Nhóm 19

Họ và tên	Mã sinh viên	Phân công
Nguyễn Quang Phú	B22DCCN621	
Trần Quốc An	B22DCCN145	
Đào Đức Duy	B22DCCN621	

1, Hàm LoadRatings

Mục đích: Hàm này được sử dụng để nạp dữ liệu từ file rating.dat vào bảng Ratings có các cột là userId, movieId, rating

Cách làm: Trước tiên cần tạo bảng Ratings với các cột là userId, movieId, rating sử dụng câu lệnh sql:

```
# Tạo bảng @ratingsTableName
cursor.execute(f"""
CREATE TABLE {ratingsTableName} (
    userid INT,
    movieid INT,
    rating FLOAT
)
""")
```

Dữ liệu trong file **rating.dat** với mỗi dòng đều có định dạng là: **userId::movieId::rating::timestamp** nên khi đọc mỗi dòng của file ta sử dụng phương thức split('::') để tách các thuộc tính thành một mảng là columns. Trong đó:

columns[0] = userId

columns[1] = movieId

columns[2] = rating

Ta chỉ quan tâm 3 thuộc tính trên còn timestamp không cần quan tâm. Sau khi lấy được 3 thuộc tính trên sử dụng câu lệnh sql để chèn dữ liệu vào bảng **Ratings**:

```
with open(ratingsFilePath, 'r') as f:
    for line in f:
        parts = line.strip().split('::')
        if len(parts) >= 3:
            user_id = int(parts[0])
            movie_id = int(parts[1])
            rating = float(parts[2])

            # Insert vào bảng
            cursor.execute(
                f"INSERT INTO {ratingsTableName} (userid, movieid, rating) VALUES (%s, %s, %s)",
                (user_id, movie_id, rating)
            )
```

2, Hàm Range_Partition

Mục đích: Tạo N phân mảnh ngang dựa trên giá trị đồng đều của giá trị rating trong bảng Ratings.

Cách làm: Dựa trên giá trị đồng đều tức là:

Vì trong bảng Ratings thuộc tính rating giá trị nhỏ nhất là **min_rating = 0** và giá trị lớn nhất là **max_rating = 5**.

Thì lúc này giá trị đồng đều giữa các phân mảnh là: **range = (5 - 0)/N.**

Giả sử:

- N = 1 thì range = 5

=> Phân mảnh 0: [0, 5]

- N = 2 thì range = 2.5

=> Phân mảnh 0: [0, 2.5] = [0, 2.5]

=> Phân mảnh 1: (2.5, 2.5 + 2.5] = (2.5, 5)

- N = 4 thì lúc này range = 5/4 = 1.25

=> Phân mảnh 0: [0, 1.25] (lấy giá trị đầu tiên đối với phân mảnh 0)

=> Phân mảnh 1: (1.25, 1.25 + 1.25] = (1.25, 2.5]

=> Phân mảnh 2: (2.5, 2.5 + 1.25] = (2.5, 3.75]

=> Phân mảnh 3: (3.75, 5] (lấy giá trị max là 5 đối với phân mảnh N - 1)

Từ 3 ví dụ trên ta có thể thấy là: **(**)**

Khoảng giá trị của phân mảnh thứ **i = (range * i, range * (i + 1)]** trong đó **i != 0** và **N - 1**

Đối với **i = 0** thì sẽ = **[min_rating, range]**

Đối với **i = N - 1** thì sẽ = **[i * range, max_rating]**

Từ bước xác định cách chia khoảng dữ liệu ta áp dụng vào code:

- Tính giá trị đồng đều: **range_size = (5 - 0)/N**
- Để tiện cho việc truy vấn, cập nhật, chèn dữ liệu trên các phân mảnh ta tạo 1 bảng **range_partition_metadata** gồm các cột như sau:
    - **partition_id: int**
    - **min_value: float**
    - **max_value: float**

Mục tiêu của bảng này là lưu thông tin về **partition**: **partition** nào, khoảng giá trị của nó ra sao, tiện cho việc truy vấn dữ liệu trên phân mảnh

- Tiếp đó lặp **từ i = 0** đến **N - 1** để tạo các phân mảnh theo các bước như sau:
- Tạo bảng phân mảnh thứ **i** bằng câu lệnh:

```
partition_name = f"range_part{i}"
cursor.execute(f"DROP TABLE IF EXISTS {partition_name}")
cursor.execute(f"""
CREATE TABLE {partition_name} (
    userid INT,
    movieid INT,
    rating FLOAT
)
""")
```

- Từ **(**)** tính 2 giá trị:
    - **lower_bound = i * range_size**
    - **upper_bound = (i + 1) * range_size**
- Từ 2 giá trị trên ta sẽ chèn thông tin phân mảnh thứ i vào bảng **range_metadata** theo câu lệnh sql như sau:

```
cursor.execute(
    "INSERT INTO range_metadata VALUES (%s, %s, %s)",
    (i, lower_bound, upper_bound)
)
```

- Dựa vào giá trị i ta sẽ điều chỉnh câu lệnh chèn dữ liệu vào phân mảnh thứ i theo mệnh đề WHERE như sau:
    - **i = 0:**

```
cursor.execute(f"""
INSERT INTO {partition_name}
SELECT * FROM {ratingsTableName}
WHERE rating >= {lower_bound} AND rating <= {upper_bound}
""")
```

    - **i != 0:** Trong trường hợp **i = N – 1** thì thay đổi **upper_bound = 5**

```
cursor.execute(f"""
INSERT INTO {partition_name}
SELECT * FROM {ratingsTableName}
WHERE rating > {lower_bound} AND rating <= {upper_bound}
""")
```

Từ các bước ở trên chúng ta đã thành công trong việc phân mảnh dữ liệu bảng Ratings thành các phân mảnh nhỏ hơn dựa trên giá trị đồng đều của thuộc tính rating.

3, Hàm RoundRobin_Partition

Mục đích: Phân mảnh ngang dữ liệu theo vòng tròn

Cách làm:

Phân mảnh vòng tròn:

Giả sử muốn tạo 5 phân mảnh, thì theo phân mảnh vòng tròn thì các hàng trong bảng sẽ được chèn vào phân mảnh tương ứng như sau:

Phân mảnh	Hàng(trong bảng Ratings)
Partition_0	Row1
Partition_1	Row2
Partition_2	Row3
Partition_3	Row4
Partition_4	Row5

Partition_0	Row6
Partiton_1	Row7

Khi chèn đến phân mảnh Partition_(N - 1) thì quay trờ về phân mảnh Partition_0, cứ tiếp tục cho đến hết dữ liệu trong bảng Ratings

Từ bước xác định phân mảnh vòng tròn là gì ta có thể áp dụng vào code để tạo phân mảnh như sau:

1. Tạo bảng metadata rrobin_metadata: để lưu thông tin các phân mảnh gồm các schema như sau:

```
cursor.execute("""
CREATE TABLE rrobin_metadata (
    partition_count INT,
    next_partition INT
)
""")
```

   Giải thích từng thuộc tính:

   - **partition_count**: Số phân mảnh
   - **next_partition**: lưu phân mảnh kế tiếp mà dữ liệu sẽ được lưu vào

2. Khi tạo xong cần khởi tạo giá trị ban đầu cho bảng metadata theo cú pháp trong python:

```
cursor.execute("INSERT INTO rrobin_metadata VALUES (%s, %s)", (numberPartitions, 0))
```

   Giải thích:

   - Do là mới tạo bảng metadata thì partition được chèn dữ liệu đầu tiên sẽ là partition_0 nên ta lưu next_partition là 0 và số lượng partition là numberPartition

3. Tạo các bảng phân mảnh từ **0** đến **N = 1**: Lặp **i = 0** đến **N - 1**:

- Mỗi giá trị **i** sẽ tạo ra các tên bảng phân mảnh tương ứng.

- Tạo bảng phân mảnh có tên đã được tạo ở trên và có các schema giống với bảng Ratings:

```
partition_name = f"rrobin_part{i}"
cursor.execute(f"DROP TABLE IF EXISTS {partition_name}")
cursor.execute(f"""
CREATE TABLE {partition_name} (
    userid INT,
    movieid INT,
    rating FLOAT
)
""")
```

4. Chèn dữ liệu vào phân mảnh:

- Lấy toàn bộ dữ liệu từ bảng Ratings
- Duyệt toàn bộ dữ liệu:
    - Mỗi lần duyệt tính: **partition_index = i % numberPartition**

Giải thích:

- **i**: là index của row trong danh danh sách dữ liệu được lấy từ bảng Ratings
- **i % numberPartition**: đảm bảo giá trị luôn nằm trong đoạn từ **0** đến **numberPartition - 1**
- Từ **partition_index** => tính được tên bảng partition cần chèn
- Có được tên bảng partition cần chèn sử dụng cú pháp sau để chèn:

```
partition_index = i % numberPartitions
partition_name = f"rrobin_part{partition_index}"
cursor.execute(
    f"INSERT INTO {partition_name} VALUES (%s, %s, %s)",
    (row[0], row[1], row[2])
)
```

Giải thích: row[0], row[1], row[2] tương ứng các thuộc tính mà được lấy từ câu truy vấn lấy tất cả bản ghi trong bảng Ratings:

```
cursor.execute(f"SELECT userid, movieid, rating FROM {ratingsTableName}")
all_rows = cursor.fetchall()
```

5. Sau khi chèn hết tất cả dữ liệu từ bảng Ratings vào các phân mảnh tương ứng ta cần phải cập nhật next_**partition** trong bảng **metadata** để lần sau chèn dữ liệu ta có thể biết **partition** nào cần được sử dụng theo cú pháp:

```
cursor.execute("UPDATE rrobin_metadata SET next_partition = %s", (len(all_rows) % numberPartitions,))
```

6. Sau toàn bộ các bước trên ta đã phân mảnh dữ liệu từ bảng Ratings vào các phân mảnh tương ứng theo phân mảnh vòng tròn.

4, RoundRobin_Insert

Mục đích: Chèn bản ghi mới vào các phân mảnh vòng tròn.

Cách làm: Như ở hàm **RoundRobin_Partition** chúng ta đã tạo 1 bảng metadata là **rrobin_metadata** trong bảng đó có thuộc tính **next_partition,** thuộc tính này được sử dụng để xác định phân mảnh nào sẽ được dùng để chèn dữ liệu mới.

Chi tiết cách làm:

1. Chèn dữ liệu vào bảng gốc Ratings:

```
cursor.execute(
    f"INSERT INTO {ratingsTableName} (userid, movieid, rating) VALUES (%s, %s, %s)",
    (userid, itemId, rating)
)
```

2. Thực hiện lệnh sql để lấy lấy dữ liệu từ bảng metadata **rrobin_metadata** như sau:

```
cursor.execute("SELECT partition_count, next_partition FROM rrobin_metadata")
partition_count, next_partition = cursor.fetchone()
```

3. Sau khi có **next_partition** ta có được tên bảng phân mảnh là => chèn dữ liệu vào phân mảnh tương ứng:

```
partition_name = f"rrobin_part{next_partition}"
cursor.execute(
    f"INSERT INTO {partition_name} (userid, movieid, rating) VALUES (%s, %s, %s)",
    (userid, itemId, rating)
)
```

4. Sau khi chèn xong cần cập nhật bảng metadata, tức cập nhật next_partition tiếp theo:

```
next_partition = (next_partition + 1) % partition_count
cursor.execute("UPDATE rrobin_metadata SET next_partition = %s", (next_partition,))
```

5. Sau từng bước trên ta đã chèn dữ liệu vào phân mảnh tương ứng theo vòng tròn.

5, Range_Insert

Mục đích: Chèn dữ liệu vào phân mảnh theo phân mảnh khoảng giá trị đồng đều của giá trị rating trong bảng Ratings

Cách làm: Như đã nói ở hàm **Range_Partition** chúng ta đã tạo bảng metadata là **range_metadata** trong đó có lưu thuộc tính **partition_id**(phân mảnh nào được dùng

để chèn), **min_value, max_value**(khoảng giá trị để xác định với giá trị rating thì bản ghi sẽ được ghi vào đâu.

Sẽ có 2 khoảng giá trị để tìm phân mảnh:

- (min_value, max_value]: thường là các phân mảnh != 0 và N - 1 (các phân mảnh này chỉ nhận cả giá trị cuối, giá trị đầu không nhận)
- [min_value, max_value]: thường là phân mảnh 0 hoặc N - 1 (do cả 2 phân mảnh này đều nhận giá trị cả ở 2 đầu)

Chi tiết:

1. Chèn dữ liệu gốc vào bảng Ratings(bảng gốc) theo cú pháp:

```
cursor.execute(
    f"INSERT INTO {ratingsTableName} (userid, movieid, rating) VALUES (%s, %s, %s)",
    (userid, itemId, rating)
)
```

2. Tìm kiếm phân mảnh theo 2 khoảng giá trị ở trên:

- (min_value, max_value]:

```
cursor.execute(
    "SELECT partition_id FROM range_metadata WHERE %s > min_value AND %s <= max_value",
    (rating, rating)
)
```

- Nếu không có kết quả từ câu truy vấn trên thì tìm tiếp trong với khoảng [min_value, max_value] như sau:

```
cursor.execute(
    "SELECT partition_id FROM range_metadata WHERE %s >= min_value AND %s <= max_value",
    (rating, rating)
)
```

Tiếp tục kiểm tra nếu vẫn không tìm thấy kết quả thì không tìm thấy phân mảnh phù hợp => rollback

Nếu tìm thấy ta lấy được **partition_id** từ đó lấy được tên bảng phân mảnh => sử dụng câu truy vấn để chèn dữ liệu vào phân mảnh tương ứng:

```
partition_name = f"range_part{partition_id}"

# Chèn vào partition tương ứng
cursor.execute(
    f"INSERT INTO {partition_name} (userid, movieid, rating) VALUES (%s, %s, %s)",
    (userid, itemId, rating)
)
```

Sau tất cả các bước mà không gặp lỗi thì **commit**, dữ liệu đã được lưu ở phân mảnh và bảng gốc **Ratings**