get_column_types("Sessions"))
214.     print(model.get_columns("Sessions"))
215.
216.     print('-----')
217.     rows = model.search_query1('2020-09-16', 'Kiev')
218.     print(rows)
219.     print('-----')
220.     rows = model.search_query2('2020-09-16', 113, 116)
221.     print(rows)
```

```
222.      print('-----')
223.      rows = model.search_query3('e', 'a')
224.      print(rows)
225.      '''
226.      model = ORMModelPostgre()
227.      tbl_name = "Sessions"
228.      item0 = model.read_item(tbl_name, None, 10)
229.      print(item0)
230.      column = "testID"
231.
232.      column_type = model.get_column_type(tbl_name, column)
233.      print(column_type)
234.
235.      column_types = model.get_column_types(tbl_name)
236.      print(column_types)
237.
238.      model.generate_numbers(5, 10)
239.      model.generate_str(5, 5)
240.      model.generate_date(90, 30)
241.
242.      rows = model.search_query1('20-09-16', 'Kiev')
243.      print(rows)
244.
245.      rows = model.search_query2('20-09-16', 60, 116)
246.      print(rows)
247.
248.      rows = model.search_query3('er', 'Fan')
249.      print(rows)
250.      '''
```

Класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД.

Файл cinema.py

```
1. from sqlalchemy import Column, String, Integer
2.
3. from base import Base
4.
5.
6. class Cinema(Base):
7.     __tablename__ = 'Cinemas'
```

```
8.
9.      id = Column(Integer, primary_key=True)
10.        c_name = Column(String(20))
11.        address = Column(String(40))
12.
13.        def __repr__(self):
14.            return "<Cinemas('%s', '%s')>" % (self.c_name,
    self.address)
```

Файл film.py

```
1. from sqlalchemy import Column, String, Integer, Date
2.
3. from base import Base
4.
5.
6. class Film(Base):
7.      __tablename__ = 'Films'
8.
9.      id = Column(Integer, primary_key=True)
10.        f_duration = Column(Integer)
11.        f_name = Column(String(20))
12.        release_year = Column(Integer)
13.        f_genre = Column(String(10))
14.        def __repr__(self):
15.            return "<Films('%s', '%s', '%s', '%s')>" % (self.f_name,
    self.release_year, self.f_genre, self.f_duration)
```

Файл session.py

```
1. from sqlalchemy import Column, String, Integer, Date, ForeignKey,
   Table
2. from sqlalchemy.orm import relationship
3.
4. from base import Base
5.
6. cinema_session_association = Table(
```

```
    'Cinema_Session', Base.metadata,
    Column('id', Integer, primary_key=True),
    Column('session_id', Integer, ForeignKey('Sessions.id')),
       Column('cinema_id', Integer, ForeignKey('Cinemas.id'))
  )


  class Session(Base):
       __tablename__ = 'Sessions'

       id = Column(Integer, primary_key=True)
       start_date = Column(Date)
       hall_name = Column(String(20))
       film_id = Column(Integer, ForeignKey('Films.id'))
       film = relationship("Film", uselist=False)
       cinemas = relationship("Cinema",
  secondary=cinema_session_association)

       def __repr__(self):
           return "<Sessions('%s', '%s', '%s')>" % (self.start_date,
  self.hall_name, self.film.s_name)
```

Програма читає дані типів з бази даних і пропонує користувачу ввести відповідні дані. Приклад введення нових даних запису до таблиці Films.

```
Введіть значення поля FilmID
Тип поля число
111
Введіть значення поля f_name
Тип поля текст
Ford v Ferrari
Введіть значення поля release_year
Тип поля число
2019
Введіть значення поля f_genre
Тип поля текст
Sport
Введіть значення поля f_duration
Тип поля число
152
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Успіх! Запис з ідентифікатором 111 було додано до таблиці Films !
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Оновлення даних таблиці Films:

```
2
Введіть значення поля FilmID
Тип поля число
111
Введіть значення поля f_name
Тип поля текст
Ford v Ferrari
Введіть значення поля release_year
Тип поля число
2019
Введіть значення поля f_genre
Тип поля текст
Sport
Введіть значення поля f_duration
Тип поля число
153
---   ---   ---   ---   ---   ---   ---   ---   ---   ---   --
Запис 111 було змінено на [111, 'Ford v Ferrari', 2019, 'Sport', 153]
---   ---   ---   ---   ---   ---   ---   ---   ---   ---   --
Для продовження натисніть Enter...
```

Вилучення запису з таблиці Films:

```
1. Додати запис
2. Оновити дані запису
3. Відобразити дані
4. Видалити дані
5. Вихід
Виберіть один з пуктів...
4
Введіть ключ для запису який необхідно видалити
111
------------------------------------------------------------
Елемент 111 було успішно видалено!
------------------------------------------------------------
Для продовження натисніть Enter...
```

Перегляд записів:

```
Для продовження натисніть Enter...
1. Додати запис
2. Оновити дані запису
3. Відобразити дані
4. Видалити дані
5. Вихід
Виберіть один з пуктів...
3
['id', 'f_name', 'release_year', 'f_genre', 'f_duration']
--- ТАБЛИЦЯ FILMS ---
1. <Films('InterStellar', '2014', 'Fantastic', '168')>
2. <Films('Joker', '2019', 'Drama', '116')>
3. <Films('Gentlemen', '2019', 'Criminal', '113')>
4. <Films('film4', '2020', 'None', 'None')>
5. <Films('film5', '2020', 'None', 'None')>
6. <Films('film1', '2020', 'Comedy', 'None')>
```

Запити з минулої роботи на мові SQL було замінено на запити засобами SQLAlchemy по роботі з об’єктами.

Запит 1.

```
SELECT f_name, f_genre, start_date, hall_name ' \
     'FROM "Films", "Sessions", "Cinema_Session", "Cinemas" ' \
     'WHERE "Films"."FilmID" = "Sessions"."FilmID"' \
     'AND "Sessions"."SessionID" = "Cinema_Session"."SessionID"' \
     'AND "Cinema_Session"."CinemaID" = "Cinemas"."CinemaID"' \
     'AND "Cinemas".c_name = \'{}\' AND "Sessions".start_date > \'{}\' '

def search_query1(self, after_date, cinema):
    date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
    query = self._session.query(Film.f_name, Film.f_genre,
Session.start_date, Session.hall_name) \
        .join(Session.film).join(Session.cinemas) \
        .filter(Cinema.c_name == cinema) \
```

```
        .filter(Session.start_date > date_obj)
    return query.all()
```

Запит 2.

```
'SELECT f_name, f_duration, start_date, c_name ' \
    'FROM "Films", "Sessions", "Cinema_Session", "Cinemas" ' \
    'WHERE "Films"."FilmID" = "Sessions"."FilmID" ' \
    'AND "Sessions"."SessionID" = "Cinema_Session"."SessionID" ' \
    'AND "Cinema_Session"."CinemaID" = "Cinemas"."CinemaID" ' \
    'AND "Sessions"."start_date" > \'{}\' ' \
    'AND "Films".f_duration BETWEEN {} AND {}'

def search_query2(self, after_date, min_duration, max_duration):
    date_obj = datetime.datetime.strptime(after_date, '%Y-%m-%d').date()
    query = self._session.query(Film.f_name, Film.f_duration,
Cinema.c_name, Session.start_date) \
        .join(Session.film).join(Session.cinemas) \
        .filter(Film.f_duration.between(min_duration, max_duration)) \
        .filter(Session.start_date > date_obj)
    return query.all()
```

Запит 3.

```
'SELECT f_name, f_genre, f_duration, release_year ' \
    'FROM "Films"' \
    'WHERE ' \
    '"Films".f_name LIKE \'%{}%\' ' \
    'AND "Films".f_genre LIKE \'%{}%\' '


def search_query3(self, str_seq, str_genre):

    query = self._session.query(Film.f_name, Film.f_genre,
Film.f_duration, Film.release_year)\
        .filter(and_(Film.f_name.ilike('%'+str_seq+'%'),
                     Film.f_genre.ilike('%'+str_genre+'%')))
    return query.all()
```

Завдання 2

Для аналізу різних типів індексів у PostgreSQL було створено таблицю з двома полями.

```
CREATE TABLE test (
    num integer,
    txt character varying
);
```

Таблицю було заповнено 1000000 записів. Для заповнення таблиці було використано операції:

```
TRUNCATE test;
INSERT INTO test (num, txt)
SELECT
    floor(random() * 100 + 1)::int AS num,
    md5(random()::character varying)
FROM generate_series(1,1000000) ORDER BY num;
```

Для створення індексів GIN, BRIN по текстовому полю використовувалися такі операції:

```
CREATE INDEX txt_idx ON test USING gin (txt gin_trgm_ops);
CREATE INDEX txt_idx ON test USING brin (txt);
```

Для створення індексів по числовому полю для BRIN (для індексу GIN не розглядалося):

```
CREATE INDEX num_idx ON test USING brin (num);
```

Результат запиту SELECT * FROM test.

	num integer	txt character varying
1	1	54561684b90d9828fca25191bddfcdb7
2	1	1a9136e816ce92efc35676b2939b60c2
3	1	8b45410e8d5519b0aa6255610a5bbf9d
4	1	68266824bfcdd35c79a9688d1f2ec21e
5	1	720afd13d34ada88d294e62a486c14a7
6	1	3eedf38cddb68e438e54537b56510718
7	1	f11698db12b102e4bea2f5380432fc70
8	1	3222e6181dabb5318c3cab8c772e4855
9	1	fbae1b693c177b12a9de116e8aa8ba60
10	1	a970a6f80cb7bf98e52dc745b3c1f2ff

Було розглянуто таки запити:

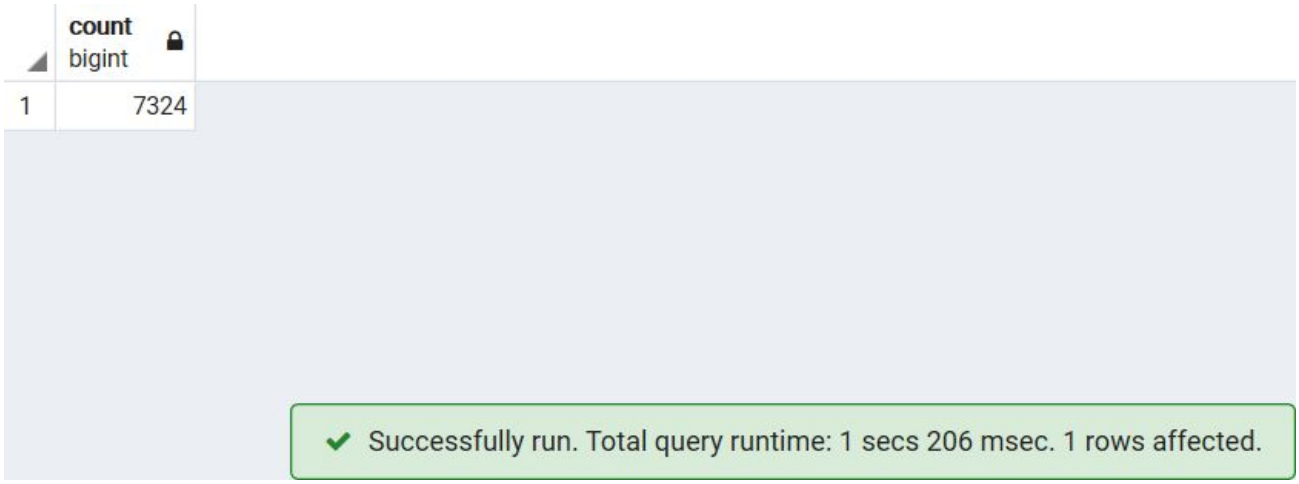
```
1. SELECT count(*) FROM test WHERE txt ILIKE '%abc%';

2. SELECT count(*) FROM test WHERE txt =
'34c6b4e97f3f5a72d8e1e7df9dbf0367';

3. SELECT txt FROM test ORDER BY txt

4. SELECT num FROM test WHERE num > 33

5. SELECT num, count(num) FROM test WHERE num > 70 GROUP BY
num
```

Кожен запит виконувався декілька разів (5 разів) для визначення більш точного результату. У таблиці наведено середній час виконання запиту.

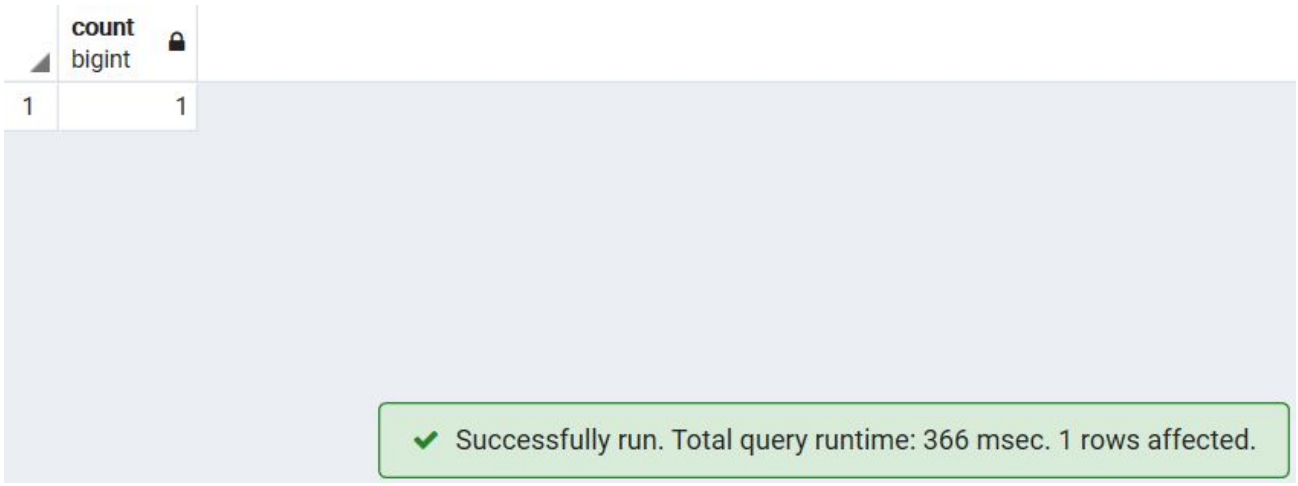
№ запиту	Без індексування	GIN	BRIN
1	1171,8	193	1131,4
2	306,4	273,6	286,4
3	4562,2	4239,6	4231,4
4	353,4		325,4
5	322,8		282

Виведення результуючих даних для варіанту без індексів:

```
1. SELECT count(*) FROM test WHERE txt ILIKE '%abc%';
```




```
2. SELECT count(*) FROM test WHERE txt =
'34c6b4e97f3f5a72d8e1e7df9dbf0367';
```



3. SELECT txt FROM test ORDER BY txt

	txt character varying
1	0000062f3d755e19afc941d9c9df649a
2	000029106c817ad5bff9085cb15d46ab
3	000039473d2082b3da186815c47e973b
4	00003f9fd53018fb2b24f3e3c92f7098
5	0000438c3bf509a72dee70b139b5eaa7
6	00005d841cea3226db01ede0d0f37b96
7	000
8	000

Successfully run. Total query runtime: 4 secs 327 msec. 1000000 rows affected.

4. SELECT num FROM test WHERE num > 33

	num integer
1	34
2	34
3	34
4	34
5	34
6	34
7	34
8	34

Successfully run. Total query runtime: 424 msec. 670626 rows affected.

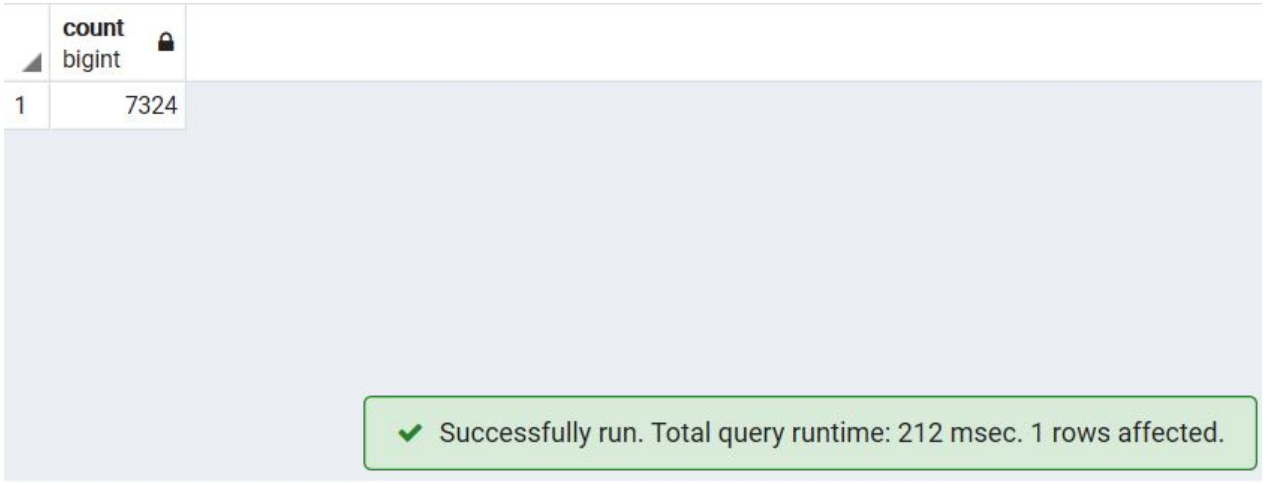
5. SELECT num, count(num) FROM test WHERE num > 70 GROUP BY

	num integer	count bigint
1	71	10145
2	72	9946
3	73	9914
4	74	10034
5	75	9997
6	76	9813
7	77	989
8	78	1003

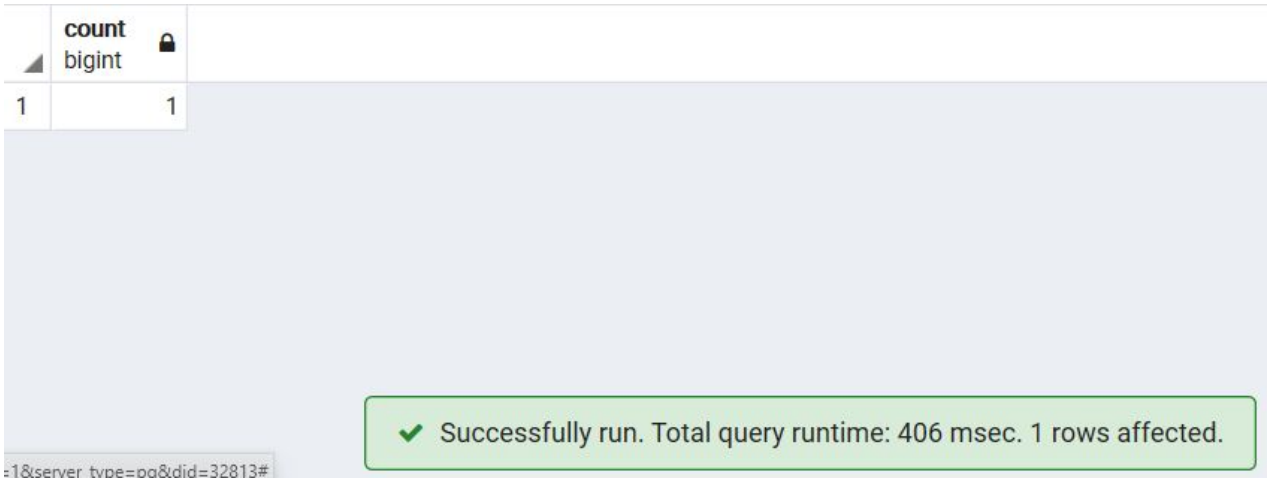
Successfully run. Total query runtime: 233 msec. 30 rows affected.

Варіант з індексом GIN:

```
1. SELECT count(*) FROM test WHERE txt ILIKE '%abc%';
```




```
2. SELECT count(*) FROM test WHERE txt =
'34c6b4e97f3f5a72d8e1e7df9dbf0367';
```




```
3. SELECT txt FROM test ORDER BY txt
```

	txt character varying
1	0000062f3d755e19afc941d9c9df649a
2	000029106c817ad5bff9085cb15d46ab
3	000039473d2082b3da186815c47e973b
4	00003f9fd53018fb2b24f3e3c92f7098
5	0000438c3bf509a72dee70b139b5eaa7
6	00005d841cea3226db01ede0d0f37b96
7	000
8	000

Successfully run. Total query runtime: 4 secs 835 msec. 1000000 rows affected.

Варіант з індексом BRIN:

```
1. SELECT count(*) FROM test WHERE txt ILIKE '%abc%';
```

	count bigint
1	7324

Successfully run. Total query runtime: 1 secs 220 msec. 1 rows affected.

```
2. SELECT count(*) FROM test WHERE txt =
'34c6b4e97f3f5a72d8e1e7df9dbf0367';
```

	count bigint
1	1

Successfully run. Total query runtime: 410 msec. 1 rows affected.

3. SELECT txt FROM test ORDER BY txt

	txt character varying
1	0000062f3d755e19afc941d9c9df649a
2	000029106c817ad5bff9085cb15d46ab
3	000039473d2082b3da186815c47e973b
4	00003f9fd53018fb2b24f3e3c92f7098
5	0000438c3bf509a72dee70b139b5eaa7
6	00005d841cea3226db01ede0d0f37b96
7	000
8	000

Successfully run. Total query runtime: 4 secs 478 msec. 1000000 rows affected.

4. SELECT num FROM test WHERE num > 33

	num integer
1	34
2	34
3	34
4	34
5	34
6	34
7	34
8	34

Successfully run. Total query runtime: 371 msec. 670626 rows affected.

5. SELECT num, count(num) FROM test WHERE num > 70 GROUP BY num

	num integer	count bigint
1	71	10145
2	72	9946
3	73	9914
4	74	10034
5	75	9997
6	76	9813
7	77	989
8	78	1003

Successfully run. Total query runtime: 361 msec. 30 rows affected.

Як бачимо індекс GIN проявив себе досить добре при пошуку рядку символів у тексті. Але на GIN можливо розраховувати лише при повнотекстовому пошуку, або при використанні операцій LIKE, ILIKE. Та пошуку змісту полів типу JSON. В інших випадках його не слід застосовувати, а також він не може бути створений для чисельних типів стовпців. Також недоліком GIN досить довге створення індексу.

Індекс BRIN показав не досить значне підвищення швидкодії, але його було розраховано на застосування до дуже великих таблиць і 100000 записів не так вже і багато. Також загальна ідея BRIN це не збільшення швидкодії, а уникнення непотрібних рядків.

BRIN-індекс має сенс застосовувати для таблиць, в яких частина даних вже за своєю природою якось відсортована. Наприклад, це характерно для логів або для історії замовлень.

BRIN прискорює оператори порівняння, але не впливає на like-запити. BRIN - не є унікальним індексом, тому не може використовуватися в якості індексу первинного ключа.

Завдання 3

Створення тригерної функції.

```
CREATE FUNCTION check_errors() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
cur_films cursor for select * from "Films";
is_found boolean;
sumstr varchar;
BEGIN
      sumstr = '';
      is_found = false;
      IF (TG_OP = 'UPDATE') THEN
            IF NEW.start_date < CURRENT_DATE OR NEW.start_date is NULL THEN
                  sumstr := sumstr || ' start_date';
                  RAISE NOTICE 'Start date % is not correct!', NEW.start_date;
            END IF;
            IF NEW.hall_name = '' OR NEW.hall_name is NULL THEN
                  sumstr := sumstr || ' hall_name';
                  RAISE NOTICE 'Hall name must be not empty!';
```

```
            END IF;
            FOR record IN cur_films LOOP
                  IF NEW.film_id = record.id  THEN
                        is_found = true;
                        EXIT;
                  END IF;
            END LOOP;
            IF is_found = false THEN
                  sumstr := sumstr || ' film_id';
                  RAISE NOTICE 'Film must be present in database!';
            END IF;
            IF sumstr <> '' THEN
                  sumstr :=  TG_OP||' Sessions'||sumstr;
                  INSERT INTO logs(text,date_added) values (sumstr,NOW());
                  --RAISE EXCEPTION 'Row % cannot updated!', NEW;
            END IF;
            RETURN NEW;
      ELSIF TG_OP = 'DELETE' THEN
            sumstr :=  TG_OP||' Sessions '|| OLD;
            INSERT INTO logs(text,date_added) values (sumstr,NOW());
            RETURN OLD;
      END IF;
      RETURN NEW;
END;
$$;


CREATE TRIGGER check_errors BEFORE DELETE OR UPDATE ON "Sessions"
FOR EACH ROW EXECUTE FUNCTION check_errors();
```

Даний тригер записує до таблиці logs всі видалення з таблиці «Sessions», спроби оновлення цієї таблиці з помилковими даними та сповіщає про помилки введення даних.

Перший приклад:

```
1  UPDATE "Sessions" SET
2  hall_name = 'Emerald',
3  film_id = 11
4  WHERE id = 10;
5
```

Data Output | Explain | Messages | Notifications

```
ЗАМЕЧАНИЕ:  Film must be present in database!

ERROR: ОШИБКА:  INSERT или UPDATE в таблице "Sessions" нарушает ограничение внешнего ключа "Sessions_film_id_fkey"
DETAIL:  Ключ (film_id)=(11) отсутствует в таблице "Films".


SQL state: 23503
```

Другий приклад:

```
5  UPDATE "Sessions" SET
6  hall_name = 'Emerald',
7  start_date = '2020-11-25'
8  WHERE id = 10;
9
```

Data Output | Explain | Messages | Notifications

```
ЗАМЕЧАНИЕ:  Start date 2020-11-25 is not correct!
UPDATE 1

Query returned successfully in 52 msec.
```

	id [PK] integer	start_date date	hall_name character varying (20)	film_id integer
1	1	2020-09-17	Almandine	1
2	2	2020-09-17	Ultramarine	2
3	3	2020-09-18	Terracotta	3
4	10	2020-11-25	Emerald	8

Третій приклад:

```
 9  DELETE FROM "Sessions" WHERE id = 100;
10
```

Data Output | Explain | Messages | Notifications

```
DELETE 1

Query returned successfully in 140 msec.
```

Зміст таблиці «logs» :

```
SELECT * FROM public.logs
```

ta Output | Explain | Messages | Notifications

text text	date_added timestamp without time zone
UPDATE Sessions start_date	2020-11-26 03:27:29.555228
DELETE Sessions (100,2020-12-12,Test,8)	2020-11-26 03:27:45.342952

Код програми знаходиться у репозиторію github за посиланням:
https://github.com/h0tw4t3r/uni_db_lab.git