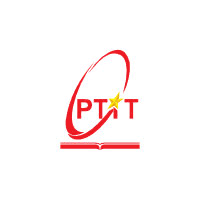
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**



**BÁO CÁO BÀI TẬP LỚN**

**HỌC PHẦN: CƠ SỞ DỮ LIỆU PHÂN TÁN**

**MÃ HỌC PHẦN: INT14148**

**ĐỀ TÀI: MÔ TẢ PHÂN MẢNH DỮ LIỆU**

Các sinh viên thực hiện:

|  |  |
| --- | --- |
| B22DCCN090 | Nguyễn Thành Công |
| B22DCCN702 | Bùi Thái Sỹ |
| B22DCCN496 | Hoàng Hải Long |

Nhóm bài tập lớn: 10

Nhóm lớp: 09

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

**HÀ NỘI 2025**

**PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN**

|  |  |  |
| --- | --- | --- |
| **TT** | **SV thực hiện** | **Công việc / Nhiệm vụ** |
| 1 | Bùi Thái Sỹ | Nhóm trưởng, phân chia công việc, tối ưu hàm loadratings(), cải thiện hiệu suất các hàm. |
| 2 | Nguyễn Thành Công | Tối ưu hàm rangeinsert() và roundrobininsert() sử dụng metadata trong hai hàm partition(). |
| 3 | Hoàng Hải Long | Tối ưu rangepartition() ,roundrobinpartition() dùng unlogged table và tối ưu truy vấn. |

MỤC LỤC

[MỤC LỤC 3](#_Toc200478057)

[DANH MỤC CÁC HÌNH VẼ 5](#_Toc200478058)

[DANH MỤC CÁC BẢNG BIỂU 5](#_Toc200478059)

[MỞ ĐẦU 6](#_Toc200478060)

[CHƯƠNG 1. TỔNG QUAN VỀ BÀI TOÁN 7](#_Toc200478061)

[1.1 Đề bài 7](#_Toc200478062)

[1.2 Nhiệm vụ yêu cầu 7](#_Toc200478063)

[1.3 Yêu cầu bắt buộc: 8](#_Toc200478064)

[1.4 Vấn đề của bài toán 9](#_Toc200478065)

[1.5 Kết chương 9](#_Toc200478066)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 10](#_Toc200478067)

[2.1 Khái quát 10](#_Toc200478068)

[2.2 Phân mảnh ngang phân chia dựa trên N khoảng giá trị đồng đều 10](#_Toc200478069)

[2.3 Phân mảnh kiểu vòng tròn 11](#_Toc200478070)

[2.4 So sánh hai kiểu phân mảnh ngang 12](#_Toc200478071)

[2.5 Kết chương 12](#_Toc200478072)

[CHƯƠNG 3. TRIỂN KHAI, CÀI ĐẶT VÀ ĐÁNH GIÁ 13](#_Toc200478073)

[3.1 Cấu trúc thư mục 13](#_Toc200478074)

[3.2 Cài thư viện hỗ trợ 13](#_Toc200478075)

[3.3 Cài đặt PostgreSQL 13](#_Toc200478076)

[3.4 Tải tệp rating.dat 15](#_Toc200478077)

[3.5 Luồng hoạt động của file Assignment1Tester.py 15](#_Toc200478078)

[3.6 Các thư viện sử dụng và các const 16](#_Toc200478079)

[3.7 Các hàm phụ cần cài đặt 17](#_Toc200478080)

[3.7.1 mesure\_time 17](#_Toc200478081)

[3.7.2 getopenconnection 17](#_Toc200478082)

[3.7.3 init\_range\_metadata\_table 18](#_Toc200478083)

[3.7.4 init\_rrobin\_metadata\_table 18](#_Toc200478084)

[3.7.5 update\_range\_metadata 18](#_Toc200478085)

[3.7.6 update\_rrobin\_metadata 19](#_Toc200478086)

[3.7.7 get\_range\_metadata 19](#_Toc200478087)

[3.7.8 get\_rrobin\_metadata 19](#_Toc200478088)

[3.7.9 create\_db 20](#_Toc200478089)

[3.7.10 count\_partitions 20](#_Toc200478090)

[3.8 Hàm loadratings() 21](#_Toc200478091)

[3.8.1 Tối ưu hơn mà không dùng thư viện 22](#_Toc200478092)

[3.8.2 Sử dụng thư viện DuckDB 25](#_Toc200478093)

[3.8.3 Sử dụng thư viện Polars 27](#_Toc200478094)

[3.9 Hàm rangepartition() 29](#_Toc200478095)

[3.9.1 Sử dụng UNLOGGED TABLE 30](#_Toc200478096)

[3.9.2 Tối ưu truy vấn không sử dụng UNLOGGED TABLE 32](#_Toc200478097)

[3.10 Hàm roundrobinpartition() 34](#_Toc200478098)

[3.10.1 Sử dụng UNLOGGED TABLE 35](#_Toc200478099)

[3.10.2 Tối ưu truy vấn không sử dụng UNLOGGED TABLE 37](#_Toc200478100)

[3.11 Hàm rangeinsert() 39](#_Toc200478101)

[3.12 Hàm roundrobininsert() 44](#_Toc200478102)

[3.13 Kết chương 49](#_Toc200478103)

[KẾT LUẬN 50](#_Toc200478104)

[TÀI LIỆU THAM KHẢO 51](#_Toc200478105)

DANH MỤC CÁC HÌNH VẼ

[Hình 1 – Cấu trúc thư mục 15](#_Toc200479324)

[Hình 2 – Các thư viện trong requirements.txt 15](#_Toc200479325)

[Hình 3 – Cài đặt các thư viện hỗ trợ 15](#_Toc200479326)

[Hình 4 – Giao diện pgAdmin 16](#_Toc200479327)

[Hình 5 – Danh sách các database 16](#_Toc200479328)

[Hình 6 – Danh sách các bảng trong db dds\_assgn1 16](#_Toc200479329)

[Hình 7 – 20 dòng dữ liệu đầu tiên của file rating.dat 17](#_Toc200479330)

[Hình 8 – Luồng hoạt động file Assignment1Tester.py 17](#_Toc200479331)

[Hình 9 – Các thư viện sử dụng 18](#_Toc200479332)

[Hình 10 – Các const 18](#_Toc200479333)

[Hình 11 – Hàm mesure\_time 19](#_Toc200479334)

[Hình 12 – Hàm getopenconnection 20](#_Toc200479335)

[Hình 13 – Hàm init\_range\_metadata\_table 20](#_Toc200479336)

[Hình 14 – Hàm init\_rrobin\_metadata\_table 20](#_Toc200479337)

[Hình 15 – Hàm update\_range\_metadata 21](#_Toc200479338)

[Hình 16 – Hàm update\_rrobin\_metadata 21](#_Toc200479339)

[Hình 17 – Hàm get\_ range\_metadata 21](#_Toc200479340)

[Hình 18 – Hàm get\_ rrobin\_metadata 22](#_Toc200479341)

[Hình 19 – Hàm create\_db 22](#_Toc200479342)

[Hình 20 – Hàm count\_partitions 23](#_Toc200479343)

[Hình 21 – Hàm loadratings() mẫu 23](#_Toc200479344)

[Hình 22 – Thời gian thực thi hàm loadratings() mẫu 23](#_Toc200479345)

[Hình 23 – Hàm loadratings() tối ưu không dùng thư viện có sẵn 25](#_Toc200479346)

[Hình 24 – Thời gian thực thi hàm loadratings() tối ưu không dùng thư viện có sẵn 25](#_Toc200479347)

[Hình 25 – Hàm loadratings() sử dụng DuckDB – 1 28](#_Toc200479348)

[Hình 26 – Hàm loadratings() sử dụng DuckDB – 2 28](#_Toc200479349)

[Hình 27 – Thời gian thực thi hàm loadratings() sử dụng DuckDB 29](#_Toc200479350)

[Hình 28 – Hàm loadratings() sử dụng Polars 30](#_Toc200479351)

[Hình 29 – Thời gian thực thi hàm loadratings() sử dụng Polars 30](#_Toc200479352)

[Hình 30 – Table ratings sau khi thực hiện hàm loadratings() 31](#_Toc200479353)

[Hình 31 – Hàm rangepartition() mẫu 32](#_Toc200479354)

[Hình 32 – Thời gian thực thi hàm rangepartition() mẫu 32](#_Toc200479355)

[Hình 33 – Hàm rangepartition() dùng unlogged table 33](#_Toc200479356)

[Hình 34 – Thời gian thực thi hàm rangepartition() dùng unlogged table 33](#_Toc200479357)

[Hình 35 – Hàm rangepartition() không dùng unlogged table 35](#_Toc200479358)

[Hình 36 – Thời gian thực thi hàm rangepartition() không dùng unlogged table 35](#_Toc200479359)

[Hình 37 – Hàm rangepartition() không dùng unlogged table và có bảng metadata 36](#_Toc200479360)

[Hình 38 – Thời gian thực thi hàm rangepartition() không dùng unlogged table và có bảng metadata 36](#_Toc200479361)

[Hình 39 – Hàm roundrobinpartition() mẫu 37](#_Toc200479362)

[Hình 40 – Thời gian thực thi hàm roundrobinpartition() mẫu 37](#_Toc200479363)

[Hình 41 – Hàm roundrobinpartition() dùng unlogged table 38](#_Toc200479364)

[Hình 42 – Thời gian thực thi hàm roundrobinpartition() dùng unlogged table 38](#_Toc200479365)

[Hình 43 – Hàm roundrobinpartition() không dùng unlogged table 40](#_Toc200479366)

[Hình 44 – Thời gian thực thi hàm roundrobinpartition() không dùng unlogged table 40](#_Toc200479367)

[Hình 45 – Hàm roundrobinpartition() không dùng unlogged table và có bảng metadata 41](#_Toc200479368)

[Hình 46 – Thời gian thực thi hàm roundrobinpartition() không dùng unlogged table và có bảng metadata 41](#_Toc200479369)

[Hình 47 – Hàm rangeinsert() mẫu 42](#_Toc200479370)

[Hình 48 – Thời gian thực thi hàm rangeinsert() mẫu 42](#_Toc200479371)

[Hình 49 – Hàm rangeinsert() khi bảng metadata. 43](#_Toc200479372)

[Hình 50 – Thời gian thực thi hàm rangeinsert() khi sử dụng bảng metadata. 43](#_Toc200479373)

[Hình 51 – Table range\_metadata sau khi thực hiện hàm rangeinsert() 43](#_Toc200479374)

[Hình 52 – Table range\_part0 sau khi thực hiện hàm rangeinsert() 44](#_Toc200479375)

[Hình 53 – Table range\_part1 sau khi thực hiện hàm rangeinsert() 44](#_Toc200479376)

[Hình 54 – Table range\_part2 sau khi thực hiện hàm rangeinsert() 45](#_Toc200479377)

[Hình 55 – Table range\_part3 sau khi thực hiện hàm rangeinsert() 45](#_Toc200479378)

[Hình 56 – Table range\_part4 sau khi thực hiện hàm rangeinsert() 46](#_Toc200479379)

[Hình 57 – Hàm roundrobininsert() mẫu 47](#_Toc200479380)

[Hình 58 – Thời gian thực thi hàm roundrobininsert() mẫu 47](#_Toc200479381)

[Hình 59 – Hàm roundrobininsert() khi bảng metadata. 48](#_Toc200479382)

[Hình 60 – Thời gian thực thi hàm roundrobininsert() khi sử dụng bảng metadata. 48](#_Toc200479383)

[Hình 61 – Table range\_metadata sau khi thực hiện hàm roundrobininsert() 48](#_Toc200479384)

[Hình 62 – Table rrobin\_part0 sau khi thực hiện hàm roundrobininsert() 49](#_Toc200479385)

[Hình 63 – Table rrobin\_part1 sau khi thực hiện hàm roundrobininsert() 49](#_Toc200479386)

[Hình 64 – Table rrobin\_part2 sau khi thực hiện hàm roundrobininsert() 50](#_Toc200479387)

[Hình 65 – Table rrobin\_part3 sau khi thực hiện hàm roundrobininsert() 50](#_Toc200479388)

[Hình 66 – Table rrobin\_part4 sau khi thực hiện hàm roundrobininsert() 51](#_Toc200479389)

DANH MỤC CÁC BẢNG BIỂU

[Bảng 1. Ví dụ phân mảnh kiểu vòng tròn 14](#_Toc200479488)

[Bảng 2. So sánh hai kiểu phân mảnh ngang 15](#_Toc200479489)

[Bảng 3. So sánh solution mẫu với phương pháp tối ưu không dùng thư viện có sẵn 27](#_Toc200479490)

[Bảng 4. So sánh phương pháp tối ưu không dùng thư viện với dùng DuckDB 30](#_Toc200479491)

[Bảng 5. So sánh dùng DuckDB với dùng Polars 32](#_Toc200479492)

[Bảng 6. So sánh solution mẫu với dùng unlogged table hàm rangepartition() 35](#_Toc200479493)

[Bảng 7. So sánh solution mẫu với dùng unlogged table hàm rrobinpartition() 40](#_Toc200479494)

MỞ ĐẦU

Báo cáo bài tập lớn gồm 3 chương với nội dung chính như sau:

- Chương 1 tổng quan về bài toán mô phỏng các phương pháp phân mảnh dữ liệu trên một hệ quản trị cơ sở dữ liệu quan hệ, bao gồm các nội dung khái quát về bài toán, vấn đề của bài toán.

- Chương 2 thực hiện việc nghiên cứu các cơ sở lý thuyết của các thành phần trong bài toán mô phỏng phân mảnh dữ liệu, trong đó bao gồm phân mảnh ngang phân chia dựa trên N khoảng giá trị đồng đều và phân mảnh kiểu vòng tròn.

- Chương 3 thực hiện việc triển khai, cài đặt và đánh giá phương pháp phân mảnh, trong đó bao gồm việc triển khai, cài đặt theo nhiều cách, từ đó so sánh, cải thiện tốc độ của các hàm cần thực hiện.

1. TỔNG QUAN VỀ BÀI TOÁN
   1. Đề bài

Nhiệm vụ yêu cầu là 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.

Dữ liệu đầu vào là một tập dữ liệu đánh giá phim được thu thập từ trang web MovieLens ([http://movielens.org](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

* 1. Nhiệm vụ yêu cầu

1. Tải về máy ảo có môi trường giống với máy chấm điểm. Có thể dùng máy của mình, nhưng không đảm bảo mã sẽ chạy đúng trên máy chấm điểm. Nếu dùng máy ảo được cung cấp thì bỏ qua Bước 2.

**Cấu hình máy ảo**: Python 3.12.x, OS: Ubuntu hoặc Windows 10.

1. Cài đặt PostgreSQL hoặc MySQL.
2. Tải tệp rating.dat từ trang MovieLens: <http://files.grouplens.org/datasets/movielens/ml-10m.zip>

Có thể sử dụng dữ liệu một phần để kiểm tra. Một tệp dữ liệu kiểm tra được cung cấp trong repository này.

1. Cài đặt hàm Python LoadRatings() nhận vào một đường dẫn tuyệt đối đến tệp

rating.dat.

LoadRatings() sẽ tải nội dung tệp vào một bảng trong PostgreSQL có tên **Ratings** với schema sau:

* UserID (int)
* MovieID (int)
* Rating (float)

1. Cài đặt hàm Python Range\_Partition() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL) và (2) 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.

1. Cài đặt hàm Python RoundRobin\_Partition() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL) và (2) 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 (hoặc MySQL), sử dụng phương pháp phân mảnh kiểu **vòng tròn (round robin)** (đã được giải thích trong lớp).

1. Cài đặt hàm Python RoundRobin\_Insert() nhận vào: (1) bảng Ratings trong PostgreSQL, (2) UserID, (3) ItemID, (4) Rating.

RoundRobin\_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh theo cách round robin.

1. Cài đặt hàm Python Range\_Insert() nhận vào: (1) bảng Ratings trong PostgreSQL (hoặc MySQL), (2) UserID, (3) ItemID, (4) Rating.

Range\_Insert() sẽ chèn một bộ mới vào bảng Ratings và vào đúng phân mảnh dựa trên giá trị của Rating.

* 1. Yêu cầu bắt buộc:
* Số phân mảnh bắt đầu từ 0, nếu có 3 phân mảnh thì tên các bảng sẽ là range\_part0, range\_part1, range\_part2 cho phân mảnh theo khoảng, và tương tự cho phân mảnh vòng tròn.
* Không được thay đổi tiền tố tên bảng phân mảnh đã được cung cấp trong

assignment\_tester.py.

* Không được mã hóa cứng tên tệp đầu vào.
* Không được đóng kết nối bên trong các hàm đã triển khai.
* Không được mã hóa cứng tên cơ sở dữ liệu.
* Lược đồ bảng phải giống với mô tả ở bước 4.
* Không dùng biến toàn cục trong quá trình triển khai. Cho phép sử dụng bảng meta-data.
* Không được sửa đổi tệp dữ liệu.
* Việc vượt qua tất cả các testcase kiểm thử không đảm bảo kết quả đúng hoàn toàn. Nó chỉ có nghĩa là mã không có lỗi biên dịch. Để kiểm tra đầy đủ, cần kiểm tra nội dung các bảng trong cơ sở dữ liệu.
* Hai hàm chèn có thể được gọi nhiều lần bất kỳ lúc nào. Chúng được thiết kế để duy trì các bảng trong cơ sở dữ liệu khi có thao tác chèn.
  1. Vấn đề của bài toán

Tệp dữ liệu đầu vào là tệp rating.dat có khoảng 10 triệu dòng dữ liệu. Đây là lượng dữ liệu rất là lớn nên những hàm cần tối ưu cần thiết nhất là ba hàm LoadRatings(), Range\_Partition(), RoundRobin\_Partition(). Tiếp theo đó là hai hàm chèn một bộ dữ liệu mới vào bảng Ratings là RoundRobin\_Insert() và Range\_Insert(). Với một hàng dữ liệu thì có thể thời gian 0.02s là nhỏ. Nhưng khi có nhiều dòng dữ liệu giả sử như 10 triệu bản ghi thì thời gian để thêm dữ liệu vào bảng ước chừng 56 giờ.

* 1. Kết chương

Chương này đã khái quát lại các yêu cầu của bài toán cũng như đưa ra các vấn đề cần xử lý. Chương tiếp theo sẽ tìm hiểu về cơ sở lý thuyết áp dụng chính trong bài toán.

1. CƠ SỞ LÝ THUYẾT
   1. Khái quát

Như đã đề cập trong phần Mở đầu, chương này tập trung vào các cơ sở lý thuyết của phương pháp phân mảnh dữ liệu:

* Phân mảnh ngang phân chia dựa trên N khoảng giá trị đồng đều.
* Phân mảnh kiểu vòng tròn.
  1. Phân mảnh ngang phân chia dựa trên N khoảng giá trị đồng đều

Phân mảnh ngang phân chia dựa trên N khoảng giá trị đồng đều là một phương pháp trong hệ quản trị cơ sở dữ liệu phân tán, trong đó một quan hệ (bảng) được chia ra thành N phân mảnh ngang, dựa trên một thuộc tính có giá trị liên tục (hoặc rời rạc có thể sắp xếp) sao cho mỗi phân mảnh chứa các bản ghi có giá trị thuộc tính nằm trong một khoảng giá trị xác định trước.

Ví dụ:

Bảng Customer(ID, Age, Name, Tel)

Muốn phân mảnh theo thuộc tính Age từ 18 đến 60.

Số lượng phân mảnh: N = 5.

Khi đó mỗi khoảng tuổi là: (làm tròn thành 9)

Ví dụ các phân mảnh:

F1: Age >= 18 AND Age < 27

F2: Age >= 27 AND Age < 36

F3: Age >= 36 AND Age < 45

F4: Age >= 45 AND Age < 54

F5: Age >= 54 AND Age <= 60

Đặc điểm:

* Các phân mảnh rời nhau, không trùng lặp.
* Gộp lại sẽ được toàn bộ bảng ban đầu.
* Phù hợp với truy vấn theo dải giá trị.
* Không đảm bảo số bản ghi trong mỗi phân mảnh là bằng nhau, mà chỉ đồng đều theo giá trị thuộc tính chia.

Ưu điểm:

* Dễ triển khai.
* Phân bố dữ liệu hợp lý khi thuộc tính chia có phân phối đều.
* Hỗ trợ tốt cho truy vấn vùng.

Nhược điểm:

* Nếu dữ liệu không phân bố đều, có thể dẫn đến mất cân bằng tải.
* Không tối ưu nếu truy vấn không dựa trên thuộc tính phân mảnh.
  1. Phân mảnh kiểu vòng tròn

Phân mảnh kiểu vòng tròn (Round-Robin Fragmentation) là một kiểu phân mảnh ngang không dựa trên điều kiện của thuộc tính, mà dựa trên vị trí (thứ tự) của các bản ghi, chia tuần tự theo vòng tròn qua các nút hoặc phân mảnh.

Ví dụ:

Bảng Employee với 9 bản ghi, và muốn phân thành 3 phân mảnh:

|  |  |  |
| --- | --- | --- |
| STT | Bản ghi | Phân mảnh |
| 1 | e1 | F1 |
| 2 | e2 | F2 |
| 3 | e3 | F3 |
| 4 | e4 | F1 |
| 5 | e5 | F2 |
| 6 | e6 | F3 |
| 7 | e7 | F1 |
| 8 | e8 | F2 |
| 9 | e9 | F3 |

1. Ví dụ phân mảnh kiểu vòng tròn

Mỗi phân mảnh sẽ chứa các bản ghi theo chu kỳ tròn: F1 nhận e1, e4, e7; F2 nhận e2, e5, e8; F3 nhận e3, e6, e9.

Ưu điểm:

* Cân bằng tải vì mỗi phân mảnh có số bản ghi gần như bằng nhau.

Nhược điểm:

* Không tối ưu cho truy vấn. Truy vấn theo điều kiện (ví dụ: WHERE Age > 40) có thể cần truy vấn mọi phân mảnh, do không có phân vùng theo giá trị.
* Không dựa trên logic thuộc tính nên có thể gât ra chậm. Ví dụ truy vấn chỉ cần truy cập một phần nhỏ của hệ thống (ví dụ: một phân mảnh hoặc một nút) thì cách này sẽ quét toàn bộ dữ liệu của một tập phân mảnh rất lớn. Đây gọi là khó tận dụng locality(tính địa phương) trong truy vấn theo vùng giá trị.
  1. So sánh hai kiểu phân mảnh ngang

|  |  |  |
| --- | --- | --- |
| Tính chất | Phân mảnh theo khoảng giá trị | Phân mảnh kiểu vòng tròn |
| Dựa trên giá trị thuộc tính | Có | Không |
| Cân bằng tải | Phục thuộc phân phối dữ liệu | Luôn đồng đều |
| Hỗ trợ truy vấn theo | Vùng (Ví dụ: tìm số nhân viên có tuổi > 20) | Toàn bộ bảng (Ví dụ: tính tổng, trung bình,…) |
| Đơn giản hóa thiết kế | Không | Rất đơn giản |

1. So sánh hai kiểu phân mảnh ngang
   1. Kết chương

Chương này mô tả cơ sở lý thuyết cho hai kiểu phân mảnh ngang. Đây sẽ là nền tảng và cơ sở cho chương 3 thực hiện triển khai, cài đặt và phân tích tối ưu cho các hàm cần cài đăt.

1. TRIỂN KHAI, CÀI ĐẶT VÀ ĐÁNH GIÁ
   1. Cấu trúc thư mục

A screen shot of a computer code

AI-generated content may be incorrect.

1. – Cấu trúc thư mục

Thư mục gồm 2 file Assignment1Tester.py, testHelper.py, test\_data.dat dùng để test. Các file loadratingsupdate.py, rangepartitionupdate.py, roundrobinpartitionupdate.py để chứa các phiên bản update của các hàm tương ứng. File Interface\_Sample.py là file solution mẫu. File Interface.py là file solution của nhóm.

* 1. Cài thư viện hỗ trợ

Cài thư viện hỗ trợ:

* Mở thư mục dự án trên cmd
* Cài đặt các thư viện trong requirements.txt:

A white background with black text

AI-generated content may be incorrect.

1. – Các thư viện trong requirements.txt

pip install -r .\requirements.txt

A white background with black text

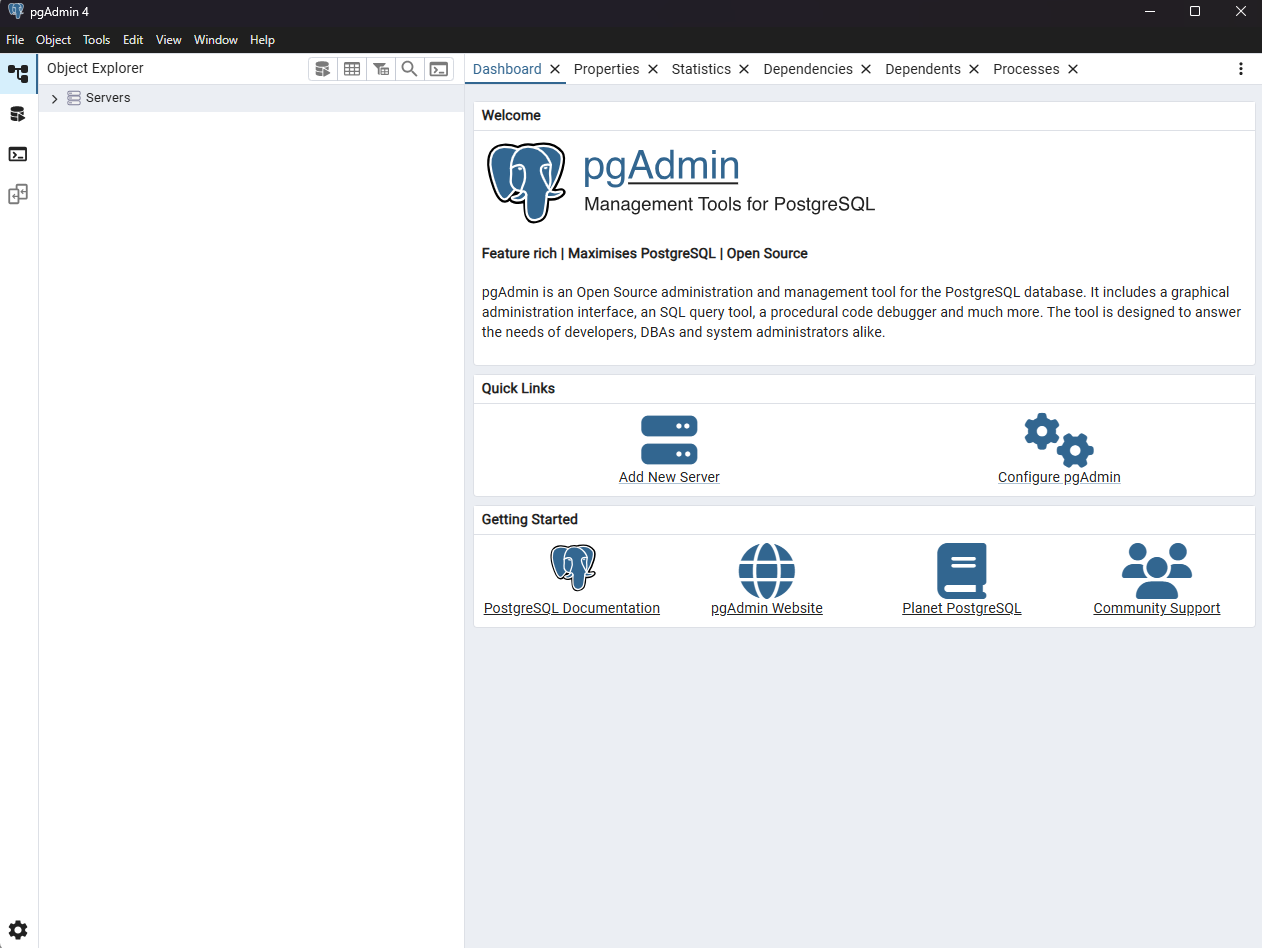
AI-generated content may be incorrect.

1. – Cài đặt các thư viện hỗ trợ
   1. Cài đặt PostgreSQL

Tải PostgreSQL từ <https://www.postgresql.org/download/windows/>

Sau khi download xong được tệp postgresql-17.5-1-windows-x64.exe

Làm theo hướng dẫn thì được giao diện pgAdmin để thao tác với PostgreSQL

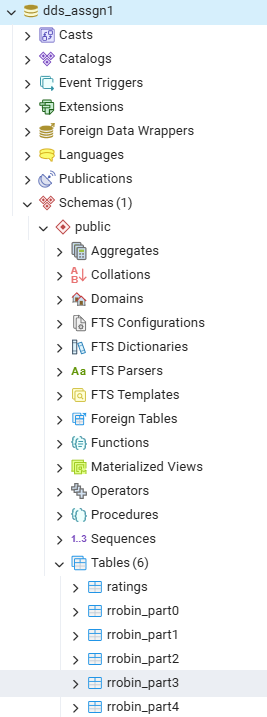


1. – Giao diện pgAdmin

A screenshot of a computer

AI-generated content may be incorrect.

1. – Danh sách các database



1. – Danh sách các bảng trong db dds\_assgn1
   1. Tải tệp rating.dat

Tải tệp từ <http://files.grouplens.org/datasets/movielens/ml-10m.zip>

Dữ liệu gồm có 10000054 dòng. Có cấu trúc mỗi dòng:

UserID::MovieID::Rating::Timestamp

A screenshot of a computer

AI-generated content may be incorrect.

1. – 20 dòng dữ liệu đầu tiên của file rating.dat
   1. Luồng hoạt động của file Assignment1Tester.py

A diagram of a diagram

AI-generated content may be incorrect.

1. – Luồng hoạt động file Assignment1Tester.py

Khi chạy file Assignment1Tester.py thì:

* Đầu tiên chương trình thực hiện xóa hết tất cả các bảng public.
* Thực hiện hàm loadratings() để load dữ liệu từ file rating.dat vào trong database.
* Thực hiện hàm rangepartition() để phân mảnh ngang theo n khoảng đồng đều.
* Thực hiện hàm rangeinsert() để insert một bản ghi vào bảng phân mảnh phù hợp.
* Xóa hết tất cả các bảng public.
* Thực hiện hàm loadratings() để load dữ liệu từ file rating.dat vào trong database
* Thực hiện hàm roundrobinpartition() để phân mảnh ngang theo n khoảng đồng đều.
* Thực hiện hàm roundrobininsert() để insert một bản ghi vào bảng phân mảnh phù hợp.
* Xóa hết tất cả các bảng public.
  1. Các thư viện sử dụng và các const

A screen shot of a computer code

AI-generated content may be incorrect.

1. – Các thư viện sử dụng

* psycopg2: Kết nối và thao tác với cơ sở dữ liệu PostgreSQL.
* StringIO từ io: Tạo đối tượng file giả từ chuỗi trong bộ nhớ (giống file thực, nhưng nằm trong RAM).
* time: Đo thời gian thực thi hàm.
* functools: Dùng để giữ metadata của hàm khi sử dụng decorator.
* duckdb: Xử lý tệp dữ liệu nhanh chóng bằng SQL nội bộ mà không cần database server.
* polars: Dùng để đọc từng dòng dữ liệu.
* os, tempfile: Dùng để quản lý hệ thống file.

A computer code with black text

AI-generated content may be incorrect.

1. – Các const

* DATABASE\_NAME: Tên cơ sở dữ liệu được sử dụng.
* RANGE\_METADATA\_TABLE: Tên bảng metadata dùng cho phân vùng theo khoảng.
* RROBIN\_METADATA\_TABLE: Tên bảng metadata dùng cho phân vùng kiểu round-robin.
  1. Các hàm phụ cần cài đặt
     1. mesure\_time

Hàm measure\_time() là một decorator dùng để đo thời gian thực thi của một hàm bất kỳ.

A screenshot of a computer code

AI-generated content may be incorrect.

1. – Hàm mesure\_time

Dùng @measure\_time để áp dụng decorator này lên một hàm.

Khi hàm đó được gọi, decorator sẽ:

1. Ghi lại thời gian bắt đầu.
2. Gọi hàm gốc.
3. Ghi lại thời gian kết thúc.
4. Tính và in ra thời gian chênh lệch.
   * 1. getopenconnection

Hàm getopenconnection() có nhiệm vụ kết nối đến cơ sở dữ liệu PostgreSQL bằng thư viện psycopg2.

A computer code with many colored text

AI-generated content may be incorrect.

1. – Hàm getopenconnection
   * 1. init\_range\_metadata\_table

Hàm init\_range\_metadata\_table() có nhiệm vụ tạo bảng metadata cho phân vùng theo range.

A computer screen shot of a computer code

AI-generated content may be incorrect.

1. – Hàm init\_range\_metadata\_table
   * 1. init\_rrobin\_metadata\_table

Hàm init\_rrobin\_metadata\_table() có nhiệm vụ tạo bảng metadata cho phân vùng theo round robin.

A computer screen shot of a computer code

AI-generated content may be incorrect.

1. – Hàm init\_rrobin\_metadata\_table
   * 1. update\_range\_metadata

Hàm update\_range\_metadata() có nhiệm vụ cập nhật số lượng phân vùng cho bảng range\_metadata.

A screen shot of a computer code

AI-generated content may be incorrect.

1. – Hàm update\_range\_metadata
   * 1. update\_rrobin\_metadata

Hàm update\_rrobin\_metadata() có nhiệm vụ cập nhật số lượng và trạng thái phân vùng cuối cho bảng rrobin\_metadata.

A computer screen shot of a computer code

AI-generated content may be incorrect.

1. – Hàm update\_rrobin\_metadata
   * 1. get\_range\_metadata

Hàm get\_range\_metadata() có nhiệm vụ lấy số lượng phân vùng RANGE của một bảng.

A screen shot of a computer

AI-generated content may be incorrect.

1. – Hàm get\_ range\_metadata
   * 1. get\_rrobin\_metadata

Hàm get\_rrobin\_metadata() có nhiệm vụ lấy metadata của bảng ROUND ROBIN (số lượng và chỉ số phân vùng cuối).

A screen shot of a computer code

AI-generated content may be incorrect.

1. – Hàm get\_ rrobin\_metadata
   * 1. create\_db

Hàm create\_db() có nhiệm vụ tạo một cơ sở dữ liệu mới trong PostgreSQL.

A screenshot of a computer code

AI-generated content may be incorrect.

1. – Hàm create\_db
   * 1. count\_partitions

Hàm count\_partitions() có nhiệm vụ đếm số lượng bảng trong cơ sở dữ liệu hiện tại mà tên bảng bắt đầu bằng chuỗi `prefix`.

Ví dụ: nếu prefix = 'range\_ratings\_part', và trong CSDL có các bảng:

        - range\_ratings\_part0

        - range\_ratings\_part1

        - rrobin\_ratings\_part0

    thì hàm sẽ trả về 2 (vì chỉ có 2 bảng có tên bắt đầu bằng 'range\_ratings\_part').

A screenshot of a computer code

AI-generated content may be incorrect.

1. – Hàm count\_partitions
   1. Hàm loadratings()

Solution mẫu:

A screenshot of a computer

AI-generated content may be incorrect.

1. – Hàm loadratings() mẫu



1. – Thời gian thực thi hàm loadratings() mẫu

Ý tưởng chính:

* Tạo bảng gồm 7 cột userid, extra1, movieid, extra2, rating, extra3, timestamp (trong đó extra1, 2, 3 là 3 cột chứa các dấu ‘:’).
* Tách dòng dữ liệu UserID::MovieID::Rating::Timestamp thành 7 cột được tách bởi dấu ‘:’ (Chỉ có thể truyền vào đơn kí tự chứ không truyền được 2 kí tự nên không truyền được “::”).
* Nạp dữ liệu vào bảng.
* Xóa đi 3 cột dấu ‘:’ và cột timestamp.

Ý tưởng rất hay khi xử lý nhanh với copy\_from để copy dữ liệu thẳng vào database. Sau đó xử lý phần dữ liệu thừa bằng câu lệnh drop column.

Tuy nhiên, cũng có thể thử một số cách khác để xem có cải thiện được về mặt thời gian hay không.

* + 1. Tối ưu hơn mà không dùng thư viện

Ý tưởng chính:

* Tăng buffer đọc file: Đọc file với buffering=65536 (64KB) để giảm số lần truy cập đĩa.
* Dùng StringIO: Tránh phải tạo file CSV trung gian trên ổ đĩa, tất cả xử lý trong bộ nhớ RAM.
* Tắt synchronous\_commit: Bỏ qua ghi WAL để tăng tốc độ ghi.
* Tăng work\_mem, maintenance\_work\_mem: Cho phép PostgreSQL dùng nhiều RAM hơn trong quá trình COPY và tạo bảng.
* Ngoài ra việc không sử dụng thư viện ngoài cũng giảm chi phí về mặt bộ nhớ.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm loadratings() tối ưu không dùng thư viện có sẵn



1. – Thời gian thực thi hàm loadratings() tối ưu không dùng thư viện có sẵn

Giải thích vì sao nhanh hơn solution mẫu:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Solution mẫu | Tối ưu không sử dụng thư viện có sẵn |
| Tối ưu PostgreSQL | Không có | Có (3 lệnh SET) |
| Tạo bảng | 7 cột (4 cột thừa) | 3 cột cần thiết |
| Thao tác ALTER | 1 lần (DROP 4 cột) | 0 lần |
| Ghi vào PostgreSQL siêu nhanh | Dùng copy\_from() | Dùng copy\_from() với tab-separated và buffer lớn (size=65536) |
| Tổng thao tác DB | 2 | 1 |
| Tối ưu hệ thống PostgreSQL | Không | SET synchronous\_commit=OFF, work\_mem, fillfactor tối đa |

1. So sánh solution mẫu với phương pháp tối ưu không dùng thư viện có sẵn

Cách thứ hai nghĩ đến để không cần phải thao tác nhiều là sử dụng thư viện có sẵn.

duckdb và polars là hai thư viện phổ biến để xử lý. Hai thư viện này thực hiện việc nạp dữ liệu đa luồng nên tốc độ sẽ được cải thiện đáng kể.

*DuckDB*

DuckDB là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) trong bộ nhớ (in- memory) với khả năng xử lý và truy vấn dữ liệu cực nhanh. Nó được thiết kế để làm việc với các tập dữ liệu lớn, sử dụng tối đa tài nguyên của hệ thống để xử lý dữ liệu mà không gặp phải các hạn chế về hiệu suất và bộ nhớ.

Cơ chế tối ưu:

Vectorized Execution: DuckDB sử dụng mô hình thực thi dạng vector (Vectorized Execution), trong đó dữ liệu được xử lý theo từng khối (block) thay vì từng hàng (row). Điều này tối ưu hóa hiệu suất bằng cách tận dụng tốt hơn bộ nhớ đệm CPU.

Columnar Storage: Dữ liệu được lưu trữ theo định dạng cột (columnar format), giúp tăng tốc các thao tác quét cột (column scans), thường gặp trong các bài toán phân tích.

Late Materialization: DuckDB chỉ tải các cột cần thiết cho phép toán, giúp giảm tải bộ nhớ và tăng hiệu suất.

Parallel Query Execution: DuckDB tự động thực hiện các phép toán truy vấn song song (multi-threading), tận dụng hiệu quả các lõi CPU có sẵn.

Zero-Copy Integration: DuckDB hỗ trợ tích hợp không sao chép (zero-copy) với các framework dữ liệu như Pandas, PyArrow, và NumPy, cho phép thao tác trực tiếp trên dữ liệu mà không cần chuyển đổi hay sao chép.

Disk-Based Query Execution: Nếu dữ liệu quá lớn để chứa trong bộ nhớ, DuckDB có thể xử lý trực tiếp trên đĩa mà không cần cấu hình phức tạp, điều này rất hữu ích cho dữ liệu mười triệu dòng.

Nói tóm lại, DuckDB có điểm mạnh như sau:

Khả năng xử lý song song: Tự động tối ưu hóa việc sử dụng đa luồng (multi-threading), mang lại hiệu suất cao mà không cần cấu hình thêm.

Tích hợp dễ dàng: Hỗ trợ nhiều định dạng dữ liệu như CSV, Parquet, Arrow, giúp xử lý trực tiếp mà không cần chuyển đổi.

Hiệu quả bộ nhớ: Nhờ việc chỉ tải các cột cần thiết và sử dụng vectorized execution, DuckDB tiêu thụ rất ít bộ nhớ trong so với các công cụ khác như Pandas hay Spark.

*Polars*

Polars là một thư viện xử lý dữ liệu nhanh chóng, tối ưu hóa hiệu suất bằng cách  
sử dụng kỹ thuật mô hình hóa dữ liệu và tính toán song song (parallel computation). Polars tận dụng tính toán song song bằng cách chia dữ liệu thành các phần nhỏ và xử lý đồng thời.

Cơ chế tối ưu:

Chia nhỏ dữ liệu (Data Chunking): Phân chia dữ liệu thành các phần nhỏ giúp giảm bớt độ phức tạp và tối ưu tốc độ.

Tính toán song song (Parallelism): sử dụng nhiều luồng xử lý hoặc phân phối công việc giữa các lõi của CPU giúp giảm thời gian xử lý tổng thể.

Tối ưu hóa bộ nhớ (Memory Optimization): Polars sử dụng cấu trúc dữ liệu hiệu  
quả hơn để tối thiểu hóa việc sử dụng bộ nhớ và nâng cao tốc độ truy xuất dữ liệu.

Nói tóm lại, Polars có điểm mạnh như sau:

Tốc độ xử lý: Polars có thể xử lý dữ liệu nhanh gấp 3-4 lần so với Pandas khi làm việc với các bộ dữ liệu lớn. Nhờ vào tính toán song song và tối ưu hóa bộ nhớ.

Quản lý bộ nhớ: Polars sử dụng bộ nhớ ít hơn khi làm việc với các tập dữ liệu lớn, giúp tiết kiệm tài nguyên hệ thống và giảm nguy cơ tràn bộ nhớ.

Khả năng mở rộng: Polars có thể mở rộng trên nhiều máy tính và sử dụng tài nguyên của một hệ thống phân tán, giúp xử lý dữ liệu lớn vượt ra ngoài khả năng của một máy tính đơn lẻ.

* + 1. Sử dụng thư viện DuckDB

Ý tưởng chính:

* Định nghĩa 4 cột đầu vào đều là VARCHAR để DuckDB đọc file gốc.
* Tạo một file tạm để lưu dữ liệu đã được DuckDB xử lý.
* DuckDB đọc file ratings.dat với delimiter “::”.
* Chuyển đổi các trường sang đúng kiểu (userid và movieid sang INTEGER, rating sang FLOAT).
* Xuất dữ liệu sạch ra file CSV tạm.
* Dùng copy\_expert để nạp dữ liệu từ CSV vào PostgreSQL.
* Xóa file tạm sau khi hoàn tất.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm loadratings() sử dụng DuckDB – 1

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm loadratings() sử dụng DuckDB – 2



1. – Thời gian thực thi hàm loadratings() sử dụng DuckDB

Giải thích vì sao nhanh hơn dùng phương pháp tối ưu không thư viện:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Tối ưu không thư viện | DuckDB |
| Tốc độ xử lý file lớn | Chậm hơn (Python phải đọc từng dòng, xử lý từng dòng, ghi buffer) | Rất nhanh (DuckDB tối ưu đọc file, chuyển đổi, lọc lỗi, xuất CSV cực nhanh) |
| Xử lý lỗi dữ liệu | Python phải tự kiểm tra từng dòng, nếu không sẽ lỗi hoặc phải try/except từng dòng | DuckDB có thể bỏ qua dòng lỗi (*ignore\_errors*=True), không raise exception, không dừng toàn bộ |
| Chuyển đổi kiểu dữ liệu | Không có - xử lý SQL trực tiếp khi đọc file | DuckDB chuyển đổi trực tiếp trong SQL (CAST), rất nhanh và an toàn |
| Đa luồng | Không | Có - DuckDB tự động tận dụng nhiều core CPU |

1. So sánh phương pháp tối ưu không dùng thư viện với dùng DuckDB
   * 1. Sử dụng thư viện Polars

Ý tưởng chính:

* Đọc từng dòng file thành một cột duy nhất (line).
* Dùng Polars để tách từng dòng thành các trường userid, movieid, rating (bỏ qua timestamp).
* Chuyển đổi kiểu dữ liệu ngay trong Polars (cast sang int/float).
* Ghi dữ liệu đã xử lý ra file CSV tạm.
* Tạo bảng PostgreSQL nếu chưa có, tối ưu bằng fillfactor=100.
* COPY toàn bộ dữ liệu vào PostgreSQL bằng lệnh copy\_expert.
* Xóa file tạm sau khi hoàn tất.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm loadratings() sử dụng Polars



1. – Thời gian thực thi hàm loadratings() sử dụng Polars

Giải thích vì sau nhanh hơn dùng DuckDB:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | DuckDB | Polars |
| **Đọc file** | Đọc file qua SQL, phải parse query | Đọc từng dòng, tách trường trực tiếp, vectorized |
| **Chuyển đổi kiểu dữ liệu** | CAST trong SQL, phải parse và thực thi query | .cast() trên DataFrame, vector hóa, rất nhanh |
| **Tận dụng CPU/SIMD/đa luồng** | Có, nhưng phải qua lớp SQL, có thể không tối ưu bằng Polars | Có, tối ưu hóa nội tại, thao tác vector hóa, đa luồng |
| **Overhead SQL** | Có (phải parse query, build execution plan) | Không có |

1. So sánh dùng DuckDB với dùng Polars

Sau khi chạy xong thì tất cả các cách trên đều cho ra kết quả pass.

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table ratings sau khi thực hiện hàm loadratings()
   1. Hàm rangepartition()

Solution mẫu:

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm rangepartition() mẫu



1. – Thời gian thực thi hàm rangepartition() mẫu

Ý tưởng chính:

* Tính khoảng phân vùng (delta) bằng vì rating nằm trong khoảng.
* Với mỗi phân vùng, tạo một bảng mới và sao chép các dòng phù hợp với khoảng rating tương ứng vào bảng đó.
* Phân vùng đầu tiên (i == 0) bao gồm cả ranh giới dưới (>=) và trên (<=), các phân vùng sau chỉ bao gồm ranh giới trên.
  + 1. Sử dụng UNLOGGED TABLE

Ý tưởng chính:

* Tính khoảng giá trị (delta) theo số lượng phân mảnh.
* Lặp qua các khoảng và với mỗi phần:
  + Tạo bảng con range\_part{i} không ghi log (UNLOGGED) để tăng tốc.
  + Chọn các dòng có rating thuộc khoảng giá trị tương ứng.
* Tối ưu hiệu suất với các thiết lập PostgreSQL: synchronous\_commit, work\_mem, maintenance\_work\_mem.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm rangepartition() dùng unlogged table



1. – Thời gian thực thi hàm rangepartition() dùng unlogged table

Giải thích vì sao nhanh hơn dùng solution mẫu:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Solution mẫu | Dùng unlogged table |
| Cách tạo bảng | Dùng CREATE UNLOGGED TABLE AS SELECT nên vừa tạo vừa chèn dữ liệu ngay | Dùng CREATE TABLE rồi INSERT INTO để chèn dữ liệu sau |
| Tối ưu hiệu suất | Có sử dụng: SET synchronous\_commit, work\_mem, maintenance\_work\_mem nên giảm I/O, tăng tốc RAM | Không có tối ưu hệ thống PostgreSQL |
| Tạo bảng ghi log? | Không ghi log (dùng UNLOGGED TABLE) nên nhanh hơn nhiều lần | Ghi log đầy đủ (mặc định) nên tốn thời gian ghi vào WAL |
| Số lần truy vấn SQL | 1 truy vấn cho mỗi bảng (CREATE UNLOGGED TABLE AS SELECT) | 2 truy vấn cho mỗi bảng (CREATE TABLE + INSERT INTO) |
| Khả năng mở rộng | Tốt hơn, xử lý tốt lượng dữ liệu lớn nhờ giảm overhead | Chậm hơn khi số lượng bản ghi lớn hoặc nhiều phân vùng |

1. So sánh solution mẫu với dùng unlogged table hàm rangepartition()

Tốc độ đã cải thiện rất nhiều. Tuy nhiên, việc dùng UNLOGGED TABLE làm cho việc bảo toàn dữ liệu mất đi:

Nếu PostgreSQL bị tắt đột ngột, crash thì toàn bộ dữ liệu trong UNLOGGED TABLE sẽ bị mất.

Nên vì vậy đây chỉ là cách tham khảo để cải thiện về mặt tốc độ.

* + 1. Tối ưu truy vấn không sử dụng UNLOGGED TABLE

Ý tưởng chính:

* Tính khoảng giá trị (delta) theo số lượng phân mảnh.
* Lặp qua các khoảng và với mỗi phần:
  + Tạo bảng con range\_part{i}.
  + Chọn các dòng có rating thuộc khoảng giá trị tương ứng.
* Tối ưu hiệu suất với các thiết lập PostgreSQL: synchronous\_commit, work\_mem, maintenance\_work\_mem.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm rangepartition() không dùng unlogged table



1. – Thời gian thực thi hàm rangepartition() không dùng unlogged table

Thời gian rõ ràng là chậm hơn. Tuy nhiên, cách này giúp đảm bảo toàn vẹn dữ liệu. Đây là điều cần thiết.

Bên cạnh đó để hàm rangeinsert() có thể chạy nhanh hơn gấp 10 đến 100 lần (điều này sẽ được nói ở phần 3.9) thì cần thêm hai hàm init\_range\_metadata\_table() và hàm update\_range\_metadata() để tạo bảng metadata cho hàm rangeinsert() lấy số phân mảnh một cách nhanh hơn.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm rangepartition() không dùng unlogged table và có bảng metadata



1. – Thời gian thực thi hàm rangepartition() không dùng unlogged table và có bảng metadata

Thời gian tăng lên 0.02 - 0.1s tùy theo từng lần chạy. Đánh đổi với việc thời gian insert tăng lên 10 đến 100 lần là chấp nhận được.

Sau khi chạy xong thì tất cả các cách trên đều cho ra kết quả pass.

* 1. Hàm roundrobinpartition()

Solution mẫu:

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm roundrobinpartition() mẫu



1. – Thời gian thực thi hàm roundrobinpartition() mẫu

Ý tưởng chính:

* Tạo numberofpartitions bảng mới với tên rrobin\_part0, rrobin\_part1, ..., rrobin\_partN.
* Gán chỉ số thứ tự dòng (row number) cho mỗi dòng trong bảng gốc bằng ROW\_NUMBER() OVER (). (Không có ORDER BY nên mặc định là theo thứ tự đọc vật lý.).
* Với mỗi bảng con thứ i, chèn các dòng sao cho (row\_number - 1) % numberofpartitions == i. Điều này đảm bảo các dòng được phân phối tuần tự vào các bảng: dòng 1 vào bảng 0, dòng 2 vào bảng 1, ..., quay vòng lại bảng 0.
* Mỗi bảng sẽ nhận được số dòng gần như bằng nhau, bất kể giá trị rating.
  + 1. Sử dụng UNLOGGED TABLE

Ý tưởng chính:

* Tính chỉ số phân vùng bằng cách sử dụng ROW\_NUMBER() để đánh số thứ tự dòng, rồi lấy phần dư khi chia cho numberofpartitions để xác định phân vùng tương ứng (partition\_id).
* Tạo bảng tạm rrobin\_temp dạng UNLOGGED để lưu thông tin đã đánh số và phân vùng, nhằm tránh lặp lại phép tính trong mỗi vòng lặp và tăng tốc độ thực thi.
* Lặp qua từng phân vùng:
  + Tạo bảng con range\_part{i} không ghi log (UNLOGGED) để tăng tốc.
  + Sao chép các dòng từ rrobin\_temp có partition\_id = i vào bảng tương ứng.
* Tối ưu hiệu suất với các thiết lập PostgreSQL: synchronous\_commit, work\_mem, maintenance\_work\_mem.

A screenshot of a computer code

AI-generated content may be incorrect.

1. – Hàm roundrobinpartition() dùng unlogged table



1. – Thời gian thực thi hàm roundrobinpartition() dùng unlogged table

Giải thích vì sao nhanh hơn dùng solution mẫu:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Solution mẫu | Dùng unlogged table |
| Tạo bảng tạm | Không có bảng tạm | Tạo 1 bảng tạm UNLOGGED chứa kết quả ROW\_NUMBER + partition\_id |
| Tối ưu hiệu suất | Có sử dụng: SET synchronous\_commit, work\_mem, maintenance\_work\_mem nên giảm I/O, tăng tốc RAM | Không có tối ưu hệ thống PostgreSQL |
| Tạo bảng phân vùng | Tạo từng bảng bằng CREATE UNLOGGED TABLE ... AS SELECT ... một lần duy nhất | Tạo bảng bằng CREATE TABLE, sau đó INSERT ... SELECT ... riêng lẻ |
| Tổng số truy vấn | 1 truy vấn lớn để chuẩn bị dữ liệu, sau đó chia tách | N truy vấn INSERT riêng biệt với cùng cấu trúc |

1. So sánh solution mẫu với dùng unlogged table hàm rrobinpartition()

Tốc độ đã cải thiện rất nhiều. Tuy nhiên, việc dùng UNLOGGED TABLE làm cho việc bảo toàn dữ liệu mất đi:

Nếu PostgreSQL bị tắt đột ngột, crash thì toàn bộ dữ liệu trong UNLOGGED TABLE sẽ bị mất.

Nên vì vậy đây chỉ là cách tham khảo để cải thiện về mặt tốc độ.

* + 1. Tối ưu truy vấn không sử dụng UNLOGGED TABLE

Ý tưởng chính:

* Ý tưởng tương tự cách trên, chỉ là không sử dụng UNLOGGED TABLE và thêm xử lý ngoại lệ.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm roundrobinpartition() không dùng unlogged table



1. – Thời gian thực thi hàm roundrobinpartition() không dùng unlogged table

Thời gian rõ ràng là chậm hơn. Tuy nhiên, cách này giúp đảm bảo toàn vẹn dữ liệu. Đây là điều cần thiết.

Bên cạnh đó để hàm roundroubininsert() có thể chạy nhanh hơn gấp 10 đến 300 lần (điều này sẽ được nói ở phần 3.10) thì cần thêm hai hàm init\_range\_metadata\_table() và hàm update\_range\_metadata() để tạo bảng metadata cho hàm rangeinsert() lấy số phân mảnh một cách nhanh hơn.

A screenshot of a computer program

AI-generated content may be incorrect.

1. – Hàm roundrobinpartition() không dùng unlogged table và có bảng metadata



1. – Thời gian thực thi hàm roundrobinpartition() không dùng unlogged table và có bảng metadata

Thời gian tăng lên 0.01-0.1s tùy theo từng lần chạy. Đánh đổi với việc thời gian insert tăng lên 10 đến 300 lần là chấp nhận được.

* 1. Hàm rangeinsert()

Solution mẫu:

A white background with text

AI-generated content may be incorrect.

1. – Hàm rangeinsert() mẫu



1. – Thời gian thực thi hàm rangeinsert() mẫu

Ý tưởng chính:

* Đếm số lượng phân mảnh range (numberofpartitions) thông qua hàm count\_partitions().
* Tính khoảng giá trị delta ứng với mỗi phân mảnh bằng 5.0 / numberofpartitions.
* Xác định phân mảnh (bảng con) phù hợp để chèn dữ liệu index = int (rating / delta) (với rating là giá trị đánh giá của bản ghi mới được thêm vào).
  + Nếu rating chia hết cho delta và index 0 thì cần điều chỉnh lại index: index = index – 1.
* Sau đó insert bản ghi mới vào trong phân mảnh tương ướng.

Ý tưởng trên đã giải quyết được bài toán khi thêm 1 bản ghi mới vào trong phân mảnh thích hợp. Tuy nhiên việc đếm số lượng phân mảnh thông qua hàm count\_partitions() vẫn chưa tối ưu làm cho thời gian thực hiện của hàm rangeinsert() quá lâu. Với 1 bản ghi mất 0.014s để thực hiện. Như vậy với 1 triệu bản ghi sẽ mất khoảng 14.000s tương ứng với khoảng gần 4 giờ. Để giảm thời gian thực hiện hàm rangeinsert, có thể tạo thêm 1 bảng metadata để lưu lại số lượng phân mảnh. Khi cần chỉ cần truy vấn từ bảng này ra mà không cần phải thực hiện hàm count\_partition().

A screenshot of a computer code

AI-generated content may be incorrect.

1. – Hàm rangeinsert() khi bảng metadata.



1. – Thời gian thực thi hàm rangeinsert() khi sử dụng bảng metadata.

Kết quả bên trên đang làm tròn đến 8 chữ số sau dấu phẩy, điều này có thể cho thấy khi sử dụng bảng metadata để thì thời gian thực thi hàm rangeinsert() đã giảm đi gấp nhiều lần.

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_metadata sau khi thực hiện hàm rangeinsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_part0 sau khi thực hiện hàm rangeinsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_part1 sau khi thực hiện hàm rangeinsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_part2 sau khi thực hiện hàm rangeinsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_part3 sau khi thực hiện hàm rangeinsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_part4 sau khi thực hiện hàm rangeinsert()

Nhận được số lượng bản ghi là:

479168 + 908584 + 2726855 + 3755614 + 2129834 = 10000055

Điều này đúng vì sau khi thêm 1 bản ghi thì số lượng tổng của 5 bảng phân mảnh là 10000054 + 1 = 10000055.

* 1. Hàm roundrobininsert()

Solution mẫu:

A screenshot of a computer

AI-generated content may be incorrect.

1. – Hàm roundrobininsert() mẫu



1. – Thời gian thực thi hàm roundrobininsert() mẫu

Ý tưởng chính:

* Thêm bản ghi mới vào trong bảng chính.
* Lấy ra tổng số lượng dòng trong bảng chính (total\_rows).
* Đếm số lượng phân mảnh range (numberofpartitions) thông qua hàm count\_partitions().
* Xác định phân mảnh (bảng con) phù hợp để chèn dữ liệu index = (total\_rows - 1) % numberofpartitions
* Sau đó insert bản ghi mới vào trong phân mảnh tương ướng.

Ý tưởng trên đã giải quyết được bài toán khi thêm 1 bản ghi mới vào trong phân mảnh thích hợp. Tuy nhiên việc đếm số lượng phân mảnh thông qua hàm count\_partitions() vẫn chưa tối ưu làm cho thời gian thực hiện của hàm roundrobininsert() quá lâu. Với 1 bản ghi mất 0.317s để thực hiện. Như vậy với 1 triệu bản ghi sẽ mất khoảng 317.000s tương ứng với khoảng gần 88 giờ. Để giảm thời gian thực hiện hàm roundrobininsert(), có thể tạo thêm 1 bảng metadata để lưu lại số lượng phân mảnh cùng với chỉ số của phân mảnh cuối cùng vừa được thêm dữ liệu. Khi cần số lượng phân mảnh chỉ cần truy vấn từ bảng này ra mà không cần phải thực hiện hàm count\_partition(), và việc xác định bản ghi mới thuộc phân mảnh nào cũng sẽ không cần phải thông quả tổng số bản ghi trong bảng chính mà chỉnh cần dựa vào chỉ số của phân mảnh cuối cùng vừa được thêm dữ liệu (điều này cũng được lưu trong bảng metadata).

A screenshot of a computer

AI-generated content may be incorrect.

1. – Hàm roundrobininsert() khi bảng metadata.



1. – Thời gian thực thi hàm roundrobininsert() khi sử dụng bảng metadata.

Kết quả trên cho thấy khi sử dụng bảng metadata tốc độ xử lý của hàm roundrobininsert() có thể nhanh hơn cách bên trên đến khoảng 300 lần. Với việc insert 1 bản ghi đơn lẻ sẽ không thấy rõ ưu điểm, nhưng với số lượng bản ghi càng lớn thì ưu điểm của cách này sẽ càng rõ ràng hơn.

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table range\_metadata sau khi thực hiện hàm roundrobininsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table rrobin\_part0 sau khi thực hiện hàm roundrobininsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table rrobin\_part1 sau khi thực hiện hàm roundrobininsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table rrobin\_part2 sau khi thực hiện hàm roundrobininsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table rrobin\_part3 sau khi thực hiện hàm roundrobininsert()

A screenshot of a computer

AI-generated content may be incorrect.

1. – Table rrobin\_part4 sau khi thực hiện hàm roundrobininsert()

Nhận được số lượng bản ghi là:

2000011 + 2000011 + 2000011 + 2000011 + 2000011 = 10000055

Điều này đúng vì sau khi thêm 1 bản ghi thì số lượng tổng của 5 bảng phân mảnh là 10000054 + 1 = 10000055. Và vì 10000055 là một số chia hết cho 5 nên cả 5 phân mảnh có số lượng bằng nhau.

* 1. Kết chương

Chương này mô tả việc triển khai, cài đặt, phân tích và cải thiện tốc độ cho các 5 hàm cần thực hiện theo yêu cầu đề bài. Hàm loadratings() ghi nhận tốc độ tối ưu là 6.0697 giây, hàm rangepartition() ghi nhận tốc độ tối ưu là 9.756473 giây, hàm roundrobinpartition() ghi nhận tốc độ tối ưu là 24.048627 giây và hai hàm insert đã giảm đi rất nhiều thời gian.

KẾT LUẬN

**Các kết quả đạt được**

Nhóm thực hiện đề tài “MÔ TẢ PHÂN MẢNH DỮ LIỆU” đã thực hiện tìm hiểu, xây dựng, mô phỏng, tối ưu phân mảnh ngang dữ liệu theo hai cách. Các hàm đã cải thiện nhiều về mặt tốc độ. Tuy nhiên tốc độ chạy ở mỗi lần có thể khác nhau nhưng không đáng kể.:

**Hướng phát triển**

Đề tài này có thể được mở rộng theo các hướng sau:

* Tối ưu ETL:Kết hợp Polars/DuckDB cho ETL nhanh,nhưng có thể thử thêm các công cụ như Pandas, Dask, hoặc Spark cho dữ liệu cực lớn…
* Tối ưu cấu hình PostgreSQL (RAM, work\_mem, shared\_buffers, WAL, v.v.)
* Dùng multiprocessing hoặc multi-threading cho các thao tác phân mảnh, ETL, hoặc import dữ liệu.
* Tạo index phù hợp trên các bảng phân mảnh để tăng tốc truy vấn.
* Web UI cho phép upload file, theo dõi tiến trình import, xem trạng thái phân mảnh, kiểm tra dữ liệu.
* Kết nối với các mô hình machine learning để huấn luyện/training trực tiếp trên dữ liệu phân mảnh.

TÀI LIỆU THAM KHẢO

1. Sử dụng Polars và DuckDB để tối ưu hàm loadratings(), <https://www.youtube.com/watch?v=utTaPW32gKY> .
2. Dùng bảng UNLOGGED TABLE, <https://www.postgresql.org/docs/current/sql-createtable.html> .
3. StringIO, <https://www.geeksforgeeks.org/stringio-module-in-python/> .
4. Thư viện psycopg2, <https://www.psycopg.org/docs/> .
5. Giảm độ trễ cho Postgre, <https://www.postgresql.org/docs/current/wal-async-commit.html> .
6. Cấu hình Postgre, <https://dangxuanduy.com/database/cac-tham-so-cau-hinh-memory-trong-postgresql/> .
7. Polars Dataframe, <https://docs.pola.rs/api/python/stable/reference/dataframe/index.html> .
8. DuckDB, <https://duckdb.org/docs/stable/clients/python/dbapi.html> .