HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

Cơ sở dữ liệu phân tán

Đề bài: Phân mảnh theo khoảng và phân mảnh vòng tròn

Giảng viên hướng dẫn:	**Kim Ngọc Bách**
Nhóm:	**12**
Ngô Hồng Phúc	**B22DCCN629**
Đặng Trung Kiên	**B22DCCN426**
Nguyễn Việt Tiến	**B22DCCN726**

Hà Nội – 2025

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin cảm ơn thầy Kim Ngọc Bách - giảng viên phụ trách môn Cơ Sở Dữ Liệu Phân Tán đã chỉ dạy chúng em tận tình, hướng dẫn, giảng dạy cho chúng em những kiến thức quý báu ở trong thực tiễn.

Chúng em cũng xin gửi lời cảm ơn đến các thầy cô trong khoa công nghệ thông tin 1, Học viện Công nghệ Bưu chính Viễn thông, đã tạo điều kiện thuận lợi cho chúng em trong suốt quá trình học tập và nghiên cứu.

Đây là bài báo cáo chúng em đã làm hết sức mình nhưng sẽ không thể tránh được những sai sót, mong thầy góp ý để bản báo cáo hoàn thiện hơn ạ.

Chúng em xin cảm ơn thầy !

Mục lục

Phần 1: Mở đầu 1
Phần 2 : Cơ sở lý thuyết 1
2.1. Mục tiêu và lợi ích của phân mảnh dữ liệu : 1
2.2 Phân loại các phương pháp phân mảnh dữ liệu : 3
2.2.1. Phân mảnh Ngang (Horizontal Fragmentation) : 3
2.2.2 Phân mảnh Dọc (Vertical Fragmentation): 4
2.2.3. Phân mảnh Hỗn hợp/Lai (Mixed/Hybrid Fragmentation) : 5
2.3. Giới thiệu về PostgreSQL : 6
Phần 3 : Cài đặt và triển khai báo cáo 6
3.1 Cài đặt môi trường : 6
3.2 Chuẩn bị dữ liệu : 7
3.3 Xây dựng các hàm xử lý chính. 7
3.3.1 Hàm LoadRatings(ratingstablename, ratingsfilepath, openconnection) 7
3.3.2 Hàm rangepartition(ratingstablename, numberofpartitions, openconnection) 8
3.3.3 Hàm roundrobinpartition(ratingstablename, numberofpartitions, openconnection) 9
3.3.4 Hàm rangeinsert(ratingstablename, userid, itemid, rating, openconnection) 10
3.3.5 Hàm roundrobininsert(ratingstablename, userid, itemid, rating, openconnection) 11
3.4 Các hàm mở rộng 12
3.4.1 Hàm getopenconnection 12
3.4.2 Hàm create_db 13
3.4.3 Hàm deleteallpublictables 14
Phần 4: Kết luận 14

Phần 1: Mở đầu

1. Giới thiệu :

Trong bối cảnh dữ liệu ngày càng trở nên lớn và phức tạp, việc quản lý, lưu trữ và truy vấn dữ liệu hiệu quả là một thách thức quan trọng đối với các hệ quản trị cơ sở dữ liệu. Một trong những kỹ thuật quan trọng được sử dụng để cải thiện hiệu năng và khả năng mở rộng của hệ thống là *phân mảnh dữ liệu* (data fragmentation).

Phân mảnh dữ liệu là quá trình chia nhỏ một bảng dữ liệu lớn thành nhiều phần nhỏ hơn, gọi là các mảnh (fragments hoặc partitions), nhằm tối ưu hóa việc lưu trữ, tăng tốc độ truy xuất, và hỗ trợ khả năng xử lý song song. Các kỹ thuật phân mảnh phổ biến bao gồm *phân mảnh ngang* (horizontal fragmentation), *phân mảnh dọc* (vertical fragmentation), và *phân mảnh hỗn hợp* (hybrid fragmentation).

Trong đó, *phân mảnh ngang* là phương pháp chia một bảng thành nhiều phần theo hàng (record), mỗi phần chứa một tập con các bản ghi thỏa mãn điều kiện nhất định. Kỹ thuật này đặc biệt hiệu quả khi các bản ghi có thể được phân chia theo vùng địa lý, thời gian, hoặc theo thuộc tính nào đó của dữ liệu.

Trong PostgreSQL, việc *phân mảnh dữ liệu* được hỗ trợ thông qua table partitioning, cho phép triển khai phân mảnh ngang một cách trực tiếp. Bên cạnh đó, để tối ưu hiệu năng truy vấn, kỹ thuật sử dụng *vòng tròn băm* (consistent hashing) cũng có thể được kết hợp nhằm phân phối dữ liệu đồng đều giữa các phân vùng hoặc nút trong hệ thống phân tán.

Báo cáo này sẽ trình bày chi tiết về quá trình thiết kế và triển khai *phân mảnh ngang* kết hợp *kỹ thuật vòng tròn băm* trong *PostgreSQL*, nhằm đạt được hiệu quả cao trong quản lý và truy vấn dữ liệu.

Phần 2 : Cơ sở lý thuyết

2.1. Mục tiêu và lợi ích của phân mảnh dữ liệu :

- **Mục tiêu của Phân mảnh Dữ liệu:**

- **Cải thiện Hiệu năng Truy vấn** (Improve Query Performance): Đây là mục tiêu hàng đầu. Bằng cách chia một bảng lớn thành các phân mảnh nhỏ hơn, một truy vấn chỉ cần truy cập và xử lý trên một hoặc một vài phân mảnh liên quan, thay vì quét toàn bộ bảng. Điều này làm giảm đáng kể lượng dữ liệu cần đọc từ đĩa và truyền qua mạng, giúp truy vấn trả về kết quả nhanh hơn, đặc biệt là các truy vấn chỉ cần một tập con dữ liệu dựa trên tiêu chí phân mảnh.

- **Tăng Tính Khả dụng và Độ Tin cậy** (Increase Availability & Reliability): Mặc dù phân mảnh đơn thuần không đủ để đảm bảo chịu lỗi hoàn toàn (cần kết hợp với nhân bản), việc phân tán dữ liệu giúp hệ thống resilient hơn. Nếu một node gặp sự cố, chỉ phần

dữ liệu trên node đó bị ảnh hưởng, các phân mảnh khác trên các node còn lại vẫn có thể truy cập được.

- **Hỗ trợ Khả năng Mở rộng** (Support Scalability): Khi lượng dữ liệu hoặc số lượng người dùng tăng lên, hệ thống có thể mở rộng bằng cách thêm các node mới và phân mảnh dữ liệu mới (hoặc tái phân mảnh dữ liệu hiện có) vào các node đó. Điều này giúp phân tán tải và tránh tình trạng quá tải trên một node duy nhất.

- **Tăng Tính Cục bộ Dữ liệu** (Enhance Data Locality): Phân mảnh cho phép lưu trữ dữ liệu gần với nơi mà nó được sử dụng thường xuyên nhất. Ví dụ, dữ liệu khách hàng ở Hà Nội có thể được lưu trên máy chủ ở Hà Nội, giúp giảm độ trễ khi người dùng ở Hà Nội truy cập dữ liệu của họ.

- **Giảm Lưu lượng Mạng** (Reduce Network Traffic): Khi dữ liệu được lưu trữ cục bộ gần với nơi sử dụng, nhiều truy vấn có thể được xử lý hoàn toàn trên node cục bộ mà không cần truyền dữ liệu qua mạng đến các node khác, từ đó giảm tải cho mạng.

- **Lợi ích của Phân mảnh Dữ liệu:**

Dựa trên các mục tiêu trên, phân mảnh mang lại những lợi ích cụ thể sau:

- **Thời gian Phản hồi Truy vấn Nhanh hơn:** Kết quả trực tiếp từ mục tiêu cải thiện hiệu năng. Các truy vấn nhắm đúng phân mảnh sẽ chạy rất nhanh.

- **Khả năng Xử lý Song song Cao hơn:** Các truy vấn hoặc giao tác cần truy cập dữ liệu từ nhiều phân mảnh khác nhau trên các node khác nhau có thể được thực thi đồng thời trên các node đó, tăng thông lượng (throughput) của hệ thống.

- **Giảm Tải Hệ Thống:** Phân tán dữ liệu và tải xử lý ra nhiều node giúp không có node nào trở thành điểm nghẽn cổ chai (bottleneck) duy nhất cho toàn bộ hệ thống.

- **Quản lý Dễ dàng hơn với Tập Con Dữ liệu Nhỏ:** Việc quản lý, sao lưu, phục hồi hoặc tối ưu hóa chỉ trên một phân mảnh dữ liệu nhỏ hơn có thể đơn giản và nhanh hơn so với việc thực hiện trên toàn bộ bảng rất lớn.

- **Tính Bảo mật Cải thiện**: Việc phân mảnh có thể hỗ trợ bảo mật bằng cách cho phép áp dụng các chính sách truy cập khác nhau cho các phân mảnh khác nhau, giới hạn người dùng chỉ được truy cập vào phần dữ liệu họ cần.

Tóm lại, phân mảnh dữ liệu là một kỹ thuật thiết yếu trong CSDL phân tán nhằm "chia để trị" - chia nhỏ dữ liệu để dễ quản lý, tăng hiệu suất truy cập, cải thiện tính sẵn sàng và cho phép hệ thống mở rộng hiệu quả hơn. Tuy nhiên, việc phân mảnh cũng cần được thiết kế cẩn thận để tránh các truy vấn phức tạp cần kết hợp dữ liệu từ nhiều phân mảnh một cách thường xuyên, điều này có thể làm giảm hiệu năng.

2.2 Phân loại các phương pháp phân mảnh dữ liệu :

Có ba phương pháp phân mảnh dữ liệu chính:

- Phân mảnh Ngang (Horizontal Fragmentation)
- Phân mảnh Dọc (Vertical Fragmentation)
- Phân mảnh Hỗn hợp/Lai (Mixed/Hybrid Fragmentation)

Chúng ta sẽ đi vào chi tiết từng loại:

2.2.1. Phân mảnh Ngang (Horizontal Fragmentation) :

- Khái niệm: Phân mảnh ngang chia một bảng ban đầu thành các tập con các hàng (rows). Mỗi phân mảnh ngang chứa một tập hợp các hàng từ bảng gốc mà thỏa mãn một điều kiện (một vị từ - predicate) cụ thể. Các cột của bảng gốc được giữ nguyên trong mỗi phân mảnh.

- Đặc điểm:

- Mỗi phân mảnh ngang là một tập con của bảng gốc.
- Toàn bộ các phân mảnh ngang phải bao phủ toàn bộ các hàng của bảng gốc (tính đầy đủ - completeness).
- Các phân mảnh ngang thường là rời rạc (disjoint), tức là một hàng không thể xuất hiện trong nhiều hơn một phân mảnh (để tránh dư thừa và phức tạp khi cập nhật). Tuy nhiên, trong một số trường hợp nhân bản phân mảnh, tính rời rạc này có thể bị phá vỡ.

- **Phân loại Phân mảnh Ngang:**

- Phân mảnh Ngang Cơ sở (Primary Horizontal Fragmentation - PHF): Việc phân mảnh được thực hiện dựa trên một vị từ (điều kiện) áp dụng trực tiếp lên một hoặc nhiều thuộc tính của bảng đang được phân mảnh.

Ví dụ: Bảng USERS được phân mảnh dựa trên user_type.

Fragment 1: SELECT * FROM USERS WHERE user_type = 'SINH_VIEN'

Fragment 2: SELECT * FROM USERS WHERE user_type = 'GIANG_VIEN'

Fragment 3: SELECT * FROM USERS WHERE user_type = 'NHAN_VIEN'

- **Phân mảnh Ngang Dẫn xuất** (Derived Horizontal Fragmentation - DHF): Việc phân mảnh một bảng được thực hiện dựa trên một vị từ áp dụng lên một bảng khác, thường là bảng "cha" trong mối quan hệ 1-nhiều hoặc nhiều-nhiều. Sau khi bảng "cha" được phân mảnh, bảng "con" sẽ được phân mảnh tương ứng dựa trên khóa ngoại liên kết với bảng "cha". Thường sử dụng phép semi-join (hoặc join và chiếu lên khóa chính) để xác định các hàng tương ứng.

Ví dụ: Giả sử bảng *CUSTOMERS* được phân mảnh ngang cơ sở dựa trên vùng địa lý (Miền Bắc, Miền Nam). Bảng *ORDERS* có khóa ngoại liên kết với *CUSTOMERS*. Ta có thể phân mảnh ORDERS thành *ORDERS_MIEN_BAC* và *ORDERS_MIEN_NAM* sao cho *ORDERS_MIEN_BAC* chỉ chứa các đơn hàng của khách hàng thuộc Miền

Bắc, và *ORDERS_MIEN_NAM* chứa các đơn hàng của khách hàng thuộc Miền Nam. Phân mảnh bảng *ORDERS* lúc này là DHF, dẫn xuất từ PHF của bảng *CUSTOMERS*.

- **Ưu điểm** của Phân mảnh Ngang:

  - Cải thiện hiệu năng cho các truy vấn chỉ cần một tập con các hàng.
  - Tăng tính cục bộ dữ liệu.
  - Hỗ trợ xử lý song song.

- **Nhược điểm** của Phân mảnh Ngang:

  - Các truy vấn cần toàn bộ dữ liệu (ví dụ: tổng hợp báo cáo toàn hệ thống) đòi hỏi thao tác UNION (kết hợp lại các fragment), có thể tốn kém.
  - Thiết kế vị từ phân mảnh không phù hợp có thể dẫn đến việc truy vấn thường xuyên phải truy cập nhiều fragment.

2.2.2 Phân mảnh Dọc (Vertical Fragmentation):

- Khái niệm: Phân mảnh dọc chia một bảng ban đầu thành các tập con các cột (columns). Mỗi phân mảnh dọc chứa một tập hợp các cột từ bảng gốc, và quan trọng là luôn bao gồm khóa chính của bảng gốc để có thể tái tạo lại bảng ban đầu. Mỗi hàng trong các phân mảnh dọc tương ứng với một hàng trong bảng gốc.

- **Đặc điểm:**

  - Mỗi phân mảnh dọc là một tập con các cột của bảng gốc, luôn bao gồm khóa chính.
  - Toàn bộ các phân mảnh dọc (kết hợp với khóa chính) phải bao phủ toàn bộ các cột của bảng gốc (tính đầy đủ).

- **Cách thực hiện:** Việc phân mảnh dọc thường dựa trên phân tích sự cùng xuất hiện (co-access) của các cột trong các truy vấn phổ biến. Các cột thường được truy cập cùng nhau sẽ được nhóm vào cùng một phân mảnh dọc.

Ví dụ: Bảng *BOOKS* có các cột *ISBN* (PK), *Title*, *Author*, *Publisher*, *Category*, *Description*, *PublicationYear*, *CoverImage*.

Fragment 1 (BOOKS_BASIC): *ISBN*, *Title*, *Author*, *PublicationYear*, *Publisher*, *Category*. (Dùng cho các truy vấn tìm kiếm và hiển thị danh sách cơ bản)

Fragment 2 (BOOKS_DETAILS): *ISBN*, *Description*, *CoverImage*. (Dùng cho các truy vấn xem chi tiết sách)

- **Ưu điểm** của Phân mảnh Dọc:

  - Cải thiện hiệu năng cho các truy vấn chỉ cần một tập con nhỏ các cột, vì giảm lượng dữ liệu cần đọc từ đĩa (I/O).
  - Giảm lượng dữ liệu truyền qua mạng cho các truy vấn chỉ cần ít cột.

- Có thể cải thiện khả năng sử dụng cache của bộ nhớ đệm.

- **Nhược điểm** của Phân mảnh Dọc:

- Các truy vấn cần dữ liệu từ nhiều phân mảnh dọc (nhiều nhóm cột) đòi hỏi thao tác JOIN (kết hợp lại các fragment dựa trên khóa chính), có thể tốn kém.
- Việc tái tạo bảng gốc luôn cần thao tác JOIN.

2.2.3. Phân mảnh Hỗn hợp/Lai (Mixed/Hybrid Fragmentation) :

- **Khái niệm:** Phân mảnh hỗn hợp kết hợp cả phân mảnh ngang và phân mảnh dọc. Nó thường được thực hiện bằng cách áp dụng một phương pháp phân mảnh (ngang hoặc dọc) trước, sau đó áp dụng phương pháp còn lại lên các fragment kết quả.

- **Cách thực hiện:**

Bước 1: Phân mảnh bảng gốc theo chiều ngang hoặc dọc.

Bước 2: Phân mảnh tiếp các fragment kết quả từ Bước 1 theo chiều còn lại.

Ví dụ: Bảng *EMPLOYEES* được phân mảnh ngang cơ sở theo phòng ban (*DEPARTMENT*). Sau đó, mỗi phân mảnh theo phòng ban (ví dụ: *EMPLOYEES*_SALES) được phân mảnh dọc thành *EMPLOYEES_SALES_BASIC* (ID, Name, Title) và *EMPLOYEES_SALES_SALARY* (ID, Salary, BonusRate).

- **Ưu điểm** của Phân mảnh Hỗn hợp:
  - Cho phép thiết kế phân mảnh rất linh hoạt, có thể tối ưu hóa cho nhiều loại truy vấn khác nhau và đáp ứng yêu cầu phân tán phức tạp.
- **Nhược điểm** của Phân mảnh Hỗn hợp:
  - Việc quản lý và tái tạo dữ liệu trở nên rất phức tạp do phải thực hiện nhiều bước kết hợp (UNION và JOIN).
  - Thiết kế và tối ưu hóa khó khăn hơn nhiều so với hai phương pháp đơn lẻ.

=> Tóm lại:

- *Phân mảnh Ngang:* chia theo hàng dựa trên điều kiện logic (vị từ). Thích hợp khi các truy vấn thường nhắm đến các tập con hàng dựa trên giá trị cột.

- *Phân mảnh Dọc:* chia theo cột dựa trên sự cùng xuất hiện của các cột và luôn giữ khóa chính. Thích hợp khi các truy vấn thường chỉ cần một tập con nhỏ các cột.

- *Phân mảnh Hỗn hợp*: kết hợp cả hai, cho phép thiết kế phức tạp hơn nhưng cũng phức tạp hơn trong quản lý.

Việc lựa chọn phương pháp phân mảnh nào (và cách thức cụ thể) phụ thuộc vào đặc điểm dữ liệu, tần suất và loại các truy vấn phổ biến, yêu cầu về hiệu năng, tính sẵn

sàng và khả năng mở rộng của hệ thống. Đây là một khâu quan trọng trong thiết kế CSDL phân tán.

2.3. Giới thiệu về PostgreSQL :

Trong bài tập này, chúng em lựa chọn sử dụng PostgreSQL làm hệ quản trị cơ sở dữ liệu (DBMS) chính được triển khai trên các node phân tán. PostgreSQL là một hệ quản trị CSDL quan hệ đối tượng (ORDBMS) mã nguồn mở, nổi tiếng với sự mạnh mẽ, độ tin cậy cao và hệ thống tính năng phong phú.

Việc sử dụng PostgreSQL mang lại nhiều lợi ích cho hệ thống phân tán của đồ án, bao gồm:

- *Sự ổn định và đáng tin cậy:* Đảm bảo tính toàn vẹn dữ liệu trong môi trường nhiều node.

- *Hiệu năng tốt:* Khả năng xử lý các truy vấn phức tạp hiệu quả.

- *Hỗ trợ các tính năng nâng cao:* Bao gồm khả năng mở rộng, các công cụ hỗ trợ nhân bản (replication) và phân vùng (partitioning), rất quan trọng cho việc triển khai các kỹ thuật CSDL phân tán như phân mảnh và nhân bản dữ liệu giữa các node.

Nhờ những đặc điểm này, PostgreSQL trở thành nền tảng vững chắc để xây dựng và thử nghiệm hệ thống quản lý thư viện phân tán của chúng em.

Phần 3 : Cài đặt và triển khai báo cáo

3.1 Cài đặt môi trường :

- Hệ cơ sở dữ liệu : PostgreSQL 14.18

- Hệ điều hành : Ubuntu 22.04.5

- Ngôn ngữ lập trình : python3 (3.10.12)

Để thực hiện, triển khai các phương pháp phân mảnh trong báo cáo, chúng em đã cài đặt những môi trường như trên (hoặc có thể version mới hơn). Môi trường này đã bao gồm các công cụ, phần mềm đầy đủ nhất để thực hiện, xử lí, triển khai các dữ liệu cần thiết.

3.2 Chuẩn bị dữ liệu :

Nhóm đã sử dụng tập dữ liệu *Movie Lens 10M Dataset* là tập dữ liệu đánh giá phim được thu thập từ trang web *MovieLens*(https://movielens.org), để triển khai các phương pháp phân mảnh trong báo cáo. Dữ liệu thô nằm trong tệp *ratings.dat.*

Tải và giải nén file từ đường trang MovieLens:

(http://files.grouplens.org/datasets/movielens/ml-10m.zip).

Ta thu được tệp dữ liệu ratings.dat

Đây là bộ dữ liệu 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 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

3.3 Xây dựng các hàm xử lý chính.

3.3.1 Hàm LoadRatings(ratingstablename, ratingsfilepath, openconnection)

- Mục đích:

+ Tạo bảng để chứa dữ liệu từ file ratings.dat.

+ Load toàn bộ dữ liệu từ file vào bảng đó.

```python
def loadratings(ratingstablename, ratingsfilepath, openconnection):
    """
    Tạo bảng ratingstablename và load dữ liệu từ file format userID::movieID::rating::timestamp
    """
    cur = openconnection.cursor()
    try:
        cur.execute(f"DROP TABLE IF EXISTS {ratingstablename}")
        cur.execute(
            f"CREATE TABLE {ratingstablename} ("
            "userid INT NOT NULL, movieid INT, rating REAL, timestamp BIGINT)"
        )
        # copy in batches
        with open(ratingsfilepath) as f:
            for batch in iter(lambda: tuple(islice(f, 5000)), ()):
                buf = StringIO()
                buf.write(''.join(line.replace('::', ',') for line in batch))
                buf.seek(0)
                cur.copy_from(buf, ratingstablename,
                              sep=',',
                              columns=('userid','movieid','rating','timestamp'))
        # nếu không cần timestamp thì drop
        cur.execute(f"ALTER TABLE {ratingstablename} DROP COLUMN timestamp")
        openconnection.commit()
    except:
        openconnection.rollback()
        raise
    finally:
        cur.close()
```

- Cách hoạt động:

- Xóa bảng nếu đã tồn tại (tránh lỗi tạo trùng).

- Tạo bảng mới với cột: userid, movieid, rating, timestamp.
- Đọc file ratings.dat theo từng batch 5000 dòng để tránh đầy RAM.
- Thay thế dấu phân cách :: bằng , để psycopg2.copy_from() sử dụng được.
- Load dữ liệu cực nhanh bằng copy_from() thay vì nhiều câu INSERT.
- Xóa cột timestamp vì chỉ sử dụng 3 cột đầu.

3.3.2 Hàm rangepartition(ratingstablename, numberofpartitions, openconnection)

- Mục đích:

+ Chia bảng ratingstablename thành các bảng con theo khoảng giá trị rating.

```python
def rangepartition(ratingstablename, numberofpartitions, openconnection):
    """
    Partition theo range: tạo range_part0..range_part{n-1}
    """
    cur = openconnection.cursor()
    try:
        # xóa các partition cũ
        for i in range(numberofpartitions):
            cur.execute(f"DROP TABLE IF EXISTS range_part{i}")
        # tạo và insert lại theo range
        interval = 5.0 / numberofpartitions
        for i in range(numberofpartitions):
            low = i * interval
            high = (i + 1) * interval if i < numberofpartitions - 1 else 5.0
            if i == 0:
                cond = f"rating >= {low} AND rating <= {high}"
            else:
                cond = f"rating > {low} AND rating <= {high}"
            cur.execute(
                f"CREATE TABLE range_part{i} AS "
                f"SELECT userid, movieid, rating FROM {ratingstablename} WHERE {cond}"
            )
        openconnection.commit()
    except:
        openconnection.rollback()
        raise
    finally:
        cur.close()
```

- Cách hoạt động:

- Xóa các bảng con cũ range_partX nếu có.
- Tính khoảng phân mảnh**:**

  VD: với 5 partition → khoảng mỗi phần = 5.0 / 5 = 1.0

  - range_part0: [0.0, 1.0]
  - range_part1: (1.0, 2.0]
  - …
- Tạo bảng con tương ứng:
  - range_part0 chứa rating >= 0.0 AND rating <= 1.0

    - range_part1 chứa rating > 1.0 AND rating <= 2.0, …
- Dữ liệu từ bảng chính được chép sang đúng partition theo điều kiện rating.

3.3.3 Hàm roundrobinpartition(ratingstablename, numberofpartitions, openconnection)

- Mục đích:

+ Chia bảng ratingstablename thành các bảng con đều nhau theo cách Round-Robin.

```
102 def roundrobinpartition(ratingstablename, numberofpartitions, openconnection):
103     """
104     cur = openconnection.cursor()
105     try:
106         # reset counter
107         if os.path.exists('.rr_counter'):
108             os.remove('.rr_counter')
109         # xóa cũ
110         for i in range(numberofpartitions):
111             cur.execute(f"DROP TABLE IF EXISTS rrobin_part{i}")
112         openconnection.commit()
113 
114         # tạo bảng tạm để gán idx
115         cur.execute(
116             f"SELECT userid, movieid, rating, "
117             f"ROW_NUMBER() OVER () - 1 AS idx "
118             f"INTO temp_rr FROM {ratingstablename}"
119         )
120         # sinh partition
121         for i in range(numberofpartitions):
122             cur.execute(
123                 f"CREATE TABLE rrobin_part{i} AS "
124                 f"SELECT userid, movieid, rating FROM temp_rr WHERE idx % {numberofpartitions} = {i}"
125             )
126         cur.execute("DROP TABLE temp_rr")
127         openconnection.commit()
128     except:
129         openconnection.rollback()
130         raise
131     finally:
132         cur.close()
```

- Cách hoạt động:

- Xóa các bảng con cũ rrobin_partX.
- Reset file .rr_counter để theo dõi lượt chèn mới.
- Tạo bảng tạm temp_rr có cột idx = ROW_NUMBER() - 1, thể hiện thứ tự dòng.
- Chia dữ liệu vào các bảng:
    - Dòng có idx % n == 0 → rrobin_part0
    - Dòng có idx % n == 1 → rrobin_part1
    - …
- Xóa bảng tạm sau khi hoàn tất.

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

- Mục đích:

- Chèn **1 dòng dữ liệu mới** vào **đúng bảng** range_partX theo giá trị rating.

```python
def rangeinsert(ratingstablename, userid, itemid, rating, openconnection):
    """
    cur = openconnection.cursor()
    try:
        # tìm các partition hiện có
        cur.execute(
            "SELECT table_name FROM information_schema.tables "
            "WHERE table_schema='public' AND table_name LIKE 'range_part%%'"
        )
        parts = sorted(r[0] for r in cur.fetchall())
        n = len(parts)
        interval = 5.0 / n

        # xác định bảng đích dựa vào điều kiện
        for i, tbl in enumerate(parts):
            low = i * interval
            high = (i + 1) * interval if i < n - 1 else 5.0
            if (i == 0 and low <= rating <= high) or (i > 0 and low < rating <= high):
                cur.execute(
                    f"INSERT INTO {tbl} (userid, movieid, rating) VALUES (%s, %s, %s)",
                    (userid, itemid, rating)
                )
                break

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

- Cách hoạt động:

- Lấy danh sách các bảng range_partX từ DB.
- Tính lại khoảng chia rating cho từng partition.
- Xác định khoảng nào chứa rating truyền vào:

VD: rating = 3.2, bảng range_part3 có khoảng (3.0, 4.0] → phù hợp.

- Chèn bản ghi vào đúng bảng đó.

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

- Mục đích:

- Chèn 1 dòng dữ liệu mới vào đúng bảng rrobin_partX theo thứ tự lượt chèn.

```
165 def roundrobininsert(ratingstablename, userid, itemid, rating, openconnection):
169         cur.execute(
170             "SELECT table_name FROM information_schema.tables "
171             "WHERE table_schema='public' AND table_name LIKE 'rrobin_part%%' "
172             "ORDER BY table_name"
173         )
174         parts = [r[0] for r in cur.fetchall()]
175         n = len(parts)
176 
177         # đọc/ghi counter
178         counter_file = '.rr_counter'
179         try:
180             with open(counter_file, 'r+') as f:
181                 cnt = int(f.read() or '0')
182                 idx = cnt % n
183                 f.seek(0); f.write(str(cnt + 1)); f.truncate()
184         except FileNotFoundError:
185             idx = 0
186             with open(counter_file, 'w') as f:
187                 f.write('1')
188         target = parts[idx]
189         cur.execute(
190             f"INSERT INTO {target} (userid, movieid, rating) VALUES (%s, %s, %s)",
191             (userid, itemid, rating)
192         )
193         openconnection.commit()
194     except:
195         openconnection.rollback()
196         raise
197     finally:
198         cur.close()
```

- Cách hoạt động:

- Lấy danh sách các bảng rrobin_partX.
- Kiểm tra (hoặc tạo mới) file .rr_counter để lấy lượt chèn hiện tại.
- Dùng counter % n để xác định bảng sẽ chèn vào.
- Tăng counter và cập nhật vào file.
- Chèn bản ghi vào bảng tương ứng.

3.4 Các hàm mở rộng

3.4.1 Hàm getopenconnection

- Mục đích:

- Hàm này dùng để tạo một kết nối đến cơ sở dữ liệu PostgreSQL.
- Kết nối này sẽ được sử dụng bởi các hàm khác để thực hiện truy vấn như tạo bảng, chèn dữ liệu, đọc dữ liệu,…

```python
def getopenconnection(user='postgres', password='1234',
                      dbname='dds_assgn1', host='localhost', port='5432'):
    """
    Trả về connection đến database dds_assgn1.
    """
    conn = psycopg2.connect(dbname=dbname,
                            user=user,
                            password=password,
                            host=host,
                            port=port)
    conn.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    return conn
```

- Cách hoạt động:

- Sử dụng thư viện psycopg2 để kết nối đến một database PostgreSQL**.**
- Các tham số gồm:
  - dbname: tên cơ sở dữ liệu (dds_assgn1 mặc định)
  - user: tên người dùng PostgreSQL
  - password: mật khẩu đăng nhập
  - host: địa chỉ máy chủ (thường là localhost)
  - port: cổng kết nối (mặc định PostgreSQL là 5432)
- Tắt chế độ transaction tự động → chuyển sang autocommit.
- Mọi lệnh SQL sẽ thực thi ngay lập tức mà không cần gọi conn.commit().
- Trả về đối tượng kết nối để dùng trong các hàm như loadratings, rangepartition,…

3.4.2 Hàm create_db

- Mục đích:

- Tạo mới database (ở đây là dds_assgn1) nếu nó chưa tồn tại trong PostgreSQL.

```python
def create_db(dbname='dds_assgn1'):
    """
    Tạo database nếu chưa tồn tại.
    """
    con = psycopg2.connect(dbname='postgres', user='postgres',
                           password='1234', host='localhost')
    con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)
    cur = con.cursor()
    cur.execute(
        "SELECT COUNT(*) FROM pg_catalog.pg_database WHERE datname=%s",
        (dbname,)
    )
    if cur.fetchone()[0] == 0:
        cur.execute(f"CREATE DATABASE {dbname}")
    cur.close()
    con.close()
```

- Cách hoạt động:

- Kết nối tới cơ sở dữ liệu mặc định postgres:
- Đặt chế độ tự động thực hiện lệnh**:**

con.set_isolation_level(psycopg2.extensions.ISOLATION_LEVEL_AUTOCOMMIT)

- Tạo đối tượng cursor để truy vấn SQL:
- Kiểm tra database đã tồn tại chưa:
  Truy vấn vào hệ thống PostgreSQL (pg_catalog.pg_database) để xem có database nào trùng tên không.
  Nếu kết quả là 0, nghĩa là chưa tồn tại.
- Tạo database nếu chưa có:

3.4.3 Hàm deleteallpublictables

- Mục đích:

- Xóa toàn bộ bảng trong schema public của cơ sở dữ liệu PostgreSQL (schema mặc định).
- Dùng để dọn sạch database, thường khi reset hoặc kiểm thử lại toàn bộ.

```
200 def deleteallpublictables(openconnection):
201     """
202     Xóa tất cả bảng trong public schema.
203     """
204     cur = openconnection.cursor()
205     cur.execute(
206         "SELECT table_name FROM information_schema.tables "
207         "WHERE table_schema='public'"
208     )
209     for (tbl,) in cur.fetchall():
210         cur.execute(f"DROP TABLE IF EXISTS {tbl} CASCADE")
211     openconnection.commit()
212     cur.close()
213
```

- Cách hoạt động:

- Tạo cursor để thực thi lệnh SQL**:**
- Lấy danh sách tất cả các bảng trong schema public**:**
- Xóa từng bảng bằng vòng lặp**:**
- Commit để xác nhận xóa**:**
- Đóng cursor:

Phần 4: Kết luận

Báo cáo này đã trình bày một cách tổng quan về hệ thống cơ sở dữ liệu phân tán, tập trung vào kỹ thuật phân mảnh dữ liệu như một giải pháp then chốt để quản lý và tối ưu hóa hiệu năng trên các tập dữ liệu lớn. Chúng em đã đi sâu vào mục tiêu và

lợi ích của việc phân mảnh, cũng như phân loại các phương pháp phân mảnh phổ biến, đặc biệt là phân mảnh ngang.

Với đề tài cụ thể là áp dụng *phân mảnh ngang* theo khoảng và kết hợp kỹ thuật *vòng tròn băm* (consistent hashing) trên nền tảng PostgreSQL, dự án nhằm minh chứng tính hiệu quả của việc phân tán dữ liệu trong môi trường thực tế. Chúng em đã giới thiệu môi trường cài đặt cần thiết (PostgreSQL, Ubuntu, Python) và phương pháp chuẩn bị dữ liệu từ tập *MovieLens 10M* để tiến hành thử nghiệm.

Mặc dù phần triển khai chi tiết các hàm xử lý và cấu hình phân mảnh/vòng tròn băm chưa được trình bày đầy đủ trong báo cáo này, mục tiêu cuối cùng là xây dựng một hệ thống có khả năng phân phối dữ liệu đánh giá phim một cách đồng đều giữa các node CSDL. Qua đó, chúng em kỳ vọng đạt được những cải thiện đáng kể về hiệu năng khi truy vấn dữ liệu, giảm tải cho từng node, tăng tính sẵn sàng và hỗ trợ khả năng mở rộng hệ thống khi cần thiết.

Báo cáo này không chỉ giúp chúng em củng cố kiến thức lý thuyết về *cơ sở dữ liệu phân tán* mà còn cung cấp kinh nghiệm thực tế trong việc thiết kế và triển khai các giải pháp quản lý dữ liệu trên quy mô lớn, mở ra hướng ứng dụng cho các hệ thống cần xử lý big data trong tương lai.

Đóng góp của từng cá nhân trong bài tập lớn

Họ tên	MSV	Nhiệm vụ	Trạng thái
Đặng Trung Kiên	B22DCCN426	- Trưởng nhóm - Cài đặt và kết nối CSDL - Cài đặt và triển khai chương trình -Tim hiểu về LoadRatings	Hoàn thành
Ngô Hồng Phúc	B22DCCN629	- Viết báo cáo - Tìm hiểu về Rangepartition, Rangeinsert	Hoàn thành
Nguyễn Việt Tiến	B22DCCN726	- Viết báo cáo - Tìm hiểu về Roundrobinpartition, Roundrobininsert	Hoàn thành