

# Проектування високонавантажених систем.

## Лабораторна 3, звіт

Михайло Голуб

8 грудня 2025 р.

## 1 Завдання лабораторної роботи

### 1.1 Загальне

Необхідно декількома способами реалізувати оновлення значення каунтера в СКБД PostgreSQL та оцінити час кожного із варіантів.

Таблиця `user_counter` з колонками `USER_ID`, `Counter`, `Version`.

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

### 1.2 Вимоги до звіту та реалізації

- Мова реалізації будь-яка
- Не використовувати ORM-фреймворки (Hibernate, SQLAlchemy, ...)
- Не забувати про необхідність окремої транзакції на кожен запис

## 2 Хід роботи

### 2.1 Реалізація клієнта

Лістинг 1: `lost_update_or_serialize.py`

```
1 import threading
2 import time
3 import random
4 from tqdm import tqdm
```

```

5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 SERIALISE = True
19
20 print_lock = threading.Lock()
21
22 def get_conn():
23     if SERIALISE:
24         conn = psycopg2.connect(
25             host=DB_HOST,
26             dbname=DB_NAME,
27             user=DB_USER,
28             password=DB_PASS,
29             port=DB_PORT,
30         )
31         conn.set_isolation_level(extensions.
ISOLATION_LEVEL_SERIALIZABLE)
32         return conn
33     else:
34         return psycopg2.connect(
35             host=DB_HOST,
36             dbname=DB_NAME,
37             user=DB_USER,
38             password=DB_PASS,
39             port=DB_PORT
40         )
41 def init_db():
42     conn = get_conn()
43     cur = conn.cursor()
44     cur.execute("""
45         CREATE TABLE IF NOT EXISTS user_counter (
46             USER_ID SERIAL PRIMARY KEY,
47             Counter BIGINT NOT NULL,
48             Version BIGINT NOT NULL
49         );
50     """)
51     # Reset counter to 0
52     cur.execute("DELETE FROM user_counter;")
53     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
54     conn.commit()
55     cur.close()
56     conn.close()
57
58 def worker(client_id):
59     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
str(client_id), leave=False)

```

```

60     conn = get_conn()
61     cur = conn.cursor()
62     for i in bar:
63
64         cur.execute("SELECT counter FROM user_counter WHERE user_id
        = 1")
65         counter = cur.fetchone()[0]
66         counter += 1
67         cur.execute("UPDATE user_counter SET counter = %s WHERE
        user_id = %s", (counter, 1))
68         conn.commit()
69     cur.close()
70     conn.close()
71     with print_lock:
72         bar.close()
73
74 def main():
75     if SERIALISE:
76         print("Serialise")
77     else:
78         print("Lost-upddate")
79     print("Initialising DB")
80     init_db()
81     time.sleep(1)
82
83     conn = get_conn()
84     cur = conn.cursor()
85     print("DB initialised.")
86     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
        1;")
87     print("Counter state:", cur.fetchone()[0])
88     conn.commit()
89     cur.close()
90     conn.close()
91     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
        calls each...")
92
93
94     start = time.time()
95
96     threads = []
97     for i in range(NUM_CLIENTS):
98         t = threading.Thread(target=worker, args=(i,))
99         t.start()
100         threads.append(t)
101     print("MAIN: Threads created")
102     for t in threads:
103         t.join()
104     print("MAIN: Threads joined")
105     end = time.time()
106     elapsed = end - start
107     final_count = -1
108     i = 0
109     conn = get_conn()
110     cur = conn.cursor()
111     print("DB initialised.")
112     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =

```

```

113         1;")
114         final_count = cur.fetchone()[0]
115         conn.commit()
116         cur.close()
117         conn.close()
118
119         total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
120         throughput = total_calls / elapsed
121
122         print(f"Final count: {final_count}")
123         print(f"Total time: {elapsed:.2f} s")
124         print(f"Throughput: {throughput:.2f} requests/sec")
125
126 if __name__ == "__main__":
127     main()
128     input()

```

Приклад логу під час роботи клієнта:

```

Lost-upddate
Initialising DB
DB initialised.
Counter state: 0
Starting 10 clients x 10000 calls each...
MAIN: Threads created
0:  0%| 42/10000 [00:03<10:55, 15.20it/s]
1:  0%| 44/10000 [00:03<11:07, 14.92it/s]
2:  0%| 44/10000 [00:03<11:21, 14.62it/s]
3:  0%| 44/10000 [00:03<10:48, 15.35it/s]
4:  0%| 40/10000 [00:02<11:04, 14.98it/s]
5:  0%| 43/10000 [00:03<14:04, 11.79it/s]
6:  0%| 44/10000 [00:03<11:08, 14.90it/s]
7:  0%| 42/10000 [00:03<12:22, 13.41it/s]
8:  0%| 43/10000 [00:03<12:32, 13.22it/s]
9:  0%| 46/10000 [00:03<11:12, 14.79it/s]

```

Приклад логу після завершення роботи клієнта:

```

Lost-upddate
Initialising DB
DB initialised.
Counter state: 0
Starting 10 clients x 10000 calls each...
MAIN: Threads created
MAIN: Threads joined
DB initialised.
Final count: 10661
Total time:  15.21 s
Throughput:  6573.22 requests/sec

```

## 2.2 Lost-update

Лістинг об'єднаного lost-update та Serializable (тип роботи обирається змінною) клієнта був наведений вище.

Результати тестування:

Час роботи: 15.21 секунди

Пропускна здатність: 6573 запитів/с

Кінцеве значення лічильника: 10661

## 2.3 Serializable

При запуску таких самих запитів, але з рівнем ізоляції `SERIALIZABLE`, програма видає наступну помилку на всіх воркерах, окрім одного:

```
cur.execute("UPDATE user_counter SET counter = %s WHERE user_id = %s", (counter, 1))
psycopg2.errors.SerializationFailure:
ПОМИЛКА: не вдалося серіалізувати доступ через паралельне оновлення
```

Результати тестування:

Час роботи: 3 секунди

Пропускна здатність: — запитів/с

Кінцеве значення лічильника: 10к

З певного моменту, лише один клієнт робив інкрементацію лічильника.

Таку помилкову поведінку можна виправити використавши In-place update або робити повторні запити у разі помилки.

## 2.4 Serializable з повтором

Якщо додати до Serializable повторний запит до таблиці у разі отримання помилки серіалізації, можна отримати щось що схоже на оптимістичне блокування: якщо все ок — підтвердити операцію; якщо не вдалося серіалізувати операції — повернути помилку, клієнти очікують випадковий час і пробують зробити операцію знову

Лістинг 2: `serialize_and_repeat.py`

```
1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
```

```

15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     conn = psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT,
27     )
28     conn.set_isolation_level(extensions.
ISOLATION_LEVEL_SERIALIZABLE)
29     return conn
30 def init_db():
31     conn = get_conn()
32     cur = conn.cursor()
33     cur.execute("""
34         CREATE TABLE IF NOT EXISTS user_counter(
35             USER_ID SERIAL PRIMARY KEY,
36             Counter BIGINT NOT NULL,
37             Version BIGINT NOT NULL
38         );
39     """)
40     # Reset counter to 0
41     cur.execute("DELETE FROM user_counter;")
42     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
43     conn.commit()
44     cur.close()
45     conn.close()
46
47 def worker(client_id):
48     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
str(client_id), leave=False)
49     for i in bar:
50         conn = get_conn()
51         cur = conn.cursor()
52         while True:
53             try:
54
55                 cur.execute("SELECT counter FROM user_counter WHERE
user_id = 1")
56                 counter = cur.fetchone()[0]
57                 counter += 1
58                 cur.execute("UPDATE user_counter SET counter = %s
WHERE user_id = %s", (counter, 1))
59                 conn.commit()
60
61                 break
62             except:
63                 try:
64                     cur.close()
65                     conn.close()
66                 except:
67                     pass

```

```

68         time.sleep(random.uniform(0.01, 0.05))
69         conn = get_conn()
70         cur = conn.cursor()
71         cur.close()
72         conn.close()
73     with print_lock:
74         bar.close()
75
76 def main():
77     print("Serialise and repeat")
78     print("Initialising DB")
79     init_db()
80     time.sleep(1)
81
82     conn = get_conn()
83     cur = conn.cursor()
84     print("DB initialised.")
85     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
86 1;")
87     print("Counter state:", cur.fetchone()[0])
88     conn.commit()
89     cur.close()
90     conn.close()
91     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
92 calls each...")
93
94     start = time.time()
95
96     threads = []
97     for i in range(NUM_CLIENTS):
98         t = threading.Thread(target=worker, args=(i,))
99         t.start()
100         threads.append(t)
101     print("MAIN: Threads created")
102     for t in threads:
103         t.join()
104     print("MAIN: Threads joined")
105     end = time.time()
106     elapsed = end - start
107     final_count = -1
108     i = 0
109     conn = get_conn()
110     cur = conn.cursor()
111     print("DB initialised.")
112     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
113 1;")
114     final_count = cur.fetchone()[0]
115     conn.commit()
116     cur.close()
117     conn.close()
118
119     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
120     throughput = total_calls / elapsed
121
122     print(f"Final count: {final_count}")
123     print(f"Total time: {elapsed:.2f} s")

```

```

122     print(f"Throughput: {throughput:.2f} requests/sec")
123
124 if __name__ == "__main__":
125     main()
126     input()

```

Результати тестування:

Час роботи: 860 секунд

Пропускна здатність: 116.15 запитів/с

Кінцеве значення лічильника: 100к

## 2.5 In-place update

Лістинг 3: Inplace.py

```

1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28
29 def init_db():
30     conn = get_conn()
31     cur = conn.cursor()
32     cur.execute("""
33         CREATE TABLE IF NOT EXISTS user_counter (
34             USER_ID SERIAL PRIMARY KEY,
35             Counter BIGINT NOT NULL,
36             Version BIGINT NOT NULL
37         );
38     """)
39     # Reset counter to 0
40     cur.execute("DELETE FROM user_counter;")
41     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
42     conn.commit()

```

```

42     cur.close()
43     conn.close()
44
45 def worker(client_id):
46     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
47         str(client_id), leave=False)
48     conn = get_conn()
49     cur = conn.cursor()
50     for i in bar:
51         cur.execute("UPDATE user_counter SET Counter = Counter + 1
52             WHERE USER_ID = %s", (1,))
53         conn.commit()
54     cur.close()
55     conn.close()
56     with print_lock:
57         bar.close()
58
59 def main():
60     print("In-place update")
61     print("Initialising DB")
62     init_db()
63     time.sleep(1)
64
65     conn = get_conn()
66     cur = conn.cursor()
67     print("DB initialised.")
68     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
69         1;")
70     print("Counter state:", cur.fetchone()[0])
71     conn.commit()
72     cur.close()
73     conn.close()
74     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
75         calls each...")
76
77     start = time.time()
78
79     threads = []
80     for i in range(NUM_CLIENTS):
81         t = threading.Thread(target=worker, args=(i,))
82         t.start()
83         threads.append(t)
84     print("MAIN: Threads created")
85     for t in threads:
86         t.join()
87     print("MAIN: Threads joined")
88     end = time.time()
89     elapsed = end - start
90     final_count = -1
91     i = 0
92     conn = get_conn()
93     cur = conn.cursor()
94     print("DB initialised.")
95     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
96         1;")

```

```

94     final_count = cur.fetchone()[0]
95     conn.commit()
96     cur.close()
97     conn.close()
98
99     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
100     throughput = total_calls / elapsed
101
102     print(f"Final count: {final_count}")
103     print(f"Total time: {elapsed:.2f} s")
104     print(f"Throughput: {throughput:.2f} requests/sec")
105
106 if __name__ == "__main__":
107     main()
108     input()

```

Результати тестування:

Час роботи: 12.56 секунд

Пропускна здатність: 7963.53 запитів/с

Кінцеве значення лічильника: 100к

## 2.6 Row-level locking

Лістинг 4: Rowlevel.py

```

1 import threading
2 import time
3 import random
4 from tqdm import tqdm
5 import psycopg2
6 from psycopg2.extras import RealDictCursor
7 from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():
21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28
29 def init_db():
30     conn = get_conn()
31     cur = conn.cursor()
32     cur.execute("""
33         CREATE TABLE IF NOT EXISTS user_counter (

```

```

33         USER_ID SERIAL PRIMARY KEY,
34         Counter BIGINT NOT NULL,
35         Version BIGINT NOT NULL
36     );
37     """
38     # Reset counter to 0
39     cur.execute("DELETE FROM user_counter;")
40     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
41     conn.commit()
42     cur.close()
43     conn.close()
44
45 def worker(client_id):
46     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
47         str(client_id), leave=False)
48     conn = get_conn()
49     cur = conn.cursor()
50     for i in bar:
51         cur.execute("SELECT Counter FROM user_counter WHERE USER_ID
52             = 1 FOR UPDATE")
53         counter = cur.fetchone()
54         counter = counter[0] + 1
55         cur.execute("UPDATE user_counter SET Counter = %s WHERE
56             USER_ID = %s", (counter, 1))
57         conn.commit()
58         cur.close()
59         conn.close()
60         with print_lock:
61             bar.close()
62
63 def main():
64     print("Row-level locking")
65     print("Initialising DB")
66     init_db()
67     time.sleep(1)
68
69     conn = get_conn()
70     cur = conn.cursor()
71     print("DB initialised.")
72     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
73         1;")
74     print("Counter state:", cur.fetchone()[0])
75     conn.commit()
76     cur.close()
77     conn.close()
78     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
79         calls each...")
80
81     start = time.time()
82
83     threads = []
84     for i in range(NUM_CLIENTS):
85         t = threading.Thread(target=worker, args=(i,))
86         t.start()
87         threads.append(t)
88     print("MAIN: Threads created")

```

```

85     for t in threads:
86         t.join()
87     print("MAIN: Threads joined")
88     end = time.time()
89     elapsed = end - start
90     final_count = -1
91     i = 0
92     conn = get_conn()
93     cur = conn.cursor()
94     print("DB initialised.")
95     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
96     1;")
97     final_count = cur.fetchone()[0]
98     conn.commit()
99     cur.close()
100    conn.close()
101
102    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
103    throughput = total_calls / elapsed
104
105    print(f"Final count: {final_count}")
106    print(f"Total time: {elapsed:.2f} s")
107    print(f"Throughput: {throughput:.2f} requests/sec")
108
109    if __name__ == "__main__":
110        main()
111        input()

```

Результати тестування:

Час роботи: 20.06 секунд

Пропускна здатність: 4985.25 запитів/с

Кінцеве значення лічильника: 100к

## 2.7 Optimistic concurrency control

Лістинг 5: Optimistic.py

```

1  import threading
2  import time
3  import random
4  from tqdm import tqdm
5  import psycopg2
6  from psycopg2.extras import RealDictCursor
7  from psycopg2 import extensions
8
9
10 DB_HOST = "localhost"
11 DB_NAME = "HLS"
12 DB_USER = "postgres"
13 DB_PASS = "misube2105"
14 DB_PORT = 5432
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_conn():

```

```

21     return psycopg2.connect(
22         host=DB_HOST,
23         dbname=DB_NAME,
24         user=DB_USER,
25         password=DB_PASS,
26         port=DB_PORT
27     )
28 def init_db():
29     conn = get_conn()
30     cur = conn.cursor()
31     cur.execute("""
32         CREATE TABLE IF NOT EXISTS user_counter (
33             USER_ID SERIAL PRIMARY KEY,
34             Counter BIGINT NOT NULL,
35             Version BIGINT NOT NULL
36         );
37     """)
38     # Reset counter to 0
39     cur.execute("DELETE FROM user_counter;")
40     cur.execute("INSERT INTO user_counter VALUES (1, 0, 0);")
41     conn.commit()
42     cur.close()
43     conn.close()
44
45 def worker(client_id):
46     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
47         str(client_id), leave=False)
48     conn = get_conn()
49     cur = conn.cursor()
50     for i in bar:
51         while True:
52             cur.execute("SELECT Counter, Version FROM user_counter
53             WHERE USER_ID = 1")
54             counter, version = cur.fetchone()
55             counter = counter + 1
56             cur.execute("update user_counter set counter = %s,
57             version = %s where user_id = %s and version = %s",
58                 (counter, version + 1, 1, version))
59             conn.commit()
60             count = cur.rowcount
61             if (count > 0): break
62         cur.close()
63         conn.close()
64         with print_lock:
65             bar.close()
66
67 def main():
68     print("Row-level locking")
69     print("Initialising DB")
70     init_db()
71     time.sleep(1)
72
73     conn = get_conn()
74     cur = conn.cursor()
75     print("DB initialised.")
76     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
77     1;")

```

```

74     print("Counter state:", cur.fetchone()[0])
75     conn.commit()
76     cur.close()
77     conn.close()
78     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
calls each...")
79
80
81     start = time.time()
82
83     threads = []
84     for i in range(NUM_CLIENTS):
85         t = threading.Thread(target=worker, args=(i,))
86         t.start()
87         threads.append(t)
88     print("MAIN: Threads created")
89     for t in threads:
90         t.join()
91     print("MAIN: Threads joined")
92     end = time.time()
93     elapsed = end - start
94     final_count = -1
95     i = 0
96     conn = get_conn()
97     cur = conn.cursor()
98     print("DB initialised.")
99     cur.execute("SELECT Counter FROM user_counter WHERE USER_ID =
1;")
100    final_count = cur.fetchone()[0]
101    conn.commit()
102    cur.close()
103    conn.close()
104
105    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
106    throughput = total_calls / elapsed
107
108    print(f"Final count: {final_count}")
109    print(f"Total time: {elapsed:.2f} s")
110    print(f"Throughput: {throughput:.2f} requests/sec")
111
112 if __name__ == "__main__":
113     main()
114     input()

```

Результати тестування:

Час роботи: 88 секунд

Пропускна здатність: 1136.31 запитів/с

Кінцеве значення лічильника: 100к

### 3 Результати

Варіант	Час роботи	Пропускна здатність	Значення лічильника
Lost-update	15.21	6573	10661
Serializable	3	—	10к
Ser. з повтором	860	116.15	100k
Inplace update	12.56	7963	100k
Row-level locking	20.06	4985	100k
Optimistic concurrency control	88	1136.31	100k