



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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Лабораторна робота №3**

з дисципліни  
**«Бази даних і засоби управління»**

**Тема:**

**«Засоби оптимізації роботи СУБД PostgreSQL»**

Виконав: студент III курсу

ФПМ групи КВ-04

Пригоцький А.П.

Перевірив:

Київ – 2023

### **Лабораторна робота №3**

*Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.*

*Завдання роботи полягає у наступному:*

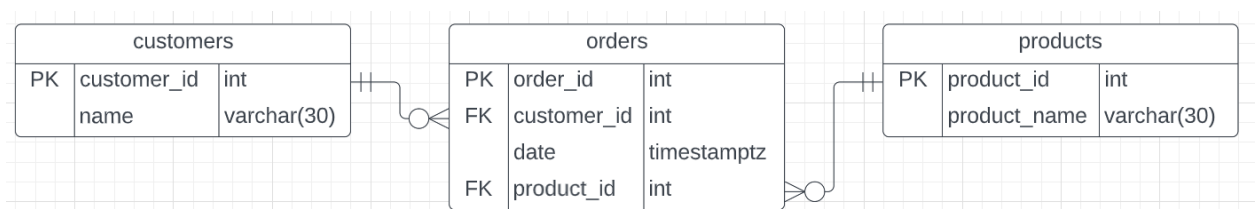
1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

## Завдання № 1

### 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" timestampz 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:



### Перетворення функцій:

Виклики запитів замінені засобами SQLAlchemy по роботі з об'єктами. Реалізована вставка, вилучення та редагування екземплярів класів-сутностей.

## Демонстрація роботи:

Скріншоти будуть містити меню тільки коли виконується нова команда (так воно виводиться після кожної команди).

```
SELECT press 1
ADD press 2
DELETE press 3
UPDATE press 4
EXIT press 0

Enter command : 1
Your table names: customers, orders, products
Enter table name: products
[<Customer(product_id='1', product_name='Cola')>, <Customer(product_id='2', product_name='Fanta')>]
```

```
Enter command : 1
Your table names: customers, orders, products
Enter table name: orders
[]
```

```
Enter command : 1
Your table names: customers, orders, products
Enter table name: customers
[<Customer(customer_id='1', name='Anton')>, <Customer(customer_id='2', name='Nikolas')>]
```

```
SELECT press 1
ADD press 2
DELETE press 3
UPDATE press 4
EXIT press 0
```

```
Enter command : 2
Your table names: customers, orders, products
Enter table name: orders
Please, enter customer id:
1
Please, enter product id:
2
```

```
Enter command : 2
Your table names: customers, orders, products
Enter table name: orders
Please, enter customer id:
2
Please, enter product id:
1
```

```
Enter command : 1
Your table names: customers, orders, products
Enter table name: orders
[<Order(order_id='1', customer_id='1', date='2023-01-14 21:47:00.720472+02:00', product_id='2')>, <Order(order_id='2', customer_id='2', date='2023-01-14 21:47:09.999274+02:00', product_id='1')>]
```

```
SELECT press 1
ADD press 2
DELETE press 3
UPDATE press 4
EXIT press 0
```

Enter command : 3

Your table names: customers, orders, products

Enter table name: *orders*

Please, choose column to filter with

There are 3 columns {customer\_id, order\_id, product\_id}

Please, choose 1 column:

*order\_id*

Please, enter column value: 2

Enter command : 1

Your table names: customers, orders, products

Enter table name: *orders*

[<Order(order\_id='1', customer\_id='1', date='2023-01-14 21:47:00.720472+02:00', product\_id='2')>]

```
SELECT press 1
```

```
ADD press 2
```

```
DELETE press 3
```

```
UPDATE press 4
```

```
EXIT press 0
```

Enter command : 4

Your table names: customers, orders, products

Enter table name: *orders*

Please, choose column to filter with

There are 3 columns {customer\_id, order\_id, product\_id}

Please, choose 1 column:

*order\_id*

Please, enter column value: 1

Please, choose column to update

There are 3 columns {customer\_id, order\_id, product\_id}

Please, choose 1 column:

*product\_id*

Please, enter column value: 1

Enter command : 1

Your table names: customers, orders, products

Enter table name: *orders*

[<Order(order\_id='1', customer\_id='1', date='2023-01-14 21:47:00.720472+02:00', product\_id='1')>]

```

1 SELECT *
2 FROM orders
3 LEFT JOIN customers
4 USING (customer_id)
5 LEFT JOIN products
6 USING (product_id);

```

Data Output Messages Notifications

	product_id integer	customer_id integer	order_id integer	date timestamp with time zone	name character varying	product_name character varying
1	1	1	1	2023-01-14 21:47:00.720472+02	Anton	Cola

### Демонстрація опрацювання помилок:

```

SELECT press 1
ADD press 2
DELETE press 3
UPDATE press 4
EXIT press 0

Enter command : 8
ERROR: You have to enter the number from 0 to 4

Enter command : 1
Your table names: customers, orders, products
Enter table name: 100
The table name is wrong ERROR

SELECT press 1
ADD press 2
DELETE press 3
UPDATE press 4
EXIT press 0

Enter command : 4
Your table names: customers, orders, products
Enter table name: customers
Please, choose column to filter with
There are 2 columns {customer_id, name}
Please, choose 1 column:
wrong
The column name is wrong ERROR

```

## **Код програми:**

Структура кода була змінена, через те що створення класів SQLAlchemy по всім стандартам має відбуватися в окремому файлі (це дуже сильно збільшує читабельність коду).

- control\_func – без змін.
- view – без змін.
- crud – файл models тепер називається саме так, щоб не було плутанини з наступним файлом.
- model – реалізація класів-таблиць БД.

Функції які реалізують запити до БД були перероблені майже повністю (код функцій), через то що використання SQLAlchemy достатньо сильно змінює реалізацію однієї й тієї ж задачі, при цьому, інтерфейси функцій залишилися без змін.

## control\_func.py

```
from crud import *
from view import *

def request():
    input_command = command_identification()

    # Select function.
    if input_command == '1':
        table_name = table()
        select(table_name)

    # Add function.
    elif input_command == '2':
        table_name = table()
        add(values_to_add(table_name))

    # Delete function.
    elif input_command == '3':
        table_name = table()
        filter_column, filter_value = filter_column_data(table_name)
        delete(table_name, filter_column, filter_value)

    # Update function.
    elif input_command == '4':
        table_name = table()
        filter_column, filter_value = filter_column_data(table_name)
        update_column, update_value = column_to_update_data(table_name)
        update(table_name, filter_column, filter_value, update_column,
update_value)

    # Exit from menu.
    elif input_command == '0':
        exiting()

    else:
        command_error()
        main()

    main()

def main():
    menu()
    request()

if __name__ == '__main__':
    # Uncomment to create new database.
    # recreate_database()
    main()
```



## view.py

```
import sys

from models import *

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

def column_invalid():
    print("The column name is wrong ERROR")
    sys.exit()

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

def command_error():
    print("ERROR: You have to enter the number from 0 to 4")

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

def table():
    print('Your table names: customers, orders, products')
    table_name = input('Enter table name: ')

    tables = {'customers': Customer, 'orders': Order, 'products': Product}
    if table_name not in tables.keys():
        return table_invalid()
    return tables[table_name]

def values_to_add(table_name):
    if table_name == Order:
        print("Please, enter customer id:")
        customer_id = input()
        print("Please, enter product id:")
        product_id = input()

        if not customer_id.isdigit() or not product_id.isdigit():
            print("ERROR: Id should be a number")
            sys.exit()

        return Order(int(customer_id), int(product_id))
    else:
        print(f"Please, enter {'product_name' if table_name == Product else 'name'}")
        return table_name(input())

def column(table_name):
    if table_name == Order:
        return Order.order_id
    elif table_name == Customer:
        return Customer.customer_id
    else:
        return Product.product_id
```

```

def get_column(table_name):
    if table_name == Order:
        print('There are 3 columns {customer_id, order_id, product_id}')
        print('Please, choose 1 column:')
        column = input()
        columns = {'customer_id': Order.customer_id, 'order_id':
Order.order_id, 'product_id': Order.product_id}
        if column not in columns.keys():
            return column_invalid()
        return columns[column]
    elif table_name == Customer:
        print('There are 2 columns {customer_id, name}')
        print('Please, choose 1 column:')
        column = input()
        columns = {'customer_id': Customer.customer_id, 'name': Customer.name}
        if column not in columns.keys():
            return column_invalid()
        return columns[column]
    else:
        print('There are 2 columns {product_id, product_name}')
        print('Please, choose 1 column:')
        column = input()
        columns = {'product_id': Product.product_id, 'product_name':
Product.product_name}
        if column not in columns.keys():
            return column_invalid()
        return columns[column]

def get_data():
    return input('Please, enter column value: ')

def filter_column_data(table_name):
    print("Please, choose column to filter with")
    filter_column = get_column(table_name)
    filter_value = get_data()
    return filter_column, filter_value

def column_to_update_data(table_name):
    print("Please, choose column to update")
    update_column = get_column(table_name)
    update_value = get_data()
    return update_column, update_value

def menu():
    print()
    print("SELECT press 1")
    print("ADD press 2")
    print("DELETE press 3")
    print("UPDATE press 4")
    print('EXIT press 0')
    print()

```

## models.py

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey

Base = declarative_base()

class Customer(Base):
    __tablename__ = 'customers'

    customer_id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)

    orders = relationship("Order", cascade="all,delete", passive_deletes=True)

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return "<Customer(customer_id='{}', name='{}')>" \
            .format(self.customer_id, self.name)

class Order(Base):
    __tablename__ = 'orders'

    order_id = Column(Integer, primary_key=True)
    customer_id = Column(Integer, ForeignKey('customers.customer_id',
ondelete='CASCADE'))
    date = Column(DateTime(timezone=True), server_default=func.now())
    product_id = Column(Integer, ForeignKey('products.product_id',
ondelete='CASCADE'), nullable=False)

    def __init__(self, customer_id, product_id):
        self.customer_id = customer_id
        self.product_id = product_id

    def __repr__(self):
        return "<Order(order_id='{}', customer_id='{}', date='{}',
product_id='{}')>" \
            .format(self.order_id, self.customer_id, self.date,
self.product_id)

class Product(Base):
    __tablename__ = 'products'

    product_id = Column(Integer, primary_key=True)
    product_name = Column(String, nullable=False)

    products = relationship("Order", cascade="all,delete",
passive_deletes=True)

    def __init__(self, product_name):
        self.product_name = product_name

    def __repr__(self):
        return "<Customer(product_id='{}', product_name='{}')>" \
            .format(self.product_id, self.product_name)
```

## crud.py

```
import sys

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError

from view import *

engine = create_engine('postgresql://postgres:password@localhost:5432/test')

Session = sessionmaker(bind=engine)

def db_error(err):
    print("WARNING: Error has occurred\n")
    print(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 recreate_database():
    Base.metadata.drop_all(engine)
    Base.metadata.create_all(engine)

def delete(table, column, row):
    s = Session()
    try:
        s.query(table).filter(column == row).delete()
        s.commit()
    except SQLAlchemyError as err:
        db_error(err)

    s.close()

def add(element):
    s = Session()
    try:
        s.add(element)
        s.commit()
    except SQLAlchemyError as err:
        db_error(err)

    s.close()

def select(table):
    s = Session()
    try:
        # Change to return if needed.
        print(s.query(table).all())
    except SQLAlchemyError as err:
        db_error(err)

def update(table, filter_column, filter_column_value, column_to_upd,
updated_value):
    s = Session()
    try:
```

```

s.query(table).filter(filter_column ==
filter_column_value).update({column_to_upd: updated_value})
s.commit()
except SQLAlchemyError as err:
    db_error(err)

s.close()

```

## Завдання №2

### BTree

Для дослідження індексу була створена таблиця, яка має дві колонки: числову і текстову. Вони проіндексовані як BTree. У таблицю було занесено 1000000 записів.

#### Створення таблиці:

```

DROP TABLE IF EXISTS "test_btree";

CREATE TABLE "test_btree"(
    "id" bigserial PRIMARY KEY,
    "test_text" varchar(255)
);

```

#### Заповнення таблиці:

```

INSERT INTO "test_btree"("test_text") SELECT substr(characters, (random() *
length(characters) + 1)::integer, 10) FROM
(VALUE('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as
symbols(characters), generate_series(1, 1000000) as q;

```

#### Вибір даних без індексу:

The screenshot shows a database query editor interface. The query editor contains the following SQL query:

```
1 SELECT COUNT(*) FROM "test_btree" WHERE "id" % 2 = 0;
```

The query has been executed successfully. The data output table shows the following result:

count	bigint
1	500000

Below the data output table, there are two green status messages:

- ✓ Successfully run. Total query runtime: 140 msec. 1 rows affected.
- ✓ Successfully run. Total query runtime: 162 msec. 1 rows affected.

postgres/postgres@lola

Query Editor Query History Scratch Pad

```
1 SELECT COUNT(*) FROM "test_btree" WHERE ("test_text" @@ to_tsquery('bnm'));
```

Data Output Explain Messages Notifications

	count bigint
1	19414

✓ Successfully run. Total query runtime: 1 secs 555 msec. 1 rows affected.

postgres/postgres@lola

Query Editor Query History Scratch Pad

```
1 SELECT COUNT(*), SUM("id") FROM "test_btree" WHERE "test_text" LIKE 'b%' GROUP BY "id" ORDER BY "id" ASC;
```

Data Output Explain Messages Notifications

	count bigint	sum numeric
1	9710	4853336674
2	9532	4790314864

✓ Successfully run. Total query runtime: 133 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 132 msec. 2 rows affected.

## Створюємо індекс:

```
DROP INDEX IF EXISTS "test_btree_test_text_index";
```

```
CREATE INDEX "test_btree_test_text_index" ON "test_btree" USING btree ("test_text");
```

## Вибір даних з створеним індексом:

postgres/postgres@lola

Query Editor Query History Scratch Pad

```
1 SELECT COUNT(*) FROM "test_btree" WHERE "id" % 2 = 0;
```

Data Output Explain Messages Notifications

	count bigint
1	500000

✓ Successfully run. Total query runtime: 137 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 136 msec. 1 rows affected.

postgres/postgres@lola

Query EditorQuery HistoryScratch Pad

```
1 SELECT COUNT(*) FROM "test_btree" WHERE ("test_text" @@ to_tsquery('bnm'));
2
```

Data OutputExplainMessagesNotifications

	count	bigint
1	19414	

✓ Successfully run. Total query runtime: 1 secs 525 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 1 secs 486 msec. 1 rows affected.

postgres/postgres@lola

Query EditorQuery HistoryScratch Pad

```
1 ECT COUNT(*), SUM("id") FROM "test_btree" WHERE "test_text" LIKE 'b%' GROUP BY "id" % 2;
```

Data OutputExplainMessagesNotifications

	count	bigint	sum	numeric
1	9710		4853336674	
2	9532		4790314864	

✓ Successfully run. Total query runtime: 140 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 137 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 132 msec. 2 rows affected.

## BRIN

Для дослідження індексу була створена таблиця, яка має дві колонки: t\_data типу timestamp without time zone (дата та час (без часового поясу)) і t\_number типу integer. Колонка t\_data проіндексована як BRIN. У таблицю занесено 1000000 записів.

### Створення таблиці:

```
DROP TABLE IF EXISTS "test_brin";

CREATE TABLE "test_brin"(
    "id" bigserial PRIMARY KEY,
    "test_time" timestamp
);
```

### Заповнення таблиці:

```
INSERT INTO "test_brin"("test_time")
SELECT
    (timestamp '2021-01-01' + random() * (timestamp '2020-01-01' - timestamp '2022-01-01'))
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000000) as q;
```

### Вибір даних без індексу:

The screenshot shows a PostgreSQL query editor interface. The query editor contains the following SQL query:

```
1 SELECT COUNT(*) FROM "test_brin" WHERE "id" % 2 = 0;
```

The query is executed, and the results are displayed in the Data Output tab. The results show a single row with a count of 500000.

	count bigint
1	500000

Below the Data Output tab, there are three green status messages indicating successful execution of the query:

- ✓ Successfully run. Total query runtime: 142 msec. 1 rows affected.
- ✓ Successfully run. Total query runtime: 136 msec. 1 rows affected.
- ✓ Successfully run. Total query runtime: 141 msec. 1 rows affected.



postgres/postgres@lola

Query Editor Query History Scratch Pad

```
1 SELECT COUNT(*) FROM "test_brin" WHERE "time" >= '20191001';
```

Data Output Explain Messages Notifications

	count	bigint
1	627085	

✓ Successfully run. Total query runtime: 128 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 129 msec. 1 rows affected.

postgres/postgres@lola

Query Editor Query History Scratch Pad

```
1 SELECT AVG("id") FROM "test_brin" WHERE "time" >= '20191001' AND "time" <= '20211207';
2
```

Data Output Explain Messages Notifications

	avg	numeric
1	499894.886803224443	

✓ Successfully run. Total query runtime: 166 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 175 msec. 1 rows affected.

## Створюємо індекс:

```
DROP INDEX IF EXISTS "test_brin_time_index";
CREATE INDEX "test_brin_time_index" ON "test_brin" USING brin("time")
```

# Вибір даних з створеним індексом:

postgres/postgres@lola

Query Editor

Query History

Scratch Pad

1

SELECT COUNT(\*) FROM "test\_brin" WHERE "id" % 2 = 0;

Data Output

Explain

Messages

Notifications

count

bigint

1

500000

✓ Successfully run. Total query runtime: 139 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 135 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 138 msec. 1 rows affected.

postgres/postgres@lola

Query History

Scratch Pad

SELECT COUNT(\*) FROM "test\_brin" WHERE "time" >= '20191001';

Explain

Messages

Notifications

5

✓ Successfully run. Total query runtime: 128 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 132 msec. 1 rows affected.



## Завдання №3

Розробити тригер бази даних PostgreSQL.  
Умова для тригера – before update, delete.

### Таблиці:

```
DROP TABLE IF EXISTS "reader";
CREATE TABLE "reader"(
  "readerID" bigserial PRIMARY KEY, "readerName" varchar(255)
);
```

```
DROP TABLE IF EXISTS "readerLog";
CREATE TABLE "readerLog"(
  "id" bigserial PRIMARY KEY, "readerLogID" bigint, "readerLogName" varchar(255)
);
```

### Тригер:

```
CREATE OR REPLACE FUNCTION update_delete_func() RETURNS TRIGGER as $$
```

```
DECLARE
```

```
CURSOR_LOG CURSOR FOR SELECT * FROM "readerLog";
row_Log "readerLog"%ROWTYPE;
```

```
begin
```

```
IF old."readerID" % 2 = 0 THEN
  INSERT INTO "readerLog"("readerLogID", "readerLogName") VALUES (old."readerID",
    old."readerName");
  UPDATE "readerLog" SET "readerLogName" = trim(BOTH 'x' FROM "readerLogName");
  RETURN NEW;
```

```
ELSE
```

```
  RAISE NOTICE 'readerID is odd';
  FOR row_log IN cursor_log LOOP
    UPDATE "readerLog" SET "readerLogName" = 'x' || row_Log."readerLogName" || 'x' WHERE "id" =
      row_log."id";
  END LOOP;
  RETURN NEW;
END IF;
END;
```

```
$$ language plpgsql;
```

### Ініціалізація тригера:

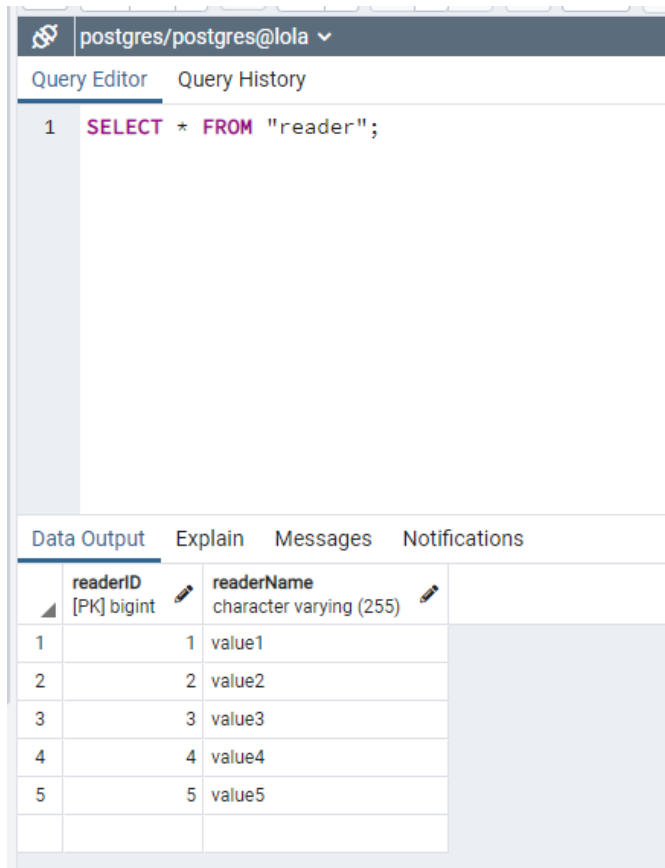
```
create trigger test_trigger before update or delete on reader for each row
execute procedure update_delete_func();
```

### Принцип тригера:

Спрацьовує коли проходить оновлення чи видалення рядка з парним номером і заноситься у таблицю Logs.

## Ініціалізуємо таблицю:

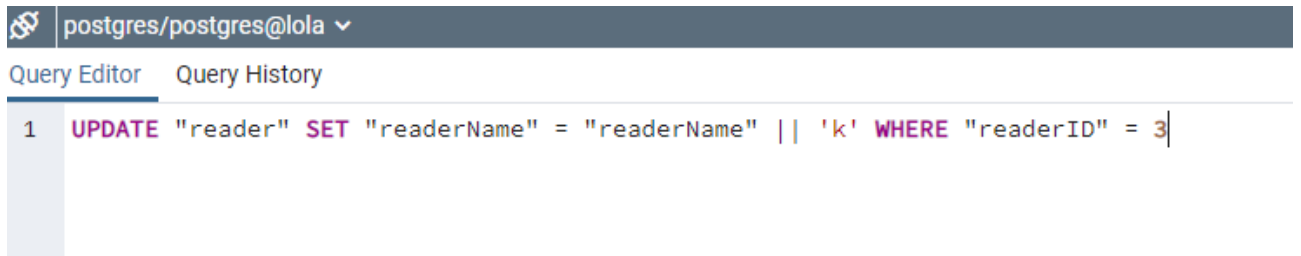
```
INSERT INTO "reader"("readerName")  
VALUES ('value1'), ('value2'), ('value3'), ('value4'), ('value5');
```



The screenshot shows a PostgreSQL query editor interface. At the top, the database connection is 'postgres/postgres@lola'. Below the connection bar, there are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying a SQL query: `1 SELECT * FROM "reader";`. Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the results of the query. The table has two columns: 'readerID [PK] bigint' and 'readerName character varying (255)'. The results are as follows:

	readerID [PK] bigint	readerName character varying (255)
1	1	value1
2	2	value2
3	3	value3
4	4	value4
5	5	value5

## Робимо запит:



The screenshot shows the same PostgreSQL query editor interface. The 'Query Editor' tab is active, displaying a SQL query: `1 UPDATE "reader" SET "readerName" = "readerName" || 'k' WHERE "readerID" = 3`. The query is partially executed, as indicated by the cursor at the end of the line.

Після виконання запиту, бачимо, що у рядок за номером 3 було додано “k” та не був доданий рядок у logs.

Data Output		Explain	Messages	Notifications
	<b>readerID</b> [PK] bigint	<b>readerName</b> character varying (255)		
1	1	value1		
2	2	value2		
3	3	value3k		
4	4	value4		
5	5	value5		

Data Output	Explain	Messages	Notifications								
<table> <tr> <th></th><th>id [PK] bigint</th><th>readerLogID bigint</th><th>readerLogName character varying (255)</th></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>		id [PK] bigint	readerLogID bigint	readerLogName character varying (255)							
	id [PK] bigint	readerLogID bigint	readerLogName character varying (255)								

Виконуємо такий же запит для 4 рядка:

postgres/postgres@lola

Query Editor    Query History

```
1 UPDATE "reader" SET "readerName" = "readerName" || 'k' WHERE "readerID" = 4
```

	readerID [PK] bigint	readerName character varying (255)
1	1	value1
2	2	value2
3	3	value3k
4	4	value4k
5	5	value5

	id [PK] bigint	readerLogID bigint	readerLogName character varying (255)
1	1	4	value4

Як бачимо, тригер спрацював і зробив запис до Logs.

Робиму запит на видалення за номером рядка 4:

```
DELETE FROM "reader" WHERE "readerID" = 4
```

Data Output Explain Messages Notification

	readerID [PK] bigint	readerName character varying (255)	
1	1	value1	
2	2	value2	
3	3	value3k	
4	4	value4k	
5	5	value5	

	id [PK] bigint	readerLogID bigint	readerLogName character varying (255)	
1	1	4	value4	
2	2	4	value4k	

Тригер спрацював на видалення також.

**GitHub repo:**

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