**Міністерство освіти і науки України**

**Національний технічний університет України**

**“Київський політехнічний інститут ім. Ігоря Сікорського”**

**Факультет прикладної математики**

**Кафедра спеціалізованих комп’ютерних систем**

**Лабораторна робота № 3**

з дисципліни «Бази даних і засоби управління»

«Ознайомлення з базовими операціями СУБД PostgreSQL»

Виконав:

студент групи КВ-73

Булаєвський Ігор Олегович

Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання

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.

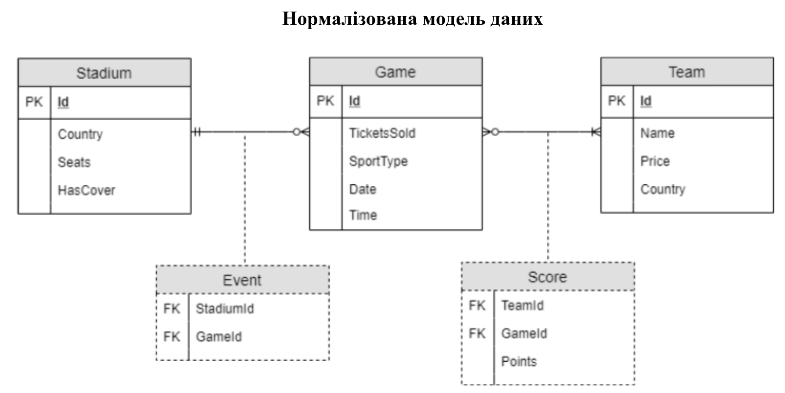
4.Навести приклади та проаналізувати рівні ізоляції транзакцій уPostgreSQL.

Порядок виконання роботи

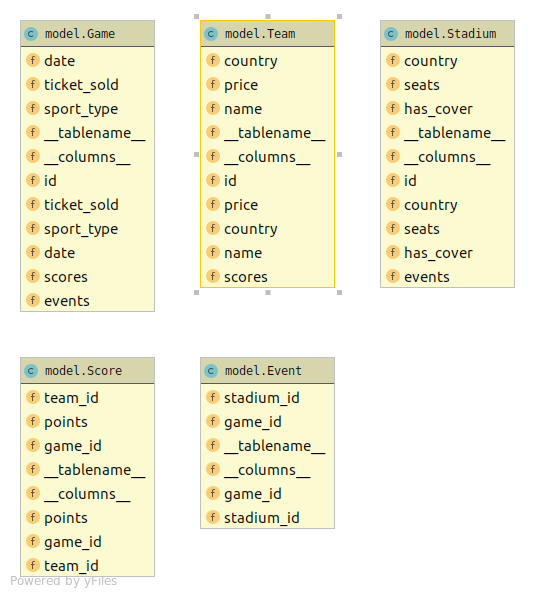
В ході роботи розроблено:

1. Логічну модель БД та Діаграму класів;
2. Функціонал програмного додатку;
3. ОО програмний додаток роботи з БД " … ". Для взаємодії з БД використано ORM модуль SQLAlchemy.

Логічна модель бази даних наведена на Рис 1.

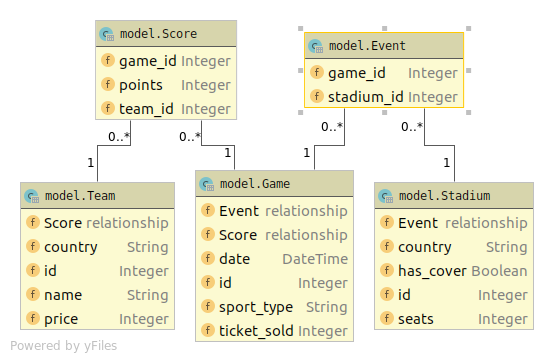


*Рис 1. Логічна модель бази даних*

Сутнісні класи програми наведені на Рис 2.

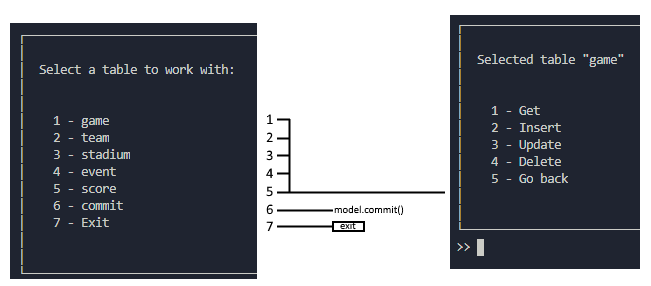
*Рис 1. Фрагмент UML діаграми сутнісних класів*

Зв’язки між сутнісними класами, сгенеровані за допомогою SqlAlchemy, наведені на Рис 3.



*Рис 3. Зв’язки між сутнісними класами*

Меню програми наведене на Рис 4.



*Рис 4. Меню програми*

Лістинг програми

controller.py

from consolemenu import SelectionMenu

import model

import view

import reader

def handle\_error(func):

def wrapper(tname):

try:

func(tname)

except Exception as e:

show\_table\_menu(tname, str(e))

return wrapper

def show\_start\_menu(tname=None, err=''):

tables = list(model.TABLES.keys())

menu = SelectionMenu(tables + ['commit'], subtitle=err,

title="Select a table to work with:")

menu.show()

index = menu.selected\_option

if index < len(tables):

tname = tables[index]

show\_table\_menu(tname)

elif index == len(tables):

model.commit()

show\_start\_menu(err='All chages were commited')

else:

print('Bye! Have a nice day!')

def show\_table\_menu(tname, subtitle=''):

opts = ['Get', 'Insert', 'Update', 'Delete']

steps = [get, insert, update, delete]

if tname == 'team':

opts.append('Create 100\_000 random teams')

steps.append(create\_random\_team)

steps.append(show\_start\_menu)

menu = SelectionMenu(

opts, subtitle=subtitle,

title=f'Selected table "{tname}"', exit\_option\_text='Go back', )

menu.show()

index = menu.selected\_option

steps[index](tname=tname)

@handle\_error

def get(tname):

query = reader.multiple\_input(tname, 'Enter requested fields:', empty=True)

data = model.get(tname, query)

view.print\_entities(tname, data)

reader.press\_enter()

show\_table\_menu(tname)

@handle\_error

def insert(tname):

data = reader.multiple\_input(tname, 'Enter new fields values:')

model.insert(tname, data)

show\_table\_menu(tname, 'Insertion was made successfully')

@handle\_error

def update(tname):

condition = reader.single\_input(

tname, 'Enter requirement of row to be changed:')

query = reader.multiple\_input(tname, 'Enter new fields values:')

model.update(tname, condition, query)

show\_table\_menu(tname, 'Update was made successfully')

@handle\_error

def delete(tname):

query = reader.multiple\_input(

tname, 'Enter requirement of row to be deleted:')

model.delete(tname, query)

show\_table\_menu(tname, 'Deletion was made successfully')

@handle\_error

def create\_random\_team(tname):

model.create\_random\_teams()

show\_table\_menu(tname, '100\_000 random teams were successfully added')

model.py

from sqlalchemy import Column, Integer, String, DateTime, \

Boolean, ForeignKey, create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import relationship

from sqlalchemy.orm import sessionmaker

db\_str = 'postgres://admin:admin@localhost:5432/kpi'

db = create\_engine(db\_str)

Base = declarative\_base()

class Game(Base):

\_\_tablename\_\_ = 'game'

\_\_columns\_\_ = ('id', 'ticket\_sold', 'sport\_type', 'date')

id = Column(Integer, primary\_key=True)

ticket\_sold = Column(Integer)

sport\_type = Column(String)

date = Column(DateTime)

scores = relationship('Score')

events = relationship('Event')

def \_\_init\_\_(self, ticket\_sold=None, sport\_type=None, date=None):

*self*.ticket\_sold = ticket\_sold

*self*.sport\_type = sport\_type

*self*.date = date

class Stadium(Base):

\_\_tablename\_\_ = 'stadium'

\_\_columns\_\_ = ('id', 'country', 'seats', 'has\_cover')

id = Column(Integer, primary\_key=True)

country = Column(String)

seats = Column(Integer)

has\_cover = Column(Boolean)

events = relationship('Event')

def \_\_init\_\_(self, country=None, seats=None, has\_cover=None):

*self*.country = country

*self*.seats = seats

*self*.has\_cover = has\_cover

class Team(Base):

\_\_tablename\_\_ = 'team'

\_\_columns\_\_ = ('id', 'price', 'country', 'name')

id = Column(Integer, primary\_key=True)

price = Column(Integer)

country = Column(String)

name = Column(String)

scores = relationship('Score')

def \_\_init\_\_(self, price=None, country=None, name=None):

*self*.price = price

*self*.country = country

*self*.name = name

class Score(Base):

\_\_tablename\_\_ = 'score'

\_\_columns\_\_ = ('points', 'game\_id', 'team\_id')

points = Column(Integer)

game\_id = Column(Integer, ForeignKey('game.id'), primary\_key=True)

team\_id = Column(Integer, ForeignKey('team.id'), primary\_key=True)

def \_\_init\_\_(self, points, game\_id=None, team\_id=None):

*self*.points = points

*self*.game\_id = game\_id

*self*.team\_id = team\_id

class Event(Base):

\_\_tablename\_\_ = 'event'

\_\_columns\_\_ = ('game\_id', 'stadium\_id')

game\_id = Column(Integer, ForeignKey('game.id'), primary\_key=True)

stadium\_id = Column(Integer, ForeignKey('stadium.id'), primary\_key=True)

def \_\_init\_\_(self, game\_id=None, stadium\_id=None):

*self*.game\_id = game\_id

*self*.stadium\_id = stadium\_id

session = sessionmaker(db)()

Base.metadata.create\_all(db)

MODELS = {

'game': Game, 'team': Team, 'stadium': Stadium,

'event': Event, 'score': Score

};

TABLES = dict((tname, MODELS[tname].\_\_columns\_\_) for tname in MODELS)

def insert(tname, opts):

object\_class = MODELS[tname]

obj = object\_class(\*\*opts)

session.add(obj)

def get(tname, opts=None):

objects\_class = MODELS[tname]

objects = session.query(objects\_class)

for key, item in opts.items():

objects = objects.filter(getattr(objects\_class, key) == item)

return list(objects)

def update(tname, condition, opts):

column, value = condition

object\_class = MODELS[tname]

filter\_attr = getattr(object\_class, column)

obj = session.query(object\_class).filter(filter\_attr == value).one()

for key, item in opts.items():

setattr(obj, key, item)

def delete(tname, opts):

objects\_class = MODELS[tname]

objects = session.query(objects\_class)

for key, item in opts.items():

objects = objects.filter(getattr(objects\_class, key) == item)

objects.delete()

def create\_random\_teams():

with open('scripts/random.sql', 'r') as file:

sql = file.read()

session.execute(sql)

def commit():

session.commit()

reader.py

import model

def single\_input(colname=None, msg=None):

if msg:

print(msg)

if colname:

print(f'{colname}=', end='')

return input()

def single\_input(tname, msg):

print(msg)

print('(use format <attribute>=<value>)')

print(f'({"/".join(model.TABLES[tname])})', end='\n\n')

while True:

data = input()

if not data or data.count('=') != 1:

print('Invalid input, try one more time')

continue

data = data.split('=')

col, val = data[0].strip(), data[1].strip()

if col.lower() in [tcol.lower() for tcol in model.TABLES[tname]]:

return col, val

else:

print(f'Invalid column name "{col}" for table "{tname}"')

def multiple\_input(tname, msg, empty=False):

print(msg)

print('(use format <attribute>=<value>)')

print(f'({"/".join(model.TABLES[tname])})', end='\n\n')

res = {}

while True:

data = input()

if not data:

break

if data.count('=') != 1:

print('Invalid input')

continue

data = data.split('=')

col, val = data[0].strip(), data[1].strip()

if col.lower() in [tcol.lower() for tcol in model.TABLES[tname]]:

res[col] = val

else:

print(f'Invalid column name "{col}" for table "{tname}"')

if not res:

if empty:

return {}

raise Exception('You entered nothing')

return res

def press\_enter():

input()

view.py

COLUMN\_WIDTH = 30

def print\_entities(tname, data):

print(f'Working with table "{tname}"', end='\n\n')

if not data:

print('List is empty')

return

entities = data

cols = data[0].\_\_columns\_\_

separator\_line = '-' \* COLUMN\_WIDTH \* len(cols)

print(separator\_line)

print(''.join([f'{col} |'.rjust(30, ' ') for col in cols]))

print(separator\_line)

for entity in entities:

print(''.join([f'{getattr(entity, col)} |'.rjust(30, ' ') for col in cols]))

print(separator\_line)

main.py

if \_\_name\_\_ == '\_\_main\_\_':

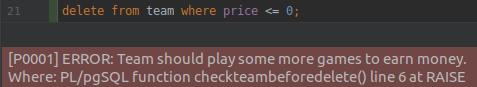
import controller

controller.show\_start\_menu()

**Тригер**

BEFORE DELETE

При видаленні з таблиці Team перевіряється чи команда має додатню ціну. В інакшому випадку видалення не відбувається.



Код функції тригеру:

create or replace function checkTeamBeforeDelete()

returns trigger

language plpgsql

as $$

begin

if (old::team).price <= 0 then

raise exception 'Team should play some more games to earn money.';

end if;

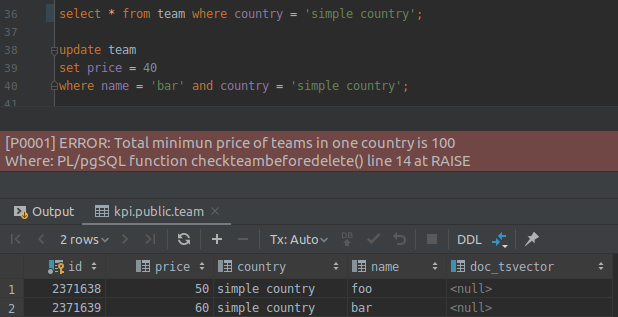
return old;

end;

$$;

BEFORE UPDATE

При оновлені в таблиці Team підраховується загальна вартість команд з поточної країни. Якщо після оновлення загальна вартість буде менша за 100, то оновлення не дозволяється.



Код функції тригеру:

create or replace function checkTeamBeforeDelete()

returns trigger

language plpgsql

as $$

declare

newTeam team = (new::team);

teams cursor is select \* from team where country = newTeam.country and id != newTeam.id;

minimumTotalPrice integer = 100;

totalPrice integer = newTeam.price;

begin

for team in teams

loop

totalPrice := totalPrice + team.price;

end loop;

if totalPrice < minimumTotalPrice then

raise exception 'Total minimun price of teams in one country is 100';

end if;

return new;

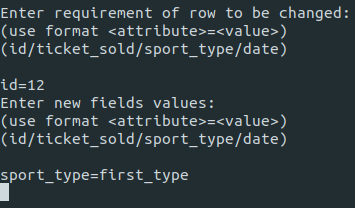
end;

$$;

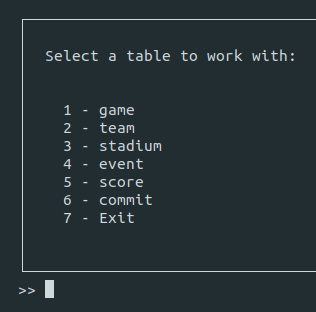
**Дослідження рівнів ізоляції**

1. READ COMMITTED

Клієнт #1



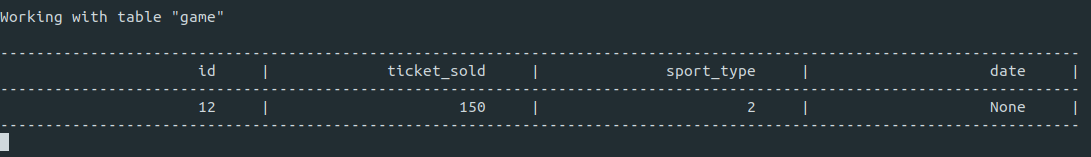
Клієнт #2



Клієнт #1



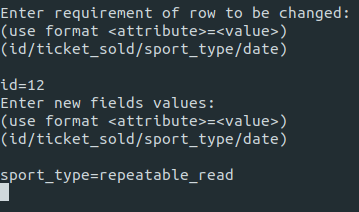
Клієнт #2



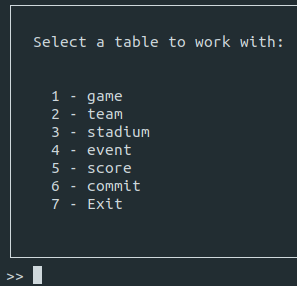
Другий клієнт так і не побачив закомічені зміни першого клієнта, тому що транзакція другого клієнта почалася до коміту першого.

2. REPEATABLE READ

Клієнт #1



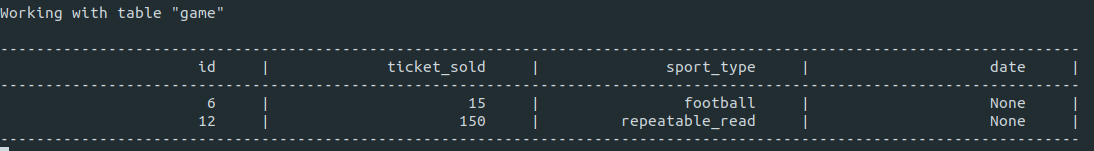
Клієнт #2



Клієнт #1



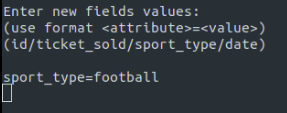
Клієнт #2

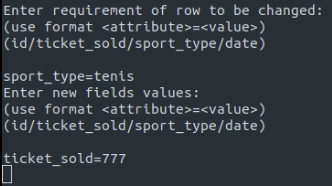


Другий клієнт побачив закомічені зміни першого клієнта, тому що транзакція другого клієнта почалася до коміту першого.

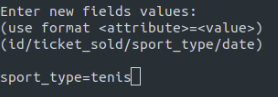
3. SERIALIZABLE

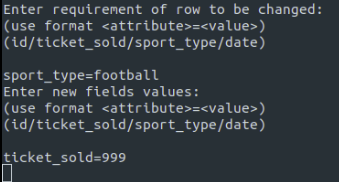
Клієнт #1





Клієнт #2





Клієнт #1



Клієнт #2