

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра програмного забезпечення та комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент ІII курсу

ФПМ групи КП-73

Булаєвський Ігор Олегович

Перевірив(ла):

Київ – 2019

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Мета роботи:* здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

*Завдання* роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги:*

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

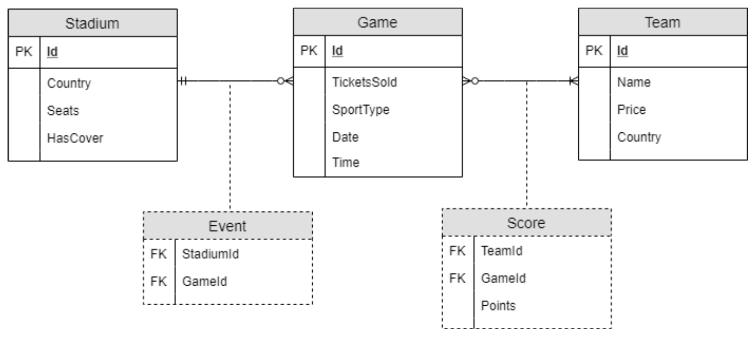
*Вимоги до інтерфейсу користувача:*

1. Використовувати консольний інтерфейс користувача.

**Варіант 3**

Пошук за атрибутами має відбуватися по переліченню строк та логічному типу.

Повнотекстовий пошук з обов’язковим входженням цілого слова або для випадку, коли вказане слово не входить в документ.

** Нормалізована модель даних**

Таблиця Stadium знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Stadium знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Stadium знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Game знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Game знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Game знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Team знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Team знаходиться у 2 НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Team знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Event знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Event знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинних ключів GameId, StadiumId. Таблиця Event знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Score знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Score знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинних ключів GameId, TeamId. Таблиця Score знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

**Опис програми**

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з модулів model , view та controller.

У модулі model реалізовані функції , що здійснюють SQL запити до Бази Даних.

У модулі view реалізовані функції виводу даних з таблиць.

У класі Controller реалізовані функції для відповідних меню та допоміжні функції.

**Опис структури меню програми**

 Меню програми можна розглядати як її концептуальну модель

**Лістинг програми**

if \_\_name\_\_ == '\_\_main\_\_':

    import controller

    controller.show\_start\_menu()

**Controller**

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='', err=''):

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

    menu = SelectionMenu(tables + ['Custom SQL query'], 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):

        custom\_query()

    else:

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

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

    opts = ['Get all', 'Get by atrribute', 'Insert', 'Update', 'Delete']

    steps = [get\_all, get\_by\_attr, insert,

             update, delete]

    if tname == 'Game':

        opts += ['Get games by stadium has cover',

                 'Full text search in game document']

        steps += [get\_games\_hascover, fts]

    elif tname == 'Team':

        opts += ['Get teams by sport type', 'Create 10\_000 random teams']

        steps += [get\_team\_sporttype, create\_random\_team]

    steps += [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\_all(tname):

    data = model.get(tname)

    view.print\_entities(tname, data)

    reader.press\_enter()

    show\_table\_menu(tname)

@handle\_error

def get\_by\_attr(tname):

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

    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 get\_games\_hascover(tname):

    query = reader.specified\_input(

        'hasCover', 'Enter hasCover value:'

    ).lower() in ['true', 't', 'yes', 'y', '+']

    data = model.get\_games\_by\_stadium\_hascover(query)

    view.print\_entities(f'Games with hasCover={query}', data)

    reader.press\_enter()

    show\_table\_menu(tname)

@handle\_error

def get\_team\_sporttype(tname):

    types = []

    query = reader.specified\_input(

        msg='Enter your queries for sportype:').lower()

    while query:

        types.append(query)

        query = reader.specified\_input().lower()

    data = model.get\_teams\_by\_sporttype(types)

    view.print\_entities(

        f'Teams which played at least one of this games: {types}', data)

    reader.press\_enter()

    show\_table\_menu(tname)

@handle\_error

def fts(tname):

    query = reader.specified\_input(

        'query', 'Enter your query to search in document:')

    contains = reader.specified\_input(

        msg='Query word shoul be in document?'

    ).lower() in ['true', 't', 'yes', 'y', '+']

    data = model.fts(query, contains)

    view.print\_entities(

        f'Documents corresponding to query={query} ({"" if contains else "not "}contains)', data)

    reader.press\_enter()

    show\_table\_menu(tname)

@handle\_error

def create\_random\_team(tname):

    model.create\_random\_teams()

    show\_table\_menu(tname, '10\_000 random teams were successfully added')

def custom\_query():

    try:

        sql = reader.multiline\_input('Enter your SQL query')

        data = model.execute(sql)

        view.print\_entities('Custom query result', data)

        reader.press\_enter()

        show\_start\_menu()

    except Exception as e:

        show\_start\_menu(err=str(e))

**View**

COLUMN\_WIDTH = 30

def print\_entities(tname, data):

    entities, cols = data

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

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

    print(separator\_line)

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

    print(separator\_line)

    for entity in entities:

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

    print(separator\_line)

**Model**

import psycopg2

conn = psycopg2.connect("dbname='kpi' user='admin'"

                        "host='localhost' password='admin'")

cursor = conn.cursor()

TABLES = {

    'Game': ('Id', 'TicketSold', 'SportType', 'Date'),

    'Stadium': ('Id', 'Country', 'Seats', 'Hascover'),

    'Team': ('Id', 'Price', 'Country', 'Name'),

    'Score': ('GameId', 'TeamId', 'Points'),

    'Event': ('GameId', 'StadiumId')

}

def create\_tables():

    with open('scripts/create.sql') as file:

        command = file.read()

        cursor.execute(command)

        conn.commit()

def insert(tname, opts):

    try:

        cols = opts.keys()

        vals = [f"'{val}'" for val in opts.values()]

        comand = f'insert into {tname} ({", ".join(cols)}) ' + \

            f'values ({", ".join(vals)})'

        cursor.execute(comand)

    finally:

        conn.commit()

def get(tname, opts=None):

    comand = f'select \* from {tname}'

    if opts:

        conditions = [f"{col}='{opts[col]}'" for col in opts]

        comand = f'{comand} where {" and ".join(conditions)}'

    cursor.execute(comand)

    return cursor.fetchall(), TABLES[tname]

def update(tname, condition, opts):

    try:

        column, value = condition

        updates = ', '.join([f"{col} = '{opts[col]}'" for col in opts])

        comand = f'update {tname} set {updates} where {column}={value}'

        cursor.execute(comand)

        conn.commit()

    finally:

        conn.commit()

def delete(tname, opts):

    try:

        conditions = [f"{col}='{opts[col]}'" for col in opts]

        comand = f'delete from {tname} where {" and ".join(conditions)}'

        cursor.execute(comand)

    finally:

        conn.commit()

def get\_teams\_by\_sporttype(sporttypes):

    sporttypes = [f"'{stype}'" for stype in sporttypes]

    comand = f'''

    select name from team

    join score on team.id=score.teamid

    join game on game.id=score.gameid

    where lower(sporttype) in ({", ".join(sporttypes)})'''

    cursor.execute(comand)

    return cursor.fetchall(), ('TeamName',)

def get\_games\_by\_stadium\_hascover(hascover):

    comand = f'''

    select \* from game

    where id in (select gameid from event

    join stadium on stadium.id=event.stadiumid

    where hascover={hascover})'''

    cursor.execute(comand)

    return cursor.fetchall(), TABLES['Game']

def fts(query, contains):

    sql = f'''

    select gameid, name, sporttype from (

        select

            gameid,

            name,

            sporttype,

            to\_tsvector(name) ||

            to\_tsvector(sporttype) as document

        from score

        join game g on score.gameid = g.id

        join team t on score.teamid = t.id) search

    where search.document @@ to\_tsquery('{'' if contains else '!'}{query}')'''

    cursor.execute(sql)

    return cursor.fetchall(), ('GameId', 'TeamName', 'SportType')

def create\_random\_teams():

    try:

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

            sql = file.read()

            cursor.execute(sql)

    finally:

        conn.commit()

def execute(sql):

    try:

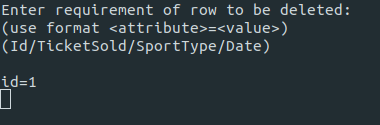
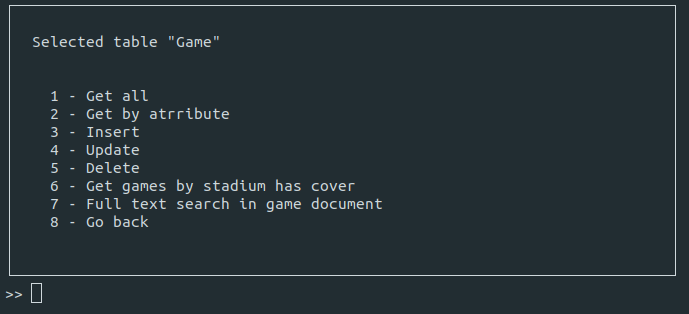
        cursor.execute(sql)

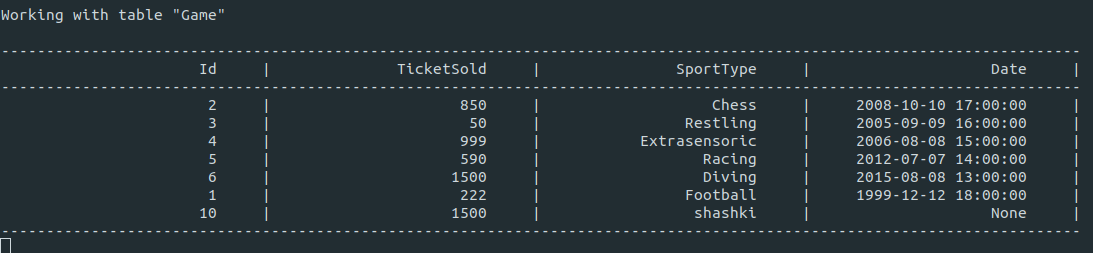
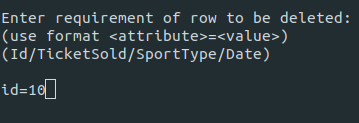
        return cursor.fetchall(), [desc[0] for desc in cursor.description]

    finally:

        conn.commit()

**Скріншот результатів виконання операції вилучення**

****Спроба вилучення кортежу на який є посилання

Вдале видалення і деомонстрація результату вилучення

