

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

# «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Тема:

Засоби оптимізації роботи СУБД PostgreSQL

Виконала:

cтуденка групи КВ-83

Мягкова Дар’я

Київ – 2020

**Завдання**

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.

*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:М, М:М 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*

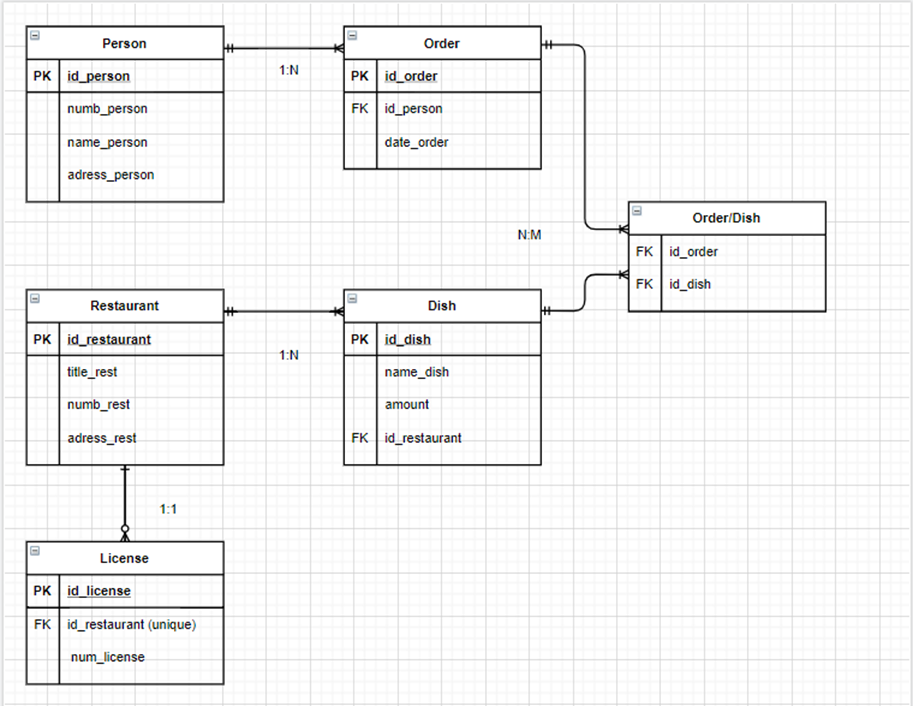
Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування 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.

Варіант 14

|  |  |  |
| --- | --- | --- |
| *14* | *Btree, Hash* | *after insert, update* |



Для роботи з sqlaclemy було імпортовано такі засоби:

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.orm import relationship

from sqlalchemy import Column, Integer, String, Date,

ForeignKey

from sqlalchemy import text

Підключення до бази та створення сесії:

engine =

create\_engine('postgresql://postgres:nvidia123@localhost:5432/postgres2')

Session = sessionmaker(bind=engine)

Base = declarative\_base()

Створені класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками.

class Person(base.Base):  
 \_\_tablename\_\_ = 'Person'  
 id\_person = Column('id\_person', Integer, primary\_key=True)  
 numb\_person = Column('numb\_person', Text)   
 name\_person = Column('name\_person', Text)   
 adress\_person = Column('adress\_person', Text)  
 order = relationship('Order')   
  
 def \_\_repr\_\_(self):  
 return "<Person(id\_peson='{}', numb\_person='{}', name\_person='{}', adress\_person='{}')>"\  
 .format(self.id\_person, self.numb\_person, self.name\_person, self.adress\_person)  
  
class Order(base.Base):  
 \_\_tablename\_\_ = 'Order'  
 id\_order = Column('id\_order', Integer, primary\_key=True)  
 id\_person = Column('id\_person', Integer, ForeignKey('Person.id\_person'))   
 date\_order = Column('date\_order', Text)   
 order/dish = relationship('Order/Dish')   
  
 def \_\_repr\_\_(self):  
 return "<Order(id\_order='{}', id\_person='{}', date\_order='{}')>"\  
 .format(self.id\_order, self.id\_person, self.date\_order)

class License(base.Base):  
 \_\_tablename\_\_ = 'License'  
 id\_license = Column('id\_license', Integer, primary\_key=True)  
 id\_restaurant = Column('id\_restaurant', Integer, ForeignKey('Restaurant.id\_restaurant'))   
 num\_license = Column('num\_license', Text)   
  
 def \_\_repr\_\_(self):  
 return "<License(id\_license='{}', id\_restaurant='{}', num\_license='{}')>"\  
 .format(self.id\_license, self.id\_restaurant, self.num\_license)

class Restaurant(base.Base):  
 \_\_tablename\_\_ = 'Restaurant'  
 id\_restaurant = Column('id\_restaurant ', Integer, primary\_key=True)  
 title\_rest = Column('title\_rest’, Text)   
 numb\_rest = Column('numb\_rest', Text) adress\_rest = Column('adress\_rest', Text) license = relationship('License') dish = relationship('Dish')  
  
 def \_\_repr\_\_(self):  
 return "<Restaurant(id\_restaurant='{}', title\_rest='{}', numb\_rest='{}', adress\_rest='{}')>"\  
 .format(self.id\_restaurant, self.title\_rest, self.numb\_rest, self.adress\_rest)

class Dish(base.Base):  
 \_\_tablename\_\_ = 'Dish'  
 id\_dish = Column('id\_dish', Integer, primary\_key=True)  
 amount = Column('amount’, Text)   
 name\_dish = Column('name\_dish', Text) id\_restaurant = Column('id\_restaurant', Integer, ForeignKey('Restaurant.id\_restaurant'))   
  
  
 def \_\_repr\_\_(self):  
 return "<Dish(id\_dish='{}', amount='{}', name\_dish='{}', id\_restaurant='{}')>"\  
 .format(self.id\_dish, self.amount, self.name\_dish, self.id\_restaurant)

class Order/Dish(base.Base):  
 \_\_tablename\_\_ = 'Order/Dish'  
 id\_dish = Column('id\_dish', Integer, ForeignKey('Dish.id\_dish'))  
 id\_order = Column('id\_order', Integer, ForeignKey('Order.id\_order'))   
   
  
 def \_\_repr\_\_(self):  
 return "<Order/Dish(id\_dish='{}', id\_order='{}')>"\  
 .format(self.id\_dish, self.id\_order)

Функції, що реалізують вставки, вилучення та редагування екземплярів класів-сутностей.

def add\_row1(self, table\_name, fields\_name):

try:

obj = get\_class\_by\_tablename(table\_name)(\*\*fields\_name)

self.session.add(obj)

self.session.commit()

return True

except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

print(error)

self.session.rollback()

return False

def del\_row1(self, table\_name, key, value):

try:

table\_class = get\_class\_by\_tablename(table\_name)

filter\_txt = "{} = '{}'".format(key, value)

self.session.query(table\_class).filter(text(filter\_txt)).delete(synchronize\_session=False)

self.session.commit()

except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

print(error)

self.session.rollback()

return False

def edit\_value1(self, table\_name, key, key\_change, new\_val, key\_val):

try:

table\_class = get\_class\_by\_tablename(table\_name)

filter\_txt = "{} = '{}'".format(key, key\_val)

update\_values = {key\_change: new\_val}

self.session.query(table\_class) \

.filter(text(filter\_txt)) \

.update(update\_values, synchronize\_session=False)

self.session.commit()

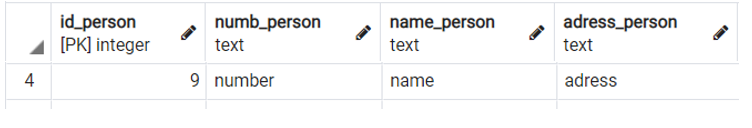
except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

print(error)

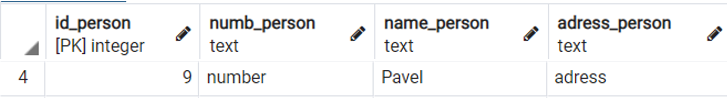
self.session.rollback()

return False

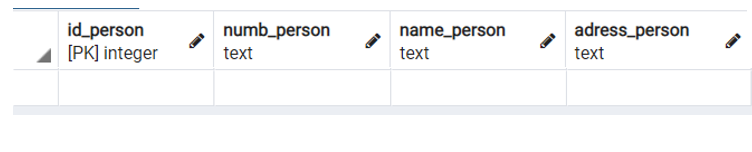
db.add\_row('Person', {'id\_person': '9', 'numb\_person': 'number', 'name\_person': 'name', 'adress\_person': 'adress'})



db.edit\_value('Person', 'id\_person', 'adress\_person', 'numb\_person', 'name\_person', 'Pavel', 'number', 'adress', '9'})

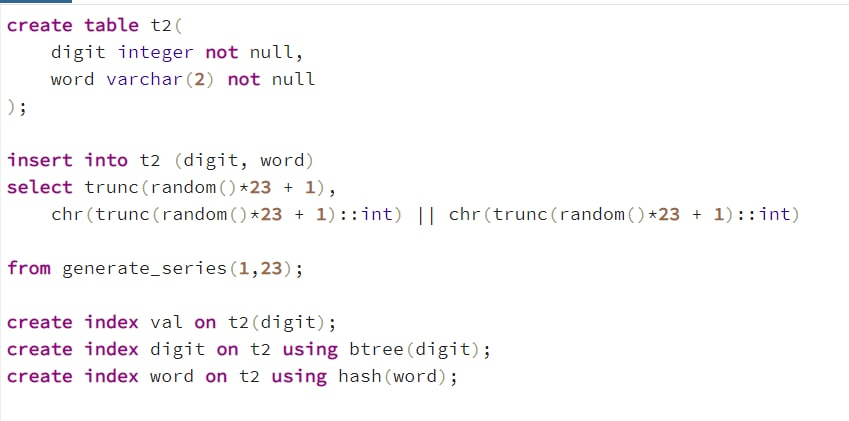


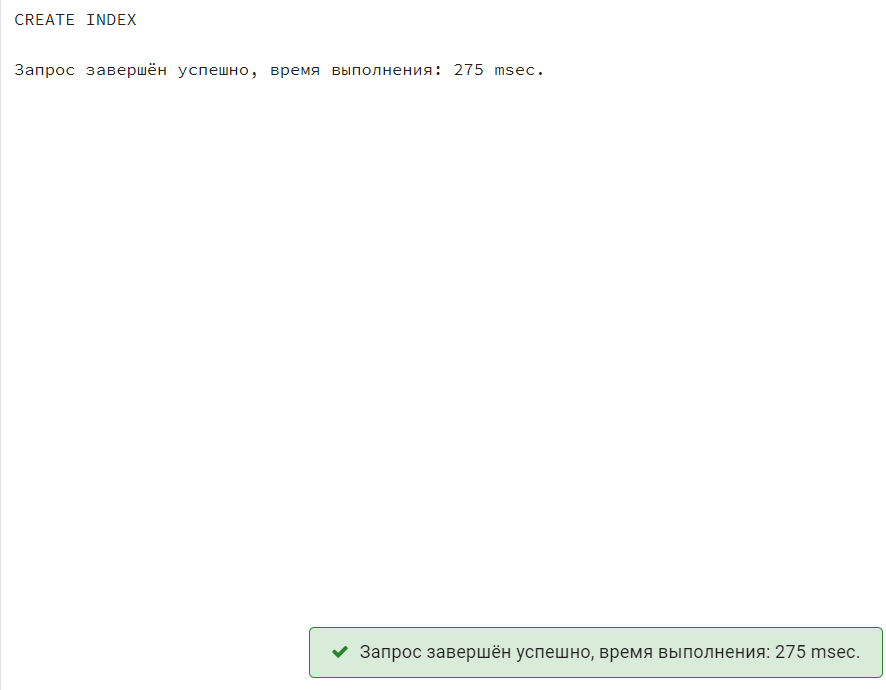
db.del\_row('Person', 'id\_person', '9')

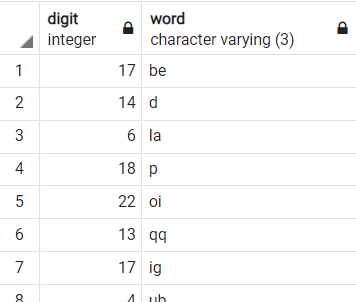


**Завдання 2**

В даному завданні розглянемо використання індексів Btree, Hash. Для створення індексів для текстового поля використовувалися такі команди:







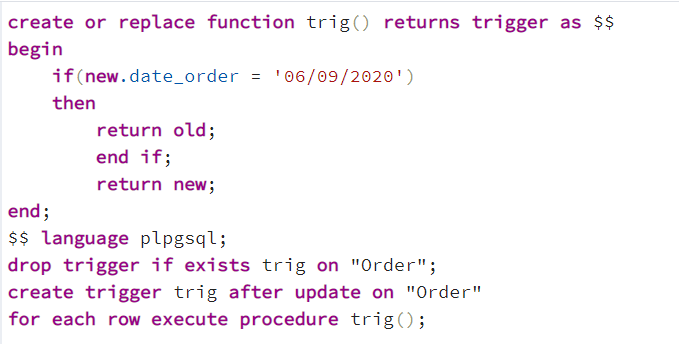
Висновок:

Індекси допомагають SQL серверу знаходити дані. Вони прискорюють вибір даних, вказуючи серверу положення табличних даних на диску.

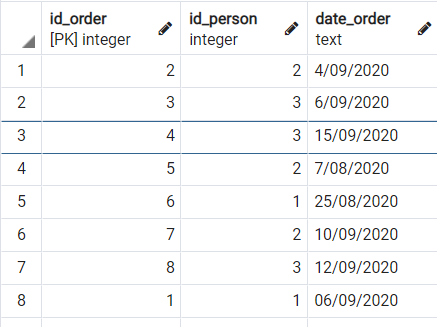
Перевага Btree: BTree, на відміну від інших дерев, створені спеціально для ефективної роботи з дисковою пам'яттю, а точніше - вони мінімізують звернення типу введення-виведення. B-дерево безсумнівно є більш вигідною структурою даних для випадків, коли ми маємо великий обсяг інформації. Переміщуючи вузол дерева в оперативну пам'ять, ми переміщуємо виділений блок послідовної пам'яті, тому ця операція досить швидко працює.

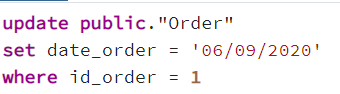
Індекс Hash добре використовувати коли необхідно прискорити пошук і також важливим є розмір індексу. Що до розміру індексу Hash, то він є «плоским» в порівнянні з індексом Btree, тому є значно меншим за розміром.

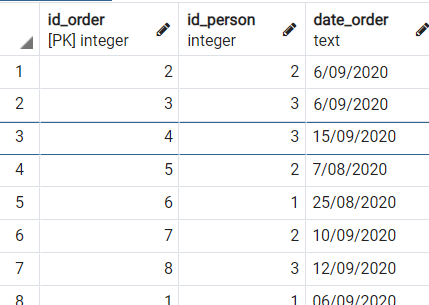
**Завдання 3**

****

SELECT \* FROM public.“Order”







**Завдання 4**

Необхідні файли додані до репозиторію.