

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**

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

Виконав:

студент ІII курсу

групи КВ-02

Недоколенко Д. Р.

Перевірив: Павловський В.І.

Київ – 2022-2023

*Мета роботи*: здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи:

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 у PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Варіант 19**

Види індексів: BTree, BRIN.

Умови для тригера: before insert, delete.

Посилання на Github: <https://github.com/Zarrylon/db_kpi>

**Логічна модель (схема) БД «Бібліотека»**

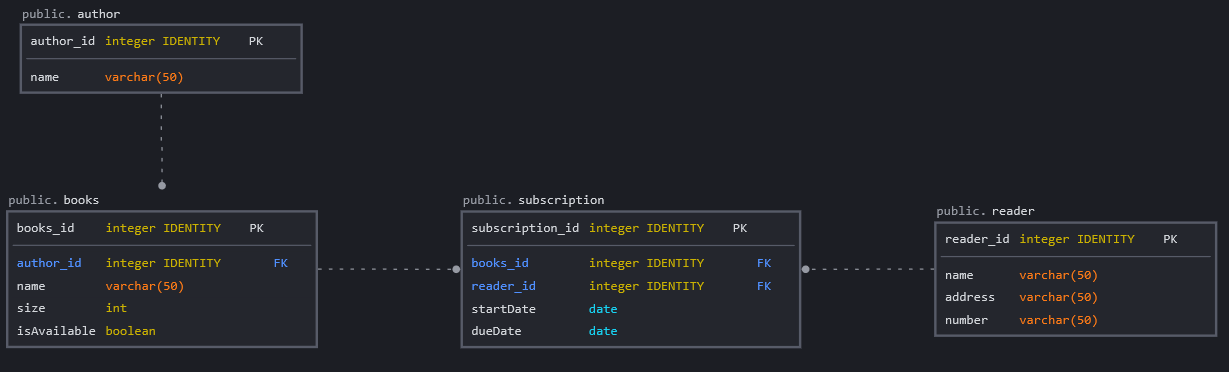
Логічну модель (схему бази даних) наведено на Рисунку 1.

Рисунок 1. – Схема бази даних (інструмент: sqldb.com)

Є незначні зміни в назвах: в isAvailable, startDate та dueDate була прибрана капіталізація.

**Класи ORM**

Model.py:

**from** sqlalchemy **import** create\_engine

**from** sqlalchemy **import** Column, Integer, String, Boolean, Date, ForeignKey

**from** sqlalchemy.ext.declarative **import** declarative\_base

**from** sqlalchemy.orm **import** relationship, sessionmaker

*# author books subscription reader*

*# author 1:M reader*

*# books M:N sub*

*# reader M:N sub*

engine = create\_engine('postgresql+psycopg2://postgres:qwerty'

'@localhost:5434/Library', echo=True)

Session = sessionmaker(bind=engine)

s = Session()

Base = declarative\_base()

**def** recreate\_db():

Base.metadata.drop\_all(engine)

Base.metadata.create\_all(engine)

**class** author(Base):

\_\_tablename\_\_ = 'author'

author\_id = Column(Integer, primary\_key=True)

name = Column(String)

bk = relationship("books", order\_by="books.books\_id",

back\_populates="aut")

**class** subscription(Base):

\_\_tablename\_\_ = 'subscription'

subscription\_id = Column(Integer, primary\_key=True)

books\_id = Column(Integer, ForeignKey('books.books\_id'), primary\_key=True)

reader\_id = Column(Integer, ForeignKey('reader.reader\_id'), primary\_key=True)

startdate = Column(Date)

duedate = Column(Date)

**class** books(Base):

\_\_tablename\_\_ = 'books'

books\_id = Column(Integer, primary\_key=True)

author\_id = Column(Integer, ForeignKey('author.author\_id'))

name = Column(String)

size = Column(Integer)

isavailable = Column(Boolean)

aut = relationship("author", back\_populates="bk")

rd = relationship("reader", secondary='subscription', back\_populates='bk')

**class** reader(Base):

\_\_tablename\_\_ = 'reader'

reader\_id = Column(Integer, primary\_key=True)

name = Column(String)

address = Column(String)

number = Column(String)

bk = relationship("books", secondary='subscription', back\_populates='rd')

**class** Model\_db:

**def** \_\_init\_\_(self):

self.session = Session()

self.connection = engine.connect()

@staticmethod

**def** insert\_data(table\_name, values):

**if** table\_name == 'author':

s.add(author(author\_id=values[0], name=values[1]))

s.commit()

**elif** table\_name == 'books':

s.add(books(books\_id=values[0], author\_id=values[1], name=values[2],

size=values[3], isavailable=bool(values[4])))

s.commit()

**elif** table\_name == 'reader':

s.add(reader(reader\_id=values[0], name=values[1], address=values[2],

number=values[3]))

s.commit()

**elif** table\_name == 'subscription':

s.add(subscription(subscription\_id=values[0],

books\_id=values[1], reader\_id=values[2],

startdate=values[3], duedate=values[4]))

s.commit()

@staticmethod

**def** delete\_data(table\_name, value\_id):

**if** table\_name == 'author':

s.query(author).filter\_by(author\_id=value\_id).delete()

s.commit()

**elif** table\_name == 'books':

s.query(books).filter\_by(books\_id=value\_id).delete()

s.commit()

**elif** table\_name == 'reader':

s.query(reader).filter\_by(reader\_id=value\_id).delete()

s.commit()

**elif** table\_name == 'subscription':

s.query(subscription).filter\_by(subscription\_id=value\_id).delete()

s.commit()

@staticmethod

**def** update\_data(table\_name, values):

**if** table\_name == 'author':

s.query(author).filter\_by(author\_id=values[0]).update({author.name: values[1]})

s.commit()

**elif** table\_name == 'books':

s.query(books).filter\_by(books\_id=values[0]).update(

{books.name: values[1], books.size: values[2],

books.isavailable: values[3]})

s.commit()

**elif** table\_name == 'reader':

s.query(reader).filter\_by(reader\_id=values[0]).update({reader.name: values[1],

reader.address: values[2],

reader.number: values[3]})

s.commit()

**elif** table\_name == 'subscription':

s.query(subscription).filter\_by(subscription\_id=values[0]).update({subscription.startdate: values[1],

subscription.duedate: values[2]})

s.commit()

**Приклади запитів ORM**

1. Вставка рядка у таблицю. Робота запиту зображена на Рисунку 2.

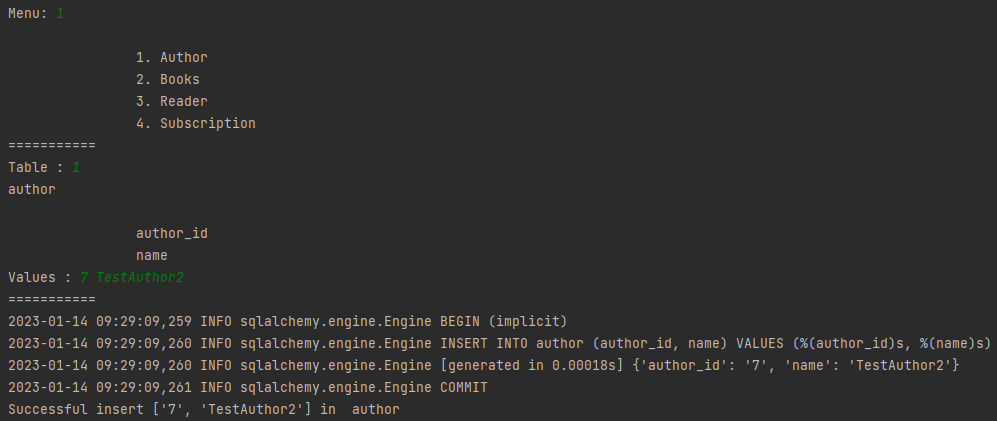


Рисунок 2 – Вставка рядка з author\_id = 7 в таблицю «author»

1. Видалення рядка таблиці. Робота запиту зображена на Рисунку 3.

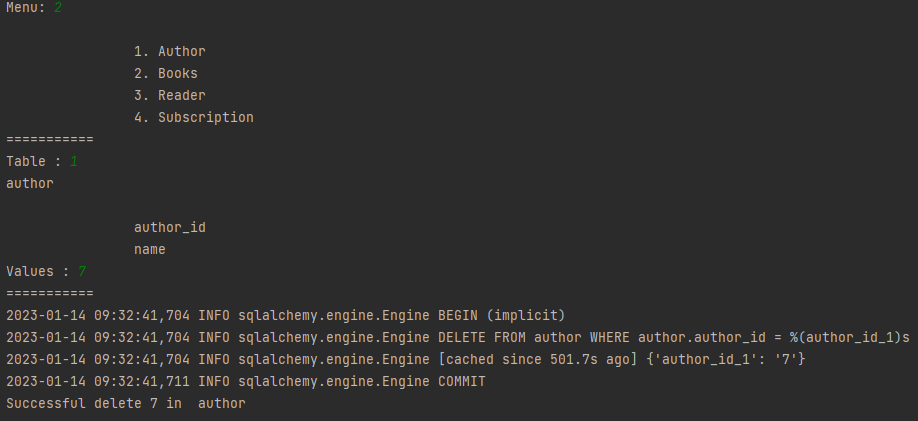


Рисунок 3 – Видалення рядка з author\_id = 7 з таблиці «author»

1. Оновлення даних рядка таблиці. Таблиця «reader» до вставки зображена на Рисунку 4. Робота запиту оновлення зображена на Рисунку 5. Результат роботи зображений на Рисунку 6.

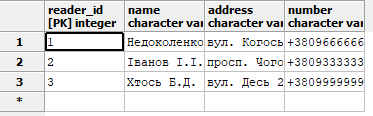


Рисунок 4 – Вміст таблиці «reader» (pgAdmin 3)

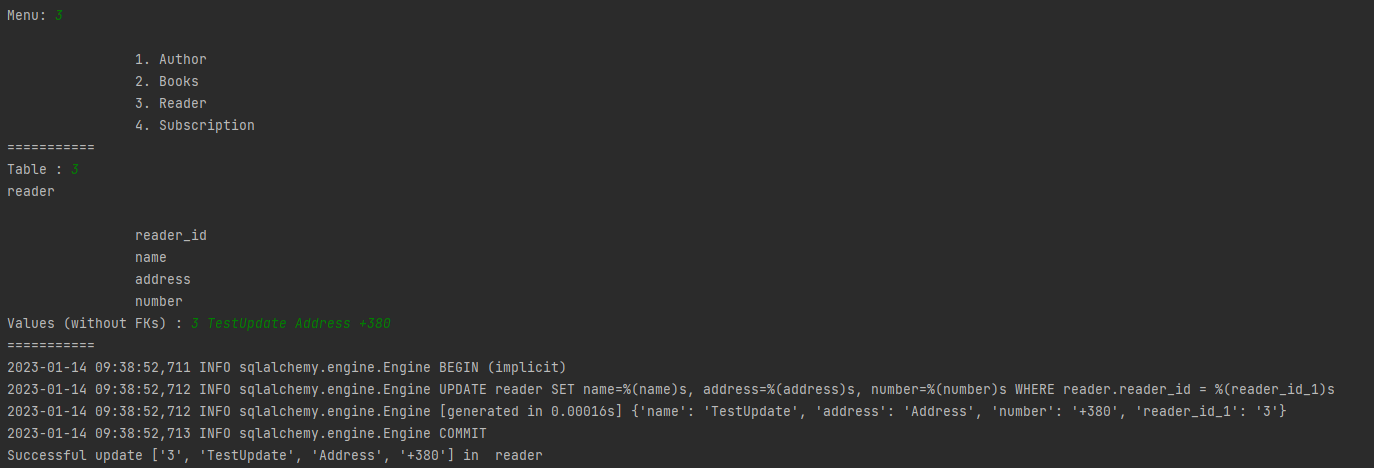


Рисунок 5 – Оновлення даних рядка з reader\_id = 3

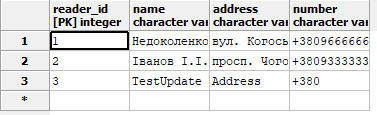


Рисунок 5 – Вміст таблиці «reader» після оновлення (pgAdmin 3)

**Індекс BTree**

Для дослідження індексу була створена таблиця з 100000 записами, яка зображена на Рисунку 6. Колонка name проіндексована як BTree.

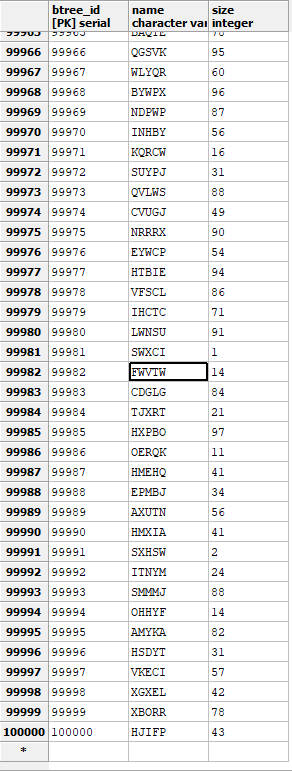


Рисунок 6 – Фрагмент таблиці «btree\_t» (pgAdmin 3)

Створення таблиці та її заповнення:

*-- Table: btree\_t*

*-- DROP TABLE btree\_t;*

**CREATE** **TABLE** btree\_t

(

btree\_id serial **NOT** **NULL**,

**name** **character varying** **NOT** **NULL**,

size **integer** **NOT** **NULL**,

**CONSTRAINT** btreepk **PRIMARY** **KEY** (btree\_id)

)

**WITH** (

**OIDS**=**FALSE**

);

**ALTER** **TABLE** btree\_t

**OWNER** **TO** postgres;

**insert** **into** btree\_t(**name**, size)

**SELECT** concat(**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**)),

**floor**(**random**() \* 100 + 1)::**int**

**FROM** **generate\_series**(1, 100000) ;

Запит на створення індексу:

**DROP** **INDEX** **IF** **EXISTS** btree\_test;

**CREATE** **INDEX** btree\_test **ON** btree\_t **USING** btree (**name**);

Для пошуку було використано 4 запити. Порівняємо швидкість з та без індексу.

1. SELECT \* FROM btree\_t where name='FWVTW' and size=14;

Без індексування (Рисунок 7):

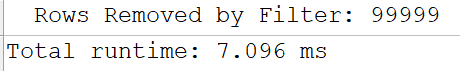


Рисунок 7 – Швидкодія першого запиту без індексу

З індексуванням BTree (Рисунок 8):



Рисунок 8 – Швидкодія першого запиту з індексом BTree

1. SELECT \* FROM btree\_t ORDER BY btree\_id DESC;

Без індексування (Рисунок 9):



Рисунок 9 – Швидкодія другого запиту без індексу

З індексуванням BTree (Рисунок 10):



Рисунок 10 – Швидкодія другого запиту з індексом BTree

1. SELECT COUNT(btree\_id), SUM(btree\_id) FROM btree\_t WHERE name LIKE '%TE%' GROUP BY btree\_id % 4;

Без індексування (Рисунок 11):

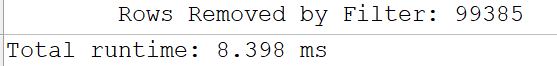


Рисунок 11 – Швидкодія третього запиту без індексу

З індексуванням BTree (Рисунок 12):

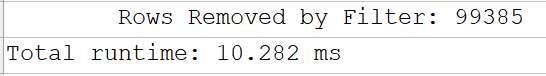


Рисунок 12 - Швидкодія третього запиту з індексом BTree

1. SELECT \* FROM btree\_t WHERE btree\_id < 1000 OR btree\_id > 50000 AND name LIKE 'AQ%';

Без індексування (Рисунок 13):

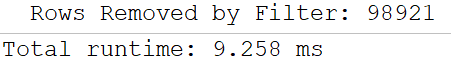


Рисунок 13 - Швидкодія четвертого запиту без індексу

З індексуванням BTree (Рисунок 14):

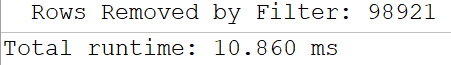


Рисунок 14 - Швидкодія четвертого запиту з індексом Btree

Даний метод індексування ідеально підходить для пошуку унікальних простих значень і як видно в 1 запиті, різниця в швидкості значна. В 2 запиті видно, що використання усіх даних не є ефективним і не дуже прискорює роботу. 3 та 4 запити показують в яких випадках не доцільно використовувати даний спосіб індексування.

**Індекс BRIN**

Для дослідження індексу була створена таблиця з 100000 записами, яка зображена на Рисунку 15. Колонка size проіндексована як BRIN.

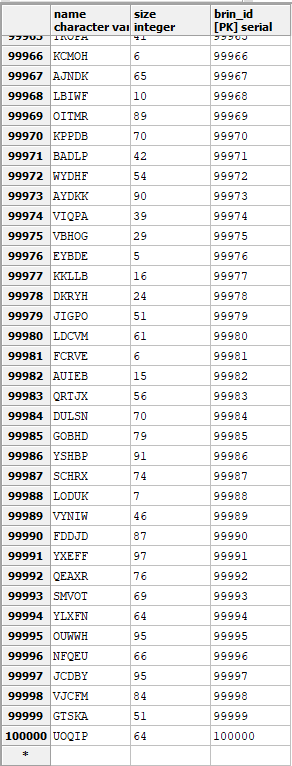


Рисунок 15 – Фрагмент таблиці «brin\_t» (pgAdmin 3)

Створення таблиці та її заповнення:

*-- Table: brin\_t*

*-- DROP TABLE brin\_t;*

**CREATE** **TABLE** brin\_t

(

**name** **character varying** **NOT** **NULL**,

size **integer** **NOT** **NULL**,

brin\_id serial **NOT** **NULL**,

**CONSTRAINT** brin\_pk **PRIMARY** **KEY** (brin\_id)

)

**WITH** (

**OIDS**=**FALSE**

);

**ALTER** **TABLE** brin\_t

**OWNER** **TO** postgres;

**insert** **into** brin\_t(**name**, size)

**SELECT** concat(**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**),

**chr**(**floor**(**random**() \* 25 + 65)::**int**)),

**floor**(**random**() \* 100 + 1)::**int**

**FROM** **generate\_series**(1, 100000) ;

На жаль, в pgAdmin 3 не існує методу індексування BRIN (Рисунок 16).

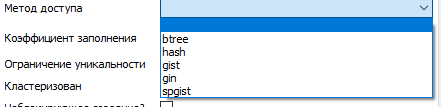


Рисунок 16 – Види доступних індексів (pgAdmin 3)

Даний метод ідеально підходить для індексування великої кількості природньо відсортованих даних. За часом пошуку, BRIN є повільнішим за BTree, проте займає дуже мало місця.

**Тригер бази даних PostgreSQL. Умова для тригера – before insert, delete**

Таблиця «subscription»:

*-- Table: subscription*

*-- DROP TABLE subscription;*

**CREATE** **TABLE** subscription

(

subscription\_id **integer** **NOT** **NULL** **DEFAULT** **nextval**('"Subscription\_subscription\_id\_seq"'::regclass),

books\_id **integer** **NOT** **NULL** **DEFAULT** **nextval**('"Subscription\_books\_id\_seq"'::regclass),

reader\_id **integer** **NOT** **NULL** **DEFAULT** **nextval**('"Subscription\_reader\_id\_seq"'::regclass),

startdate **date** **NOT** **NULL**,

duedate **date** **NOT** **NULL**,

**CONSTRAINT** "subscription\_idPK" **PRIMARY** **KEY** (subscription\_id),

**CONSTRAINT** "books\_idFK" **FOREIGN** **KEY** (books\_id)

**REFERENCES** books (books\_id) **MATCH** **FULL**

**ON** **UPDATE** **NO** **ACTION** **ON** **DELETE** **NO** **ACTION**,

**CONSTRAINT** "reader\_idFK" **FOREIGN** **KEY** (reader\_id)

**REFERENCES** reader (reader\_id) **MATCH** **FULL**

**ON** **UPDATE** **NO** **ACTION** **ON** **DELETE** **NO** **ACTION**

)

**WITH** (

**OIDS**=**FALSE**

);

**ALTER** **TABLE** subscription

**OWNER** **TO** postgres;

Тригер:

**CREATE** **FUNCTION** delete\_old\_sub() **RETURNS** **trigger** **AS** $$

**BEGIN**

**DELETE** **FROM** subscription **WHERE** reader\_id = **NEW**.reader\_id;

**RETURN** **NEW**;

**END**;

$$

**LANGUAGE** 'plpgsql';

**CREATE** **TRIGGER** delete\_before\_insert\_new\_sub

**BEFORE** **INSERT** **ON** subscription

**FOR** **EACH** **ROW** **EXECUTE** **PROCEDURE**

delete\_old\_sub();

**Принцип роботи тригеру:**

Тригер спрацьовує перед вставкою даних в таблицю «subscription». Перед додаванням нового абонементу усі старі абонементи з тим самим reader\_id будуть видалені.

**Приклади**

На Рисунку 17 зображено таблицю «subscription», а на Рисунках 18-19 - приклади роботи тригеру.

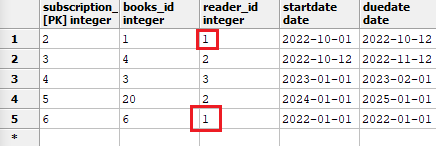


Рисунок 17 – Початковий вигляд таблиці «subscription» (pgAdmin 3)

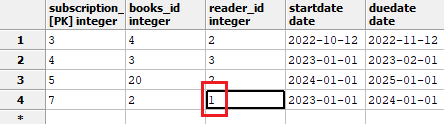


Рисунок 18 – Вигляд таблиці після додавання рядка з reader\_id = 1 (pgAdmin 3)

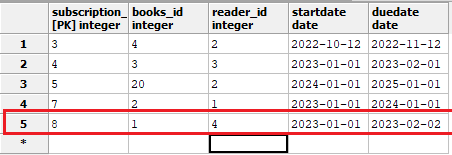


Рисунок 19 – Вигляд таблиці після додавання рядка з reader\_id = 4 (pgAdmin 3)

Як можна помітити, тригер працює правильно. На Рисунку 18 перед вставкою було видалено рядки з subscription\_id = 2 та 6, оскільки їх reader\_id співпадає з доданим рядком. На Рисунку 19 було додано рядок з reader\_id = 4, такого значення в таблиці не було, тому виконалась лише вставка нового рядка.