

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 Баз даних і засоби управління
на тему:
“Засоби оптимізації роботи СУБД PostgreSQL”

Виконав:
студент 3 курсу
групи КВ-03
Стецюренко І. С.
Перевірив:
Петрашенко А. В.

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.

Варіант №22

<i>№ варіанта</i>	<i>Види індексів</i>	<i>Умови для тригера</i>
22	Hash, BRIN	after delete, insert

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.

Корисні посилання: [тут](#) і [тут](#).

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).

Корисні посилання: [Hash](#), [B-tree](#), [GIN](#), [BRIN](#).

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.

Корисні посилання: [тут](#), [тут](#).

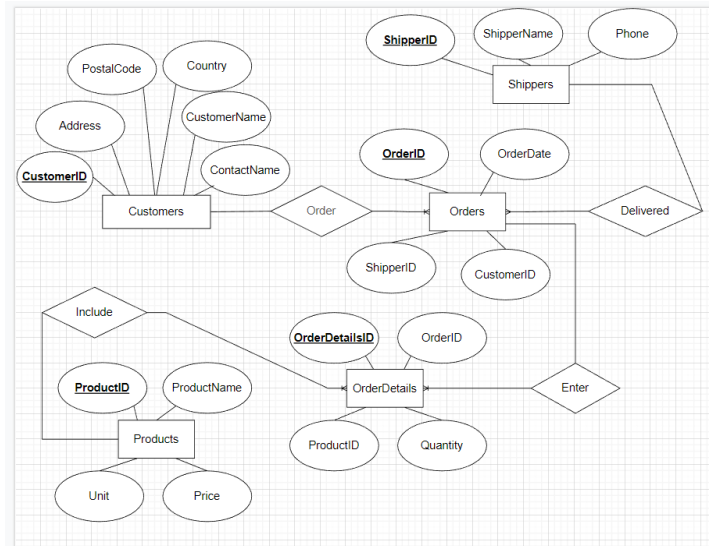
Вимоги до пункту завдання №4

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.

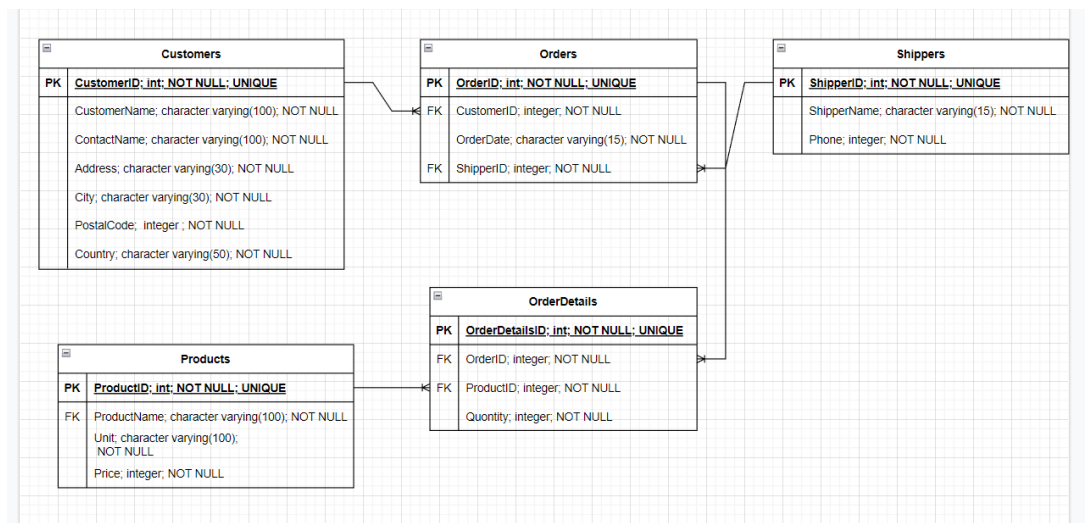
[URL репозиторію з вихідним кодом](#)

Завдання 1

Модель “сутність - зв’язок” галузі “доставка продуктів”



мал. 1 ER - діаграма, побудована за нотацією Чена



мал. 2 - ER- діаграма, переведена у таблиці БД

Зміни у порівнянні з першою лабораторною роботою відсутні.

Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась програмування Python та середовище розробки PyCharm 2021.2.1.

Для підключення до серверу бази даних PostgreSQL використано сторонню бібліотеку psycopg2, для реалізації моделі ORM

використовувалася стороння бібліотека SQLAlchemy, середовище для відлагодження SQL-запитів до бази даних - pgAdmin 4.

Класи ORM

У даній лабораторній роботі було реалізовано 5 класів відповідно до 5 існуючих таблиць: Customers, Orders, OrderDetails, Shippers, Products.

Таблиця Customers має стовпчики CustomerId, CustomerName, ContactName, Address, City, PostalCode, Country, а також зв'язок 1:N із таблицею Orders, тому в класі Customers встановлений зв'язок relationship(Orders).

Таблиця Orders має стовпчики OrderID, OrderData, ShipperID, CustomerID, а також зв'язок 1:N із таблицею Shippers, тому в класі Orders встановлений зв'язок relationship(Shippers). Ця таблиця має зв'язок 1:N з таблицею OrderDetails, тому в класі Orders встановлений зв'язок relationship("OrderDetails"). Ця таблиця має зв'язок 1:N з таблицею Shippers, тому в класі Orders встановлений зв'язок relationship("Shippers").

Таблиця OrderDetails має стовпчики OrderDetailID, OrderID, ProductID, Quantity, а також зв'язок 1:N із таблицею Orders, тому в класі OrderDetails встановлений зв'язок relationship(Orders). Ця таблиця має зв'язок 1:N з таблицею Products, тому в класі OrderDetails встановлений зв'язок relationship("Products").

Таблиця Products має стовпчики ProductID, ProductName, Price, Unit, а також зв'язок 1:N із таблицею OrderDetails, тому в класі Products встановлений зв'язок relationship(OrderDetails).

Таблиця Shippers має стовпчики ShipperID, ShipperName, Phone, а також зв'язок 1:N із таблицею Orders, тому в класі Shippers встановлений зв'язок relationship(Orders).

Нижче наведена програмна реалізація класів ORM мовою Python (лістинги усіх модулів надані нижче):

```
class Customers(Base):
    __tablename__ = "Customers"

    CustomerID = Column("CustomerID", Integer, primary_key=True, nullable=False)
    CustomerName = Column("CustomerName", String, nullable=False)
```

```
ContactName = Column("ContactName", String, nullable=False)
Address = Column("Address", String, nullable=False)
City = Column("City", String, nullable=False)
PostalCode = Column("PostalCode", Integer, nullable=False)
Country = Column("Country", String, nullable=False)
```

```
orders = relationship('Orders')
```

```
def __init__(self, customerid, customername, contactname, address, city, postalcode, country):
    self.CustomerID = customerid
    self.CustomerName = customername
    self.ContactName = contactname
    self.Address = address
    self.City = city
    self.PostalCode = postalcode
    self.Country = country

def __repr__(self):
    return f"{self.CustomerID} {self.CustomerName} {self.ContactName} {self.Address} {self.City} {self.PostalCode} {self.Country}"
```

```
class OrderDetails(Base):
```

```
    __tablename__ = "OrderDetails"
```

```
OrderDetailID = Column("OrderDetailID", Integer, primary_key=True, nullable=False)
OrderID = Column("OrderID", Integer, ForeignKey('Orders.OrderID'), nullable=False)
ProductID = Column("ProductID", Integer, ForeignKey('Products.ProductID'), nullable=False)
Quantity = Column("Quantity", Integer, nullable=False)
```

```
products = relationship('Products')
```

```
orders = relationship('Orders')
```

```
def __init__(self, orderdetailid, orderid, productid, quantity):
```

```
    self.OrderDetailID = orderdetailid
    self.OrderID = orderid
    self.ProductID = productid
    self.Quantity = quantity
```

```
def __repr__(self):
```

```
    return f"{self.OrderDetailID} {self.OrderID} {self.ProductID} {self.Quantity} "
```

```
class Orders(Base):
```

```
    __tablename__ = "Orders"
```

```
OrderID = Column("OrderID", Integer, primary_key=True, nullable=False)
CustomerID = Column("CustomerID", Integer, ForeignKey('Customers.CustomerID'), nullable=False)
OrderDate = Column("OrderDate", String, nullable=False)
ShipperID = Column("ShipperID", Integer, ForeignKey('Shippers.ShipperID'), nullable=False)
```

```
customers = relationship('Customers')
```

```
order_details = relationship('OrderDetails')
```

```
shippers = relationship('Shippers')
```

```

def __init__(self, orderid, customerid, orderdate, shipperid):
    self.OrderID = orderid
    self.CustomerID = customerid
    self.OrderDate = orderdate
    self.ShipperID = shipperid

def __repr__(self):
    return f"{self.OrderID} {self.CustomerID} {self.OrderDate} {self.ShipperID} "

class Products(Base):
    __tablename__ = "Products"

    ProductID = Column("ProductID", Integer, primary_key=True, nullable=False)
    ProductName = Column("ProductName", String, nullable=False)
    Unit = Column("Unit", String, nullable=False)
    Price = Column("Price", Numeric, nullable=False)

    order_details = relationship("OrderDetails")
    orders = relationship("Orders")

    def __init__(self, productid, productname, unit, price):
        self.ProductID = productid
        self.ProductName = productname
        self.Unit = unit
        self.Price = price

    def __repr__(self):
        return f"{self.ProductID} {self.ProductName} {self.Unit} {self.Price} "

class Shippers(Base):
    __tablename__ = "Shippers"

    ShipperID = Column("ShipperID", Integer, primary_key=True, nullable=False)
    ShipperName = Column("ShipperName", String, nullable=False)
    Phone = Column("Phone", String, nullable=False)

    def __init__(self, shipperid, shippername, phone):
        self.ShipperID = shipperid
        self.ShipperName = shippername
        self.Phone = phone

    def __repr__(self):
        return f"{self.ShipperID} {self.ShipperName} {self.Phone} "

```

Приклад запитів у вигляді ORM

Запит вставки

Цей запит реалізовано за допомогою функції insert(для кожної таблиці своя функція). Спочатку у меню користувач обирає опцію вставки,

далі обирає таблицю, до якої хоче додати запис і вводить необхідні дані. Є перевірка введених даних. У разі успішного додавання запису користувач бачить відповідне повідомлення. Реалізацію запиту вставки продемонструємо на прикладі таблиці Customers .

```
"Welcome!

Main menu
1. Show one table
2. Show all table
3. Insert data
4. Delete data
5. Update date
6. Exit

Make your choice => 3
"

1. Customers
2. OrderDetails
3. Orders
4. Products
5. Shippers

Make your choice => 1
CustomerID = 9
Incorrect number entered, please enter again => 6
CustomerName = Illia
ContactName = Den
Address = Soborna Street 76
City = Novoukrainka
PostalCode = 27100
Country = Ukraine
Continue work with DB? Y/N =>
```


Query

Query History

1

SELECT * FROM "Customers"

Data output

Messages

Notifications

	CustomerID [PK] integer	CustomerName character varying (100)	ContactName character varying (100)	Address character varying (30)	City character varying (30)	PostalCode integer	Country character varying (50)
1	1	Alfreds Futterkiste	Maria Anders	Obere Street 57	Berlin	12209	Germany
2	2	Du monde entier	Janine Labrune	67, rue des Cinquante Otag...	Nantes	44000	France
3	3	Franchi S.p.A.	Paolo Accorti	Via Monte Bianco 34	Torino	10100	Italy
4	5	Dmytro Lyashuk	Pavlenko Roman	Korsunska Street 16	Kyiv	79022	Ukraine
5	6	Illia	Den	Soborna Street 76	Novoukrainka	27100	Ukraine
6	4	Poshchyalov Oleh	Larin Dmytro	Soborna Street 77a	Kyiv	8131	Ukraine

Лістинг функцій Insert

```
def insert_customers():
    Session = sessionmaker(bind=engine)
    session = Session()
    print('CustomerID =', end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id != (session.query(Customers).count() + 1)):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('CustomerName =', end=' ')
    name = input()
    print('ContactName =', end=' ')
    cname = input()
    print('Address =', end=' ')
    address = input()
    print('City =', end=' ')
    city = input()
    print('PostalCode =', end=' ')
    code = input()
    print('Country =', end=' ')
    country = input()

    session.add(Customers(id, name, cname, address, city, code, country))
    session.commit()
```

```
def insert_order_details():
    Session = sessionmaker(bind=engine)
    session = Session()

    print('OrderDetailID =', end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id != (session.query(OrderDetails).count() + 1)):
```

```

        print('Incorrect number entered, please enter again =>', end=' ')
    else:
        break
    print('OrderID =', end=' ')
    while 1:
        o_id = input()
        o_id = int(o_id)
        if (o_id > session.query(Orders).count()):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('ProductID =', end=' ')
    while 1:
        p_id = input()
        p_id = int(p_id)
        if (p_id > session.query(Products).count()):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('Quantity =', end=' ')
    quantity = input()

    session.add(OrderDetails(id, o_id, p_id, quantity))
    session.commit()

```

```

def insert_orders():
    Session = sessionmaker(bind=engine)
    session = Session()

    print('OrderID =', end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id != (session.query(Orders).count() + 1)):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('CustomerID =', end=' ')
    while 1:
        c_id = input()
        c_id = int(c_id)
        if (c_id > session.query(Customers).count()):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('OrderDate =', end=' ')
    date = input()
    print('ShipperID =', end=' ')
    while 1:
        s_id = input()
        s_id = int(s_id)
        if (s_id > session.query(Shippers).count()):
            print('Incorrect number entered, please enter again =>', end=' ')

```

```

        else:
            break

session.add(Orders(id, c_id, date, s_id))
session.commit()

def insert_products():
    Session = sessionmaker(bind=engine)
    session = Session()

    print('ProductID =', end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id != (session.query(Products).count() + 1)):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('ProductName =', end=' ')
    name = input()
    print('Unit =', end=' ')
    unit = input()
    print('Price =', end=' ')
    price = input()

    session.add(Products(id, name, unit, price))
    session.commit()

def insert_shippers():
    Session = sessionmaker(bind=engine)
    session = Session()

    print('Shippers =', end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id != (session.query(Shippers).count() + 1)):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    print('ShipperName =', end=' ')
    name = input()
    print('Phone =', end=' ')
    phone = input()

    session.add(Shippers(id, name, phone))
    session.commit()

```

Запит редагування

Цей запит реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Також потрібно обрати атрибут, що редагується. У разі успішного редагування користувач побачить відповідне повідомлення. Редагування запиту продемонструємо на прикладі таблиці Customers .

```
Continue work with DB? Y/N => y
''
Main menu
1. Show one table
2. Show all table
3. Insert data
4. Delete data
5. Update date
6. Exit

Make your choice => 5

1. Customers
2. OrderDetails
3. Orders
4. Products
5. Shippers

Make your choice => 1
Enter the number CustomerID, which you want to change => 6

1. CustomerName
2. ContactName
3. Address
4. City
5. PostalCode
6. Country

Make your choice => 4
New value of City => Dnipro
Continue work with DB? Y/N => |
```

1 **SELECT** * **FROM** "Customers"

Data output

Messages

Notifications

	CustomerID [PK] integer	CustomerName character varying (100)	ContactName character varying (100)	Address character varying (30)	City character varying (30)	PostalCode integer	Country character varyi
1	1	Alfreds Futterkiste	Maria Anders	Obere Street 57	Berlin	12209	Germany
2	2	Du monde entier	Janine Labrune	67, rue des Cinquante Otag...	Nantes	44000	France
3	3	Franchi S.p.A.	Paolo Accorti	Via Monte Bianco 34	Torino	10100	Italy
4	5	Dmytro Lyashuk	Pavlenko Roman	Korsunska Street 16	Kyiv	79022	Ukraine
5	6	Illia	Den	Soborna Street 76	Dnipro	27100	Ukraine
6	4	Poshchyvalov Oleh	Larin Dmytro	Soborna Street 77a	Kyiv	8131	Ukraine

Лістинг функцій Update

def update():

 Session = sessionmaker(bind=engine)

 session = Session()

 tables = ['Customers', 'OrderDetails', 'Orders', 'Products', 'Shippers']

 width = shutil.get_terminal_size().columns

 position = (width - max(map(len, tables))) // 2

print(" ")

for line **in** range(5):

print(' ' * position + f'{line + 1}. ' + tables[line])

print("Make your choice =>", end=' ')

while 1:

 number = **input**()

 number = **int**(number)

if 0 > number > 5:

print('Incorrect number entered, please enter again =>', end=' ')

else:

break

if number == 1:

print("Enter the number CustomerID, which you want to change =>", end=' ')

while 1:

 id = **input**()

 id = **int**(id)

if (id > session.query(Customers).count()):

print('Incorrect number entered, please enter again =>', end=' ')

else:

break

 lines = ['CustomerName', 'ContactName', 'Address', 'City', 'PostalCode', 'Country']

 position = (width - max(map(len, lines))) // 2

print(" ")

for line **in** range(6):

print(' ' * position + f'{line + 1}. ' + lines[line])

print("Make your choice =>", end=' ')

while 1:

 number = **input**()

 number = **int**(number)

if 0 > number > 6:

print('Incorrect number entered, please enter again =>', end=' ')

else:

```

        break
    if number == 1:
        print("New value of CustomerName =>", end=' ')
        name = input()
        i = session.query(Customers).get(id)
        i.CustomerName = name
    elif number == 2:
        print("New value of ContactName =>", end=' ')
        name = input()
        i = session.query(Customers).get(id)
        i.ContactName = name
    elif number == 3:
        print("New value of Address =>", end=' ')
        address = input()
        i = session.query(Customers).get(id)
        i.Address = address
    elif number == 4:
        print("New value of City =>", end=' ')
        city = input()
        i = session.query(Customers).get(id)
        i.City = city
    elif number == 5:
        print("New value of PostalCode =>", end=' ')
        code = input()
        i = session.query(Customers).get(id)
        i.PostalCode = code
    elif number == 6:
        print("New value of Country =>", end=' ')
        country = input()
        i = session.query(Customers).get(id)
        i.Country = country
    session.add(i)
    session.commit()
elif number == 2:
    print("Enter the number OrderDetailID, which you want to change =>", end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id > session.query(OrderDetails).count()):
            print("Incorrect number entered, please enter again =>", end=' ')
        else:
            break
    lines = ['OrderID', 'ProductID', 'Quantity']
    position = (width - max(map(len, lines))) // 2
    print(" ")
    for line in range(3):
        print(' ' * position + f'{line + 1}. ' + lines[line])
    print("Make your choice =>", end=' ')
    while 1:
        number = input()
        number = int(number)
        if 0 > number > 3:
            print("Incorrect number entered, please enter again =>", end=' ')
        else:
            break

```

```

if number == 1:
    print("New value of OrderID =>", end=' ')
    while 1:
        o_id = input()
        o_id = int(o_id)
        if (o_id > session.query(Orders).count()):
            print("Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    i = session.query(OrderDetails).get(id)
    i.OrderID = o_id
elif number == 2:
    print("New value of ProductID =>", end=' ')
    while 1:
        p_id = input()
        p_id = int(p_id)
        if (id > session.query(Products).count()):
            print("Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    i = session.query(OrderDetails).get(id)
    i.ProductID = p_id
elif number == 3:
    print("New value of Quantity =>", end=' ')
    quantity = input()
    i = session.query(OrderDetails).get(id)
    i.Quantity = quantity
session.add(i)
session.commit()
elif number == 3:
    print("Enter the number OrderID, which you want to change =>", end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id > session.query(OrderDetails).count()):
            print("Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    lines = ['CustomerID', 'OrderDate', 'ShipperID']
    position = (width - max(map(len, lines))) // 2
    print(" ")
    for line in range(3):
        print(' ' * position + f'{line + 1}.' + lines[line])
    print("Make your choice =>", end=' ')
    while 1:
        number = input()
        number = int(number)
        if 0 > number > 3:
            print("Incorrect number entered, please enter again =>', end=' ')
        else:
            break
if number == 1:
    print("New value of CustomerID =>", end=' ')
    while 1:
        c_id = input()

```

```

        c_id = int(c_id)
        if (c_id > session.query(Customers).count()):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
        i = session.query(Orders).get(id)
        i.CustomerID = c_id
    elif number == 2:
        print("New value of OrderDate =>", end=' ')
        date = input()
        i = session.query(Orders).get(id)
        i.OrderDate = date
    elif number == 3:
        print("New value of ShipperID =>", end=' ')
        while 1:
            s_id = input()
            s_id = int(s_id)
            if (s_id > session.query(Products).count()):
                print('Incorrect number entered, please enter again =>', end=' ')
            else:
                break
        i = session.query(Orders).get(id)
        i.ShipperID = s_id
    session.add(i)
    session.commit()
elif number == 4:
    print("Enter the number ProductID, which you want to change =>", end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id > session.query(Products).count()):
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    lines = ['ProductName', 'Unit', 'Price']
    position = (width - max(map(len, lines))) // 2
    print(" ")
    for line in range(3):
        print(' ' * position + f'{line + 1}. ' + lines[line])
    print("Make your choice =>", end=' ')
    while 1:
        number = input()
        number = int(number)
        if 0 > number > 3:
            print('Incorrect number entered, please enter again =>', end=' ')
        else:
            break
    if number == 1:
        print("New value of ProductName =>", end=' ')
        name = input()
        i = session.query(Products).get(id)
        i.ProductName = name
    elif number == 2:
        print("New value of Unit =>", end=' ')
        unit = input()

```



```

        i = session.query(Products).get(id)
        i.Unit = unit
    elif number == 3:
        print("New value of Price =>", end=' ')
        price = input()
        i = session.query(Products).get(id)
        i.Price = price
    session.add(i)
    session.commit()
elif number == 5:
    print("Enter the number OrderDetailID, which you want to change =>", end=' ')
    while 1:
        id = input()
        id = int(id)
        if (id > session.query(Shippers).count()):
            print("Incorrect number entered, please enter again =>", end=' ')
        else:
            break
    lines = ['ShipperName', 'Phone']
    position = (width - max(map(len, lines))) // 2
    print(" ")
    for line in range(2):
        print(' ' * position + f'{line + 1}. ' + lines[line])
    print("Make your choice =>", end=' ')
    while 1:
        number = input()
        number = int(number)
        if 0 > number > 2:
            print("Incorrect number entered, please enter again =>", end=' ')
        else:
            break
    if number == 1:
        print("New value of ShipperName =>", end=' ')
        name = input()
        i = session.query(Shippers).get(id)
        i.ShipperName = name
    elif number == 2:
        print("New value of Phone =>", end=' ')
        phone = input()
        i = session.query(Shippers).get(id)
        i.Phone = phone
    session.add(i)
    session.commit()

```

Запит видалення

Цей запит реалізовано за допомогою функції delete. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора запису для видалення. Якщо такого ідентифікатора не існує, то користувач побачить повідомлення про помилку. У разі успішного видалення запису користувач побачить відповідне повідомлення

Реалізацію запиту видалення продемонструємо на таблиці Customers.

```

Main menu
1. Show one table
2. Show all table
3. Insert data
4. Delete data
5. Update date
6. Exit

Make your choice => 4

1. Customers
2. OrderDetails
3. Orders
4. Products
5. Shippers

Make your choice => 1
Enter the number CustomerID, which you want to delete => 4

```

Query

Query History

1

SELECT * FROM "Customers"

Data output

Messages

Notifications

	CustomerID [PK] integer	CustomerName character varying (100)	ContactName character varying (100)	Address character varying (30)	City character varying (30)	PostalCode integer	Country character varying (50)
1	1	Alfreds Futterkiste	Maria Anders	Obere Street 57	Berlin	12209	Germany
2	2	Du monde entier	Janine Labrune	67, rue des Cinquante Otag...	Nantes	44000	France
3	3	Franchi S.p.A.	Paolo Accorti	Via Monte Bianco 34	Torino	10100	Italy
4	5	Dmytro Lyashuk	Pavlenko Roman	Korsunska Street 16	Kyiv	79022	Ukraine
5	4	Poshchyvalov Oleh	Larin Dmytro	Soborna Street 77a	Kyiv	8131	Ukraine

Лістинг функцій Delete

```

def delete():
    Session = sessionmaker(bind=engine)
    session = Session()

    tables = ['Customers', 'OrderDetails', 'Orders', 'Products', 'Shippers']
    width = shutil.get_terminal_size().columns
    position = (width - max(map(len, tables))) // 2

    print(" ")
    for line in range(5):
        print(' ' * position + f'{line + 1}. ' + tables[line])
    print("Make your choice =>", end=' ')
    while 1:
        number = input()
        number = int(number)
        if 0 > number > 5:
            print('Incorrect number entered, please enter again =>', end=' ')
        else:

```

```

        break
    if number == 1:
        print("Enter the number CustomerID, which you want to delete =>", end=' ')
        while 1:
            id = input()
            id = int(id)
            if (id > session.query(Customers).count()):
                print("Incorrect number entered, please enter again =>", end=' ')
            else:
                break
        session.delete(session.query(Customers).filter(Customers.CustomerID == id).one())
    elif number == 2:
        print("Enter the number OrderDetailsID, which you want to delete =>", end=' ')
        while 1:
            id = input()
            id = int(id)
            if (id > session.query(Customers).count()):
                print("Incorrect number entered, please enter again =>", end=' ')
            else:
                break
        session.delete(session.query(OrderDetails).filter(OrderDetails.OrderDetailID == id).one())
    elif number == 3:
        print("Enter the number OrderID, which you want to delete =>", end=' ')
        while 1:
            id = input()
            id = int(id)
            if (id > session.query(Customers).count()):
                print("Incorrect number entered, please enter again =>", end=' ')
            else:
                break
        session.delete(session.query(Orders).filter(Orders.OrderID == id).one())
    elif number == 4:
        print("Enter the number ProductID, which you want to delete =>", end=' ')
        while 1:
            id = input()
            id = int(id)
            if (id > session.query(Customers).count()):
                print("Incorrect number entered, please enter again =>", end=' ')
            else:
                break
        session.delete(session.query(Products).filter(Products.ProductID == id).one())
    elif number == 5:
        print("Enter the number ShipperID, which you want to delete =>", end=' ')
        while 1:
            id = input()
            id = int(id)
            if (id > session.query(Customers).count()):
                print("Incorrect number entered, please enter again =>", end=' ')
            else:
                break
        session.delete(session.query(Shippers).filter(Shippers.ShipperID == id).one())
    session.commit()

```

Завдання 2

Для тестування використовуються 100000 рандомних даних в окремій таблиці, згенерованим запитом з другої лабораторної роботи.

```
CREATE TABLE IF NOT EXISTS public.test
(
    string text COLLATE pg_catalog."default" NOT NULL,
    "number" integer NOT NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.test
    OWNER to postgres;
```

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

1	INSERT INTO "test" (number, string)
2	(select trunc(random() * 5000)::int,
3	chr(trunc(65 + random() * 25)::int)
4	from generate_series(1, 100000))

Data output	Messages	Notifications
-------------	----------	---------------

INSERT 0 100000

Query returned successfully in 1 secs 107 msec.

Без використання індекса

Query

Query History

1

explain analyze select * from test order by number

Data output

Messages

Notifications

QUERY PLAN

text

1

Sort (cost=9747.82..9997.82 rows=100000 width=6) (actual time=76.449..96.408 rows=100000 loops=1)

2

Sort Key: number

3

Sort Method: external merge Disk: 1768kB

4









-> Seq Scan on test (cost=0.00..1443.00 rows=100000 width=6) (actual time=0.015..11.358 rows=100000 lo...

5

Planning Time: 0.091 ms

6

Execution Time: 104.118 ms

Query	Query History	
1	<code>explain analyze select * from test where string = 'I' order by number</code>	
Data output	Messages	Notifications
<div></div>		
	<div><div>QUERY PLAN</div><div>text</div><div>1</div><div>Sort (cost=1945.96..1956.47 rows=4203 width=6) (actual time=17.385..17.788 rows=4082 loops=1)</div><div>2</div><div>Sort Key: number</div><div>3</div><div>Sort Method: quicksort Memory: 288kB</div><div>4</div><div>-> Seq Scan on test (cost=0.00..1693.00 rows=4203 width=6) (actual time=0.023..15.849 rows=4082 lo...</div><div>5</div><div>Filter: (string = 'I'::text)</div><div>6</div><div>Rows Removed by Filter: 95918</div><div>7</div><div>Planning Time: 0.143 ms</div><div>8</div><div>Execution Time: 18.153 ms</div></div>	

BRIN індекс

Query

Query History








1 CREATE INDEX ON test USING brin (string);

2 explain analyze select * from test order by number

Data output

Messages

Notifications



QUERY PLAN

text

1 Sort (cost=9747.82..9997.82 rows=100000 width=6) (actual time=71.690..89.429 rows=100000 loops=1)

2 Sort Key: number

3 Sort Method: external merge Disk: 1768kB

4 -> Seq Scan on test (cost=0.00..1443.00 rows=100000 width=6) (actual time=0.021..10.894 rows=100000 loops=1)

5 Planning Time: 0.816 ms

6 Execution Time: 96.778 ms

Як видно на зображенні, сортування з BRIN індексом працює швидше, ніж без нього. BRIN індекс працює так: всі дані діляться на секції, і кожного разу, коли ми шукаємо мінімальне число, ми дивимось на метадані кожної секції. Зазвичай там зберігається мінімальне і максимальне число секції, але може бути й по іншому. Це дозволяє не проглядати зайвий раз деякі ділянки пам'яті.

HASH індекс

Query		Query History
1	CREATE INDEX ON test USING hash (string);	
2	explain analyze select * from test where string = 'I' order by number	
Data output		Messages Notifications
	QUERY PLAN text	
1	Sort (cost=1945.96..1956.47 rows=4203 width=6) (actual time=7.888..8.065 rows=4082 loops=1)	
2	Sort Key: number	
3	Sort Method: quicksort Memory: 288kB	
4	-> Seq Scan on test (cost=0.00..1693.00 rows=4203 width=6) (actual time=0.015..7.091 rows=4082 loops=1)	
5	Filter: (string = 'I'::text)	
6	Rows Removed by Filter: 95918	
7	Planning Time: 1.339 ms	
8	Execution Time: 8.224 ms	

Як видно на зображенні, сортування з HASH індексом працює швидше, ніж без нього. HASH індекс працює так: хеш-індекси **зберігають 32-бітний хеш-код, отриманий зі значення індексованого стовпця**. Отже, такі індекси можуть обробляти лише прості порівняння рівності. Планувальник запитів розглядатиме використання хеш-індексу щоразу, коли індексований стовпець бере участь у порівнянні за допомогою оператора рівності: =.

Завдання 3

Для тестування тригерів створимо таблицю, для зберігання повідомлень від тригеру.

```
1 -- Table: public.messages
2
3 -- DROP TABLE public.messages;
4
5 CREATE TABLE public.messages
6 (
7     id integer NOT NULL DEFAULT nextval('messages_id_seq'::regclass),
8     date_time timestamp with time zone,
9     message character varying COLLATE pg_catalog."default"
10 )
11
12 TABLESPACE pg_default;
13
14 ALTER TABLE public.messages
15     OWNER to postgres;
```

Команда створення тригеру:

```
1 CREATE OR REPLACE FUNCTION my_trigger_func() RETURNS trigger AS $$
2 DECLARE
3     curs CURSOR FOR SELECT * FROM users;
4     ROW users%ROWTYPE;
5 BEGIN
6     IF (TG_OP = 'DELETE') THEN
7         INSERT INTO messages (message, date_time) VALUES ('After Delete operation from users table', NOW());
8         RAISE NOTICE 'Successful delete';
9         RETURN OLD;
10    ELSEIF (TG_OP = 'INSERT') THEN
11        IF NEW.user_id < 2000 THEN
12            RAISE NOTICE 'Id can't be less than 2000';
13            RETURN NULL;
14        END IF;
15
16        FOR ROW IN curs LOOP
17            IF NEW.user_name like ROW.user_name THEN
18                NEW.user_name = NEW.user_name || "_upd";
19            END IF;
20        END LOOP;
21
22        INSERT INTO messages (message, date_time) VALUES ('After Delete operation from users table', NOW());
23        RAISE NOTICE 'Successful insert';
24        RETURN NEW;
25    END IF;
26 END;
27 $$ language plpgsql;
28
29 DROP TRIGGER IF EXISTS my_trigger on users;
30
31 CREATE TRIGGER my_trigger
32 AFTER DELETE OR INSERT
33 ON users
34 FOR EACH ROW EXECUTE PROCEDURE my_trigger_func();
```

Data Output Notifications Explain Messages Query History

CREATE TRIGGER

Query returned successfully in 64 msec.

Виконаємо запит видалення з таблиці:

Query Editor

```
1 DELETE FROM users WHERE user_id = 58503 OR user_id = 85912
```

Data Output Notifications Explain Messages Query History

NOTICE: Successful delete
NOTICE: Successful delete
DELETE 2

Query returned successfully in 68 msec.

Бачимо повідомлення від триггеру. Зміст таблиці Messages також був змінений:

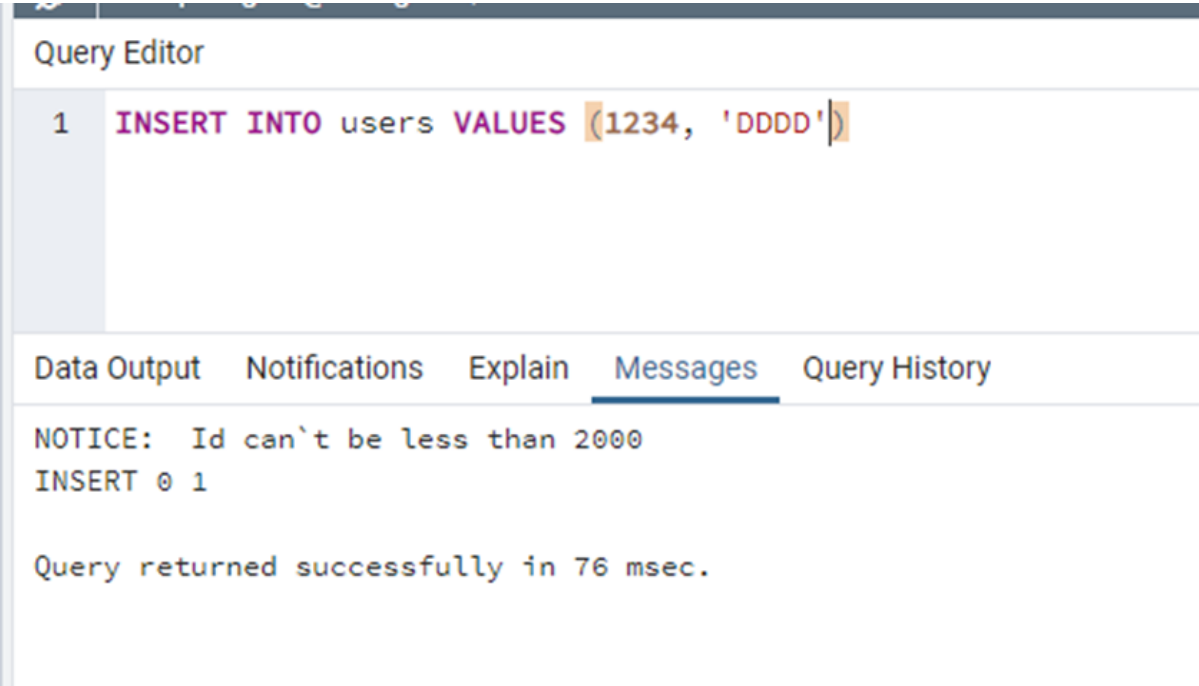
Query Editor

```
1 SELECT * FROM public.messages  
2
```

Data Output Notifications Explain Messages Query History

	id integer	date_time timestamp with time zone	message character varying
1	1	2020-12-05 03:05:29.754673+03	After Delete operation from users table
2	2	2020-12-05 03:05:58.225874+03	After Delete operation from users table
3	3	2020-12-05 03:05:58.225874+03	After Delete operation from users table

Виконаємо запит додавання даних:



The screenshot shows a 'Query Editor' window with a single SQL statement: `1 INSERT INTO users VALUES (1234, 'DDDD')`. Below the editor, there are tabs for 'Data Output', 'Notifications', 'Explain', 'Messages', and 'Query History'. The 'Messages' tab is selected, displaying the following output:
NOTICE: Id can't be less than 2000
INSERT 0 1

Query returned successfully in 76 msec.

Тригер повернув «Помилку» через те, що ми не виконали його умову додавання даних. Виконаємо запит, з збільшенням індексів:



The screenshot shows a 'Query Editor' window with two SQL statements: `1 INSERT INTO users VALUES (123456, 'DDDD');` and `2 INSERT INTO users VALUES (123457, 'DDDD');`. Below the editor, the 'Messages' tab is selected, displaying the following output:
NOTICE: Successful insert
NOTICE: Successful insert
INSERT 0 1

Бачимо повідомлення від тригера. Зміст таблиці users був змінений наступним чином:

```
1 SELECT * from users WHERE user_name LIKE '%DDDD%'
```

Data Output Notifications Explain Messages Query History

	user_id [PK] integer	user_name character varying
1	123456	DDDD
2	123457	DDDD_upd

Зміст таблиці Logs також був змінений:

Query Editor

```
1 SELECT * FROM public.messages
2
```

Data Output Notifications Explain Messages Query History

	id integer	date_time timestamp with time zone	message character varying
1	1	2020-12-05 03:05:29.754673+03	After Delete operation from users table
2	2	2020-12-05 03:05:58.225874+03	After Delete operation from users table
3	3	2020-12-05 03:05:58.225874+03	After Delete operation from users table
4	4	2020-12-05 03:13:10.783992+03	After Delete operation from users table
5	5	2020-12-05 03:13:37.307308+03	After Delete operation from users table
6	6	2020-12-05 03:13:37.307308+03	After Delete operation from users table
7	7	2020-12-05 03:15:15.630531+03	After Delete operation from users table
8	8	2020-12-05 03:15:15.630531+03	After Delete operation from users table
9	9	2020-12-05 03:17:10.451649+03	After Delete operation from users table
10	10	2020-12-05 03:17:10.451649+03	After Delete operation from users table
11	11	2020-12-05 03:17:11.861673+03	Insert operation from users table
12	12	2020-12-05 03:17:11.861673+03	Insert operation from users table
13	13	2020-12-05 03:17:43.12545+03	After Delete operation from users table
14	14	2020-12-05 03:17:43.12545+03	After Delete operation from users table
15	15	2020-12-05 03:17:46.188994+03	Insert operation from users table
16	16	2020-12-05 03:17:46.188994+03	Insert operation from users table

Завдання 4

Створення таблиці:

```
-- Table: public.transaction

-- DROP TABLE IF EXISTS public.transaction;

CREATE TABLE IF NOT EXISTS public.transaction
(
    id bigint NOT NULL DEFAULT nextval('transaction_id_seq'::regclass),
    "numeric" bigint,
    text text COLLATE pg_catalog."default",
    CONSTRAINT transaction_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.transaction
    OWNER to postgres;
```

Query	Query History
1	INSERT INTO "transaction" ("numeric", "text") VALUES (11, 'text1'), (22, 'text2'), (33, 'text3'), (44, 'text4');
Data output	Messages Notifications
INSERT 0 4	
Query returned successfully in 51 msec.	

READ COMMITTED

Read Committed — рівень ізоляції транзакції, який вибирається в Postgres Pro за замовчуванням. У транзакції, що працює на цьому рівні, запит SELECT бачить ті дані, які були зафіксовані до початку запиту; він ніколи не побачить незафіксованих даних або змін, внесених у процесі виконання запиту паралельними транзакціями. По суті, запит SELECT бачить знімок бази даних у момент початку виконання запиту. Тобто, доки паралельні транзакції не завершать своє виконання (commit), якихось змін у даних транзакції видно не буде.

Дані після вставки, видалення та редагування у одній транзакції та інших:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
      8-бітові символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
postgres=# ;
START TRANSACTION
postgres=# set TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# delete from "transaction" where "numeric" = 33;
DELETE 1
postgres=# insert into "transaction"("numeric","text") values (777, 'quertyuiop');
INSERT 0 1
postgres=# update "transaction" set "text" = 'quertyuiop';
UPDATE 4
postgres=# commit;
COMMIT
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
      8-бітові символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | text1
  2  |      22 | text2
  3  |      33 | text3
  4  |      44 | text4
(4 Е Фам)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | text1
  2  |      22 | text2
  3  |      33 | text3
  4  |      44 | text4
(4 Е Фам)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | text1
  2  |      22 | text2
  3  |      33 | text3
  4  |      44 | text4
(4 Е Фам)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | quertyuiop
  2  |      22 | quertyuiop
  4  |      44 | quertyuiop
  5  |      777 | quertyuiop
(4 Е Фам)

postgres=#
```

Друга транзакція не може вносити змін доки не завершиться перша транзакція:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
      8-бітові символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# UPDATE "transaction" set "text" = 'asdfghj';
UPDATE 4
postgres=#

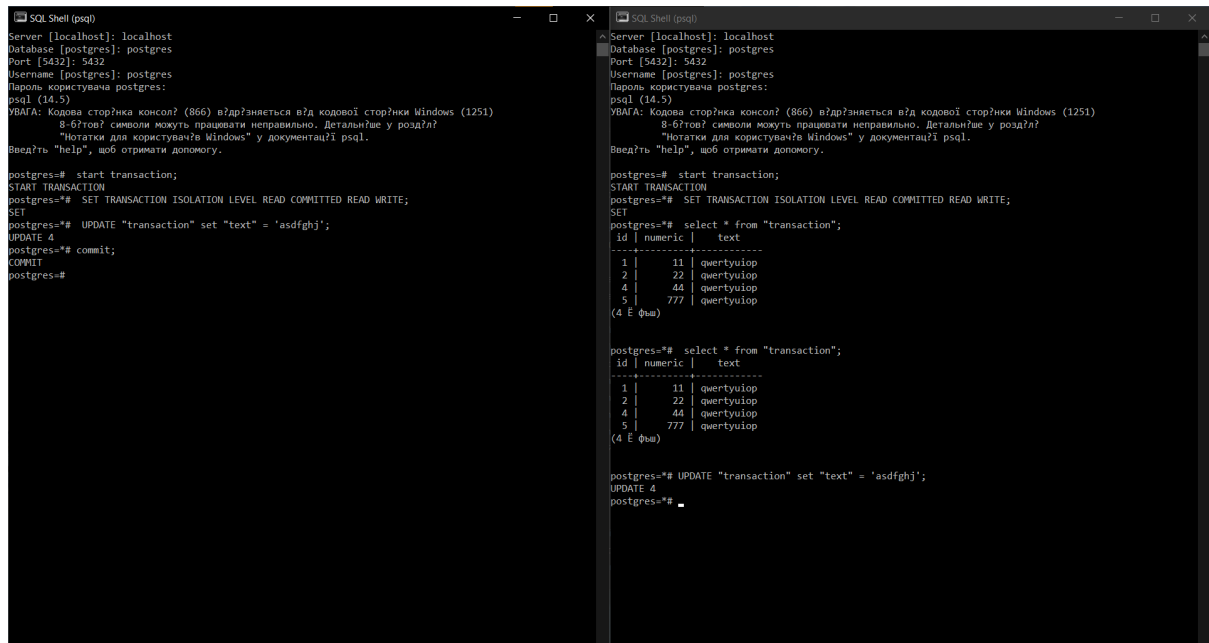
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
      8-бітові символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | quertyuiop
  2  |      22 | quertyuiop
  4  |      44 | quertyuiop
  5  |      777 | quertyuiop
(4 Е Фам)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1  |      11 | quertyuiop
  2  |      22 | quertyuiop
  4  |      44 | quertyuiop
  5  |      777 | quertyuiop
(4 Е Фам)

postgres=# UPDATE "transaction" set "text" = 'asdfghj';
```

Після того, як запити у першій транзакції були “закомічені”, друга транзакція має змогу викликати запити та виконувати їх:



```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) влідр?няється влідр кодової сторінки Windows (1251)
      8-6?тов? символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть? "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# UPDATE "transaction" set "text" = 'asdfghj';
UPDATE 4
postgres=# commit;
COMMIT
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) влідр?няється влідр кодової сторінки Windows (1251)
      8-6?тов? символи можуть працювати неправильно. Детальніше у розділі?
      "Нотатки для користувачів Windows" у документації psql.
Введіть? "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
  1 |      11 | queryuiop
  2 |      22 | queryuiop
  4 |      44 | queryuiop
  5 |      77 | queryuiop
(4 Е фья)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1 |      11 | queryuiop
  2 |      22 | queryuiop
  4 |      44 | queryuiop
  5 |      77 | queryuiop
(4 Е фья)

postgres=# UPDATE "transaction" set "text" = 'asdfghj';
UPDATE 4
postgres=#
```

Тобто, коли друга транзакція бачить зміни у першій транзакції за допомогою запитів UPDATE та DELETE, то виникає феномен повторного читання (транзакція, що читає, «не бачить» зміни даних, які були нею раніше прочитані), а при INSERT виникає читання фантомів (ситуація, коли при повторному читанні в рамках однієї транзакції одна і та ж вибірка дає різні множини рядків). Тобто, на цьому рівні забезпечується захист від чорнового, «брудного» читання, проте, в процесі роботи однієї транзакції інша може бути успішно завершена та зроблені нею зміни зафіксовані.

REPEATABLE READ

У режимі Repeatable Read видно лише ті дані, які були зафіксовані до початку транзакції, але не видно незафіксовані дані та зміни, здійснені іншими транзакціями в процесі виконання цієї транзакції (однак запит бачитиме ефекти попередніх змін у своїй транзакції, незважаючи на те, що вони не зафіксовані).

Друга транзакція не бачить змін першої:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в?др?зняється в?д кодової стор?нки Windows (1251)
8-б?тов? символи можуть працювати неправильно. Детальн?ше у розд?n?
"Нотатки для користувач?в Windows" у документац?ї psql.
Введ?ть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# update "transaction" set "numeric" = "numeric" + 3;
UPDATE 4
postgres=# insert into "transaction"("numeric","text") values (56, 'zxcv');
INSERT 0 1
postgres=# delete from "transaction" where "id" = 5;
DELETE 1
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в?др?зняється в?д кодової стор?нки Windows (1251)
8-б?тов? символи можуть працювати неправильно. Детальн?ше у розд?n?
"Нотатки для користувач?в Windows" у документац?ї psql.
Введ?ть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----+-----+-----
  1 |      11 | asdfghj
  2 |      22 | asdfghj
  4 |      44 | asdfghj
  5 |      777 | asdfghj
(4 Е ?м)

postgres=# select * from "transaction";
 id | numeric | text
-----+-----+-----
  1 |      11 | asdfghj
  2 |      22 | asdfghj
  4 |      44 | asdfghj
  5 |      777 | asdfghj
(4 Е ?м)

postgres=#
```

Спроба втручання до тих самих даних після commit другою транзакцією:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в?др?зняється в?д кодової стор?нки Windows (1251)
8-б?тов? символи можуть працювати неправильно. Детальн?ше у розд?n?
"Нотатки для користувач?в Windows" у документац?ї psql.
Введ?ть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----+-----+-----
  1 |      14 | asdfghj
  2 |      25 | asdfghj
  4 |      47 | asdfghj
  6 |      56 | zxcv
(4 Е ?м)

postgres=# update "transaction" set "numeric" = "numeric" + 1;
UPDATE 4
postgres=# delete from "transaction" where "text" = 'zxcv';
DELETE 1
postgres=# insert into "transaction"("numeric","text") values (44, 'zxcv');
INSERT 0 1
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в?др?зняється в?д кодової стор?нки Windows (1251)
8-б?тов? символи можуть працювати неправильно. Детальн?ше у розд?n?
"Нотатки для користувач?в Windows" у документац?ї psql.
Введ?ть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----+-----+-----
  1 |      14 | asdfghj
  2 |      25 | asdfghj
  4 |      47 | asdfghj
  6 |      56 | zxcv
  8 |      83 | zxcv
(5 Е ?м)

postgres=# insert into "transaction"("numeric","text") values (83, 'zxcv');
INSERT 0 1
postgres=# select * from "transaction";
 id | numeric | text
-----+-----+-----
  1 |      14 | asdfghj
  2 |      25 | asdfghj
  4 |      47 | asdfghj
  6 |      56 | zxcv
  8 |      83 | zxcv
(5 Е ?м)

postgres=# delete from "transaction" where "numeric" = 56;
```

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# update "transaction" set "numeric" = "numeric" + 3;
UPDATE 4
postgres=# insert into "transaction"("numeric","text") values (56, 'zxcv');
INSERT 0 1
postgres=# delete from "transaction" where "id" = 5;
DELETE 1
postgres=# commit;
COMMIT
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    11 | asdfghj
 2 |    22 | asdfghj
 4 |    44 | asdfghj
 5 |    777 | asdfghj
(4 Е ФАВ)

postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    11 | asdfghj
 2 |    22 | asdfghj
 4 |    44 | asdfghj
 5 |    777 | asdfghj
(4 Е ФАВ)

postgres=# update "transaction" set "numeric" = "numeric" + 1;
ПОМИЛКА: не вдалося серйозувати доступ через паралельне оновлення
postgres=#
postgres=#
```

```
SQL Shell (psql)
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    14 | asdfghj
 2 |    25 | asdfghj
 4 |    47 | asdfghj
 6 |    56 | zxcv
(4 Е ФАВ)

postgres=# update "transaction" set "numeric" = "numeric" + 1;
UPDATE 4
postgres=# delete from "transaction" where "text" = 'zxcv';
DELETE 1
postgres=# insert into "transaction"("numeric","text") values (44, 'zxcv');
INSERT 0 1
postgres=# commit;
COMMIT
postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    15 | asdfghj
 2 |    26 | asdfghj
 4 |    48 | asdfghj
13 |    44 | zxcv
(4 Е ФАВ)

postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    15 | asdfghj
 2 |    26 | asdfghj
 4 |    48 | asdfghj
13 |    44 | zxcv
(4 Е ФАВ)
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level repeatable read read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    14 | asdfghj
 2 |    25 | asdfghj
 4 |    47 | asdfghj
 6 |    56 | zxcv
(4 Е ФАВ)

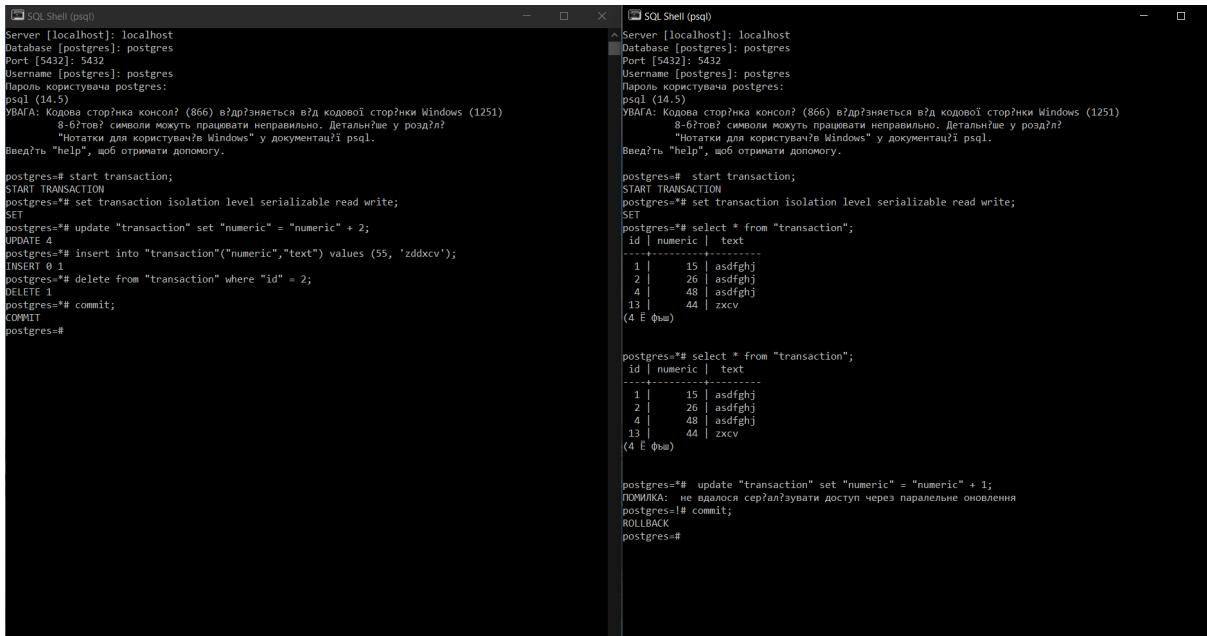
postgres=# insert into "transaction"("numeric","text") values (92, 'zxcv');
INSERT 0 1
postgres=# select * from "transaction";
 id | numeric | text
-----
 1 |    14 | asdfghj
 2 |    25 | asdfghj
 4 |    47 | asdfghj
 6 |    56 | zxcv
12 |    92 | zxcv
(5 Е ФАВ)

postgres=# delete from "transaction" where "numeric" = 56;
ПОМИЛКА: не вдалося серйозувати доступ через паралельне оновлення
postgres=# commit;
ROLLBACK
postgres=#
```

Видно, що читання фантомів в цьому випадку виникає, тобто можна використати insert у другій транзакції, поки не закінчилася перша, тут спрацює перший commit, і усі зміни у другій транзакції, можна сказати, будуть зроблені просто так та ніде не будуть збережені. У цьому рівні ізоляції є заборона іншим транзакціям змінювати рядки, які були зчитані незавершеною транзакцією. Однак інші транзакції можуть вставляти нові рядки. Користуватися даним та вищими рівнями транзакцій без необхідності зазвичай не рекомендується.

SERIALIZABLE

Найвищий рівень ізолюваності; транзакції повністю ізолюються одна від одної, кожна виконується так, ніби паралельних транзакцій не існує. Тільки на цьому рівні паралельні транзакції не схильні до ефекту “фантомного читання”. Дії у першій та другій транзакції:



```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в джерелі від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Потятки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level serializable read write;
SET
postgres=# update "transaction" set "numeric" = "numeric" + 2;
UPDATE 4
postgres=# insert into "transaction"("numeric","text") values (55, 'zddxcv');
INSERT 0 1
postgres=# delete from "transaction" where "id" = 2;
DELETE 1
postgres=# commit;
COMMIT
postgres=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Пароль користувача postgres:
psql (14.5)
УВАГА: Кодова сторінка консолі? (866) в джерелі від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Потятки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# start transaction;
START TRANSACTION
postgres=# set transaction isolation level serializable read write;
SET
postgres=# select * from "transaction";
 id | numeric | text
-----
  1 |      15 | asdfghj
  2 |      26 | asdfghj
  4 |      48 | asdfghj
 13 |      44 | zxcv
(4 E 0ms)

postgres=# select * from "transaction";
 id | numeric | text
-----
  1 |      15 | asdfghj
  2 |      26 | asdfghj
  4 |      48 | asdfghj
 13 |      44 | zxcv
(4 E 0ms)

postgres=# update "transaction" set "numeric" = "numeric" + 1;
ПОМИЛКА: не вдалося серіалізувати доступ через паралельне оновлення
postgres=# commit;
ROLLBACK
postgres=#
```

Даний рівень призначений для недопущення читання фантомів. На цьому рівні ізоляції гарантується максимальна узгодженість даних.