HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

BỘ MÔN CƠ SỞ DỮ LIỆU PHÂN TÁN

BÁO CÁO BÀI TẬP LỚN

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

Nhóm

STT	Tên	Mã sinh viên
1	Nguyễn Văn Học	B22DCCN352
2	Nguyễn Hữu Lộc	B22DCCN508
3	Nguyễn Mạnh Tuấn	B22DCCN760

GIẢNG VIÊN HƯỚNG DẪN: TS. KIM NGỌC BÁCH

Hà Nội – 2025

Bảng phân công

Thành viên	Thực hiện code	Viết báo cáo
Nguyễn Văn Học	Hàm loadratings , Hàm range_partition	Phần 4 và 5
Nguyễn Hữu Lộc	Hàm roundrobin_partition , Hàm roundrobin_insert_partition	Phần 3 và tổng hợp
Nguyễn Mạnh Tuấn	Hàm range_insert_partition	Phần 1 , 2 , 6

Danh mục hình ảnh

PHẦN I: GIỚI THIỆU

1.1. Mục tiêu

Bài tập này yêu cầu sinh viên mô phỏng các **phương pháp phân mảnh dữ liệu ngang** (horizontal data fragmentation) trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, cụ thể

là PostgreSQL hoặc MySQL. Nhiệm vụ chính là xây dựng một bộ các hàm Python để thực hiện ba công việc cốt lõi: (1) tải dữ liệu đánh giá phim từ tệp đầu vào vào một bảng quan hệ; (2) phân mảnh bảng dữ liệu này bằng hai phương pháp khác nhau là **phân mảnh theo khoảng** (range partitioning) và **phân mảnh theo vòng tròn** (round-robin partitioning); và (3) chèn các bản ghi dữ liệu mới vào đúng phân mảnh tương ứng sau khi đã phân mảnh.

Thông qua bài tập này, sinh viên sẽ có cơ hội tìm hiểu sâu và áp dụng các khái niệm quan trọng trong quản trị cơ sở dữ liệu phân tán. Cụ thể, các mục tiêu học tập bao gồm:

- Hiểu rõ khái niệm phân mảnh dữ liệu ngang và lợi ích của nó trong việc quản lý các tập dữ liệu lớn.
- Nắm vững và triển khai được hai kỹ thuật phân mảnh ngang phổ biến: phân mảnh theo khoảng giá trị và phân mảnh theo vòng tròn.
- Thực hành làm việc với một hệ quản trị cơ sở dữ liệu quan hệ (PostgreSQL hoặc MySQL) thông qua ngôn ngữ lập trình Python, bao gồm các thao tác tạo bảng, tải dữ liệu hàng loạt (batch insertion), và truy vấn dữ liệu.
- Phát triển kỹ năng thiết kế và sử dụng các bảng metadata để quản lý thông tin về các phân mảnh, một khía cạnh quan trọng trong các hệ thống phân mảnh thủ công.
- Rèn luyện kỹ năng xử lý các ràng buộc và yêu cầu kỹ thuật cụ thể trong một dự án phần mềm thực tế.

1.2. Dữ liệu đầu vào

Mô tả Dữ liệu ratings.dat

Dữ liệu đầu vào cho bài tập là tệp ratings.dat, một phần của bộ dữ liệu MovieLens 10M. Bộ dữ liệu này được cung cấp bởi GroupLens Research.

- **Nguồn:** http://files.grouplens.org/datasets/movielens/ml-10m.zip.
- **Định dạng:** Mỗi dòng trong tệp ratings.dat biểu diễn một đánh giá của một người dùng cho một bộ phim, có cấu trúc UserID::MovieID::Rating::Timestamp. Dấu :: được sử dụng làm ký tự phân tách các trường.
- **Nội dung:** Tệp chứa khoảng 10 triệu đánh giá. Các trường có ý nghĩa như sau:
  - UserID: Số nguyên, định danh người dùng.
  - MovieID: Số nguyên, định danh phim.
  - Rating: Số thực, điểm đánh giá trên thang điểm 5 sao, có thể chia nửa sao (ví dụ: 0.5, 1.0,..., 4.5, 5.0).
  - Timestamp: Số nguyên, số giây kể từ nửa đêm UTC ngày 1 tháng 1 năm 1970.

Bảng 1: Định dạng dữ liệu các trường trong ratings.dat và Kiểu dữ liệu Python tương ứng

Tên Trường	Kiểu Dữ liệu Gốc	Kiểu Dữ liệu Python Mục tiêu	Ví dụ
UserID	Chuỗi số	int	1

MovieID	Chuỗi số	int	122
Rating	Chuỗi số thập phân	float	5 hoặc 4.5
Timestamp	Chuỗi số	int	838985046

1.3. Yêu cầu kỹ thuật

Môi trường:

- Sử dụng Python 3.13.x
- Hệ điều hành: Ubuntu
- Hệ quản trị CSDL: PostgreSQL

Quy tắc triển khai:

- Không được thay đổi tiền tố tên bảng phân mảnh (*range_part*, *round_robin_part*).
- Không mã hóa cứng tên tệp đầu vào và tên cơ sở dữ liệu.
- Lược đồ bảng *Ratings* phải tuân thủ đúng mô tả.
- Kết nối đến cơ sở dữ liệu không được đóng bên trong các hàm đã triển khai.
- Không sử dụng biến toàn cục.

PHẦN 2: CƠ SỞ LÝ THUYẾT

2.1 Các nguyên tắc cốt lõi của phân mảnh dữ liệu

- Phân mảnh dữ liệu là một kỹ thuật trung tâm trong thiết kế cơ sở dữ liệu phân tán, bao gồm việc chia một quan hệ (relation) hoặc bảng (table) trong cơ sở dữ liệu thành nhiều phần nhỏ hơn, được gọi là các phân mảnh (fragments). Mỗi phân mảnh này chứa một tập con dữ liệu của quan hệ gốc và sau đó có thể được lưu trữ tại các vị trí (sites) hoặc nút (nodes) khác nhau trong một mạng máy tính phân tán. Quá trình này có thể được thực hiện theo chiều ngang, chia quan hệ thành các tập con các bộ (rows/tuples), hoặc theo chiều dọc, chia quan hệ thành các tập con các thuộc tính (columns/attributes), hoặc thậm chí là sự kết hợp của cả hai phương pháp, được gọi là phân mảnh hỗn hợp (hybrid fragmentation).

- Các mục tiêu chính mà kỹ thuật phân mảnh hướng tới bao gồm:
    - Tăng tính cục bộ của tham chiếu (Increased Locality of Reference)
    - Cho phép thực thi song song (Enable Parallel Execution)
    - Tính song song giữa các truy vấn (Interquery Parallelism)
    - Tính song song trong một truy vấn (Intraquery Parallelism
    - Cải thiện độ tin cậy và tính sẵn sàng (Improved Reliability and Availability)
    - Cân bằng tải (Load Balancing)
    - Tăng hiệu quả truy vấn (Query Efficiency)
    - Tính đầy đủ (Completeness)

- Khả năng tái tạo (Reconstructability)
- Tính rời rạc (Disjointness)

2.2 Các loại phân mảnh

2.2.1 Phân mảnh ngang

- Là quá trình chia một bảng quan hệ thành các tập hợp con (phân mảnh) dựa trên các hàng (rows) của bảng, sao cho mỗi phân mảnh chứa một tập hợp các hàng thỏa mãn một điều kiện cụ thể. Các phân mảnh ngang vẫn giữ nguyên cấu trúc (schema) của bảng gốc, tức là tất cả các cột (attributes) của bảng gốc được giữ lại, nhưng chỉ một phần dữ liệu (hàng) được lưu trong mỗi phân mảnh.
- VD: Trong bảng Ratings với Schema (UserID, MovieID, Rating), nếu phân mảnh ngang dựa trên giá trị cột Ratings, ta có thể chia như sau:
  + Phân mảnh 0: Các hàng có Rating từ 0 đến 2.5
  + Phân mảnh 1: Các hàng có Rating từ 2.5 đến 5
- **Ưu điểm:**
- + Cải thiện hiệu suất truy vấn khi chỉ cần truy cập một phân mảnh thay vì toàn bộ bảng.
  + Dễ dàng quản lý dữ liệu trong hệ thống phân tán (mỗi phân mảnh có thể lưu trên một máy chủ khác nhau).
  + Phù hợp với các truy vấn lọc dữ liệu dựa trên một điều kiện cụ thể.
- **Nhược điểm:**
  + Nếu truy vấn cần kết hợp dữ liệu từ nhiều phân mảnh, hiệu suất có thể giảm do cần thực hiện phép hợp (union).
  + Việc quản lý các phân mảnh đòi hỏi thêm logic để đảm bảo dữ liệu được chèn vào đúng phân mảnh.

2.2.2 Phân mảnh dọc

- Là quá trình chia bảng quan hệ thành các tập hợp con dựa trên các cột (attributes) thay vì các hàng. Mỗi phân mảnh chứa tất cả các hàng của bảng gốc nhưng chỉ bao gồm một tập hợp con các cột. Để đảm bảo tính duy nhất, khóa chính (hoặc một định danh duy nhất) thường được giữ lại trong tất cả các phân mảnh.
- VD:
- Với bảng Ratings (UserID, MovieID, Rating), ta có thể chia thành:
- + Phân mảnh 1: (UserID, MovieID)
- + Phân mảnh 2: (UserID, Rating)

- **Ưu điểm**

  + Giảm kích thước dữ liệu cần truy cập khi truy vấn chỉ liên quan đến một số cột cụ thể.

  + Tăng hiệu suất trong các hệ thống phân tán khi các cột thường xuyên được truy cập cùng nhau được lưu chung.

- Nhược điểm

  + Cần thực hiện nối (join) để tái tạo lại bảng gốc khi truy vấn cần dữ liệu từ nhiều phân mảnh.

  + Quản lý khóa chính để đảm bảo tính toàn vẹn dữ liệu có thể phức tạp.

1) Phân mảnh vòng tròn

- Là một dạng của **phân mảnh ngang**, trong đó các hàng của bảng được phân phối lần lượt qua các phân mảnh theo một chu trình tuần hoàn (round-robin). Không dựa trên giá trị của bất kỳ thuộc tính nào, các hàng được gán vào các phân mảnh một cách đều đặn theo thứ tự.
- **Cách hoạt động:**

  + Giả sử có N phân mảnh, hàng thứ nhất được gán vào phân mảnh 0, hàng thứ hai vào phân mảnh 1, ..., hàng thứ N vào phân mảnh N-1, rồi quay lại phân mảnh 0 cho hàng thứ N+1, và tiếp tục như vậy.

  + VD: Với N=3 phân mảnh, nếu có 6 bảng:

    - Hàng 1 -> Phân mảnh 0
    - Hàng 2 -> Phân mảnh 1
    - Hàng 3 -> Phân mảnh 2
    - Hàng 4 -> Phân mảnh 0
    - Hàng 5 -> Phân mảnh 1
    - Hàng 6 -> Phân mảnh 2

- **Ưu điểm**

  + Phân phối dữ liệu đều đặn, tránh tình trạng một phân mảnh chưa quá nhiều dữ liệu

  + Đơn giản để triển khai, không cần điều kiện phúc tạp để xác định phân mảnh

- **Nhược điểm**

  + Không tối ưu cho các truy vấn dựa trên giá trị cụ thể (vì dữ liệu không được tổ chức theo điều kiện).

  + Việc tìm kiếm một hàng cụ thể có thể yêu cầu kiểm tra tất cả các phân mảnh.

2) Phân mảnh theo khoảng

- Là một dạng của phân mảnh ngang, trong đó các hàng được chia vào các phân mảnh dựa trên giá trị của một thuộc tính nằm trong một khoảng giá trị xác định. Các khoảng này thường được xác định sao cho dữ liệu được phân phối đồng đều (uniform ranges).
- **Cách hoạt động:**
    + Thuộc tính được chọn (ví dụ: Rating) được chia thành các khoảng giá trị.
    + Mỗi phân mảnh chứa các hàng có giá trị thuộc tính nằm trong khoảng tương ứng.
    + VD: Với bảng Ratings và N=3 phân mảnh, thuộc tính Rating (từ 0 đến 5) được chia như sau:
        - Phân mảnh 0: Rating từ 0 đến 1.67
        - Phân mảnh 1: Rating từ 1.67 đến 3.34
        - Phân mảnh 2: Rating từ 3.34 đến 5
- **Ưu điểm**
    + Tối ưu cho các truy vấn dựa trên khoảng giá trị (ví dụ: tìm tất cả các đánh giá có Rating từ 3 đến 5).
    + Phân phối dữ liệu có thể được điều chỉnh để cân bằng tải giữa các phân mảnh.
- **Nhược điểm**
    + Nếu dữ liệu không phân phối đều (ví dụ: nhiều giá trị Rating tập trung ở một khoảng), một số phân mảnh có thể chứa nhiều dữ liệu hơn các phân mảnh khác.
    + Cần xác định các khoảng giá trị phù hợp để đảm bảo phân phối đồng đều.

PHẦN 3: THIẾT KẾ HỆ THỐNG VÀ CƠ SỞ DỮ LIỆU

3.1.Thiết kế các hàm

3.1.1. Hàm loadratings

Bước 1: Chuẩn bị Bảng (Bên trong Giao dịch)

- **Bắt đầu đo thời gian:** Ghi lại thời gian bắt đầu để tính toán hiệu suất ở cuối.
- **Xóa và Tạo bảng:** Hàm thực thi lệnh DROP TABLE IF EXISTS để xóa bảng cũ và CREATE TABLE để tạo một bảng mới hoàn toàn rỗng.

Bước 2: Xử lý và Chuẩn bị Dữ liệu vào Bộ đệm (Buffer)

- **Tạo Bộ đệm trong Bộ nhớ:** buffer = StringIO() tạo ra một đối tượng hoạt động giống như một file văn bản nhưng hoàn toàn nằm trong bộ nhớ RAM. Việc này giúp tránh phải đọc/ghi xuống ổ đĩa, làm tăng tốc độ.
- **Đọc File và Ghi vào Bộ đệm:**
    - Hàm mở và đọc từng dòng của tệp ratings.dat trên đĩa cứng.
    - Thay vì chèn trực tiếp, nó xử lý mỗi dòng và ghi ra một chuỗi mới vào buffer. Chuỗi này có định dạng đặc biệt: các trường userid, movieid, rating được **ngăn cách bởi một dấu tab**. Đây là định dạng chuẩn mà lệnh COPY của PostgreSQL có thể đọc được.
- **Reset Bộ đệm:** buffer.seek(0) là một bước cực kỳ quan trọng. Nó di chuyển con trỏ đọc của buffer về lại vị trí đầu tiên, để chuẩn bị cho bước tiếp theo đọc toàn bộ nội dung của bộ đệm.

Bước 3: Tải dữ liệu Tốc độ cao với COPY

- Đây là trái tim của sự tối ưu hóa. Hàm gọi cur.copy_from(...).
- Lệnh này sử dụng trực tiếp tính năng COPY của PostgreSQL, được thiết kế riêng cho việc tải dữ liệu số lượng lớn (bulk-loading).
- Thay vì chương trình Python gửi hàng triệu lệnh INSERT riêng lẻ, nó chỉ cần truyền một luồng (stream) dữ liệu từ buffer cho server PostgreSQL, và server sẽ tự làm phần việc còn lại một cách hiệu quả nhất. Tốc độ nhanh hơn INSERT rất nhiều lần.

Bước 4: Tối ưu hóa bằng cách Thêm Khóa chính sau cùng

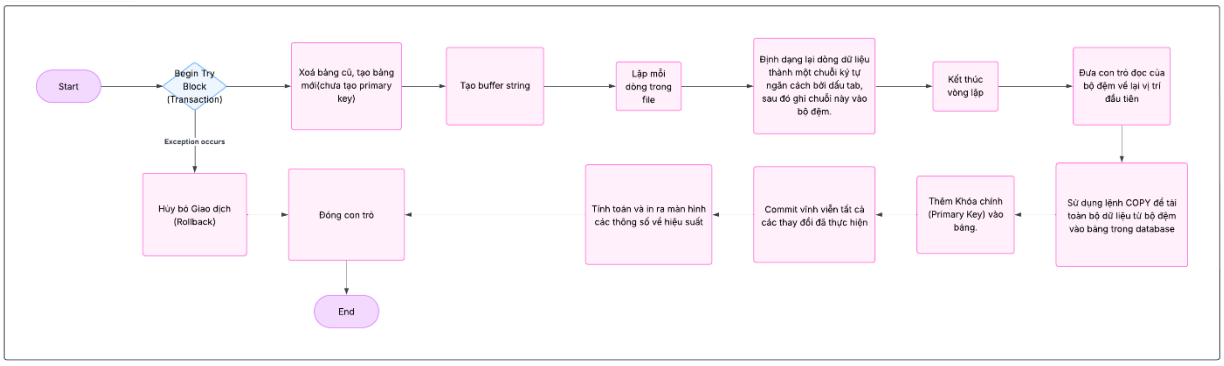
- Sau khi **toàn bộ** 10 triệu dòng dữ liệu đã nằm trong bảng, hàm mới thực thi lệnh ALTER TABLE ... ADD PRIMARY KEY ....
- Đây là một kỹ thuật tối ưu hiệu suất phổ biến. Việc chèn dữ liệu vào một bảng không có index (và Primary Key là một loại index) sẽ nhanh hơn. Sau đó, việc xây dựng index một lần trên toàn bộ dữ liệu sẽ hiệu quả hơn là cập nhật index sau mỗi lần chèn một dòng.

Bước 5: Commit

- Khi tất cả các bước trên đã thành công, openconnection.commit() được gọi.
- Lệnh này sẽ lưu vĩnh viễn **toàn bộ** các thay đổi trong một lần: việc tạo bảng, việc COPY 10 triệu dòng dữ liệu, và việc thêm khóa chính.

Bước 6: Thống kê và Báo cáo Hiệu suất

- Hàm tính toán tổng thời gian thực hiện.
- Nó truy vấn lại database để lấy tổng số bản ghi đã được chèn và in ra các thông số hiệu suất như tổng thời gian, tổng số bản ghi, và tốc độ xử lý trung bình (bản ghi/giây).



Hình 1 Sơ đồ luồng hoạt động của hàm loadratings

3.1.2. Hàm rangepartion

Bước 1: Chuẩn bị Bảng Siêu dữ liệu (Metadata)

- **Tạo bảng Metadata:** Hàm thực thi lệnh CREATE TABLE IF NOT EXISTS RangePartitionsMetadata. Thao tác này đảm bảo rằng có một bảng để lưu trữ thông tin về các phân mảnh. Nếu bảng này đã tồn tại, lệnh sẽ không làm gì cả. Bảng này có các cột để lưu chỉ số, tên bảng, và giới hạn rating của mỗi phân mảnh.
- **Xóa Metadata cũ:** Ngay sau đó, thực thi DELETE FROM RangePartitionsMetadata;. Thao tác này dọn dẹp mọi thông tin phân mảnh từ lần chạy trước, đảm bảo metadata chỉ ghi lại thông tin của lần phân mảnh hiện tại.

Bước 2: Tính toán các Khoảng Giá trị Rating

- **Tính kích thước khoảng:** Tính range_size = 5.0 / numberofpartitions. Ví dụ, nếu muốn chia thành 5 phân mảnh (numberofpartitions=5), thì kích thước mỗi khoảng sẽ là 5.0 / 5 = 1.0.
- **Xác định các ranh giới:** Hàm tạo ra một danh sách các điểm ranh giới. Với ví dụ trên, danh sách boundaries sẽ là [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]. Danh sách này sẽ được dùng để xác định cận trên và cận dưới cho mỗi phân mảnh.

Bước 3: Tạo và Phân phối Dữ liệu vào từng Phân mảnh Hàm bắt đầu một vòng lặp chạy numberofpartitions lần (ví dụ: 5 lần, với i từ 0 đến 4). Trong mỗi vòng lặp:

- **Chuẩn bị bảng phân mảnh:**
    - Xác định tên bảng con, ví dụ range_part0.
    - Xóa bảng con này nếu nó đã tồn tại từ lần chạy trước (DROP TABLE IF EXISTS).

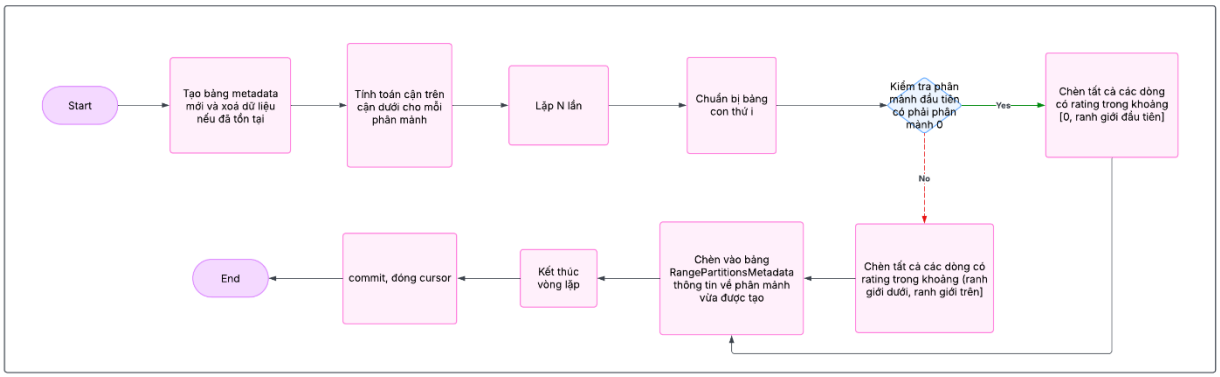
- Tạo ra một bảng con mới, rỗng, với cấu trúc giống hệt bảng ratings (CREATE TABLE).

- **Chèn dữ liệu theo khoảng:** Đây là logic cốt lõi.
  - **Trường hợp đặc biệt (i = 0):** Đối với phân mảnh đầu tiên (range_part0), nó sẽ lấy tất cả các dòng từ bảng ratings có giá trị rating từ 0 đến ranh giới đầu tiên. Câu lệnh SQL sẽ là WHERE rating >= 0 AND rating <= 1.0.
  - **Các trường hợp khác (i > 0):** Đối với các phân mảnh còn lại, nó sẽ lấy các dòng có rating lớn hơn ranh giới dưới và nhỏ hơn hoặc bằng ranh giới trên. Ví dụ, cho range_part1 (i=1), câu lệnh SQL sẽ là WHERE rating > 1.0 AND rating <= 2.0. Việc dùng > ở cận dưới đảm bảo mỗi dòng dữ liệu chỉ thuộc về duy nhất một phân mảnh.

- **Ghi lại vào metadata**
  - Ngay sau khi chèn dữ liệu vào một bảng phân mảnh (ví dụ: range_part1), hàm sẽ chèn một dòng mới vào bảng RangePartitionsMetadata.
  - Dòng này ghi lại đầy đủ thông tin: chỉ số là 1, tên bảng là 'range_part1', cận dưới là 1.0, và cận trên là 2.0.

Bước 4: Hoàn tất và Lưu thay đổi

- Sau khi vòng lặp kết thúc và tất cả các phân mảnh đã được tạo và điền dữ liệu, hàm gọi con.commit() để lưu vĩnh viễn tất cả các thay đổi (tạo bảng, chèn dữ liệu) vào cơ sở dữ liệu.
- Cuối cùng, đóng con trỏ cur.close() để giải phóng tài nguyên.



Hình 2 Sơ đồ luồng hoạt động hàm rangepartition

3.1.3. Hàm rangeinsert

Bước 1: Chèn vào Bảng Gốc

- Hàm thực thi một lệnh INSERT để chèn dòng dữ liệu mới (UserID, MovieID, Rating) vào bảng ratings chính.
- Tại thời điểm này, thay đổi vẫn chỉ là tạm thời và chưa được lưu vĩnh viễn.

Bước 2: Tra cứu RangePartitionsMetadata

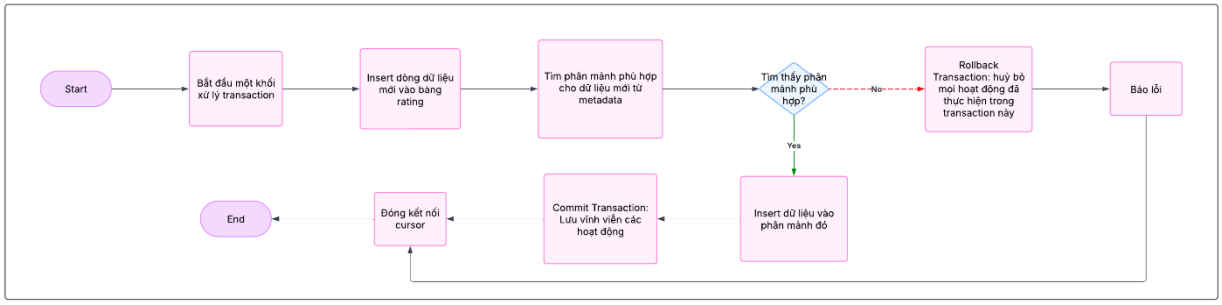
- Hàm thực thi một câu lệnh SELECT trên bảng RangePartitionsMetadata.
- Tìm kiếm một dòng trong bảng metadata nơi mà giá trị rating **lớn hơn** min_rating_exclusive và **nhỏ hơn hoặc bằng** max_rating_inclusive.
- Câu lệnh SELECT này sẽ tìm thấy dòng metadata tương ứng và trả về tên bảng.

Bước 3: Chèn vào Bảng Phân mảnh

- Sau khi xác định được tên bảng phân mảnh phù hợp, hàm thực thi một lệnh INSERT thứ hai.
- Lần này, chèn cùng một dòng dữ liệu vào bảng phân mảnh phù hợp.

Bước 4: Kiểm tra và commit

- **Kiểm tra:** Hàm chạy một lệnh SELECT COUNT(*) trên bảng range_part2 để kiểm tra xem dòng dữ liệu vừa rồi có thực sự được chèn vào hay không. Nếu không tìm thấy, sẽ báo lỗi.
- **Commit:** Nếu tất cả các bước trên thành công không có lỗi, lệnh con.commit() được gọi. Lệnh này sẽ lưu vĩnh viễn **cả hai lệnh INSERT** (ở Bước 1 và Bước 3) vào cơ sở dữ liệu.



Hình 3 Sơ đồ luồng hoạt động hàm rangeinsert

3.1.4. Hàm roundrobinpartition

Bước 1: Chuẩn bị Bảng Trạng thái (RoundRobinState)

- **Tạo bảng:** Hàm tạo bảng metadata RoundRobinState nếu nó chưa tồn tại. Bảng này sẽ được dùng bởi hàm roundrobininsert sau này.
- **Xóa trạng thái cũ:** Xóa mọi dữ liệu cũ trong bảng RoundRobinState để đảm bảo trạng thái được thiết lập lại từ đầu cho lần phân mảnh này.

Bước 2: Chuẩn bị các Bảng Phân mảnh

- Hàm chạy một vòng lặp N lần (với N là numberofpartitions).
- Trong mỗi vòng lặp, thực hiện:
    - **Xóa bảng cũ:** DROP TABLE IF EXISTS rrobin_part{i}; để dọn dẹp kết quả từ lần chạy trước.
    - **Tạo bảng mới:** CREATE TABLE rrobin_part{i} (...); để tạo ra một bảng con rỗng, sẵn sàng nhận dữ liệu.

Bước 3: Phân phối Dữ liệu

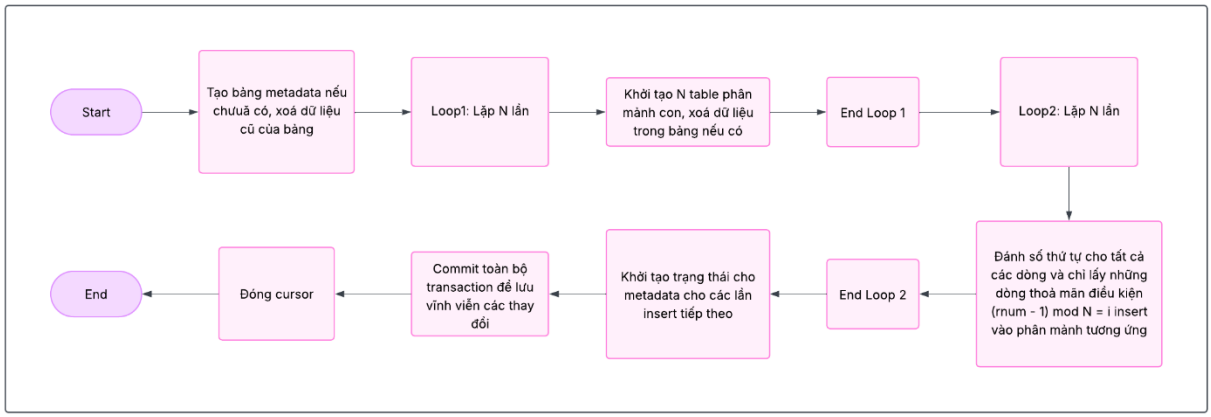
- Hàm chạy một vòng lặp N lần, để điền dữ liệu vào các bảng đã tạo ở trên.
- Trong mỗi vòng lặp (ví dụ với i=0) thực thi.
    - **Đánh số thứ tự:** Phần SELECT ..., ROW_NUMBER() OVER () as rnum FROM {ratingstablename} hoạt động như một "cỗ máy đánh số". Nó đọc toàn bộ bảng ratings và gán cho mỗi dòng một số thứ tự tuần tự duy nhất (1, 2, 3, 4, 5, ...).
    - **Phép chia lấy dư (Modulo):** Phần WHERE MOD(rnum - 1, {numberofpartitions}) = {i}
        - Lấy số thứ tự của dòng (rnum), trừ đi 1 để bắt đầu từ 0.
        - Chia số đó cho N và lấy phần dư (MOD).
        - Chỉ chọn những dòng có phần dư bằng với chỉ số i của vòng lặp hiện tại.

Bước 4: Khởi tạo Trạng thái cho lần chèn tiếp theo

- Sau khi tất cả dữ liệu đã được phân phối, hàm chèn một dòng duy nhất vào bảng RoundRobinState.
- Dòng này có giá trị (singleton_id=1, next_partition_index=0, num_partitions=N).
- Việc đặt next_partition_index=0 là để báo cho hàm roundrobininsert biết rằng, lần chèn mới đầu tiên sau khi phân mảnh sẽ bắt đầu từ bảng rrobin_part0.

Bước 5: Hoàn tất và Lưu thay đổi

- Cuối cùng, con.commit() được gọi để lưu vĩnh viễn tất cả các thay đổi: việc tạo bảng, phân phối dữ liệu, và khởi tạo trạng thái.
- Đóng con trỏ cursor



Hình 4 Sơ đồ luồng hoạt động hàm roundrobinpartition

3.1.5. Hàm roundrobininsert

Bước 1: Chèn vào Bảng Gốc

- Đầu tiên là chèn dòng dữ liệu mới vào bảng ratings chính.
- Thay đổi này vẫn đang ở trạng thái tạm thời trong giao dịch.

Bước 2: Đọc Trạng thái ("Bộ nhớ")

- Hàm thực thi một lệnh SELECT để đọc dòng duy nhất từ bảng RoundRobinState.
- Từ đó, nó lấy ra được hai thông tin quan trọng:
    - next_partition_index (Đây là chỉ số của bảng con sẽ nhận dữ liệu lần này).
    - num_partitions (Tổng số bảng con).

Bước 3: Chèn vào Bảng Phân mảnh Đích

- Dựa vào next_partition_index vừa đọc được, hàm xác định được bảng đích là rrobin_part{}.
- Thực thi một lệnh INSERT thứ hai để chèn dòng dữ liệu vào bảng rrobin_part{}.

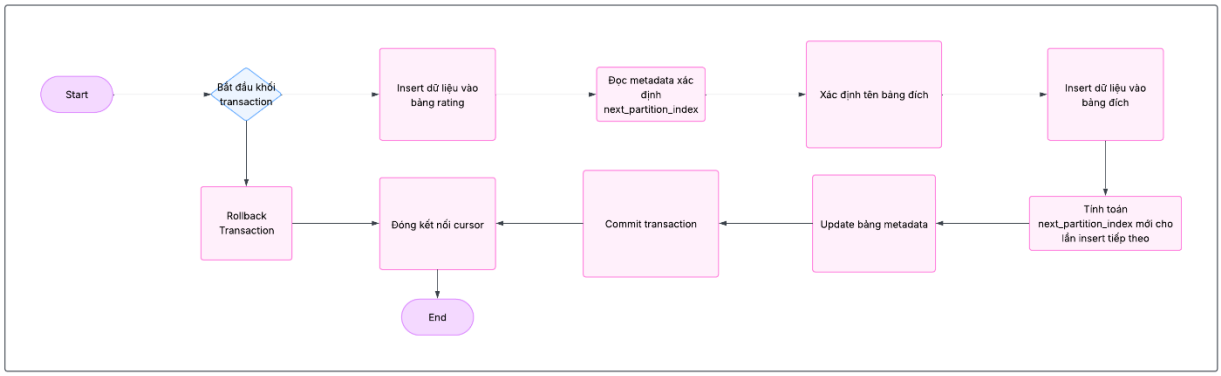
Bước 4: Cập nhật Trạng thái cho Tương lai

- Đây là bước chuẩn bị cho lần gọi hàm tiếp theo.
- **Tính toán chỉ số mới:** Hàm tính toán chỉ số tiếp theo bằng công thức (next_partition_index + 1) % num_partitions. Phép chia lấy dư (%) đảm bảo rằng nếu chỉ số hiện tại là N-1, chỉ số tiếp theo sẽ quay vòng về 0.

- **Cập nhật "bộ nhớ":** Hàm thực thi một lệnh UPDATE trên bảng RoundRobinState, đặt giá trị của next_partition_index.

Bước 5: Commit

- Khi tất cả các bước trên đã thành công, lệnh con.commit() được gọi.
- Lệnh này sẽ lưu vĩnh viễn **cả 3 thay đổi** vào cơ sở dữ liệu:
    - Dữ liệu mới trong bảng ratings.
    - Dữ liệu mới trong bảng rrobin_part{}.
    - Giá trị next_partition_index mới trong bảng RoundRobinState.



Hình 5 Sơ đồ luồng hoạt động hàm roundrobininsert

3.2. Thiết kế cơ sở dữ liệu

3.2.1. Bảng ratings (Bảng chính)

- **Mục đích:** Đây là bảng trung tâm, chứa toàn bộ 10 triệu dòng dữ liệu gốc được tải từ tệp ratings.dat. Nó đóng vai trò là nguồn dữ liệu chính, không bị thay đổi trong quá trình phân mảnh và luôn chứa bản sao đầy đủ của tất cả các đánh giá.
- **Cấu trúc:**
    - UserID (INTEGER): Mã định danh của người dùng đã thực hiện đánh giá.
    - MovieID (INTEGER): Mã định danh của bộ phim được đánh giá.
    - Rating (FLOAT): Điểm đánh giá mà người dùng đã cho, trên thang điểm 5 sao và có thể chia nửa sao.
- Các bảng phân mảnh cũng có cấu trúc tương tự bảng này

3.2.2. Bảng RangePartitionsMetadata (Bảng metadata cho Range Partition)

- **Mục đích:** Bảng này hoạt động như một "bản đồ" ghi lại các quy tắc phân chia của phương pháp Range Partition. Khi hàm rangeinsert cần chèn một dòng mới, nó sẽ tra cứu bảng này để tìm ra chính xác tên của bảng con (partition) nào tương ứng với giá trị rating của dòng dữ liệu đó.
- **Cấu trúc:**
    - partition_index (INTEGER): Chỉ số của phân mảnh (0, 1, 2,...).
    - partition_table_name (VARCHAR): Tên thật của bảng phân mảnh (ví dụ: 'range_part0', 'range_part1').
    - min_rating_exclusive (FLOAT): Cận dưới (không bao gồm) của khoảng rating cho phân mảnh này.
    - max_rating_inclusive (FLOAT): Cận trên (bao gồm) của khoảng rating cho phân mảnh này.

3.2.3. Bảng RoundRobinState (Bảng metadata cho Round Robin Partition)

- **Mục đích:** Bảng này đóng vai trò là "bộ nhớ" cho phương pháp Round Robin. Vì mỗi lần chèn dữ liệu (roundrobininsert) là một lần chạy chương trình riêng biệt, bảng này giúp hệ thống "ghi nhớ" được phân mảnh nào sẽ là nơi nhận dữ liệu ở lượt tiếp theo, đảm bảo tính tuần tự của vòng xoay.
- **Cấu trúc:** Bảng này luôn chỉ có duy nhất một dòng.
    - singleton_id (INTEGER): Một mã định danh kỹ thuật (luôn là 1) để đảm bảo bảng chỉ có một dòng trạng thái.
    - next_partition_index (INTEGER): Chỉ số của phân mảnh sẽ nhận dữ liệu ở lần chèn **tiếp theo**. Đây là giá trị được đọc và cập nhật liên tục bởi hàm roundrobininsert.
    - num_partitions (INTEGER): Tổng số phân mảnh đã được tạo ra. Con số này cần thiết để tính toán việc quay vòng khi chỉ số đi hết một lượt.

PHẦN 4: CÀI ĐẶT CHƯƠNG TRÌNH

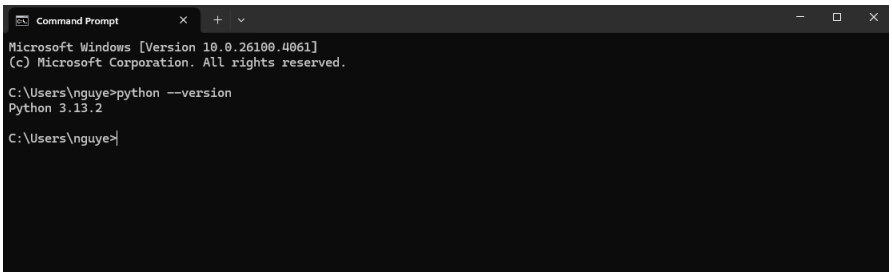
1.1. Môi trường thực hiện

- Để thực hiện việc mô phỏng hai phương pháp phân mảnh dữ liệu phổ biến trong hệ quản trị cơ sở dữ liệu phân tán, nhóm đã tiến hành xây dựng một môi trường lập trình trên máy tính cá nhân. Môi trường này bao gồm các công cụ và phần mềm mã nguồn mở, đảm bảo hỗ trợ đầy đủ cho việc xử lý dữ liệu, triển khai cơ sở dữ liệu, và thực hiện phân mảnh.
- Cụ thể, môi trường bao gồm:
    • Hệ điều hành: Windows 10 hoặc 11 (64-bit)

• Ngôn ngữ lập trình: Python 3.13.x
• Hệ quản trị cơ sở dữ liệu: PostgreSQL (phiên bản từ 13 trở lên)
• Công cụ quản lý cơ sở dữ liệu: pgAdmin 4
• Mã nguồn : https://github.com/Hocnv0204/CSDLPT

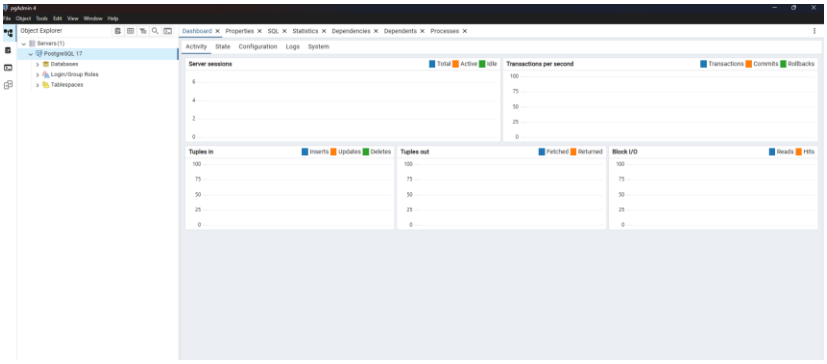
1.2. Cài đặt Python

- Truy cập trang chính thức : https://www.python.org/downloads/
- Chọn phiên bản python 3.13.x
- Chạy file python vừa tải → tích chọn "Add Python 3.13 to PATH"
- Hoàn tất và kiểm tra bằng lệnh python –version trong Command Prompt



Hình 6 Cài đặt Python

- Truy cập trang chính thức : https://www.postgresql.org/download/
- Chọn Windows → click chọn Download the installer
- Chọn phiên bản 17.5 (mới nhất)
- Chạy file tải xuống
- Nhập password và để cổng mặc định là 5432
- Mở menu Start và tìm kiếm pgAdmin 4



Hình 7 Cài đặt PostgreSQL

1.3. Cài đặt các thư viện cần thiết

- Cài đặt thư viện psycopg2 hoặc psycopg2-binary : Đây là thư viện quan trọng nhất. Nó là một trình điều khiển (adapter) cơ sở dữ liệu PostgreSQL cho Python, cho phép các tập lệnh Python của bạn kết nối và tương tác với cơ sở dữ liệu PostgreSQL
- Mở Command Prompt và thực hiện câu lệnh : pip install psycopg2-binary

```
C:\Users\nguye>pip install psycopg2-binary
Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.10-cp313-cp313-win_amd64.whl.metadata (4.8 kB)
Downloading psycopg2_binary-2.9.10-cp313-cp313-win_amd64.whl (2.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.6/2.6 MB 10.7 MB/s eta 0:00:00
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.10

[notice] A new release of pip is available: 24.3.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Hình 8 Cài đặt thư viện psycopg-binary

- Mở Command Prompt và thực hiện câu lệnh : pip install python-dotenv

```
C:\Users\nguye>pip install python-dotenv
Collecting python-dotenv
  Downloading python_dotenv-1.1.0-py3-none-any.whl.metadata (24 kB)
Downloading python_dotenv-1.1.0-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.0

[notice] A new release of pip is available: 24.3.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Hình 9 Cài đặt thư viện python-dotenv

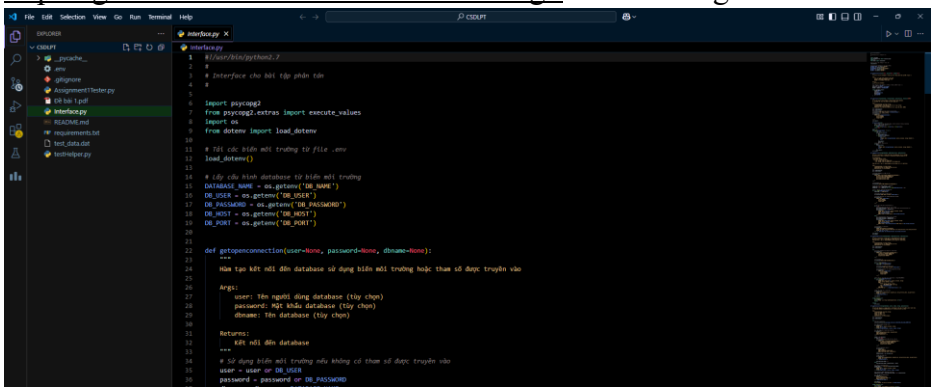
1.4. Chuẩn bị dữ liệu

- Truy cập liên kết : http://files.grouplens.org/datasets/movielens/ml-10m.zip.
- File zip được tải xuống bao gồm

allbut.pl	753	381	PL File	1/6/2009 12:06...	A0EEF3C2
movies.dat	522,197	176,630	DAT File	1/6/2009 6:02 ...	250483F3
ratings.dat	265,105,635	64,374,010	DAT File	1/6/2009 6:08 ...	0E67AF00
README.html	11,563	4,094	Microsoft Edge H...	1/30/2016 12:3...	58CC1488
split_ratings.sh	1,304	472	sh_auto_file	2/17/2016 12:0...	074407B1
tags.dat	3,584,119	1,009,348	DAT File	1/6/2009 6:08 ...	0F6D8DD1

Hình 10 Thư mục dữ liệu

- Mở Command Prompt và chạy lệnh : git clone https://github.com/Hocnv0204/CSDLPT.git để tải mã nguồn



Hình 11 Mã nguồn

1.5. Thực hiện chương trình

6.1 Hàm create_db()

```python
def create_db(dbname):
    """
    Tạo database bằng cách kết nối đến user và database mặc định của Postgres.
    Hàm kiểm tra xem database đã tồn tại chưa, nếu chưa thì tạo mới.

    Args:
        dbname: Tên database cần tạo
    """
    # Kết nối đến database mặc định
    con = getopenconnection(dbname='postgres')
    con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    cur = con.cursor()

    # Kiểm tra xem database đã tồn tại chưa
    cur.execute('SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname=\'%s\'' % (dbname,))
    count = cur.fetchone()[0]
    if count == 0:
        cur.execute('CREATE DATABASE %s' % (dbname,))  # Tạo database
    else:
        print('Database "{0}" đã tồn tại'.format(dbname))

    # Dọn dẹp
    cur.close()
    con.close()
```

Hình 12 Hàm create_db

6.2 Hàm count_partitions()

```python
def count_partitions(prefix, openconnection):
    """
    Hàm đếm số lượng bảng có chứa @prefix trong tên.

    Args:
        prefix: Tiền tố cần tìm
        openconnection: Kết nối database

    Returns:
        Số lượng bảng tìm thấy
    """
    con = openconnection
    cur = con.cursor()
    cur.execute("select count(*) from pg_stat_user_tables where relname like " + "'" + prefix + "%';")
    count = cur.fetchone()[0]
    cur.close()

    return count
```

Hình 13 Hàm count_partitions

6.3 Hàm loadratings()

```python
def loadratings(ratingstablename, ratingsfilepath, openconnection):    "loadratings": Unknown word.
    """
    Hàm tải dữ liệu từ file ratings vào bảng ratings trong PostgreSQL.    "liệu": Unknown word.
    Sử dụng phương pháp đơn giản và hiệu quả nhất.    "dụng": Unknown word.

    Args:
        ratingstablename: Tên bảng ratings    "ratingstablename": Unknown word.
        ratingsfilepath: Đường dẫn đến file chứa dữ liệu ratings    "ratingsfilepath": Unknown word.
        openconnection: Kết nối database    "openconnection": Unknown word.
    """
    start_time = time.time()
    print(f"\nBắt đầu tải dữ liệu vào {ratingstablename}...")    "liệu": Unknown word.
    print(f"Thời gian bắt đầu: {datetime.now().strftime('%H:%M:%S')}")    "Thời": Unknown word.

    cur = openconnection.cursor()    "openconnection": Unknown word.

    try:
        # 1. Tạo bảng    "bảng": Unknown word.
        cur.execute(f"DROP TABLE IF EXISTS {ratingstablename};")    "ratingstablename": Unknown word.
        cur.execute(f"""
            CREATE TABLE {ratingstablename} (    "ratingstablename": Unknown word.
                userid INT,
                movieid INT,    "movieid": Unknown word.
                rating FLOAT
            );
        """)
```

Hình 14 Hàm loadratings - 1

```
    # 2. Xử lý và tải dữ liệu    "liệu": Unknown word.
    buffer = StringIO()
    with open(ratingsfilepath, 'r') as f:    "ratingsfilepath": Unknown word.
        for line in f:
            parts = line.strip().split('::')
            if len(parts) >= 3:
                buffer.write(f"{parts[0]}\t{parts[1]}\t{parts[2]}\n")
    buffer.seek(0)

    # 3. Sử dụng COPY để tải dữ liệu    "dụng": Unknown word.
    cur.copy_from(buffer, ratingstablename, sep='\t', columns=('userid', 'movieid', 'rating'))    "ratingstablename": Unknown word.

    # 4. Tạo primary key
    cur.execute(f"ALTER TABLE {ratingstablename} ADD PRIMARY KEY (userid, movieid);")    "ratingstablename": Unknown word.

    # 5. Commit transaction
    openconnection.commit()    "openconnection": Unknown word.

    # 6. In thống kê    "thống": Unknown word.
    end_time = time.time()
    total_time = end_time - start_time
    print(f"\nThống kê hiệu suất:")    "Thống": Unknown word.
    print(f"Tổng thời gian thực hiện: {total_time:.2f} giây")    "Tổng": Unknown word.
    print(f"Thời gian kết thúc: {datetime.now().strftime('%H:%M:%S')}")    "Thời": Unknown word.

    # Đếm số bản ghi đã tải
    cur.execute(f"SELECT COUNT(*) FROM {ratingstablename};")    "ratingstablename": Unknown word.
    total_records = cur.fetchone()[0]
    print(f"Tổng số bản ghi đã tải: {total_records:,}")    "Tổng": Unknown word.
    print(f"Tốc độ xử lý trung bình: {total_records/total_time:.0f} bản ghi/giây")    "trung": Unknown word.

except Exception as e:
    openconnection.rollback()    "openconnection": Unknown word.
    print(f"Error loading data: {str(e)}")
    raise e

finally:
    cur.close()
```

Hình 15 Hàm loadratings - 2

Data Output | Messages | Notifications

	userid [PK] integer	movieid [PK] integer	rating double precision
1	1	122	5
2	1	185	5
3	1	231	5
4	1	292	5
5	1	316	5
6	1	329	5
7	1	355	5
8	1	356	5
9	1	362	5
10	1	364	5
11	1	370	5
12	1	377	5
13	1	420	5

Total rows: 10000054 | Query complete 00:00:02.956

Hình 16 Bảng ratings

```
Thống kê hiệu suất:
Tổng thời gian thực hiện: 12.64 giây
Thời gian kết thúc: 22:45:48
Tổng số bản ghi đã tải: 10,000,054
Tốc độ xử lý trung bình: 791182 bản ghi/giây
loadratings function pass!
```

Hình 17 Thời gian thực thi hàm loadratings

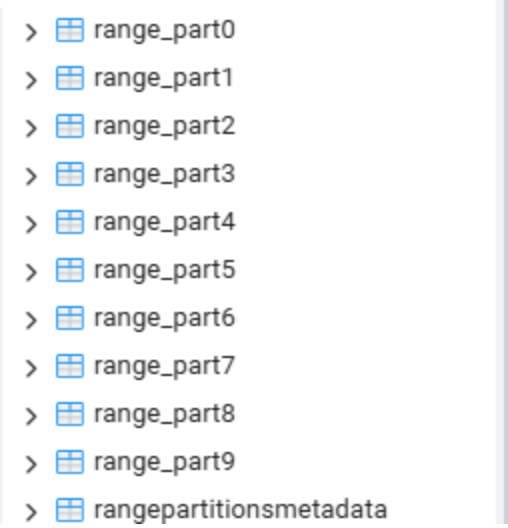
6.4 Hàm rangepartition()

```python
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    """
    Hàm tạo các partition của bảng chính dựa trên khoảng giá trị của ratings.
    Mỗi partition sẽ chứa các bản ghi có rating nằm trong một khoảng cụ thể.

    Với N partition:
    - Kích thước khoảng = 5.0/N
    - Partition i (0 đến N-1) chứa:
        - i=0: ratings trong [0, range_size]
        - i>0: ratings trong (i*range_size, (i+1)*range_size]

    Đồng thời tạo và duy trì bảng metadata để lưu thông tin về các partition.

    Args:
        ratingstablename: Tên bảng ratings
        numberofpartitions: Số lượng partition cần tạo
        openconnection: Kết nối database
    """
    con = openconnection
    cur = con.cursor()

    try:
        # Tạo bảng metadata nếu chưa tồn tại
        cur.execute("""
            CREATE TABLE IF NOT EXISTS RangePartitionsMetadata (
                partition_index INTEGER PRIMARY KEY,
                partition_table_name VARCHAR(255),
                min_rating_exclusive FLOAT,
                max_rating_inclusive FLOAT,
                total_records INTEGER DEFAULT 0
            );
        """)

        # Xóa metadata cũ
        cur.execute("DELETE FROM RangePartitionsMetadata;")

        # Tính toán kích thước khoảng và ranh giới
        range_size = 5.0 / numberofpartitions
        boundaries = [i * range_size for i in range(numberofpartitions + 1)]
```

Hình 18 Hàm rangepartition - 1

```
        # Tính toán kích thước khoảng và ranh giới
        range_size = 5.0 / numberofpartitions
        boundaries = [i * range_size for i in range(numberofpartitions + 1)]

        # Tạo và điền dữ liệu cho từng partition
        for i in range(numberofpartitions):
            table_name = f'range_part{i}'

            # Xóa bảng partition nếu đã tồn tại
            cur.execute(f"DROP TABLE IF EXISTS {table_name};")

            # Tạo bảng partition với schema giống bảng ratings
            cur.execute(f"""
                CREATE TABLE {table_name} (
                    userid INTEGER,
                    movieid INTEGER,
                    rating FLOAT,
                    PRIMARY KEY (userid, movieid)
                );
            """)

            # Tính toán ranh giới cho partition này
            if i == 0:
                # Partition đầu tiên bao gồm cả giá trị 0
                min_rating_exclusive = -0.1  # Giá trị đặc biệt cho partition đầu tiên
                max_rating_inclusive = boundaries[1]
                cur.execute(f"""
                    INSERT INTO {table_name} (userid, movieid, rating)
                    SELECT userid, movieid, rating
                    FROM {ratingstablename}
                    WHERE rating >= 0 AND rating <= {max_rating_inclusive};
                """)
            else:
                # Các partition khác loại trừ giá trị dưới
                min_rating_exclusive = boundaries[i]
                max_rating_inclusive = boundaries[i + 1]
                cur.execute(f"""
                    INSERT INTO {table_name} (userid, movieid, rating)
                    SELECT userid, movieid, rating
                    FROM {ratingstablename}
                    WHERE rating > {min_rating_exclusive} AND rating <= {max_rating_inclusive};
                """)

            # Đếm số bản ghi trong partition
            cur.execute(f"SELECT COUNT(*) FROM {table_name};")
            total_records = cur.fetchone()[0]

            # Lưu metadata của partition
            cur.execute("""
                INSERT INTO RangePartitionsMetadata
                (partition_index, partition_table_name, min_rating_exclusive, max_rating_inclusive, total_records)
                VALUES (%s, %s, %s, %s, %s);
            """, (i, table_name, min_rating_exclusive, max_rating_inclusive, total_records))

        con.commit()
        print(f"Debug - Đã tạo {numberofpartitions} range partitions thành công")

        # Thêm kiểm tra sau khi tạo partition
        verify_range_partitions(openconnection)

    except Exception as e:
        con.rollback()
        print(f"Debug - Lỗi trong rangepartition: {str(e)}")
        raise e

    finally:
        cur.close()
```

Hình 19 Hàm rangepartition - 2



Hình 20 Các bảng phân mảnh range và metadata

Data Output | Messages | Notifications

	userid [PK] integer	movieid [PK] integer	rating double precision
1	4	231	1
2	5	1	1
3	5	708	1
4	5	736	1
5	5	780	1
6	5	1391	1
7	6	3986	1
8	6	4270	1
9	7	1917	1
10	7	2478	1
11	7	5094	1
12	8	1721	1
13	8	2379	1

Total rows: 384180 | Query complete 00:00:00.176

Hình 21 Một phân mảnh rangepartition

	partition_index [PK] integer	partition_table_name character varying (255)	min_rating_exclusive double precision	max_rating_inclusive double precision	total_records integer
1	0	range_part0	-0.1	0.5	94988
2	1	range_part1	0.5	1	384180
3	2	range_part2	1	1.5	118278
4	3	range_part3	1.5	2	790306
5	4	range_part4	2	2.5	370178
6	5	range_part5	2.5	3	2356677
7	6	range_part6	3	3.5	879764
8	7	range_part7	3.5	4	2875850
9	8	range_part8	4	4.5	585022
10	9	range_part9	4.5	5	1544812

Hình 22 Bảng metadata của rangepartition

```
Tổng số bản ghi trong tất cả partition: 10000054
rangepartition function pass!
Debug - Đã chèn/cập nhật thành công vào range_part5
rangeinsert function pass!
```

Hình 23 Kết quả của thực thi chương trình

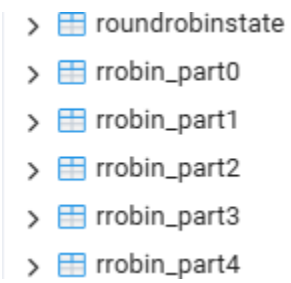
6.5 Hàm roundrobinpartition()

```python
def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
    """
    Hàm tạo các partition của bảng chính sử dụng phương pháp round robin.
    Mỗi partition sẽ chứa các dòng được phân phối theo kiểu round-robin.
    Đồng thời tạo và duy trì bảng metadata để lưu trạng thái round-robin.

    Args:
        ratingstablename: Tên bảng ratings
        numberofpartitions: Số lượng partition cần tạo
        openconnection: Kết nối database
    """
    con = openconnection
    cur = con.cursor()

    try:
        # Tạo bảng metadata cho trạng thái round-robin
        cur.execute("""
            CREATE TABLE IF NOT EXISTS RoundRobinState (
                singleton_id INTEGER PRIMARY KEY,
                next_partition_index INTEGER,
                num_partitions INTEGER,
                total_records INTEGER DEFAULT 0,
                last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            );
        """)

        # Xóa metadata cũ
        cur.execute("DELETE FROM RoundRobinState;")

        # Tạo các bảng partition
        for i in range(numberofpartitions):
            table_name = f'rrobin_part{i}'

            # Xóa bảng partition nếu đã tồn tại
            cur.execute(f"DROP TABLE IF EXISTS {table_name};")

            # Tạo bảng partition với schema giống bảng ratings
            cur.execute(f"""
                CREATE TABLE {table_name} (
                    userid INTEGER,
                    movieid INTEGER,
                    rating FLOAT,
                    PRIMARY KEY (userid, movieid)
                );
            """)

        # Phân phối dữ liệu theo kiểu round-robin sử dụng ROW_NUMBER()
        for i in range(numberofpartitions):
            cur.execute(f"""
                INSERT INTO rrobin_part{i} (userid, movieid, rating)
                SELECT userid, movieid, rating
                FROM (
                    SELECT userid, movieid, rating,
                           ROW_NUMBER() OVER () as rnum
                    FROM {ratingstablename}
                ) as temp
                WHERE MOD(rnum - 1, {numberofpartitions}) = {i};
            """)
```

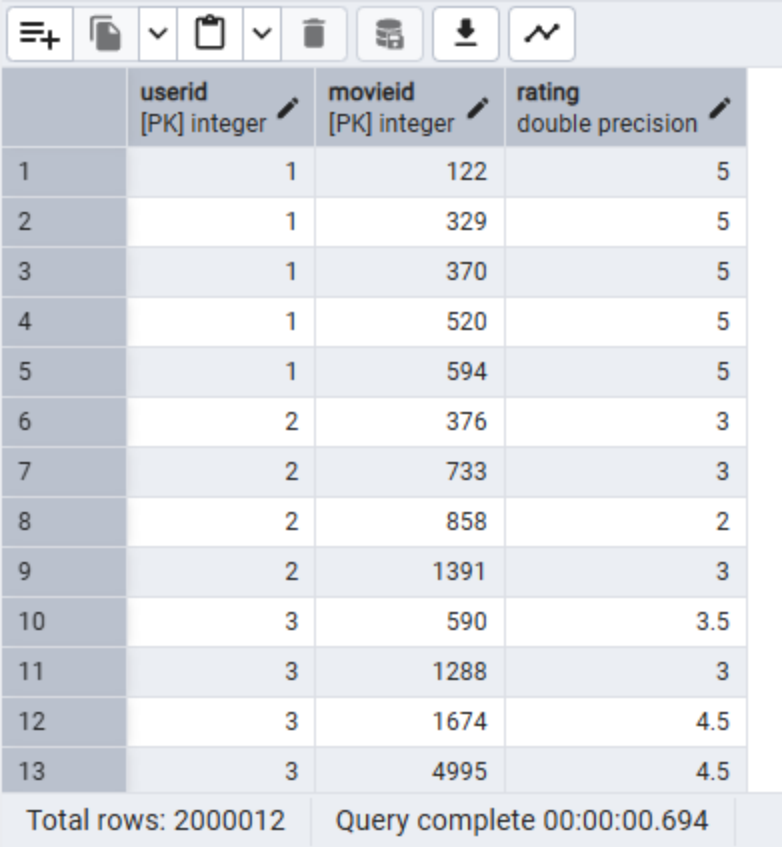
Hình 24 Hàm roundrobinpartition - 1

```
        # Phân phối dữ liệu theo kiểu round-robin sử dụng ROW_NUMBER()
        for i in range(numberofpartitions):
            cur.execute(f"""
                INSERT INTO rrobin_part{i} (userid, movieid, rating)
                SELECT userid, movieid, rating
                FROM (
                    SELECT userid, movieid, rating,
                           ROW_NUMBER() OVER () as rnum
                    FROM {ratingstablename}
                ) as temp
                WHERE MOD(rnum - 1, {numberofpartitions}) = {i};
            """)

            # Đếm số bản ghi trong partition
            cur.execute(f"SELECT COUNT(*) FROM rrobin_part{i};")
            partition_records = cur.fetchone()[0]

            # Cập nhật tổng số bản ghi
            cur.execute("""
                UPDATE RoundRobinState
                SET total_records = total_records + %s,
                    last_updated = CURRENT_TIMESTAMP
                WHERE singleton_id = 1;
            """, (partition_records,))

        # Khởi tạo trạng thái round-robin
        cur.execute("""
            INSERT INTO RoundRobinState
            (singleton_id, next_partition_index, num_partitions, total_records, last_updated)
            VALUES (1, 0, %s, 0, CURRENT_TIMESTAMP);
        """, (numberofpartitions,))

        con.commit()
        print(f"Debug - Đã tạo {numberofpartitions} round-robin partitions thành công")

        # Thêm kiểm tra sau khi tạo partition
        verify_roundrobin_partitions(openconnection)

    except Exception as e:
        con.rollback()
        print(f"Debug - Lỗi trong roundrobinpartition: {str(e)}")
        raise e

    finally:
        cur.close()
```

Hình 25 Hàm roundrobinpartition - 2



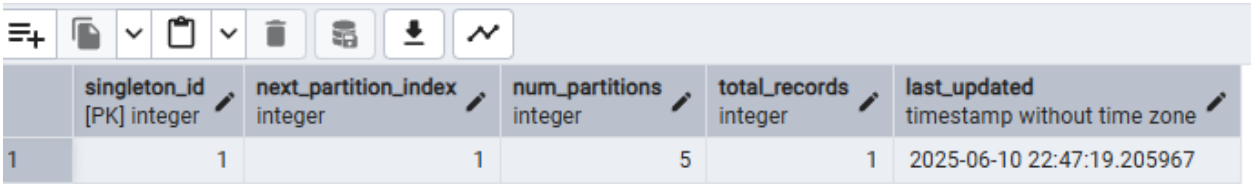
Hình 26 Các bảng phân mảnh và metadata



	userid [PK] integer	movieid [PK] integer	rating double precision
1	1	122	5
2	1	329	5
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

Total rows: 2000012 Query complete 00:00:00.694

Hình 27 Một phân mảnh của roundrobinpartition



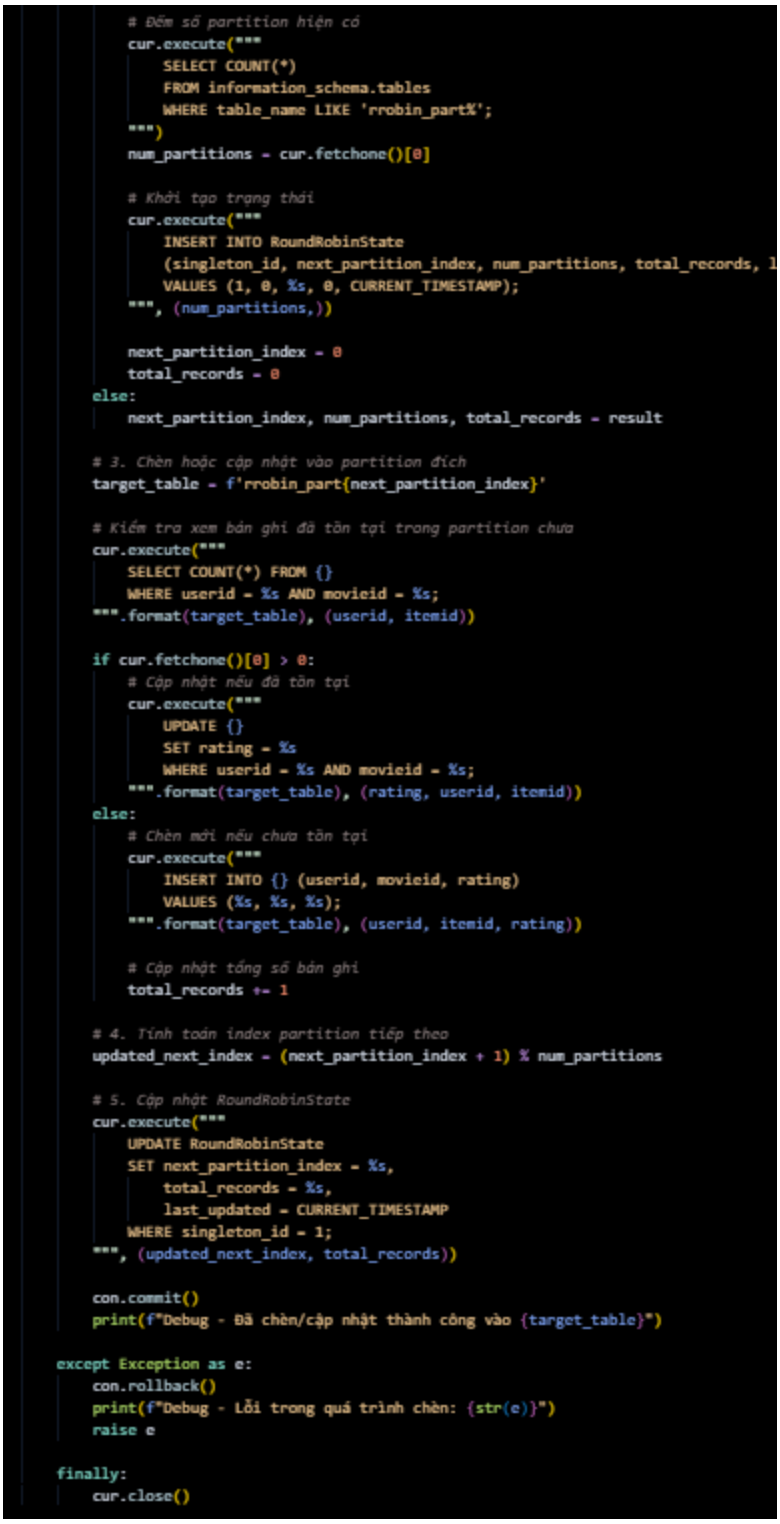
	singleton_id [PK] integer	next_partition_index integer	num_partitions integer	total_records integer	last_updated timestamp without time zone
1	1	1	5	1	2025-06-10 22:47:19.205967

Hình 28 Bảng metadata của roundrobinpartition

6.6 Hàm roundrobininsert()

```python
def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
    """
    Hàm chèn một dòng mới vào bảng chính và partition cụ thể dựa trên phương pháp round robin.
    Sử dụng bảng metadata RoundRobinState để xác định partition tiếp theo.

    Args:
        ratingstablename: Tên bảng ratings
        userid: ID người dùng
        itemid: ID phim
        rating: Giá trị rating
        openconnection: Kết nối database
    """
    con = openconnection
    cur = con.cursor()

    try:
        # Bắt đầu transaction
        con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_READ_COMMITTED)

        # 1. Kiểm tra xem bản ghi đã tồn tại chưa
        cur.execute("""
            SELECT COUNT(*) FROM {}
            WHERE userid = %s AND movieid = %s;
        """.format(ratingstablename), (userid, itemid))

        if cur.fetchone()[0] > 0:
            # Nếu bản ghi đã tồn tại, cập nhật rating
            cur.execute("""
                UPDATE {}
                SET rating = %s
                WHERE userid = %s AND movieid = %s;
            """.format(ratingstablename), (rating, userid, itemid))
        else:
            # Nếu bản ghi chưa tồn tại, chèn mới
            cur.execute("""
                INSERT INTO {} (userid, movieid, rating)
                VALUES (%s, %s, %s);
            """.format(ratingstablename), (userid, itemid, rating))

        # 2. Lấy trạng thái round-robin hiện tại
        cur.execute("""
            SELECT next_partition_index, num_partitions, total_records
            FROM RoundRobinState
            WHERE singleton_id = 1;
        """)

        result = cur.fetchone()
        if result is None:
            # Nếu RoundRobinState chưa tồn tại, tạo mới
            cur.execute("""
                CREATE TABLE IF NOT EXISTS RoundRobinState (
                    singleton_id INTEGER PRIMARY KEY,
                    next_partition_index INTEGER,
                    num_partitions INTEGER,
                    total_records INTEGER DEFAULT 0,
                    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                );
            """)
```

Hình 29 Hàm roundrobininsert - 1



```
            # Đếm số partition hiện có
            cur.execute("""
                SELECT COUNT(*)
                FROM information_schema.tables
                WHERE table_name LIKE 'rrobin_part%';
            """)
            num_partitions = cur.fetchone()[0]

            # Khởi tạo trạng thái
            cur.execute("""
                INSERT INTO RoundRobinState
                (singleton_id, next_partition_index, num_partitions, total_records, l
                VALUES (1, 0, %s, 0, CURRENT_TIMESTAMP);
            """, (num_partitions,))

            next_partition_index = 0
            total_records = 0
        else:
            next_partition_index, num_partitions, total_records = result

        # 3. Chèn hoặc cập nhật vào partition đích
        target_table = f'rrobin_part{next_partition_index}'

        # Kiểm tra xem bản ghi đã tồn tại trong partition chưa
        cur.execute("""
            SELECT COUNT(*) FROM {}
            WHERE userid = %s AND movieid = %s;
        """.format(target_table), (userid, itemid))

        if cur.fetchone()[0] > 0:
            # Cập nhật nếu đã tồn tại
            cur.execute("""
                UPDATE {}
                SET rating = %s
                WHERE userid = %s AND movieid = %s;
            """.format(target_table), (rating, userid, itemid))
        else:
            # Chèn mới nếu chưa tồn tại
            cur.execute("""
                INSERT INTO {} (userid, movieid, rating)
                VALUES (%s, %s, %s);
            """.format(target_table), (userid, itemid, rating))

            # Cập nhật tổng số bản ghi
            total_records += 1

        # 4. Tính toán index partition tiếp theo
        updated_next_index = (next_partition_index + 1) % num_partitions

        # 5. Cập nhật RoundRobinState
        cur.execute("""
            UPDATE RoundRobinState
            SET next_partition_index = %s,
                total_records = %s,
                last_updated = CURRENT_TIMESTAMP
            WHERE singleton_id = 1;
        """, (updated_next_index, total_records))

        con.commit()
        print(f"Debug - Đã chèn/cập nhật thành công vào {target_table}")

    except Exception as e:
        con.rollback()
        print(f"Debug - Lỗi trong quá trình chèn: {str(e)}")
        raise e

    finally:
        cur.close()
```

Hình 30 Hàm roundrobininsert - 2

6.7 Hàm rangeinsert()

```python
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    Sử dụng bảng metadata RangePartitionsMetadata để xác định partition phù hợp.

    Args:
        ratingstablename: Tên bảng ratings
        userid: ID người dùng
        itemid: ID phim
        rating: Giá trị rating
        openconnection: Kết nối database
    """
    con = openconnection
    cur = con.cursor()

    try:
        # Bắt đầu transaction
        con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_READ_COMMITTED)

        # 1. Kiểm tra xem bản ghi đã tồn tại chưa
        cur.execute("""
            SELECT COUNT(*) FROM {}
            WHERE userid = %s AND movieid = %s;
        """.format(ratingstablename), (userid, itemid))

        if cur.fetchone()[0] > 0:
            # Nếu bản ghi đã tồn tại, cập nhật rating
            cur.execute("""
                UPDATE {}
                SET rating = %s
                WHERE userid = %s AND movieid = %s;
            """.format(ratingstablename), (rating, userid, itemid))
        else:
            # Nếu bản ghi chưa tồn tại, chèn mới
            cur.execute("""
                INSERT INTO {} (userid, movieid, rating)
                VALUES (%s, %s, %s);
            """.format(ratingstablename), (userid, itemid, rating))

        # 2. Tìm partition phù hợp sử dụng metadata
        cur.execute("""
            SELECT partition_table_name, min_rating_exclusive, max_rating_inclusive
            FROM RangePartitionsMetadata
            WHERE %s > min_rating_exclusive
            AND %s <= max_rating_inclusive;
        """, (rating, rating))

        result = cur.fetchone()
        if result is None:
            raise Exception(f"Không tìm thấy partition phù hợp cho giá trị rating: {rating}")

        partition_table = result[0]

        # 3. Kiểm tra xem bản ghi đã tồn tại trong partition chưa
        cur.execute("""
            SELECT COUNT(*) FROM {}
            WHERE userid = %s AND movieid = %s;
        """.format(partition_table), (userid, itemid))

        if cur.fetchone()[0] > 0:
            # Cập nhật nếu đã tồn tại
            cur.execute("""
                UPDATE {}
                SET rating = %s
                WHERE userid = %s AND movieid = %s;
            """.format(partition_table), (rating, userid, itemid))
        else:
            # Chèn mới nếu chưa tồn tại
            cur.execute("""
                INSERT INTO {} (userid, movieid, rating)
                VALUES (%s, %s, %s);
            """.format(partition_table), (userid, itemid, rating))

            # Cập nhật số bản ghi trong metadata
```

Hình 31 Hàm rangeinsert – 1

```
        # 3. Kiểm tra xem bản ghi đã tồn tại trong partition chưa
        cur.execute("""
            SELECT COUNT(*) FROM {}
            WHERE userid = %s AND movieid = %s;
        """.format(partition_table), (userid, itemid))

        if cur.fetchone()[0] > 0:
            # Cập nhật nếu đã tồn tại
            cur.execute("""
                UPDATE {}
                SET rating = %s
                WHERE userid = %s AND movieid = %s;
            """.format(partition_table), (rating, userid, itemid))
        else:
            # Chèn mới nếu chưa tồn tại
            cur.execute("""
                INSERT INTO {} (userid, movieid, rating)
                VALUES (%s, %s, %s);
            """.format(partition_table), (userid, itemid, rating))

            # Cập nhật số bản ghi trong metadata
            cur.execute("""
                UPDATE RangePartitionsMetadata
                SET total_records = total_records + 1
                WHERE partition_table_name = %s;
            """, (partition_table,))

        con.commit()
        print(f"Debug - Đã chèn/cập nhật thành công vào {partition_table}")

    except Exception as e:
        con.rollback()
        print(f"Debug - Lỗi trong quá trình chèn: {str(e)}")
        raise e

    finally:
        cur.close()
```

Hình 32 Hàm rangeinsert - 2

PHẦN 5: KIỂM THỬ

5.1 Test case 1 : Load dữ liệu vào bảng

- Tải dữ liệu từ file ratings.dat vào bảng dữ liệu
  - Mục tiêu : kiểm tra số dòng dữ liệu được tải file vào bảng có đủ hay không
  - Tiêu chí : số dòng trong bảng bằng với số dòng của file dữ liệu
- Truy vấn kiểm tra : SELECT COUNT (*) FROM ratings
  - Truy vấn lấy ra tổng số dòng trong bảng sau khi tải dữ liệu
- Đánh giá

- Thời gian thực thi : 12,16s

5.2 Test case 2 : Kiểm tra phân mảnh theo khoảng

- Phân mảnh ngang dữ liệu bằng phương pháp phân mảnh theo khoảng
  - **Mục đích**:
    - Các bảng phân mảnh (range_part0, range_part1, ...) được tạo đúng số lượng.
    - Mỗi bảng phân mảnh chứa các bản ghi có rating trong khoảng đúng (theo công thức: [0, range_size] cho phân mảnh đầu tiên, và (i*range_size, (i+1)*range_size] cho các phân mảnh còn lại).
    - Tổng số bản ghi trong tất cả các phân mảnh bằng tổng số bản ghi trong bảng chính.
    - Bảng RangePartitionsMetadata chứa thông tin chính xác về các khoảng và số bản ghi.
  - Truy vấn kiểm tra
    - Kiểm tra số lượng phân mảnh
      - SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE 'range_part%';
    - Kiểm tra số bản ghi trong bảng chính và tổng số bản ghi trong phân mảnh
      - SELECT (SELECT COUNT(*) FROM ratings) AS total_main_table,

(SELECT SUM(total_records) FROM RangePartitionsMetadata) AS total_partitions;

    - Kiểm tra tính toàn vẹn của từng phân mảnh
      - SELECT partition_index,

        partition_table_name,

        min_rating_exclusive,

        max_rating_inclusive,

        total_records,

        (SELECT COUNT(*) FROM range_part0

        WHERE rating >= 0 AND rating <= max_rating_inclusive) AS actual_count_0,

        (SELECT COUNT(*) FROM range_part1 WHERE rating > min_rating_exclusive AND rating <= max_rating_inclusive) AS actual_count_1

        FROM RangePartitionsMetadata

        WHERE partition_index IN (0, 1);

  - Output phân mảnh với N = 10

5.3 Test case 3 : Kiểm tra thêm bản ghi vào phân mảnh theo khoảng

- **Mục đích**: Đảm bảo rằng:
  - Bản ghi mới được chèn vào bảng chính (ratings) và bảng phân mảnh phù hợp (range_part*) dựa trên giá trị rating.
  - Nếu bản ghi đã tồn tại, giá trị rating được cập nhật trong cả bảng chính và phân mảnh.
  - Bảng RangePartitionsMetadata được cập nhật số bản ghi (total_records) khi chèn mới.
  - Bản ghi được chèn vào đúng phân mảnh theo khoảng rating.
- Truy vấn kiểm tra
  - Kiểm tra bản ghi trong bảng chính
    - SELECT userid, movieid, rating FROM ratings WHERE userid = 9999 AND movieid = 8888;
  - Kiểm tra bản ghi trong phân mảnh phù hợp
    - SELECT partition_table_name, min_rating_exclusive, max_rating_inclusive FROM RangePartitionsMetadata WHERE 3.5 > min_rating_exclusive AND 3.5 <= max_rating_inclusive;
  - Kiểm tra bản ghi trong phân mảnh
    - SELECT userid, movieid, rating FROM range_part2 WHERE userid = 9999 AND movieid = 8888;

5.4 Test case 4 : Kiểm tra phân mảnh vòng tròn

- **Mục đích**: Đảm bảo rằng:
  - Các bảng phân mảnh (rrobin_part0, rrobin_part1, ...) được tạo đúng số lượng.
  - Các bản ghi được phân phối đều theo kiểu round-robin (số bản ghi trong mỗi phân mảnh gần bằng nhau, chênh lệch tối đa 1).
  - Tổng số bản ghi trong tất cả các phân mảnh bằng tổng số bản ghi trong bảng chính.
  - Bảng RoundRobinState chứa thông tin chính xác về số phân mảnh và trạng thái.
- Truy vấn kiểm tra
  - Kiểm tra số lượng bảng phân mảnh

- SELECT COUNT(*) FROM pg_stat_user_tables WHERE relname LIKE 'rrobin_part%';

  - Kiểm tra số lượng bản ghi trong bảng chính và tổng số bản ghi trong các phân mảnh

- SELECT (SELECT COUNT(*) FROM ratings) AS total_main_table, (SELECT total_records FROM RoundRobinState WHERE singleton_id = 1) AS total_partitions;

  - Kiểm tra sự phân phối đều của bản ghi

- SELECT table_name, (SELECT COUNT(*) FROM rrobin_part0) AS record_count FROM information_schema.tables WHERE table_name LIKE 'rrobin_part%' UNION ALL SELECT table_name, (SELECT COUNT(*) FROM rrobin_part1) AS record_count FROM information_schema.tables WHERE table_name LIKE 'rrobin_part%'

5.5 Test case 5 : Kiểm tra thêm bản ghi vào phân mảnh vòng tròn

- **Mục đích**: Đảm bảo rằng:
  - Bản ghi mới được chèn vào bảng chính (ratings) và bảng phân mảnh vòng tròn (rrobin_part*) theo trạng thái next_partition_index.
  - Nếu bản ghi đã tồn tại, giá trị rating được cập nhật trong cả bảng chính và phân mảnh.
  - Bảng RoundRobinState được cập nhật (total_records tăng, next_partition_index thay đổi).
  - Bản ghi được chèn vào đúng phân mảnh theo thứ tự round-robin.
- Truy vấn kiểm tra
    - Kiểm tra bản ghi trong bảng chính

- SELECT userid, movieid, rating FROM ratings WHERE userid = 9999 AND movieid = 8888;

    - Kiểm tra trạng thái RoudRobinState
        - SELECT next_partition_index, num_partitions, total_records FROM RoundRobinState WHERE singleton_id = 1;
    - Kiểm tra bản ghi trong phân mảnh đích
        - SELECT userid, movieid, rating FROM rrobin_part0 WHERE userid = 9999 AND movieid = 8888;
    - Kiểm tra số bản ghi và trạng thái sau khi chèn
        - SELECT total_records, next_partition_index FROM RoundRobinState WHERE singleton_id = 1;

PHẦN 6: KẾT LUẬN

6.1. Tổng kết

Bài tập lớn đã được thực hiện với mục tiêu chính là tìm hiểu và mô phỏng các phương pháp phân mảnh dữ liệu ngang (Horizontal Partitioning) trên một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở là PostgreSQL. Cụ thể, dự án đã tập trung vào việc triển khai hai chiến lược phân mảnh phổ biến: Phân mảnh theo khoảng (Range Partitioning) và Phân mảnh vòng tròn (Round-Robin Partitioning). Toàn bộ logic được xây dựng bằng ngôn ngữ Python, sử dụng thư viện psycopg2 để tương tác với cơ sở dữ liệu, và xử lý tập dữ liệu MovieLens 10M với hơn 10 triệu bản ghi đánh giá phim.

6.2. Các kết quả đã đạt được

Qua quá trình thực hiện, nhóm đã hoàn thành tất cả các yêu cầu đề ra, xây dựng thành công một bộ các hàm Python với các chức năng cụ thể:

- **Tải dữ liệu (loadratings):** Xây dựng thành công hàm tải dữ liệu ban đầu, cho phép tải hơn 10 triệu bản ghi vào cơ sở dữ liệu chỉ trong vài chục giây
- **Phân mảnh theo khoảng (rangepartition):** Triển khai thuật toán chia bảng ratings thành N phân mảnh dựa trên các khoảng giá trị đồng đều của cột rating. Một bảng siêu dữ liệu (RangePartitionsMetadata) đã được sử dụng để lưu lại "bản đồ" phân mảnh, giúp hệ thống trở nên linh hoạt và dễ truy vấn.
- **Phân mảnh vòng tròn (roundrobinpartition):** Triển khai thành công thuật toán phân phối đều các dòng dữ liệu vào N phân mảnh bằng cách sử dụng hàm cửa sổ ROW_NUMBER() trong SQL, đảm bảo các phân mảnh có số lượng bản ghi gần như bằng nhau.
- **Chèn dữ liệu (rangeinsert và roundrobininsert):** Xây dựng các hàm chèn dữ liệu mới có khả năng tự động xác định và chèn bản ghi vào đúng bảng phân mảnh tương ứng bằng cách tra cứu các bảng siêu dữ liệu.
- **Đảm bảo toàn vẹn dữ liệu:** Tất cả các hàm thực hiện ghi/thay đổi dữ liệu đều được đặt trong các khối giao dịch (try...except...finally) để đảm bảo tính toàn vẹn theo nguyên tắc "tất cả hoặc không có gì" thông qua commit và rollback.

6.3. Hướng phát triển và cải tiến trong tương lai

Mặc dù dự án đã hoàn thành các mục tiêu chính, vẫn có nhiều hướng để cải tiến và phát triển thêm:

- **So sánh với Phân mảnh gốc:** PostgreSQL (từ phiên bản 10) đã hỗ trợ tính năng phân mảnh khai báo (Declarative Partitioning). Một hướng phát triển thú vị là triển khai lại bài toán bằng tính năng gốc này và so sánh hiệu suất, cũng như sự tiện lợi so với cách mô phỏng thủ công.
- **Mở rộng các chiến lược phân mảnh:** Có thể triển khai thêm các phương pháp phân mảnh khác như Phân mảnh theo danh sách (List Partitioning) hoặc Phân mảnh theo hàm băm (Hash Partitioning).
- **Đo lường hiệu suất truy vấn:** Xây dựng các kịch bản để đo lường và so sánh tốc độ truy vấn trên bảng gốc so với khi truy vấn trên các bảng đã được phân mảnh để thấy rõ lợi ích của việc phân mảnh.