НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота № 2

з дисципліни

«Бази даних та засоби управління»

Тема роботи: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент групи КВ-11
Карлаш Іван

Telegram: @jnguar

Київ – 2023

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

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

Завдання *роботи полягає у наступному:*

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктнореляційної 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.

Посилання на Github: https://github.com/vaniapelmen/DB_lab

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.

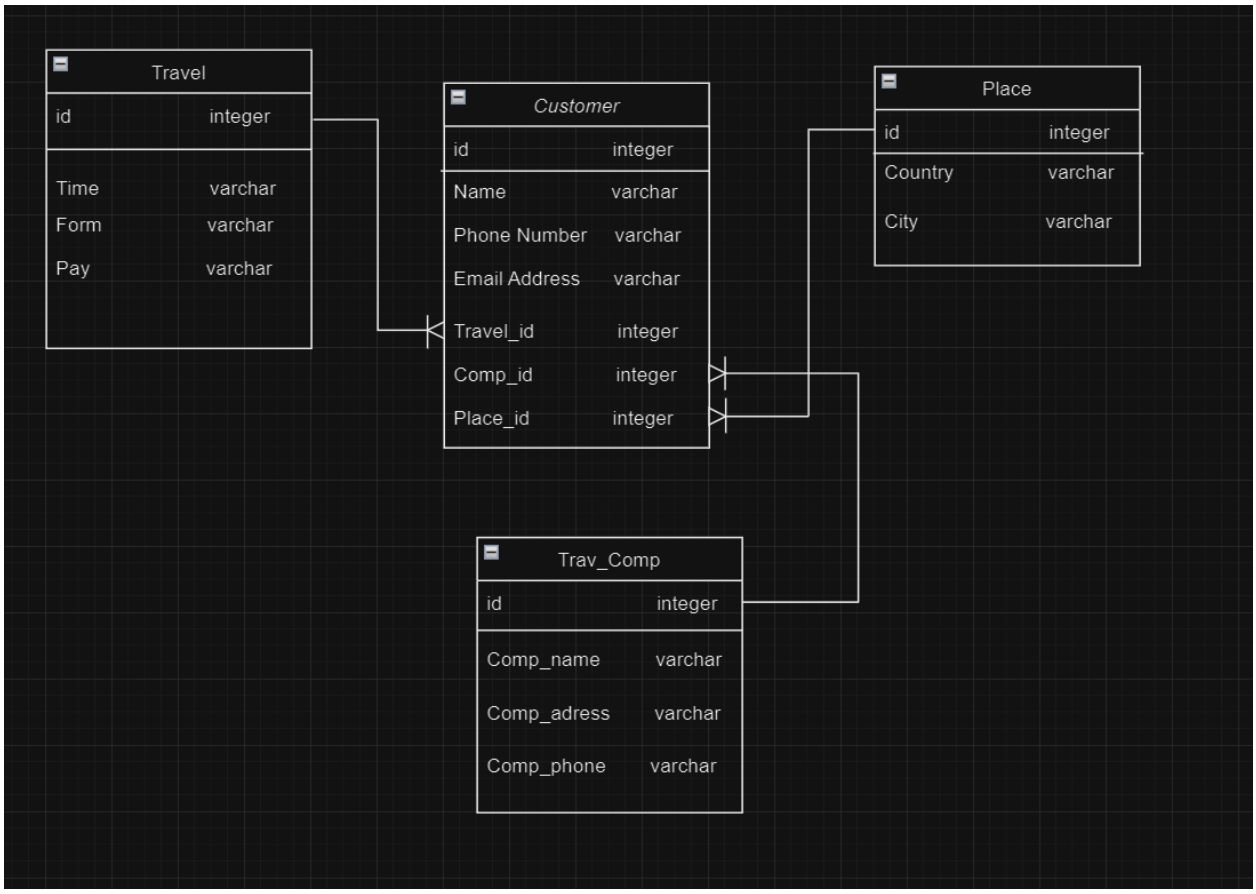
Вимоги до пункту завдання №4

Проаналізувати на прикладах використання рівнів ізоляції транзакцій READ COMMITTED, REPEATABLE READ та SERIALIZABLE, продемонструвавши феномени, які виникають, і спосіб їх уникнення завдяки встановленню відповідного рівня ізоляції транзакцій. Для виконання завдання необхідно відкрити дві транзакції у різних вікнах pgAdmin4 і виконати послідовність запитів INSERT, UPDATE або DELETE у обох транзакціях, що доводять наявність або відсутність певних феноменів.

Варіант №9

№ варіанта	*Види індексів*	*Умови для тригера*
9	*BTree, BRIN*	*before delete, update*

Скріншот розробленої моделі «сутність-зв’язок»:



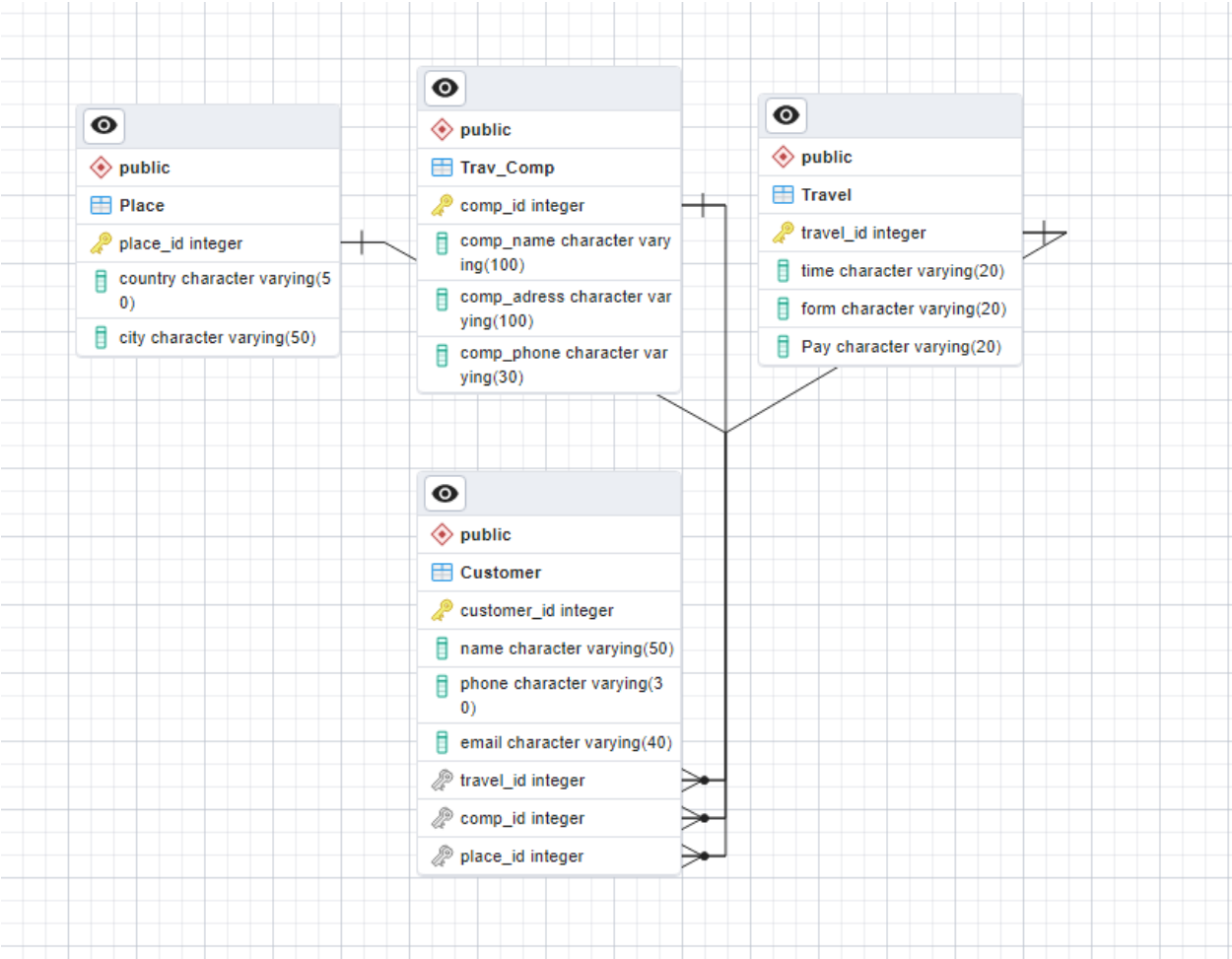
Опис сутностей:

1) Сутність «Customer» (покупець) має 7 атрибутів:
    - Id – ідентифікатор покупця, змінюється відповідно кількості покупців;
    - Name – текстове поле, що позначає ім’я покупця;
    - Phone Number – текстове поле, що позначає номер телефону покупця;
    - Email Adress – текстове поле, що позначає емеіл адрессу;
    - Travel_id – зв’язок з сутністю Travel;
    - Comp_id – зв’язок з сутністю Trav_Comp;
    - Place_id - зв’язок з сутністю Place;
2) Сутність «Travel» (подорож) має 4 атрибути:
    - Id – унікальний ідентифікатор поїздки;
    - Time – час відправлення;
    - Form – Якість поїздки(преміум, звичайна);
    - Pay – Статус оплати, оплачено, чи ні;
3) Сутність «Trav_comp» (компанія транспортування) має 4 атрибути:
    - Id – ідентифікатор транспортної компанії;
    - Comp_name – текстове поле, з назвою компанії;
    - Comp_adress – текстове поле з адресою компанії;

- Comp_phone – текстове поле з номером телефону компанії;

4) Сутність «Place» (місце подорожі) має 3 атрибути:
   - Id – ідентифікатор місця подорожі;
   - Country – текстове поле, назва країни;
   - City – текстове поле, назва міста;

Схема у графічному вигляді:



Завдання №1

У даній лабораторній роботі було реалізовано 4 класи відповідно до 4 існуючих таблиць у розробленій базі даних, а саме:

1. *Customer*
2. *Trcomp*
3. *Travel*
4. *Place*

Customer

Таблиця customer має наступні стовпці: customer_id(ідентифікатор покупця), name(його ім'я), phone(номер телефону), email(емеіл), travel_id(ід подорожі), comp_id(ід компанії перевізника), place_id(ід місця подорожі), пов'язана з іншими таблицями через відповідні ід.

Програмна реалізація:

```
class customer(Base):
    __tablename__ = 'customer'
    customer_id = Column(Integer, primary_key=True)
    name = Column(String)
    phome = Column(String)
    email = Column(String)

    travel_id = Column(Integer, ForeignKey('travel.travel_id'))
    comp_id= Column(Integer, ForeignKey('trcomp.comp_id'))
    place_id= Column(Integer, ForeignKey('place.place_id'))

        def __init__(self, name, phone, email, travel_id, comp_id, place_id):
        self.name = name
        self.phone = phone
        self.email = email
        self.travel_id = travel_id
        self.comp_id = comp_id
        self.place_id = place_id

    def __repr__(self):
        return f"<customer(customer_id={self.customer_id}, name={self.name},
phone={self.phone}, email={self.email}, travel_id={self.travel_id},
comp_id={self.comp_id}}, place_id={self.place_id})>"
```

trcomp

Таблиця trcomp має наступні стовпці: comp_id(ідентифікатор компанії), comp_name(назва компанії), comp_adress(адрес компанії), comp_phone(номер телефону, ця таблиця зв'язана з таблицею customer використовуючи comp_id.

Програмна реалізація:

```
class trcomp(Base):
    __tablename__ = 'trcomp'
    comp_id = Column(Integer, primary_key=True)
    comp_name = Column(String)
    comp_address = Column(String)
    comp_phone = Column(String)

    customer = relationship("customer")

    def __init__(self, comp_name, comp_address, comp_phone):
        self.comp_name = comp_name
        self.comp_address = comp_address
        self.comp_phone = comp_phone

    def __repr__(self):
        return f"<trcomp(comp_id={self.comp_id}, comp_name={self.comp_name},
comp_address={self.comp_address}, comp_phone={self.comp_phone},
phone_number={self.phone_number})>"
```

travel

Таблиця travel має наступні стовпці: travel_id(ідентифікатор рейсу), time(час відправлення), form(тип місця), pay(статус оплати), ця таблиця зв’язана з таблицею customer використовуючи travel_id.

Програмна реалізація:

```
class travel(Base):
    __tablename__ = 'travel'
    travel_id = Column(Integer, primary_key=True)
    time = Column(String)
    form = Column(String)
    pay = Column(String)

    customer = relationship("customer")

    def __init__(self, time, form, pay):
        self.time = time
        self.form = form
        self.pay = pay

    def __repr__(self):
        return f"<travel(travel_id={self.travel_id}, time={self.time},
form={self.form}, pay={self.pay})>"
```

place

Таблиця place має наступні стовпці: place_id(ідентифікатор місця подорожі), country(країна подорожі), city(місто подорожі), ця таблиця зв’язана з таблицею customer використовуючи place_id.

Програмна реалізація:

```
class place(Base):
    __tablename__ = 'place'
    place_id = Column(Integer, primary_key=True)
    country = Column(String)
    city = Column(String)

    customer = relationship("customer")

    def __init__(self, country, city):
        self.country = country
        self.city = city

    def __repr__(self):
        return f"<place(place_id={self.place_id}, country={self.country},
city={self.city})>"
```

Вигляд меню:

```
Menu:
1.Add line
2.Show table
3.Update line
4.Delete line
5.Exit
Choose:
```

Пункт №1 (Add line)

```
Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table:
```

Після вибору таблиці потрібно буде ввести всі потрібні дані, для додання рядка;

Пункт №2 (Show table)

```
Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table:
```

Після вибору таблиці буде виведена інформація таблиці;

Пункт №3 (Update line)

```
Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table:
```

Після вибору таблиці потрібно буде ввести рядок таблиці який необхідно змінить, а потім ввести нові дані;

Пункт №4 (Delete line)

```
Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table:
```

Після вибору таблиці потрібно ввести ід рядка який потрібно видалити;

Пункт №5 (Exit)

Виходить з програми;

Приклади запитів у вигляді ORM

Запити вставки реалізовані за допомогою функцій insert. Спочатку в меню користувач обирає опцію додавання, далі обирає таблицю, до якої хоче додати рядок і вводить необхідні дані.

Додамо рядок в таблицю customer:

	customer_id [PK] integer	name character varying (50)	phone character varying (30)	email character varying (40)	travel_id integer	comp_id integer	place_id integer
1	2	Maria Lonsey	637562834	lonsi_sas@gmail.com	3	2	5
2	3	Ilon Conra	964539821	g123kh@gmai.com	4	1	2
3	4	Rian Valdoro	638354286	samsobi@gmail.com	4	2	2
4	5	Lina Plaso	976245923	linopla@gmail.com	1	3	6
5	6	Tot Otot	675463865	totttot@gmail.com	1	2	3
6	7	OJSW	975553649	K64FK	2	2	2
7	8	KQPP	732369843	H8KH	2	3	4
8	9	ACGE	747826667	I77KK	3	4	3
9	10	RKCO	236068775	Q31FT	2	2	1

```
Menu:
1.Add line
2.Show table
3.Update line
4.Delete line
5.Exit
Choose: 1

Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table: 1

Adding customer:
Enter customer name: Norvey
Enter customer phone: 0978654325
Enter customer email: nor@gmail.com
Enter customer travel_id: 1
Enter customer comp_id: 1
Enter customer place_id: 2
Customer added successfully!
```

Після вставки:

	customer_id [PK] integer	name character varying (50)	phone character varying (30)	email character varying (40)	travel_id integer	comp_id integer	place_id integer
1	1	Norvey	978654325	nor@gmail.com	1	1	2
2	2	Maria Lonsey	637562834	lonsi_sas@gmail.com	3	2	5
3	3	Ilon Conra	964539821	g123kh@gmai.com	4	1	2
4	4	Rian Valdoro	638354286	samsobi@gmail.com	4	2	2
5	5	Lina Plaso	976245923	linopla@gmail.com	1	3	6
6	6	Tot Otot	675463865	totttot@gmail.com	1	2	3
7	7	OJSW	975553649	K64FK	2	2	2
8	8	KQPP	732369843	H8KH	2	3	4
9	9	ACGE	747826667	I77KK	3	4	3
10	10	RKCO	236068775	Q31FT	2	2	1

Лістинг функцій insert для кожної таблиці

```
@staticmethod
def insert_customer(name: str, phone: str, email: str, travel_id: int,
comp_id: int, place_id: int) -> None:
    Customer = customer(name=name, phone=phone, email=email,
travel_id=travel_id, comp_id=comp_id, place_id=place_id)
    s.add(Customer)
    s.commit()

@staticmethod
def insert_trcomp(comp_name: str, comp_address: str, comp_phone: str) ->
None:
    Trcomp = trcomp(comp_name=comp_name, comp_address=comp_address,
comp_phone=comp_phone)
    s.add(Trcomp)
    s.commit()

@staticmethod
def insert_travel(time: str, form: str, pay: str) -> None:
    Travel = travel(time=time, form=form, pay=pay)
    s.add(Travel)
    s.commit()

@staticmethod
def insert_place(country: str, city: str) -> None:
    Place = place(country=country, city=city)
    s.add(Place)
    s.commit()
```

Запити показу реалізовані за допомогою функцій show. Спочатку в меню користувач обирає опцію показу, далі обирає таблицю, яку хоче побачити.

Подивимось таблицю trcomp:

	comp_id [PK] integer	comp_name character varying (100)	comp_address character varying (100)	comp_phone character varying (30)
1	1	Altion	Lviv, Ukrain	+380976548723
2	2	Eurostar	Paris, France	+380674563421
3	3	Eurolines	Brussels, Belgium	+380936740326
4	4	Lambada	Kyiv, Ukrane	+380976543723

```
Menu:
1.Add line
2.Show table
3.Update line
4.Delete line
5.Exit
Choose: 2

Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table: 2

trcomps:
ID: 2, Name: Eurostar, Address: Paris, France, Phone: +380674563421
ID: 3, Name: Eurolines, Address: Brussels, Belgium, Phone: +380936740326
ID: 4, Name: Lambada, Address: Kyiv, Ukrane, Phone: +380976543723
ID: 1, Name: Altion, Address: Lviv, Ukrain, Phone: +380976548723
```

Лістинг функцій show для кожної таблиці:

```
@staticmethod
def show_customers():
    return s.query(customer.name, customer.phone, customer.email,
customer.travel_id, customer.comp_id, customer.place_id).all()

@staticmethod
def show_trcomps():
    return s.query(trcomp.comp_name, trcomp.comp_address,
trcomp.comp_phone).all()

@staticmethod
def show_travels():
    return s.query(travel.time, travel.form, travel.pay).all()
```

```
@staticmethod
def show_places():
    return s.query(place.country, place.city).all()
```

Запит редагування реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Потім треба ввести всі необхідні дані для реадгування рядка.

Відредагуємо рядок таблиці place:

	place_id [PK] integer	country character varying (50)	city character varying (50)
1	1	Ukrain	Lviv
2	2	Poland	Krakow
3	3	Italy	Rome
4	4	Italy	Milan
5	5	France	Paris
6	6	Germany	Berlin
7	7	Latvia	Riga

```
Menu:
1.Add line
2.Show table
3.Update line
4.Delete line
5.Exit
Choose: 3

Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table: 4

Updating place:
Enter place ID: 2
Enter place country: Italy
Enter place city: Paris
Place updated successfully!
```

	place_id [PK] integer	country character varying (50)	city character varying (50)
1	1	Ukrain	Lviv
2	2	Italy	Paris
3	3	Italy	Rome
4	4	Italy	Milan
5	5	France	Paris
6	6	Germany	Berlin
7	7	Latvia	Riga

Лістинг функцій update для кожної таблиці:

```
@staticmethod
def update_customer(customer_id: int, name: str, phone: str, email: str,
travel_id: int, comp_id: int, place_id: int) -> None:

s.query(customer).filter_by(customer_id=customer_id).update({customer.name:
name, customer.phone: phone, customer.email: email, customer.travel_id:
travel_id, customer.comp_id: comp_id, customer.place_id: place_id})
    s.commit()

@staticmethod
def update_trcomp(comp_id: int, comp_name: str, comp_address: str,
comp_phone: str) -> None:
    s.query(trcomp).filter_by(comp_id=comp_id).update({trcomp.comp_name:
comp_name, trcomp.comp_address: comp_address, trcomp.comp_phone: comp_phone})
    s.commit()

@staticmethod
def update_travel(travel_id: int, time: str, form: str, pay: str) -> None:
    s.query(travel).filter_by(travel_id=travel_id).update({travel.time: time,
travel.form: form, travel.pay: pay})
    s.commit()

@staticmethod
def update_place(place_id: int, country: str, city: str) -> None:
    s.query(place).filter_by(place_id=place_id).update({place.country:
country, place.city: city})
    s.commit()
```

Запити видалення реалізовані за допомогою функцій delete. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора рядка для видалення.

Видалимо рядок з таблиці customer:

	customer_id [PK] integer	name character varying (50)	phone character varying (30)	email character varying (40)	travel_id integer	comp_id integer	place_id integer
1	1	Norvey	978654325	nor@gmail.com	1	1	2
2	2	Maria Lonsey	637562834	lonsi_sas@gmail.com	3	2	5
3	3	Ilon Conra	964539821	g123kh@gmai.com	4	1	2
4	4	Rian Valdoro	638354286	samsobi@gmail.com	4	2	2
5	5	Lina Plaso	976245923	linopla@gmail.com	1	3	6
6	6	Tot Otot	675463865	totttot@gmail.com	1	2	3
7	7	OJSW	975553649	K64FK	2	2	2
8	8	KQPP	732369843	H8KH	2	3	4
9	9	ACGE	747826667	I77KK	3	4	3
10	10	RKCO	236068775	Q31FT	2	2	1

```
Menu:
1.Add line
2.Show table
3.Update line
4.Delete line
5.Exit
Choose: 4

Tables:
1.Customer
2.Trav_comp
3.Travel
4.Place
5.Back to menu
Choose table: 1

Deleting customer:
Enter customer ID: 3
Customer deleted successfully!
```

Таблиця після видалення:

	customer_id [PK] integer	name character varying (50)	phone character varying (30)	email character varying (40)	travel_id integer	comp_id integer	place_id integer
1	1	Norvey	978654325	nor@gmail.com	1	1	2
2	2	Maria Lonsey	637562834	lonsi_sas@gmail.com	3	2	5
3	4	Rian Valdoro	638354286	samsobi@gmail.com	4	2	2
4	5	Lina Plaso	976245923	linopla@gmail.com	1	3	6
5	6	Tot Otot	675463865	totttot@gmail.com	1	2	3
6	7	OJSW	975553649	K64FK	2	2	2
7	8	KQPP	732369843	H8KH	2	3	4
8	9	ACGE	747826667	I77KK	3	4	3
9	10	RKCO	236068775	Q31FT	2	2	1

Лістинг функцій delete для кожної таблиці:

```
@staticmethod
def delete_customer(customer_id) -> None:
    Customer = s.query(customer).filter_by(customer_id=customer_id).one()
    s.delete(Customer)
    s.commit()

@staticmethod
def delete_trcomp(comp_id) -> None:
    Trcomp = s.query(trcomp).filter_by(comp_id=comp_id).one()
    s.delete(Trcomp)
    s.commit()


@staticmethod
def delete_travel(travel_id) -> None:
    Travel = s.query(travel).filter_by(travel_id=travel_id).one()
    s.delete(Travel)
    s.commit()

@staticmethod
def delete_place(place_id) -> None:
    Place = s.query(place).filter_by(place_id=place_id).one()
    s.delete(Place)
    s.commit()
```

Завдання №2

BTree

Btree — це збалансована деревоподібна структура даних, яка підтримує відсортовані дані та дозволяє здійснювати пошук, послідовний доступ, вставки та видалення в логарифмічному часі. Б-дерево узагальнює двійкове дерево пошуку, допускаючи вузли з більше ніж двома дочірніми елементами.

На відміну від інших самобалансуючих двійкових дерев пошуку, Б-дерево добре підходить для систем зберігання даних, які читають і записують відносно великі блоки даних, таких як бази даних і файлові системи.

Для дослідження створена таблиця tree_test, має колонки “id” та “name”:

```
CREATE TABLE "tree_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "tree_test"("id", "string")
SELECT generate_series as "id", md5(random()::text)
FROM generate_series(1, 1000000)
```

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

Data Output | Messages | Notifications

	id [PK] bigint	string character varying (100)
1	1	57d78d18999acd8cdaf10315d6693d...
2	2	41f355c1871dec47342c95d665ccecd4
3	3	58a6db339a65ab96b912056d402939...
4	4	bcbaae6ac0d669c53234e6a0392ecce5
5	5	1ca88a55fab1e22b5390f62bd8be46c3
6	6	3b2e6915ae1dfbc152e6f1fe31153ea5
7	7	1313a24bac91dd38a3e521910edb5f5e
8	8	c6acd764a282925f2e4f3e8f11de2a6e
9	9	bedd1d163fc9189c27867da2333068...
10	10	215a011db0cfb6385708b4f863cf48c8
11	11	e0cf41783b03ed3b30839a6ec536b825
12	12	96d7b3955a36ef61bb0d31f351fc6740
13	13	cdaffb2e3e334f5a4b5b382962e2e76e
14	14	bfc9fb3d9892aa7ec42c6013697c9cfb
15	15	96597440c7963cbc361248d8ad43d4...
16	16	3ec0199cf4bb5deed56f300a2a21532f
17	17	9c705851a937cb793837a3def393f3fb
18	18	cdbbdee9ea656eb68b0feff16c72a1cb
19	19	9c39c186150fc46eb003250c591f8496
20	20	69b4efbf434858f096698cd2e68483a4
21	21	066b50d73cabe71f7277a21c408a128c
22	22	7ee8e3d03d4b095c26488ee1a74c04bf
23	23	1e0382c7def625d90574e8040b6081...
24	24	52cdd65b70fe8f8a71187d40e544298e

Total rows: 1000 of 1000000 | Query complete 00:00:00.2

Для тестування візьмемо 5 запитів:

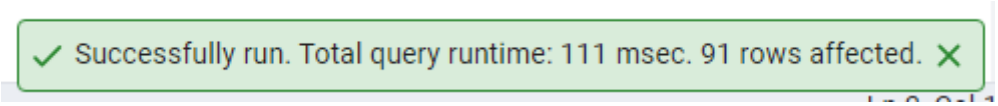
```
SELECT COUNT(*), "string" FROM tree_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";
SELECT * FROM tree_test ORDER BY "string" DESC;
SELECT * FROM tree_test WHERE "string" = 'e6ab3d8deb5a2d3aa7b71b175ae58aa5';
SELECT COUNT(*), "string" FROM tree_test GROUP BY "string";
SELECT COUNT(*), "string" FROM tree_test GROUP BY "string" ORDER BY COUNT(*) DESC;
```

Створення тригера:

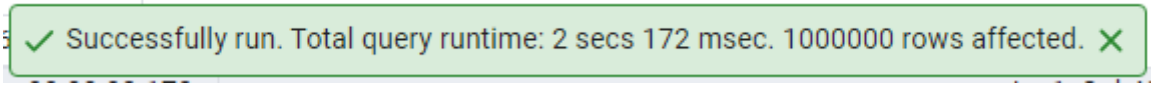
```
CREATE UNIQUE INDEX tree_index ON tree_test USING BTREE ("string");
```

Результати без використання Btree:

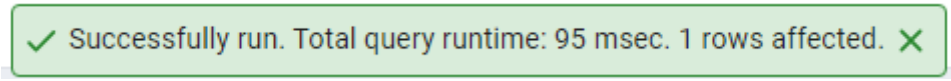
1. SELECT COUNT(*), "string" FROM tree_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";



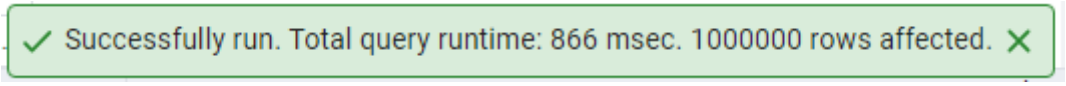
2. SELECT * FROM tree_test ORDER BY "string" DESC;



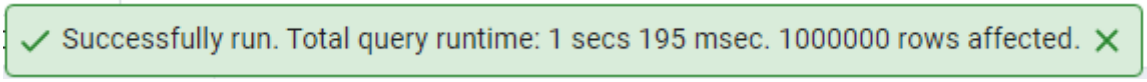
3. SELECT * FROM tree_test WHERE "string" = 'e6ab3d8deb5a2d3aa7b71b175ae58aa5';



4. SELECT COUNT(*), "string" FROM tree_test GROUP BY "string";

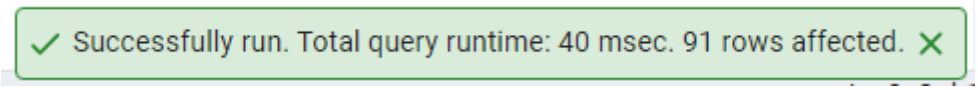


5. SELECT COUNT(*), "string" FROM tree_test GROUP BY "string" ORDER BY COUNT(*) DESC;

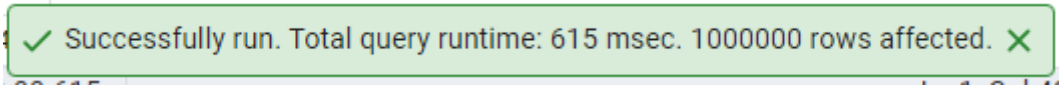


Результати з тригером:

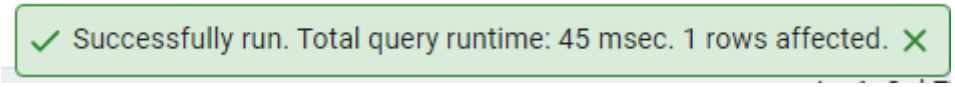
1. SELECT COUNT(*), "string" FROM tree_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";



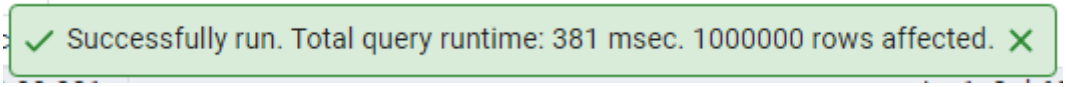
2. SELECT * FROM tree_test ORDER BY "string" DESC;



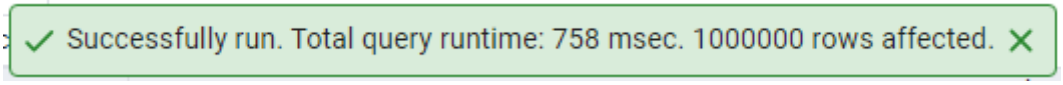
3. SELECT * FROM tree_test WHERE "string" = 'e6ab3d8deb5a2d3aa7b71b175ae58aa5';



4. SELECT COUNT(*), "string" FROM tree_test GROUP BY "string";



5. SELECT COUNT(*), "string" FROM tree_test GROUP BY "string" ORDER BY COUNT(*) DESC;



Висновок: Як бачимо всі запити значно збільшили швидкість виконання, особливо можна зазначити результати запитів 1, 2 і 3, адже кожен з них став працювати у 2-3 рази ефективніше, а запити 4 і 5, хоч і не показують таких високих результатів, але теж достатньо суттєво прискорились. Причина прискорення лежить в основах алгоритму, адже через свою структуру і спосіб упорядкування, він дозволяє ефективніше оперувати великою кількістю даних, що і демонструє наш приклад з 100000 рядків даних.

BRIN

Індекс BRIN (Block Range INdex) - це інший тип індексу в PostgreSQL, який призначений для оптимізації пошуку та фільтрації даних великих таблиць, особливо тих, які мають упорядкований порядок значень за деяким стовпцем. BRIN індекси призначені для ефективної обробки діапазонних запитів.

Основна ідея BRIN індексу полягає в тому, щоб розділити дані на блоки та зберігати індексовані значення для кожного блоку. Це дозволяє базі даних швидко визначати, які блоки можуть містити потрібні рядки, і використовувати індекс для обробки тільки цих блоків.

Створимо таблицю brin_test:

```
CREATE TABLE "brin_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "brin_test"("id", "string")
SELECT generate_series as "id", md5(random()::text)
FROM generate_series(1, 1000000)
```

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

Data Output | Messages | Notifications

	id [PK] bigint	string character varying (100)
1	1	9960deafbfd223b41d09e6341d1333...
2	2	e2c747a5df1f5d55e7b95188fd7aca94
3	3	becb69d3916e864e0afbad3d7859c7...
4	4	df4ee3c1c63387c5975431e8ab0406...
5	5	1222b82dd6f91854eb349e39a6744a...
6	6	9a46803596062061aa29af0d9e4ce6...
7	7	bc8bf337f169ad45cff6c9bd46c8d48c
8	8	ed3dc6e49dbcdb66811c1c93c3593f...
9	9	694c498dedc638ad97c5068a660868...
10	10	cd9439e3fe5e2a2798427c499f8a89a6
11	11	08633861a776a01dcd6e42197721fd...
12	12	e13ef051494e8abd34db10c34f9b2c...
13	13	b67f418892bdd9b339ddc1d47e6f1d...
14	14	279be555206116d755a2f425481f91...
15	15	3055daf171f2597b6e0b76e3ab984d2f
16	16	9a2f9cc9b8d2d269f8dd37da7bda95...
17	17	1cd366fa59c20a80dd319625e7fe42...
18	18	65da2ba3c720f72f7879d920d8994d...
19	19	d4adc1fd127e633f8de7c25bc1f5e42c

Total rows: 1000 of 1000000 | Query complete 00:00:00.4

Для тестування візьмемо 5 запитів:

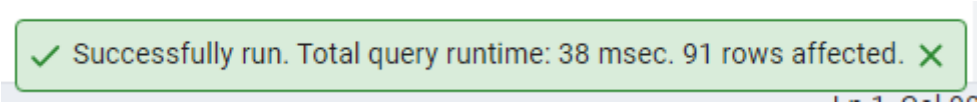
```
SELECT COUNT(*), "string" FROM brin_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";
SELECT * FROM brin_test ORDER BY "string" DESC;
SELECT * FROM brin_test WHERE "string" = 'e13ef051494e8abd34db10c34f9b2c50';
SELECT COUNT(*), "string" FROM brin_test GROUP BY "string";
SELECT COUNT(*), "string" FROM brin_test GROUP BY "string" ORDER BY COUNT(*) DESC;
```

Створення тригера BRIN:

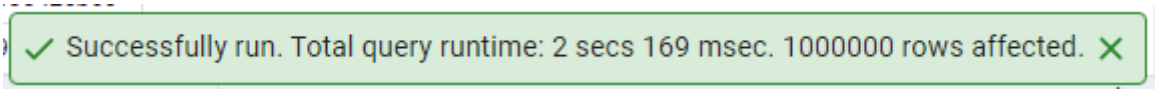
```
CREATE INDEX brin_index ON brin_test USING brin("string");
```

Результати без використання BRIN:

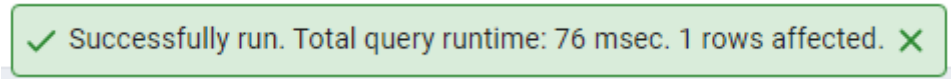
1. SELECT COUNT(*), "string" FROM brin_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";



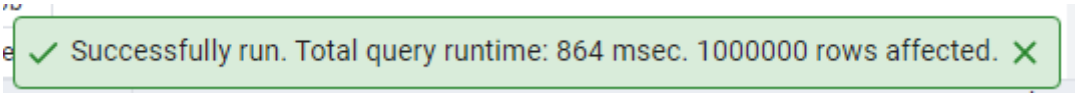
2. SELECT * FROM brin_test ORDER BY "string" DESC;



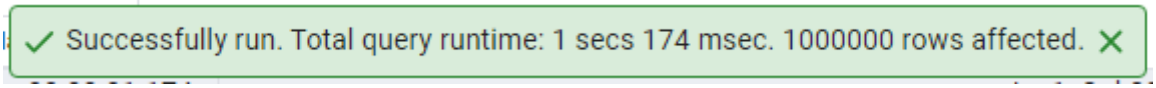
3. SELECT * FROM brin_test WHERE "string" = 'e13ef051494e8abd34db10c34f9b2c50';



4. SELECT COUNT(*), "string" FROM brin_test GROUP BY "string";

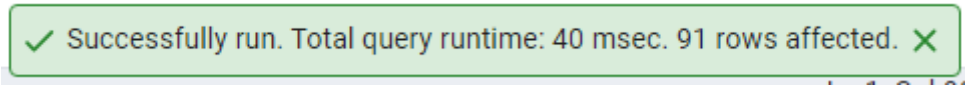


5. SELECT COUNT(*), "string" FROM brin_test GROUP BY "string" ORDER BY COUNT(*) DESC;

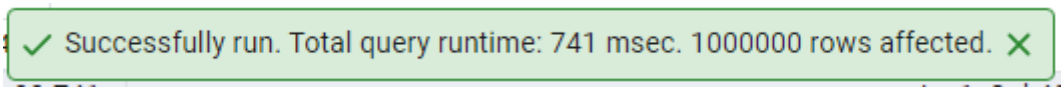


Результати з BRIN:

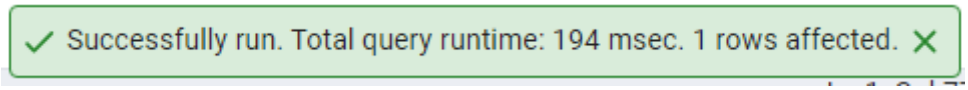
1. SELECT COUNT(*), "string" FROM brin_test WHERE "id" BETWEEN 10 AND 100 GROUP BY "string";



2. SELECT * FROM brin_test ORDER BY "string" DESC;

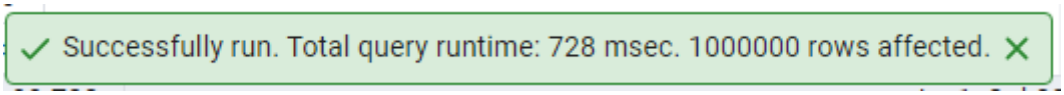


3. SELECT * FROM brin_test WHERE "string" = 'e13ef051494e8abd34db10c34f9b2c50';



4. SELECT COUNT(*), "string" FROM brin_test GROUP BY "string";

5. SELECT COUNT(*), "string" FROM brin_test GROUP BY "string" ORDER BY COUNT(*) DESC;



Висновок: Бачимо, що пришвидшились не всі запити. Запит №1 майже не змінив швидкості виконання, запит 2 дуже пришвидшився, запит 3, навпаки достатньо сильно сповільнився, а 4 і 5 просто непогано пришвидшились. Як бачимо BRIN не у всіх випадках гарно себе показує, у випадку пошуку конкретного елементу BRIN показує поганий результат.

Ефективність індексів може сильно залежати від конкретного сценарію використання, обсягу даних та конкретного плану виконання, який вибирає оптимізатор запитів бази даних.