

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №3**

з дисципліни **Бази даних і засоби управління**

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

Виконав:

студент ІII курсу

групи КВ-01

Філіпенко Д. О.

Перевірив: Павловський В. І.

Київ – 2022

**Варіант 17**

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.

**Логічна модель бази даних**

Нижче (Рисунок 1) наведено логічну модель бази даних:

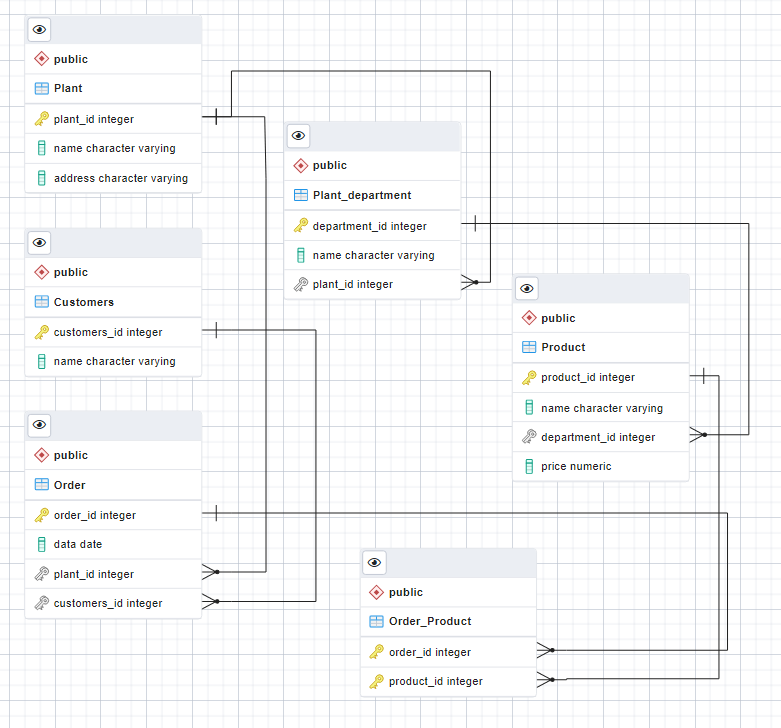


Рисунок 1 – Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні.

Посилання на репозиторій: <https://github.com/Dobrunya123/bdzy>

**Класи ORM**

class Plant(Base):  
 \_\_tablename\_\_ = 'Plant'  
 plant\_id = Column(Integer, primary\_key=True)  
 name = Column(String)  
 address = Column(String)  
  
 def \_\_init\_\_(self, name, address):  
 self.name = name  
 self.address = address  
 super(Plant, self).\_\_init\_\_()  
  
  
class PlantDepartment(Base):  
 \_\_tablename\_\_ = 'Plant\_department'  
 department\_id = Column(Integer, primary\_key=True)  
 name = Column(String)  
 plant\_id = Column(Integer, ForeignKey('Plant.plant\_id'))  
  
 plant = relationship('Plant', foreign\_keys=[plant\_id])  
  
 def \_\_init\_\_(self, name, plant\_id):  
 self.name = name  
 self.plant\_id = plant\_id  
 super(PlantDepartment, self).\_\_init\_\_()  
  
  
class Product(Base):  
 \_\_tablename\_\_ = 'Product'  
 product\_id = Column(Integer, primary\_key=True)  
 name = Column(String)  
 department\_id = Column(Integer, ForeignKey('Plant\_department.department\_id'))  
 price = Column(Integer)  
  
 department = relationship('PlantDepartment', foreign\_keys=[department\_id])  
  
 def \_\_init\_\_(self, name, department\_id, price):  
 self.name = name  
 self.department\_id = department\_id  
 self.price = price  
 super(Product, self).\_\_init\_\_()  
  
  
class Order(Base):  
 \_\_tablename\_\_ = 'Order'  
 order\_id = Column(Integer, primary\_key=True)  
 plant\_id = Column(Integer, ForeignKey('Plant.plant\_id'))  
 data = Column(sqlalchemy.Time)  
 customers\_id = Column(Integer, ForeignKey('Customers.customers\_id'))  
  
 plant = relationship("Plant", foreign\_keys=[plant\_id])  
 customers = relationship("Customers", foreign\_keys=[customers\_id])  
  
 def \_\_init\_\_(self, plant\_id, date, customers\_id):  
 self.plant\_id = plant\_id  
 self.date = date  
 self.customers\_id = customers\_id  
 super(Order, self).\_\_init\_\_()  
  
  
class OrderProduct(Base):  
 \_\_tablename\_\_ = 'Order\_Product'  
 order\_id = Column(Integer, ForeignKey('Order.order\_id'), primary\_key=True)  
 product\_id = Column(Integer, ForeignKey('Product.product\_id'), primary\_key=True)  
  
 order = relationship("Order", foreign\_keys=[order\_id])  
 product = relationship("Product", foreign\_keys=[product\_id])  
  
 def \_\_init\_\_(self, order\_id, product\_id):  
 self.order\_id = order\_id  
 self.product\_id = product\_id  
 super(OrderProduct, self).\_\_init\_\_()  
  
  
class Customers(Base):  
 \_\_tablename\_\_ = 'Customers'  
 customers\_id = Column(Integer, primary\_key=True)  
 name = Column(String)  
  
 def \_\_init\_\_(self, name):  
 self.name = name  
 super(Customers, self).\_\_init\_\_()

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

**Таблиці:**

CREATE TABLE employee (

id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

username varchar NOT NULL,

salary integer NOT NULL,

last\_update timestamp

);

INSERT INTO employee (username, salary, last\_update)

VALUES ('Billy', 1000, current\_timestamp);

CREATE TABLE employee\_audit (

operation varchar(10) NOT NULL,

stamp timestamp NOT NULL,

employee\_id integer NOT NULL,

username varchar NOT NULL,

salary integer NOT NULL

);

**Функція:**

CREATE OR REPLACE FUNCTION employee\_stamp() RETURNS trigger AS $employee\_stamp$

BEGIN

IF (TG\_OP = 'DELETE') THEN

INSERT INTO employee\_audit SELECT 'DELETE', now(), OLD.id, OLD.username, OLD.salary;

ELSIF (TG\_OP = 'UPDATE') THEN

IF NEW.salary < 0 THEN

RAISE EXCEPTION '% cannot have a negative salary', NEW.username;

END IF;

INSERT INTO employee\_audit SELECT 'UPDATE', now(), NEW.id, NEW.username, NEW.salary;

END IF;

NEW.last\_update := current\_timestamp;

RETURN NEW;

END;

$employee\_stamp$ LANGUAGE plpgsql;

**Тригер:**

CREATE TRIGGER employee\_stamp BEFORE UPDATE OR DELETE ON employee

FOR EACH ROW EXECUTE FUNCTION employee\_stamp();

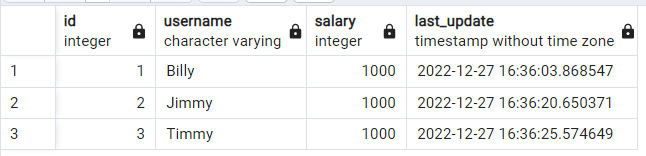
**Принцип роботи тригеру**

Тригер спрацьовує перед редагуванням або видаленням в таблиці «employee». При редагуванні або видаленні данних з таблиці «employee» у таблицю «employee\_audit» буде заноситись рядок про тип операції, часова мітка виконання та дані про робітника. Якщо обрана операція була UPDATE, тоді ще проводиться додаткова перевірка на ввід некоректних даних (зарплатня не може бути менше 0).

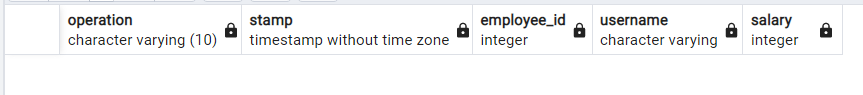
***Приклад***

**Таблиці до змін:**

employee



employee\_audit



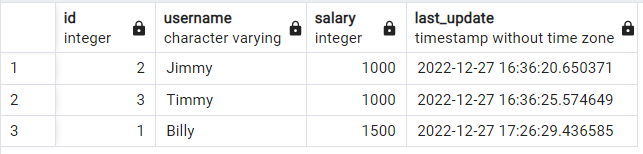
**Виконаємо оновлення одного рядка:**

UPDATE employee

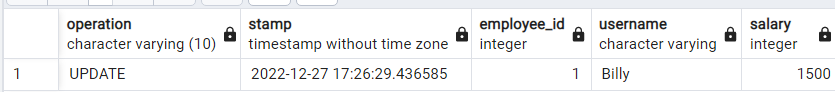
SET salary = 1500

WHERE id = 1;

employee



employee\_audit



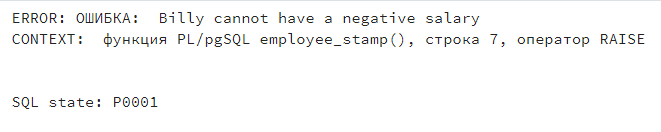
**Спробуємо задати некоректні дані:**

UPDATE employee

SET salary = -1500

WHERE id = 1;

Отримуємо помилку, а самі таблиці ніяк не змінились

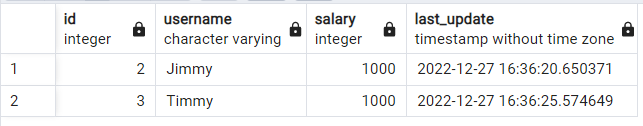


**Виконаємо видалення рядка:**

DELETE FROM employee

WHERE id = 1;

employee



employee\_audit

