

VILNIUS TECH UNIVERSITETAS

Fundamentinių mokslų fakultetas

Sharat Nair

FINfu-21

Basics of Distributed Systems

Lab 2

Lecturer: Artūr Radzivilov

Vilnius, 2024

Contents

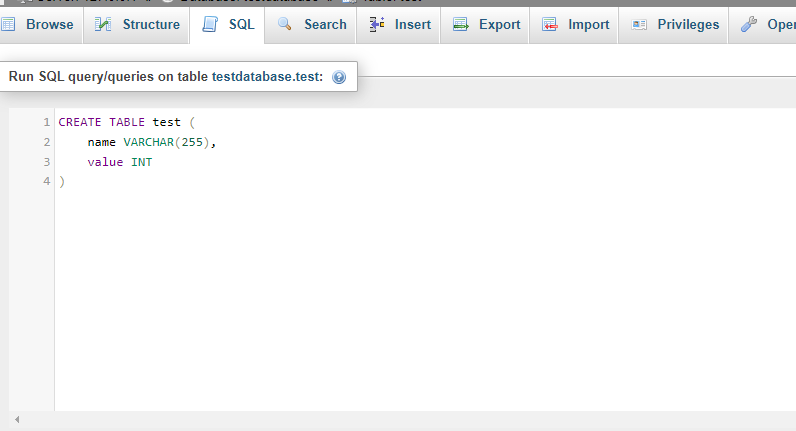
[Test Steps 3](#_Toc162081436)

[Results 10](#_Toc162081437)

# Test Steps

1. First step was to install XAMPP for using MySQL for our database from <https://www.apachefriends.org/> and starting apache and MySQL server.
2. Once the server is ran we start the process of creating tables.

Table 1: Non indexed

* First we create a table by going to MySQL and creating a database named `testdatabase`
* Then in the SQL section we use this command to create the table:
* 
* This creates a table with the name test. Now we can start the process of putting data into the table
* For inputting the data we can create a python script that can
  + Input in 100000 in one transaction
  + Input in 100000 separate transaction
  + Input in batch of 1000 entries to a total of 100000 entries
* We can do this using the below script:

import random

import string

import mysql.connector

def generate\_random\_string(length=10):

    """Generate a random string of given length."""

    return ''.join(random.choices(string.ascii\_letters + string.digits, k=length))

def insert\_data\_in\_one\_transaction(connection, num\_entries):

    """Insert data into the 'test' table in one transaction."""

    cursor = connection.cursor()

    try:

        # Begin the transaction

        cursor.execute("START TRANSACTION")

        for \_ in range(num\_entries):

            name = generate\_random\_string()

            value = random.randint(1, 1000)

            sql = "INSERT INTO test (name, value) VALUES (%s, %s)"

            cursor.execute(sql, (name, value))

        # Commit the transaction

        cursor.execute("COMMIT")

    except mysql.connector.Error as error:

        # Rollback the transaction if an error occurs

        connection.rollback()

        print("Transaction rolled back due to error:", error)

    finally:

        cursor.close()

def insert\_data\_in\_separate\_transactions(connection, num\_entries):

    """Insert data into the 'test' table in separate transactions."""

    cursor = connection.cursor()

    try:

        for \_ in range(num\_entries):

            # Begin a new transaction for each record

            cursor.execute("START TRANSACTION")

            name = generate\_random\_string()

            value = random.randint(1, 1000)

            sql = "INSERT INTO test (name, value) VALUES (%s, %s)"

            cursor.execute(sql, (name, value))

            # Commit the transaction

            cursor.execute("COMMIT")

    except mysql.connector.Error as error:

        # Rollback the transaction if an error occurs

        connection.rollback()

        print("Transaction rolled back due to error:", error)

    finally:

        cursor.close()

def insert\_data\_in\_batches(connection, num\_entries, batch\_size):

    """Insert data into the 'test' table in separate transactions with batching."""

    cursor = connection.cursor()

    try:

        for batch\_num in range(num\_entries // batch\_size):

            # Begin a new transaction for each batch

            cursor.execute("START TRANSACTION")

            for \_ in range(batch\_size):

                name = generate\_random\_string()

                value = random.randint(1, 1000)

                sql = "INSERT INTO test\_partitioned (name, value) VALUES (%s, %s)"

                cursor.execute(sql, (name, value))

            # Commit the transaction

            cursor.execute("COMMIT")

    except mysql.connector.Error as error:

        # Rollback the transaction if an error occurs

        connection.rollback()

        print("Transaction rolled back due to error:", error)

    finally:

        cursor.close()

def main():

    # Connect to the MySQL server

    connection = mysql.connector.connect(

        host="localhost",

        user="root",

        password="",  # Use your MySQL root password

        database="testdatabase"

    )

    # Number of entries to insert

    num\_entries = 100000

    batch\_size = 1000  # Batch size for batching

    # Insert data in one transaction

    insert\_data\_in\_one\_transaction(connection, num\_entries)

    # Insert data in separate transactions

    insert\_data\_in\_separate\_transactions(connection, num\_entries)

    # Insert data in separate transactions with batching

    insert\_data\_in\_batches(connection, num\_entries, batch\_size)

    # Close the connection

    connection.close()

    print(f"{3 \* num\_entries} entries inserted successfully.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* Once the program runs we get 300000 entries inserted successfully.
* For the next step we repeat the same process but this time we add some code to simulate a connection error.
* For step 3 we have to simulate a connection error, to do this we can modify our python script:

import random

import string

import mysql.connector

def get\_initial\_record\_count(connection):

    cursor = connection.cursor()

    cursor.execute("SELECT COUNT(\*) FROM test2")

    count = cursor.fetchone()[0]

    cursor.close()

    return count

def generate\_random\_string(length=10):

    """Generate a random string of given length."""

    return ''.join(random.choices(string.ascii\_letters + string.digits, k=length))

def perform\_insertion\_with\_connection\_error\_test(connection, num\_entries, batch\_size=None):

    initial\_count = get\_initial\_record\_count(connection)

    cursor = connection.cursor()

    try:

        for i in range(num\_entries):

            if batch\_size and i % batch\_size == 0:

                connection.commit()  # Commit the batch

                connection.start\_transaction()  # Start a new transaction

            name = generate\_random\_string()

            value = random.randint(1, 1000)

            cursor.execute("INSERT INTO test2 (name, value) VALUES (%s, %s)", (name, value))

            # Simulate connection error after every 1000 insertions

            if i % 1000 == 0:

                # Simulate connection error (e.g., by stopping MySQL service)

                print("Simulating connection error...")

                raise mysql.connector.Error("Simulated connection error")

    except mysql.connector.Error as e:

        print("Error occurred:", e)

        connection.rollback()  # Rollback the transaction in case of error

    finally:

        # Re-establish connection

        connection.reconnect()

        # Get the count of records after re-establishing connection

        final\_count = get\_initial\_record\_count(connection)

        print("Initial count:", initial\_count)

        print("Final count:", final\_count)

        print("Records inserted during interrupted process:", final\_count - initial\_count)

        cursor.close()

# Call the function with appropriate parameters for each case

connection = mysql.connector.connect(host="localhost", user="root", password="", database="testdatabase")

perform\_insertion\_with\_connection\_error\_test(connection, num\_entries=10000)  # One transaction

perform\_insertion\_with\_connection\_error\_test(connection, num\_entries=10000, batch\_size=1000)  # Separate transactions in batches

perform\_insertion\_with\_connection\_error\_test(connection, num\_entries=10000, batch\_size=1)  # Separate transactions

* Upon running this program we see the following output:  
  *Simulating connection error...*

*Error occurred: Simulated connection error*

*Initial count: 400000*

*Final count: 400000*

*Records inserted during interrupted process: 0*

*Simulating connection error...*

*Error occurred: Simulated connection error*

*Initial count: 400000*

*Final count: 400000*

*Records inserted during interrupted process: 0*

*Simulating connection error...*

*Error occurred: Simulated connection error*

*Initial count: 400000*

*Final count: 400000*

*Records inserted during interrupted process: 0*

* The output suggests that no entries are added when encountering a connection error in MySQL.
* For performing data sampling performance we can run the following SQL queries for each table:
  + *EXPLAIN SELECT COUNT(\*) FROM test, ANALYZE SELECT COUNT(\*) FROM test;*
  + *EXPLAIN COUNT(\*) FROM test, ANALYZE COUNT(\*) FROM test;*
  + *EXPLAIN SELECT \* FROM test WHERE value > 500, ANALYZE SELECT \* FROM test WHERE value > 500*
  + *EXPLAIN SELECT \* FROM test t1 CROSS JOIN test t2 LIMIT 100;* *EXPLAIN SELECT \* FROM test t1 CROSS JOIN test t2 LIMIT 100; (A limit was used here because running analyze on a dataset with this many results took a significantly long time).*
* The results for these test would be presented in the Results section.

Table 2: Indexed Data

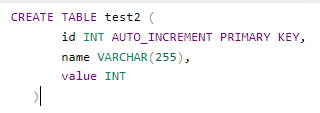
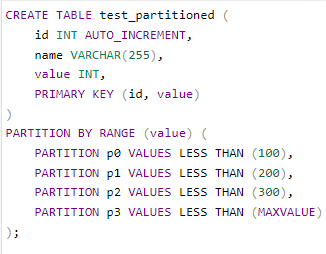
* Now we perform the same steps described above with a few changes, first we create a new table test2 with the following SQL query to use index:  
  
* Once this is created we repeat the data insertion process by using the same code as before but just writing test2 instead of test in the executed queries.
* And for the data sampling test we run the same SQL queries as above but replace test with test2.

Table 3: Partitioned Table

* We create a new table test\_partitioned using the following SQL query:  
  
* Once this is created the data insertion and analysis process is repeated by using the same code but by including test\_partitioned everywhere it’s required.

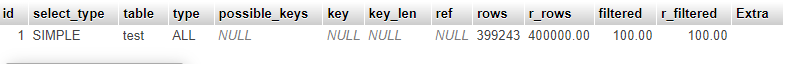
# Results

Table 1: test (Non-indexed):

* EXPLAIN SELECT COUNT(\*) FROM test;



* ANALYZE SELECT COUNT(\*) FROM test;



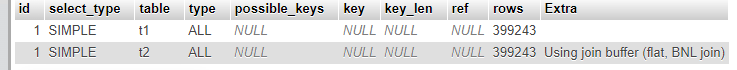
* EXPLAIN SELECT \* FROM test;



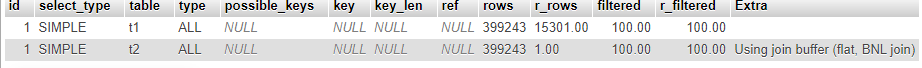
* ANALYZE SELECT \* FROM test;
* QUERY EXECUTION SPEED
* EXPLAIN SELECT \* FROM test WHERE value > 500
* ANALYZE SELECT \* FROM test WHERE value > 500;



* QUERY EXECUTION SPEED: 
* EXPLAIN SELECT \* FROM test t1 CROSS JOIN test t2 LIMIT 100;



* ANALYZE SELECT \* FROM test t1 CROSS JOIN test t2 LIMIT 100;



* QUERY EXECUTION SPEED: 

Table 2: test2 (indexed):

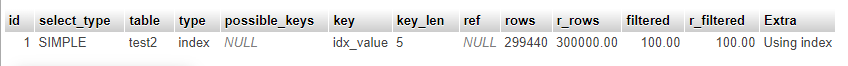
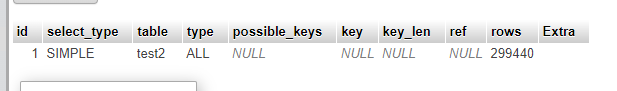
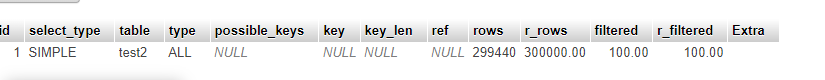
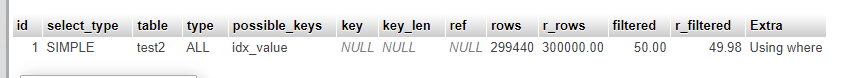
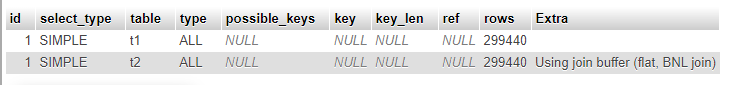
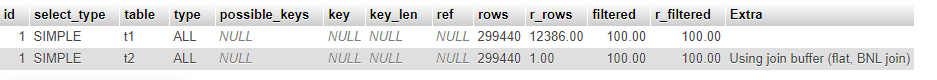
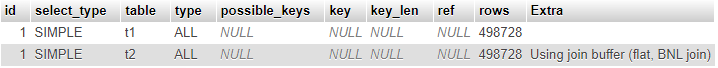
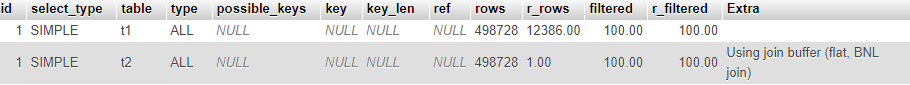
* EXPLAIN SELECT COUNT(\*) FROM test2;
* ANALYZE SELECT COUNT(\*) FROM test2; 
* EXPLAIN SELECT \* FROM test2; 
* ANALYZE SELECT \* FROM test2; 
* QUERY EXECUTION SPEED: 
* EXPLAIN SELECT \* FROM test2 WHERE value > 500; 
* ANALYZE SELECT \* FROM test2 WHERE value > 500; 
* QUERY EXECUTION SPEED: 
* EXPLAIN SELECT \* FROM test2 t1 CROSS JOIN test2 t2 LIMIT 100; 
* ANALYZE SELECT \* FROM test2 t1 CROSS JOIN test2 t2 LIMIT 100; 
* QUERY EXECUTION SPEED: 

Table 3: test\_partitioned (Partitioned table)

* EXPLAIN SELECT COUNT(\*) FROM test\_partitioned; 
* ANALYZE SELECT COUNT(\*) FROM test\_partitioned; 
* EXPLAIN SELECT \* FROM test\_partitioned; 
* ANALYZE SELECT \* FROM test\_partitioned; 
* QUERY EXECUTION SPEED: 
* EXPLAIN SELECT \* FROM test\_partitioned WHERE value > 500; 
* ANALYZE SELECT \* FROM test\_partitioned WHERE value > 500; 
* QUERY EXECUTION SPEED: 
* EXPLAIN SELECT \* FROM test\_partitioned t1 CROSS JOIN test\_partitioned t2 LIMIT 100; 
* ANALYZE SELECT \* FROM test\_partitioned t1 CROSS JOIN test\_partitioned t2 LIMIT 100; 
* QUERY EXECUTION SPEED: 