

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського”
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих
комп’ютерних систем**

Лабораторна робота №3

З дисципліни

“Бази даних і засоби управління”

На тему: *“Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав:
студент групи КВ-93
Вітковський В.Б.
Перевірив:
Павловський В.І.

Київ 2021

Постановка задачі

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 вигляді об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

№ варіанта	Види індексів	Умови для тригера
4	<i>GIN, BRIN</i>	<i>before update, insert</i>

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом:

https://github.com/Vvold/DataBase_lab

Використана мова програмування: Python 3.10

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), sqlalchemy (для реалізації ORM), та інші.

Завдання №1

Обрана предметна галузь передбачає можливість користувачеві вибрати зі списку пісні, та додавати їх у власну бібліотеку.

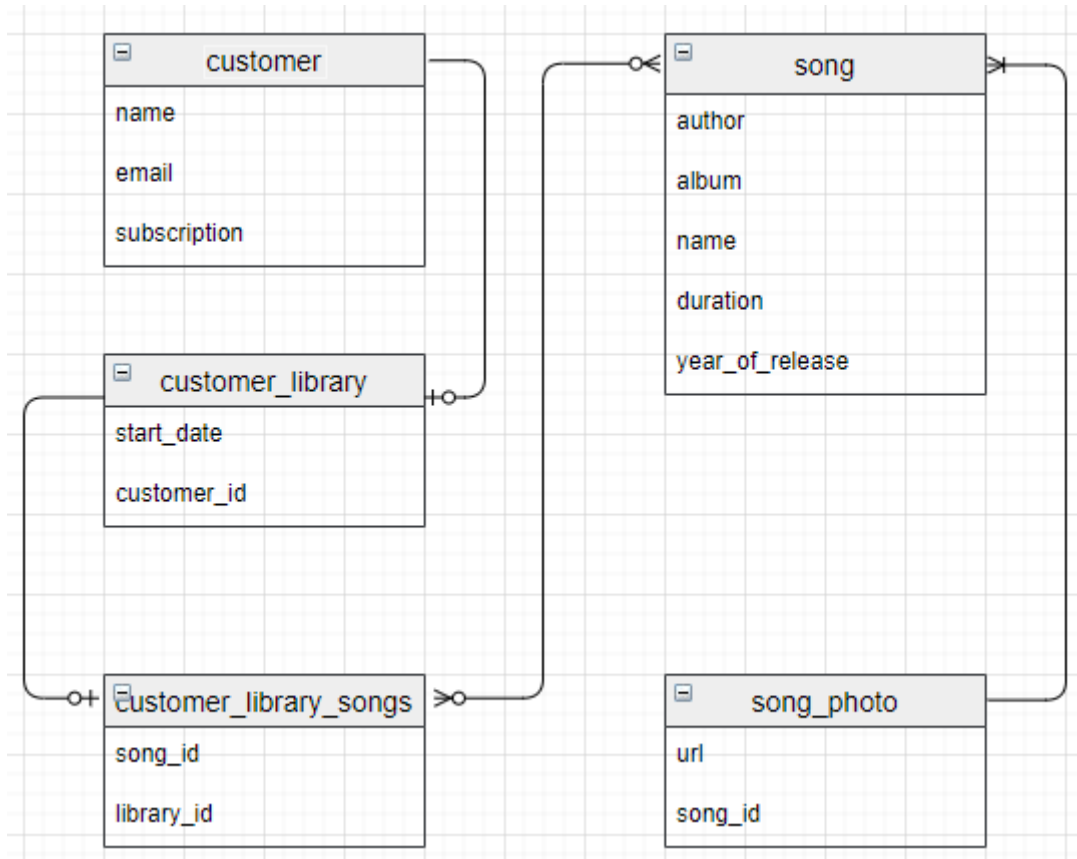


Рисунок 1. Схема бази даних

Програмний модуль “model.py”, з реалізованими класами ORM:

```
from sqlalchemy import Column, Integer, String, Boolean, ForeignKey, select,
and_
from sqlalchemy.orm import relationship
from db import Library, Session, engine
```

```
def recreate_database():
    Library.metadata.drop_all(engine)
    Library.metadata.create_all(engine)
```

```
class Customer(Library):
    __tablename__ = 'customer'
```

```

id = Column(Integer, primary_key=True)
name = Column(String)
email = Column(String)
subscription = Column(Boolean)
libraries = relationship("Customer_library")

def __init__(self, id, name, email, subscription):
    self.id = id
    self.name = name
    self.email = email
    self.subscription = subscription

def __repr__(self):
    return "{:>10}{:>35}{:>15}{:>10}" \
        .format(self.id, self.name, self.email, self.subscription)

class Customer_library(Library):
    __tablename__ = 'customer_library'
    id = Column(Integer, primary_key=True)
    start_date = Column(Integer)
    customer_id = Column(Integer, ForeignKey('customer.id'))
    customers = relationship("Customer_library_songs")

    def __init__(self, id, start_date, customer_id):
        self.id = id
        self.start_date = start_date
        self.customer_id = customer_id

    def __repr__(self):
        return "{:>10}{:>10}{:>10}" \
            .format(self.id, self.start_date, self.customer_id)

class Customer_library_songs(Library):
    __tablename__ = 'customer_library_songs'
    id = Column(Integer, primary_key=True)
    library_id = Column(Integer, ForeignKey('customer_library.id'))
    song_id = Column(Integer, ForeignKey('song.id'))

    def __init__(self, id, library_id, song_id):
        self.id = id
        self.library_id = library_id
        self.song_id = song_id

    def __repr__(self):
        return "{:>10}{:>15}{:>10}" \
            .format(self.id, self.library_id, self.song_id)

class Song(Library):
    __tablename__ = 'song'
    id = Column(Integer, primary_key=True)
    author = Column(String)
    album = Column(String)
    name = Column(String)
    duration = Column(Integer)
    year_of_release = Column(Integer)
    song_photos = relationship("Song_photo")

```

```

customer_library_songs = relationship("Customer_library_songs")

def __init__(self, id, author, album, name, duration, year_of_release):
    self.id = id
    self.author = author
    self.album = album
    self.name = name
    self.duration = duration
    self.year_of_release = year_of_release

def __repr__(self):
    return "{:>10}{:>25}{:>20}{:>20}{:>20}{:>20}" \
        .format(self.id, self.author, self.album, self.name,
self.duration, self.year_of_release)

class Song_photo(Library):
    __tablename__ = 'song_photo'
    id = Column(Integer, primary_key=True)
    url = Column(String)
    song_id = Column(Integer, ForeignKey('song.id'))

    def __init__(self, id, url, song_id):
        self.id_product = id
        self.url = url
        self.song_id = song_id

    def __repr__(self):
        return "{:>10}{:>30}{:>10}" \
            .format(self.id, self.url, self.song_id)

class Model:
    def __init__(self):
        self.session = Session()
        self.connection = engine.connect()

    def find_pk_customer(self, key_value: int):
        return self.session.query(Customer).filter_by(id=key_value).first()

    def find_pk_customer_library(self, key_value: int):
        return
self.session.query(Customer_library).filter_by(id=key_value).first()

    def find_fk_customer_library(self, key_value: int):
        return
self.session.query(Customer_library).filter_by(customer_id=key_value).first()

    def find_pk_customer_library_songs(self, key_value: int):
        return
self.session.query(Customer_library_songs).filter_by(id=key_value).first()

    def find_fk_customer_library_songs(self, key_value: int, table_name:
str):
        if table_name == "customer_library":
            return
self.session.query(Customer_library_songs).filter_by(library_id=key_value).
first()

```

```

        elif table_name == "song":
            return
self.session.query(Customer_library_songs).filter_by(song_id=key_value).first()

    def find_pk_song(self, key_value: int):
        return self.session.query(Song).filter_by(id=key_value).first()

    def find_pk_song_photo(self, key_value: int):
        return
self.session.query(Song_photo).filter_by(id=key_value).first()

    def find_fk_song_photo(self, key_value: int):
        return
self.session.query(Song_photo).filter_by(song_id=key_value).first()

    def print_customer(self):
        return
self.session.query(Customer).order_by(Customer.id.asc()).all()

    def print_customer_library(self):
        return
self.session.query(Customer_library).order_by(Customer_library.id.asc()).all()

    def print_customer_library_songs(self):
        return
self.session.query(Customer_library_songs).order_by(Customer_library_songs.id.asc()).all()

    def print_song(self):
        return self.session.query(Song).order_by(Song.id.asc()).all()

    def print_song_photo(self):
        return
self.session.query(Song_photo).order_by(Song_photo.id.asc()).all()

    def delete_data_customer(self, id) -> None:
        self.session.query(Customer).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_customer_library(self, id) -> None:
        self.session.query(Customer_library).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_customer_library_songs(self, id) -> None:
        self.session.query(Customer_library_songs).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_song(self, id) -> None:
        self.session.query(Song).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_song_photo(self, id) -> None:
        self.session.query(Song_photo).filter_by(id=id).delete()
        self.session.commit()

    def update_data_customer(self, id: int, name: str, email: str,
subscription: bool) -> None:

```

```

        self.session.query(Customer).filter_by(id=id) \
            .update({Customer.name: name, Customer.email: email,
Customer.subscription: subscription})
        self.session.commit()

    def update_data_customer_library(self, id: int, start_date: int,
customer_id: int) -> None:
        self.session.query(Customer_library).filter_by(id=id) \
            .update({Customer_library.start_date: start_date,
Customer_library.customer_id: customer_id})
        self.session.commit()

    def update_data_customer_library_songs(self, id: int, library_id: int,
song_id: int) -> None:
        self.session.query(Customer_library_songs).filter_by(id=id) \
            .update({Customer_library_songs.library_id: library_id,
Customer_library_songs.song_id: song_id})
        self.session.commit()

    def update_data_song(self, id: int, author: str, album: str, name: str,
duration: int,
                        year_of_release: str) -> None:
        self.session.query(Song).filter_by(id=id) \
            .update({Song.author: author, Song.album: album, Song.name:
name, Song.duration: duration,
                    Song.year_of_release: year_of_release})
        self.session.commit()

    def update_data_song_photo(self, id: int, url: str, song_id: int) ->
None:
        self.session.query(Song_photo).filter_by(id=id) \
            .update({Song_photo.url: url, Song.song_id: song_id})
        self.session.commit()

    def insert_data_customer(self, id: int, name: str, email: str,
subscription: str) -> None:
        customer = Customer(id=id, name=name, email=email,
subscription=subscription)
        self.session.add(customer)
        self.session.commit()

    def insert_data_customer_library(self, id: int, start_date: int,
customer_id: int) -> None:
        customer_library = Customer_library(id=id, start_date=start_date,
customer_id=customer_id)
        self.session.add(customer_library)
        self.session.commit()

    def insert_data_customer_library_songs(self, id: int, library_id: int,
song_id: int) -> None:
        customer_library_songs = Customer_library_songs(id=id,
library_id=library_id, song_id=song_id)
        self.session.add(customer_library_songs)
        self.session.commit()

    def insert_data_song(self, id: int, author: str, album: str, name: str,
duration: int,
                        year_of_release: int) -> None:
        song = Song(id=id, author=author, album=album, name=name,

```

```

duration=duration, year_of_release=year_of_release)
    self.session.add(song)
    self.session.commit()

    def insert_data_song_photo(self, id: int, url: str, song_id: str) ->
None:
        song_photo = Song_photo(id=id, url=url, song_id=song_id)
        self.session.add(song_photo)
        self.session.commit()

    def customer_data_generator(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.customer select (SELECT
(MAX(id)+1) FROM public.customer), "
                        "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) "
                        "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''),",
                        "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) "
                        "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''),",
                        "(SELECT TRUE or FALSE );")

    def customer_library_data_generator(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.customer_library select
(SELECT MAX(id)+1 FROM public.customer_library), "
                        "FLOOR(RANDOM()*(200000-1)+1),",
                        "(SELECT id FROM public.customer LIMIT 1 OFFSET "
                        "(round(random() *((SELECT COUNT(id) FROM
public.customer)-1)))));")

    def customer_library_songs_data_generator(self, times: int) -> None:
        for i in range(times):
            self.request(
                "insert into public.customer_library_songs select (SELECT
MAX(id)+1 FROM public.customer_library_songs), "
                "(SELECT id FROM public.customer_library LIMIT 1 OFFSET "
                "(round(random() *((SELECT COUNT(id) FROM
public.customer_library)-1))))",
                "(SELECT id FROM public.song LIMIT 1 OFFSET "
                "(round(random() *((SELECT COUNT(id) FROM public.song)-
1)))));")

    def song_data_generator(self, times: int) -> None:
        for i in range(times):
            self.request("insert into public.song select (SELECT (MAX(id)+1)
FROM public.song), "
                        "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) "
                        "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''),",
                        "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) "
                        "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer)), ''),",
                        "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) ")

```



```

integer)), ''), "
                                "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
                                "FLOOR(RANDOM()*(200000-1)+1),"
                                "FLOOR(RANDOM()*(2021-1)+1); ")

def song_photo_data_generator(self, times: int) -> None:
    for i in range(times):
        self.request("insert into public.song_photo select (SELECT
MAX(id)+1 FROM public.song_photo), "
                    "array_to_string(ARRAY(SELECT chr((97 +
round(random() * 25)) :: integer) "
                    "FROM generate_series(1, FLOOR(RANDOM()*(10-3)+3)::
integer))), '');"

                                "(SELECT id FROM public.song LIMIT 1 OFFSET "
                                "(round(random() *((SELECT COUNT(id) FROM
public.song)-1)))));")

def search_data_two_tables(self):
    return self.session.query(Customer) \
        .join(Customer_library) \
        .filter(and_(
            Customer.id.between(0, 5),
            Customer_library.id.between(0, 5)
        )) \
        .all()

def search_data_three_tables(self):
    return self.session.query(Customer) \
        .join(Customer_library).join(Customer_library_songs) \
        .filter(and_(
            Customer.id.between(0, 5),
            Customer_library.id.between(0, 5),
            Customer_library_songs.id.between(0, 5)
        )) \
        .all()

def search_data_four_tables(self):
    return self.session.query(Customer) \
        .join(Customer_library).join(Customer_library_songs).join(Song)
\
        .filter(and_(
            Customer.id.between(0, 5),
            Customer_library.id.between(0, 5),
            Customer_library_songs.id.between(0, 5),
            Song.duration.between(0, 260)
        )) \
        .all()

```

Запити у вигляді ORM

Продемонструємо вставку, вилучення, редагування даних на прикладі таблиці song:

Початковий стан:

```
PS C:\my\Python\pythonProject19> python main.py print_table song
song table:
  id          author      album      name      duration  year_of_release
  1          Author_1     Album_1     song_1      240        1999
  2          Author_2     Album_2     song_2      250        2000
  3          Author_3     Album_3     song_3      260        2001
  4          Author_4     Album_4     song_4      270        2002
  5          ujrirt       hfwwqb      firrodg     144626     594
  6          ueaskcyh      fxhhszx     cusat       54720      312
  7          ckivkijik     cdbrdcp     huxsjcr     175775     1057
  8          rgvshsifk     wtqvwftd     vnnkfpe     151762     1026
  9          gnjyxpbon      nkmelgss     cuuvhvxxq  130528     507
PS C:\my\Python\pythonProject19>
```

Видалення запису:

```
PS C:\my\Python\pythonProject19> python main.py delete_record song 6
PS C:\my\Python\pythonProject19> python main.py print_table song
song table:
  id          author      album      name      duration  year_of_release
  1          Author_1     Album_1     song_1      240        1999
  2          Author_2     Album_2     song_2      250        2000
  3          Author_3     Album_3     song_3      260        2001
  4          Author_4     Album_4     song_4      270        2002
  5          ujrirt       hfwwqb      firrodg     144626     594
  7          ckivkijik     cdbrdcp     huxsjcr     175775     1057
  8          rgvshsifk     wtqvwftd     vnnkfpe     151762     1026
  9          gnjyxpbon      nkmelgss     cuuvhvxxq  130528     507
PS C:\my\Python\pythonProject19>
```

Вставка запису:

```
PS C:\my\Python\pythonProject19> python main.py insert_record song 6 Author_6 Album_6 song_6 290 2006
PS C:\my\Python\pythonProject19> python main.py print_table song
song table:
  id          author      album      name      duration  year_of_release
  1          Author_1     Album_1     song_1      240        1999
  2          Author_2     Album_2     song_2      250        2000
  3          Author_3     Album_3     song_3      260        2001
  4          Author_4     Album_4     song_4      270        2002
  5          ujrirt       hfwwqb      firrodg     144626     594
  6          Author_6     Album_6     song_6      290        2006
  7          ckivkijik     cdbrdcp     huxsjcr     175775     1057
  8          rgvshsifk     wtqvwftd     vnnkfpe     151762     1026
  9          gnjyxpbon      nkmelgss     cuuvhvxxq  130528     507
PS C:\my\Python\pythonProject19>
```

Редагування запису:

```
PS C:\my\Python\pythonProject19> python main.py update_record song 8 Author_8 Album_8 song_8 250 2000
PS C:\my\Python\pythonProject19> python main.py print_table song
song table:
  id          author      album      name      duration  year_of_release
  1          Author_1     Album_1     song_1      240        1999
  2          Author_2     Album_2     song_2      250        2000
  3          Author_3     Album_3     song_3      260        2001
  4          Author_4     Album_4     song_4      270        2002
  5          ujrirt       hfwwqb      firrodg     144626     594
  7          ckivkijik     cdbrdcp     huxsjcr     175775     1057
  8          Author_8     Album_8     song_8      250        2000
  9          gnjyxpbon      nkmelgss     cuuvhvxxq  130528     507
PS C:\my\Python\pythonProject19>
```

Запити пошуку та генерації рандомізованих даних також було реалізовано, логіку пошуку було змінено у порівнянні з лабораторною роботою №2 (усі дані для пошуку передвизначено,

тепер вони не вводяться з клавіатури). Запити на пошук ті самі, що і л.р. №2.

Запит на генерацію даних продемонструємо на прикладі таблиці customer_library.

Початковий стан:

```
PS C:\my\Python\pythonProject19> python main.py print_table customer_library
customer_library table:
      id      start_date      customer_id
      1         191832           1
      2         191900           3
      3         193333           2
      4         124468           2
      5         100263           2
      6          78273           3
      7         159248           2
      8          55696           3
PS C:\my\Python\pythonProject19>
```

Вставка 3-х випадково згенерованих записів:

```
PS C:\my\Python\pythonProject19> python main.py generate_randomly customer_library 3
PS C:\my\Python\pythonProject19> python main.py print_table customer_library
customer_library table:
      id      start_date      customer_id
      1         191832           1
      2         191900           3
      3         193333           2
      4         124468           2
      5         100263           2
      6          78273           3
      7         159248           2
      8          55696           3
      9         101590          12
     10          31660           3
     11          86579          13
PS C:\my\Python\pythonProject19>
```

Пошук за двома атрибутами у двох таблицях, за трьома атрибутами у трьох таблицях, за чотирма атрибутами у чотирьох таблицях (виводяться відповідні записи з таблиці Product):

```

PS C:\my\Python\pythonProject19> python main.py search_records
specify the number of tables you'd like to search in: 2
search result:
1          Andriy          andriy@gmail.com          1
2          Vitaliy         vitaliy@gmail.com         0
3          Bogdan          bogdan@gmail.com         1
PS C:\my\Python\pythonProject19> python main.py search_records
specify the number of tables you'd like to search in: 3
search result:
1          Andriy          andriy@gmail.com          1
3          Bogdan          bogdan@gmail.com         1
PS C:\my\Python\pythonProject19> python main.py search_records
specify the number of tables you'd like to search in: 4
search result:
1          Andriy          andriy@gmail.com          1
3          Bogdan          bogdan@gmail.com         1
PS C:\my\Python\pythonProject19>

```

Завдання №2

Для тестування індексів було створено окремі таблиці у базі даних з 1000000 записів.

GIN

GIN призначений для обробки випадків, коли елементи, що підлягають індексації, є складеними значеннями (наприклад - реченнями), а запити, які обробляються індексом, мають шукати значення елементів, які з'являються в складених елементах (повторювані частини слів або речень). Індекс GIN зберігає набір пар (ключ, список появи ключа), де список появи — це набір ідентифікаторів рядків, у яких міститься ключ. Один і той самий ідентифікатор рядка може знаходитись у кількох списках, оскільки елемент може містити більше одного ключа. Кожне значення ключа зберігається лише один раз, тому індекс GIN дуже швидкий для випадків, коли один і той же ключ з'являється багато разів.

Стверення таблиці БД:

```

DROP TABLE IF EXISTS "gin_test";
CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" text,
"gin_vector"

```

```
tsvector);
INSERT INTO "gin_test"("string") SELECT substr(characters, (random() *
length(characters) + 1)::integer, 10) FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as
symbols(characters), generate_series(1, 1000000) as q;
UPDATE "gin_test" set "gin_vector" = to_tsvector("string");
```

Запити для тестування:

```
SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@
to_tsquery('QWERTYUIOP')) OR
("gin_vector" @@ to_tsquery('bnm'));
SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@
to_tsquery('bnm'))
GROUP BY "id" % 2;
```

Створення індексу:

```
DROP INDEX IF EXISTS "gin_index";
CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
```

Запити без індексування:

```
Секундомер включен.
postgres=# DROP INDEX IF EXISTS "gin_index";
ПОВІДОМЛЕННЯ:  індекс "gin_index" не існує, пропускається
DROP INDEX
Время: 1,634 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
 count
-----
 500000
(1 строка)

Время: 203,518 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
 count
-----
 19142
(1 строка)

Время: 474,229 мс
postgres=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
 sum
-----
23943769938
(1 строка)

Время: 1188,034 мс (00:01,188)
postgres=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id" % 2;
 min | max
-----+-----
 100 | 999994
  45 | 999937
(2 строки)

Время: 1120,586 мс (00:01,121)
```

Запити з індексуванням:

```

postgres=# CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
CREATE INDEX
Время: 355,983 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 строка)

Время: 156,321 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
count
-----
19142
(1 строка)

Время: 25,425 мс
postgres=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
sum
-----
23943769938
(1 строка)

Время: 243,217 мс
postgres=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id" % 2;
min | max
-----+-----
45  | 999937
100 | 999994
(2 строки)

Время: 13,533 мс

```

BRIN

BRIN – це Block Range Index, головна концепція якого не знаходження необхідного значення, а уникнення перегляду свідомо непотрібних.

Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

Стверення таблиці БД:

```

DROP TABLE IF EXISTS "brin_test";
CREATE TABLE "brin_test"("id" bigserial PRIMARY KEY, "time" timestamp);
INSERT INTO "brin_test"("time") SELECT (timestamp '2021-01-01' + random() *
(timestamp
'2020-01-01' - timestamp '2022-01-01')) FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as
symbols(characters), generate_series(1, 1000000) as q;

```

Запити для тестування:

```

SELECT COUNT(*) FROM "brin_test" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "brin_test" WHERE "time" >= '20191001';
SELECT AVG("id") FROM "brin_test" WHERE "time" >= '20191001' AND "time" <=
'20211207';

```

```
SELECT SUM("id"), MAX("id") FROM "brin_test" WHERE "time" >= '20200505' AND  
"time" <=  
'20210505' GROUP BY "id" % 2;
```

Створення індексу:

```
DROP TABLE IF EXISTS "brin_test";  
CREATE INDEX "time_brin_index" ON "brin_test" USING brin("id");
```

Запити без індексування:

```
Секундомер включён.  
postgres=# DROP INDEX IF EXISTS "time_brin_index";  
ПОВІДОМЛЕННЯ:  індекс "time_brin_index" не існує, пропускається  
DROP INDEX  
Время: 2,492 мс  
postgres=# SELECT COUNT(*) FROM "brin_test" WHERE "id" % 2 = 0;  
count  
-----  
500000  
(1 строка)  
  
Время: 111,332 мс  
postgres=# SELECT COUNT(*) FROM "brin_test" WHERE "time" >= '20191001';  
count  
-----  
626919  
(1 строка)  
  
Время: 163,147 мс  
postgres=# SELECT AVG("id") FROM "brin_test" WHERE "time" >= '20191001' AND "time" <= '20211207';  
avg  
-----  
500254.786621557171  
(1 строка)  
  
Время: 176,904 мс  
postgres=# SELECT SUM("id"), MAX("id") FROM "brin_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id" % 2;  
sum | max  
-----+-----  
82418773060 | 999999  
82457311990 | 999994  
(2 строки)  
  
Время: 120,725 мс
```

Запити з індексуванням:

```
postgres=# CREATE INDEX "time_brin_index" ON "brin_test" USING brin("id");  
CREATE INDEX  
Время: 3745,561 мс (00:03,746)  
postgres=# SELECT COUNT(*) FROM "brin_test" WHERE "id" % 2 = 0;  
count  
-----  
500000  
(1 строка)  
  
Время: 95,619 мс  
postgres=# SELECT COUNT(*) FROM "brin_test" WHERE "time" >= '20191001';  
count  
-----  
626336  
(1 строка)  
  
Время: 100,444 мс  
postgres=# SELECT AVG("id") FROM "brin_test" WHERE "time" >= '20191001' AND "time" <= '20211207';  
avg  
-----  
500044.170913056251  
(1 строка)  
  
Время: 155,144 мс  
postgres=# SELECT SUM("id"), MAX("id") FROM "brin_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id" % 2;  
sum | max  
-----+-----  
82587011394 | 999994  
82348391481 | 999989  
(2 строки)  
  
Время: 155,314 мс
```

Завдання №3

Для тестування тригера було створено таблицю:

```
DROP TABLE IF EXISTS "reader";
CREATE TABLE "reader"(
    "readerID" bigserial PRIMARY KEY,
    "readerName" varchar(255)
);
```

Початкові дані у таблицях:

```
INSERT INTO "reader"("readerName")
VALUES ('reader1'), ('reader2'), ('reader3'), ('reader4'), ('reader5');
```

Тригер:

```
CREATE OR REPLACE FUNCTION update_insert_func() RETURNS TRIGGER as $$
DECLARE
    curs CURSOR FOR SELECT * FROM "reader";
    m_row "reader"%ROWTYPE;
begin
    IF TG_OP = 'INSERT' then
        for m_row in curs loop
            UPDATE "reader" SET "readerName"=m_row."readerName" || 'a'
WHERE current of curs;
        END LOOP;
        RAISE NOTICE 'Triggered on inserting!';
        return m_row;
    else
        RAISE NOTICE 'Triggered on updating!';
        return NULL;
    END IF;
END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER "test_trigger"
AFTER UPDATE OR INSERT ON "reader"
FOR EACH ROW
EXECUTE procedure update_insert_func();
```

Скріншоти зі змінами у таблицях бази даних

Початковий стан:

Результат		
	readerID [PK] bigint	readerName character varying (255)
1	1	reader1
2	2	reader2
3	3	reader3
4	4	reader4
5	5	reader5

Після виконання запиту вставки:

```
SELECT * FROM "reader";  
INSERT INTO "reader"("readerName") VALUES ('reader6');
```

Результат		
	readerID [PK] bigint	readerName character varying (255)
1	1	reader1
2	2	reader2
3	3	reader3
4	4	reader4
5	5	reader5
6	6	reader6

Після виконання запиту на оновлення:

```
SELECT * FROM "reader";  
--INSERT INTO "reader"("readerName") VALUES ('reader6');  
UPDATE "reader" SET "readerName" = 'READER';
```



Результат		
	readerID [PK] bigint	readerName character varying (255)
1	1	READER
2	2	READER
3	3	READER
4	4	READER
5	5	READER
6	6	READER

Перевірка роботи тригера на дію відмінни від оновлення та вставки (в даному випадку візьмемо видалення):

Запит:

```
SELECT * FROM "reader";  
DELETE FROM "reader" WHERE "readerID" = 6;  
--INSERT INTO "reader"("readerName") VALUES ('reader6');
```

Результат

	readerID [PK] bigint 	readerName character varying (255) 
1	1	READER
2	2	READER
3	3	READER
4	4	READER
5	5	READER