НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Проектування високонавантажених систем

Лабораторна робота №2

Виконав:
Поночевний Назар ФІ-92
Перевірив:
Родіонов А. М.

Київ – 2022

ЛАБОРАТОРНА РОБОТА №2

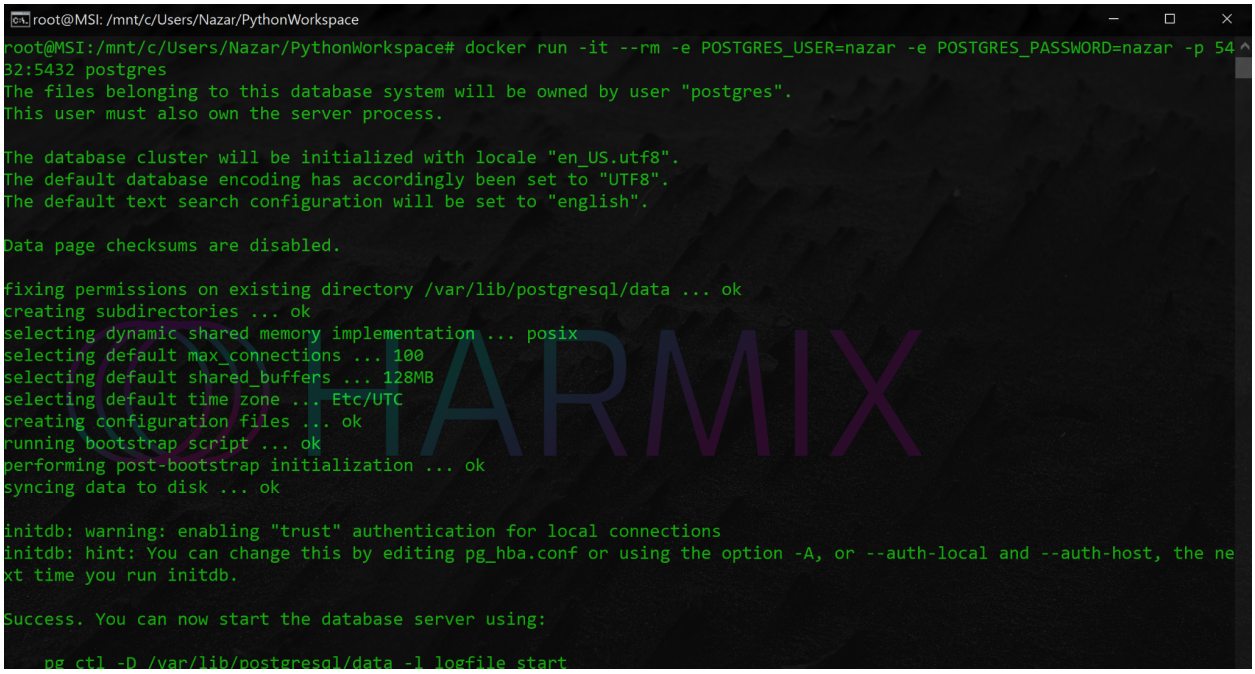
Реалізація каунтера з використанням PostgreSQL

Завдання:

1) Необхідно декількома способами реалізувати оновлення значення каунтера в СКБД PostgreSQL та оцінити час кожного із варіантів.
    a) Lost-update (реалізація що втрачатиме значення)
    b) In-place update
    c) Row-level locking
    d) Optimistic concurrency control

Результати

1) Інсталяція PostgreSQL



2) Код (для економії часу інкремент до 1000 в 10 потоках):

```
import time
import psycopg2
import threading


# Docker: docker run -it --rm -e POSTGRES_USER=nazar -e
POSTGRES_PASSWORD=nazar -p 5432:5432 postgres
config = {
    "host": "localhost",
    "port": 5432,
```

```
    "dbname": "postgres",
    "user": "nazar",
    "password": "nazar",
}
conn = psycopg2.connect(**config)
cursor = conn.cursor()
inc_value = 1000


def reset_db(conn, cursor):
    cursor.execute("DROP TABLE IF EXISTS user_counter")
    cursor.execute("CREATE TABLE user_counter (user_id INT PRIMARY
KEY, counter INT, version INT)")
    conn.commit()

    cursor.execute("INSERT INTO user_counter (user_id, counter,
version) VALUES (1, 0, 1)")
    conn.commit()

def get_counter(conn, cursor):
    cursor.execute("SELECT counter FROM user_counter WHERE user_id =
1")
    counter = cursor.fetchone()[0]
    return counter

def inference(func):
    threads = []
    for t in range(10):
        thread = threading.Thread(target=func)
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

def lost_update():
    with psycopg2.connect(**config) as conn:
        conn.autocommit = True
        cursor = conn.cursor()
        for _ in range(inc_value):
            cursor.execute("SELECT counter FROM user_counter WHERE
user_id = 1")
            counter = cursor.fetchone()[0]
```

```
            counter += 1
            cursor.execute(f"UPDATE user_counter SET counter =
{counter} WHERE user_id = 1")

def in_place_update():
    with psycopg2.connect(**config) as conn:
        conn.autocommit = True
        cursor = conn.cursor()
        for _ in range(inc_value):
            cursor.execute("UPDATE user_counter SET counter = counter
+ 1 WHERE user_id = 1")

def row_level_locking():
    with psycopg2.connect(**config) as conn:
        conn.autocommit = True
        cursor = conn.cursor()
        for _ in range(inc_value):
            cursor.execute("SELECT counter FROM user_counter WHERE
user_id = 1 FOR UPDATE")
            counter = cursor.fetchone()[0]
            counter += 1
            cursor.execute(f"UPDATE user_counter SET counter =
{counter} WHERE user_id = 1")

def optimistic_concurrency():
    with psycopg2.connect(**config) as conn:
        conn.autocommit = True
        cursor = conn.cursor()
        for _ in range(inc_value):
            while True:
                cursor.execute("SELECT counter, version FROM
user_counter WHERE user_id = 1")
                counter, version = cursor.fetchone()
                counter += 1
                cursor.execute(f"UPDATE user_counter SET counter =
{counter}, version = {version + 1} WHERE user_id = 1 AND version =
{version}")
                if cursor.rowcount > 0:
                    break


reset_db(conn, cursor)
start_time = time.time()
```

```
inference(lost_update)
print("--- Lost-update ---")
print(f"Final value of counter: {get_counter(conn, cursor)}")
print(f"Execution time: {round(time.time() - start_time, 2)}")

reset_db(conn, cursor)
start_time = time.time()
inference(in_place_update)
print("--- In-place update ---")
print(f"Final value of counter: {get_counter(conn, cursor)}")
print(f"Execution time: {round(time.time() - start_time, 2)}")

reset_db(conn, cursor)
start_time = time.time()
inference(row_level_locking)
print("--- Row-level locking ---")
print(f"Final value of counter: {get_counter(conn, cursor)}")
print(f"Execution time: {round(time.time() - start_time, 2)}")

reset_db(conn, cursor)
start_time = time.time()
inference(optimistic_concurrency)
print("--- Optimistic concurrency control ---")
print(f"Final value of counter: {get_counter(conn, cursor)}")
print(f"Execution time: {round(time.time() - start_time, 2)}")

conn.close()
```

3) Знімки екрана з логами виконання: