



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 і засоби управління

на тему: “Засоби оптимізації СУБД PostgreSQL”

Виконав: студент III курсу

групи КВ-23

Домуші Д.Д.

Перевірів: _____

Київ – 2024

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’єктно-реляційної проєкції (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.

Варіант 8

Індекси для аналізу у завдання №2: Btree, Gin

Посилання на репозиторій: <https://github.com/asdqweghj/BDLAB2.git>

Телеграм: @cvvhella

Виконання роботи Сутності предметної області

1. Користувач(User), з атрибутами: код користувача, ім’я, прізвище, електронна пошта, номер телефона, дата реєстрації. Призначена для збереження інформації про користувачів платформи, що бронюють спортивні майданчики;
2. Спортивні Заклади (Venues), з атрибутами: код спортивного закладу, назва закладу, адреса, місто, місткість закладу. Призначена для зберігання даних про спортивні заклади, які пропонують можливості для занять спортом;
3. Майданчик/Зал (Facility), з атрибутами: код майданчика, код закладу, назва майданчика, тип майданчика. Призначена для зберігання інформації про майданчики, доступні для бронювання в спортивних закладах;
4. Бронювання (Booking), з атрибутами: код бронювання, код користувача, код майданчика, дата, час початку, час завершення, статус. Призначена для: управління бронюваннями користувачів на майданчиках у спортивних закладах;

5. Оплата (Payment), з атрибутами: код оплати, код бронювання, сума, дата оплати, статус. Призначена для зберігання інформації про оплати користувачів за бронювання майданчиків;

Зв'язки між сутностями предметної області

Сутність "Користувач" (User) має зв'язок 1:N по відношенню до сутності "Бронювання" (Booking), оскільки один користувач може створити кілька бронювань.

Сутність "Спортивний заклад" (Venue) має зв'язок 1:N по відношенню до сутності "Майданчик/Зал" (Facility), оскільки один спортивний заклад може містити кілька майданчиків або зон для занять.

Сутність "Майданчик/Зал" (Facility) має зв'язок 1:N по відношенню до сутності "Бронювання" (Booking), оскільки один майданчик може бути заброньований кілька разів на різні дати та час.

Сутність "Бронювання" (Booking) має зв'язок 1:1 по відношенню до сутності "Оплата" (Payment), оскільки кожне бронювання має лише одну відповідну оплату.

Графічне подання логічної моделі «Сутність-зв'язок» зображено на рисунку 1.

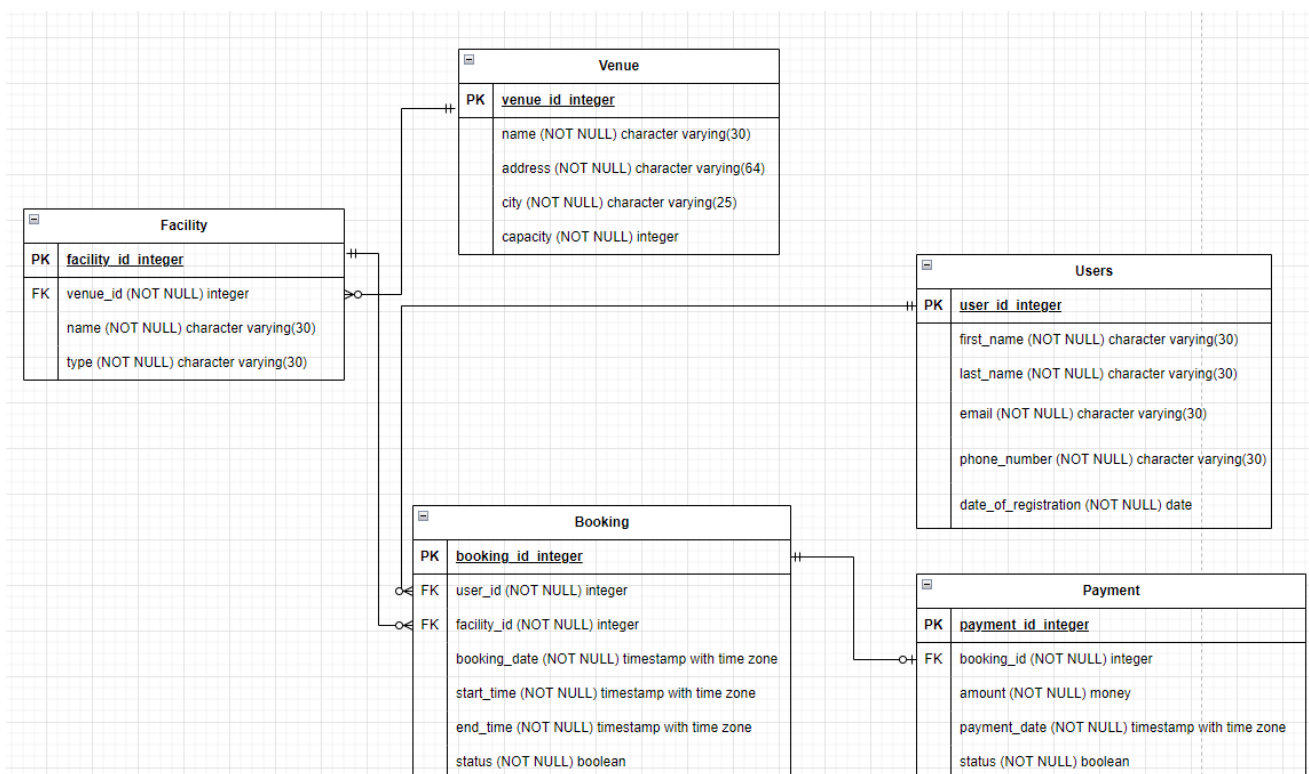


Рисунок 1– Логічна модель

Середовище для налаштування, підключення та розробки бази даних

Мова програмування : Python 3.12.3

Модуль “psycopg3” був використаний для підключення до сервера бази даних

Для перетворення модуля “Model” у вигляд об’єктно-реляційної моделі використовується бібліотека «SQLAlchemy”

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.

Оновлений код програми:

Новий модуль “alch” використовується для підключення бібліотеки

```
alch.py > ...
1  from sqlalchemy import create_engine
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import sessionmaker
4
5  DATABASE_URL = 'postgresql://postgres:admin@localhost:5432/postgres'
6  engine = create_engine(DATABASE_URL)
7
8  Base = declarative_base()
9
10 Session = sessionmaker()
```

Оновлений model.py в Booking

```
import alch

from sqlalchemy import Column, Integer, String, ForeignKey, DateTime, Time

from sqlalchemy.orm import relationship


class Booking(alch.Base):

    __tablename__ = 'booking'

    booking_id = Column(Integer, primary_key=True)

    user_id = Column(Integer, ForeignKey('users.user_id'), nullable=False)

    facility_id = Column(Integer, ForeignKey('facility.facility_id'), nullable=False)

    booking_date = Column(DateTime, nullable=False)

    start_time = Column(Time, nullable=False)

    end_time = Column(Time, nullable=False)

    status = Column(String, nullable=False)

    user = relationship("User", back_populates="bookings")

    facility = relationship("Facility", back_populates="bookings")

    payments = relationship("Payment", back_populates="booking", cascade="all, delete-orphan")


class ModelBooking:

    def __init__(self, db_model):

        self.conn = db_model.conn

        self.engine = alch.create_engine(alch.DATABASE_URL)

        self.session = alch.Session.configure(bind=self.engine)

        self.session = alch.Session()

    def add_booking(self, booking_id, user_id, facility_id, booking_date, start_time, end_time, status):

        try:
```

```

        user_exists = self.session.query(Booking).filter_by(user_id=user_id).first()

        facility_exists =
self.session.query(Booking).filter_by(facility_id=facility_id).first()

        if not user_exists or not facility_exists:

            print("Error: User ID or Facility ID does not exist.")

            return False

        new_booking = Booking(

            booking_id=booking_id,

            user_id=user_id,

            facility_id=facility_id,

            booking_date=booking_date,

            start_time=start_time,

            end_time=end_time,

            status=status

        )

        self.session.add(new_booking)

        self.session.commit()

        return True

    except Exception as e:

        self.session.rollback()

        print(f"Error With Adding A Booking: {str(e)}")

        return False

def get_all_bookings(self):

    c = self.conn.cursor()

    try:

        c.execute('SELECT * FROM "booking"')

        return c.fetchall()

    except Exception as e:

        print(f"Error With Retrieving Bookings: {str(e)}")

        return None

```

```

    def update_booking(self, booking_id, user_id, facility_id, booking_date, start_time,
end_time, status):

        try:

            user_exists = self.session.query(Booking).filter_by(user_id=user_id).first()

                                                                 facility_exists =
self.session.query(Booking).filter_by(facility_id=facility_id).first()

            if not user_exists or not facility_exists:

                print("Error: User ID or Facility ID does not exist.")

                return False

                booking = self.session.query(Booking).filter(Booking.booking_id ==
booking_id).first()

                if booking:

                    booking.user_id = user_id

                    booking.facility_id = facility_id

                    booking.booking_date = booking_date

                    booking.start_time = start_time

                    booking.end_time = end_time

                    booking.status = status

                    self.session.commit()

                    return True

                return False

        except Exception as e:

            self.session.rollback()

            print(f"Error With Updating A Booking: {str(e)}")

            return False

    def delete_booking(self, booking_id):

        try:

                booking = self.session.query(Booking).filter(Booking.booking_id ==
booking_id).first()

                if booking:

```

```

        self.session.delete(booking)

        self.session.commit()

        return True

    return False

except Exception as e:

    self.session.rollback()

    print(f"Error With Deleting A Booking: {str(e)}")

    return False

```

Оновлений model.py в Facility

```

import alch

from sqlalchemy import Column, Integer, String, ForeignKey

from sqlalchemy.orm import relationship

class Facility(alch.Base):

    __tablename__ = 'facility'

    facility_id = Column(Integer, primary_key=True)

    facility_name = Column(String, nullable=False)

    facility_type = Column(String, nullable=False)

    venue_id = Column(Integer, ForeignKey('venue.venue_id'), nullable=False)

    bookings = relationship("Booking", back_populates="facility")

class ModelFacility:

    def __init__(self, db_model):

        self.conn = db_model.conn

        self.engine = alch.create_engine(alch.DATABASE_URL)

        self.session = alch.Session.configure(bind=self.engine)

```



```

self.session = alch.Session()

def add_facility(self, facility_id, facility_name, facility_type, venue_id):
    """
    Додає новий об'єкт до таблиці Facility.
    """
    try:
        venue_exists = self.session.query(Facility).filter_by(venue_id=venue_id).first()

        if not venue_exists:
            print("Error: Venue ID does not exist.")
            return False

        new_facility = Facility(
            facility_id=facility_id,
            facility_name=facility_name,
            facility_type=facility_type,
            venue_id=venue_id
        )

        self.session.add(new_facility)
        self.session.commit()

        return True

    except Exception as e:
        self.session.rollback()

        print(f"Error With Adding A Facility: {str(e)}")

        return False

def get_all_facilities(self):
    c = self.conn.cursor()

    try:
        c.execute('SELECT * FROM "facility"')

```

```

        return c.fetchall()

    except Exception as e:

        print(f"Error With Retrieving Facilities: {str(e)}")

        return None

def update_facility(self, facility_id, facility_name, facility_type, venue_id):

    """

    Оновлює дані об'єкта Facility за його ID.

    """

    try:

        venue_exists = self.session.query(Facility).filter_by(venue_id=venue_id).first()

        if not venue_exists:

            print("Error: Venue ID does not exist.")

            return False

        facility = self.session.query(Facility).filter(Facility.facility_id == facility_id).first()

        if facility:

            facility.facility_name = facility_name

            facility.facility_type = facility_type

            facility.venue_id = venue_id

            self.session.commit()

            return True

        return False

    except Exception as e:

        self.session.rollback()

        print(f"Error With Updating A Facility: {str(e)}")

        return False

def delete_facility(self, facility_id):

```

```

"""
    Видаляє об'єкт Facility за його ID.
"""

try:

    facility = self.session.query(Facility).filter(Facility.facility_id ==
facility_id).first()

    if facility:

        self.session.delete(facility)

        self.session.commit()

        return True

    return False

except Exception as e:

    self.session.rollback()

    print(f"Error With Deleting A Facility: {str(e)}")

    return False

```

Оновлений model.py в Payment

```

import alch

from sqlalchemy import Column, Integer, Float, DateTime, Boolean, ForeignKey

from sqlalchemy.orm import relationship

class Payment(alch.Base):

    __tablename__ = 'payment'

    payment_id = Column(Integer, primary_key=True)

    booking_id = Column(Integer, ForeignKey('booking.booking_id'), nullable=False)

    amount = Column(Float, nullable=False)

    payment_date = Column(DateTime, nullable=False)

    payment_status = Column(Boolean, nullable=False)

    booking = relationship("Booking", back_populates="payments")

```

```

# Клас для операцій із таблицею Payment

class ModelPayment:

    def __init__(self, db_model):

        self.conn = db_model.conn # Для сумісності зі старою структурою

        self.engine = alch.create_engine(alch.DATABASE_URL)

        self.session = alch.Session.configure(bind=self.engine)

        self.session = alch.Session()

    def add_payment(self, payment_id, booking_id, amount, payment_date, payment_status):

        """

        Додає новий платіж до таблиці.

        """

        try:

            # Перевірка існування booking_id

            booking_exists = self.session.query(Payment).filter_by(booking_id=booking_id).first()

            if not booking_exists:

                print("Error: Booking ID does not exist.")

                return False

            new_payment = Payment(

                payment_id=payment_id,

                booking_id=booking_id,

                amount=amount,

                payment_date=payment_date,

                payment_status=payment_status

            )

            self.session.add(new_payment)

            self.session.commit()

```

```

        return True

    except Exception as e:

        self.session.rollback()

        print(f"Error With Adding A Payment: {str(e)}")

        return False

def get_all_payments(self):

    c = self.conn.cursor()

    try:

        c.execute('SELECT * FROM "payment"')

        return c.fetchall()

    except Exception as e:

        print(f"Error With Retrieving Payments: {str(e)}")

        return None

def update_payment(self, payment_id, booking_id, amount, payment_date, payment_status):

    """

    Оновлює дані платежу за його ID.

    """

    try:

        booking_exists = self.session.query(Payment).filter_by(booking_id=booking_id).first()

        if not booking_exists:

            print("Error: Booking ID does not exist.")

            return False

        payment = self.session.query(Payment).filter(Payment.payment_id == payment_id).first()

        if payment:

            payment.booking_id = booking_id

            payment.amount = amount

```

```

        payment.payment_date = payment_date

        payment.payment_status = payment_status

        self.session.commit()

        return True

    return False

except Exception as e:

    self.session.rollback()

    print(f"Error With Updating A Payment: {str(e)}")

    return False


def delete_payment(self, payment_id):

    """

    Видаляє платіж за його ID.

    """

    try:

        payment = self.session.query(Payment).filter(Payment.payment_id ==
payment_id).first()

        if payment:

            self.session.delete(payment)

            self.session.commit()

            return True

        return False

    except Exception as e:

        self.session.rollback()

        print(f"Error With Deleting A Payment: {str(e)}")

        return False

```

Оновлений model.py в Users

```

import alch

from sqlalchemy import Column, Integer, String, Date

```

```

from sqlalchemy.orm import relationship

class User(alch.Base):

    __tablename__ = 'users'

    user_id = Column(Integer, primary_key=True)

    first_name = Column(String, nullable=False)

    last_name = Column(String, nullable=False)

    email = Column(String, nullable=False)

    phone_number = Column(String, nullable=False)

    date_of_registration = Column(Date, nullable=False)

    bookings = relationship("Booking", back_populates="user")


class ModelUser:

    def __init__(self, db_model):

        self.conn = db_model.conn

        self.engine = alch.create_engine(alch.DATABASE_URL)

        self.session = alch.Session.configure(bind=self.engine)

        self.session = alch.Session()

        def add_user(self, user_id, first_name, last_name, email, phone_number,
date_of_registration):

            """

            Додає нового користувача у таблицю.

            """

            try:

                new_user = User(

                    user_id=user_id,

                    first_name=first_name,

                    last_name=last_name,

```

```

        email=email,

        phone_number=phone_number,

        date_of_registration=date_of_registration

    )

    self.session.add(new_user)

    self.session.commit()

    return True

except Exception as e:

    self.session.rollback()

    print(f"Error adding user: {str(e)}")

    return False


def get_all_users(self):

    c = self.conn.cursor()

    try:

        c.execute('SELECT * FROM "users"')

        return c.fetchall()

    except Exception as e:

        print(f"Error retrieving users: {str(e)}")

        return None


def update_user(self, user_id, **kwargs):

    """

    Оновлює дані користувача за його ID.

    """

    try:

        user = self.session.query(User).filter(User.user_id == user_id).first()

        if user:

            for key, value in kwargs.items():

                setattr(user, key, value)

```



```

        self.session.commit()

        return True

    return False

except Exception as e:

    self.session.rollback()

    print(f"Error updating user: {str(e)}")

    return False


def delete_user(self, user_id):

    """

    Видаляє користувача за його ID.

    """

    try:

        user = self.session.query(User).filter(User.user_id == user_id).first()

        if user:

            self.session.delete(user)

            self.session.commit()

            return True

        return False

    except Exception as e:

        self.session.rollback()

        print(f"Error deleting user: {str(e)}")

        return False

```

Оновлений model.py в Venue

```

import alch

from sqlalchemy import Column, Integer, String

class Venue(alch.Base):

    __tablename__ = 'venue'

```

```

venue_id = Column(Integer, primary_key=True)

name = Column(String(30), nullable=False)

address = Column(String(64), nullable=False)

city = Column(String(25), nullable=False)

capacity = Column(Integer, nullable=False)


class ModelVenue:

    def __init__(self, db_model):

        self.conn = db_model.conn

        self.engine = alch.create_engine(alch.DATABASE_URL)

        self.session = alch.Session.configure(bind=self.engine)

        self.session = alch.Session()


    def add_venue(self, venue_id, name, address, city, capacity):

        try:

            new_venue = Venue(

                venue_id=venue_id,

                name=name,

                address=address,

                city=city,

                capacity=capacity

            )

            self.session.add(new_venue)

            self.session.commit()

            return True

        except Exception as e:

            self.session.rollback()

            print(f"Error Adding Venue: {str(e)}")

            return False

```

```

def get_all_venues(self):

    c = self.conn.cursor()

    try:

        c.execute('SELECT * FROM "venue"')

        return c.fetchall()

    except Exception as e:

        print(f"Error Retrieving Venues: {str(e)}")

        return None


def update_venue(self, venue_id, **kwargs):

    try:

        venue = self.session.query(Venue).filter(Venue.venue_id == venue_id).first()

        if venue:

            for key, value in kwargs.items():

                setattr(venue, key, value)

            self.session.commit()

            return True

        return False

    except Exception as e:

        self.session.rollback()

        print(f"Error Updating Venue: {str(e)}")

        return False


def delete_venue(self, venue_id):

    try:

        venue = self.session.query(Venue).filter(Venue.venue_id == venue_id).first()

        if venue:

            self.session.delete(venue)

            self.session.commit()

```

```

        return True

    return False

except Exception as e:

    self.session.rollback()

    print(f"Error Deleting Venue: {str(e)}")

    return False

```

Бачимо, що програма працює так само

```

Main Menu:
1. Add New Booking
2. Add New Facility
3. Add New Payment
4. Add New User
5. Add New Venue
6. Show Bookings
7. Show Facilities
8. Show Payments
9. Show Users
10. Show Venues
11. Update Booking
12. Update Facility
13. Update Payment
14. Update User
15. Update Venue
16. Remove Booking
17. Remove Facility
18. Remove Payment
19. Remove User
20. Remove Venue
21. Create Data By Random
22. Delete All Data
23. View Analytics
24. Exit
Choose an action : 9
Users:
ID: 1, First name: John, Last name: Smith, Email: john.smith@gmail.com, Phone number: 380738444431, Date of registration: 2024-03-12
ID: 2, First name: Emma, Last name: Johnes, Email: emma.johnes@gmail.com, Phone number: 380984043221, Date of registration: 2024-05-13
ID: 3, First name: John, Last name: Taylor, Email: john.taylor@gmail.com, Phone number: 380548242483, Date of registration: 2024-02-09
ID: 4, First name: Stan, Last name: Fire, Email: stan.fire@gmail.com, Phone number: 380372170323, Date of registration: 2024-08-07
ID: 5, First name: Ava, Last name: Wilson, Email: ava.wilson@gmail.com, Phone number: 380842453551, Date of registration: 2024-04-16
ID: 6, First name: Tom, Last name: Smith, Email: tom.smith@gmail.com, Phone number: 380158681611, Date of registration: 2024-01-13
ID: 7, First name: Michael, Last name: Johnes, Email: michael.johnes@gmail.com, Phone number: 380421228610, Date of registration: 2024-09-14
ID: 8, First name: Stan, Last name: Evans, Email: stan.evans@gmail.com, Phone number: 380882999152, Date of registration: 2024-03-22
ID: 9, First name: Emma, Last name: Brown, Email: emma.brown@gmail.com, Phone number: 380869372059, Date of registration: 2024-05-22
ID: 10, First name: Michael, Last name: Tesla, Email: michael.tesla@gmail.com, Phone number: 380101274963, Date of registration: 2024-08-25
ID: 11, First name: Alex, Last name: Tesla, Email: alex.tesla@gmail.com, Phone number: 380841383866, Date of registration: 2024-07-25

```

```

Choose an action : 19
Input User ID: 11
Successfully Deleted A User

```

```

Choose an action : 9
Users:
ID: 1, First name: John, Last name: Smith, Email: john.smith@gmail.com, Phone number: 380738444431, Date of registration: 2024-03-12
ID: 2, First name: Emma, Last name: Johnes, Email: emma.johnes@gmail.com, Phone number: 380984043221, Date of registration: 2024-05-13
ID: 3, First name: John, Last name: Taylor, Email: john.taylor@gmail.com, Phone number: 380548242483, Date of registration: 2024-02-09
ID: 4, First name: Stan, Last name: Fire, Email: stan.fire@gmail.com, Phone number: 380372170323, Date of registration: 2024-08-07
ID: 5, First name: Ava, Last name: Wilson, Email: ava.wilson@gmail.com, Phone number: 380842453551, Date of registration: 2024-04-16
ID: 6, First name: Tom, Last name: Smith, Email: tom.smith@gmail.com, Phone number: 380158681611, Date of registration: 2024-01-13
ID: 7, First name: Michael, Last name: Johnes, Email: michael.johnes@gmail.com, Phone number: 380421228610, Date of registration: 2024-09-14
ID: 8, First name: Stan, Last name: Evans, Email: stan.evans@gmail.com, Phone number: 380882999152, Date of registration: 2024-03-22
ID: 9, First name: Emma, Last name: Brown, Email: emma.brown@gmail.com, Phone number: 380869372059, Date of registration: 2024-05-22
ID: 10, First name: Michael, Last name: Tesla, Email: michael.tesla@gmail.com, Phone number: 380101274963, Date of registration: 2024-08-25

```

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).

Варіант:

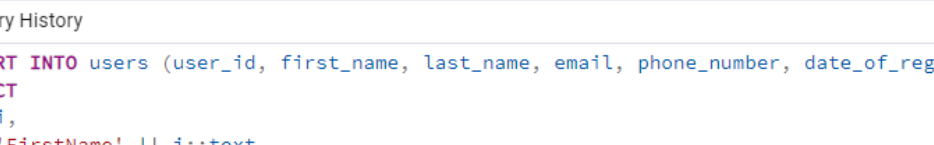
8	<i>BTree, GIN</i>	<i>after insert, update</i>
---	-------------------	-----------------------------

Index B-tree

Створюємо індекс

The screenshot shows a database query editor interface. At the top, there is a toolbar with various icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, displaying a SQL statement: `1 CREATE INDEX idx_users_email_btree ON users USING btree (email);`. The 'Messages' tab is also visible, showing the output: `CREATE INDEX` and `Query returned successfully in 48 msec.`

Генеруємо тестові дані



The screenshot shows the SQL Developer interface. At the top is a toolbar with icons for file operations, query execution, and help. Below the toolbar, the 'Query' tab is active, displaying a SQL script. The script is a multi-line INSERT statement that generates 100,000 rows. The script is as follows:

```
1 INSERT INTO users (user_id, first_name, last_name, email, phone_number, date_of_registration)
2 SELECT
3     i,
4     'FirstName' || i::text,
5     'LastName' || i::text,
6     'email' || i::text || '@example.com',
7     '1234567890',
8     CURRENT_DATE - (i % 100) * INTERVAL '1 day'
9 FROM generate_series(1, 100000) AS i;
```

Below the query editor, the 'Data Output' tab is active, showing the result of the query: 'INSERT 0 99990'. A status message at the bottom indicates 'Query returned successfully in 1 secs 15 msec.'

Робимо запит без індексу

Query Query History

```

1 SET enable_seqscan = ON;
2
3 EXPLAIN ANALYZE
4 SELECT * FROM users WHERE email = 'email50000@example.com';|

```

Data Output Messages Notifications

	QUERY PLAN	
	text	🔒
1	Index Scan using idx_users_email_btree on users (cost=0.42..8.44 rows=1 width=68) (actual time=0.018..0.018 rows=1 loop...)	
2	Index Cond: (email = 'email50000@example.com')::text)	
3	Planning Time: 1.181 ms	
4	Execution Time: 0.030 ms	

Робимо запит з індексом

The screenshot shows a database query interface with a toolbar at the top containing icons for file operations, filters, and execution. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

```
1 SET enable_seqscan = OFF;  
2  
3 EXPLAIN ANALYZE  
4 SELECT * FROM users WHERE email = 'email50000@example.com';
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the query plan results:

	QUERY PLAN	
1	Index Scan using idx_users_email_btree on users (cost=0.42..8.44 rows=1 width=68) (actual time=0.022..0.023 rows=1 loop...	
2	Index Cond: (email = 'email50000@example.com'::text)	
3	Planning Time: 0.056 ms	
4	Execution Time: 0.033 ms	

Index GIN

Створюємо індекс

The screenshot shows a database query interface with a toolbar at the top. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

```
1 CREATE INDEX idx_users_name_gin ON users USING gin (to_tsvector('english', first_name || ' ' || last_name));
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the following message:

```
CREATE INDEX  
  
Query returned successfully in 643 msec.
```

Робимо запит без індексу

The screenshot shows a database query tool interface. The query is as follows:

```
1 SET enable_seqscan = ON;  
2  
3 EXPLAIN ANALYZE  
4 SELECT * FROM users WHERE to_tsvector('english', first_name || ' ' || last_name) @@ to_tsquery('FirstName50000 & LastName50000');
```

The "Data Output" tab is selected, showing the "QUERY PLAN" for the query. The plan details are as follows:

Step	Operation
1	Bitmap Heap Scan on users (cost=30.32..39.12 rows=2 width=68) (actual time=0.040..0.040 rows=1 loops=1)
2	Recheck Cond: (to_tsvector('english':regconfig, ((first_name ' ':text) last_name)) @@ to_tsquery('FirstName50000 & LastName50000':t...
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on idx_users_name_gin (cost=0.00..30.32 rows=2 width=0) (actual time=0.027..0.027 rows=1 loops=1)
5	Index Cond: (to_tsvector('english':regconfig, ((first_name ' ':text) last_name)) @@ to_tsquery('FirstName50000 & LastName50000':t...
6	Planning Time: 1.375 ms
7	Execution Time: 0.064 ms

Робимо запит з індексом

The screenshot shows the same database query tool interface, but with the query modified to disable sequential scans:

```
1 SET enable_seqscan = OFF;  
2  
3 EXPLAIN ANALYZE  
4 SELECT * FROM users WHERE to_tsvector('english', first_name || ' ' || last_name) @@ to_tsquery('FirstName50000 & LastName50000');
```

The "Data Output" tab is selected, showing the "QUERY PLAN" for the query. The plan details are as follows:

Step	Operation
1	Bitmap Heap Scan on users (cost=30.32..39.12 rows=2 width=68) (actual time=0.019..0.019 rows=1 loops=1)
2	Recheck Cond: (to_tsvector('english':regconfig, ((first_name ' ':text) last_name)) @@ to_tsquery('FirstName50000 & LastName50000':t...
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on idx_users_name_gin (cost=0.00..30.32 rows=2 width=0) (actual time=0.013..0.013 rows=1 loops=1)
5	Index Cond: (to_tsvector('english':regconfig, ((first_name ' ':text) last_name)) @@ to_tsquery('FirstName50000 & LastName50000':t...
6	Planning Time: 0.083 ms
7	Execution Time: 0.030 ms

Тригери:

Логування after insert

Query	Query History
-------	---------------

```
1 CREATE TABLE users_log (  
2     log_id SERIAL PRIMARY KEY,  
3     user_id INT,  
4     action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
5     action VARCHAR(50)  
6 );  
7  
8 CREATE OR REPLACE FUNCTION log_user_insert()  
9 RETURNS TRIGGER AS $$  
10 BEGIN  
11     INSERT INTO users_log (user_id, action)  
12     VALUES (NEW.user_id, 'INSERT');  
13     RETURN NEW;  
14 END;  
15 $$ LANGUAGE plpgsql;  
16  
17 CREATE TRIGGER after_user_insert  
18 AFTER INSERT ON users  
19 FOR EACH ROW  
20 EXECUTE FUNCTION log_user_insert();
```

Data Output	Messages	Notifications
-------------	----------	---------------

CREATE TRIGGER

Query returned successfully in 91 msec.

Логування after update

Query	Query History
-------	---------------

```
1 CREATE OR REPLACE FUNCTION log_user_update()  
2 RETURNS TRIGGER AS $$  
3 BEGIN  
4     INSERT INTO users_log (user_id, action)  
5     VALUES (NEW.user_id, 'UPDATE');  
6     RETURN NEW;  
7 END;  
8 $$ LANGUAGE plpgsql;  
9  
10 CREATE TRIGGER after_user_update  
11 AFTER UPDATE ON users  
12 FOR EACH ROW  
13 EXECUTE FUNCTION log_user_update();
```

Data Output	Messages	Notifications
-------------	----------	---------------

CREATE TRIGGER

Query returned successfully in 49 msec.

Перевірка логування після вставки

QueryQuery History

1

▼

INSERT INTO users (user_id, first_name, last_name, email, phone_number, date_of_registration)

2

VALUES (100001, 'Test', 'User', 'testuser@example.com', '9876543210', CURRENT_DATE);

3

4

SELECT * FROM users_log;

Data OutputMessagesNotifications

≡+

📄

▼

📋

▼

🗑

🔍

⬇

📈

SQL

	log_id [PK] integer	user_id integer	action_time timestamp without time zone	action character varying (50)
1	1	100001	2025-01-04 19:09:50.720004	INSERT

Перевірка логування після оновлення

QueryQuery History

1

▼

UPDATE users

2

SET email = 'updatedemail@example.com'

3

WHERE user_id = 100001;

4

5

SELECT * FROM users_log;|

Data OutputMessagesNotifications

≡+

📄

▼

📋

▼

🗑

🔍

⬇

📈

SQL

	log_id [PK] integer	user_id integer	action_time timestamp without time zone	action character varying (50)
1	1	100001	2025-01-04 19:09:50.720004	INSERT
2	2	100001	2025-01-04 19:11:22.598519	UPDATE