HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO CƠ SỞ DỮ LIỆU PHÂN TÁN

Nhóm lớp: 10

Nhóm: 19

Thành viên	MSV
Trần Quốc An	B22DCCN007
Đào Đức Duy	B22DCCN145
Nguyễn Quang Phú	B22DCCN621

HÀ NỘI, 06/2025

Mục lục

1: Giới thiệu

Bài tập lớn này yêu cầu mô phỏng các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, cụ thể là PostgreSQL. Nhóm đã phát triển một tập các hàm Python để thực hiện các nhiệm vụ: tải dữ liệu đánh giá phim từ tệp ratings.dat vào một bảng quan hệ, phân mảnh bảng này bằng hai phương pháp phân mảnh ngang là phân mảnh theo khoảng (Range Partitioning) và phân mảnh xoay vòng (Round-Robin Partitioning), và cuối cùng là triển khai các hàm để chèn các bản ghi dữ liệu mới vào đúng phân mảnh tương ứng ().

Mục tiêu của báo cáo này là trình bày chi tiết cách tiếp cận và giải quyết vấn đề của nhóm, bao gồm giải thích logic của từng hàm được triển khai, các quyết định thiết kế quan trọng, và cách quản lý metadata để đảm bảo tính nhất quán và hiệu quả.

2. Phân tích yêu cầu và dữ liệu đầu vào

2.1. Dữ liệu đầu vào

- Dữ liệu được sử dụng là tập dữ liệu MovieLens 10M, chứa trong tệp ratings.dat. Mỗi dòng trong tệp biểu diễn một đánh giá phim và có định dạng UserID::MovieID::Rating::Timestamp ().
    - UserID: Kiểu số nguyên.
    - MovieID: Kiểu số nguyên.
    - Rating: Kiểu số thực, trên thang điểm 5 sao, có thể chia nửa sao (ví dụ: 0.5, 1.0,..., 5.0).
    - Timestamp: Kiểu số nguyên, là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970.
- Theo yêu cầu của đề bài, bảng Ratings trong cơ sở dữ liệu sẽ có lược đồ UserID (int), MovieID (int), Rating (float) (). Do đó, cột Timestamp từ tệp dữ liệu sẽ không được lưu trữ trong bảng Ratings và các bảng phân mảnh của nó.

2.2 Các hàm cần triển khai

- **Hàm loadratings(ratingstablename, ratingsfilepath, openconnection):** Tải dữ liệu từ ratings.dat vào bảng Ratings.
- **Hàm rangepartition(ratingstablename, numberofpartitions, openconnection)**: Tạo N phân mảnh ngang của bảng ratings_table_name dựa trên các khoảng giá trị đồng đều của thuộc tính Rating.
- **Hàm roundrobinpartition(ratingstablename, numberofpartitions, openconnection):** Tạo N phân mảnh ngang của bảng ratings_table_name sử dụng phương pháp xoay vòng.
- **Hàm roundrobininsert(ratingstablename, userid, itemid, rating, openconnection)**: Chèn một bản ghi mới vào bảng ratings_table_name và vào đúng phân mảnh xoay vòng.
- **Hàm rangeinsert(ratingstablename, userid, itemid, rating, openconnection)**: Chèn một bản ghi mới vào bảng ratings_table_name và vào đúng phân mảnh theo khoảng giá trị Rating.

3 Triển khai giải quyết vấn đề

3.1 Tạo Metadata

Trước khi thực hiện các chức năng chính, định nghĩa hàm create_metadata_tables_if_not_exists(cursor) để tạo các bảng metadata nếu chúng chưa tồn tại. Các bảng này dùng để duy trì trạng thái và cấu hình cho các hoạt động phân mảnh và chèn dữ liệu sau này .

- **round_robin_metadata**: Lưu trữ original_table_name, num_partitions (số lượng phân mảnh N), và current_total_rows (tổng số hàng đã được phân phối cho các phân mảnh round-robin).

```
    # For Round Robin Partitioning
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS round_robin_metadata (
            original_table_name VARCHAR(255) PRIMARY KEY,
            num_partitions INT NOT NULL,
            current_total_rows BIGINT NOT NULL
        );
    """)
```

- **range_config_metadata**: Lưu trữ original_table_name, num_partitions (số lượng phân mảnh N), min_val (giá trị Rating nhỏ nhất, mặc định 0.0), và max_val (giá trị Rating lớn nhất, mặc định 5.0) cho phân mảnh theo khoảng.

```
    # For Range Partitioning
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS range_config_metadata (
            original_table_name VARCHAR(255) PRIMARY KEY,
            num_partitions INT NOT NULL,
            min_val FLOAT NOT NULL,
            max_val FLOAT NOT NULL
        );
    """)
```

3.2 Hàm loadratings(ratingstablename, ratingsfilepath, openconnection)

Mục tiêu: Tải dữ liệu từ tệp ratings.dat vào bảng Ratings trong PostgreSQL với lược đồ UserID (int), MovieID (int), Rating (float) ().

Triển khai:

1. Hàm tạo bảng Ratings nếu chưa tồn tại với 3 cột yêu cầu.

```
    main_table_name = ratingstablename
    main_table_identifier = sql.Identifier(main_table_name)

    cursor = openconnection.cursor()
    cursor.execute(
        sql.SQL("""
            CREATE TABLE IF NOT EXISTS {} (
                userid INT,
                movieid INT,
                rating FLOAT
            );
        """).format(main_table_identifier)
    )
```

2. Đọc tệp ratings.dat từng dòng. Với mỗi dòng, tách các trường UserID, MovieID, Rating, Timestamp bằng dấu phân cách "::" .

```
sio = io.StringIO()
try:
    with open(ratingsfilepath, 'r') as f:
        for line in f:
            parts = line.strip().split('::')
            if len(parts) == 4:
                user_id, movie_id, rating, _ = parts[0], parts[1], parts[2], parts[3]
                sio.write(f"{user_id}\t{movie_id}\t{rating}\n")
    sio.seek(0)
```

3. Chỉ lấy 3 giá trị UserID, MovieID, Rating để ghi vào một đối tượng io.StringIO. Dữ liệu được định dạng tab-separated để chuẩn bị cho lệnh COPY.

```
if len(parts) == 4:
    user_id, movie_id, rating, _ = parts[0], parts[1], parts[2], parts[3]
    sio.write(f"{user_id}\t{movie_id}\t{rating}\n")
```

4. Sử dụng phương thức cursor.copy_expert() với lệnh COPY Ratings (UserID, MovieID, Rating) FROM STDIN WITH DELIMITER AS E'\\t' để tải dữ liệu hàng loạt từ io.StringIO vào bảng Ratings. Đây là phương pháp hiệu quả nhất để tải lượng lớn dữ liệu vào PostgreSQL. Lý do sử dụng: giảm thiểu chi phí xử lí hơn so với việc phải insert từng từng hàng. (giảm chi phí giao tiếp và ít phân tích cú pháp sql hơn)

```
    copy_sql_query = sql.SQL("COPY {} (userid, movieid, rating) FROM STDIN WITH DELIMITER AS E'\\t'").format(main_table_identifier)
    cursor.copy_expert(sql=copy_sql_query, file=sio)
```

5. Thực hiện connection.commit() để lưu các thay đổi.

3.3 Hàm rangepartition(ratingstablename, numberofpartitions, openconnection)

Mục tiêu: Tạo N phân mảnh ngang của bảng ratingstablename (mặc định là Ratings) dựa trên N khoảng giá trị đồng đều của thuộc tính Rating (từ 0.0 đến 5.0) ().

Triển khai:

1. Kiểm tra N > 0.

```
    if numberofpartitions <= 0:
        print("Lỗi: Số lượng phân mảnh phải là số dương")
        return
```

2. Tính toán kích thước của mỗi khoảng (step = (5.0 - 0.0) / N).

```
min_rating_val = 0.0
max_rating_val = 5.0
step = (max_rating_val - min_rating_val) / numberofpartitions
```

3. Lặp i từ 0 đến N-1 để tạo từng bảng phân mảnh:
    - Tên bảng phân mảnh: range_part{i}.

```
partition_name = RANGE_TABLE_PREFIX + str(i)
partition_identifier = sql.Identifier(partition_name)
```

    - Tạo bảng range_part{i} với lược đồ UserID (int), MovieID (int), Rating (float) nếu chưa tồn tại.

```
    cursor.execute(
        sql.SQL("""
            CREATE TABLE IF NOT EXISTS {} (
                userid INT,
                movieid INT,
                rating FLOAT
            );
        """).format(partition_identifier)
    )
    cursor.execute(sql.SQL("TRUNCATE TABLE {};").format(partition_identifier))
```

- Xóa toàn bộ dữ liệu cũ trong bảng phân mảnh này bằng TRUNCATE TABLE để đảm bảo tính đúng đắn khi chạy lại.

```
    )
    cursor.execute(sql.SQL("TRUNCATE TABLE {};").format(partition_identifier))
```

- Xác định cận dưới (current_lower_bound) và cận trên (current_upper_bound) cho phân mảnh hiện tại. Đảm bảo phân mảnh cuối cùng (i == N-1) bao gồm chính xác giá trị max_rating_val (5.0).

```
current_lower_bound = min_rating_val + i * step
current_upper_bound = min_rating_val + (i + 1) * step
```

- Xây dựng điều kiện WHERE cho thuộc tính Rating:
  - Nếu N=1: Rating >= min_rating_val AND Rating <= max_rating_val.
  - Nếu i=0 (phân mảnh đầu tiên và N > 1): Rating >= current_lower_bound AND Rating <= current_upper_bound.
  - Nếu i > 0 (các phân mảnh tiếp theo): Rating > current_lower_bound AND Rating <= current_upper_bound.

```
if i == numberofpartitions - 1:
    current_upper_bound = max_rating_val

if numberofpartitions == 1:
    condition_sql = sql.SQL("rating >= %s AND rating <= %s")
    params = (min_rating_val, max_rating_val)
elif i == 0:
    condition_sql = sql.SQL("rating >= %s AND rating <= %s")
    params = (current_lower_bound, current_upper_bound)
else:
    condition_sql = sql.SQL("rating > %s AND rating <= %s")
    params = (current_lower_bound, current_upper_bound)
```

- Sử dụng câu lệnh INSERT INTO range_part{i} (UserID, MovieID, Rating) SELECT UserID, MovieID, Rating FROM {ratings_table_name} WHERE {condition} để điền dữ liệu vào bảng phân mảnh.

```
    populate_sql = sql.SQL("""
        INSERT INTO {} (userid, movieid, rating)
        SELECT userid, movieid, rating FROM {}
        WHERE {};
    """).format(partition_identifier, original_table_identifier, condition_sql)

    cursor.execute(populate_sql, params)
```

4. Lưu thông tin cấu hình (ratingstablename, numsofpartitions, min_rating_val=0.0, max_rating_val=5.0) vào bảng range_config_metadata.

```
meta_table_identifier = sql.Identifier("range_config_metadata")

upsert_meta_sql = sql.SQL("""
    INSERT INTO {} (original_table_name, num_partitions, min_val, max_val)
    VALUES (%s, %s, %s, %s)
    ON CONFLICT (original_table_name) DO UPDATE SET
        num_partitions = EXCLUDED.num_partitions,
        min_val = EXCLUDED.min_val,
        max_val = EXCLUDED.max_val;
""").format(meta_table_identifier)
cursor.execute(upsert_meta_sql, (ratingstablename, numberofpartitions, min_rating_val, max_rating_val))
```

5. Thực hiện connection.commit().

3.4 Hàm roundrobinpartition(ratingstablename, numberofpartitions, openconnection)

Mục tiêu: Tạo N phân mảnh ngang của bảng ratingstablename sử dụng phương pháp xoay vòng .

Triển khai:

1. Kiểm tra N > 0.

```
if numberofpartitions <= 0:
    print("Lỗi: Số phân mảnh phải là số dương.")
    return
```

2. Tạo N danh sách rỗng (partition_tables_data) để lưu trữ tạm thời các hàng cho từng phân mảnh.

```
original_table_identifier = sql.Identifier(ratingstablename)
partition_tables_data = [ [] for _ in range(numberofpartitions)]
```

3. Lặp i từ 0 đến N-1 để tạo từng bảng phân mảnh:
    - Tên bảng phân mảnh: rrobin_part{i}.
    - Tạo bảng rrobin_part{i} với lược đồ UserID (int), MovieID (int), Rating (float) nếu chưa tồn tại.
    - Xóa toàn bộ dữ liệu cũ trong bảng phân mảnh này bằng TRUNCATE TABLE.

```
for i in range(numberofpartitions):
    partition_name = RROBIN_TABLE_PREFIX + str(i)
    partition_identifier = sql.Identifier(partition_name)
    cursor.execute(
        sql.SQL("""
            CREATE TABLE IF NOT EXISTS {} (
                userid INT,
                movieid INT,
                rating FLOAT
            );
        """).format(partition_identifier)
    )
    cursor.execute(sql.SQL("TRUNCATE TABLE {};").format(partition_identifier))
```

4. Truy vấn tất cả các hàng (UserID, MovieID, Rating) từ bảng ratingstablename.

```
cursor.execute(sql.SQL("SELECT userid, movieid, rating FROM {};").format(original_table_identifier))
all_rows = cursor.fetchall()
```

5. Lặp qua từng hàng đã truy vấn, sử dụng target_partition_index = row_index % N để xác định phân mảnh đích và thêm hàng đó vào danh sách tương ứng trong partition_tables_data.

```
if not all_rows:
    # print(f"Không có dữ liệu trong phân mảnh.")
    pass
else:
    for row_index, row_data in enumerate(all_rows):
        target_partition_index = row_index % numberofpartitions
        partition_tables_data[target_partition_index].append(row_data)
```

6. Sau khi phân phối tất cả các hàng, lặp qua N danh sách trong partition_tables_data. Nếu một danh sách có dữ liệu, sử dụng psycopg2.extras.execute_values() để chèn hàng loạt tất cả các hàng trong danh sách đó vào bảng phân mảnh rrobin_part{i} tương ứng.

```
for i in range(numberofpartitions):
    if partition_tables_data[i]:
        partition_name = RROBIN_TABLE_PREFIX + str(i)
        partition_identifier = sql.Identifier(partition_name)

        insert_query = sql.SQL("INSERT INTO {} (userid, movieid, rating) VALUES %s;").format(partition_identifier)
        psycopg2.extras.execute_values(
            cursor,
            insert_query.as_string(cursor.connection),
            partition_tables_data[i],
            page_size=1000
        )
```

page_size: số lượng hàng được gửi đến server trong 1 batch

Lý do sử dụng psycopg2.extras.execute_values() do tính hiệu quả khi cần insert nhiều hàng dữ liệu, tốn ít chi phí hơn cho giao tiếp mạng và xử lí sql.

7. Lưu thông tin (ratingstablename, numberofpartitions, total_rows_distributed) vào bảng round_robin_metadata. total_rows_distributed là tổng số hàng đã được đọc từ bảng gốc.

```
    meta_table_identifier = sql.Identifier("round_robin_metadata")

    total_rows_distributed = len(all_rows)
    upsert_meta_sql = sql.SQL("""
        INSERT INTO {} (original_table_name, num_partitions, current_total_rows)
        VALUES (%s, %s, %s)
        ON CONFLICT (original_table_name) DO UPDATE SET
            num_partitions = EXCLUDED.num_partitions,
            current_total_rows = EXCLUDED.current_total_rows;
    """).format(meta_table_identifier)
    cursor.execute(upsert_meta_sql, (ratingstablename, numberofpartitions, total_rows_distributed))
```

8. Thực hiện connection.commit().

3.5 Hàm roundrobininsert(ratingstablename, userid, itemid, rating, openconnection)

Mục tiêu: Chèn một bản ghi mới (UserID, ItemID, Rating) vào bảng ratings_table_name và vào đúng phân mảnh được tạo theo phương pháp xoay vòng.

Triển khai:

1. Tham số ItemID được coi là MovieID cho việc lưu trữ.

2. Chèn bản ghi mới vào bảng ratings_table_name (bảng gốc).

```
insert_main_sql = sql.SQL("INSERT INTO {} (userid, movieid, rating) VALUES (%s, %s, %s);").format(original_table_identifier)
cursor.execute(insert_main_sql, (userid, movieid, rating))
```

3. Truy vấn bảng round_robin_metadata để lấy num_partitions (N) và current_total_rows cho ratings_table_name.

```
select_meta_sql = sql.SQL("SELECT num_partitions, current_total_rows FROM {} WHERE original_table_name = %s;").format(meta_table_identifier)
cursor.execute(select_meta_sql, (ratingstablename,))
meta_data = cursor.fetchone()
```

4. Nếu không tìm thấy metadata (ví dụ: RoundRobin_Partition chưa được chạy), báo lỗi và rollback.

```
    if not meta_data:
        print(f"Lỗi: Không tìm thấy metadate cho rrobin partition")
        con.rollback()
        return
```

5. Tính toán chỉ số phân mảnh đích: target_partition_index = int(current_total_rows % num_partitions).
6. Tên bảng phân mảnh đích: rrobin_part{target_partition_index}.
7. Chèn bản ghi mới vào bảng phân mảnh đích.

```
target_partition_index = int(current_total_rows % num_partitions)
target_partition_name = RROBIN_TABLE_PREFIX + str(target_partition_index)
target_partition_identifier = sql.Identifier(target_partition_name)

insert_partition_sql = sql.SQL("INSERT INTO {} (userid, movieid, rating) VALUES (%s, %s, %s);").format(target_partition_identifier)
cursor.execute(insert_partition_sql, (userid, movieid, rating))
```

8. Cập nhật current_total_rows trong bảng round_robin_metadata bằng cách tăng giá trị lên 1.

```
update_meta_sql = sql.SQL("UPDATE {} SET current_total_rows = current_total_rows + 1 WHERE original_table_name = %s;").format(meta_table_identifier)
cursor.execute(update_meta_sql, (ratingstablename,))
```

9. Thực hiện connection.commit().

3.6 Hàm rangeinsert(ratingstablename, userid, itemid, rating, openconnection)

Mục tiêu: Chèn một bản ghi mới (UserID, ItemID, Rating) vào bảng ratings_table_name và vào đúng phân mảnh được tạo theo phương pháp khoảng giá trị, dựa trên giá trị Rating của bản ghi mới ().

Triển khai:

1. Tham số ItemID được coi là MovieID.
2. Chèn bản ghi mới vào bảng ratings_table_name (bảng gốc).

```
original_table_identifier = sql.Identifier(ratingstablename)
meta_config_table_identifier = sql.Identifier("range_config_metadata")

movieid = itemid

insert_main_sql = sql.SQL("INSERT INTO {} (userid, movieid, rating) VALUES (%s, %s, %s);").format(original_table_identifier)
cursor.execute(insert_main_sql, (userid, movieid, rating))
```

3. Truy vấn bảng range_config_metadata để lấy num_partitions, min_rating_val, max_rating_val cho ratings_table_name.

```
select_meta_sql = sql.SQL("SELECT num_partitions, min_val, max_val FROM {} WHERE original_table_name = %s;").format(meta_config_table_identifier)
cursor.execute(select_meta_sql, (ratingstablename,))
meta_data = cursor.fetchone()
```

4. Nếu không tìm thấy metadata, báo lỗi và rollback.

```
if not meta_data:
    print(f"Không tìm thấy metadata cho bảng '{ratingstablename}' phân mảng theo range")
    con.rollback()
    return
```

5. Kiểm tra xem giá trị Rating của bản ghi mới có nằm trong khoảng [min_val, max_val] đã cấu hình không. Nếu không, chỉ commit việc chèn vào bảng gốc và không chèn vào phân mảnh nào.

```
if not (min_val <= rating <= max_val):
    con.commit()
    return
```

6. Nếu Rating hợp lệ, lặp từ i = 0 đến N-1 để tìm phân mảnh đích:

- Tính toán current_lower_bound và current_upper_bound cho phân mảnh i tương tự như trong rangepartition.

```
current_lower_bound = min_val + i * step
current_upper_bound = min_val + (i + 1) * step
if i == N - 1:
    current_upper_bound = max_val
```

- Áp dụng điều kiện bao gồm/loại trừ cận (giống như trong rangepartition) để xác định xem Rating có thuộc phân mảnh này không.

```
    is_in_partition = False
    if N == 1:
        if rating >= min_val and rating <= max_val:
            is_in_partition = True
    elif i == 0:
        if rating >= current_lower_bound and rating <= current_upper_bound:
            is_in_partition = True
    else:
        if rating > current_lower_bound and rating <= current_upper_bound:
            is_in_partition = True

    if is_in_partition:
        target_partition_name = f"range_part{i}"
        break
```

- Nếu tìm thấy, xác định target_partition_name = f"range_part{i}" và thoát khỏi vòng lặp.

```
if is_in_partition:
    target_partition_name = f"range_part{i}"
    break
```

7. Nếu target_partition_name được tìm thấy, chèn bản ghi mới vào bảng phân mảnh đó.

```
if target_partition_name:
    target_partition_identifier = sql.Identifier(target_partition_name)
    insert_partition_sql = sql.SQL("INSERT INTO {} (userid, movieid, rating) VALUES (%s, %s, %s);").format(target_partition_identifier)
    cursor.execute(insert_partition_sql, (userid, movieid, rating))
```

8. Thực hiện connection.commit().

4 Phân chia công việc