Name Atharva Burujpatte  
roll number: 05  
import concurrent.futures

import random

import time

# Sample table with rows as dictionaries (like a database table)

def generate\_table(n):

return [{"id": i, "value": random.randint(1, 100)} for i in range(n)]

# A mock "query" - filter and project

def process\_chunk(chunk):

result = []

for row in chunk:

if row["value"] > 50: # WHERE clause simulation

result.append({"id": row["id"]}) # SELECT id FROM ... WHERE value > 50

return result

# Split the data for parallel execution

def split\_data(data, num\_chunks):

chunk\_size = len(data) // num\_chunks

return [data[i:i + chunk\_size] for i in range(0, len(data), chunk\_size)]

# Parallel execution

def run\_parallel\_query(table, num\_workers=4):

chunks = split\_data(table, num\_workers)

result = []

with concurrent.futures.ThreadPoolExecutor(max\_workers=num\_workers) as executor:

futures = [executor.submit(process\_chunk, chunk) for chunk in chunks]

for future in concurrent.futures.as\_completed(futures):

result.extend(future.result())

return result

# Run comparison

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

table = generate\_table(1000000) # Simulated database table with 1 million rows

# Sequential

start = time.time()

sequential\_result = process\_chunk(table)

end = time.time()

print(f"Sequential time: {end - start:.4f} seconds. Rows matched: {len(sequential\_result)}")

# Parallel

start = time.time()

parallel\_result = run\_parallel\_query(table)

end = time.time()

print(f"Parallel time: {end - start:.4f} seconds. Rows matched: {len(parallel\_result)}")

# Check if both results match (they should)

print("Results match:", len(sequential\_result) == len(parallel\_result))

output:  
Sequential time: 0.2351 seconds. Rows matched: 494902

Parallel time: 0.0923 seconds. Rows matched: 494902

Results match: True