Sprawozdanie

Zbieranie i Analiza Danych

Labolatorium nr 3 – Zapis do bazy danych, wydajność

Oskar Kwidziński

S156013

Fizyka Techniczna

Semestr zimowy 2022/2023

Wstęp

W zadaniu utworzono program zapisujący do bazy danych logi (date oraz temp) w czterech poziomach izolacji transakcji (read uncommitted, read commited, repeatable read, serializable). Testy przeprowadzono ze względu na:

- sposób zapisu:

  - z indeksem primary key,
  - z autoinkrementującym indeksem primary key,
  - bez indeksu;

- potencjalny sposób zachowania użytkownika bazy:

  - to samo połączenie, bez commitów,
  - to samo połączenie, commit po każdym dodaniu rekordu,
  - połączenie tworzone przed każdym dodaniem rekordu, bez commitów
  - połączenie tworzone przed każdym dodaniem rekordu, commit po każdym dodaniu rekordu.

Testy dokonano dla 100, i 1000 wierszy wprowadzanych do tabeli.

Rozwiązanie

W zadaniu wykorzystano lokalną bazę MySQL (XAMMP). Wywołanie przykładowego testu – dla 1000 wstawień i zachowaniu użytkownika „to samo połączenie, commit po każdym dodaniu rekordu” wywoływane było z funkcji main w postaci bloku kodu:

```
def main():
    isolation_levels = ["READ COMMITTED", "READ UNCOMMITTED", "REPEATABLE READ", "SERIALIZABLE"]
    test_types = ['No pk', 'With pk', 'With autopk']
    insertions = 10000

    mode = "CONNECTION: one, COMMIT: after each insertion"
    print(f"\n------- {mode} -------")
    for isolation_level in isolation_levels:
        connect, cursor = create_connection()
        cursor.execute(f"SET SESSION TRANSACTION ISOLATION LEVEL {isolation_level}")
        print("---", isolation_level, "---")
        for test_type in test_types:
            total_time = 0
            time.sleep(1)
            create_table(cursor, test_type)
            for insertion in range(insertions):
                insertion_time = db_execute_commit(cursor, insertion, test_type)
                total_time += insertion_time
            time.sleep(1)
            cursor.execute("DROP TABLE IF EXISTS tab")
            print(f"{test_type}:", total_time / insertions)
            data.append(total_time / insertions)
        connect.close()
    for i in range(len(isolation_levels)):
        writer.writerows([[mode + " " + isolation_levels[i]], test_types, data[(3*i):(3*(i+1))]])
    data = []
```

Powyższy blok kodu dokonuje testu dla wszystkich poziomów izolacji oraz typów testu. Zarówno tworzenie tabeli (funkcja create_table) jak i zapisywanie rekordów do bazy (funkcja insert_row)

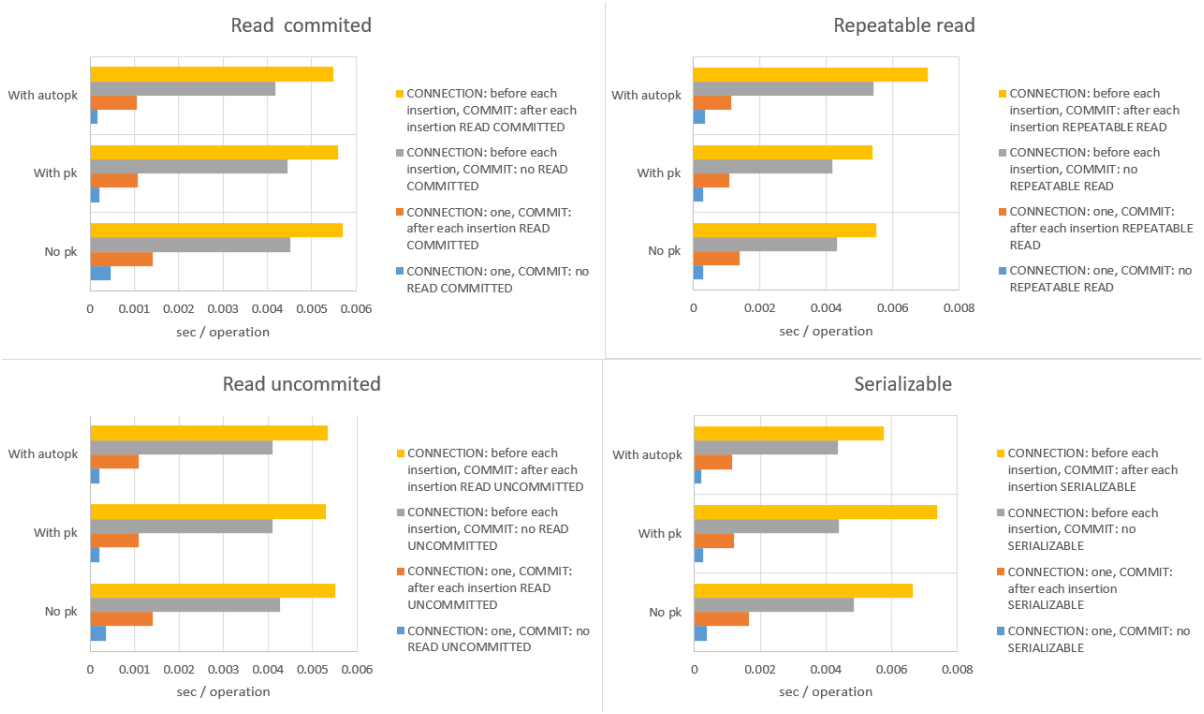
dokonywane było bez użycia ORM. W trakcie wykonywania funkcji db_execute_commit mierzony był czas operacji:

```
def db_execute_commit(cursor, i, test_type):
    timer_start = time.time()
    insert_row(cursor, i, test_type)
    cursor.execute("COMMIT")
    timer_stop = time.time()
    insertion_time = timer_stop - timer_start
    return insertion_time
```

który podlegał potem uśrednieniu ze względu na ilość wstawień w dokonywanej próbie. Na podobnej zasadzie przeprowadzono test dla wszystkich pozostałych ilości prób i trybów zachowania użytkownika bazy.

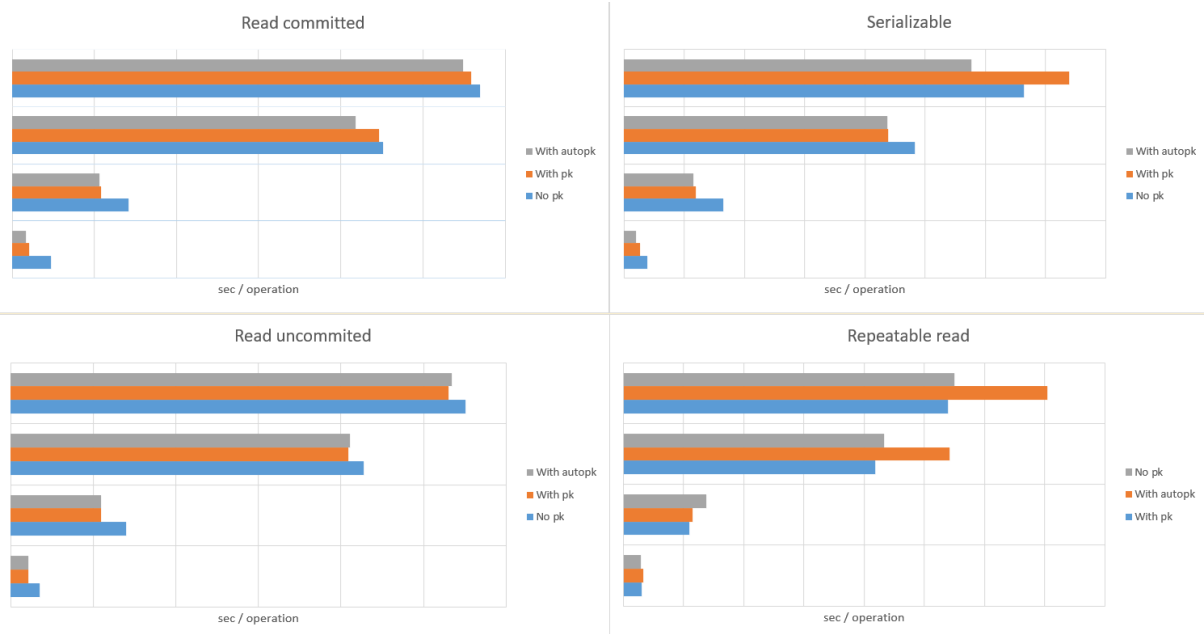
Podsumowanie

Wyniki dla wstawienia 100 wierszy do bazy prezentują się następująco:



Zgodnie z przypuszczaniem, czas potrzebny na operację, w przypadku każdorazowego nawiązywania nowego połączenia z bazą, wydłuża się. Czas operacji jest też dłuższy dla transakcji kończących się commitem.

Zależność pomiędzy poziomiem izolacji oraz sposobu zapisu przedstawia poniższy wykres:



Wynika z niego, że w przypadku poziomu izolacji read commited lub read uncommited czas potrzebnyn na transakcję w tabeli nie zawierającej klucza głównego był najdłuższy dla wszystkich przypadków.

Inaczej sprawa ma się w przypadku poziomu izolacji repeatable read – w tym przypadku najdłużej wykonywaną transakcją była ta wykonywana z autoikrementującą się wartością klucza głównego (szczególnie przy nawiązywaniu każdorazowo nowego połączenia).