

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 III курсу
ФПМ групи КП-81
Длубак Родіон Романович

Київ – 2020

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

Мета роботи: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Вимоги до інтерфейсу користувача:

1. Використовувати консольний інтерфейс користувача.

Деталізоване завдання № 1:

Забезпечити можливість введення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв'язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглий таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

а) ілюстрації обробки виняткових ситуацій (помилки) при введенні/вилучення даних:

```
Name of table: item

ОШИБКА: INSERT или UPDATE в таблице "item" нарушает ограничение внешнего ключа "fk_game_id"
DETAIL: Ключ (game)=(222) отсутствует в таблице "game".

1 - Get
2 - Delete
3 - Update
4 - Insert
5 - Back

>> █
```

рис.1. Помилка при додаванні нового рядка в таблицю 'item'

```
| Name of table: game |
| ОШИБКА: UPDATE или DELETE в таблице "game" нарушает ограничение внешнего ключа "fk_game_id" таблицы "item" |
DETAIL: На ключ (game_id)=(75) всё ещё есть ссылки в таблице "item".
|
| 1 - Get
| 2 - Delete
| 3 - Update
| 4 - Insert
| 5 - Back
|
>>
```

рис.2. Помилка при спробі видалити запис в таблиці 'game' на який посилається запис в таблиці 'item'

б) ілюстрації валідації даних при введенні користувачем.

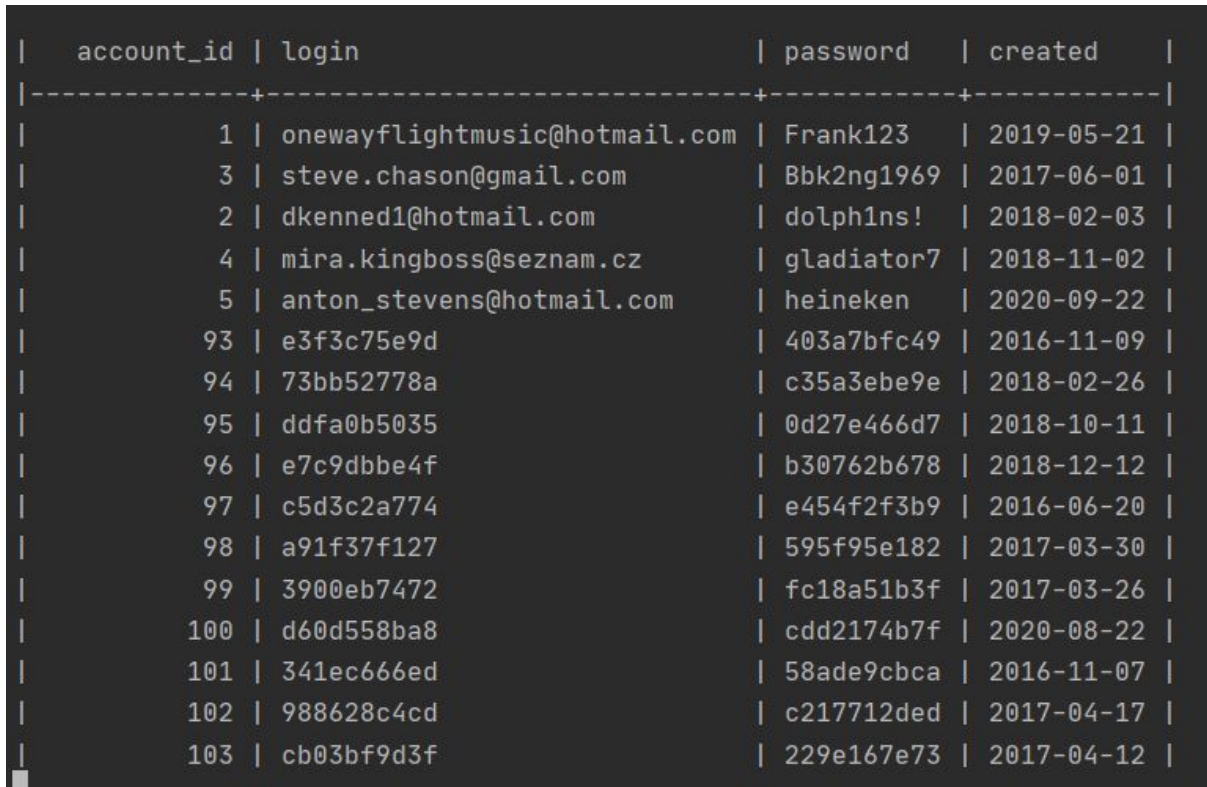
```
| Name of table: item |
| ОШИБКА: неверный синтаксис для типа numeric: "wdsfd" |
LINE 1: ...NTO item (item_name,price,game) VALUES ('ssdds2','wdsfd',2)...
                                                ^
|
| 1 - Get
| 2 - Delete
| 3 - Update
| 4 - Insert
| 5 - Back
|
>>
```

рис.3. Помилка при введенні некоректного типу даних

Деталізоване завдання № 2:

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

- *копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць.*



```
| account_id | login | password | created |
|-----+-----+-----+-----|
| 1 | onewayflightmusic@hotmail.com | Frank123 | 2019-05-21 |
| 3 | steve.chason@gmail.com | Bbk2ng1969 | 2017-06-01 |
| 2 | dkenned1@hotmail.com | dolph1ns! | 2018-02-03 |
| 4 | mira.kingboss@seznam.cz | gladiator7 | 2018-11-02 |
| 5 | anton_stevens@hotmail.com | heineken | 2020-09-22 |
| 93 | e3f3c75e9d | 403a7bfc49 | 2016-11-09 |
| 94 | 73bb52778a | c35a3ebe9e | 2018-02-26 |
| 95 | ddfa0b5035 | 0d27e466d7 | 2018-10-11 |
| 96 | e7c9dbbe4f | b30762b678 | 2018-12-12 |
| 97 | c5d3c2a774 | e454f2f3b9 | 2016-06-20 |
| 98 | a91f37f127 | 595f95e182 | 2017-03-30 |
| 99 | 3900eb7472 | fc18a51b3f | 2017-03-26 |
| 100 | d60d558ba8 | cdd2174b7f | 2020-08-22 |
| 101 | 341ec666ed | 58ade9cbca | 2016-11-07 |
| 102 | 988628c4cd | c217712ded | 2017-04-17 |
| 103 | cb03bf9d3f | 229e167e73 | 2017-04-12 |
```

рис.4. Таблиця 'account' після генерації 10-ти записів з випадковими даними

Деталізоване завдання № 3:

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість введення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

- ілюстрації введення пошукового запиту та результатів виконання запитів.

```
GET accounts and games
Enter date of creation range. First date (like: '2020-01-01'), then second (like: '2020-10-10') or leave empty
2018-01-01
2022-01-01
Enter game name (like: 'CS:GO') or leave empty
CS:GO
Enter game hours range. First number (like: '1200'), then second (like: '2000') or leave empty
1000
5000
| account_id | login | password | created | game_id | game_name | hours_played | account |
|-----+-----+-----+-----+-----+-----+-----+-----|
| 4 | mira.kingboss@seznam.cz | gladiator7 | 2018-11-02 | 3 | CS:GO | 3432 | 4 |
```

рис.5. Пошук в таблицях 'account' і 'game' з обмеженням по даті створення акаунту, конкретній грі та ігровому часу

```
GET games and items
Enter game name (like: 'CS:GO') or leave empty
DOTA2
Enter game hours range. First number (like: '1200'), then second (like: '2000') or leave empty
400
500
Enter item price. First number (like: '10'), then second (like: '1000') or leave empty
500
1000
| game_id | game_name | hours_played | account | item_name | price | game | item_id |
|-----+-----+-----+-----+-----+-----+-----+-----|
| 1 | DOTA2 | 421 | 2 | Dragonclaw hook | 900 | 1 | 1 |
```

рис.6. Пошук в таблицях 'game' і 'item' з обмеженням по грі, ігровому часу та вартості предмету

Деталізоване завдання № 4:

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL (без ORM).

main.py

```
from controller import Controller  
  
Controller().show_main_menu()
```

model.py

```
import psycopg2  
from pprint import pprint  
  
class Model:  
    def __init__(self):  
        try:  
            self.connection = psycopg2.connect(  
                "dbname = 'lab1' user='postgres' host='127.0.0.1'  
password='03rod06ion01' port='5432'")  
            self.connection.autocommit = True  
            self.cursor = self.connection.cursor()  
        except:  
            pprint("Cannot connect to database")  
  
    def get_column_names(self):  
        return [d[0] for d in self.cursor.description]  
  
    def showtable(self, tablename, condition):  
        try:  
            query = f'SELECT * FROM {tablename}'  
  
            if condition:  
                query += ' WHERE ' + condition  
  
            self.cursor.execute(query)  
            return self.get_column_names(), self.cursor.fetchall()  
        finally:  
            self.connection.commit()  
  
            self.cursor.close()  
  
    def insert(self, tablename, columns, new_record):  
        try:
```

```

        query = f'INSERT INTO {tablename} ({columns}) VALUES
({new_record});'
        self.cursor.execute(query)
        finally:
            self.connection.commit()

        #self.cursor.close()

def delete(self, tablename, condition):
    try:
        query = f'DELETE FROM {tablename} WHERE {condition};'

        self.cursor.execute(query)
    finally:
        self.connection.commit()

def update(self, tablename, condition, statement):
    try:
        query = f'UPDATE {tablename} SET {statement} WHERE {condition}'

        self.cursor.execute(query)
    finally:
        self.connection.commit()

def get_table_id(self, tablename):
    try:
        query = f'SELECT {tablename}_id FROM {tablename}'

        self.cursor.execute(query)
        return self.get_column_names(), self.cursor.fetchall()
    finally:
        self.connection.commit()

def get_accounts_and_games(self, condition):
    try:
        query = f'''
            SELECT * from account
            JOIN game on account_id=account'''
        if condition:
            query += condition
        self.cursor.execute(query)
        return self.get_column_names(), self.cursor.fetchall()
    finally:
        self.connection.commit()

def get_games_and_items(self, condition):
    try:
        query = f'''
            SELECT * from game
            JOIN item on game_id=game'''
        if condition:
            query += condition
        self.cursor.execute(query)
        return self.get_column_names(), self.cursor.fetchall()
    finally:
        self.connection.commit()

def get_accounts_games_and_items(self, condition):

```



```

try:
    query = f'''
        SELECT * from account
        JOIN game on account_id=account
        JOIN item on game_id=game'''
    if condition:
        query += condition
    self.cursor.execute(query)
    return self.get_column_names(), self.cursor.fetchall()
finally:
    self.connection.commit()

def fill_account_with_random_data(self):
    sql = """
CREATE OR REPLACE FUNCTION randomAccounts()
    RETURNS void AS $$
DECLARE
    step integer := 0;
BEGIN
    LOOP EXIT WHEN step > 10;
        INSERT INTO account (login, password, created)
        VALUES (
            substring(md5(random()::text), 1, 10),
            substring(md5(random()::text), 1, 10),
            timestamp '2014-01-10 20:00:00' + random() *
(timestamp '2020-12-30 20:00:00' - timestamp '2014-01-10 10:00:00'));
            step := step + 1;
        END LOOP ;
END;
$$ LANGUAGE PLPGSQL;
SELECT randomAccounts();
"""
    try:
        self.cursor.execute(sql)
    finally:
        self.connection.commit()

```

view.py

```

from tabulate import tabulate

class View:
    def print(self, data):
        columns, rows = data
        arr = [];
        for r in rows:
            arr.append([r[0], r[1], r[2], r[3]])
        print(tabulate(arr, headers=columns, tablefmt='orgtbl'))

    def print_2(self, data):
        columns, rows = data
        arr = [];
        for r in rows:
            arr.append([r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7]])

```

```

        print(tabulate(arr, headers=columns, tablefmt='orgtbl'))

    def print_3(self, data):
        columns, rows = data
        arr = [];
        for r in rows:
            arr.append([r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7], r[8],
r[9], r[10], r[11]])
            print(tabulate(arr, headers=columns, tablefmt='orgtbl'))

)

```

controller.py

```

from consolemenu import *
from consolemenu.items import *
from model import Model
from view import View

TABLES_NAMES = ['account', 'game', 'item']
TABLES = {
    'account': ['account_id', 'login', 'password', 'created'],
    'game': ['game_id', 'game_name', 'hours_played', 'account'],
    'item': ['item_name', 'price', 'game', 'item_id']
}

def getInput(msg, tableName=''):
    print(msg)
    if tableName:
        print(' | '.join(TABLES[tableName]), end='\n\n')
    return input()

def getInsertInput(msg, tableName):
    print(msg)
    print(' | '.join(TABLES[tableName]), end='\n\n')
    return input(), input()

def pressEnter():
    input()

class Controller:
    def __init__(self):
        self.model = Model()
        self.view = View()

    def show_main_menu(self, msg=''):
        menu = SelectionMenu(TABLES_NAMES + ['Get accounts and games'] +
                             ['Get games and items'] +
                             ['Get accounts, games and items'] +
                             ['Fill table "account" with random
data (10 items)'], "Main menu", msg)

```

```

        menu.show()

        index = menu.selected_option
        if index < len(TABLES_NAMES):
            tableName = TABLES_NAMES[index]
            self.show_entity_menu(tableName)
        elif index == 3:
            self.get_accounts_and_games()
        elif index == 4:
            self.get_games_and_items()
        elif index == 5:
            self.get_accounts_games_and_items()
        elif index == 6:
            self.fillByRandom()
        else:
            print('Goodbye')

    def show_entity_menu(self, tableName, msg=''):
        options = ['Get', 'Delete', 'Update', 'Insert']
        functions = [self.get, self.delete, self.update, self.insert]

        selectionMenu = SelectionMenu(options, f'Name of table: {tableName}',
msg,
        exit_option_text='Back')
        selectionMenu.show()
        try:
            function = functions[selectionMenu.selected_option]
            function(tableName)
        except IndexError:
            self.show_main_menu()

    def get(self, tableName):
        try:
            condition = getInput(
tableName)
            f'GET {tableName}\nEnter condition (SQL) or Leave empty:',

            data = self.model.showtable(tableName, condition)
            self.view.print(data)
            pressEnter()
            self.show_entity_menu(tableName)
        except Exception as err:
            self.show_entity_menu(tableName, str(err))

    def insert(self, tableName):
        try:
            columns, values = getInsertInput(
                f"INSERT {tableName}\nEnter columns divided with commas, then do the
same for values in format: ['value1', 'value2', ...]", tableName)

            self.model.insert(tableName, columns, values)
            self.show_entity_menu(tableName, 'Inserted successfully')
        except Exception as err:
            self.show_entity_menu(tableName, str(err))

    def delete(self, tableName):
        try:
            condition = getInput(
                f'DELETE {tableName}\nEnter condition (SQL):', tableName)

```

```

        self.model.delete(tableName, condition)
        self.show_entity_menu(tableName, 'Deleted successfully')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def update(self, tableName):
    try:
        condition = getInput(
            f'UPDATE {tableName}\nEnter condition (SQL):', tableName)
        statement = getInput(
            "Enter SQL statement in format [<key>='<value>']", tableName)

        self.model.update(tableName, condition, statement)
        self.show_entity_menu(tableName, 'Updated successfully')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def get_accounts_and_games(self):
    try:
        print(f"GET accounts and games\nEnter date of creation range. First
date (like: '2020-01-01'), "
            f" then second (like: '2020-10-10') or Leave empty")
        created = input()
        if created:
            created2 = input()
            print(f"Enter game name (like: 'CS:GO') or Leave empty")
            game_name = input()
            print(f"Enter game hours range. First number (like: '1200'), "
                f" then second (like: '2000') or Leave empty")
            hours_played = input()
            if hours_played:
                hours_played2 = input()

            condition = ''
            flag = 0
            if created:
                condition = f" WHERE NOT (created > '{created2}' OR created <
'{created}')"
                flag = 1
            if game_name:
                if flag == 1:
                    condition += f" AND game_name='{game_name}'"
                else:
                    condition = f" WHERE game_name='{game_name}'"
                    flag = 1
            if hours_played:
                if flag == 1:
                    condition += f" AND NOT (hours_played > '{hours_played2}' OR
hours_played < '{hours_played}')"
                else:
                    condition = f" WHERE NOT (hours_played > '{hours_played2}'
OR hours_played < '{hours_played}')"

            data = self.model.get_accounts_and_games(condition)
            self.view.print_2(data)
            pressEnter()
            self.show_main_menu()
    except Exception as err:

```

```

        self.show_main_menu(str(err))

    def get_games_and_items(self):
        try:
            print(f"GET games and items\nEnter game name (Like: 'CS:GO') or  
Leave empty")
            game_name = input()
            print(f"Enter game hours range. First number (Like: '1200'), "  
f" then second (Like: '2000') or Leave empty")
            hours_played = input()
            if hours_played:
                hours_played2 = input()
            print(f"Enter item price. First number (like: '10'), "  
f" then second (Like: '1000') or Leave empty")
            price = input()
            if price:
                price2 = input()

            condition = ''
            flag = 0
            if game_name:
                condition = f" WHERE game_name='{game_name}'"
                flag = 1
            if hours_played:
                if flag == 1:
                    condition += f" AND NOT (hours_played > '{hours_played2}' OR  
hours_played < '{hours_played2}')"
                else:
                    condition = f" WHERE NOT (hours_played > '{hours_played2}'  
OR hours_played < '{hours_played2}')"
                    flag = 1
            if price:
                if flag == 1:
                    condition += f" AND NOT (price > '{price2}' OR price <  
'{price2}')"
                else:
                    condition = f" WHERE NOT (price > '{price2}' OR price <  
'{price2}')"
            data = self.model.get_games_and_items(condition)
            self.view.print_2(data)
            pressEnter()
            self.show_main_menu()
        except Exception as err:
            self.show_main_menu(str(err))

    def get_accounts_games_and_items(self):
        try:
            print(f"GET accounts, games and items\nEnter date of creation range.  
First date (Like: '2020-01-01'),"  
f" then second (Like: '2020-10-10') or Leave empty")
            created = input()
            if created:
                created2 = input()
            print(f"Enter game name (Like: 'CS:GO') or Leave empty")
            game_name = input()
            print(f"Enter game hours range. First number (Like: '1200'), "  
f" then second (Like: '2000') or Leave empty")
            hours_played = input()

```

```

        if hours_played:
            hours_played2 = input()
            print(f"Enter item price. First number (like: '10'), "
                  f" then second (like: '1000') or leave empty")
            price = input()
            if price:
                price2 = input()

        condition = ''
        flag = 0
        if created:
            if flag == 1:
                condition += f" AND NOT (created > '{created2}' OR created <
'{created}')"
            else:
                condition = f" WHERE NOT (created > '{created2}' OR created
< '{created}')"
                flag = 1
        if game_name:
            condition = f" WHERE game_name='{game_name}'"
            flag = 1
        if hours_played:
            if flag == 1:
                condition += f" AND NOT (hours_played > '{hours_played2}' OR
hours_played < '{hours_played}')"
            else:
                condition = f" WHERE NOT (hours_played > '{hours_played2}'
OR hours_played < '{hours_played}')"
                flag = 1
        if price:
            if flag == 1:
                condition += f" AND NOT (price > '{price2}' OR price <
'{price}')"
            else:
                condition = f" WHERE NOT (price > '{price2}' OR price <
'{price}')"

        data = self.model.get_accounts_games_and_items(condition)
        self.view.print_3(data)
        pressEnter()
        self.show_main_menu()
    except Exception as err:
        self.show_main_menu(str(err))

def fillByRandom(self):
    try:
        self.model.fill_account_with_random_data()
        self.show_main_menu('Generated successfully')

    except Exception as err:
        self.show_main_menu(str(err))

```