

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Công nghệ thông tin 1



BÁO CÁO BÀI TẬP LỚN

CƠ SỞ DỮ LIỆU PHÂN TÁN

GV hướng dẫn : KIM NGỌC BÁCH

Nhóm 16

Thành viên

Võ Thanh Huyền B22DCCN403

Nguyễn Việt Hoàng B22DCCN343

Lý Chí Công B22DCCN088

Lời Cảm Ơn

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến Thầy Kim Ngọc Bách, giảng viên bộ môn, người đã tận tình giảng dạy và truyền đạt những kiến thức quý báu về hệ quản trị cơ sở dữ liệu phân tán trong suốt học kỳ vừa qua.

Chính nhờ sự hướng dẫn rõ ràng, phương pháp giảng dạy khoa học cùng các ví dụ thực tế sinh động của Thầy, chúng em đã có được nền tảng vững chắc để hiểu và vận dụng được các khái niệm như phân mảnh dữ liệu, tổ chức bảng, cũng như kỹ năng thao tác với cơ sở dữ liệu bằng Python.

Đề tài bài tập lớn lần này không chỉ giúp em củng cố kiến thức đã học mà còn là cơ hội quý giá để em rèn luyện tư duy logic, khả năng lập trình, và tiếp cận gần hơn với các vấn đề thực tiễn trong ngành công nghệ thông tin.

Một lần nữa, chúng em xin chân thành cảm ơn Thầy và mong rằng sẽ tiếp tục được học hỏi thêm nhiều điều bổ ích từ Thầy trong các môn học tiếp theo.

Mục Lục

Lời Cảm Ơn.....	2
Mục Lục.....	2
I. Giới thiệu.....	3
1. Nội dung thực hiện.....	3
a, Bối cảnh và ý nghĩa thực tiễn.....	3
b, Mục tiêu bài tập.....	3
2. Phân chia công việc.....	3
II. Triển khai và giải quyết các vấn đề.....	4
1. Vấn đề đặt ra.....	4
2. Hướng giải quyết.....	4
a, Đối với vấn đề chung.....	4
b, Đối với đề bài xử lý ratings.dat.....	4
3. Các bước thực hiện.....	5
a, Tạo và kết nối cơ sở dữ liệu: (sử dụng MySql).....	5
b, Tải dữ liệu vào bảng chính ratings.....	6
c, Phân mảnh theo range.....	6
d, Phân mảnh theo round robin.....	7
e, Hàm roundrobininsert().....	7
f, Hàm rangeinsert().....	8
g, Bộ Kiểm Thử Testhelper.Py, Assignment1_testrangepartition.Py, Assignment2_testroundrobin.Py.....	9
III. Kết Luận.....	11
1.Những gì đã làm được:.....	11
2.Kiến thức và kỹ năng học được:.....	11
3.Khó khăn & Bài học rút ra:.....	12
4.Tổng kết:.....	12
IV. Tài liệu tham khảo.....	12

I. Giới thiệu

1. Nội dung thực hiện

a, Bối cảnh và ý nghĩa thực tiễn

Trong bối cảnh các hệ thống thông tin ngày càng xử lý khối lượng dữ liệu lớn và đa dạng, phân mảng dữ liệu (data partitioning) là một kỹ thuật quan trọng giúp tăng hiệu năng truy vấn, tối ưu lưu trữ và hỗ trợ mở rộng hệ thống. Các phương pháp phân mảng ngang — như phân mảng theo khoảng (range) và phân mảng theo vòng tròn (round robin) — thường được sử dụng trong các hệ thống cơ sở dữ liệu phân tán, hệ thống dữ liệu lớn (Big Data), hoặc các dịch vụ web quy mô lớn.

Thông qua đề tài này, chúng ta không chỉ hiểu lý thuyết về phân mảng mà còn trực tiếp triển khai và kiểm chứng các thuật toán phân mảng trên dữ liệu thực tế, từ đó nắm bắt bản chất của cơ chế tổ chức và xử lý dữ liệu hiệu quả trong cơ sở dữ liệu quan hệ.

b, Mục tiêu bài tập

Hiểu và thao tác với cơ sở dữ liệu PostgreSQL/MySQL:

- Tạo bảng, chèn dữ liệu, truy vấn dữ liệu.
- Quản lý kết nối và xử lý truy cập bằng Python.

Lập trình xử lý dữ liệu bằng Python:

- Đọc file, phân tích và chuẩn hóa dữ liệu.
- Viết hàm tải dữ liệu và phân mảng dữ liệu theo thuật toán.

Hiểu rõ thuật toán phân mảng ngang:

- Phân mảng theo khoảng (Range Partitioning): phân chia dữ liệu theo giá trị rating.
- Phân mảng vòng tròn (Round Robin Partitioning): chia đều dữ liệu theo thứ tự vào các bảng.

Triển khai hàm chèn động (Insert):

- Chèn bản ghi mới vào bảng chính và vào đúng phân mảng.
- Duy trì tính nhất quán và tính đúng đắn của việc phân mảng sau khi chèn.

2. Phân chia công việc

Họ và tên	MSV	Công việc	Trạng thái
Võ Thanh Huyền	B22DCCN403	<ul style="list-style-type: none">- Trưởng nhóm- Tìm hiểu, xử lý và đưa ra hướng giải quyết bài toán.- Tìm hiểu, cài đặt môi trường, dữ liệu và kết nối với CSDL.- Cài đặt, triển khai các chương trình chính.	Hoàn thành
Nguyễn Việt Hoàng	B22DCCN343	<ul style="list-style-type: none">- Tìm hiểu và xử lý phân mảng Range.	Hoàn thành
Lý Chí Công	B22DCCN088	<ul style="list-style-type: none">- Tìm hiểu và xử lý phân mảng Round-Robin	Hoàn thành

II. Triển khai và giải quyết các vấn đề

1. Vấn đề đặt ra

- Làm thế nào để một hệ thống cơ sở dữ liệu lớn có thể phân phối dữ liệu hợp lý vào các phân vùng khác nhau, nhằm tăng hiệu năng truy vấn và duy trì khả năng mở rộng khi dữ liệu không ngừng tăng lên?
- Bối cảnh trong đề bài:
 - + Dữ liệu lớn: Tệp ratings.dat chứa tới 10 triệu dòng, tương ứng với 72.000 người dùng và 10.000 phim.
 - + Mỗi dòng là một bản ghi đánh giá, với điểm đánh giá (rating) là thuộc tính mục tiêu được dùng để phân mảng.

2. Hướng giải quyết

a, Vấn đề chung

Phân mảnh dữ liệu là một chiến lược tổ chức dữ liệu trong hệ quản trị cơ sở dữ liệu (DBMS) nhằm:

- Chia nhỏ bảng lớn thành nhiều bảng nhỏ hơn (phân mảnh),
- Giảm lượng dữ liệu phải quét khi truy vấn,
- Tăng khả năng mở rộng (scale-out),
- Hỗ trợ truy cập song song (parallel query processing).

b, Xử lý ratings.dat

- Thực hiện phân mảnh với 2 phương pháp:

- + Phân mảnh theo khoảng giá trị (Range Partitioning):

Ý tưởng :

- Chia dữ liệu thành các vùng dựa trên khoảng giá trị của Rating.
 - Mỗi phân mảnh (bảng con) sẽ chứa các bản ghi có Rating nằm trong một khoảng cụ thể.

Ví dụ:

Với N = 3, chia thành:

range_part0: rating $\in [0, 1.67]$

range_part1: rating $\in (1.67, 3.34]$

range_part2: rating $\in (3.34, 5]$

Mục đích: Truy vấn như “tìm tất cả đánh giá > 4.5 ” sẽ chỉ cần quét đúng 1 phân mảnh, thay vì toàn bộ bảng.

- + Phân mảnh theo vòng tròn (Round Robin Partitioning):

Ý tưởng :

- Phân phối bản ghi tuần tự và đều nhau vào các bảng con.
 - Bản ghi thứ 0 \rightarrow part0, thứ 1 \rightarrow part1, thứ 2 \rightarrow part2, ... rồi quay vòng lại.

Mục đích:

- Đảm bảo cân bằng tải giữa các phân mảnh.
 - Hữu ích khi không có quy luật rõ ràng nào để chia dữ liệu theo giá trị (ví dụ như Rating phân bố không đều).
- Cập nhật dữ liệu :
- + Range_Insert(): xác định khoảng phù hợp rồi chèn vào đúng bảng con.
 - + RoundRobin_Insert(): sử dụng chỉ số vòng tròn hiện tại để chèn lần lượt vào các bảng con.
- Đánh giá chung :

Lợi ích	Phân tích
Truy vấn nhanh hơn	Mỗi truy vấn chỉ cần đọc từ 1 hoặc vài phân mảnh thay vì cả bảng lớn.
Quản lý tốt hơn	Dễ bảo trì, dễ tối ưu chỉ mục cho từng phân mảnh.
Truy cập song song	Các truy vấn có thể chạy trên nhiều phân mảnh cùng lúc → tăng tốc độ.
Dễ mở rộng	Có thể phân tán phân mảnh sang nhiều máy chủ khác nhau.

3. Các bước thực hiện

a, Tạo và kết nối cơ sở dữ liệu: (sử dụng MySql)

- Dùng create_db(dbname) để kiểm tra và tạo database nếu chưa tồn tại.

```
● ● ●  
1 def create_db(dbname):  
2     """  
3     Tạo cơ sở dữ liệu nếu chưa tồn tại.  
4     """  
5     con = getopenconnection(dbname='mysql')  
6     cur = con.cursor()  
7  
8     # Kiểm tra xem cơ sở dữ liệu đã tồn tại chưa  
9     cur.execute("SELECT COUNT(*) FROM information_schema.schemata WHERE schema_name = %s", (dbname,))  
10    count = cur.fetchone()[0]  
11    if count == 0:  
12        cur.execute('CREATE DATABASE %s' % dbname)  
13        con.commit()  
14    else:  
15        print('A database named {} already exists'.format(dbname))  
16  
17    cur.close()  
18    con.close()
```

- Kết nối CSDL thông qua getopenconnection() được dùng ở mọi hàm.

```
● ● ●  
1 def getopenconnection(user='root', password='291004', dbname='mysql'): 2     return pymysql.connect(host='localhost', user=user, password=password, database=dbname)
```

b, Tải dữ liệu vào bảng chính ratings

- Hàm: loadratings(ratingstablename, ratingsfilepath, openconnection)
- Mục tiêu: Đọc dữ liệu từ file .dat và chèn vào bảng ratings.

```

1  def loadratings(ratingstablename, ratingsfilepath, openconnection):
2      """
3          Function to load data in @ratingsfilepath file to a table called @ratingstablename.
4      """
5      delete_db(DATABASE_NAME)
6      create_db(DATABASE_NAME)
7      con = openconnection
8      cur = con.cursor()
9      # Tạo bảng với các cột tạm thời để load dữ liệu
10     cur.execute(
11         "CREATE TABLE %s (userid INTEGER, movieid INTEGER, rating FLOAT, timestamp BIGINT)" % ratingstablename)
12
13     insert_query = f"INSERT INTO {ratingstablename} (userid, movieid, rating, timestamp) VALUES (%s, %s, %s, %s)"
14
15     with open(ratingsfilepath, 'r') as f:
16         data = []
17         for line in f:
18             values = line.strip().split '::'
19             if len(values) != 4:
20                 print(f"Skipping invalid line: {line.strip()} (expected 4 values, got {len(values)})")
21                 continue
22             try:
23                 userid = int(values[0])
24                 movieid = int(values[1])
25                 rating = float(values[2])
26                 timestamp = int(values[3])
27                 data.append((userid, movieid, rating, timestamp))
28             except ValueError as e:
29                 print(f"Skipping invalid line: {line.strip()} (error: {e})")
30                 continue
31             if not data:
32                 raise ValueError("No valid data found in the file.")
33             cur.executemany(insert_query, data)
34
35             cur.execute(
36                 "ALTER TABLE %s DROP COLUMN timestamp" % ratingstablename)
37
38             con.commit()
39             cur.close()
40

```

Cách thực hiện:

- Đọc từng dòng từ file ratings.dat → lấy 4 trường: userid, movieid, rating, timestamp.
- Xử lý bằng Python (không dùng copy_from()), sau đó dùng executemany() để chèn hàng loạt vào MySQL.
- Cuối cùng, xóa cột timestamp để chỉ giữ 3 cột cần thiết.

c, Phân mảng theo range

- Hàm: rangepartition(ratingstablename, numberofpartitions, openconnection)
- Mục tiêu: Tạo N bảng con chứa các bản ghi phân theo khoảng giá trị rating.

```

1  def rangepartition(ratingstablename, numberofpartitions, openconnection):
2      """
3          Function to create partitions of main table based on range of ratings.
4      """
5      con = openconnection
6      cur = con.cursor()
7      delta = 5.0 / numberofpartitions
8      RANGE_TABLE_PREFIX = 'range_part'
9
10     for i in range(0, numberofpartitions):
11         minRange = i * delta
12         maxRange = minRange + delta
13         table_name = RANGE_TABLE_PREFIX + str(i)
14         cur.execute("CREATE TABLE %s (userid INTEGER, movieid INTEGER, rating FLOAT)" % table_name)
15         if i == 0:
16             cur.execute(
17                 "INSERT INTO %s (userid, movieid, rating) SELECT userid, movieid, rating FROM %s WHERE rating >= %s AND rating <= %s" % (
18                     table_name, ratingstablename, minRange, maxRange))
19         else:
20             cur.execute(
21                 "INSERT INTO %s (userid, movieid, rating) SELECT userid, movieid, rating FROM %s WHERE rating > %s AND rating <= %s" % (
22                     table_name, ratingstablename, minRange, maxRange))
23
24     con.commit()
25     cur.close()

```

Cách thực hiện:

- + Chia khoảng rating từ 0 đến 5 thành N đoạn bằng nhau ($\text{delta} = 5 / N$).
- + Tạo bảng range_part0, range_part1, ..., range_partN-1.
- + Chèn dữ liệu bằng câu lệnh SQL `SELECT ... WHERE rating >= ... AND rating <=`

d, Phân mảnh theo round robin

- Hàm: `roundrobinpartition(ratingstablename, numberofpartitions, openconnection)`
- Mục tiêu: Phân phối bản ghi đều đặn theo thứ tự vào các bảng phân mảnh.

```
1 def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
2     """
3         Function to create partitions of main table using round robin approach.
4     """
5     con = openconnection
6     cur = con.cursor()
7     RROBIN_TABLE_PREFIX = 'rrobin_part'
8
9     for i in range(0, numberofpartitions):
10        table_name = RROBIN_TABLE_PREFIX + str(i)
11        cur.execute("CREATE TABLE %s (userid INTEGER, movieid INTEGER, rating FLOAT)" % table_name)
12        cur.execute(
13            "INSERT INTO %s (userid, movieid, rating) SELECT userid, movieid, rating FROM "
14            "(SELECT userid, movieid, rating, ROW_NUMBER() OVER () AS rnum FROM %s) AS temp "
15            "WHERE (rnum-1) %% %s = %s" % (table_name, ratingstablename, numberofpartitions, i)
16        )
17
18    con.commit()
19    cur.close()
```

Cách làm:

- + Tạo các bảng rrobin_part0, ..., rrobin_partN-1.
- + Dùng ROW_NUMBER() OVER () để đánh số bản ghi, rồi áp dụng MOD(index, N) để phân mảnh tuần tự.

e, Hàm roundrobininsert()

- Mục tiêu: Khi chèn bản ghi mới, xác định đúng bảng rrobin_partX theo vòng tuần tự.

```
1 def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
2     """
3         Function to insert a new row into the main table and specific partition based on round robin approach.
4     """
5     con = openconnection
6     cur = con.cursor()
7     RROBIN_TABLE_PREFIX = 'rrobin_part'
8
9     # Chèn vào bảng chính
10    cur.execute(
11        "INSERT INTO %s (userid, movieid, rating) VALUES (%s, %s, %s)" % (ratingstablename, userid, itemid, rating))
12
13    # Đếm số dòng trong bảng chính để tính partition
14    cur.execute("SELECT COUNT(*) FROM %s" % ratingstablename)
15    total_rows = cur.fetchone()[0]
16
17    # Đếm số partition
18    numberofpartitions = count_partitions(RROBIN_TABLE_PREFIX, openconnection)
19    index = (total_rows - 1) % numberofpartitions
20    table_name = RROBIN_TABLE_PREFIX + str(index)
21
22    # Chèn vào partition tương ứng
23    cur.execute("INSERT INTO %s (userid, movieid, rating) VALUES (%s, %s, %s)" % (table_name, userid, itemid, rating))
24
25    con.commit()
26    cur.close()
```

Cách làm:

- + Chèn vào bảng chính ratings.
- + Đếm tổng số dòng hiện tại → xác định bảng tiếp theo dựa trên $(\text{total_rows} - 1) \% N$.
- + Chèn tiếp vào bảng rrobin_partX.

f, Hàm rangeinsert()

- Mục tiêu: Khi chèn bản ghi mới, xác định đúng bảng range_partX theo giá trị rating.

```

1 def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
2     """
3         Function to insert a new row into the main table and specific partition based on range rating.
4     """
5     con = openconnection
6     cur = con.cursor()
7     RANGE_TABLE_PREFIX = 'range_part'
8
9     # Đếm số partition
10    numberofpartitions = count_partitions(RANGE_TABLE_PREFIX, openconnection)
11    delta = 5.0 / numberofpartitions
12    index = int(rating / delta)
13    if rating % delta == 0 and index != 0:
14        index = index - 1
15    table_name = RANGE_TABLE_PREFIX + str(index)
16
17    # Chèn vào bảng chính
18    cur.execute(
19        "INSERT INTO %s (userid, movieid, rating) VALUES (%s, %s, %s)" % (ratingstablename, userid, itemid, rating))
20
21    # Chèn vào partition tương ứng
22    cur.execute("INSERT INTO %s (userid, movieid, rating) VALUES (%s, %s, %s)" % (table_name, userid, itemid, rating))
23
24    con.commit()
25    cur.close()

```

- Cách làm:

- + Chèn vào bảng chính.
- + Tính toán index bằng rating / delta.
- + Nếu rating nằm ngay tại biên thì điều chỉnh index - 1 để không trùng lặp.

g, Bộ Kiểm Thử Testhelper.Py, Assignment1_testrangepartition.Py, Assignment2_testroundrobin.Py

>Assignment1_testrangepartition.py:

- Mục tiêu: Kiểm tra các hàm xử lý phân mảnh theo khoảng Range.
- Gồm 3 phần kiểm thử chính:
 - + **testloadratings()** :
 - Tác dụng : Kiểm tra hàm loadratings trong Interface.py để đảm bảo nó tải đúng 50 hàng dữ liệu từ file small_ratings.dat vào bảng ratings

```
1 [result, e] = testHelper.testloadratings(MyAssignment, RATINGS_TABLE, INPUT_FILE_PATH, conn, ACTUAL_ROWS_IN_INPUT_FILE)
2 if result:
3     print("loadratings function pass!")
4 else:
5     print("loadratings function fail!")
```

```
1 def testloadratings(MyAssignment, ratingstablename, filepath, openconnection, rowsininpfile):
2     try:
3         MyAssignment.loadratings(ratingstablename, filepath, openconnection)
4         with openconnection.cursor() as cur:
5             cur.execute('SELECT COUNT(*) FROM %s' % ratingstablename)
6             count = int(cur.fetchone()[0])
7             if count != rowsininpfile:
8                 raise Exception(
9                     'Expected %s rows, but %s rows in \'%s\' table' % (rowsininpfile, count, ratingstablename))
10    except Exception as e:
11        traceback.print_exc()
12    return [False, e]
13 return [True, None]
```

- Cách hoạt động :

> Hàm testloadratings() gọi hàm loadrating() trong Interface.py.

> Sau đó, nó thực thi SELECT COUNT(*) FROM ratings để đếm số hàng trong bảng ratings.

> Nếu số hàng khác 50 thì hàm ném ngoại lệ.

> Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

- + **testrangepartition()**

- Tác dụng : Kiểm tra hàm rangepartition trong Interface.py để đảm bảo nó tạo 5 bảng phân vùng range (range_part0 đến range_part4) và phân phối dữ liệu đúng.

```
1 [result, e] = testHelper.testrangepartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
2 if result:
3     print("rangepartition function pass!")
4 else:
5     print("rangepartition function fail!")
```

```
1 def testrangepartition(MyAssignment, ratingstablename, n, openconnection, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
2     try:
3         MyAssignment.rangepartition(ratingstablename, n, openconnection)
4         testrangeandrobinpartitioning(n, openconnection, RANGE_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
5         testEachRangePartition(ratingstablename, n, openconnection, RANGE_TABLE_PREFIX)
6         return [True, None]
7     except Exception as e:
8         traceback.print_exc()
9     return [False, e]
```

- Cách hoạt động :

> Hàm testrangepartition() gọi hàm rangepartition() trong Interface.py.

```
1 def testrangeandrobinpartitioning(n, openconnection, rangepartitiontableprefix, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
2     with openconnection.cursor() as cur:
3         if not isinstance(n, int) or n < 0:
4             checkpartitioncount(cur, 0, rangepartitiontableprefix)
5         else:
6             checkpartitioncount(cur, n, rangepartitiontableprefix)
7             count = totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex)
8             if count < ACTUAL_ROWS_IN_INPUT_FILE:
9                 raise Exception(
10                     "Completeness property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
11                         ACTUAL_ROWS_IN_INPUT_FILE, count))
12             if count > ACTUAL_ROWS_IN_INPUT_FILE:
13                 raise Exception(
14                     "Disjointness property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
15                         ACTUAL_ROWS_IN_INPUT_FILE, count))
16             if count != ACTUAL_ROWS_IN_INPUT_FILE:
17                 raise Exception(
18                     "Reconstruction property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
19                         ACTUAL_ROWS_IN_INPUT_FILE, count))
```

Hàm kiểm tra tính toàn vẹn và không trùng lặp của phân vùng.

> Sau đó, nó gọi testrangeandrobinpartitioning() để kiểm tra:

- Số bảng phân vùng phải là 5 (bằng checkpartitioncount).

- Tổng số hàng trong các bảng phân vùng (range_part0 đến range_part4) phải bằng 50, đảm bảo tính toàn vẹn và không trùng lặp.

```

1 def checkpartitioncount(cursor, expectedpartitions, prefix):
2     cursor.execute(
3         "SELECT COUNT(*) FROM information_schema.tables WHERE table_schema = DATABASE() AND table_name LIKE %s", (prefix + '%',))
4     count = int(cursor.fetchone()[0])
5     if count != expectedpartitions:
6         raise Exception(
7             'Range partitioning not done properly. Expected %s table(s) but found %s table(s)' % (
8                 expectedpartitions, count))

```

Hàm kiểm tra xem có đúng số lượng bảng phân vùng được tạo không bằng cách truy vấn `information_schema.tables` và đếm các bảng có tiền tố `prefix`.

```

1 def totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex):
2     selects = []
3     for i in range(partitionstartindex, n + partitionstartindex):
4         selects.append('SELECT * FROM %s%%' % (rangepartitiontableprefix, i))
5     cur.execute('SELECT COUNT(*) FROM (%s) AS T' % ' UNION ALL '.join(selects))
6     count = int(cur.fetchone()[0])
7     return count

```

Hàm tính tổng số hàng trong tất cả các bảng phân vùng, sử dụng câu lệnh `UNION ALL` để gộp dữ liệu từ các bảng và đếm. Đảm bảo tính toàn vẹn (*completeness*) và tính không trùng lặp (*disjointness*) của phân vùng.

> Gọi `testEachRangePartition()` để kiểm tra số hàng trong mỗi bảng phân vùng khớp với số hàng dự kiến từ `getCountrangepartition()`.

```

1 def testEachRangePartition(ratingstablename, n, openconnection, rangepartitiontableprefix):
2     countList = getCountrangepartition(ratingstablename, n, openconnection)
3     cur = openconnection.cursor()
4     for i in range(0, n):
5         cur.execute("SELECT COUNT(*) FROM %s%%" % (rangepartitiontableprefix, i))
6         count = int(cur.fetchone()[0])
7         if count != countList[i]:
8             raise Exception("%s%% has %s rows while the correct number should be %s" % (
9                 rangepartitiontableprefix, i, count, countList[i]))

```

Hàm so sánh số hàng thực tế trong mỗi bảng phân vùng `range` với số hàng dự kiến từ `getCountrangepartition`.

```

1 def getCountrangepartition(ratingstablename, numberofpartitions, openconnection):
2     """
3     Đếm số hàng trong mỗi phân vùng range partition.
4     """
5     cur = openconnection.cursor()
6     countlist = []
7     interval = 5.0 / numberofpartitions
8     cur.execute("SELECT COUNT(*) FROM %s WHERE rating >= %s AND rating <= %s" % (ratingstablename, 0, interval))
9     countlist.append(int(cur.fetchone()[0]))
10
11    lowerbound = interval
12    for i in range(1, numberofpartitions):
13        cur.execute("SELECT COUNT(*) FROM %s WHERE rating > %s AND rating <= %s" % (ratingstablename, lowerbound, lowerbound + interval))
14        lowerbound += interval
15        countList.append(int(cur.fetchone()[0]))
16
17    cur.close()
18    return countList
19

```

Hàm này đếm số hàng trong mỗi phân vùng range, dựa trên giá trị rating. Rating nằm trong khoảng 0-5, nó chia khoảng này thành numberofpartitions phần bằng nhau. Hàm sử dụng các câu lệnh SQL như SELECT COUNT() FROM %s WHERE rating >= %s AND rating <= %s để đếm. Kết quả là một danh sách countList chứa số hàng trong mỗi phân vùng.*

> Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

+ testrangeinsert()

```

1 def testrangeinsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):
2     try:
3         expectedtablename = RANGE_TABLE_PREFIX + expectedtableindex
4         MyAssignment.rangeinsert(ratingstablename, userid, itemid, rating, openconnection)
5         if not testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
6             raise Exception(
7                 'Range insert failed! Could not find (%s, %s, %s) tuple in %s table' % (userid, itemid, rating, expectedtablename))
8     except Exception as e:
9         traceback.print_exc()
10    return [False, e]
11    return [True, None]

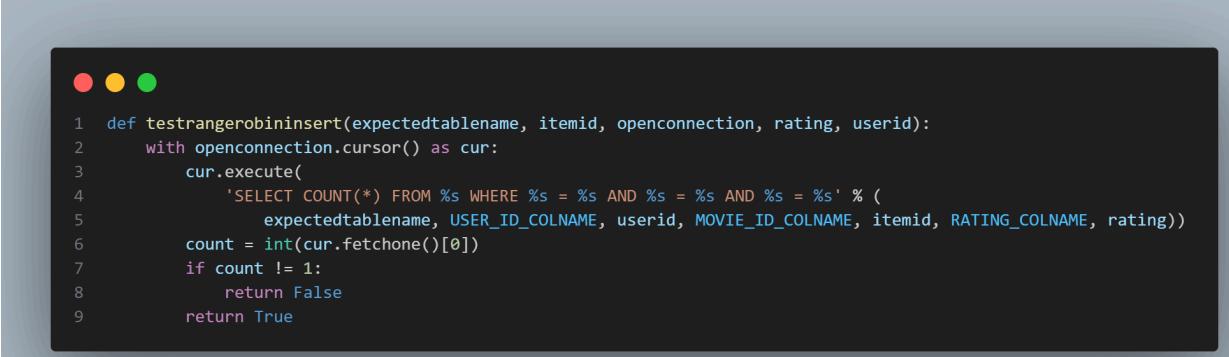
```

- Tác dụng : Kiểm tra hàm rangeinsert trong Interface.py để đảm bảo nó chèn một hàng dữ liệu mới (userid=100, movieid=2, rating=3) vào bảng ratings và bảng phân vùng range_part2 (vì rating 3 thuộc khoảng >2-3).

- Cách hoạt động:

> Hàm testrangeinsert() gọi rangeinsert() trong Interface.py.

> Sau đó, gọi teststrangerobininsert để kiểm tra xem hàng (100, 2, 3) có trong bảng range_part2 hay không. Nếu không tìm thấy hàng, ném ngoại lệ.



```
1 def teststrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
2     with openconnection.cursor() as cur:
3         cur.execute(
4             'SELECT COUNT(*) FROM %s WHERE %s = %s AND %s = %s AND %s = %s' % (
5                 expectedtablename, USER_ID_COLNAME, userid, MOVIE_ID_COLNAME, itemid, RATING_COLNAME, rating))
6         count = int(cur.fetchone()[0])
7         if count != 1:
8             return False
9     return True
```

Hàm này kiểm tra xem một hàng đã được chèn vào bảng phân vùng đúng chưa, bằng cách đếm số hàng khớp với userid, itemid, và rating trong bảng expectedtablename.

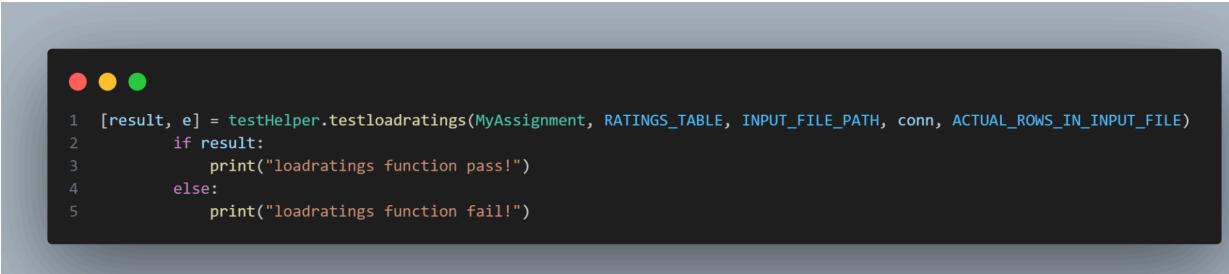
> Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

➤Assignment2_testroundrobin.py

- Mục tiêu: Kiểm tra các hàm xử lý phân mảnh theo vòng tròn.
- Gồm 3 phần kiểm thử chính:

- + **testloadratings():**

Tương tự Assignment1_testrangepartition.py, câu lệnh có tác dụng kiểm tra hàm loadratings trong Interface.py để đảm bảo nó tải đúng 50 hàng dữ liệu từ file small_ratings.dat vào bảng ratings.



```
1 [result, e] = testHelper.testloadratings(MyAssignment, RATINGS_TABLE, INPUT_FILE_PATH, conn, ACTUAL_ROWS_IN_INPUT_FILE)
2     if result:
3         print("loadratings function pass!")
4     else:
5         print("loadratings function fail!")
```

- + **testroundrobinpartition():**

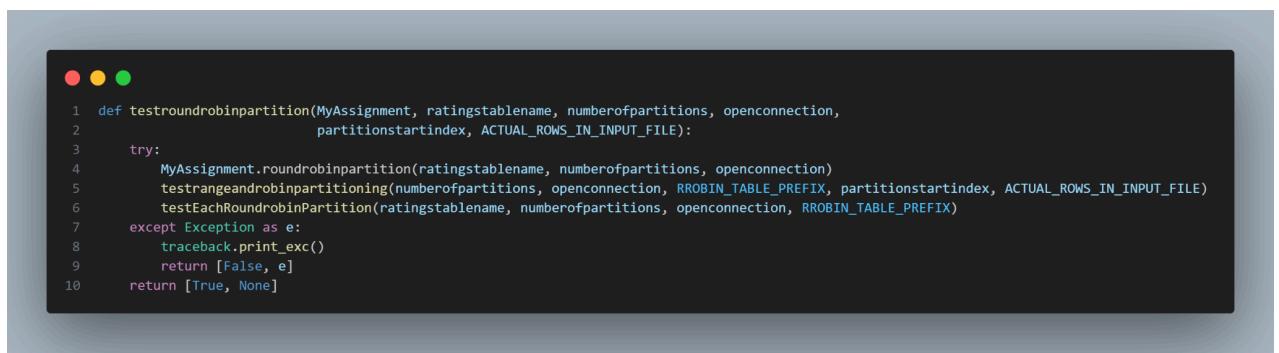
- Tác dụng : Kiểm tra hàm roundrobinpartition() trong Interface.py để đảm bảo nó tạo 5 bảng phân vùng round-robin (rrobin_part0 đến rrobin_part4) và phân phối dữ liệu đúng theo phương pháp Round-Robin.



```

1 [result, e] = testHelper.testroundrobinpartition(MyAssignment, RATINGS_TABLE, 5, conn, 0, ACTUAL_ROWS_IN_INPUT_FILE)
2     if result:
3         print("roundrobinpartition function pass!")
4     else:
5         print("roundrobinpartition function fail")

```



```

1 def testroundrobinpartition(MyAssignment, ratingstablename, numberofpartitions, openconnection,
2                             partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
3     try:
4         MyAssignment.roundrobinpartition(ratingstablename, numberofpartitions, openconnection)
5         testrangeandrobinpartitioning(numberofpartitions, openconnection, RROBIN_TABLE_PREFIX, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE)
6         testEachRoundRobinPartition(ratingstablename, numberofpartitions, openconnection, RROBIN_TABLE_PREFIX)
7     except Exception as e:
8         traceback.print_exc()
9     return [False, e]
10    return [True, None]

```

- Cách hoạt động :

> Hàm testroundrobinpartition() gọi roundrobinpartition() để tạo 5 bảng rrobin_part0 → rrobin_part4.

> Sau đó, nó gọi testrangeandrobinpartitioning để kiểm tra:

- Số bảng phân vùng phải là 5.

- Tổng số hàng trong các bảng phân vùng phải bằng 50, đảm bảo tính toàn vẹn và không trùng lặp.

```
● ● ●
1 def testrangeandrobinpartitioning(n, openconnection, rangepartitiontableprefix, partitionstartindex, ACTUAL_ROWS_IN_INPUT_FILE):
2     with openconnection.cursor() as cur:
3         if not isinstance(n, int) or n < 0:
4             checkpartitioncount(cur, 0, rangepartitiontableprefix)
5         else:
6             checkpartitioncount(cur, n, rangepartitiontableprefix)
7             count = totalrowsinallpartitions(cur, n, rangepartitiontableprefix, partitionstartindex)
8             if count < ACTUAL_ROWS_IN_INPUT_FILE:
9                 raise Exception(
10                     "Completeness property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
11                         ACTUAL_ROWS_IN_INPUT_FILE, count))
12             if count > ACTUAL_ROWS_IN_INPUT_FILE:
13                 raise Exception(
14                     "Disjointness property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
15                         ACTUAL_ROWS_IN_INPUT_FILE, count))
16             if count != ACTUAL_ROWS_IN_INPUT_FILE:
17                 raise Exception(
18                     "Reconstruction property of Partitioning failed. Expected %s rows after merging all tables, but found %s rows" % (
19                         ACTUAL_ROWS_IN_INPUT_FILE, count))
```

Hàm kiểm tra tính toàn vẹn và không trùng lặp của phân vùng.

```
● ● ●
1 def checkpartitioncount(cursor, expectedpartitions, prefix):
2     cursor.execute(
3         "SELECT COUNT(*) FROM information_schema.tables WHERE table_schema = DATABASE() AND table_name LIKE %s", (prefix + '%',))
4     count = int(cursor.fetchone()[0])
5     if count != expectedpartitions:
6         raise Exception(
7             'Range partitioning not done properly. Expected %s table(s) but found %s table(s)' % (
8                 expectedpartitions, count))
```

Hàm kiểm tra xem có đúng số lượng bảng phân vùng được tạo không bằng cách truy vấn information_schema.tables và đếm các bảng có tiền tố prefix.

> Gọi testEachRoundRobinPartition để kiểm tra số hàng trong mỗi bảng phân vùng khớp với số hàng dự kiến từ getCountroundrobinpartition.

```
● ● ●
1 def testEachRoundRobinPartition(ratingstablename, n, openconnection, roundrobinpartitiontableprefix):
2     countList = getCountroundrobinpartition(ratingstablename, n, openconnection)
3     cur = openconnection.cursor()
4     for i in range(0, n):
5         cur.execute("SELECT COUNT(*) FROM %s%s" % (roundrobinpartitiontableprefix, i))
6         count = cur.fetchone()[0]
7         if count != countList[i]:
8             raise Exception("%s has %s rows while the correct number should be %s" % (
9                 roundrobinpartitiontableprefix, i, count, countList[i]))
```

Hàm so sánh số hàng trong mỗi bảng phân vùng round-robin với kết quả từ getCountroundrobinpartition

```

1 def getCountroundrobinpartition(ratingstablename, numberofpartitions, openconnection):
2     cur = openconnection.cursor()
3     countlist = []
4     for i in range(0, numberofpartitions):
5         cur.execute(
6             "SELECT COUNT(*) FROM (SELECT *, ROW_NUMBER() OVER () AS row_num FROM %s) AS temp WHERE (row_num-1) %% %s = %s" % (
7                 ratingstablename, numberofpartitions, i))
8     countList.append(int(cur.fetchone()[0]))
9
10 cur.close()
11 return countList

```

Hàm này đếm số hàng trong mỗi phân vùng round-robin, giả định phân phối đều dựa trên thứ tự hàng. Hàm sử dụng ROW_NUMBER() OVER () để gán số thứ tự cho mỗi hàng, sau đó tính toán phân vùng bằng cách lấy dư (%) với numberofpartitions.

> Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

+ **testroundrobininsert()** :

- Tác dụng : Kiểm tra hàm roundrobininsert trong Interface.py để đảm bảo nó chèn một hàng dữ liệu mới (userid=100, movieid=1, rating=3) vào bảng ratings và bảng phân vùng rrobin_part0 (do expectedtableindex='0').

```

1 [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '0')
2 # [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '1')
3 # [result, e] = testHelper.testroundrobininsert(MyAssignment, RATINGS_TABLE, 100, 1, 3, conn, '2')
4 if result:
5     print("roundrobininsert function pass!")
6 else:
7     print("roundrobininsert function fail!")

```

```

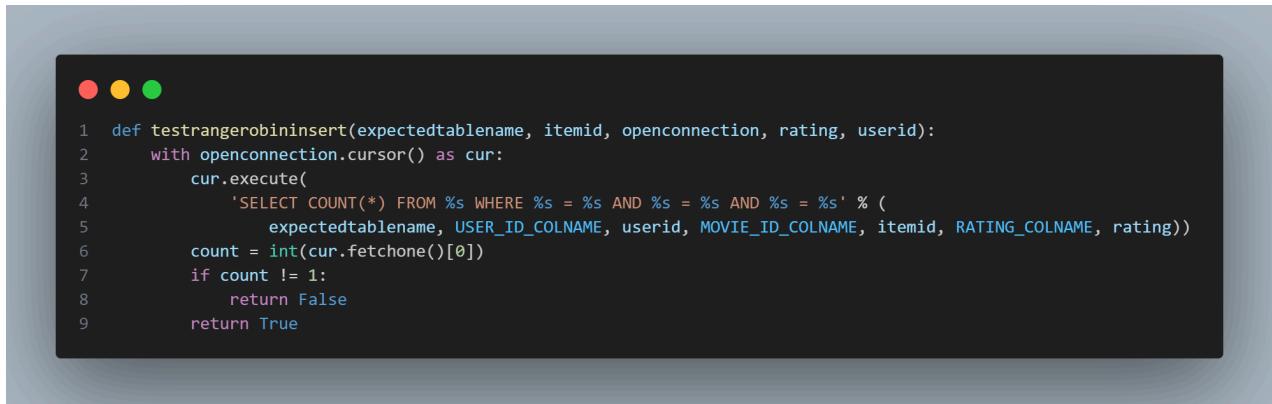
1 def testroundrobininsert(MyAssignment, ratingstablename, userid, itemid, rating, openconnection, expectedtableindex):
2     try:
3         expectedtablename = RROBIN_TABLE_PREFIX + expectedtableindex
4         MyAssignment.roundrobininsert(ratingstablename, userid, itemid, rating, openconnection)
5         if not testrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
6             raise Exception(
7                 'Round robin insert failed! Could not find (%s, %s, %s) tuple in %s table' % (userid, itemid, rating, expectedtablename))
8     except Exception as e:
9         traceback.print_exc()
10    return [False, e]
11    return [True, None]

```

Cách hoạt động :

> Hàm testroundrobininsert() trong testHelper.py gọi roundrobininsert() trong Interface.py.

> Sau đó, gọi teststrangerobininsert() để kiểm tra xem hàng (100, 1, 3) có trong bảng rrobin_part0 hay không. Nếu không tìm thấy hàng, ném ngoại lệ.



```
1 def teststrangerobininsert(expectedtablename, itemid, openconnection, rating, userid):
2     with openconnection.cursor() as cur:
3         cur.execute(
4             'SELECT COUNT(*) FROM %s WHERE %s = %s AND %s = %s AND %s = %s' % (
5                 expectedtablename, USER_ID_COLNAME, userid, MOVIE_ID_COLNAME, itemid, RATING_COLNAME, rating))
6         count = int(cur.fetchone()[0])
7         if count != 1:
8             return False
9     return True
```

Hàm này kiểm tra xem một hàng đã được chèn vào bảng phân vùng đúng chưa, bằng cách đếm số hàng khớp với userid, itemid, và rating trong bảng expectedtablename.

> Trả về [True, None] nếu thành công, hoặc [False, e] nếu thất bại.

III. Kết Luận

Qua quá trình thực hiện dự án “Mô phỏng các phương pháp phân mảnh dữ liệu trong hệ quản trị cơ sở dữ liệu quan hệ”, nhóm chúng em đã có cơ hội tiếp cận và triển khai trực tiếp hai kỹ thuật phân mảnh dữ liệu phổ biến: phân mảnh theo khoảng (range partitioning) và phân mảnh theo vòng tròn (round robin partitioning).

1. Những gì đã làm được:

- Xây dựng hoàn chỉnh các hàm xử lý dữ liệu bằng Python:
 - + Đọc và xử lý file dữ liệu thật từ MovieLens.
 - + Tạo bảng ratings và nạp dữ liệu vào hệ quản trị CSDL (MySQL).
 - + Tạo các bảng phân mảnh theo đúng thuật toán và tên quy định.
 - + Chèn bản ghi mới vào đúng phân mảnh tương ứng (range hoặc round robin).

- Tự thiết kế và sử dụng bộ kiểm thử tự động để kiểm tra:
 - + Đúng số lượng bản ghi.
 - + Đảm bảo các phân mảnh không trùng lặp.
 - + Dữ liệu có thể tái tổ hợp đầy đủ từ các phân mảnh.
- Thực hiện kiểm thử trên nhiều tình huống khác nhau với tệp dữ liệu mẫu.

2. Kiến thức và kỹ năng học được:

- Hiểu rõ bản chất và ứng dụng của phân mảnh ngang trong hệ quản trị cơ sở dữ liệu.
- Làm quen với quy trình mô hình hóa, thiết kế và kiểm thử hệ thống xử lý dữ liệu lớn.
- Luyện tập kỹ năng xử lý file, làm việc với cơ sở dữ liệu MySQL bằng Python (sử dụng pymysql).
- Rèn luyện khả năng viết code sạch, chia module rõ ràng và xây dựng hàm kiểm thử tự động.

3. Khó khăn & Bài học rút ra:

- Việc đảm bảo tính đúng đắn (completeness, disjointness, reconstruction) trong phân mảnh yêu cầu hiểu sâu về logic SQL.
- Phải xử lý kỹ lưỡng dữ liệu đầu vào để tránh lỗi cú pháp hoặc dữ liệu không hợp lệ.
- Tầm quan trọng của kiểm thử tự động trong các dự án dữ liệu — giúp tiết kiệm thời gian và phát hiện lỗi nhanh chóng.

4. Tổng kết:

Dự án không chỉ giúp nhóm em củng cố kiến thức về cơ sở dữ liệu phân tán, mà còn giúp rèn luyện tư duy lập trình hướng giải pháp, xử lý dữ liệu thực tế và khả năng triển khai các thuật toán phân mảnh một cách linh hoạt và chính xác.

IV. Tài liệu tham khảo

- Principles of distributed database systems (Ozsü M.T., Valduriez P).
- Slide bài giảng Cơ sở dữ liệu phân tán.
- <https://dev.mysql.com/doc/>
- <https://www.w3schools.com/MySQL/default.asp>