

**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 Та Засоби Управління”

ТЕМА: “Засоби оптимізації роботи СУБД PostgreSQL”

Група: KB-03

Виконав: Донченко І. Ю.

Оцінка:

Київ – 2022

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

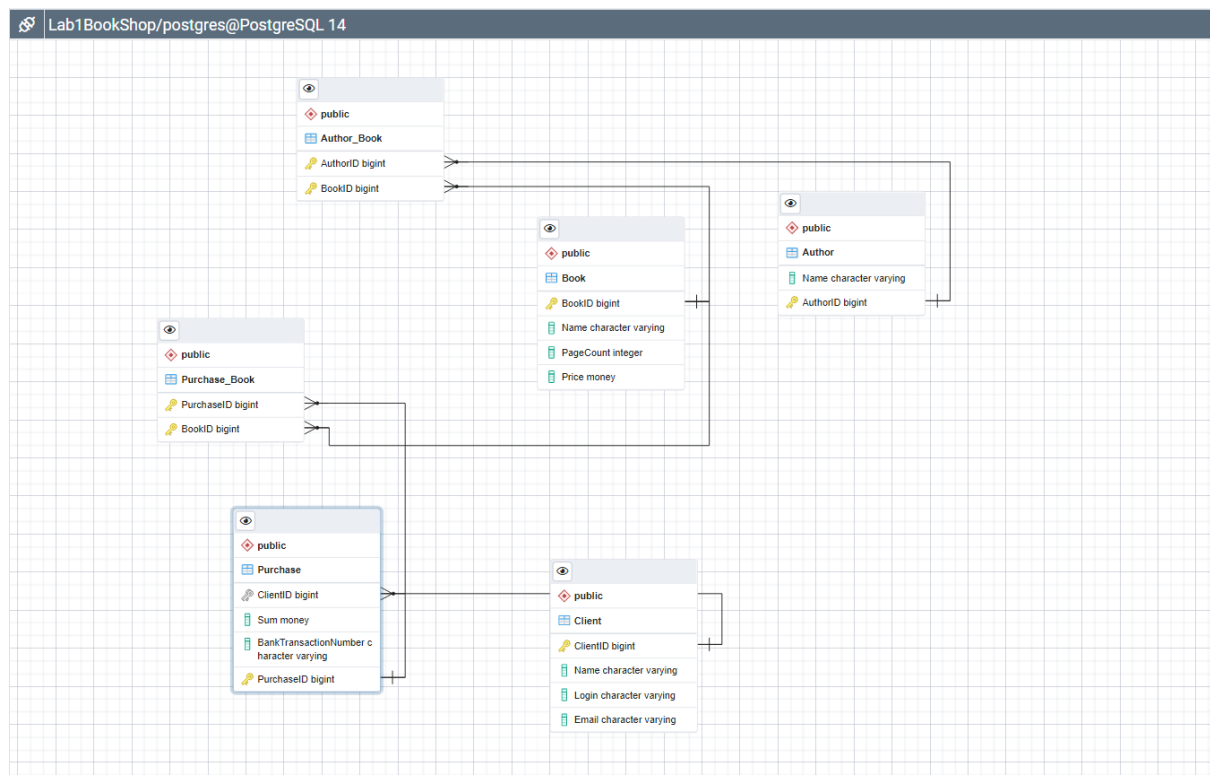
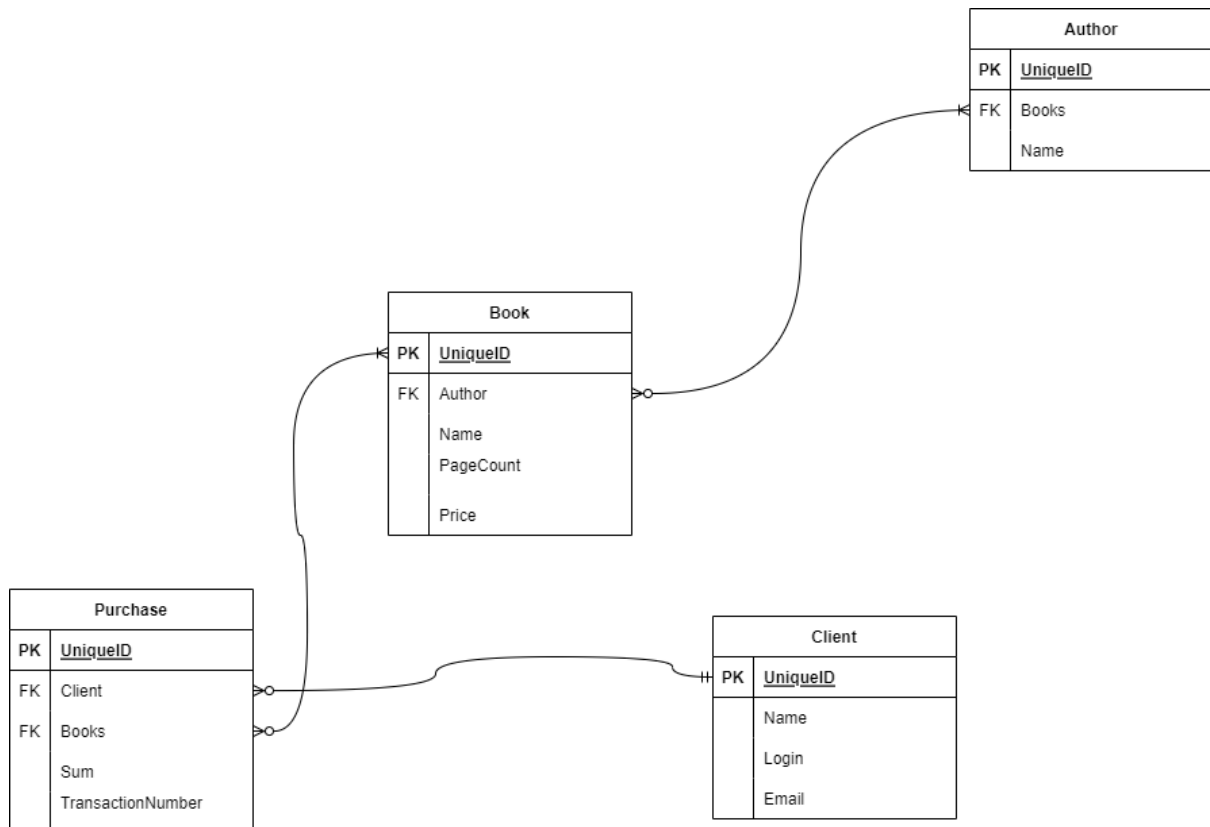
1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Звіт лабораторної роботи має містити:

- a) титульний аркуш затвердженого зразка;
- b) завдання на лабораторну роботу з обов’язковим наведенням варіанту студента;

6	<i>BTree, BRIN</i>	<i>after update, insert</i>
---	--------------------	-----------------------------

- c) копії екрану (скріншоти), що **підтверджують вимоги 1-4 завдання**, а також:
 - для завдання №1: схему бази даних у вигляді таблиць і зв’язків між ними, а також класи ORM і зв’язки між ними, що відповідають таблицям бази даних. Навести приклади запитів у вигляді ORM.



```

from sqlalchemy import create_engine, Column, ForeignKey, Integer,
String, Table, and_

from sqlalchemy.orm import sessionmaker, relationship
  
```

```

from sqlalchemy.ext.declarative import declarative_base

import psycopg2

import Controller

engine =
create_engine('postgresql://postgres:qwel23@localhost:5432/Lab1BookShop', echo=False)

Session = sessionmaker(bind=engine)

session = Session()

Base = declarative_base()

book_purchase = Table('Purchase_Book', Base.metadata,
                       Column('PurchaseID', ForeignKey('Purchase.PurchaseID'),
primary_key=True),

                       Column('BookID', ForeignKey('Book.BookID'), primary_key=True),
)

book_author = Table('Author_Book', Base.metadata,
                    Column('AuthorID', ForeignKey('Author.AuthorID'),
primary_key=True),

                    Column('BookID', ForeignKey('Book.BookID'), primary_key=True)
)

class Author(Base):
    __tablename__ = 'Author'

    AuthorID = Column(Integer, primary_key=True)

    Name = Column(String)

```

```

        Books = relationship('Book', secondary=book_author,
back_populates='Authors')

    def __init__(self, author_id, name):

        self.AuthorID = author_id

        self.Name = name

class Book(Base):

    __tablename__ = 'Book'

    BookID = Column(Integer, primary_key=True)

    Name = Column(String)

    PageCount = Column(Integer)

    Price = Column(Integer)

    Purchases = relationship('Purchase', secondary=book_purchase,
back_populates='Books')

    Authors = relationship('Author', secondary=book_author,
back_populates='Books')

    def __init__(self, book_id, name, page_count, price):

        self.BookID = book_id

        self.Name = name

        self.PageCount = page_count

        self.Price = price

class Client(Base):

    __tablename__ = 'Client'

    ClientID = Column(Integer, primary_key=True)

    Name = Column(String)

    Login = Column(String)

    Email = Column(String)

```

```

Purchases = relationship('Purchase', backref='Clients')

def __init__(self, client_id, name, login, email):

    self.ClientID = client_id

    self.Name = name

    self.Login = login

    self.Email = email


class Purchase(Base):

    __tablename__ = 'Purchase'

    PurchaseID = Column(Integer, primary_key=True)

    Sum = Column(Integer)

    ClientID = Column(Integer, ForeignKey('Client.ClientID'))

    BankTransactionNumber = Column(String)

    Books = relationship('Book', secondary=book_purchase,
back_populates='Purchases')

    def __init__(self, purchase_id, client_id, sum,
transaction_number):

        self.PurchaseID = purchase_id

        self.ClientID = client_id

        self.Sum = sum

        self.BankTransactionNumber = transaction_number


def connect():

    Base.metadata.create_all(engine)


def add_author(author_id, author_name):

    lib = Author(author_id, author_name)

    session.add(lib)

```

```
session.commit()

def get_author(author_id):

    lib = session.query(Author).filter(Author.AuthorID ==
int(author_id)).first()

    r = [lib.AuthorID, lib.Name]

    return r

def update_author(author_id, author_name):

    lib = session.query(Author).filter(Author.AuthorID ==
int(author_id)).first()

    lib.Name = author_name

    session.commit()

def delete_author(author_id):

    lib = session.query(Author).filter(Author.AuthorID ==
int(author_id)).first()

    session.delete(lib)

    session.commit()

def add_book(book_id, book_name, page_count, book_price):

    lib = Book(book_id, book_name, page_count, book_price)

    session.add(lib)

    session.commit()

def get_book(book_id):
```

```

        lib = session.query(Book).filter(Book.BookID ==
int(book_id)).first()

    r = [lib.BookID, lib.Name, lib.PageCount, lib.Price]

    return r

def update_book(book_id, book_name, page_count, book_price):

    lib = session.query(Book).filter(Book.BookID ==
int(book_id)).first()

    lib.Name = book_name

    lib.PageCount = page_count

    lib.Price = book_price

    session.commit()

def delete_book(book_id):

    lib = session.query(Book).filter(Book.BookID ==
int(book_id)).first()

    session.delete(lib)

    session.commit()

def add_author_book_pair(author_id, book_id):

    lib = session.query(Author).filter(Author.AuthorID ==
int(author_id)).first()

    lib.Books.append(session.query(Book).filter(Book.BookID ==
int(book_id)).first())

    session.add(lib)

    session.commit()

def get_authors_by_book_id(book_id):

```



```

lib = session.query(Author).join(book_author).join(Book).filter(Book.BookID == book_id).all()

r = []

for i in lib:

    r.append([i.AuthorID, i.Name])

return r

def delete_author_book_pairs_by_book_id(book_id):

    lib = session.query(Author).join(book_author).join(Book).filter(Book.BookID == book_id).all()

    for i in lib:

        session.delete(i)

    session.commit()

def add_purchase_book_pair(purchase_id, book_id):

    lib = session.query(Purchase).filter(Purchase.PurchaseID == int(purchase_id)).first()

    lib.Books.append(session.query(Book).filter(Book.BookID == int(book_id)).first())

    session.add(lib)

    session.commit()

def get_books_by_purchase_id(purchase_id):

    lib = session.query(Book).join(book_purchase).join(Purchase).filter(Purchase.PurchaseID == purchase_id).all()

    r = []

    for i in lib:

```

```

        r.append([i.BookID, i.Name, i.PageCount, i.Price])

    return r

def delete_purchase_book_pairs_by_purchase_id(purchase_id):

    lib = session.query(Book).join(book_purchase).join(Purchase).filter(Purchase.PurchaseID == purchase_id).all()

    for i in lib:

        session.delete(i)

    session.commit()

def add_purchase(purchase_id, client_id, sum, transaction_number):

    lib = Purchase(purchase_id, client_id, sum, transaction_number)

    session.add(lib)

    session.commit()

def get_purchase(purchase_id):

    lib = session.query(Purchase).filter(Purchase.PurchaseID == int(purchase_id)).first()

    r = [lib.PurchaseID, lib.Sum, lib.BankTransactionNumber, lib.ClientID]

    return r

def update_purchase(purchase_id, client_id, sum, transaction_number):

    lib = session.query(Purchase).filter(Purchase.PurchaseID == int(purchase_id)).first()

    lib.ClientID = client_id

    lib.Sum = sum

    lib.BankTransactionNumber = transaction_number

```

```
session.commit()

def delete_purchase(purchase_id):

    lib = session.query(Purchase).filter(Purchase.PurchaseID ==
int(purchase_id)).first()

    session.delete(lib)

    session.commit()

def add_client(client_id, name, login, email):

    lib = Client(client_id, name, login, email)

    session.add(lib)

    session.commit()

def get_client(client_id):

    lib = session.query(Client).filter(Client.ClientID ==
int(client_id)).first()

    r = [lib.ClientID, lib.Name, lib.Login, lib.Email]

    return r

def update_client(client_id, name, login, email):

    lib = session.query(Client).filter(Client.ClientID ==
int(client_id)).first()

    lib.Name = name

    lib.Login = login

    lib.Email = email

    session.commit()
```

```
def delete_client(client_id):
    lib = session.query(Client).filter(Client.ClientID ==
int(client_id)).first()

    session.delete(lib)

    session.commit()
```

- для завдання №2: команди створення індексів, тексти, результати і час виконання запитів SQL, пояснити чому індекси прискорюють (або не прискорюють) швидкість виконання запитів.

BTree

Запити для тестування:

-- BTree

DROP INDEX if exists or_index;

explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523';

explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';

explain analyze select * from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';

CREATE INDEX or_index ON "Purchase"("BankTransactionNumber");

explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523';

explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';

```
explain analyze select * from "Purchase" where "BankTransactionNumber" !=
'UA35253253523' and "BankTransactionNumber" != 'US352355' and
"BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" !=
'UA28533621';
```

```
SQL Shell (psql)
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=# -- BTree
Lab1BookShop=#
Lab1BookShop=# DROP INDEX if exists or_index;
NOTICE: index "or_index" does not exist, skipping
DROP INDEX
Lab1BookShop=#
Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523';
QUERY PLAN
-----
Seq Scan on "Purchase" (cost=0.00..846.31 rows=1 width=34) (actual time=0.031..12.457 rows=1 loops=1)
  Filter: (((("BankTransactionNumber")::text = 'UA35253253523')::text)
  Rows Removed by Filter: 40344
Planning Time: 0.371 ms
Execution Time: 12.479 ms
(5 rows)

Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
QUERY PLAN
-----
Result (cost=0.00..846.31 rows=1 width=34) (actual time=0.000..0.001 rows=0 loops=1)
  One-Time Filter: (false AND false)
  -> Seq Scan on "Purchase" (cost=0.00..846.31 rows=1 width=34) (never executed)
    Filter: (((("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.396 ms
Execution Time: 0.025 ms
(6 rows)

Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';
QUERY PLAN
-----
Seq Scan on "Purchase" (cost=0.00..1148.90 rows=40341 width=34) (actual time=0.016..14.706 rows=40341 loops=1)
  Filter: (((("BankTransactionNumber")::text <> 'UA35253253523')::text) AND (((("BankTransactionNumber")::text <> 'US352355')::text) AND (((("BankTransactionNumber")::text <> 'UA19658577')::text) AND (((("BankTransactionNumber")::text <> 'UA28533621')::text))
  Rows Removed by Filter: 4
Planning Time: 0.084 ms
Execution Time: 16.060 ms
(5 rows)

Lab1BookShop=#
Lab1BookShop=#
```

```
SQL Shell (psql)
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=#
Lab1BookShop=# CREATE INDEX or_index ON "Purchase"("BankTransactionNumber");
CREATE INDEX
Lab1BookShop=#
Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523';
QUERY PLAN
-----
Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=34) (actual time=0.053..0.054 rows=1 loops=1)
  Index Cond: (((("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.495 ms
Execution Time: 0.070 ms
(4 rows)

Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
QUERY PLAN
-----
Result (cost=0.29..8.31 rows=1 width=34) (actual time=0.000..0.001 rows=0 loops=1)
  One-Time Filter: (false AND false)
  -> Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=34) (never executed)
    Index Cond: (((("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.146 ms
Execution Time: 0.018 ms
(6 rows)

Lab1BookShop=# explain analyze select * from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';
QUERY PLAN
-----
Seq Scan on "Purchase" (cost=0.00..1148.90 rows=40341 width=34) (actual time=0.020..11.984 rows=40341 loops=1)
  Filter: (((("BankTransactionNumber")::text <> 'UA35253253523')::text) AND (((("BankTransactionNumber")::text <> 'US352355')::text) AND (((("BankTransactionNumber")::text <> 'UA19658577')::text) AND (((("BankTransactionNumber")::text <> 'UA28533621')::text))
  Rows Removed by Filter: 4
Planning Time: 0.125 ms
Execution Time: 13.169 ms
(5 rows)

Lab1BookShop=#
Lab1BookShop=#
```

Отже з отриманих результатів бачимо що, пошук з індексацією відбувається набагато швидше для роботи з невеликою кількість даних,

ніж без індексації. BTree має свою особливість, а саме, він показує свою ефективність тоді коли працює з невеликою кількістю даних. При сортуванні даних з індексацією бачимо, що вона відбулась дещо швидше ніж без індексації, це пов'язано також з нормальною кількістю даних для індексу BTree.

BRIN

Запити для тестування:

-- BRIN

DROP INDEX if exists or_brin_index;

```
explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' ;
```

```
explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523';
```

```
explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
```

```
explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';
```

```
CREATE INDEX or_brin_index on "Purchase" using brin ("BankTransactionNumber") with(pages_per_range=128);
```

```
explain analyze select * from "Purchase" where "BankTransactionNumber" = 'UA35253253523' ;
```

```
explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523';
```

```
explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
```

explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';

```
SQL Shell (psql)
Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=34) (actual time=0.049..0.051 rows=1 loops=1)
  Index Cond: (("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.143 ms
Execution Time: 0.075 ms
(4 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523';
QUERY PLAN
Aggregate (cost=947.17..947.18 rows=1 width=8) (actual time=13.702..13.704 rows=1 loops=1)
-> Seq Scan on "Purchase" (cost=0.00..846.31 rows=40344 width=8) (actual time=0.019..9.226 rows=40344 loops=1)
  Filter: (("BankTransactionNumber")::text <> 'UA35253253523')::text)
  Rows Removed by Filter: 1
Planning Time: 0.115 ms
Execution Time: 13.752 ms
(6 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
QUERY PLAN
Aggregate (cost=8.31..8.32 rows=1 width=8) (actual time=0.003..0.003 rows=1 loops=1)
-> Result (cost=0.29..8.31 rows=1 width=8) (actual time=0.000..0.001 rows=0 loops=1)
  One-Time Filter: (false AND false)
  -> Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=8) (never executed)
    Index Cond: (("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.148 ms
Execution Time: 0.032 ms
(7 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';
QUERY PLAN
Aggregate (cost=1249.75..1249.76 rows=1 width=8) (actual time=10.656..10.656 rows=1 loops=1)
-> Seq Scan on "Purchase" (cost=0.00..1148.90 rows=40341 width=8) (actual time=0.023..8.712 rows=40341 loops=1)
  Filter: (((("BankTransactionNumber")::text <> 'UA35253253523')::text) AND (("BankTransactionNumber")::text <> 'US352355')::text) AND (("BankTransactionNumber")::text <> 'UA19658577')::text) AND (("BankTransactionNumber")::text <> 'UA28533621')::text)
  Rows Removed by Filter: 4
Planning Time: 0.094 ms
Execution Time: 10.678 ms
(6 rows)

Lab1BookShop=#
```

```
SQL Shell (psql)
Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=34) (actual time=0.054..0.057 rows=1 loops=1)
  Index Cond: (("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.244 ms
Execution Time: 0.100 ms
(4 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523';
QUERY PLAN
Aggregate (cost=947.17..947.18 rows=1 width=8) (actual time=8.418..8.419 rows=1 loops=1)
-> Seq Scan on "Purchase" (cost=0.00..846.31 rows=40344 width=8) (actual time=0.021..5.909 rows=40344 loops=1)
  Filter: (("BankTransactionNumber")::text <> 'UA35253253523')::text)
  Rows Removed by Filter: 1
Planning Time: 0.145 ms
Execution Time: 8.455 ms
(6 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" = 'UA35253253523' and "BankTransactionNumber" = 'US352355' and "BankTransactionNumber" = 'UA19658577';
QUERY PLAN
Aggregate (cost=8.31..8.32 rows=1 width=8) (actual time=0.006..0.007 rows=1 loops=1)
-> Result (cost=0.29..8.31 rows=1 width=8) (actual time=0.001..0.001 rows=0 loops=1)
  One-Time Filter: (false AND false)
  -> Index Scan using or_index on "Purchase" (cost=0.29..8.31 rows=1 width=8) (never executed)
    Index Cond: (("BankTransactionNumber")::text = 'UA35253253523')::text)
Planning Time: 0.354 ms
Execution Time: 0.074 ms
(7 rows)

Lab1BookShop=# explain analyze select count("Sum") from "Purchase" where "BankTransactionNumber" != 'UA35253253523' and "BankTransactionNumber" != 'US352355' and "BankTransactionNumber" != 'UA19658577' and "BankTransactionNumber" != 'UA28533621';
QUERY PLAN
Aggregate (cost=1249.75..1249.76 rows=1 width=8) (actual time=16.504..16.504 rows=1 loops=1)
-> Seq Scan on "Purchase" (cost=0.00..1148.90 rows=40341 width=8) (actual time=0.023..12.757 rows=40341 loops=1)
  Filter: (((("BankTransactionNumber")::text <> 'UA35253253523')::text) AND (("BankTransactionNumber")::text <> 'US352355')::text) AND (("BankTransactionNumber")::text <> 'UA19658577')::text) AND (("BankTransactionNumber")::text <> 'UA28533621')::text)
  Rows Removed by Filter: 4
Planning Time: 0.122 ms
Execution Time: 16.528 ms
(6 rows)

Lab1BookShop=#
Lab1BookShop=#
```

BRIN показує не сильно швидші результати ніж без індексування, в деяких випадках воно швидше, а в деяких повільніше, ніж без індексування. Але це пов'язано з особливістю даного індексу а саме не в швидкодія

знаходження потрібних рядків, а уникнення перегляду зарання непотрібних даних.

- для завдання №3: команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних;

--after update, insert

Створення таблиці BookLog(Audit) баз даних для роботи з тригером:

```
DROP TRIGGER IF EXISTS "after_update_insert_trigger" ON "Book";
```

```
DROP TABLE "BookLog";
```

```
CREATE TABLE "BookLog"(  
  "ID" serial PRIMARY KEY,  
  "BookLogID" BIGINT,  
  "BookName" text  
);
```

Створення тригера:

```
CREATE OR REPLACE FUNCTION after_update_insert_func() RETURNS  
TRIGGER as $trigger$
```

```
DECLARE
```

```
BEGIN
```

```
  IF new."PageCount" <= 350 THEN
```

```
    RAISE NOTICE 'PageCount <= 350, so the book is short.';
```

```
    INSERT INTO "BookLog"("BookLogID", "BookName")  
VALUES (new."BookID", new."Name" || ' (Short)');
```

```
  ELSE
```



```

        RAISE NOTICE 'PageCount >= 350, so the book is long.';

        INSERT INTO "BookLog"("BookLogID", "BookName")
VALUES (new."BookID", new."Name" || ' (Long)');

    END IF;

    IF new."PageCount" = 666 THEN

        RAISE EXCEPTION 'PageCount == 666, so the book is unlucky
and should not be present in our shop.';

    END IF;

RETURN NEW;

END;

$trigger$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER "after_update_insert_trigger"
AFTER UPDATE OR INSERT ON "Book"
FOR EACH ROW
EXECUTE procedure after_update_insert_func();

```

Запити для перевірки роботи тригера:

```

INSERT INTO "Book"("BookID", "Name", "PageCount")
VALUES ('5321112353','Book 555', '340'), ('5122313','The Big Book 5', '909'),
('32123255', 'Python Book', '241') ;












```

```

UPDATE "Book" SET "PageCount" = 555 WHERE "Name" = 'The Big Book
5';

```

```
--UPDATE "Book" SET "PageCount" = 666 WHERE "Name" = 'Python Book';
```

Data output Messages Notifications			
       			
	ID [PK] integer 	BookLogID bigint 	BookName text 
1	1	5321112353	Book 555 (Short)
2	2	5122313	The Big Book 5 (Long)
3	3	32123255	Python Book (Short)
4	4	5122313	The Big Book 5 (Long)

Тригер виконує внесення рядку у таблицю аудиту (логу) і додає в кінці Short OR Long в залежності від того чи є кількість сторінок меншою чи більшою за 350, якщо кількість сторінок == 666, то генерується Exception.

По наведеному скріншоту можна побачити стан таблиці аудиту після виконання запиту.