Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

Протокол лабораторної роботи №3
з дисципліни Проектування високонавантажених систем
на тему: Реалізація каунтера з використанням PostgreSQL

Виконав: студент групи ФІ-21
Грунда Ярослав

Київ, 2025

Завдання

1. Реалізувати каунтер Lost-update версію
2. Реалізувати каунтер Serializable update версію
3. Реалізувати каунтер In-place update версію
4. Реалізувати каунтер Row-level locking версію
5. Реалізувати каунтер Optimistic concurrency control версію

Зміст

1 Старт роботи

1.1 Запуск Docker-контейнерів

```
# docker-compose.yml

services:
  db:
    image: postgres:latest
    container_name: pg-db
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: labdb
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U user -d labdb"]
      interval: 2s
      timeout: 2s
      retries: 20

  client:
    build: .
    container_name: pg-client
    environment:
      DB_HOST: db
      DB_PORT: "5432"
      DB_NAME: labdb
      DB_USER: user
      DB_PASS: pass
    depends_on:
      db:
        condition: service_healthy
```

1.2 Dockerfile

```
# Dockerfile
FROM python:3.13-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY counter_runner.py .

ENTRYPOINT ["python", "counter_runner.py"]
```

2 Результати

```
D:\University\semestr_7\High-Load_Systems\lab3>docker compose run --rm client
[+] Creating 1/1
 ✓ Container pg-db  Running
Table 'user_counter' is ready.

All counters have been reset.

Running test: Lost Update (user_id=1)
Lost Update finished in 330.09 seconds | 303 ops/s

Running test: Serializable (naive) (user_id=2)
Lost updates: 9006
Lost updates: 8949
Lost updates: 8921
Lost updates: 8963
Lost updates: 8761
Lost updates: 9118
Lost updates: 9045
Lost updates: 8962
Lost updates: 8974
Lost updates: 9016
Serializable (naive) finished in 38.38 seconds | 2606 ops/s

Running test: Serializable (safe) (user_id=3)
Serializable (safe) finished in 376.86 seconds | 265 ops/s

Running test: In-place (user_id=4)
In-place finished in 327.93 seconds | 305 ops/s

Running test: Row Lock (user_id=5)
Row Lock finished in 354.27 seconds | 282 ops/s

Running test: Optimistic (user_id=6)
Optimistic finished in 2755.76 seconds | 36 ops/s

Final table:
user_id | counter | version
---------------------------
      1 |   10080 |       0
      2 |   10285 |       0
      3 |  100000 |       0
      4 |  100000 |       0
      5 |  100000 |       0
      6 |  100000 |  100000

All tests completed.
```

3 Висновки

1. Підхід без блокувань (Lost Update) показав середню продуктивність (303 оп/с), проте через відсутність механізму синхронізації відбулась значна втрата даних. Значення лічильника не відповідає очікуваному результату — наслідок стану гонитви (race condition).

2. Підхід з рівнем ізоляції SERIALIZABLE (naive) забезпечив вищу швидкість (2606 оп/с), однак через відсутність повторних спроб у разі конфліктів спостерігалися значні втрати оновлень. Це демонструє, що навіть при максимальному рівні ізоляції потрібна обробка помилок SerializationFailure.

3. Підхід SERIALIZABLE (safe), що використовує повтор транзакцій у разі конфлікту, повністю зберіг цілісність даних. Всі інкременти були виконані коректно, хоча продуктивність знизилась до 265 оп/с через додаткові перезапуски транзакцій.

4. Оновлення ”на місці”(In-place update) показало середню швидкість (305 оп/с) та коректність результатів. Такий підхід є простим і ефективним, коли оновлення стосується лише одного поля без складних залежностей.

5. Песимістичне блокування (Row Lock) забезпечило повну цілісність даних (жодних втрат), але ціною стало зниження пропускної здатності (282 оп/с), оскільки транзакції очікують одна одну при блокуванні рядків.

6. Оптимістичне блокування (Optimistic) також гарантувало відсутність втрат даних, але показало найнижчу швидкість (36 оп/с) через часті перевірки версій та повторні спроби оновлень. Цей підхід підходить для сценаріїв із рідкісними конфліктами, але не для висококонкурентних потоків.

4 Код

Посилання на репозиторій: GitHub

```python
# counter_runner.py
import os, time, threading
import psycopg2
from psycopg2 import errors
from psycopg2.extensions import ISOLATION_LEVEL_SERIALIZABLE

# --- DB CONFIG ---
DB_CFG = dict(
    host=os.environ.get("DB_HOST", "localhost"),
    port=int(os.environ.get("DB_PORT", "5432")),
    dbname=os.environ.get("DB_NAME", "labdb"),
    user=os.environ.get("DB_USER", "user"),
    password=os.environ.get("DB_PASS", "pass"),
)

# --- DB INIT ---
def init_database():
    conn = psycopg2.connect(**DB_CFG)
    conn.autocommit = True
    with conn.cursor() as cur:
        cur.execute("""
```

```
            CREATE TABLE IF NOT EXISTS user_counter (
                user_id  INTEGER PRIMARY KEY,
                counter  INTEGER NOT NULL,
                version  INTEGER NOT NULL
            );
        """)
        for uid in range(1, 7):
            cur.execute(f"""
                INSERT INTO user_counter (user_id, counter, version)
                VALUES ({uid}, 0, 0)
                ON CONFLICT (user_id) DO NOTHING;
            """)
    conn.close()
    print("Table 'user_counter' is ready.\n")

def new_conn(serializable=False):
    conn = psycopg2.connect(**DB_CFG)
    conn.autocommit = False
    if serializable:
        conn.set_isolation_level(ISOLATION_LEVEL_SERIALIZABLE)
    return conn

def reset_all():
    with new_conn() as conn:
        with conn.cursor() as cur:
            cur.execute("UPDATE user_counter SET counter=0, version=0;")
        conn.commit()
    print("All counters have been reset.\n")

# --- WORKERS ---
def lost_update_worker(iters, user_id):
    conn = new_conn()
    with conn.cursor() as cur:
        for _ in range(iters):
            cur.execute(f"SELECT counter FROM user_counter WHERE user_id
               = {user_id};")
            (c,) = cur.fetchone()
            c += 1
            cur.execute(f"UPDATE user_counter SET counter = {c} WHERE
               user_id = {user_id};")
            conn.commit()
    conn.close()

def serializable_naive_worker(iters, user_id):
    conn = new_conn(serializable=True)
    mistakes = 0
    with conn.cursor() as cur:
        for _ in range(iters):
            try:
                cur.execute(f"SELECT counter FROM user_counter WHERE
                   user_id = {user_id};")
                (c,) = cur.fetchone()
                c += 1
```

```
72                  cur.execute(f"UPDATE user_counter SET counter = {c} WHERE
                        user_id = {user_id};")
73                  conn.commit()
74              except errors.SerializationFailure:
75                  conn.rollback()
76                  mistakes += 1
77                  continue
78
79      if mistakes > 0:
80          print(f"Lost updates: {mistakes}")
81
82      conn.close()
83
84  def serializable_worker(iters, user_id):
85      conn = new_conn(serializable=True)
86      with conn.cursor() as cur:
87          for _ in range(iters):
88              while True:
89                  try:
90                      cur.execute(f"SELECT counter FROM user_counter WHERE
                            user_id = {user_id};")
91                      (c,) = cur.fetchone()
92                      c += 1
93                      cur.execute(f"UPDATE user_counter SET counter = {c}
                            WHERE user_id = {user_id};")
94                      conn.commit()
95                      break
96                  except errors.SerializationFailure:
97                      conn.rollback()
98                      time.sleep(0.001)
99      conn.close()
100
101 def inplace_worker(iters, user_id):
102     conn = new_conn()
103     with conn.cursor() as cur:
104         for _ in range(iters):
105             cur.execute(f"UPDATE user_counter SET counter = counter + 1
                    WHERE user_id = {user_id};")
106             conn.commit()
107     conn.close()
108
109 def rowlock_worker(iters, user_id):
110     conn = new_conn()
111     with conn.cursor() as cur:
112         for _ in range(iters):
113             cur.execute(f"SELECT counter FROM user_counter WHERE user_id
                    = {user_id} FOR UPDATE;")
114             (c,) = cur.fetchone()
115             c += 1
116             cur.execute(f"UPDATE user_counter SET counter = {c} WHERE
                    user_id = {user_id};")
117             conn.commit()
118     conn.close()
```

```
119
120 def optimistic_worker(iters, user_id):
121     conn = new_conn()
122     with conn.cursor() as cur:
123         for _ in range(iters):
124             while True:
125                 cur.execute(f"SELECT counter, version FROM user_counter
                        WHERE user_id = {user_id};")
126                 c, v = cur.fetchone()
127                 c += 1
128                 cur.execute(f"""
129                     UPDATE user_counter
130                     SET counter = {c}, version = {v + 1}
131                     WHERE user_id = {user_id} AND version = {v};
132                 """)
133                 updated = cur.rowcount
134                 conn.commit()
135                 if updated > 0:
136                     break
137     conn.close()
138
139 MODES = {
140     1: ("Lost Update", lost_update_worker),
141     2: ("Serializable (naive)", serializable_naive_worker),
142     3: ("Serializable (safe)", serializable_worker),
143     4: ("In-place", inplace_worker),
144     5: ("Row Lock", rowlock_worker),
145     6: ("Optimistic", optimistic_worker)
146 }
147
148 # --- MAIN ---
149 def run_mode(name, func, user_id, threads, iters):
150     print(f"Running test: {name} (user_id={user_id})")
151     total_ops = threads * iters
152     start = time.perf_counter()
153     threads_list = [threading.Thread(target=func, args=(iters, user_id))
            for _ in range(threads)]
154     for t in threads_list:
155         t.start()
156     for t in threads_list:
157         t.join()
158     dur = time.perf_counter() - start
159     print(f"{name} finished in {dur:.2f} seconds | {total_ops/dur:.0f}
        ops/s\n")
160
161 def read_all():
162     with new_conn() as conn:
163         with conn.cursor() as cur:
164             cur.execute("SELECT * FROM user_counter ORDER BY user_id;")
165             rows = cur.fetchall()
166     print("Final table:")
167     print("user_id | counter | version")
168     print("---------------------------")
```

```
    for r in rows:
        print(f"{r[0]:7} | {r[1]:7} | {r[2]:7}")
    print()

def main():
    init_database()
    reset_all()

    THREADS = 10
    ITERS = 10000

    for uid, (name, func) in MODES.items():
        run_mode(name, func, uid, THREADS, ITERS)

    read_all()
    print("All tests completed.")

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