

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 ІІI курсу ФПМ групи КВ-04

Пригоцький А.П.

Перевірив:

Київ – 2023

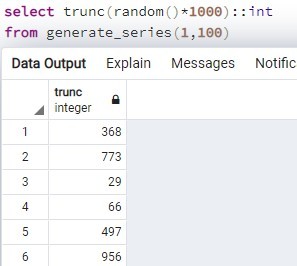
*Загальне завдання* роботи полягає в такому:

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 підлеглій 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:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061) [за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

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).

1. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення 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.

1. Програмний код організувати згідно шаблону Model-View- Controller MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). 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"),

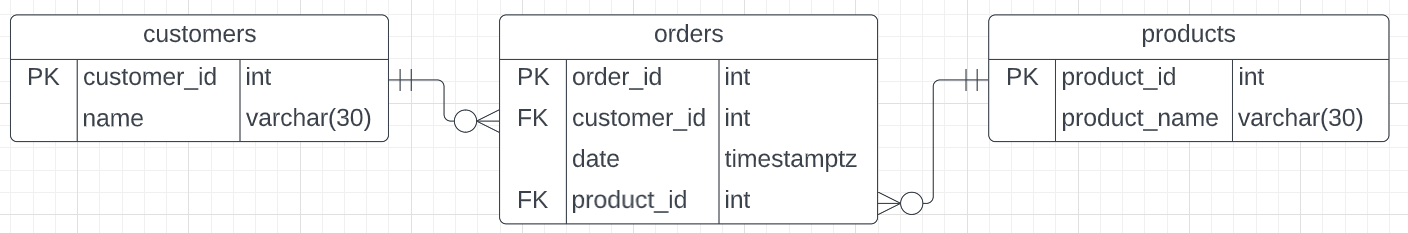
CONSTRAINT "FK\_orders.product\_id"

FOREIGN KEY ("product\_id")

REFERENCES "products"("product\_id")

);

**Схема БД:**



# Назва мови програмування:

# *Python*

# Назви використаних бібліотек:

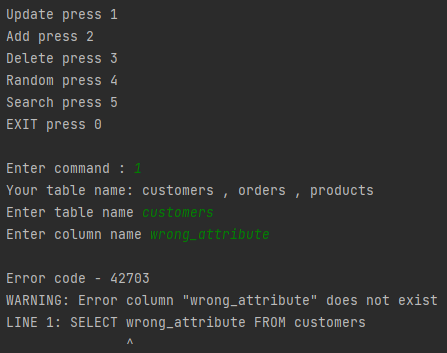
sys  
time  
psycopg2

# Відповідь на вимоги до пункту №1 деталізованого завдання:

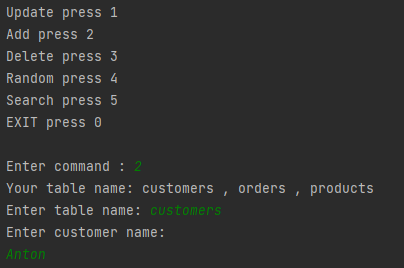
# Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних:

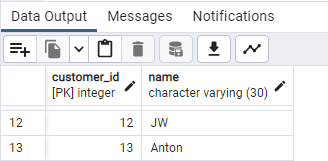
# 

Ілюстрації валідації даних при уведенні користувачем:



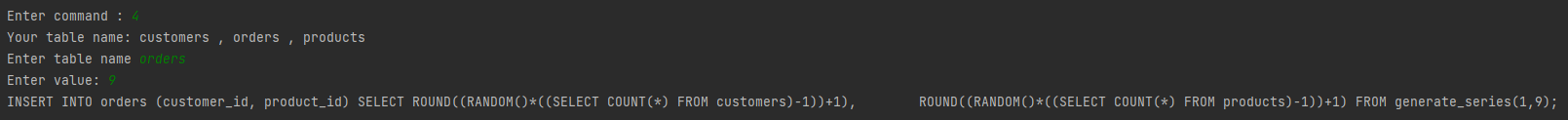
Додавання нової інформації:



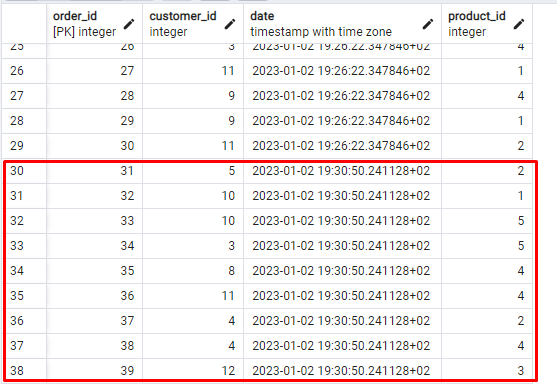


# Вимоги до пункту №2 деталізованого завдання:

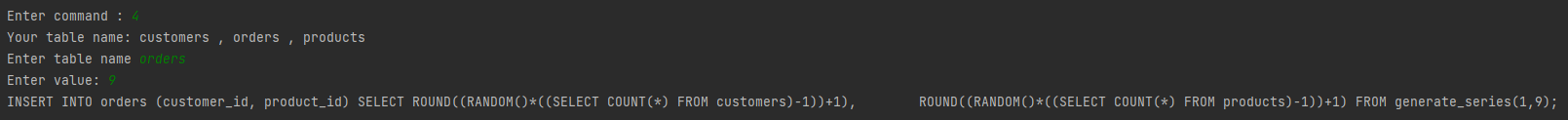
Меню генерації:



Копії екрану з фрагментами згенерованих даних таблиць:

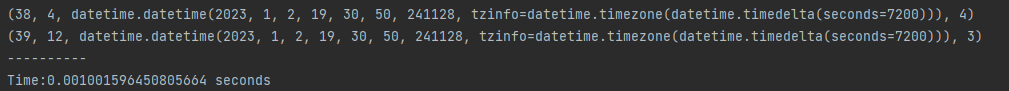
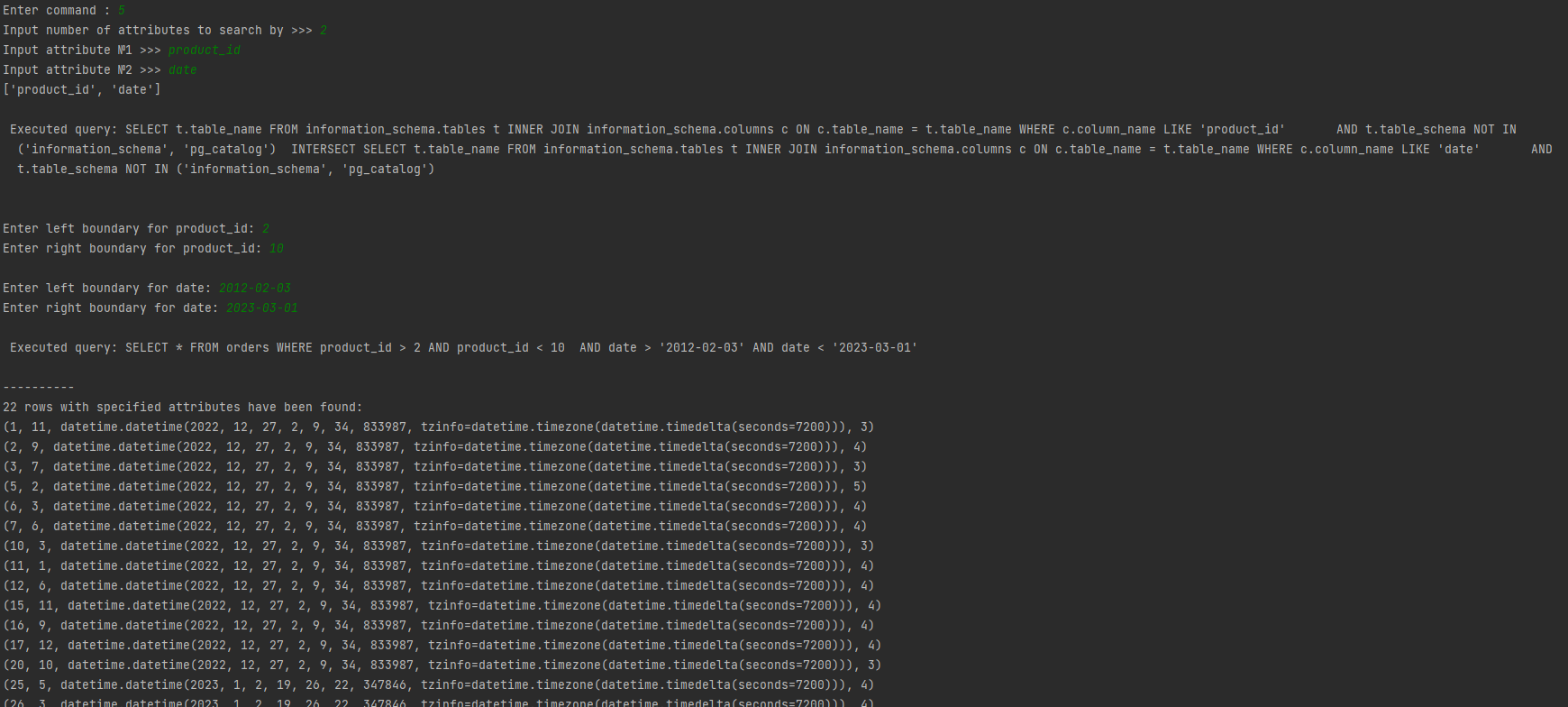


Копії SQL запитів, що ілюструють генерацію при визначених вхідних параметрах:

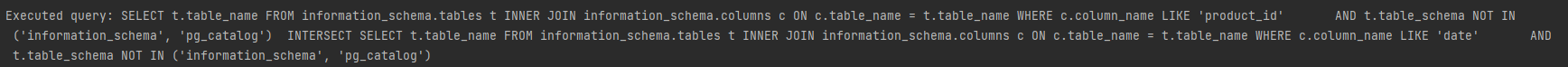


# Вимоги до пункту №3 деталізованого завдання:

Ілюстрації уведення пошукового запиту та результатів виконання запитів:



Копії SQL-запитів, що ілюструють генерацію при визначених запитів, що ілюструють пошук з зазначеними початковими параметрами



# Вимоги до пункту №4 деталізованого завдання:

Ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Всі функції роблять те, що в казано 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 = 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 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 {attribute} < {attributes[attribute][-1]} "]

**elif** attributes[attribute][0] == "timestamp with time zone":

tmp += [f"{attribute} > '{attributes[attribute][-2]}' AND {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