



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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

Лабораторна робота №2

з дисципліни
«Бази даних і засоби управління»

Тема: «Створення додатку бази даних,
орієнтованого на взаємодію з СУБД
PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-04

Пригоцький А.П.

Перевірив:

Київ – 2023

Загальне завдання роботи полягає в такому:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, 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).

Деталізоване завдання:

1. Забезпечити можливість уведення/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.
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!**

Приклад генерації 100 псевдовипадкових чисел:

```
select trunc(random()*1000)::int
from generate_series(1,100)
```

	trunc integer	
1	368	
2	773	
3	29	
4	66	
5	497	
6	956	

Приклад генерації 5 псевдовипадкових рядків:

```
select chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int)
from generate_series(1,5)
```

	?column? text	
1	NE	
2	MQ	
3	RN	
4	DW	
5	DA	

Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](#).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності foreign key).

- 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.

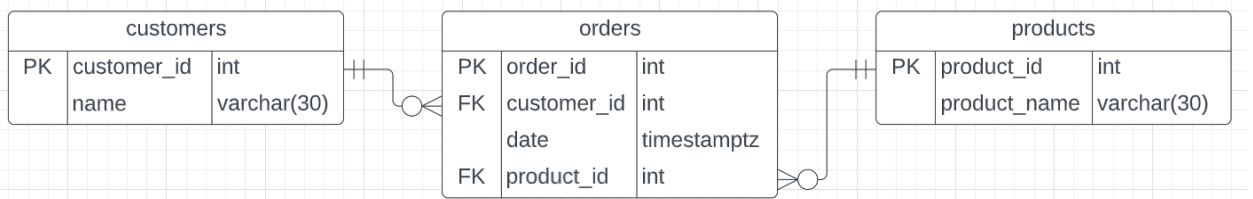
4. Програмний код організувати згідно шаблону Model-View-Controller MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](#). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2:
<http://initd.org/psycopg/docs/usage.html>)

Код БД:

```
CREATE TABLE "products" (  
  "product_id" SERIAL,  
  "product_name" varchar(30) NOT NULL,  
  PRIMARY KEY ("product_id")  
);  
  
CREATE TABLE "customers" (  
  "customer_id" SERIAL,  
  "name" varchar(30) NOT NULL,  
  PRIMARY KEY ("customer_id")  
);  
  
CREATE TABLE "orders" (  
  "order_id" SERIAL,  
  "customer_id" int NOT NULL,  
  "date" timestamptz default now(),  
  "product_id" int NOT NULL,  
  PRIMARY KEY ("order_id"),  
  CONSTRAINT "FK_orders.customer_id"  
    FOREIGN KEY ("customer_id")  
      REFERENCES "customers"("customer_id"),  
  CONSTRAINT "FK_orders.product_id"  
    FOREIGN KEY ("product_id")  
      REFERENCES "products"("product_id")  
);
```

Схема БД:



Назва мови програмування:

Python

Назви використаних бібліотек:

```
sys  
time  
psycopg2
```

Відповідь на вимоги до пункту №1 деталізованого завдання:

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

```
Update press 1
Add press 2
Delete press 3
Random press 4
Search press 5
EXIT press 0

Enter command : 69
ERROR: You have to enter the number from 0 to 5
```

Ілюстрації валідації даних при введенні користувачем:

```
Update press 1
Add press 2
Delete press 3
Random press 4
Search press 5
EXIT press 0









Enter command : 1
Your table name: customers , orders , products
Enter table name customers
Enter column name wrong_attribute

Error code - 42703
WARNING: Error column "wrong_attribute" does not exist
LINE 1: SELECT wrong_attribute FROM customers
          ^
```

Додавання нової інформації:

```
Update press 1
Add press 2
Delete press 3
Random press 4
Search press 5
EXIT press 0

Enter command : 2
Your table name: customers , orders , products
Enter table name: customers
Enter customer name:
Anton
```

Data Output			Messages	Notifications
       				
	customer_id [PK] integer	name character varying (30)		
12	12	JW		
13	13	Anton		

Вимоги до пункту №2 деталізованого завдання:
Меню генерації:

```
Enter command :  
Your table name: customers , orders , products  
Enter table name :  
Enter value:  
INSERT INTO orders (customer_id, product_id) SELECT ROUND((RANDOM()*((SELECT COUNT(*) FROM customers)-1))+1), ROUND((RANDOM()*((SELECT COUNT(*) FROM products)-1))+1) FROM generate_series(1,9);
```

Копії екрану з фрагментами згенерованих даних таблиць:

	order_id [PK] integer	customer_id integer	date timestamp with time zone	product_id integer
25	26	3	2023-01-02 19:26:22.347846+02	4
26	27	11	2023-01-02 19:26:22.347846+02	1
27	28	9	2023-01-02 19:26:22.347846+02	4
28	29	9	2023-01-02 19:26:22.347846+02	1
29	30	11	2023-01-02 19:26:22.347846+02	2
30	31	5	2023-01-02 19:30:50.241128+02	2
31	32	10	2023-01-02 19:30:50.241128+02	1
32	33	10	2023-01-02 19:30:50.241128+02	5
33	34	3	2023-01-02 19:30:50.241128+02	5
34	35	8	2023-01-02 19:30:50.241128+02	4
35	36	11	2023-01-02 19:30:50.241128+02	4
36	37	4	2023-01-02 19:30:50.241128+02	2
37	38	4	2023-01-02 19:30:50.241128+02	4
38	39	12	2023-01-02 19:30:50.241128+02	3

Копії SQLзапитів, що ілюструють генерацію при визначених
вхідних параметрах:

```
Enter command :  
Your table name: customers , orders , products  
Enter table name :  
Enter value:  
INSERT INTO orders (customer_id, product_id) SELECT ROUND((RANDOM()*((SELECT COUNT(*) FROM customers)-1))+1), ROUND((RANDOM()*((SELECT COUNT(*) FROM products)-1))+1) FROM generate_series(1,9);
```


Вимоги до пункту №3 деталізованого завдання:

Ілюстрації введення пошукового запиту та результатів виконання запитів:

```
Enter command : 1
Input number of attributes to search by >>> 1
Input attribute #1 >>> product_id
Input attribute #2 >>> date
['product_id', 'date']

Executed query: SELECT t.table_name FROM information_schema.tables t INNER JOIN information_schema.columns c ON c.table_name = t.table_name WHERE c.column_name LIKE 'product_id' AND t.table_schema NOT IN
('information_schema', 'pg_catalog') INTERSECT SELECT t.table_name FROM information_schema.tables t INNER JOIN information_schema.columns c ON c.table_name = t.table_name WHERE c.column_name LIKE 'date' AND
t.table_schema NOT IN ('information_schema', 'pg_catalog')

Enter left boundary for product_id: 2
Enter right boundary for product_id: 10

Enter left boundary for date: 2012-02-03
Enter right boundary for date: 2023-03-01

Executed query: SELECT * FROM orders WHERE product_id > 2 AND product_id < 10 AND date > '2012-02-03' AND date < '2023-03-01'

-----
22 rows with specified attributes have been found:
(1, 11, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 3)
(2, 9, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(3, 7, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 3)
(5, 2, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 5)
(6, 3, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(7, 6, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(10, 3, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 3)
(11, 1, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(12, 6, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(15, 11, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(16, 9, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(17, 12, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(20, 10, datetime.datetime(2022, 12, 27, 2, 9, 34, 833987, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 3)
(25, 5, datetime.datetime(2023, 1, 2, 19, 26, 22, 347846, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(26, 3, datetime.datetime(2023, 1, 2, 19, 26, 22, 347846, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(38, 4, datetime.datetime(2023, 1, 2, 19, 30, 50, 241128, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 4)
(39, 12, datetime.datetime(2023, 1, 2, 19, 30, 50, 241128, tzinfo=datetime.timezone(datetime.timedelta(seconds=7200))), 3)
-----
Time:0.001001596450805664 seconds
```

Копії SQL-запитів, що ілюструють генерацію при визначених запитів,

що ілюструють пошук з зазначеними початковими параметрами

```
Executed query: SELECT t.table_name FROM information_schema.tables t INNER JOIN information_schema.columns c ON c.table_name = t.table_name WHERE c.column_name LIKE 'product_id' AND t.table_schema NOT IN
('information_schema', 'pg_catalog') INTERSECT SELECT t.table_name FROM information_schema.tables t INNER JOIN information_schema.columns c ON c.table_name = t.table_name WHERE c.column_name LIKE 'date' AND
t.table_schema NOT IN ('information_schema', 'pg_catalog')
```

Вимоги до пункту №4 деталізованого завдання:

Ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Всі функції роблять те, що вказано у їх назві.

Model code:

```
import sys
import time
import psycopg2

from view import *

def db_connect():
    return psycopg2.connect(
        database="KPI_DB",
        user="postgres",
        password="",
```

```

        host="localhost",
        port="5432"
    )

def db_error(err):
    print(f"\nError code - {err.pgcode}")
    print(f'WARNING: Error {err}')
    sys.exit(-1)

def table_nf(table_name):
    print(f"ERROR: Table {table_name} was not found in the database")
    sys.exit(-1)

def select_column(table_name, column_name):
    con = db_connect()
    cursor = con.cursor()
    try:
        cursor.execute(f"SELECT {column_name} FROM {table_name}")
        print(f"SELECT {column_name} FROM {table_name}")
        for row in cursor.fetchall():
            print(row)
    except psycopg2.Error as err:
        db_error(err)

    cursor.close()
    con.close()

def random(table_name, n):
    con = db_connect()
    con.set_session(autocommit=True)
    cursor = con.cursor()
    if table_name == 'customers':
        try:
            query = "INSERT INTO customers (name) " \
                    "SELECT chr(trunc(65+random()*25)::int) || chr(trunc(65"
+ random()*25)::int) " \
                    f"FROM generate_series(1,{n});"
            cursor.execute(query)
            print(query)
        except psycopg2.Error as err:
            db_error(err)

    elif table_name == 'products':
        try:
            query = "INSERT INTO products (product_name) " \
                    "SELECT chr(trunc(65+random()*25)::int) || " \
                    "chr(trunc(65 + random()*25)::int) ||"
+ chr(trunc(65 + random()*25)::int) " \
                    f"FROM generate_series(1,{n});"
            cursor.execute(query)
            print(query)
        except psycopg2.Error as err:
            db_error(err)

    elif table_name == 'orders':
        try:
            query = "INSERT INTO orders (customer_id, product_id) " \
                    "SELECT ROUND((RANDOM() * ((SELECT COUNT(*) FROM"
+ customers)-1))+1), " \

```

```

        "          ROUND((RANDOM() * ((SELECT COUNT(*) FROM products) -
1)) + 1) " \
        f"FROM generate_series(1, {n});"
    cursor.execute(query)
    print(query)
    except psycopg2.Error as err:
        db_error(err)

    else:
        table_nf(table_name)

    cursor.close()
    con.close()

def delete(table_name, column, row):
    con = db_connect()

    con.set_session(autocommit=True)
    cursor = con.cursor()

    if table_name == 'customers':
        try:
            cursor.execute(f"DELETE FROM orders WHERE customer_id = {row};"
                           f"DELETE FROM customers WHERE customer_id =
{row};")
        except psycopg2.Error as err:
            db_error(err)
    elif table_name == 'products':
        try:
            cursor.execute(f"DELETE FROM orders WHERE product_id = {row};"
                           f"DELETE FROM products WHERE product_id = {row};")
        except psycopg2.Error as err:
            db_error(err)
    elif table_name == 'orders':
        try:
            cursor.execute(f"DELETE FROM orders WHERE {column} = {row};")
        except psycopg2.Error as err:
            db_error(err)
    else:
        table_nf(table_name)

    cursor.close()
    con.close()

def add_inf(table_name):
    con = db_connect()
    con.set_session(autocommit=True)
    cursor = con.cursor()

    if table_name == 'customers':
        print("Enter customer name:")
        customer_name = input()
        try:
            cursor.execute(f"INSERT INTO customers (name) VALUES
('{customer_name}')"")
        except psycopg2.Error as err:
            db_error(err)
    elif table_name == 'products':
        print("Enter product name:")
        product_name = input()
        try:

```

```

        cursor.execute(f"INSERT INTO products (product_name) VALUES
('{product_name}')" )
    except psycopg2.Error as err:
        db_error(err)
    elif table_name == 'orders':
        print("Enter customer_id and product_id")
        customer_id, product_id = input(), input()
        try:
            cursor.execute(f"INSERT INTO orders (customer_id, product_id) "
                           f"VALUES ({customer_id},{product_id})")
        except psycopg2.Error as err:
            db_error(err)
    else:
        table_nf(table_name)

    cursor.close()
    con.close()

def update_two(table1, table2, column, old_value, new_value):
    con = db_connect()
    con.set_session.autocommit=True
    cursor = con.cursor()
    try:
        cursor.execute(
            f"UPDATE {table1} SET {column} = {new_value} WHERE {column} =
{old_value})"
            f"UPDATE {table2} SET {column} = {new_value} WHERE {column} =
{old_value}")
    except psycopg2.Error as err:
        db_error(err)
    cursor.close()
    con.close()

def update_one(table, column, old_name, new_name):
    con = db_connect()
    con.set_session.autocommit=True
    cursor = con.cursor()
    try:
        cursor.execute(f"UPDATE {table} SET {column} = {new_name} WHERE
{column} = {old_name}")
    except psycopg2.Error as err:
        db_error(err)
    cursor.close()
    con.close()

def update(table_name, column_name):
    con = db_connect()
    con.set_session.autocommit=True
    cursor = con.cursor()
    select_column(table_name, column_name)
    if table_name in ('products', 'customers') and column_name in
('customer_id', 'product_id'):
        try:
            update_two('orders', table_name, column_name, get_old_data(),
get_new_data())
        except psycopg2.Error as err:
            db_error(err)
    elif table_name in ('products', 'customers', 'orders'):
        try:
            update_one(table_name, column_name, get_old_data(),
get_new_data())

```

```

        except psycopg2.Error as err:
            db_error(err)
    else:
        table_nf(table_name)

    cursor.close()
    con.close()

def search():
    con = db_connect()
    con.set_session(autocommit=True)
    cursor = con.cursor()

    n = int(input("Input number of attributes to search by >>> "))
    if n not in (1, 2, 3):
        print("Error, wrong number of attributes (1 or 2 or 3). You just don't need more for this database")
        exit(-1)

    attributes = [str(input(f"Input attribute №{h + 1} >>> ")) for h in range(n)]
    print(attributes)

    std_query = f"SELECT t.table_name " \
                f"FROM information_schema.tables t " \
                f"INNER JOIN information_schema.columns c ON c.table_name = " \
                f"t.table_name " \
                f"WHERE c.column_name LIKE 'attribute' " \
                f"AND t.table_schema NOT IN ('information_schema', 'pg_catalog') "

    get_table_query = ' INTERSECT '.join(std_query.replace('attribute', attribute) for attribute in attributes)

    print("\n Executed query:", get_table_query, '\n')

    cursor.execute(get_table_query)

    tables = [table_name[0] for table_name in cursor.fetchall()]

    types = []
    for attribute in attributes:
        cursor.execute(
            f"SELECT DISTINCT data_type FROM INFORMATION_SCHEMA.COLUMNS WHERE "
            f"column_name = '{attribute}' and table_schema = 'public'"
        )
        types += [list(cursor.fetchall()[0])]

    attributes = dict(zip(attributes, types))

    for attribute in attributes:
        print()
        if attributes[attribute][0] == "character varying":
            attributes[attribute] += [input(f"Enter value for {attribute}: ")]
        elif attributes[attribute][0] in ("integer", "timestamp with time zone"):
            attributes[attribute] += [input(f"Enter left boundary for {attribute}: ")]
            attributes[attribute] += [input(f"Enter right boundary for {attribute}: ")]
        else:
            print("Attribute type error")
            exit(-1)

```

```

start_time = time.time()

results = []
for table in tables:
    std_query = f"SELECT * " \
                f"FROM {table} " \
                f"WHERE "

    tmp = []

    for attribute in attributes:
        if attributes[attribute][0] == 'integer':
            tmp += [f"{attribute} > {attributes[attribute][-2]} AND "
                    f"{attribute} < {attributes[attribute][-1]} "]
        elif attributes[attribute][0] == "timestamp with time zone":
            tmp += [f"{attribute} > '{attributes[attribute][-2]}' AND "
                    f"{attribute} < '{attributes[attribute][-1]}' "]
        elif attributes[attribute][0] == "character varying":
            tmp += [f"{attribute} LIKE '{attributes[attribute][-1]}' "]

    cursor.execute(std_query + ' AND '.join(tmp))
    print("\n Executed query:", std_query + ' AND '.join(tmp), '\n')
    results += cursor.fetchall()

print('-' * 10)
print(f"{len(results)} rows with specified attributes have been found:")
print(*results, sep='\n')
print('-' * 10)

print("Time:%s seconds" % (time.time() - start_time))
cursor.close()
con.close()

```

View code:

```

import sys

def command_error():
    print("ERROR: You have to enter the number from 0 to 5")

def command_identification():
    return input('Enter command : ')

def table():
    print('Your table name: customers , orders , products')
    return input('Enter table name ')

def column():
    return input('Enter column name ')

def get_old_data():
    return input('Enter old value ')

def get_new_data():
    return input('Enter new value ')

```

```

def get_data():
    return input('Enter value: ')

def row():
    return int(input('Enter value: '))

def table_invalid():
    print('The table name is wrong ERROR')
    sys.exit()

def exiting():
    print('Exiting')
    sys.exit()

def menu():
    print()
    print("Update press 1")
    print("Add press 2")
    print("Delete press 3")
    print("Random press 4")
    print("Search press 5")
    print('EXIT press 0')
    print()

```

Control_func code:

```

from model import *
from view import *

def request():
    input_command = command_identification()

    if input_command == '1':
        table_name = table()
        column_name = column()
        update(table_name, column_name)
    elif input_command == '2':
        table_name = table()
        add_inf(table_name)
    elif input_command == '3':
        table_name = table()
        column_name = column()
        delete(table_name, column_name, get_data())
    elif input_command == '4':
        table_name = table()
        random(table_name, get_data())
    elif input_command == '5':
        search()
    elif input_command == '0':
        exiting()
    else:
        command_error()
        main()

def main():
    menu()
    request()

```

```
if __name__ == '__main__':  
    main()
```

GitHub repo:

<https://github.com/antohka151/DBMT-KPI>