HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: CƠ SỞ DỮ LIỆU PHÂN TÁN

Đề tài: Mô phỏng các phương pháp phân mảnh dữ liệu

Giảng viên: : TS.KIM NGỌC BÁCH

Nhóm : 11

Họ và tên	Mã sinh viên
Cao Đức Việt	B22DCCN894
Bùi Ngọc Thiện	B22DCCN822
Đinh Ngọc Hân	B22DCCN282

Hà Nội – 2025

MỤC LỤC

LỜI MỞ ĐẦU

Trước khi kết thúc báo cáo bài tập nhóm cuối kỳ môn *Cơ sở dữ liệu phân tán*, tập thể nhóm chúng em xin được gửi lời tri ân sâu sắc đến Thầy Kim Ngọc Bách, người đã tận tâm giảng dạy và đồng hành cùng chúng em trong suốt học kỳ vừa qua.

Thông qua bài tập cuối kỳ với chủ đề mô phỏng các kỹ thuật phân mảnh dữ liệu trong hệ thống cơ sở dữ liệu phân tán, chúng em không chỉ được củng cố kiến thức lý thuyết đã học, mà còn có cơ hội áp dụng vào bài toán thực tế một cách chủ động và sáng tạo. Đặc biệt, việc tiếp cận và triển khai các phương pháp phân mảnh ngang đã giúp chúng em hiểu rõ hơn về cách tổ chức dữ liệu sao cho phù hợp với mục tiêu tối ưu hóa hệ thống, đồng thời nâng cao khả năng mở rộng và tính linh hoạt trong quản trị cơ sở dữ liệu.

Những gì chúng em đạt được hôm nay là kết quả của sự hướng dẫn tận tình, phương pháp giảng dạy rõ ràng, cùng với tinh thần truyền cảm hứng học tập từ Thầy. Chính sự tận tâm và nghiêm túc trong từng buổi giảng đã giúp chúng em thêm yêu thích bộ môn và không ngừng cố gắng hoàn thiện kiến thức chuyên môn.

Một lần nữa, nhóm chúng em xin gửi đến Thầy lời cảm ơn chân thành và sâu sắc nhất. Kính chúc Thầy luôn mạnh khỏe, bình an và tiếp tục gặt hái nhiều thành công trong sự nghiệp giáo dục và nghiên cứu để ngày càng có thêm nhiều thế hệ sinh viên được tiếp lửa đam mê tri thức từ Thầy.

Nhóm em xin chân thành cảm ơn thầy!

I. Lý thuyết cơ sở

1. Phân mảnh dữ liệu.

- Phân mảnh là quá trình chia nhỏ một quan hệ (bảng) trong cơ sở dữ liệu thành nhiều phần nhỏ hơn (fragment), để lưu trữ tại các vị trí phân tán khác nhau trong hệ thống. Điều này giúp tối ưu hiệu năng truy vấn, cải thiện khả năng mở rộng và nâng cao tính sẵn sàng của dữ liệu.Phân mảnh trong hệ cơ sở dữ liệu phân tán được chia thành 3 loại chính:
    - Phân mảnh ngang (Horizontal Fragmentation)
    - Phân mảnh dọc (Vertical Fragmentation)
    - Phân mảnh hỗn hợp (Hybrid Fragmentation)

2. Phân mảnh ngang

- Phân mảnh ngang là quá trình tách một quan hệ toàn cục thành nhiều mảnh. Mỗi mảnh là một quan hệ con có cấu trúc giống bảng gốc, chứa một tập hợp các bản ghi thỏa mãn một điều kiện lọc nhất định, và các bản ghi giữa các mảnh là tách biệt nhau.
- Đặc điểm:
    - Mỗi mảnh có cùng cấu trúc với bảng gốc (các thuộc tính không thay đổi, chỉ khác về số dòng).
    - Dữ liệu được chia dựa trên giá trị của một hoặc nhiều thuộc tính, sử dụng các điều kiện chọn.
    - Các mảnh thường rời nhau và khi hợp lại sẽ tái tạo được bảng gốc, đảm bảo tính toàn vẹn dữ liệu.
    - Phù hợp khi bảng có số lượng bản ghi lớn, và truy vấn chủ yếu nhắm tới các phân vùng cụ thể của dữ liệu theo điều kiện lọc.

3. Phân mảnh dọc

- Phân mảnh dọc là kỹ thuật chia một bảng thành nhiều phần, mỗi phần chứa một nhóm các thuộc tính (cột) thường được sử dụng chung trong các truy vấn hoặc chức năng của hệ thống. Mục tiêu là tăng hiệu quả truy xuất dữ liệu, bằng cách nhóm các thuộc tính liên quan vào cùng một mảnh để giảm lượng dữ liệu cần xử lý hoặc truyền đi.
- Đặc điểm:
    - Mỗi mảnh chỉ bao gồm một số cột của bảng gốc, không giữ lại toàn bộ cấu trúc.

- Các thuộc tính trong cùng một mảnh thường có quan hệ sử dụng gần nhau.
- Khi cần các mảnh có thể kết hợp lại để tái tạo bảng ban đầu.
- Phù hợp khi có nhiều loại truy vấn khác nhau, mỗi truy vấn chỉ quan tâm đến một nhóm thuộc tính riêng biệt.

4. Phân mảnh hỗn hợp

- Phân mảnh hỗn hợp là kỹ thuật kết hợp cả phân mảnh ngang và phân mảnh dọc trên cùng một bảng. Điều này có nghĩa là bảng được chia nhỏ về cả dòng và cột, nhằm tối ưu hóa khả năng lưu trữ và truy vấn.
- Đặc điểm:
    - Kết hợp cả phân mảnh ngang và phân mảnh dọc trên cùng một bảng.
    - Linh hoạt hơn, phù hợp với nhiều loại truy vấn khác nhau, mỗi truy vấn chỉ quan tâm đến một phần dữ liệu.
    - Giúp loại bỏ các mảnh rỗng và các mảnh không cần thiết.

5. Phân mảnh tròn

- Phân mảnh tròn là kỹ thuật phân mảnh ngang, trong đó các bản ghi của một bảng được phân phối tuần tự và đều đặn theo vòng lặp đến các mảnh. Bản ghi thứ 1 vào mảnh 1, bản ghi thứ 2 vào mảnh 2, ..., rồi quay vòng lại.
- Đặc điểm:
    - Mỗi mảnh có cấu trúc giống bảng gốc, chỉ khác về tập dòng.
    - Không sử dụng điều kiện chọn, mà phân phối bản ghi theo thứ tự lần lượt.
    - Phân phối dữ liệu đồng đều giữa các mảnh từ đó tránh mảnh quá lớn hoặc quá nhỏ.
    - Không tối ưu cho truy vấn lọc theo điều kiện, vì dữ liệu được chia ngẫu nhiên.

6. Lợi ích của phân mảnh dữ liệu

- **Tăng hiệu suất:** Phân mảnh giúp giảm khối lượng dữ liệu mỗi truy vấn cần xử lý, từ đó tăng tốc độ truy xuất và xử lý dữ liệu.

- **Tính sẵn sàng và khả năng mở rộng:** Nếu một mảnh dữ liệu gặp sự cố, hệ thống vẫn có thể hoạt động với các mảnh khác. Ngoài ra, dễ dàng phân phối dữ liệu đến nhiều site hơn khi mở rộng hệ thống.
- **Tối ưu chi phí truyền thông:** Truy vấn chỉ truy cập đến các mảnh liên quan nên giảm đáng kể lượng dữ liệu phải truyền giữa các nút mạng.
- **Hỗ trợ tối ưu hóa truy vấn:** Các phép toán như select, join có thể được thực hiện riêng trên từng fragment phù hợp, sau đó mới kết hợp kết quả.
- **Linh hoạt trong thiết kế:** Có thể áp dụng phân mảnh ngang, dọc hoặc hỗn hợp tùy theo mô hình sử dụng thực tế và loại truy vấn phổ biến.

7. Hạn chế của phân mảnh dữ liệu

- **Tăng độ phức tạp quản lý:** Thiết kế, theo dõi và điều chỉnh phân mảnh yêu cầu phải phân tích kỹ lưỡng, đặc biệt trong hệ thống hay có nhiều thay đổi về truy vấn hoặc dữ liệu.
- **Tốn kém nhiều tài nguyên:** Khi cần truy xuất đầy đủ dữ liệu, hệ thống phải kết hợp nhiều mảnh lại, gây tốn kém tài nguyên và thời gian xử lý.
- **Không hiệu quả nếu truy vấn truy cập nhiều mảnh:** Khi một truy vấn cần dữ liệu từ nhiều mảnh khác nhau, hiệu suất có thể giảm do tăng khối lượng xử lý, khiến hệ thống kém hiệu quả hơn khi phải tập hợp kết quả từ nhiều nơi và phải Tập hợp lại toàn bộ kết quả để xử lý truy vấn cuối cùng.

II. Giới thiệu chung

1. Mục tiêu

Mô phỏng các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở (ví dụ: PostgreSQL hoặc MySQL). Cần tạo một tập các hàm Python để tải dữ liệu đầu vào vào một bảng quan hệ, phân mảnh bảng này bằng các phương pháp phân mảnh ngang khác nhau, và chèn các bộ dữ liệu mới vào đúng phân mảnh. Ở đây sử dụng hai phương pháp phân mảnh là Range Partitioning và Round Robin Partitioning.

2. Dữ liệu đầu vào

Dữ liệu đầu vào là một tập dữ liệu đánh giá phim được thu thập từ trang web MovieLens (http://movielens.org). Dữ liệu thô có trong tệp ratings.dat.

Tệp ratings.dat chứa 10 triệu đánh giá và 100.000 thẻ được áp dụng cho 10.000 bộ phim bởi 72.000 người dùng. Mỗi dòng trong tệp đại diện cho một đánh giá của một người dùng với một bộ phim, và có định dạng như sau:

UserID::MovieID::Rating::Timestamp.

Các đánh giá được thực hiện trên thang điểm 5 sao, có thể chia nửa sao. Dấu thời gian (Timestamp) là số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970. Ví dụ nội dung tệp:

1::122::5::838985046
1::185::5::838983525
1::231::5::838983392

Bài tập này chỉ yêu cầu sử dụng 3 trường đầu tiên (UserID, MovieID, Timestamp).

3. Yêu cầu

Cài đặt các hàm Python làm việc với CSDL (ở đây sử dụng PostgreSQL), đóng vai trò lưu trữ, tải và phân mảnh dữ liệu:

- **LoadRatings():** nhận vào đường dẫn tới tệp ratings.dat và thực hiện tải nội dung tệp vào bảng Ratings trong PostgreSQL với schema:
  (UserID (int), MovieID (int), Rating (float)).

Bảng Ratings sẽ là bảng gốc lưu trữ toàn bộ dữ liệu gốc từ file ratings.dat.

- **Range_Partiton()**: nhận vào: bảng Ratings trong PostgreSQL và một số nguyên N là số phân mảnh cần tạo. Range_Partition() sẽ tạo N phân mảnh ngang của bảng Ratings và lưu vào PostgreSQL. Thuật toán sẽ phân chia dựa trên N khoảng giá trị đồng đều của thuộc tính Rating.
- **RoundRobin_Partition()**: nhận vào bảng Ratings trong PostgreSQL và một số nguyên N là số phân mảnh cần tạo. Hàm sẽ tạo N phân mảnh ngang của bảng Ratings và lưu chúng trong PostgreSQL , sử dụng phương pháp phân mảnh kiểu vòng tròn (round robin).
- **Range_Insert()**: nhận vào bảng Ratings trong PostgreSQL , UserID, ItemID, Rating. Range_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh dựa trên giá trị của Rating.
- **RoundRobin_Insert()**: nhận vào bảng Ratings trong PostgreSQL, UserID, ItemID, Rating. RoundRobin_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh theo cách round robin.

III. Triển khai hệ thống

1. Chuẩn bị môi trường và công cụ

- Môi trường sử dụng: Windows 10
- Hệ quản trị CSDL: PostgreSQL
- Ngôn ngữ lập trình: Python
- Thư viện sử dụng: psycopg2 (kết nối và tương tác với PostgreSQL), io (thao tác với luồng dữ liệu text), os (các hàm liên quan đến hệ điều hành)

2. Các hàm được cài đặt

2.1. get_connection():

```python
def get_connection(
    user="postgres", password="1234", dbname="ratingsdb", host="localhost"
):
    conn = psycopg2.connect(dbname=dbname, user=user, password=password, host=host)
    conn.autocommit = False
    return conn
```

Hàm get_connection()

- Hàm này tạo kết nối với PostgreSQL và trả về đối tượng connection.
- **Tham số**: Nhận user, password, dbname, host với giá trị mặc định, cho phép linh hoạt cấu hình kết nối.
- **Kết nối**: Sử dụng psycopg2.connect để tạo đối tượng kết nối.
- **Trả về**: Đối tượng kết nối (conn) để sử dụng trong các hàm khác.

2.2. create_metadata():

```python
def create_metadata(meta_table, meta_values, conflict_updates, conn):
    cur = conn.cursor()
    try:
        # Tạo bảng metadata tuỳ theo loại RANGE hay ROUND-ROBIN
        if meta_table == RANGE_META:
            cur.execute(f"""
                CREATE TABLE IF NOT EXISTS {RANGE_META} (
                    table_name      TEXT PRIMARY KEY,
                    partition_count INT NOT NULL,
                    min_val         FLOAT NOT NULL,
                    max_val         FLOAT NOT NULL
                );
            """)
        elif meta_table == ROBIN_META:
            cur.execute(f"""
                CREATE TABLE IF NOT EXISTS {ROBIN_META} (
                    table_name       TEXT PRIMARY KEY,
                    partition_count  INT NOT NULL,
                    last_rr_index    BIGINT NOT NULL
                );
            """)
        columns = list(meta_values.keys())
        values = list(meta_values.values())
        col_str = ', '.join(columns)
        placeholder = ', '.join(['%s'] * len(values))
        conflict_set = ', '.join(f"{k} = EXCLUDED.{k}" for k in conflict_updates.keys())
        # Thực thi INSERT với ON CONFLICT
        cur.execute(
            f"""
            INSERT INTO {meta_table} ({col_str})
            VALUES ({placeholder})
            ON CONFLICT (table_name) DO UPDATE SET {conflict_set};
            """,
            values,
        )

    finally:
        cur.close()
```

Hàm create_metadata():

- **Mục đích:** Tạo bảng metadata (range_meta hoặc robin_meta) và ghi/cập nhật thông tin metadata cho phân mảnh.
- **Kiểm tra và tạo bảng metadata:**
  - Dựa trên meta_table (RANGE_META hoặc ROBIN_META), hàm tạo bảng với lược đồ phù hợp.
  - Sử dụng CREATE TABLE IF NOT EXISTS để tránh lỗi nếu bảng đã tồn tại.
- **Chuẩn bị câu lệnh INSERT với ON CONFLICT**:
  - Lấy danh sách cột và giá trị từ meta_values (dictionary) để xây dựng phần INSERT.

- Tạo chuỗi placeholder (%s) để truyền giá trị an toàn, tránh SQL injection.
- Xây dựng phần ON CONFLICT với các cột từ conflict_updates, đảm bảo cập nhật đúng các trường nếu table_name đã tồn tại.

- **Thực thi và quản lý tài nguyên**: Thực thi câu lệnh SQL với các giá trị từ meta_values.

2.3. loadratings():

```
def loadratings(table_name, filepath, conn):
    cur = conn.cursor()
    try:
        if not os.path.exists(filepath):
            raise FileNotFoundError(f"File không tồn tại: {filepath}")
        cur.execute(f"DROP TABLE IF EXISTS {table_name} CASCADE;")
        cur.execute(f"""
            CREATE TABLE {table_name} (
                userid  INT,
                movieid INT,
                rating  FLOAT
            );
        """)
        buf = io.StringIO()
        with open(filepath, "r") as f:
            for line in f:
                parts = line.strip().split("::")
                if len(parts) >= 3:
                    buf.write(f"{parts[0]}\t{parts[1]}\t{parts[2]}\n")
        buf.seek(0)
        cur.copy_expert(
            f"COPY {table_name} (userid, movieid, rating) "
            "FROM STDIN WITH (FORMAT text, DELIMITER E'\\t');",
            buf,
        )

        conn.commit()
    except Exception as e:
        conn.rollback()
        raise RuntimeError(f"Lỗi load ratings: {e}")
    finally:
        cur.close()
```

Hàm loadratings()

- **Mục đích:** Nạp dữ liệu từ ratings.dat vào bảng Ratings trong PostgreSQL
- **Tạo bảng:**
  - Khởi tạo con trỏ từ kết nối con đã mở.

    - Xóa bảng hiện có (nếu có) bằng DROP TABLE để tránh lỗi khi tạo bảng mới.
    - Tạo bảng mới với dạng (userid: INT, movieid: INT, rating: FLOAT).
- **Xử lý dữ liệu:**
    - Đọc từng dòng từ ratings.dat, tách bằng :: để lấy UserID, MovieID, Rating (bỏ Timestamp).
    - Ghi các trường này vào bộ đệm StringIO dưới dạng tab-separated.
- **Nạp dữ liệu:**
    - Sử dụng lệnh COPY qua cur.copy_expert để nạp dữ liệu từ bộ đệm vào bảng, rất hiệu quả cho tập dữ liệu lớn.
    - Chỉ định FORMAT text và DELIMITER E'\t'..
- **Kết thúc và xử lý nếu có lỗi xảy ra**
    - Xác nhận nếu thành công.
    - Hủy và báo lỗi nếu có vấn đề.
    - Đóng con trỏ.

2.4. rangepartition()

```python
def rangepartition(table_name, num_partitions, conn):
    cur = conn.cursor()
    try:
        if num_partitions < 1:
            raise ValueError("Số phân vùng phải >=1")

        for i in range(num_partitions):
            cur.execute(f"DROP TABLE IF EXISTS range_part{i} CASCADE;")
            cur.execute(f"""
                CREATE TABLE range_part{i} (
                    userid  INT,
                    movieid INT,
                    rating  FLOAT
                );
            """)
        min_rating, max_rating = 0.0, 5.0
        delta = (max_rating - min_rating) / num_partitions
        for i in range(num_partitions):
            low = min_rating + i * delta
            high = (min_rating + (i + 1) * delta) if i < num_partitions - 1 else max_rating
            cond = (
                f"rating >= {low} AND rating <= {high}"
                if i == 0
                else f"rating > {low} AND rating <= {high}"
            )
            cur.execute(f"""
                INSERT INTO range_part{i}
                SELECT userid, movieid, rating
                  FROM {table_name}
                 WHERE {cond};
            """)
            create_metadata(
                meta_table=RANGE_META,
                meta_values={
                    "table_name": table_name,
                    "partition_count": num_partitions,
                    "min_val": min_rating,
                    "max_val": max_rating,
                },
                conflict_updates={
                    "partition_count": num_partitions,
                    "min_val": min_rating,
                    "max_val": max_rating,
                },
                conn=conn,
            )

        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        cur.close()
```

Hàm rangepartition()

- **Mục đích:** Chia bảng Ratings thành N phân mảnh theo khoảng dựa trên thuộc tính Rating, tạo các bảng range_part0, ..., range_part{N-1}.

- **Kiểm tra đầu vào:**
  Đảm bảo num_partitions >= 1, nếu không thì báo lỗi.
- **Tạo bảng phân mảnh:**
  - Xóa các bảng range_part{i} cũ (nếu có).
  - Tạo các bảng mới với cùng lược đồ (userid: INT, movieid: INT, rating: FLOAT).
- **Tính khoảng phân mảnh:**
  - Xác định khoảng giá trị [0.0, 5.0] và độ rộng mỗi phân mảnh: delta = (5.0 - 0.0) / num_partitions.
- **Tính ranh giới cho mỗi phân mảnh:**
  - low = 0.0 + i * delta
  - high = 0.0 + (i + 1) * delta (hoặc 5.0 cho phân mảnh cuối).
  - Ví dụ: N=2 → phân mảnh 0: [0.0, 2.5], phân mảnh 1: (2.5, 5.0].
- **Phân phối dữ liệu:**
  - Tạo điều kiện WHERE:
  - Phân mảnh đầu (i=0): rating >= low AND rating <= high.
  - Các phân mảnh khác: rating > low AND rating <= high.
  - Chèn bản ghi từ Ratings vào range_part{i} bằng INSERT INTO ... SELECT.
- **Lưu metadata:**
  - Gọi create_metadata để lưu thông tin vào range_meta: table_name, partition_count, min_val, max_val.
  - conflict_updates đảm bảo cập nhật các cột nếu table_name đã tồn tại.
- **Kết thúc và xử lý lỗi**
  - Commit nếu thành công, rollback nếu có lỗi.
  - Đóng con trỏ.

2.5. roundrobinpartition()

```python
def roundrobinpartition(table_name, num_partitions, conn):
    cur = conn.cursor()
    sel_cur = conn.cursor()
    try:
        if num_partitions < 1:
            raise ValueError("Số phân vùng phải >= 1")
        for i in range(num_partitions):
            cur.execute(f"DROP TABLE IF EXISTS rrobin_part{i} CASCADE;")
            cur.execute(f"""
                CREATE TABLE rrobin_part{i} (
                    userid  INT,
                    movieid INT,
                    rating  FLOAT
                );
            """)
        sel_cur.execute(f"SELECT userid, movieid, rating FROM {table_name};")
        buckets = [[] for _ in range(num_partitions)]
        batch_size = 10000
        row_index = 0
        while True:
            rows = sel_cur.fetchmany(batch_size)
            if not rows:
                break
            for userid, movieid, rating in rows:
                part_idx = row_index % num_partitions
                buckets[part_idx].append((userid, movieid, rating))
                row_index += 1
            for i, bucket in enumerate(buckets):
                if len(bucket) >= 1000:
                    psycopg2.extras.execute_values(
                        cur,
                        f"INSERT INTO rrobin_part{i} (userid, movieid, rating) VALUES %s",
                        bucket,
                        page_size=1000,
                    )
                    buckets[i] = []
        sel_cur.close()
        for i, bucket in enumerate(buckets):
            if bucket:
                psycopg2.extras.execute_values(
                    cur,
                    f"INSERT INTO rrobin_part{i} (userid, movieid, rating) VALUES %s",
                    bucket,
                    page_size=1000,
                )
        last_rr_index = row_index - 1 if row_index > 0 else -1
        create_metadata(
            meta_table=ROBIN_META,
            meta_values={
                "table_name": table_name,
                "partition_count": num_partitions,
                "last_rr_index": last_rr_index,
            },
            conflict_updates={
                "partition_count": num_partitions,
                "last_rr_index": last_rr_index,
            },
            conn=conn,
        )

        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        cur.close()
```

Hàm roundrobinpartition()

- **Mục đích:** Phân mảnh bảng Ratings thành N phân mảnh vòng tròn, lưu vào các bảng rrobin_part{i}.

- **Kiểm tra đầu vào:** Đảm bảo num_partitions >= 1.
- **Tạo bảng phân mảnh:** Xóa các bảng rrobin_part{i} cũ và tạo mới với cùng lược đồ.
- **Phân phối dữ liệu:**
    - Dùng con trỏ riêng (sel_cur) để lấy bản ghi từ Ratings theo lô (batch_size = 10000) nhằm quản lý bộ nhớ.
    - Sử dụng danh sách buckets để thu thập bản ghi cho mỗi phân mảnh.
    - Gán bản ghi vào phân mảnh bằng part_idx = row_index % num_partitions (ví dụ: bản ghi 0 → phân mảnh 0, bản ghi 1 → phân mảnh 1, ...).
    - Tăng row_index để theo dõi số bản ghi.
- **Chèn theo lô:**
    - Khi bucket đạt 1000 bản ghi, chèn vào rrobin_part{i} bằng execute_values.
    - Chèn các bản ghi còn lại sau khi xử lý xong.
- **Lưu metadata:**
    - Tính last_rr_index = row_index - 1 (hoặc -1 nếu không có bản ghi).
    - Gọi create_metadata để lưu table_name, partition_count, last_rr_index vào rrobin_meta.
- **Kết thúc:**
    - Commit nếu thành công, rollback nếu có lỗi.
    - Đóng con trỏ.

2.6. rangeinsert()

```python
def rangeinsert(table_name, userid, movieid, rating, conn):
    cur = conn.cursor()
    try:
        if not (0.0 <= rating <= 5.0):
            raise ValueError(f"Rating phải trong khoảng [0.0, 5.0], nhận được: {rating}")
        cur.execute(
            f"""
            INSERT INTO {table_name}(userid, movieid, rating)
            VALUES (%s, %s, %s);
            """,
            (userid, movieid, rating),
        )
        cur.execute(
            f"""
            SELECT partition_count, min_val, max_val
              FROM {RANGE_META}
             WHERE table_name = %s;
            """,
            (table_name,),
        )
        meta = cur.fetchone()
        if not meta:
            raise RuntimeError("Phải gọi rangepartition trước khi rangeinsert")
        N, min_rating, max_rating = meta
        delta = (max_rating - min_rating) / N
        idx = N - 1
        for i in range(N):
            low = min_rating + i * delta
            high = (min_rating + (i + 1) * delta) if i < N - 1 else max_rating
            if (i == 0 and low <= rating <= high) or (i != 0 and low < rating <= high):
                idx = i
                break
        cur.execute(
            f"""
            INSERT INTO range_part{idx}(userid, movieid, rating)
            VALUES (%s, %s, %s);
            """,
            (userid, movieid, rating),
        )

        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        cur.close()
```

Hàm rangeinsert()

- **Mục đích:** Chèn bản ghi mới vào bảng Ratings và phân mảnh theo khoảng tương ứng.
- **Kiểm tra đầu vào:** Đảm bảo rating trong [0.0, 5.0].
- **Chèn vào bảng chính:** Thêm bản ghi vào Ratings bằng truy vấn tham số hóa.
- **Lấy metadata:**
    - Truy vấn range_meta để lấy partition_count, min_val, max_val.
    - Báo lỗi nếu metadata không tồn tại (chưa gọi rangepartition).

- **Tìm phân mảnh:** Tính delta = (max_rating - min_rating) / N.
- **Duyệt các phân mảnh:**
  - Phân mảnh đầu: low <= rating <= high.
  - Các phân mảnh khác: low < rating <= high.
- **Chèn vào phân mảnh:** Thêm bản ghi vào range_part{idx}.
- **Kết thúc**
  - Commit nếu thành công, rollback nếu có lỗi.
  - Đóng con trỏ

2.7. roundrobininsert()

```
def roundrobininsert(table_name, userid, movieid, rating, conn):
    cur = conn.cursor()
    try:
        if not (0.0 <= rating <= 5.0):
            raise ValueError(f"Rating phải trong khoảng [0.0, 5.0], nhận được: {rating}")
        cur.execute(
            f"""
            INSERT INTO {table_name}(userid, movieid, rating)
            VALUES (%s, %s, %s);
            """,
            (userid, movieid, rating),
        )
        cur.execute(
            f"""
            SELECT partition_count, last_rr_index
              FROM {ROBIN_META}
             WHERE table_name = %s;
            """,
            (table_name,),
        )
        meta = cur.fetchone()
        if not meta:
            raise RuntimeError("Phải gọi roundrobinpartition trước khi gọi roundrobininsert")
        num_partitions, last_idx = meta
        new_last_idx = last_idx + 1
        part_idx = new_last_idx % num_partitions
        cur.execute(
            f"""
            INSERT INTO rrobin_part{part_idx}(userid, movieid, rating)
            VALUES (%s, %s, %s);
            """,
            (userid, movieid, rating),
        )
        cur.execute(
            f"""
            UPDATE {ROBIN_META}
               SET last_rr_index = %s
             WHERE table_name = %s;
            """,
            (new_last_idx, table_name),
        )

        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        cur.close()
```

Hàm roundrobininsert()

- **Mục đích:** Chèn bản ghi mới vào bảng Ratings và phân mảnh vòng tròn tiếp theo.
- **Kiểm tra đầu vào:** Đảm bảo rating trong [0.0, 5.0].
- **Chèn vào bảng chính:** Thêm bản ghi vào Ratings.
- **Lấy metadata:**
  - Truy vấn rrobin_meta để lấy partition_count và last_rr_index.
  - Báo lỗi nếu metadata không tồn tại.
- **Tính phân mảnh:** Tính part_idx = (last_rr_index + 1) % num_partitions.
- **Chèn vào phân mảnh:** Thêm bản ghi vào rrobin_part{part_idx}.
- **Cập nhật metadata:** Cập nhật last_rr_index trong rrobin_meta.
- **Kết thúc**
  - Commit nếu thành công, rollback nếu có lỗi.
  - Đóng con trỏ

2.8. count_partitions()

```
def count_partitions(prefix, conn):
    cur = conn.cursor()
    try:
        cur.execute(
            """
            SELECT COUNT(*) FROM pg_stat_user_tables
            WHERE relname LIKE %s;
            """,
            (prefix + "%",),
        )
        return cur.fetchone()[0]
    finally:
        cur.close()
```

Hàm count_partitions()

- **Mục đích:** Đếm số bảng phân mảnh có tên bắt đầu bằng prefix (ví dụ: range_part%, rrobin_part%).
- Truy vấn pg_stat_user_tables để đếm bảng khớp với prefix%.
- Trả về số lượng bảng.
- Đóng con trỏ.

IV. Kiểm thử hệ thống

1. Mục đích kiểm thử

- Kiểm thử nhằm đảm bảo các hàm Python thao tác với PostgreSQL hoạt động chính xác, đúng logic phân mảnh, và dữ liệu được lưu trữ vào các bảng tương ứng theo yêu cầu. Việc kiểm thử giúp phát hiện lỗi sớm và xác nhận chương trình chạy đúng cả về chức năng và dữ liệu.

2. Chuẩn bị

- **Tệp kiểm thử:** Sử dụng tệp ratings.dat (bản rút gọn) chứa 100 dòng dữ liệu, đủ để kiểm tra hiệu quả việc phân mảnh và chèn.
- **Nội dung trong tệp:**

1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
….
4::420::3::844417004
4::432::3::844417070

3. Kiểm thử

- **Hàm LoadRatings():**

```
1  SELECT * FROM public.rating
2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
79	4	110	5
80	4	150	5
81	4	153	5
82	4	161	5
83	4	165	5
84	4	208	3
85	4	231	1
86	4	253	3
87	4	266	5
88	4	292	3
89	4	316	5
90	4	317	5
91	4	329	5
92	4	344	2
93	4	349	3
94	4	364	5
95	4	367	3
96	4	377	3
97	4	380	3
98	4	410	5
99	4	420	3
100	4	432	3

Bảng rating

Kết quả: Với bảng rating chứa đúng 100 bản ghi

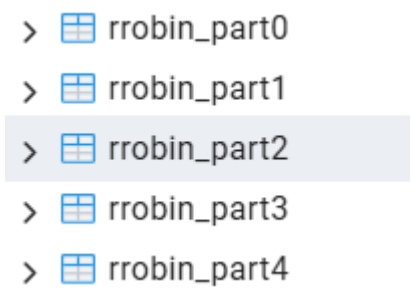
- **Hàm Range_Partiton()**: Tạo các bảng range_part0 → range_part4, mỗi bảng chứa đúng 20 rating

	userid integer	movieid integer	rating double precision
84	4	208	3
85	4	231	1
86	4	253	3
87	4	266	5
88	4	292	3
89	4	316	5
90	4	317	5
91	4	329	5
92	4	344	2
93	4	349	3
94	4	364	5
95	4	367	3
96	4	377	3
97	4	380	3
98	4	410	5
99	4	420	3
100	4	432	3
101	100	1	3

- **Hàm Range_Insert():**

Bảng rating sau khi Insert

- **Hàm RoundRobin_Partition():**



- **Bảng rrobin_part0:**

```
1  SELECT * FROM public.rrobin_part0
2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
3	1	370	5
4	1	520	5
5	1	594	5
6	2	376	3
7	2	733	3
8	2	858	2
9	2	1391	3
10	3	590	3.5
11	3	1288	3
12	3	1674	4.5
13	3	4995	4.5
14	3	6287	3
15	3	8529	4
16	4	21	3
17	4	153	5
18	4	253	3
19	4	329	5
20	4	377	3

- **Bảng rrobin_part1:**

Query | Query History

```
1  SELECT * FROM public.rrobin_part1
2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
3	1	377	5
4	1	539	5
5	1	616	5
6	2	539	3
7	2	736	3
8	2	1049	3
9	2	1544	3
10	3	1148	4
11	3	1408	3.5
12	3	3408	4
13	3	5299	3
14	3	6377	4
15	3	8533	4.5
16	4	34	5
17	4	161	5
18	4	266	5
19	4	344	2
20	4	380	3

- **Bảng rrobin_part2:**

Query | Query History

```
1  SELECT * FROM public.rrobin_part2
2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
3	1	420	5
4	1	586	5
5	2	110	5
6	2	590	5
7	2	780	3
8	2	1073	3
9	3	110	4.5
10	3	1246	4
11	3	1552	2
12	3	3684	4.5
13	3	5505	2
14	3	6539	5
15	3	8783	5
16	4	39	3
17	4	165	5
18	4	292	3
19	4	349	3
20	4	410	5

Total rows: 20 | Query complete 00:00:00.123

- **Bảng rrobin_part3:**

Query | Query History

```
1  SELECT * FROM public.rrobin_part3
2
```

Data Output | Messages | Notifications

Copy options

		movieid integer	rating double precision
3	1	466	5
4	1	588	5
5	2	151	3
6	2	648	2
7	2	786	3
8	2	1210	4
9	3	151	4.5
10	3	1252	4
11	3	1564	4.5
12	3	4535	4
13	3	5527	4.5
14	3	7153	4
15	3	27821	4.5
16	4	110	5
17	4	208	3
18	4	316	5
19	4	364	5
20	4	420	3

Total rows: 20 | Query complete 00:00:00.147

- **Bảng rrobin_part4:**

Query | Query History

```
1  SELECT * FROM public.rrobin_part4
2
```

Data Output | Messages | Notifications

	userid integer	movieid integer	rating double precision
3	1	480	5
4	1	589	5
5	2	260	5
6	2	719	3
7	2	802	2
8	2	1356	3
9	3	213	5
10	3	1276	3.5
11	3	1597	4.5
12	3	4677	4
13	3	5952	3.5
14	3	7155	3.5
15	3	33750	3.5
16	4	150	5
17	4	231	1
18	4	317	5
19	4	367	3
20	4	432	3

Total rows: 20 | Query complete 00:00:00.109

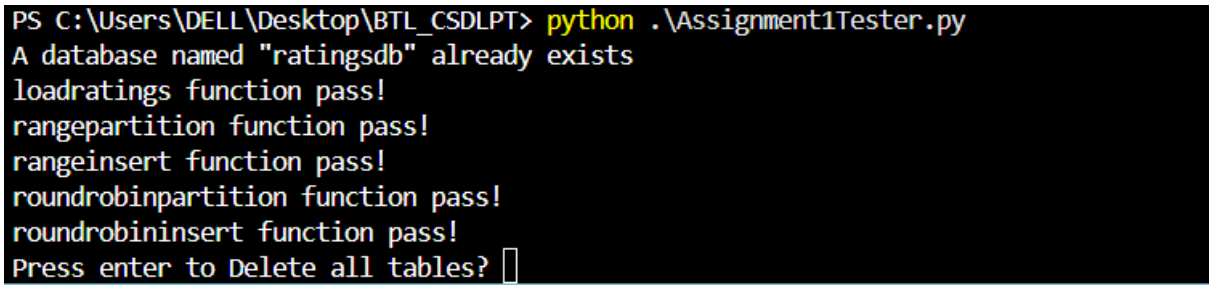
Kết quả: Phân mảnh và tạo ra 5 bảng rrobin_part0 đến rrobin_part4 với mỗi bảng chứa đúng 20 bản ghi.

- Hàm **RoundRobin_Insert():**

	userid integer	movieid integer	rating double precision
4	1	520	5
5	1	594	5
6	2	376	3
7	2	733	3
8	2	858	2
9	2	1391	3
10	3	590	3.5
11	3	1288	3
12	3	1674	4.5
13	3	4995	4.5
14	3	6287	3
15	3	8529	4
16	4	21	3
17	4	153	5
18	4	253	3
19	4	329	5
20	4	377	3
21	100	1	3
Total rows: 21	Query complete 00:00:00.132		

Bảng rrobin_part0 sau khi insert

4. Kết quả kiểm thử tổng hợp



Ảnh kết quả kiểm thử tổng hợp

- Qua kết quả khi chạy file Assignment1Tester.py .Tất cả các hàm được kiểm thử lần lượt: loadratings(), rangepartition(), rangeinsert(), roundrobinpartition() và roundrobininsert(). Mỗi hàm đều trả về trạng thái pass chứng minh rằng hệ thống phân mảnh và chèn dữ liệu vận hành đúng yêu cầu.

V. Kết luận.

- Thông qua quá trình triển khai và kiểm thử, nhóm đã xây dựng thành công hệ thống mô phỏng các phương pháp phân mảnh dữ liệu trong cơ sở dữ liệu phân tán sử dụng PostgreSQL và ngôn ngữ lập trình Python. Các chức năng chính như tải dữ liệu (loadratings), phân mảnh theo khoảng (rangepartition), phân mảnh vòng tròn (roundrobinpartition) và chèn dữ liệu mới vào đúng phân mảnh (rangeinsert, roundrobininsert) đều được thực hiện đầy đủ và chính xác.
- Mô phỏng thành công 2 cơ chế phân mảnh phổ biến: Range Partitioning và Round Robin Partitioning: Triển khai chính xác logic chia dữ liệu theo giá trị rating (Range) và chia tuần tự theo lượt (Round Robin)
- Đảm bảo các yếu tố: đúng số lượng bảng, không mất mát hay trùng lặp dữ liệu: Việc tạo bảng phân mảnh tuân theo đúng định dạng và dữ liệu sau khi phân tách có thể ghép lại chính xác như dữ liệu gốc ban đầu.
- Các bản ghi mới được chèn đúng vào phân mảnh phù hợp, không gây lỗi hoặc sai lệch: Hàm rangeinsert() và roundrobininsert() đều xử lý đúng logic và cập nhật đúng bảng tương ứng. Không phát sinh lỗi hệ thống, đồng thời dữ liệu được ghi nhận chính xác.
- Việc hoàn thành bài tập này không chỉ giúp nhóm củng cố kiến thức về cơ sở dữ liệu phân tán, mà còn rèn luyện kỹ năng triển khai, kiểm thử và xử lý dữ liệu lớn một cách có hệ thống. Đây là nền tảng quan trọng cho việc xây dựng các hệ thống dữ liệu phân tán quy mô lớn trong thực tiễn sau này.

Bảng phân chia công việc

Họ và tên	Mã sinh viên	Công việc
Cao Đức Việt	B22DCCN894	- Cài đặt và kết nối đến cơ sở dữ liệu PostgreSQL - Viết báo cáo - Tìm hiểu về **LoadRatings**
Bùi Ngọc Thiện	B22DCCN822	- Viết báo cáo - Tìm hiểu về **Range_Partiton** - Tìm hiểu về **Range_Insert()**
Đinh Ngọc Hân	B22DCCN282	- Viết báo cáo - Tìm hiểu về RoundRobin_Partition - Tìm hiểu về RoundRobin_Insert()