ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU (CO3021)

Báo cáo Bài tập lớn
Postgres và MongoDB

Giáo viên hướng dẫn:	LÊ THỊ BẢO THU	
Các thành viên nhóm	Đào Duy Tùng	2033364
	Huỳnh Huynh Mân	2033364
	Vũ Trường Khoa	2252867
	Tôn Trọng Tín	2033338

Tp. Hồ Chí Minh, Tháng 03/2025

Mục lục

Giới thiệu

- Mục tiêu:
  - So sánh sự khác biệt giữa Postgres (CSDL quan hệ) và MongoDB (CSDL NoSQL dạng document) theo các khía cạnh: lưu trữ & quản lý dữ liệu, indexing, xử lý truy vấn, giao dịch, kiểm soát đồng thời, sao lưu & phục hồi dữ liệu.
  - Minh họa bằng các ví dụ code cụ thể cho từng loại hình cũng như nêu ra các loại (types) hay phương pháp mà mỗi hệ thống hỗ trợ.
- Lý do chọn đề tài:
  - Giúp hiểu rõ ưu, nhược điểm của hai hệ thống để lựa chọn phù hợp với yêu cầu ứng dụng thực tế.
  - Cung cấp cơ sở cho việc xây dựng ứng dụng demo, ví dụ: ứng dụng thương mại điện tử đơn giản trên MongoDB thể hiện các khía cạnh đã nêu.

Trong quá trình hiện thực và báo cáo, nếu có vấn đề sai sót, nhóm rất mong nhận được ý kiến đóng góp từ thầy và các bạn để hoàn thiện hơn.

1 Giới thiệu tổng quan

1.1 Các khái niệm cơ bản

1.1.1 Cơ sở dữ liệu quan hệ (Relational Database)

Khái niệm

Là hệ thống quản lý cơ sở dữ liệu dựa trên mô hình bảng, trong đó dữ liệu được lưu trữ dưới dạng các bảng có các hàng (record) và cột (field). Các bảng có thể liên kết với nhau thông qua các mối quan hệ (relationship) như khóa chính (primary key) và khóa ngoại (foreign key).

Đặc điểm

- **Schema cố định**: Cần định nghĩa cấu trúc dữ liệu trước khi lưu trữ.
- **Ngôn ngữ truy vấn**: Sử dụng SQL (Structured Query Language) để thao tác dữ liệu.
- **Tính nhất quán cao**: Hỗ trợ giao dịch theo chuẩn ACID (Atomicity, Consistency, Isolation, Durability).
- **Ví dụ**: PostgreSQL, MySQL, Oracle.

1.1.2 Cơ sở dữ liệu NoSQL dạng document (Document-oriented NoSQL Database)

Khái niệm

Là hệ thống quản lý cơ sở dữ liệu NoSQL, lưu trữ dữ liệu dưới dạng các tài liệu (documents) thường ở định dạng JSON, BSON hoặc XML. Mỗi tài liệu có thể chứa các cấu trúc dữ liệu phức tạp và không cần phải tuân theo một schema cố định.

Đặc điểm

- **Schema linh hoạt**: Cho phép lưu trữ dữ liệu với cấu trúc khác nhau trong cùng một collection.
- **Khả năng mở rộng**: Dễ dàng mở rộng theo chiều ngang (scaling out) để xử lý khối lượng dữ liệu lớn.
- **Truy vấn qua API hoặc ngôn ngữ truy vấn riêng**: Sử dụng cú pháp truy vấn đặc thù như MongoDB Query Language (MQL).
- **Ví dụ**: MongoDB, CouchDB.

1.2 PostgreSQL là gì?

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở mạnh mẽ, nổi tiếng với tính tuân thủ chuẩn SQL, tính năng mở rộng cao và hỗ trợ đầy đủ các giao dịch ACID. Nó được thiết kế để xử lý khối lượng công việc phức tạp, hỗ trợ nhiều kiểu dữ liệu (bao gồm JSON, XML) và cung cấp nhiều tính năng tiên tiến như MVCC (Multi-Version Concurrency Control), stored procedures, trigger, và hỗ trợ lập trình mở rộng qua các ngôn ngữ như PL/pgSQL, Python,...

1.2.1 Khi nào nên dùng PostgreSQL?

- **Ứng dụng đòi hỏi tính toàn vẹn và nhất quán cao**: Khi ứng dụng cần đảm bảo các giao dịch tuân thủ chuẩn ACID (Atomicity, Consistency, Isolation, Durability), chẳng hạn như các hệ thống tài chính, ngân hàng, hay quản lý đơn hàng.
- **Yêu cầu xử lý truy vấn phức tạp**: PostgreSQL rất mạnh mẽ với các truy vấn liên quan đến join, subquery, aggregate... Điều này phù hợp với các hệ thống phân tích, báo cáo và truy vấn dữ liệu phức tạp.
- **Định nghĩa schema rõ ràng**: Khi dữ liệu có cấu trúc xác định và cần quản lý bằng các ràng buộc như khóa chính, khóa ngoại, unique, check constraints để đảm bảo tính hợp lệ của dữ liệu.
- **Tính mở rộng theo chiều dọc**: Ứng dụng cần tận dụng tài nguyên của một máy chủ mạnh (CPU, RAM, ổ đĩa) để xử lý khối lượng lớn giao dịch.
- **Hỗ trợ đa dạng kiểu dữ liệu và mở rộng**: Khi cần lưu trữ các kiểu dữ liệu phức tạp như JSON hoặc các kiểu dữ liệu địa lý, PostgreSQL cung cấp các module như PostGIS cho dữ liệu không gian.

1.2.2 Khi nào không nên dùng PostgreSQL?

- **Ứng dụng yêu cầu mở rộng theo chiều ngang cực lớn**: Nếu hệ thống cần mở rộng qua nhiều máy chủ để xử lý khối lượng dữ liệu khổng lồ (big data) với khả năng phân mảnh (sharding) tự động, một số cơ sở dữ liệu NoSQL hoặc hệ thống NewSQL có thể phù hợp hơn.
- **Yêu cầu hiệu năng cao cho các tác vụ đơn giản, không cần giao dịch phức tạp**: Trong các ứng dụng chỉ cần xử lý các thao tác đọc/ghi đơn giản, với yêu cầu linh hoạt về schema (ví dụ: lưu trữ log, dữ liệu không cấu trúc) thì các hệ thống như MongoDB hoặc các cơ sở dữ liệu key-value có thể là lựa chọn tối ưu hơn.
- **Độ phức tạp quản trị**: Mặc dù PostgreSQL rất mạnh mẽ, nhưng với các ứng dụng nhỏ hoặc không đòi hỏi các tính năng nâng cao, cấu hình và quản trị của PostgreSQL có thể phức tạp hơn so với một số hệ thống nhẹ hơn.

PostgreSQL là lựa chọn tuyệt vời cho các ứng dụng yêu cầu tính nhất quán, xử lý giao dịch phức tạp và truy vấn dữ liệu mạnh mẽ. Tuy nhiên, nếu hệ thống cần mở rộng theo chiều ngang quy mô lớn hoặc không cần

nhiều tính năng phức tạp, các lựa chọn khác như NoSQL hoặc cơ sở dữ liệu key-value có thể là giải pháp hợp lý hơn.

1.3 MongoDB là gì?

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL dạng document, nghĩa là nó lưu trữ dữ liệu dưới dạng các tài liệu (documents) – thường ở định dạng JSON hoặc BSON – trong các collection. Điều này mang lại sự linh hoạt cao vì không cần phải định nghĩa một schema cứng nhắc trước khi lưu trữ dữ liệu.

1.3.1 Khi nào nên dùng MongoDB?

- **Ứng dụng cần tính linh hoạt về dữ liệu**: Khi dữ liệu có thể thay đổi cấu trúc theo thời gian hoặc khi dữ liệu có tính chất phi cấu trúc. Ví dụ: các ứng dụng nội dung, blog, hoặc lưu trữ log, nơi mỗi bản ghi có thể có các thuộc tính khác nhau.
- **Khả năng mở rộng theo chiều ngang**: MongoDB được thiết kế để dễ dàng phân mảnh (sharding) và mở rộng qua nhiều máy chủ, phù hợp với các ứng dụng cần xử lý khối lượng dữ liệu lớn hoặc cần tăng tốc độ xử lý bằng cách phân phối tải.
- **Phát triển nhanh và linh hoạt**: Trong các dự án yêu cầu phát triển nhanh, không muốn ràng buộc dữ liệu với schema cứng nhắc, MongoDB cho phép thay đổi cấu trúc dữ liệu dễ dàng trong quá trình phát triển.
- **Ứng dụng thời gian thực**: Các ứng dụng cần hiệu năng ghi/đọc cao, ví dụ như các dịch vụ web, hệ thống theo dõi (monitoring) hay trò chuyện trực tuyến, có thể hưởng lợi từ kiến trúc NoSQL của MongoDB.

1.3.2 Khi nào không nên dùng MongoDB?

- **Yêu cầu giao dịch phức tạp và tính nhất quán cao**: Nếu ứng dụng cần thực hiện các giao dịch phức tạp với tính toàn vẹn dữ liệu nghiêm ngặt (ACID) như trong hệ thống tài chính hoặc ngân hàng, các cơ sở dữ liệu quan hệ như PostgreSQL thường phù hợp hơn.
- **Các phép join và truy vấn liên quan đến dữ liệu có mối quan hệ chặt chẽ**: MongoDB không hỗ trợ join giữa các collection một cách tự nhiên như SQL. Nếu dữ liệu có nhiều mối liên kết phức tạp cần join, thì hệ thống CSDL quan hệ sẽ có lợi thế hơn.
- **Yêu cầu xử lý truy vấn phức tạp**: Khi ứng dụng cần thực hiện các truy vấn phân tích và tổng hợp phức tạp, các hệ thống RDBMS thường có optimizer mạnh mẽ và cú pháp SQL hỗ trợ tốt hơn.

MongoDB là một giải pháp tuyệt vời cho các ứng dụng cần sự linh hoạt trong lưu trữ dữ liệu, khả năng mở rộng theo chiều ngang và tốc độ xử lý nhanh cho các tác vụ ghi/đọc đơn giản. Tuy nhiên, nếu ứng dụng của bạn đòi hỏi tính toàn vẹn dữ liệu cao, giao dịch phức tạp, hoặc các phép toán liên quan đến nhiều mối quan hệ dữ liệu (join), một hệ quản trị cơ sở dữ liệu quan hệ (như PostgreSQL) có thể là lựa chọn phù hợp hơn.

2 Nội dung phân tích

2.1 Data Storage & Management

2.1.1 Postgres

- **Đặc điểm**:
  - Dữ liệu được lưu trữ trong các bảng với cấu trúc (schema) cố định.
  - Ràng buộc kiểu dữ liệu, khóa chính, khóa ngoại, ... giúp đảm bảo tính nhất quán.
- **Ví dụ Code (SQL)**:

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);
```

2.1.2 MongoDB

- **Đặc điểm**:
  - Dữ liệu được lưu dưới dạng document (JSON/BSON) trong các collection.
  - Schema linh hoạt, dễ dàng mở rộng với dữ liệu phi cấu trúc.
- **Ví dụ Code (Mongo Shell)**:

```javascript
db.users.insertOne({
    name: "John Doe",
    email: "john.doe@example.com"
});
```

2.1.3 So sánh

- **Schema cố định so sánh linh hoạt**: **Postgres** yêu cầu định nghĩa cấu trúc dữ liệu trước, còn **MongoDB** cho phép lưu trữ dữ liệu không đồng nhất.
- **Quản lý dữ liệu**: Các ràng buộc (constraints) mạnh mẽ của **Postgres** hỗ trợ tính toàn vẹn dữ liệu, trong khi **MongoDB** cung cấp tính mở rộng dễ dàng.

2.2 Indexing

Indexing là một cơ chế tối ưu hóa trong cơ sở dữ liệu, hoạt động như "mục lục" của một cuốn sách, giúp hệ thống định vị và truy xuất dữ liệu một cách nhanh chóng.

2.2.1 Mục đích sử dụng Indexing

- **Tăng tốc độ truy vấn**: Index giúp hệ thống tìm kiếm các bản ghi mà không cần quét toàn bộ bảng (full table scan). Điều này đặc biệt quan trọng với các bảng chứa hàng triệu bản ghi.
- **Tối ưu hóa hiệu suất**: Khi truy vấn có điều kiện (WHERE, JOIN, ORDER BY, GROUP BY), index cho phép truy vấn truy cập dữ liệu theo cách đã được sắp xếp và tối ưu, giảm tải cho CPU và I/O.

2.2.2 Khi nào nên sử dụng Indexing

- **Bảng dữ liệu lớn**: Với bảng có số lượng bản ghi lớn, index giúp cải thiện thời gian truy vấn một cách đáng kể.
- **Truy vấn có điều kiện thường xuyên**: Nếu các truy vấn của bạn thường tìm kiếm theo một hoặc vài cột cụ thể, việc tạo index trên những cột này sẽ tăng tốc độ tìm kiếm.
- **Truy vấn sắp xếp và nhóm dữ liệu**: Các câu lệnh sử dụng ORDER BY hoặc GROUP BY có thể được tối ưu khi có index phù hợp.

2.2.3 Khi nào không nên sử dụng Indexing

- **Bảng dữ liệu nhỏ**: Với bảng chứa ít bản ghi, lợi ích của việc sử dụng index không đáng kể vì chi phí duy trì index có thể vượt quá lợi ích cải thiện tốc độ truy vấn.
- **Thao tác ghi dữ liệu nhiều**: Index không chỉ cải thiện truy vấn mà còn cần được cập nhật mỗi khi có INSERT, UPDATE hoặc DELETE. Trong các bảng có tần suất ghi/chỉnh sửa cao, nhiều index có thể làm chậm hiệu năng ghi dữ liệu.
- **Quá nhiều index trên một bảng**: Việc tạo quá nhiều index có thể dẫn đến việc quản lý và cập nhật index trở nên nặng nề, làm giảm hiệu suất của các thao tác ghi.

Tóm lại, **index** là công cụ mạnh mẽ để tối ưu hóa truy vấn, nhưng cần được áp dụng một cách hợp lý để tránh tác động tiêu cực đến hiệu năng ghi dữ liệu.

2.2.4 Postgres

- Các loại index hỗ trợ:
  - **B-tree**: Mặc định cho hầu hết các loại truy vấn.
  - **Hash, GiST, GIN, SP-GiST**: Dùng cho các trường hợp đặc thù như tìm kiếm văn bản, dữ liệu không gian, ...
- Ví dụ Code (SQL):

```sql
-- Tạo B-tree index trên cột email
CREATE INDEX idx_users_email ON users(email);
```

2.2.5 MongoDB

- Các loại index hỗ trợ:
  - **Single Field Index**: Index trên một trường.
  - **Compound Index**: Index trên nhiều trường.
  - **Text Index**: Dành cho tìm kiếm văn bản.
  - **Geospatial Index**: Dành cho dữ liệu không gian.
- Ví dụ Code (Mongo Shell):

```
// Tạo index trên trường email
db.users.createIndex({ email: 1 });
```

2.2.6 So sánh

- **Đa dạng index**: Postgres có nhiều loại index phục vụ cho các mục đích tối ưu hóa phức tạp, trong khi MongoDB tập trung vào các loại index hỗ trợ nhanh cho việc tìm kiếm theo document.
- **Cách triển khai**: Cách định nghĩa index khá khác nhau giữa SQL và JSON-like query language.

2.3 Query Processing

Query Processing (Xử lý truy vấn) là quá trình chuyển đổi câu lệnh truy vấn do người dùng nhập (ví dụ: SQL trong các cơ sở dữ liệu quan hệ hay MQL trong MongoDB) thành một kế hoạch thực thi hiệu quả để truy xuất dữ liệu từ cơ sở dữ liệu.

Query processing (xử lý truy vấn) là một quy trình phức tạp trong hệ thống cơ sở dữ liệu, bao gồm các bước từ khi nhận truy vấn từ người dùng đến khi trả về kết quả. Quy trình này đóng vai trò then chốt, quyết định hiệu suất và độ tin cậy của hệ thống cơ sở dữ liệu.

Query processing là chuỗi các hoạt động xử lý truy vấn trong hệ thống cơ sở dữ liệu, bao gồm:

- Phân tích cú pháp truy vấn
- Kiểm tra tính hợp lệ
- Tối ưu hóa truy vấn
- Tạo kế hoạch thực thi
- Thực thi truy vấn
- Tổng hợp và trả về kết quả

Vai trò và Lợi ích của Query Processing

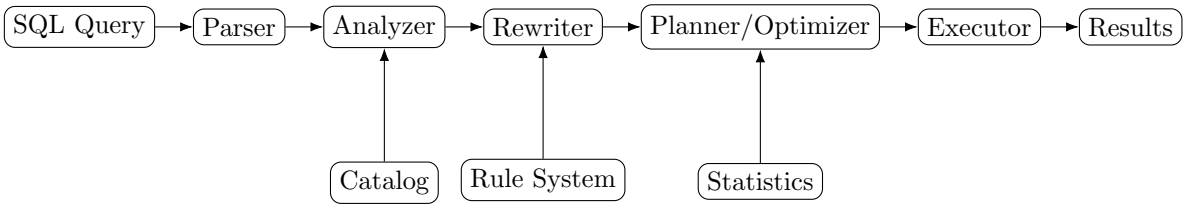
- Tăng hiệu năng truy xuất dữ liệu:
  - Giúp giảm thiểu thời gian truy vấn thông qua tối ưu hóa kế hoạch thực thi.
- Quản lý tài nguyên hiệu quả:
  - Giúp hệ thống sử dụng CPU, bộ nhớ và I/O một cách hợp lý, tránh việc quét toàn bộ bảng không cần thiết.
- Hỗ trợ truy vấn phức tạp:
  - Cho phép xử lý các truy vấn có nhiều phép toán như join, subquery, aggregate... một cách hiệu quả.

Query Processing đóng vai trò quan trọng trong việc đảm bảo rằng hệ thống cơ sở dữ liệu thực thi các truy vấn một cách nhanh chóng và hiệu quả, góp phần nâng cao hiệu năng tổng thể của ứng dụng.

2.3.1 Postgres

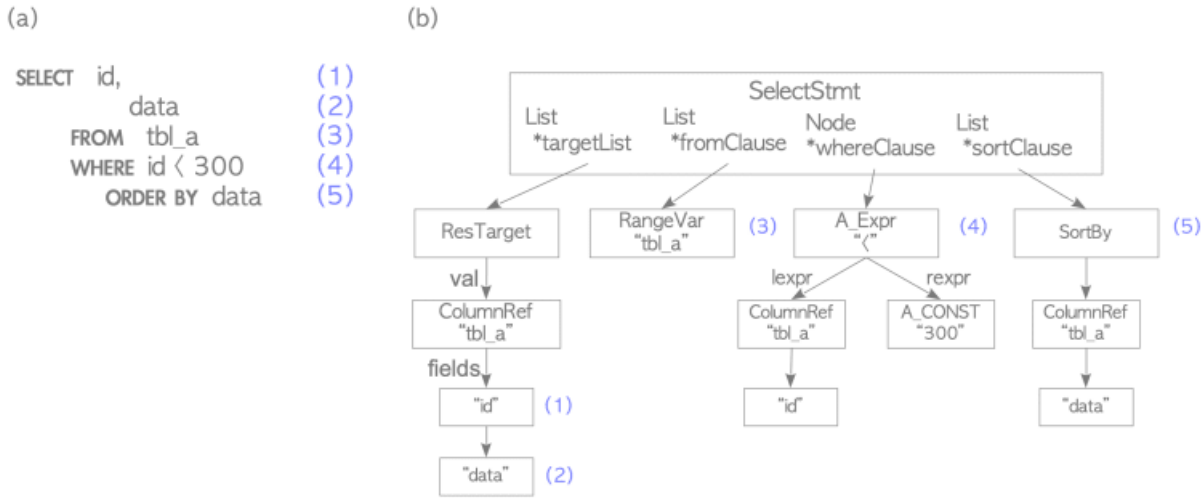
PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở với kiến trúc xử lý truy vấn phức tạp và hiệu quả.

Dưới đây là mô hình hoạt động của query processing trong PostgreSQL:



Quá trình này bao gồm một số bước chính:

1. **Parser (Bộ phân tích cú pháp)**:
   - Phân tích câu truy vấn để kiểm tra tính hợp lệ về cú pháp và đảm bảo rằng các từ khóa, tên bảng, tên trường, ... được viết đúng theo quy tắc của ngôn ngữ truy vấn.
   - Trình phân tích cú pháp truy vấn phân tích cấu trúc cú pháp của truy vấn, đảm bảo nó tuân thủ các quy tắc ngữ pháp của ngôn ngữ PostgreSQL. Nó kiểm tra lỗi, xác thực cấu trúc truy vấn và tạo cây phân tích cú pháp. Đầu ra là một cây phân tích cú pháp :



Hình 1: *Ví dụ cây phân tích cú pháp*

2. **Semantic Analysis**:
   - Sau khi phân tích cú pháp thành công, trình phân tích ngữ nghĩa (semantic analyzer) sẽ kiểm tra ý nghĩa của câu truy vấn. Nó xác minh sự tồn tại của các bảng, cột, hàm và các đối tượng cơ sở dữ liệu khác được tham chiếu trong truy vấn. Nó cũng kiểm tra quyền truy cập của người dùng để đảm bảo họ có quyền thực hiện truy vấn.
   - Kiểm tra tính hợp lệ của truy vấn

- Xác minh rằng các bảng, cột và hàm được tham chiếu tồn tại
- Phân giải các tên cột và kiểm tra quyền truy cập
- Chuyển đổi parse tree thành query tree
- Xác định kiểu dữ liệu cho mỗi biểu thức

3. **Query Rewriting (Chuyển đổi truy vấn)**:

- Giai đoạn này có thể xảy ra nếu có các quy tắc (rules) hoặc view được định nghĩa. Trình viết lại (rewriter) có thể sửa đổi câu truy vấn ban đầu dựa trên các quy tắc này để tối ưu hóa hoặc thực hiện các logic phức tạp.
- Biến đổi câu truy vấn ban đầu thành dạng dễ tối ưu hóa hơn, chẳng hạn như đơn giản hóa các biểu thức logic hoặc chuyển đổi cấu trúc của câu truy vấn.
- Áp dụng các quy tắc từ hệ thống rules
- Mở rộng view thành các truy vấn trên bảng cơ sở
- Áp dụng các ràng buộc bảo mật (Row-Level Security)
- Xử lý các trigger và policy

4. **Planner/Optimizer (Bộ lập kế hoạch/tối ưu hóa)**:

- Đây là giai đoạn quan trọng nhất. Trình lập kế hoạch (planner) sẽ tạo ra nhiều kế hoạch thực thi khác nhau cho cùng một câu truy vấn. Mỗi kế hoạch này bao gồm các bước cụ thể để truy cập và xử lý dữ liệu, chẳng hạn như sử dụng index, thực hiện sequential scan, hoặc chọn thuật toán join phù hợp.
- PostgreSQL sẽ tạo kế hoạch thực thi (execution plan) bằng cách chọn phương án tối ưu nhất dựa trên các chỉ mục và thống kê dữ liệu. Điều này bao gồm việc quyết định thứ tự thực thi các bước trong truy vấn.
- Xác định kế hoạch thực thi tối ưu bằng cách cân nhắc nhiều phương án khác nhau. Hệ quản trị cơ sở dữ liệu sẽ dựa vào các thống kê dữ liệu, cấu trúc bảng, chỉ mục, ... để lựa chọn chiến lược tốt nhất nhằm giảm thiểu thời gian và tài nguyên cần thiết cho truy vấn.
- Trình tối ưu hóa (optimizer) sẽ đánh giá chi phí ước tính của mỗi kế hoạch thực thi được tạo ra bởi trình lập kế hoạch. Chi phí này thường được tính toán dựa trên các yếu tố như thời gian thực thi dự kiến, số lượng I/O cần thiết và mức sử dụng tài nguyên hệ thống. Trình tối ưu hóa sẽ chọn kế hoạch có chi phí thấp nhất để thực thi.
- Phân tích nhiều kế hoạch thực thi có thể
- Ước tính chi phí cho mỗi kế hoạch dựa trên thống kê
- Lựa chọn kế hoạch có chi phí thấp nhất
- Xác định chiến lược quét (table scan, index scan)
- Quyết định phương pháp join (nested loop, hash join, merge join)

- Tối ưu hóa các phép lọc và sắp xếp

5. **Query Execution (Thực thi truy vấn)**:
   - Thực hiện kế hoạch truy vấn đã được tối ưu hóa, truy xuất dữ liệu từ các bảng hoặc collections theo cách hiệu quả nhất.
6. **Formatting (Định dạng kết quả)**:
   - Sau khi truy xuất dữ liệu, kết quả được định dạng lại và trả về cho người dùng hoặc ứng dụng theo yêu cầu.

Đặc điểm :

- Sử dụng SQL, hỗ trợ các truy vấn phức tạp như join, subquery, aggregate.
- Có query optimizer mạnh mẽ giúp tối ưu thời gian thực thi.

Ví dụ Code (SQL):

```sql
SELECT u.name, o.order_date
FROM users u
JOIN orders o ON u.id = o.user_id
WHERE u.email = 'john.doe@example.com';
```

Listing 1: Ví dụ query trong Postgres

Query Optimization trong PostgreSQL

Query optimization trong PostgreSQL là quá trình tự động hoặc thủ công để cải thiện hiệu suất của các câu truy vấn SQL. Mục tiêu là giảm thiểu thời gian thực thi và tài nguyên hệ thống cần thiết.

Có nhiều kỹ thuật để tối ưu hóa truy vấn trong PostgreSQL:

1. **Sử dụng Index**: Index là cấu trúc dữ liệu đặc biệt giúp PostgreSQL tìm kiếm các hàng cụ thể trong bảng một cách nhanh chóng. Tạo index trên các cột thường xuyên được sử dụng trong mệnh đề **WHERE**, **JOIN**, **ORDER BY**, và **GROUP BY**.
   - **Index Optimization**: Sử dụng B-tree, Hash, GIN, GiST.
     - **B-tree indexes**: Chỉ mục chuẩn cho các phép so sánh =, <, >, <=, >=:

```sql
CREATE INDEX idx\_employee\_name ON employees(name);
```

Listing 2: Ví dụ sử dụng B-tree indexes trong Postgres

     - **Hash indexes** : Hiệu quả cho phép so sánh bằng.

```sql
CREATE INDEX idx_employee_id_hash ON employees USING HASH (employee_id);
```

Listing 3: Ví dụ sử dụng Hash indexes trong Postgres

  - **GiST indexes**: Cho dữ liệu không chuẩn (hình học, văn bản)

```sql
CREATE INDEX idx_location ON stores USING GIST (location);
```

Listing 4: Ví dụ sử dụng Hash indexes trong Postgres

  - **GIN indexes**: Cho dữ liệu có nhiều giá trị trong một trường (arrays, jsonb)

```sql
CREATE INDEX idx_tags ON products USING GIN (tags);
```

Listing 5: Ví dụ sử dụng GIN indexes trong Postgres

  - **BRIN indexes**: Cho dữ liệu có tổ chức tự nhiên (timestamps)

```sql
CREATE INDEX idx_timestamp ON logs USING BRIN (created_at);
```

Listing 6: Ví dụ sử dụng BRIN indexes trong Postgres

  - **Partial indexes**: Chỉ đánh chỉ mục cho một phần dữ liệu

```sql
CREATE INDEX idx_active_users ON users(email) WHERE status = 'active';
```

Listing 7: Ví dụ sử dụng Partial indexes trong Postgres

  - **Multi-column indexes**: Đánh chỉ mục nhiều cột

```sql
CREATE INDEX idx_name_dept ON employees(department_id, last_name, first_name);
```

Listing 8: Ví dụ sử dụng Multi-column indexes trong Postgres

- **Join Ordering**: Chọn thứ tự join tối ưu.
  - Chọn thứ tự join tối ưu
  - Lựa chọn phương pháp join phù hợp (nested loop, hash, merge)
  - Sử dụng join column indexes
  - Tận dụng foreign keys
- **Parallel Query**: Thực thi song song.
- **WHERE Clause Optimization**
  - Sử dụng điều kiện có thể tận dụng các chỉ mục
  - Tránh sử dụng các hàm trên cột được lọc
  - Tận dụng phép lọc có độ chọn lọc cao
  - Sử dụng điều kiện IN thay vì nhiều điều kiện OR

```sql
-- Tốt
SELECT * FROM orders WHERE status IN ('pending', 'processing');

-- Kém hiệu quả
SELECT * FROM orders WHERE status = 'pending' OR status = 'processing';
```

Listing 9: Ví dụ tối ưu WHERE Clause Optimization trong Postgres

- **Partition Pruning**:

- Chia dữ liệu thành các phân vùng
- Chỉ quét các phân vùng liên quan

```
CREATE TABLE sales (
    id SERIAL,
    sale_date DATE,
    amount NUMERIC
) PARTITION BY RANGE (sale_date);

CREATE TABLE sales_2023 PARTITION OF sales
    FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');

CREATE TABLE sales_2024 PARTITION OF sales
    FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');
```

Listing 10: Ví dụ Partition Pruning trong Postgres

- **Materialized Views**: Lưu kết quả truy vấn thường dùng.

```
CREATE MATERIALIZED VIEW sales_summary AS
SELECT
    date_trunc('month', sale_date) AS month,
    product_id,
    SUM(quantity) AS total_quantity,
    SUM(amount) AS total_amount
FROM sales
GROUP BY date_trunc('month', sale_date), product_id;

REFRESH MATERIALIZED VIEW sales_summary;
```

Listing 11: Ví dụ Materialized Views trong Postgres

2. **Phân tích Kế hoạch Thực thi với *EXPLAIN***: Lệnh **EXPLAIN** cho phép bạn xem kế hoạch thực thi mà PostgreSQL sẽ sử dụng cho một câu truy vấn cụ thể. Điều này giúp bạn hiểu cách PostgreSQL truy cập dữ liệu và xác định các bước có thể được tối ưu hóa. Sử dụng **EXPLAIN_ANALYZE** để có thông tin chi tiết về thời gian thực tế của từng bước.

```
EXPLAIN SELECT * FROM orders WHERE order_date > '2024-01-01';
EXPLAIN ANALYZE SELECT * FROM employees
JOIN departments ON employees.department_id = departments.id
WHERE salary > 50000;


Nested Loop  (cost=0.29..34.38 rows=11 width=242) (actual time=0.038..0.078 rows=10 loops
    =1)
  ->  Index Scan using employees_department_id_idx on employees  (cost=0.29..16.79 rows
    =11 width=122) (actual time=0.020..0.033 rows=10 loops=1)
        Filter: (salary > 50000)
  ->  Index Scan using departments_pkey on departments  (cost=0.14..1.59 rows=1 width
    =120) (actual time=0.003..0.003 rows=1 loops=10)
        Index Cond: (id = employees.department_id)
```

```
Planning Time: 0.362 ms
Execution Time: 0.112 ms
```

Listing 12: Ví dụ sử dụng EXPLAIN trong Postgres

3. **Viết Truy vấn Hiệu quả**:
   - Đơn giản hóa biểu thức phức tạp
   - Loại bỏ các subqueries không cần thiết
   - Chuyển đổi HAVING thành WHERE khi có thể
   - Tối ưu hóa các phép UNION, INTERSECT và EXCEPT
   - **Chỉ chọn các cột cần thiết**: Sử dụng **SELECT column1, column2** thay vì **SELECT ***.
   - **Sử dụng mệnh đề *WHERE* hiệu quả**: Lọc dữ liệu càng sớm càng tốt.
   - **Tránh sử dụng các hàm trên các cột đã được index trong mệnh đề *WHERE***: Điều này có thể ngăn PostgreSQL sử dụng index. Thay vào đó, hãy xem xét tạo **functional index**.
   - **Tối ưu hóa các phép toán *JOIN***: Hiểu rõ các loại **JOIN** và sử dụng loại phù hợp. Đảm bảo các cột tham gia trong **JOIN** được index.
   - **Hạn chế sử dụng *DISTINCT***: Nếu có thể, hãy tìm cách khác để loại bỏ các bản ghi trùng lặp.
   - **Sử dụng *LIMIT***: Nếu bạn chỉ cần một số lượng bản ghi nhất định, hãy sử dụng **LIMIT**.
4. **Cập nhật Thống kê với *ANALYZE***: PostgreSQL sử dụng thống kê về dữ liệu để đưa ra các quyết định tối ưu hóa. Chạy lệnh **ANALYZE** định kỳ trên các bảng để cập nhật thống kê.

```
ANALYZE customers;
```

Listing 13: Ví dụ sử dụng ANALYZE trong Postgres

5. **pg_stat_statements** : Theo dõi thống kê thực thi truy vấn

```
SELECT query, calls, total_exec_time, rows, mean_exec_time
FROM pg_stat_statements
ORDER BY total_exec_time DESC
LIMIT 10;
```

Listing 14: Ví dụ theo dõi thống kê thực thi trong Postgres

6. **auto_explain** : Tự động ghi lại kế hoạch thực thi của các truy vấn chậm

```
LOAD 'auto_explain';
SET auto_explain.log_min_duration = '100ms';
SET auto_explain.log_analyze = true;
```

Listing 15: Ví dụ theo dõi truy vấn chậm trong Postgres

7. **Tối ưu hóa Cấu trúc Bảng**: Thiết kế bảng với các kiểu dữ liệu phù hợp và cân nhắc việc phân vùng bảng cho các bảng lớn.

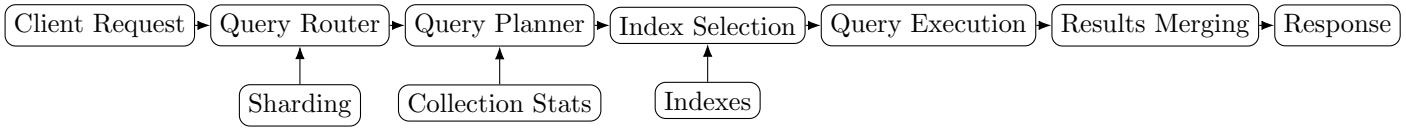
8. **Cấu hình PostgreSQL**: Điều chỉnh các tham số cấu hình như **shared_buffers**, **work_mem** có thể ảnh hưởng đến hiệu suất truy vấn.
    - shared_buffers: 25% của RAM hệ thống
    - effective_cache_size: 75% của RAM hệ thống
    - work_mem: 4MB-64MB tùy ứng dụng
    - maintenance_work_mem: 64MB-256MB
    - checkpoint_timeout: 15-30 phút
    - default_statistics_target: 100-1000

2.3.2 MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL, hướng tài liệu. Quá trình xử lý truy vấn trong MongoDB khác biệt so với PostgreSQL do sự khác biệt trong mô hình dữ liệu và ngôn ngữ truy vấn.

MongoDB xử lý truy vấn theo kiến trúc khác với cơ sở dữ liệu quan hệ do tính chất NoSQL và lưu trữ dữ liệu dạng document.

Mô hình Query Processing trong MongoDB



Đặc điểm :

- Sử dụng MongoDB Query Language (MQL) với cú pháp JSON-like.
- Hỗ trợ các phép toán như find, aggregate (aggregation pipeline) để xử lý dữ liệu.
- MongoDB sử dụng ngôn ngữ truy vấn dạng JSON.

Query Routing

- Trong môi trường phân tán (sharded cluster), MongoDB Router (mongos) xác định các shard chứa dữ liệu cần truy vấn
- Sử dụng metadata trong config servers để định tuyến truy vấn
- Tận dụng shard key để xác định chính xác shard chứa dữ liệu

Query Parsing và Planning

- Chuyển đổi truy vấn JSON thành cấu trúc nội bộ
- Phân tích các toán tử và điều kiện

- Xác định các chỉ mục có thể sử dụng
- Lựa chọn chiến lược quét (COLLSCAN, IXSCAN)

Index Selection

- Thực thi truy vấn trên các documents
- Áp dụng các giai đoạn xử lý (stages)
- Thực hiện các phép lọc, phép chiếu
- Tổng hợp kết quả từ các shard khác nhau trong môi trường phân tán

Query Execution

- Đánh giá các chỉ mục có thể sử dụng
- Lựa chọn chỉ mục tối ưu nhất
- Xác định các chỉ mục có thể sử dụng
- Tính toán hiệu quả của việc sử dụng chỉ mục

Cơ chế Query Processing của MongoDB

Query Stages Processing

MongoDB xử lý truy vấn thông qua các stages:

- **IXSCAN**: Quét chỉ mục
- **FETCH**: Lấy documents từ collection
- **SORT**: Sắp xếp kết quả
- **LIMIT**: Giới hạn số kết quả
- **PROJECTION**: Chọn lọc các trường
- **GROUP**: Nhóm và tính toán
- **MERGE**: Kết hợp kết quả từ các shard

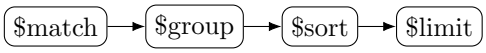
Pipeline Processing trong Aggregation Framework

MongoDB Aggregation Framework sử dụng mô hình pipeline:

```
db.sales.aggregate([
  { $match: { status: "completed" } },
  { $group: { _id: "$product", total: { $sum: "$amount" } } },
  { $sort: { total: -1 } },
  { $limit: 5 }
])
```

Listing 16: Ví dụ MongoDB Query

Trong đó mỗi stage xử lý đầu vào và chuyển kết quả cho stage tiếp theo:



Read Concern và Write Concern

- Read Concern: Xác định mức độ cô lập và tính nhất quán của dữ liệu đọc
  - local: Đọc dữ liệu mới nhất trên primary replica
  - majority: Đảm bảo dữ liệu đã được ghi nhận bởi đa số replica
  - linearizable: Đảm bảo đọc dữ liệu mới nhất, nhưng có độ trễ cao
- Write Concern: Xác định mức độ xác nhận khi ghi dữ liệu
  - w: 1 (chỉ primary), w: majority (đa số replica)
  - j: true/false (journal confirmation)
  - wtimeout: thời gian chờ xác nhận

Query Optimization trong MongoDB

1. **Chiến lược Query Optimization trong MongoDB**
   (a) **Index Optimization**
      i. **Single Field Index**: Đánh chỉ mục một trường

```
db.employees.createIndex({ last_name: 1 })
```

Listing 17: Ví dụ Single Field Index trong MongoDB

      ii. **Compound Index**: Đánh chỉ mục nhiều trường

```
db.employees.createIndex({ department: 1, hire_date: -1 })
```

Listing 18: Ví dụ Compound Index trong MongoDB

      iii. **Multikey Index**: Đánh chỉ mục các trường array

```
db.products.createIndex({ tags: 1 })
```

Listing 19: Ví dụ Multikey Index trong MongoDB

iv. **Text Index**: Đánh chỉ mục full-text

```
db.articles.createIndex({ content: "text" })
```

Listing 20: Ví dụ Text Index trong MongoDB

v. **Geospatial Index**: Đánh chỉ mục dữ liệu địa lý

```
db.stores.createIndex({ location: "2dsphere" })
```

Listing 21: Ví dụ Geospatial Index trong MongoDB

vi. **Partial Index**: Chỉ đánh chỉ mục một tập con documents

```
db.orders.createIndex(
{ order_date: 1 },
{ partialFilterExpression: { status: "active" } }
)
```

Listing 22: Ví dụ Partial Index trong MongoDB

vii. **TTL Index**: Chỉ mục có thời gian sống

```
db.sessions.createIndex(
{ last_updated: 1 },
{ expireAfterSeconds: 3600 }
)
```

Listing 23: Ví dụ TTL Index trong MongoDB

(b) **Query Structure Optimization**

i. **Covered Queries**: Truy vấn chỉ sử dụng các trường có trong chỉ mục

```
// Với chỉ mục { email: 1, name: 1 }
db.users.find(
{ email: "user@example.com" },
{ _id: 0, name: 1, email: 1 }
)
```

Listing 24: Ví dụ Covered Queries trong MongoDB

ii. **Tối ưu toán tử**: Sử dụng toán tử hiệu quả

```
// Thay vì
db.products.find({ price: { $gte: 10, $lte: 50 } })

// Có thể dùng
db.products.find({ price: { $elemMatch: { $gte: 10, $lte: 50 } } })
```

Listing 25: Ví dụ Tối ưu toán tử trong MongoDB

iii. **Projection**: Chỉ lấy các trường cần thiết

```
db.customers.find(
{ status: "active" },
{ name: 1, email: 1, _id: 0 }
```

```
)
```

Listing 26: Ví dụ Multikey Index trong MongoDB

2. **Sharding Optimization**

   - Chọn shard key phù hợp để phân phối dữ liệu đồng đều
   - Tối ưu hóa truy vấn để hạn chế truy vấn đa shard
   - Sử dụng truy vấn có chứa shard key
   - Tránh đồng bộ hóa đa shard kết quả (scatter-gather)

Công cụ phân tích và tối ưu hóa trong MongoDB

1. **explain()** : Phân tích kế hoạch thực thi truy vấn:

```
db.employees.find({ department: "Engineering" }).explain("executionStats")
```

Listing 27: Ví dụ explain() trong MongoDB

Kết quả giải thích có thể bao gồm:

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "company.employees",
    "winningPlan": {
      "stage": "FETCH",
      "inputStage": {
        "stage": "IXSCAN",
        "keyPattern": { "department": 1 },
        "indexName": "department_1",
        "direction": "forward"
      }
    },
    "rejectedPlans": []
  },
  "executionStats": {
    "executionSuccess": true,
    "nReturned": 145,
    "executionTimeMillis": 5,
    "totalKeysExamined": 145,
    "totalDocsExamined": 145
  }
}
```

Listing 28: Kết quả explain() trong MongoDB

2. **MongoDB Profiler**: Bật profiler để theo dõi các truy vấn chậm

```
db.setProfilingLevel(1, { slowms: 100 })
db.getProfilingStatus()
db.system.profile.find().sort({ ts: -1 }).limit(10)
```

Listing 29: Ví dụ Bật profiler để theo dõi các truy vấn chậm trong MongoDB

3. **Database Profiler** và **Monitoring Tools**
   - **MongoDB Compass**: Giao diện đồ họa để phân tích hiệu suất
   - **MongoDB Atlas**: Công cụ theo dõi hiệu suất tích hợp
   - **MongoDB Cloud Manager**: Giám sát và quản lý cơ sở dữ liệu
4. **serverStatus()** và **dbStats()** : Thu thập thông tin về hiệu suất hệ thống và cơ sở dữ liệu

```
db.serverStatus()
db.stats()
```

Listing 30: Ví dụ serverStatus() và dbStats() trong MongoDB

Các kỹ thuật tối ưu hóa trong MongoDB

1. **Schema Design Optimization**
   (a) **Embedding vs. Referencing**: Cân nhắc giữa lưu trữ các tài liệu liên quan trong một document (embedded) hoặc tham chiếu chúng bằng ID

```
// Embedded - tất cả thông tin trong một document
{
  "_id": ObjectId(),
  "name": "John Doe",
  "addresses": [
    { "street": "123 Main St", "city": "New York" },
    { "street": "456 Elm St", "city": "Boston" }
  ]
}

// Referencing - tách thành nhiều collection
// users collection
{ "_id": ObjectId("user123"), "name": "John Doe" }

// addresses collection
{ "_id": ObjectId(), "user_id": ObjectId("user123"), "street": "123 Main St" }
{ "_id": ObjectId(), "user_id": ObjectId("user123"), "street": "456 Elm St" }
```

Listing 31: Ví dụ Embedding vs. Referencing trong MongoDB

   (b) **Normalization** vs. **Denormalization**: Cân bằng giữa trùng lặp dữ liệu và hiệu suất truy vấn
   (c) **Field Names**: Sử dụng tên trường ngắn gọn để tiết kiệm không gian
2. **Working Set Optimization**

- Đảm bảo working set (dữ liệu được truy cập thường xuyên) vừa với RAM
- Tránh truy cập ngẫu nhiên vào nhiều document
- Sử dụng projection để giảm kích thước working set

3. **Bulk Operations** : Sử dụng các thao tác hàng loạt để giảm overhead

```
const bulkOp = db.products.initializeUnorderedBulkOp();
bulkOp.find({ category: "electronics" }).update({ $set: { onSale: true } });
bulkOp.find({ inventory: { $lt: 5 } }).update({ $set: { status: "low_stock" } });
bulkOp.execute();
```

Listing 32: Ví dụ Bulk Operations trong MongoDB

4. **Read Preference Optimization** : Cấu hình read preference phù hợp

```
db.collection.find().readPref("secondaryPreferred")
```

Listing 33: Ví dụ Read Preference Optimization trong MongoDB

5. **Write Concern Optimization** : Cân bằng giữa hiệu suất và độ tin cậy

```
db.collection.insertOne(
  { document },
  { writeConcern: { w: "majority", j: true } }
)
```

Listing 34: Ví dụ Write Concern Optimization trong MongoDB

2.3.3 So sánh

- **Ngôn ngữ truy vấn**: SQL truyền thống của Postgres đối lập với MQL của MongoDB.
- **Khả năng join**: Postgres hỗ trợ join trực tiếp qua SQL, trong khi MongoDB sử dụng aggregation pipeline để thực hiện join giữa các collection.

Khía cạnh	**PostgreSQL**	**MongoDB**
Mô hình dữ liệu	Quan hệ (tables, rows, columns)	NoSQL (collections, documents)
Ngôn ngữ truy vấn	SQL (chuẩn hóa)	MongoDB Query Language (JSON-like)
Cấu trúc truy vấn	Khai báo (declarative)	Khai báo và thao tác (document-oriented)
Tối ưu hóa	Cost-based optimizer	Index-based optimizer
Join processing	Hỗ trợ nhiều loại join (nested loop, hash, merge)	Thực hiện qua $*lookup* trong aggregation
Transaction	ACID đầy đủ	ACID từ phiên bản 4.0
Distributed queries	Foreign Data Wrappers	Native sharding

Bảng 1: So sánh phương pháp Query Processing giữa PostgreSQL và MongoDB

Khía cạnh	PostgreSQL	MongoDB
Đọc đơn giản	Hiệu quả với chỉ mục tốt	Rất hiệu quả với chỉ mục tốt
Ghi đơn giản	Kiểm tra ràng buộc cần thời gian	Rất nhanh
Truy vấn phức tạp	Hiệu quả với joins và subqueries	Cần sử dụng aggregation pipeline
Phân tích dữ liệu	Native support cho window functions, CTE	Aggregation framework
Scaling	Vertical scaling chủ yếu	Horizontal scaling với sharding

Bảng 2: So sánh Performance Considerations giữa PostgreSQL và MongoDB

Khía cạnh	**PostgreSQL**	**MongoDB**
Indexing strategy	Multiple specialized index types	Fewer index types but flexible
Statistics	Detailed statistics in pg_statistics	Basic statistics on collections
Plan visualization	EXPLAIN with detailed costs	explain() with execution stages
Join optimization	Sophisticated join order selection	Limited join capabilities with $lookup

Bảng 3: So sánh Query Optimization Differences giữa PostgreSQL và MongoDB

2.4 Transaction

Transaction (giao dịch) trong cơ sở dữ liệu là một đơn vị công việc gồm một hoặc nhiều thao tác được thực hiện như một khối duy nhất. Điều này có nghĩa là hoặc toàn bộ các thao tác trong giao dịch đều thành công, hoặc nếu có bất kỳ thao tác nào thất bại thì toàn bộ giao dịch sẽ bị hủy bỏ (rollback), giữ cho dữ liệu luôn ở trạng thái nhất quán.

Các Tính Chất ACID của Transaction

- **Atomicity (Tính nguyên tử)**: Toàn bộ các thao tác trong giao dịch được thực hiện thành một khối; nếu một thao tác thất bại, toàn bộ giao dịch sẽ bị rollback.
- **Consistency (Tính nhất quán)**: Giao dịch luôn đưa cơ sở dữ liệu từ trạng thái hợp lệ này sang trạng thái hợp lệ khác, tuân thủ các ràng buộc và luật lệ của dữ liệu.
- **Isolation (Tính cô lập)**: Các giao dịch đồng thời sẽ không ảnh hưởng lẫn nhau; mỗi giao dịch hoạt động như thể nó là giao dịch duy nhất trên hệ thống.
- **Durability (Tính bền vững)**: Sau khi giao dịch được commit, các thay đổi sẽ được lưu vĩnh viễn, ngay cả trong trường hợp hệ thống gặp sự cố.

Transaction giúp đảm bảo rằng một loạt các thao tác trên cơ sở dữ liệu được thực hiện một cách an toàn và nhất quán, giúp bảo vệ tính toàn vẹn của dữ liệu ngay cả khi có lỗi xảy ra trong quá trình thực hiện. Điều này là đặc biệt quan trọng trong các ứng dụng yêu cầu giao dịch phức tạp, như hệ thống tài chính, ngân hàng, và các ứng dụng thương mại điện tử.

2.4.1 Posgres

Đặc điểm :

- Hỗ trợ giao dịch đầy đủ theo tiêu chuẩn ACID, phù hợp với các hệ thống yêu cầu tính nhất quán cao.

Ví dụ Code (SQL):

```sql
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

2.4.2 MongoDB

Đặc điểm :

- Trước phiên bản 4.0: chỉ hỗ trợ giao dịch trên single document.
- Từ phiên bản 4.0: hỗ trợ multi-document transactions (với một số hạn chế về hiệu năng và thiết kế).

Ví dụ Code (Mongo Shell):

```
const session = client.startSession();
session.startTransaction();
try {
    await db.collection('accounts').updateOne(
        { _id: 1 },
        { $inc: { balance: -100 } },
        { session }
    );
    await db.collection('accounts').updateOne(
        { _id: 2 },
        { $inc: { balance: 100 } },
        { session }
    );
    await session.commitTransaction();
} catch (error) {
    await session.abortTransaction();
} finally {
    session.endSession();
}

```

2.4.3 So sánh

Độ tin cậy giao dịch: Postgres luôn đảm bảo ACID cho mọi giao dịch, trong khi MongoDB chỉ hỗ trợ giao dịch đa tài liệu từ phiên bản mới, cần chú ý các giới hạn về hiệu năng.

2.5 Concurrency Control

Concurrency Control (Kiểm soát đồng thời) là cơ chế quản lý việc truy cập và thay đổi dữ liệu trong cơ sở dữ liệu khi có nhiều giao dịch (transactions) hoặc truy vấn diễn ra đồng thời. Mục đích chính của nó là đảm bảo rằng các thao tác đồng thời không gây ra xung đột, mất mát hoặc làm hỏng tính nhất quán của dữ liệu. Điều này được thực hiện thông qua các kỹ thuật và chiến lược như:

- **Locking (Khóa dữ liệu)**: Cho phép một giao dịch khóa dữ liệu trong khi thực hiện các thay đổi, ngăn các giao dịch khác can thiệp vào dữ liệu đó trong thời gian khóa.
- **Multi-Version Concurrency Control (MVCC)**: Cung cấp các phiên bản khác nhau của dữ liệu cho các giao dịch đọc và ghi, cho phép đọc dữ liệu mà không bị ảnh hưởng bởi các giao dịch ghi đang diễn ra. PostgreSQL sử dụng MVCC để giảm thiểu việc khóa dữ liệu.
- **Optimistic Concurrency Control**: Giả định rằng các giao dịch không xung đột và kiểm tra sự mâu thuẫn chỉ khi giao dịch kết thúc, thích hợp với môi trường có ít xung đột.
- **Timestamp Ordering**: Gán thời gian cho các giao dịch và sắp xếp thực hiện dựa trên thứ tự thời gian nhằm đảm bảo tính nhất quán.

Khi nào cần chú ý đến Concurrency Control?

- **Hệ thống có nhiều người dùng**: Khi có nhiều giao dịch truy cập và cập nhật dữ liệu đồng thời, cần đảm bảo dữ liệu không bị xung đột.
- **Giao dịch yêu cầu tính nhất quán cao**: Các hệ thống tài chính, ngân hàng hay thương mại điện tử yêu cầu tính toàn vẹn và nhất quán của dữ liệu, nên cần áp dụng cơ chế kiểm soát đồng thời chặt chẽ.
- **Tránh hiện tượng "dirty read", "lost update", "non-repeatable read"**: Concurrency Control giúp ngăn chặn các lỗi do truy cập dữ liệu không đồng bộ.

Tóm lại, Concurrency Control là một thành phần quan trọng trong cơ sở dữ liệu, đảm bảo rằng ngay cả khi nhiều giao dịch diễn ra cùng lúc, dữ liệu luôn được bảo vệ và duy trì tính nhất quán.

2.5.1 Postgres

- **Đặc điểm**:
  - Sử dụng MVCC (Multi-Version Concurrency Control) cho phép thực hiện nhiều giao dịch song song mà không gây blocking (không khóa toàn bộ bảng).
- **Cách quản lý**:
  - Người dùng không cần can thiệp trực tiếp, hệ thống tự động quản lý phiên bản dữ liệu.

Ví dụ Code (SQL):

Trong PostgreSQL (sử dụng MVCC):

```
BEGIN;
-- Giao dịch A đọc dữ liệu mà không ảnh hưởng đến giao dịch khác
SELECT * FROM orders WHERE id = 1;
-- Trong khi đó, giao dịch B có thể thay đổi dữ liệu của hàng đó, và PostgreSQL sẽ tạo ra một phiên bản riêng cho mỗi giao dịch.
COMMIT;
```

2.5.2 MongoDB

- **Đặc điểm**:
  - Sử dụng cơ chế locking ở cấp độ document; mỗi thao tác cập nhật trên document được thực hiện atomically (nguyên tử).
- **Cách quản lý**:
  - Trong MongoDB, thao tác cập nhật trên một document đã đảm bảo tính nguyên tử nên không cần cấu hình thêm.

Ví dụ Code (Mongo Shell):

```
// Cập nhật đơn giản, đảm bảo atomicity trên document
db.accounts.updateOne({ _id: 1 }, { $inc: { balance: -100 } });
```

2.5.3 So sánh

Quy mô lock: Postgres có thể quản lý lock ở mức bảng hoặc dòng thông qua MVCC, còn MongoDB chủ yếu tập trung vào lock ở cấp document, phù hợp với các ứng dụng yêu cầu tốc độ cao và cập nhật không đồng bộ.

2.6 Data Backup and Recovery

Data Backup and Recovery (Sao lưu và phục hồi dữ liệu) là quá trình bảo vệ dữ liệu của hệ thống thông qua hai hoạt động chính:

- **Data Backup (Sao lưu dữ liệu)**: Là quá trình tạo ra các bản sao lưu của dữ liệu từ hệ thống hiện tại và lưu trữ chúng ở một nơi an toàn (có thể là thiết bị lưu trữ khác, đám mây, ...). Mục tiêu là bảo vệ dữ liệu trước các rủi ro như lỗi phần cứng, sự cố hệ thống, tấn công mạng, hoặc lỗi người dùng.
- **Data Recovery (Phục hồi dữ liệu)**: Là quá trình khôi phục lại dữ liệu từ các bản sao lưu khi dữ liệu gốc bị mất, hỏng hoặc bị xâm phạm. Quá trình này đảm bảo rằng hệ thống có thể trở lại trạng thái hoạt động bình thường một cách nhanh chóng sau sự cố.

Vai Trò và Tầm Quan Trọng

- **Đảm bảo tính liên tục của kinh doanh**: Trong trường hợp xảy ra sự cố, việc có các bản sao lưu giúp hệ thống phục hồi nhanh chóng, giảm thiểu thời gian gián đoạn.
- **Bảo vệ dữ liệu quan trọng**: Các bản sao lưu đảm bảo rằng dữ liệu quan trọng không bị mất vĩnh viễn do lỗi hệ thống, tấn công mạng hoặc thảm họa tự nhiên.
- **Hỗ trợ kiểm thử và phát triển**: Ngoài việc phục vụ khôi phục dữ liệu, các bản sao lưu còn có thể được sử dụng để tạo môi trường kiểm thử (testing environment) cho việc phát triển và kiểm tra hệ thống.

Như vậy, **Data Backup and Recovery** là một phần thiết yếu trong quản lý hệ thống dữ liệu, giúp bảo vệ dữ liệu khỏi mất mát và đảm bảo hệ thống luôn có thể phục hồi sau các sự cố không lường trước.

2.6.1 Postgres

Các công cụ & phương pháp:

- **pg_dump**: Dùng để xuất dữ liệu dưới dạng SQL script.
- **pg_basebackup**: Dùng cho backup toàn bộ hệ thống.
- **PITR (Point-in-Time Recovery)**: Cho phép phục hồi dữ liệu tới một thời điểm cụ thể.

Ví dụ Code (Command Line):

Sử dụng lệnh pg_dump để tạo bản sao lưu của cơ sở dữ liệu.

```
pg_dump -U username dbname > dbname_backup.sql
```

Sử dụng lệnh pg_restore hoặc chạy script SQL từ file backup để khôi phục dữ liệu.

```
pg_restore -U username -d dbname dbname_backup.sql
```

2.6.2 MongoDB

Các công cụ & phương pháp:

- **mongodump & mongorestore**: Dùng để sao lưu và phục hồi dữ liệu.
- **Replica Sets**: Cung cấp tính sẵn sàng cao và khả năng tự phục hồi tự động.

Ví dụ Code (Command Line):

Sử dụng công cụ mongodump để tạo bản sao lưu của dữ liệu.

```
mongodump --db dbname --out /backup/path
```

Sử dụng công cụ mongorestore để phục hồi dữ liệu từ bản sao lưu.

```
mongorestore --db dbname /backup/path/dbname
```

2.6.3 So sánh

Công cụ và quy trình: **Postgres** có nhiều lựa chọn cho việc backup và phục hồi với khả năng phục hồi theo thời gian (PITR), trong khi **MongoDB** sử dụng các công cụ riêng biệt và kiến trúc replica set để đảm bảo an toàn dữ liệu.

2.7 Kết luận

Sau quá trình phân tích và so sánh chi tiết giữa PostgreSQL và MongoDB theo các khía cạnh như lưu trữ & quản lý dữ liệu, indexing, xử lý truy vấn, giao dịch, kiểm soát đồng thời và sao lưu – phục hồi dữ liệu, chúng ta có thể rút ra một số nhận định tổng quát và đưa ra các đề xuất thực tiễn như sau:

2.7.1 Tổng Kết Những Điểm Mạnh và Hạn Chế

PostgreSQL

- **Ưu điểm**:
  - **Tính nhất quán và toàn vẹn dữ liệu**: Với cấu trúc schema cố định và các ràng buộc dữ liệu chặt chẽ (primary key, foreign key, unique, check), PostgreSQL đảm bảo rằng dữ liệu luôn ở trạng thái hợp lệ.
  - **Giao dịch mạnh mẽ**: Hỗ trợ đầy đủ các thuộc tính ACID, giúp thực hiện các giao dịch phức tạp mà không sợ mất mát hay sai lệch dữ liệu.
  - **Xử lý truy vấn phức tạp**: Cung cấp optimizer và công cụ phân tích truy vấn tiên tiến, rất phù hợp với các ứng dụng yêu cầu join, subquery và các phép toán phức tạp.
- **Nhược điểm**:
  - **Khả năng mở rộng theo chiều ngang hạn chế**: PostgreSQL thường được mở rộng theo chiều dọc (vertical scaling) trên một máy chủ mạnh, khiến việc mở rộng quy mô trên nhiều node trở nên phức tạp.
  - **Cấu hình phức tạp**: Đối với các ứng dụng có quy mô nhỏ hay yêu cầu thay đổi linh hoạt về dữ liệu, việc duy trì và cấu hình PostgreSQL có thể đòi hỏi nhiều nỗ lực quản trị.

MongoDB

- **Ưu điểm**:
  - **Linh hoạt và dễ mở rộng**: Với kiến trúc lưu trữ dạng document, MongoDB không yêu cầu schema cứng nhắc, giúp dễ dàng thích ứng với sự thay đổi của dữ liệu. Hơn nữa, khả năng phân mảnh (sharding) cho phép mở rộng theo chiều ngang một cách linh hoạt để xử lý khối lượng dữ liệu lớn.
  - **Hiệu năng ghi/đọc cao**: Đặc biệt phù hợp với các ứng dụng thời gian thực, nơi tốc độ xử lý và phản hồi là ưu tiên hàng đầu.
  - **Tích hợp các công cụ hỗ trợ sao lưu và phục hồi**: Thông qua replica sets và các công cụ như mongodump – mongorestore, MongoDB cho phép quản lý dữ liệu an toàn và khôi phục nhanh chóng sau sự cố.
- **Nhược điểm**:

- **Hạn chế trong giao dịch phức tạp**: Mặc dù từ phiên bản 4.0 trở đi đã hỗ trợ multi-document transactions, nhưng khả năng xử lý giao dịch phức tạp của MongoDB vẫn chưa thể sánh bằng các hệ quản trị cơ sở dữ liệu quan hệ truyền thống.
- **Thiếu tính nhất quán trong các truy vấn liên kết**: Việc xử lý các truy vấn liên quan đến join giữa các collection không trực tiếp và tự nhiên như SQL, khiến việc xử lý dữ liệu có mối quan hệ chặt chẽ trở nên khó khăn hơn.

2.7.2 Đề Xuất Lựa Chọn Theo Tính Chất Ứng Dụng

- **Ứng dụng yêu cầu độ tin cậy cao và giao dịch phức tạp (ví dụ: ngân hàng, tài chính, quản lý đơn hàng)**: PostgreSQL là lựa chọn ưu việt nhờ khả năng đảm bảo tính nhất quán và hỗ trợ giao dịch theo chuẩn ACID. Nếu dữ liệu cần quản lý chặt chẽ với các ràng buộc nghiêm ngặt, PostgreSQL sẽ cung cấp giải pháp an toàn và hiệu quả.
- **Ứng dụng cần tính linh hoạt và mở rộng nhanh (ví dụ: ứng dụng web, hệ thống quản lý nội dung, xử lý log, dữ liệu thời gian thực)**: MongoDB sẽ phù hợp hơn nhờ khả năng thay đổi cấu trúc dữ liệu, xử lý số lượng lớn giao dịch đọc/ghi và mở rộng theo chiều ngang. Đặc biệt, các ứng dụng cần tích hợp nhanh chóng, thích ứng với sự thay đổi dữ liệu mà không cần quá nhiều cấu hình phức tạp, MongoDB là lựa chọn tối ưu.

2.7.3 Kết Luận Chung

Trong bối cảnh hiện nay, việc lựa chọn hệ quản trị cơ sở dữ liệu phù hợp với nhu cầu của ứng dụng không chỉ dừng lại ở việc so sánh về mặt kỹ thuật mà còn phải cân nhắc đến các yếu tố như chi phí vận hành, khả năng mở rộng, tốc độ phản hồi và yêu cầu bảo trì. PostgreSQL và MongoDB mỗi bên đều có những ưu, nhược điểm riêng:

- **PostgreSQL** là một giải pháp ổn định, mạnh mẽ cho các ứng dụng đòi hỏi tính toàn vẹn dữ liệu và xử lý giao dịch phức tạp.
- **MongoDB** lại tỏ ra linh hoạt, dễ dàng mở rộng và phù hợp với các ứng dụng cần tốc độ xử lý cao và khả năng thích ứng nhanh với sự thay đổi của dữ liệu.

Quyết định lựa chọn hệ thống nào phụ thuộc vào đặc thù của ứng dụng và môi trường vận hành. Dù bạn chọn PostgreSQL hay MongoDB, việc hiểu rõ cơ chế hoạt động và các tính năng nổi bật của từng hệ thống sẽ giúp tối ưu hóa hiệu năng và đảm bảo tính ổn định của dự án. Đồng thời, thông qua ứng dụng demo so sánh, nhóm nghiên cứu có thể đánh giá một cách trực quan và thực tiễn được những khác biệt quan trọng giữa hai hệ thống, từ đó đưa ra quyết định đúng đắn cho từng dự án cụ thể.

Với những nhận định và đề xuất trên, kết luận này không chỉ tổng hợp lại các nội dung đã phân tích mà còn mở ra hướng đi cho các ứng dụng thực tiễn, giúp người phát triển hiểu rõ hơn về cách thức vận dụng các tính năng của PostgreSQL và MongoDB vào các bài toán cụ thể.

3 Tài liệu tham khảo

- https://github.com/
- https://google.com/
- https://www.w3schools.com/
- https://www.postgresql.org/
- https://www.mongodb.com/