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.

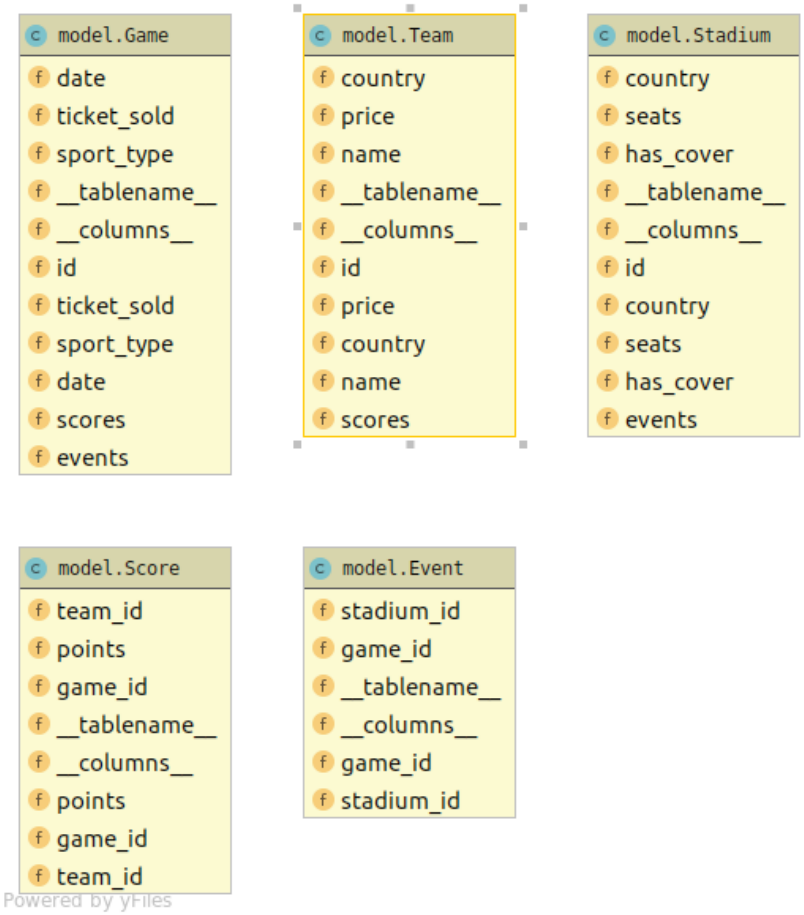


Рис 1. Фрагмент UML діаграми сутнісних класів

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

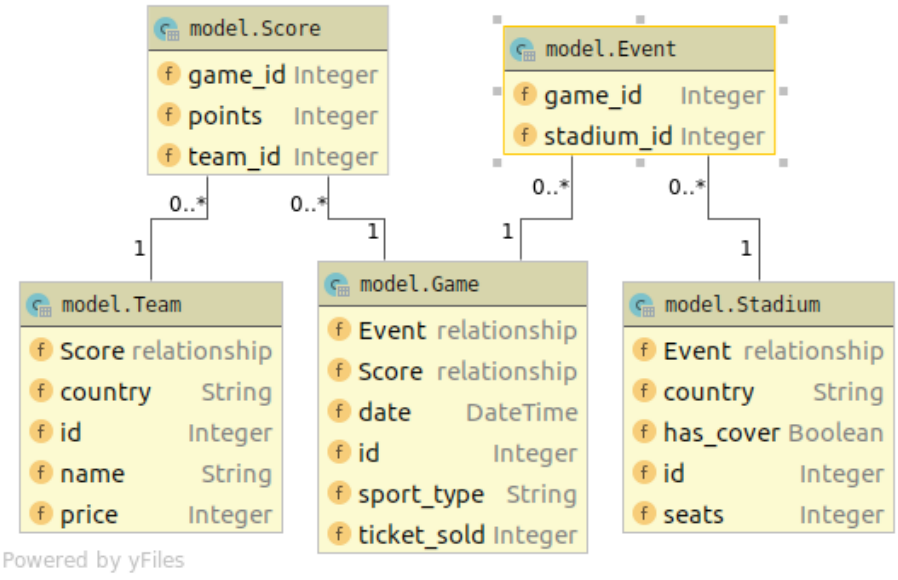


Рис 3.

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)
```

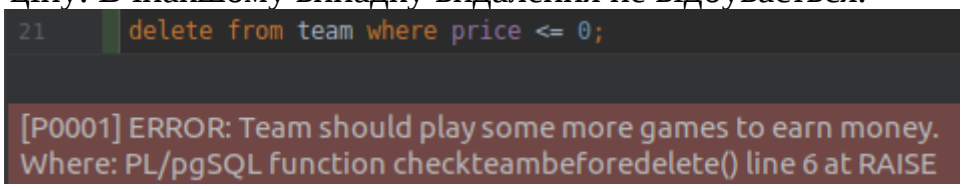
main.py

```
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.

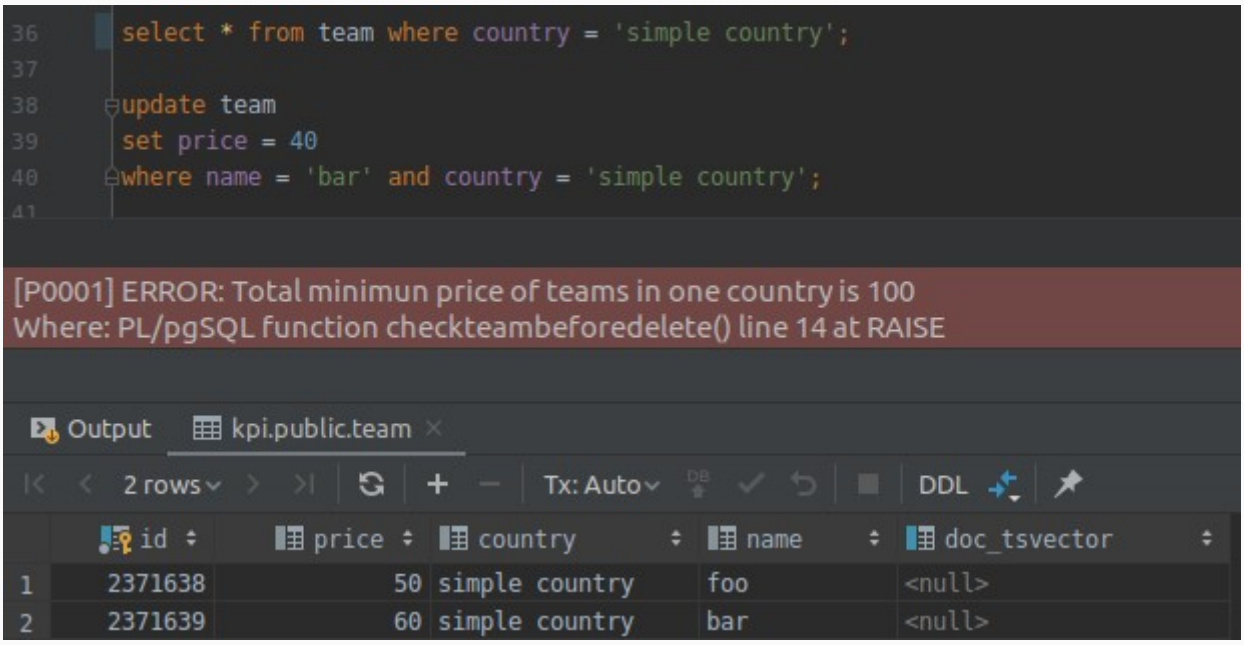


Код функції тригеру:

```
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

```
Enter requirement of row to be changed:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

id=12
Enter new fields values:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

sport_type=first_type
```

Клієнт #2

```
Select a table to work with:

  1 - game
  2 - team
  3 - stadium
  4 - event
  5 - score
  6 - commit
  7 - Exit

>>
```

Клієнт #1

```
All chages were commited
```

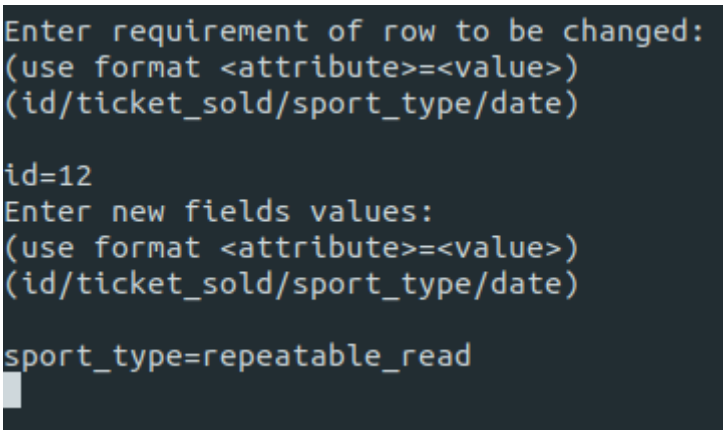
Клієнт #2

```
Working with table "game"
-----------------------------------------------------------------------------------------------
|                  id     |            ticket_sold     |             sport_type     |                  date     |
-----------------------------------------------------------------------------------------------
|                  12     |                  150     |                      2     |                  None     |
-----------------------------------------------------------------------------------------------
```

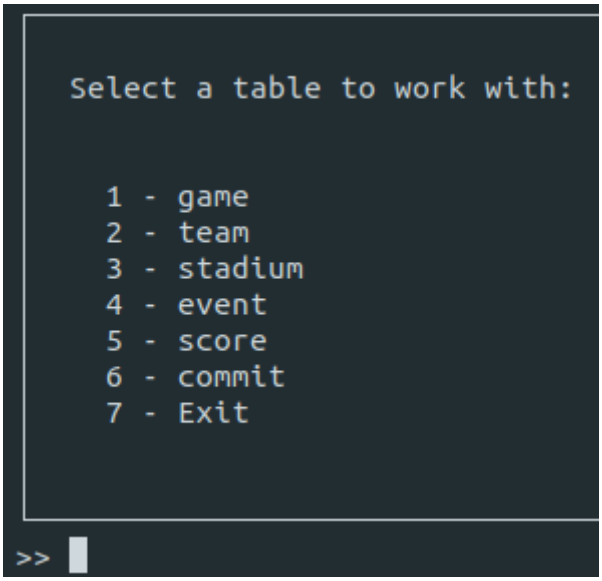
Другий клієнт так і не побачив закомічені зміни першого клієнта, тому що транзакція другого клієнта почалася до коміту першого.

2. REPEATABLE READ

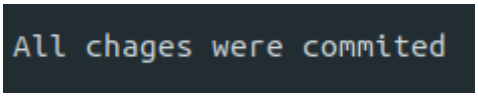
Клієнт #1



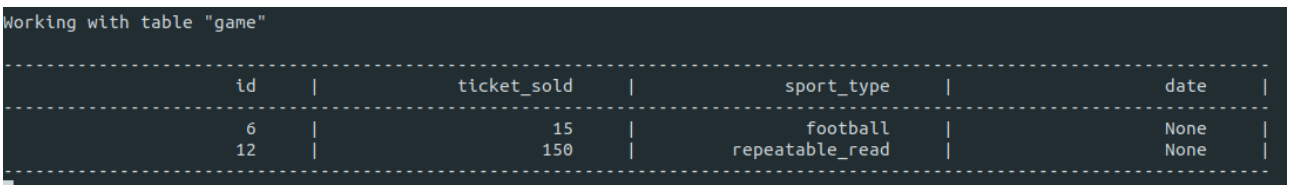
Клієнт #2



Клієнт #1



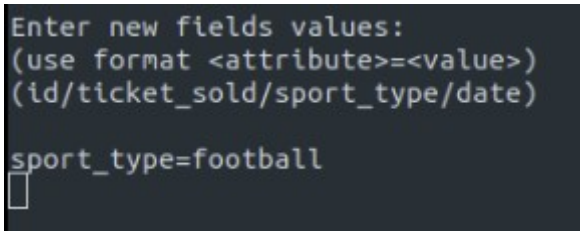
Клієнт #2



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

3. SERIALIZABLE

Клієнт #1



```
Enter requirement of row to be changed:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

sport_type=tenis
Enter new fields values:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

ticket_sold=777
```

Клієнт #2

```
Enter new fields values:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

sport_type=tenis
```

```
Enter requirement of row to be changed:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

sport_type=football
Enter new fields values:
(use format <attribute>=<value>)
(id/ticket_sold/sport_type/date)

ticket_sold=999
```

Клієнт #1

```
All chages were commited
```

Клієнт #2

```
    (psycopg2.errors.SerializationFailure) could not serialize access due to read/write dependencies among transactions
DETAIL:  Reason code: Canceled on identification as a pivot, during conflict out checking.
HINT:  The transaction might succeed if retried.
```