VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE MANAGEMENT SYSTEM (CO3021)

Assignment

"E-COMMERCE APPLICATION USING POSTGRE VS MONGODB"

Instructor(s): Lê Thị Bảo Thu

Students: *Nguyễn Xuân Huy Hoàng - L01 - 2311070* (Leader)
Lê Đình Đức - L01 - 2310774
Bùi Hữu Lợi - L01 - 2311972
Nguyễn Văn Công Thành - L01 - 2313133
Nguyễn Trần Yến Nhi - L01 - 2312506

HO CHI MINH CITY, FEBRUARY 2026

Contents

List of Figures

List of Tables

Member list & Workload

No.	Fullname	Student ID	Problems	% done
1	Nguyễn Xuân Huy Hoàng	2311070	- Hoàn thành Concurrency Control - Làm Backend và Database	100%
2	Lê Đình Đức	2310774	- Hoàn thành Query Processing - Làm Backend và Database	100%
3	Bùi Hữu Lợi	2311972	- Hoàn thành Transaction - Làm Backend và Database	100%
4	Nguyễn Văn Công Thành	2313133	- Hoàn thành Indexing - Làm Frontend	100%
5	Nguyễn Trần Yến Nhi	2312506	- Hoàn thành Data Storage & Management - Làm Frontend	100%

Table 1: Member list & workload

1 GIỚI THIỆU

Trong kỷ nguyên công nghệ thông tin hiện đại, dữ liệu đóng vai trò cốt lõi trong việc vận hành và ra quyết định của mọi hệ thống phần mềm. Sự bùng nổ của Big Data và nhu cầu xử lý đa dạng các loại dữ liệu đã dẫn đến sự chuyển dịch từ việc thống trị tuyệt đối của các hệ quản trị cơ sở dữ liệu quan hệ sang sự ra đời và phát triển mạnh mẽ của các hệ quản trị cơ sở dữ liệu phi quan hệ (NoSQL).

Tuy nhiên, không có một giải pháp "vạn năng" cho mọi bài toán. Trong khi RDBMS (đại diện là **PostgreSQL**) nổi bật với tính nhất quán dữ liệu và tuân thủ chặt chẽ các thuộc tính ACID, thì NoSQL (đại diện là **MongoDB**) lại mang đến sự linh hoạt về lược đồ (Schema-less), khả năng mở rộng và tốc độ truy xuất vượt trội đối với dữ liệu phi cấu trúc.

Trong phạm vi bài tập lớn này, nhóm tập trung nghiên cứu, so sánh và đánh giá hiệu năng giữa hai đại diện tiêu biểu: **PostgreSQL** (SQL) và **MongoDB** (NoSQL). Sự so sánh được thực hiện dựa trên 5 khía cạnh kỹ thuật cốt lõi của một hệ quản trị cơ sở dữ liệu:

- Data Storage & Management (Lưu trữ và quản lý dữ liệu)
- Indexing (Chỉ mục)
- Query Processing (Xử lý truy vấn)
- Transaction (Giao dịch)
- Concurrency Control (Điều khiển đồng thời)

Để minh họa một cách thực tiễn sự khác biệt và khả năng bổ trợ lẫn nhau của hai hệ quản trị này, nhóm quyết định xây dựng một ứng dụng **Sàn Thương mại điện tử (E-commerce Platform)** với kiến trúc lai (Hybrid Architecture). Đây là miền ứng dụng đặc thù đòi hỏi sự kết hợp ưu điểm của cả hai loại CSDL:

- **PostgreSQL:** Chịu trách nhiệm quản lý các dữ liệu yêu cầu tính toàn vẹn cao, cấu trúc chặt chẽ và ràng buộc quan hệ phức tạp như: Thông tin người dùng (Users), Đơn hàng (Orders), Kho hàng (Inventory) và Thanh toán.
- **MongoDB:** Chịu trách nhiệm lưu trữ các dữ liệu có cấu trúc linh động, phân cấp hoặc khối lượng lớn như: Thông tin chi tiết sản phẩm (Products - với các thuộc tính đa dạng tùy loại hàng), Đánh giá (Reviews) và Log hệ thống.

Thông qua việc phân tích lý thuyết và thực nghiệm trên ứng dụng E-commerce, báo cáo này sẽ cung cấp cái nhìn sâu sắc về ưu nhược điểm của từng loại DBMS, từ đó đưa ra kết luận về ngữ cảnh sử dụng phù hợp nhằm tối ưu hóa hiệu năng hệ thống.

2 KIẾN TRÚC ỨNG DỤNG

2.1 Mô hình tổng quát

Hệ thống Sàn Thương mại điện tử được xây dựng dựa trên mô hình kiến trúc **Client-Server (3-tier architecture)**. Việc áp dụng mô hình này giúp tách biệt rõ ràng giữa giao diện người dùng (Presentation Layer), xử lý logic nghiệp vụ (Business Logic Layer) và lưu trữ dữ liệu (Data Access Layer), tạo điều kiện thuận lợi cho việc bảo trì và mở rộng hệ thống trong tương lai.

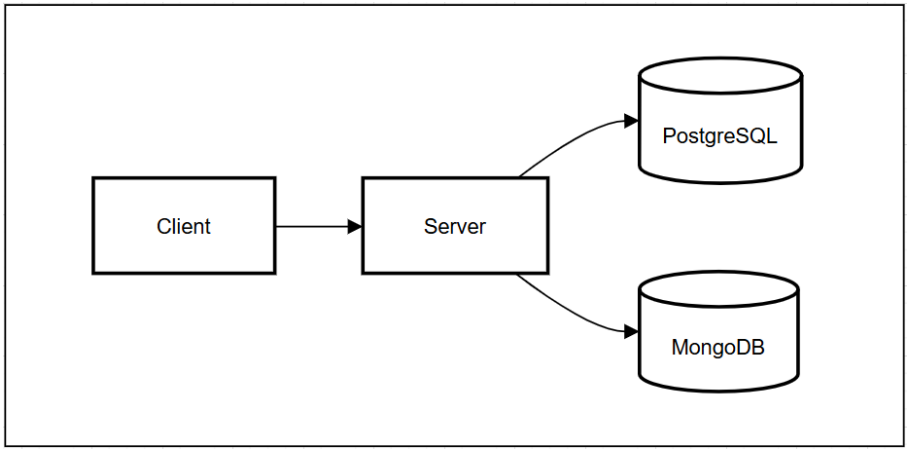

Figure 1: Sơ đồ kiến trúc tổng quan của hệ thống E-commerce

2.2 Các công nghệ sử dụng

Dựa trên các tiêu chuẩn phát triển web hiện đại và yêu cầu của đề tài, nhóm đề xuất bộ công nghệ (Tech stack) như sau:

- **Frontend:** Sử dụng thư viện **ReactJS** kết hợp với ngôn ngữ **TypeScript** để đảm bảo tính chặt chẽ, dễ kiểm soát lỗi của mã nguồn. Giao diện được thiết kế sử dụng **TailwindCSS** giúp tăng tốc độ phát triển và tối ưu trải nghiệm người dùng.
- **Backend:** Hệ thống Backend được xây dựng trên nền tảng **NodeJS** với framework **ExpressJS**. Đây là lựa chọn phù hợp cho kiến trúc hướng sự kiện (event-driven), xử lý tốt các tác vụ I/O non-blocking, đặc biệt hiệu quả cho các ứng dụng thương mại điện tử có lượng truy cập đồng thời lớn.
- **Database:** Áp dụng kiến trúc lai (Hybrid) với sự kết hợp giữa **PostgreSQL** và **MongoDB**.

2.3 Thiết kế Cơ sở dữ liệu lai (Hybrid Database Design)

Để tận dụng tối đa ưu điểm của cả hai dòng cơ sở dữ liệu SQL và NoSQL, nhóm áp dụng chiến lược **Polyglot Persistence** với sự phân chia trách nhiệm dữ liệu cụ thể như sau:

2.3.1 PostgreSQL (RDBMS - SQL)

PostgreSQL đóng vai trò là kho lưu trữ chính cho các dữ liệu quan trọng (Core Data), yêu cầu tính cấu trúc cao, toàn vẹn tham chiếu và tuân thủ chặt chẽ các thuộc tính ACID.

- **Users & Authentication:** Lưu trữ thông tin tài khoản, mật khẩu và phân quyền (Role-based access control).
- **Orders & Transactions:** Quản lý đơn hàng, chi tiết đơn hàng và lịch sử thanh toán. Đây là dữ liệu không được phép sai sót hay mất mát.
- **Inventory:** Quản lý tồn kho. Cơ chế Transaction và Locking của PostgreSQL giúp giải quyết triệt để bài toán cạnh tranh (Race condition) khi nhiều người dùng cùng đặt mua một sản phẩm cuối cùng.

2.3.2 MongoDB (NoSQL - Document Store)

MongoDB được sử dụng để lưu trữ các dữ liệu có cấu trúc linh động (Schema-less), yêu cầu tốc độ đọc ghi nhanh và khả năng mở rộng chiều ngang (Scaling out).

- **Products (Catalog):** Do tính chất đa dạng của các mặt hàng (ví dụ: Laptop có cấu hình CPU/RAM, Quần áo có Size/Màu sắc), việc sử dụng cấu trúc Document (JSON/BSON) cho phép lưu trữ linh hoạt mà không cần chuẩn hóa dữ liệu phức tạp như RDBMS.
- **Reviews & Comments:** Dữ liệu đánh giá thường có khối lượng lớn và phi cấu trúc. MongoDB cho phép truy xuất nhanh nội dung này để hiển thị mà không cần thực hiện các phép JOIN tốn kém.
- **Logs:** Lưu trữ lịch sử hoạt động, hành vi người dùng (User behavior) để phục vụ phân tích dữ liệu và gợi ý sản phẩm (Recommendation).

2.4 Phân tích và mô tả yêu cầu dữ liệu

2.4.1 Mô tả nền tảng

Hệ thống E-commerce là một nền tảng thương mại điện tử hiện đại được xây dựng theo kiến trúc kết hợp giữa cơ sở dữ liệu quan hệ PostgreSQL và cơ sở dữ liệu NoSQL MongoDB (Polyglot Persistence).

Hệ thống phục vụ hai nhóm người dùng chính: khách hàng (customer) và quản trị viên (admin).

Khách hàng có thể đăng ký tài khoản, quản lý thông tin cá nhân và nhiều địa chỉ giao hàng, tìm kiếm sản phẩm theo nhiều tiêu chí, xem chi tiết sản phẩm, thêm sản phẩm vào giỏ hàng, thực hiện đặt hàng và theo dõi trạng thái đơn hàng. Ngoài ra, khách hàng có thể đánh giá và nhận xét các sản phẩm đã mua.

Quản trị viên có thể quản lý danh mục sản phẩm, thêm, chỉnh sửa hoặc xóa sản phẩm, theo dõi tồn kho, quản lý đơn hàng và thống kê doanh thu.

Hệ thống đảm bảo:

- Tính toàn vẹn dữ liệu giao dịch thông qua PostgreSQL
- Tính linh hoạt và mở rộng dữ liệu sản phẩm thông qua MongoDB
- Khả năng xử lý đồng thời và bảo mật đăng nhập thông qua cơ chế token

Các chức năng chính bao gồm: quản lý người dùng, quản lý sản phẩm, tìm kiếm nâng cao, xử lý đơn hàng, thanh toán, quản lý tồn kho, đánh giá sản phẩm, ghi log hành vi người dùng và thống kê hệ thống.

2.4.2 Mô tả các kiểu thực thể, thuộc tính và mối liên kết

Cơ sở dữ liệu của hệ thống E-commerce quản lý thông tin về người dùng, vai trò, sản phẩm, đơn hàng, kho hàng, thanh toán, đánh giá và các hoạt động liên quan.

- Thực thể `User` (Người dùng - PostgreSQL): Mỗi người dùng có các thuộc tính: `userId` (định danh duy nhất), `email` (duy nhất), `password`, `fullName`, `phoneNum`, `createdAt`, `role`. Mỗi người dùng có thể có nhiều địa chỉ giao hàng. Mỗi người dùng có thể tạo nhiều đơn hàng. Mỗi người dùng có thể có nhiều token đăng nhập (`refresh token`) để quản lý phiên làm việc.
- Thực thể `Address` (Địa chỉ người dùng - PostgreSQL). Mỗi địa chỉ gồm: `addressID` duy nhất, `addressLine`, `city`, `district`, `isDefault`. Mỗi người dùng có thể có nhiều địa chỉ giao hàng.
- Thực thể `AuthToken` (Phiên đăng nhập - PostgreSQL). Mỗi bản token gồm: `tokenID` duy nhất, `refreshToken`, `userAgent`, `expiresAt`, `isRevoked`. Mỗi `token` thuộc về đúng một người dùng, cho phép quản lý đăng nhập từ nhiều thiết bị.
- Thực thể `Inventory` (Kho hàng). Mỗi bản ghi của kho gồm: `inventoryID` duy nhất, `productID` (tham chiếu logic tới MongoDB), `sku` duy nhất, `stockQuantity`, `lastUpdated`. Mỗi sản phẩm có thể có nhiều biến thể kho (SKU khác nhau).
- Thực thể `Order` (Đơn hàng - PostgreSQL). Mỗi đơn hàng có: `orderID` duy nhất, `status` (`PENDING`, `PROCESSING`, `SHIPPED`, `DELIVERED`, `CANCELLED`), `totalAmount`, `shippingAdd` (snapshot), `createdAt`. Mỗi đơn hàng thuộc về một người dùng. Mỗi đơn hàng gồm nhiều chi tiết đơn hàng (order item). Mỗi đơn hàng có đúng một thanh toán (`payment`).
- Thực thể `Item` (Chi tiết đơn hàng - PostgreSQL). Mỗi hàng hoá trong đơn hàng bao gồm: `itemID` (duy nhất), `productID`, `productName` (snapshot), `quantity`, `unitPrice`. Mỗi `OrderItem` thuộc về đúng một đơn hàng.
- Thực thể `Payment` (Thanh toán - PostgreSQL). Mỗi thực thể bao gồm: `paymentID` (duy nhất), `method`, `status`, `transactionDate`. Mỗi thanh toán gắn với đúng một đơn hàng.
- Thực thể `Review` (Đánh giá – MongoDB). Mỗi đánh giá bao gồm: `reviewID` (duy nhất), `productID`, `userID`, `userName`, `rating` (1–5 sao), `comment`, `images`, `createdAt`. Một người dùng có thể gửi nhiều đánh giá và một sản phẩm có thể nhận nhiều đánh giá từ nhiều người dùng.
- Thực thể `UserActivityLog` (Log hành vi – MongoDB). Mỗi bản log bao gồm: `logID` (duy nhất), `userID` (có thể null), `actionType`, `targetID`, `metadata`, `timestamp`. Hệ thống ghi nhận các hoạt động như xem sản phẩm, tìm kiếm, thêm vào giỏ hàng.
- Thực thể `Product` (Hàng hoá - `MongoDB`). Mỗi sản phẩm có: `productID` duy nhất, `name`, `slug`, `category`, `basePrice`, `description`, `images`, `attributes` (thuộc tính động như `RAM`, màu sắc, kích thước, cấu hình...), `isActive`, `createdAt`. Một sản phẩm có thể nhận nhiều đánh giá (review). Sản phẩm không lưu trực tiếp số lượng tồn kho mà được tham chiếu logic sang `PostgreSQL`.

2.4.3 Các ràng buộc ngữ nghĩa không biểu diễn được bằng (E-)ERD

- Email người dùng phải duy nhất và hợp lệ.
- Số lượng tồn kho không được âm.
- Tổng tiền đơn hàng phải lớn hơn hoặc bằng 0.
- Một đơn hàng chỉ có một bản ghi thanh toán.
- Trạng thái đơn hàng phải tuân theo luồng xử lý hợp lệ (Pending → Processing → Shipped → Delivered hoặc Cancelled).
- Refresh token chỉ hợp lệ trong thời gian cho phép và có thể bị thu hồi.
- Một sản phẩm chỉ được phép đặt hàng khi còn tồn kho.
- Đánh giá sản phẩm chỉ được gửi sau khi người dùng đã hoàn thành đơn hàng chứa sản phẩm đó.
- Các thuộc tính động của sản phẩm phải phù hợp với danh mục sản phẩm.
- Khi một sản phẩm bị xóa, các bản ghi kho và đơn hàng liên quan phải được xử lý tương ứng.
- Dữ liệu liên kết giữa PostgreSQL và MongoDB phải được đồng bộ chính xác thông qua productID và userID.

2.4.4 Các biểu đồ mô tả

2.4.4.1 Biểu đồ EERD cho SQL database

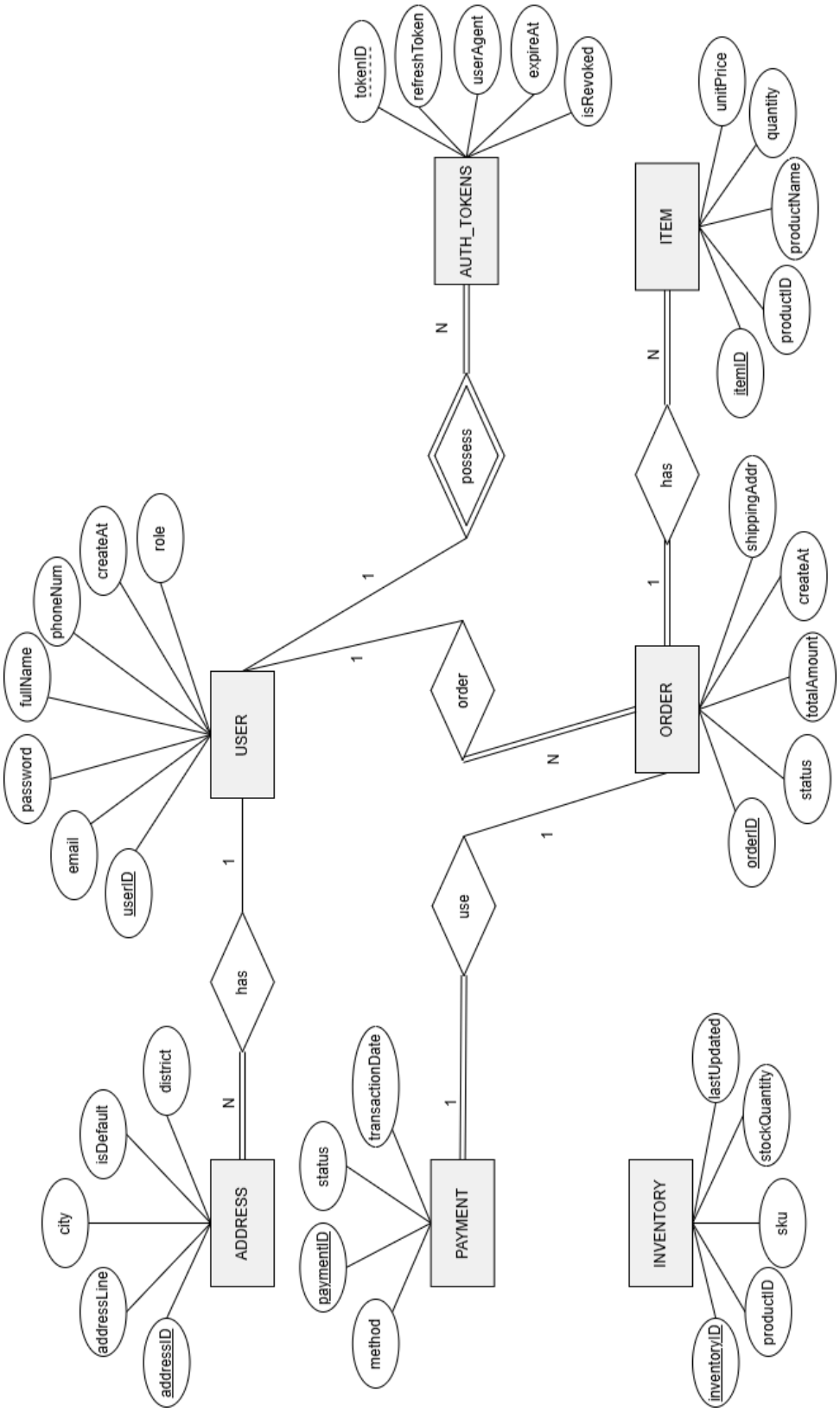

Figure 2: Biểu đồ EERD cho SQL database

2.4.4.2 Biểu đồ mô tả cho NoSQL database

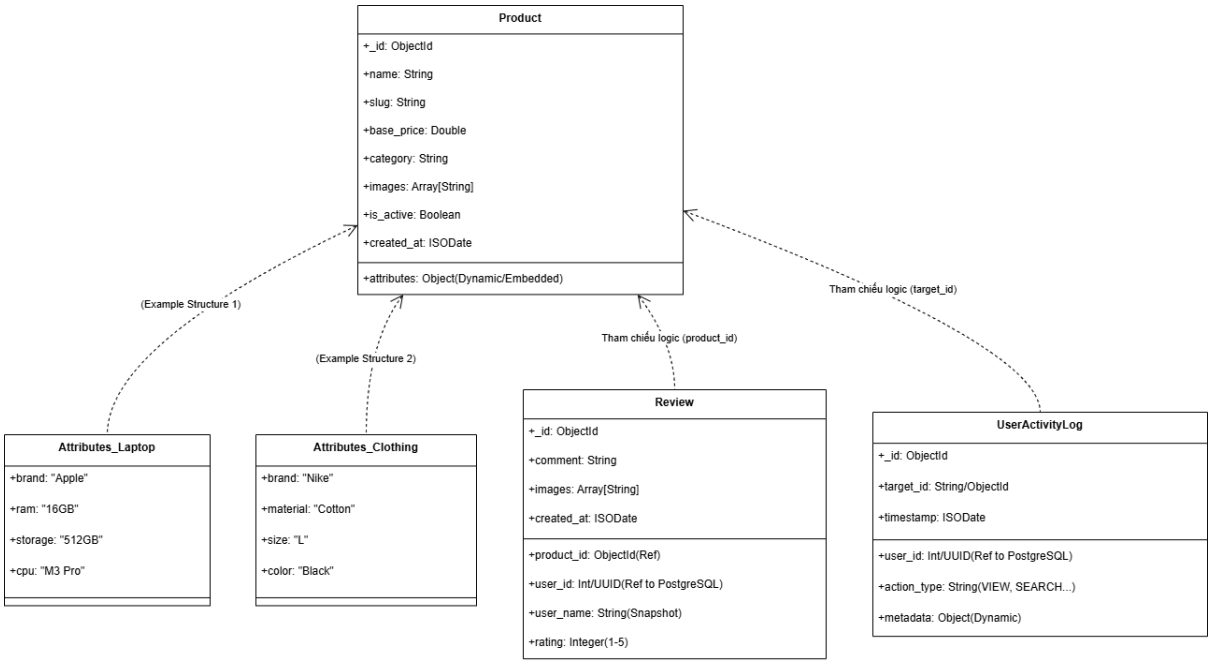

Figure 3: Biểu đồ mô tả cho NoSQL database

2.4.4.3 Biểu đồ mô tả các quan hệ logic giữa SQL và NoSQL

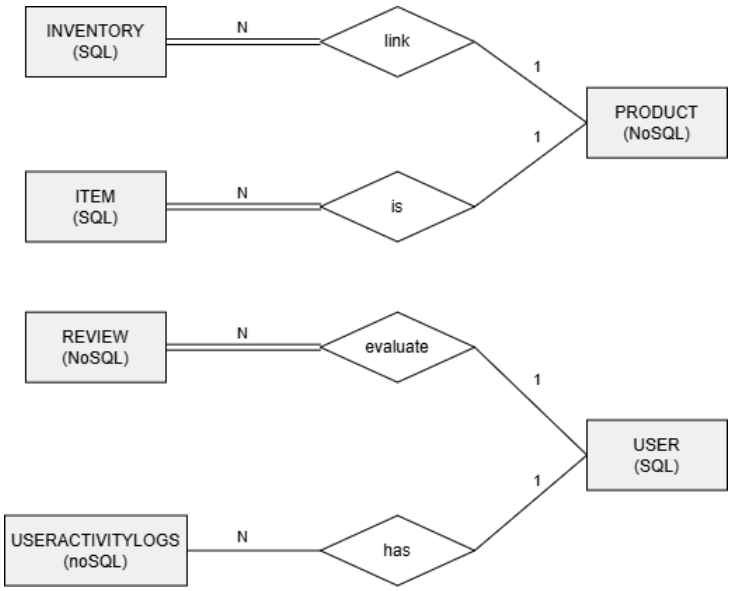

Figure 4: Biểu đồ mô tả các quan hệ logic giữa NoSQL database và SQL database

2.5 Ánh xạ các lược đồ Cơ sở dữ liệu

2.5.1 Ánh xạ mô hình dữ liệu của SQL

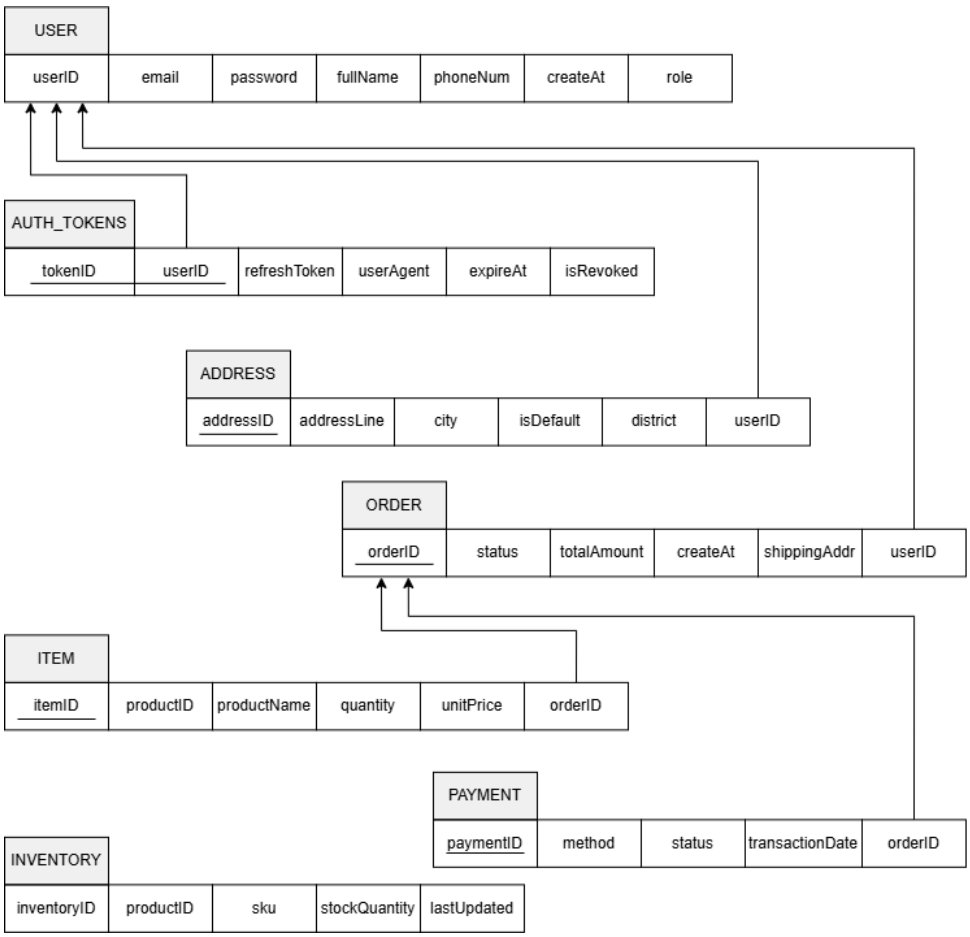

Figure 5: Lược đồ Cơ sở dữ liệu

2.5.2 Ánh xạ bộ sưu tập dữ liệu của NoSQL

- Sản phẩm (Products) - Ảnh (images):
  - Quan hệ: 1 - Ít (Mỗi sản phẩm chỉ khoảng 5-10 ảnh).
  - Quyết định: Embed. Lưu trực tiếp mảng URL ảnh trong document sản phẩm để lấy ra ngay lập tức.
- Sản phẩm (Products) - Thuộc tính (attributes):
  - Quan hệ: 1 - 1 (Mỗi sản phẩm có 1 bộ thông số).
  - Quyết định: Embed. Dữ liệu này luôn đi kèm sản phẩm và thay đổi cấu trúc tùy loại hàng (Polymorphic Pattern).
- Sản phẩm - Đánh giá (reviews):
  - Quan hệ: 1 - Rất nhiều (Một sản phẩm có thể có hàng nghìn review).
  - Quyết định: Reference. Nếu lồng review vào sản phẩm sẽ dễ bị tràn giới hạn 16MB của MongoDB. Do đó, tách Review ra collection riêng và tham chiếu bằng product_id.

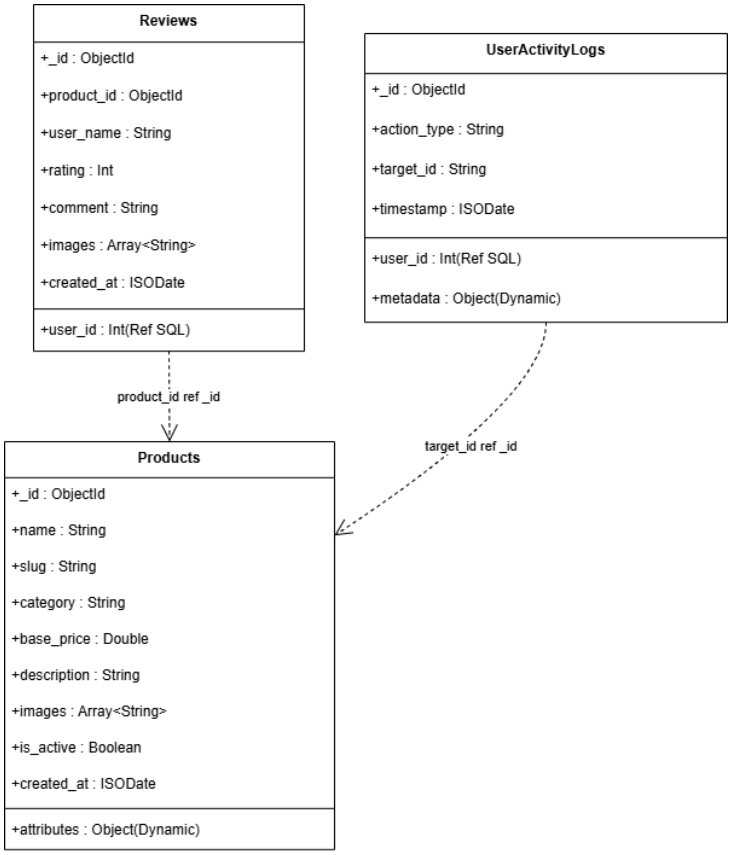

Figure 6: Bộ sưu tập dữ liệu

3 SO SÁNH VÀ THỰC NGHIỆM

3.1 Lưu trữ và Quản lý Dữ liệu (Data Storage & Management)

3.1.1 Mô hình dữ liệu và Cấu trúc logic (Logical Data Model)

Sự khác biệt trong cấu trúc tổ chức dữ liệu giữa hai hệ thống dẫn đến những phương thức tiếp cận khác nhau trong việc quản lý tính nhất quán và khả năng thay đổi cấu trúc dữ liệu theo thời gian:

- PostgreSQL sử dụng mô hình dữ liệu quan hệ (Relational Model) truyền thống. Dữ liệu được tổ chức theo cấu trúc phân cấp *Database* → *Schema* → *Table* → *Row* → *Column*. Tính nhất quán được đảm bảo thông qua việc định nghĩa lược đồ (Schema) cố định; mọi thay đổi về cấu trúc bảng đều cần các câu lệnh DDL (Data Definition Language).
- MongoDB sử dụng mô hình hướng tài liệu (Document Model). Dữ liệu được tổ chức theo cấp độ: *Database* → *Collection* → *Document*. MongoDB không yêu cầu định nghĩa lược đồ trước (Schema-less), cho phép mỗi Document trong cùng một Collection có số lượng trường và kiểu dữ liệu khác nhau, giúp tăng tốc độ phát triển ứng dụng khi yêu cầu thay đổi liên tục.

Về phương diện biểu diễn dữ liệu, hai hệ thống sử dụng các định dạng lưu trữ khác nhau để tối ưu hóa khả năng tương thích và không gian đĩa:

- PostgreSQL cung cấp hệ thống kiểu dữ liệu phong phú và nghiêm ngặt (Integer, Numeric, String, Boolean, Date...). Đặc biệt, PostgreSQL hỗ trợ các kiểu dữ liệu nâng cao như Array, Range, và các kiểu hình học. Đối với dữ liệu bán cấu trúc, PostgreSQL sử dụng **JSONB** (dạng nhị phân của JSON), cho phép lưu trữ và truy vấn hiệu quả các thuộc tính động bên trong một cột quan hệ.
- MongoDB sử dụng định dạng **BSON** (Binary JSON) để lưu trữ. So với JSON thông thường, BSON mở rộng thêm các kiểu dữ liệu đặc thù như `Timestamp`, `BinData`, `ObjectId`, và `Decimal128`. Việc lưu trữ dưới dạng nhị phân giúp MongoDB tối ưu hóa không gian đĩa và tốc độ duyệt (scan) qua các Document mà không cần thực hiện thao tác phân tách văn bản (parse).

Khi kích thước bản ghi vượt quá giới hạn trang vật lý tiêu chuẩn, mỗi hệ quản trị áp dụng một cơ chế riêng để quản lý các đối tượng dữ liệu lớn mà không làm ảnh hưởng đến hiệu năng chung:

- PostgreSQL sử dụng cơ chế **TOAST** (*The Oversized-Attribute Storage Technique*). Khi một hàng vượt quá kích thước trang (thường là 8KB), hệ thống sẽ tự động nén hoặc di chuyển dữ liệu lớn sang bảng lưu trữ phụ, chỉ để lại một con trỏ (Pointer) ở bảng chính, giúp duy trì tốc độ quét bảng (Sequence Scan).
- MongoDB sử dụng cơ chế **GridFS** để quản lý các tệp vượt quá giới hạn 16MB của một Document. GridFS chia nhỏ dữ liệu lớn thành các phần (chunks) lưu trữ trong các Collection riêng biệt, cho phép truy cập ngẫu nhiên vào một phần của tệp mà không cần nạp toàn bộ dữ liệu vào bộ nhớ đệm.

Đối với việc thiết lập mối liên hệ giữa các thực thể, hai hệ thống đại diện cho hai triết lý đối lập giữa việc phân rã để tối ưu bộ nhớ và tích hợp để tối ưu tốc độ truy xuất:

- PostgreSQL ưu tiên phương pháp **Normalization** (Chuẩn hóa). Dữ liệu được chia nhỏ vào nhiều bảng để tránh dư thừa; các mối quan hệ được thiết lập thông qua khóa ngoại (Foreign Keys) và được kết hợp lại khi cần thiết bằng các phép JOIN.
- MongoDB ưu tiên phương pháp **Embedding** (Nhúng) hoặc **Linking** (Liên kết). Dữ liệu liên quan thường được gom nhóm trực tiếp vào trong một tài liệu duy nhất (Denormalization), giúp giảm thiểu các truy vấn chéo và tối ưu hóa hiệu suất đọc cho các đối tượng dữ liệu phức tạp.

3.1.2 Kiến trúc lưu trữ vật lý (Physical Storage Architecture)

Cách thức tổ chức các đơn vị lưu trữ vật lý phản ánh sự khác biệt trong việc tối ưu hóa tài nguyên phần cứng và quản lý I/O giữa hai hệ quản trị:

- PostgreSQL tổ chức dữ liệu dưới dạng các **Pages** cố định (mặc định 8KB). Khi cập nhật dữ liệu, Postgres áp dụng cơ chế *Copy-on-write*, tạo ra phiên bản mới của dòng thay vì ghi đè. Điều này đảm bảo an toàn dữ liệu nhưng gây ra hiện tượng phân mảnh (Bloat), đòi hỏi tiến trình *Vacuum* thường xuyên để dọn dẹp không gian đĩa.
- MongoDB (với Storage Engine **WiredTiger**) quản lý dữ liệu linh hoạt hơn qua cơ chế **Block Manager**. WiredTiger sử dụng các đơn vị lưu trữ biến thiên và hỗ trợ nén dữ liệu gốc (Native Compression), giúp giảm dung lượng đĩa và tối ưu băng thông I/O thông qua việc ghi dữ liệu theo các điểm kiểm tra (*Checkpoints*).

Phương thức quản trị việc ghi dữ liệu và phục hồi sau sự cố cũng được triển khai dựa trên các cơ chế nhật ký đặc thù:

- PostgreSQL sử dụng **Write-Ahead Log (WAL)** và **Free Space Map (FSM)**. WAL ghi lại mọi thay đổi vật lý trước khi áp dụng vào tệp chính để đảm bảo khả năng phục hồi, trong khi FSM theo dõi các khoảng trống trong Pages để tái sử dụng hiệu quả cho dữ liệu mới.
- MongoDB sử dụng cơ chế **Journaling** kết hợp với quản lý vùng nhớ linh hoạt. Journaling bảo vệ tính toàn vẹn của dữ liệu tương tự WAL, nhưng WiredTiger cho phép tái sử dụng không gian trống ngay lập tức sau khi xóa dữ liệu mà không cần các tiến trình dọn dẹp nặng nề như Postgres.

3.1.3 Quản trị lược đồ và Sự thay đổi dữ liệu (Schema Management & Data Evolution)

Cách thức quản lý sự thay đổi cấu trúc dữ liệu theo thời gian phản ánh triết lý khác nhau về tính linh hoạt và tính kiểm soát:

- PostgreSQL tuân thủ phương pháp **Schema-on-write**. Cấu trúc dữ liệu phải được xác lập chặt chẽ trước khi lưu trữ; mọi thay đổi (`ALTER TABLE`) đều được kiểm soát bởi hệ thống. Điều này đảm bảo dữ liệu luôn sạch và đồng nhất nhưng có thể gây chậm trễ trong quy trình phát triển nhanh (Agile).
- MongoDB áp dụng phương pháp **Schema-on-read**. Hệ thống không bắt buộc cấu trúc cố định, cho phép ứng dụng tự định nghĩa trường mới trực tiếp khi ghi. Điều này mang lại sự linh hoạt tối đa cho các hệ thống có cấu trúc thay đổi thường xuyên, giúp loại bỏ các thao tác di cư dữ liệu (Migration) phức tạp.

Cơ chế duy trì các ràng buộc và quy tắc quản trị dữ liệu được triển khai nhằm đảm bảo tính toàn vẹn thông tin theo các cấp độ khác nhau:

- PostgreSQL cung cấp hệ thống ràng buộc (**Constraints**) nội tại mạnh mẽ như `NOT NULL`, `UNIQUE`, `FOREIGN KEY`. Các quy tắc này được thực thi ở mức nhân cơ sở dữ liệu, đảm bảo không có dữ liệu sai cấu trúc nào có thể được ghi vào hệ thống.
- MongoDB quản lý tính nhất quán thông qua **JSON Schema Validation**. Thay vì khóa chặt cấu trúc, người quản trị định nghĩa các quy tắc kiểm tra tại mức Collection. Phương pháp này cho phép MongoDB vừa duy trì tính linh hoạt của NoSQL, vừa đảm bảo chất lượng dữ liệu tương đương với hệ quản trị quan hệ khi cần thiết.

3.2 Indexing

3.2.1 Cơ sở lý thuyết

Indexing (đánh chỉ mục) là một kỹ thuật tổ chức dữ liệu nhằm tối ưu hóa tốc độ truy xuất dữ liệu từ database bằng cách giảm số lượng bản ghi cần phải quét thay vì phải quét toàn bộ bảng hoặc Collection (Full Table Scan). Tuy nhiên, việc xây dựng và duy trì indexing cũng kéo theo phát sinh thêm chi phí về bộ nhớ và thời gian xử lý các thao tác.

3.2.1.1 PostgreSQL PostgreSQL hỗ trợ đa dạng các loại indexng khác nhau, cho phép tối ưu hóa hiệu quả cho nhiều kiểu dữ liệu và mô hình truy vấn.

- **B-tree:** Là loại indexing mặc định và phổ biến nhất. B-tree lưu trữ các khóa theo thứ tự tăng dần trong một cấu trúc cây cân bằng (`balanced tree`). Từ đó cho phép thực hiện hiệu quả các phép tìm kiếm bằng (`=`), so sánh (`<, >, BETWEEN`) và sắp xếp (`ORDER BY`) với độ phức tạp $O(logn)$.
- Bên cạnh đó, PostgreSQL còn hỗ trợ các loại chỉ mục nâng cao giúp tối ưu cho từng kiểu dữ liệu và truy vấn đặc thù:
  - **Hash:** Lưu Hash value của dữ liệu, từ đó tăng tốc độ truy vấn so sánh bằng (`=`)
  - **GIN (Generalized Inverted Index):** GIN ánh xạ từng phần tử con bên trong dữ liệu tới các bản ghi chứa nó. Phù hợp cho các trường dữ liệu chứa nhiều giá trị như Array, JSONB hoặc khi thực hiện full-text search.
  - **GiST (Generalized Search Tree):** Là một framework index tổng quát cho phép xây dựng các cấu trúc cây tùy chỉnh, linh hoạt. Thường được sử dụng cho dữ liệu hình học, tọa độ không gian hoặc các bài toán tìm kiếm phức tạp.
  - **BRIN (Block Range Index):** Thay vì in dex từng dòng, BRIN lưu giá trịMin/Max của từng khối dữ liệu (block), giúp tiết kiệm dung lượng đĩa gấp hàng trăm lần so với B-tree. Đây là một loại index đặc biệt dành cho dữ liệu cực lớn (Big Data) có tính thứ tự tự nhiên (ví dụ: cột `created_at`).

3.2.1.2 MongoDB MongoDB sử dụng cơ chế chỉ mục dựa trên cấu trúc **B-tree** nhằm tăng tốc độ truy vấn trên các Collection. Mỗi index lưu trữ giá trị của một hoặc nhiều trường (field) và con trỏ trực tiếp đến document tương ứng trong Collection. Theo mặc định, MongoDB tự động tạo index trên trường `_id` để đảm bảo tính duy nhất và hiệu quả truy xuất dữ liệu.

MongoDB hỗ trợ nhiều loại cơ chế index khác nhau, phù hợp với mô hình dữ liệu dạng document và các kiểu truy vấn phổ biến:

- **Single Field Index:** Index được tạo trên một field duy nhất của document, phù hợp cho các truy vấn lọc hoặc so sánh một thuộc tính cụ thể.
- **Compound Index:** Index được xây dựng trên nhiều field theo một thứ tự xác định, đặc biệt hiệu quả với các truy vấn kết hợp nhiều điều kiện lọc và sắp xếp (`sort`), tuân theo nguyên tắc tiền tố (prefix rule).
- **Multikey Index:** Áp dụng khi trường được đánh index là một mảng (array). MongoDB sẽ tự động tạo index cho từng phần tử trong mảng, cho phép truy vấn hiệu quả các document chứa giá trị cần tìm.
- **Text Index:** Hỗ trợ tìm kiếm toàn văn (full-text search) trên các field kiểu chuỗi. MongoDB thực hiện phân tách từ (tokenization), loại bỏ stop words và cho phép gán trọng số (weight) cho các field để cải thiện độ chính xác của kết quả tìm kiếm.
- **Hashed Index:** Sử dụng giá trị băm của field thay vì giá trị gốc, chỉ tối ưu cho các truy vấn so sánh bằng (`=`) và thường được sử dụng trong các hệ thống phân mảnh dữ liệu (sharding) để đảm bảo phân bố dữ liệu đồng đều.
- **TTL Index (Time-To-Live):** Cho phép tự động xóa document sau một khoảng thời gian nhất định dựa trên field kiểu `Date`, thường được áp dụng cho các dữ liệu tạm thời như session, log hoặc cache.

3.2.2 Kịch bản thử nghiệm (Test Scenario)

Nhóm thiết lập kịch bản mô phỏng các thao tác các truy vấn phổ biến như tìm kiếm, lọc và sắp xếp sản phẩm

- **Dữ liệu:**
  - Bảng / Collection `Products` chứa 1.000.000 sản phẩm.
  - Mỗi sản phẩm có các thuộc tính chính: `id`, `category`, `price`, `name`, `createdAt`.
- **Truy vấn phổ biến:**
  - Tìm kiếm sản phẩm theo `category`.
  - Lọc sản phẩm theo khoảng giá (`price BETWEEN x AND y`).
  - Sắp xếp sản phẩm theo thời gian tạo (`createdAt`) mới nhất.
  - Tìm kiếm sản phẩm theo `keyword` trong `name`.
- **Tiêu chí đánh giá:**
  - Thời gian thực thi truy vấn.
  - Số lượng bản ghi được quét.
  - Chi phí ghi dữ liệu.

3.2.3 Giải pháp và Đánh giá

3.2.3.1 PostgreSQL Nhóm sử dụng các cơ chế chỉ mục B-tree và GIN để tối ưu truy vấn trên bảng `Products`.

- **Giải pháp:**
  - Tạo **B-tree index** trên các cột `category` và `price` nhằm tăng tốc các truy vấn lọc và so sánh.

- Sử dụng **Multi-column B-tree index** trên (`category`, `createdAt`) để tối ưu truy vấn lọc theo danh mục và sắp xếp theo thời gian.
- Sử dụng **GIN index** kết hợp với `tsvector` cho trường `name` nhằm hỗ trợ tìm kiếm từ khóa trong văn bản hiệu quả.

◦ **Đánh giá:**

- **Ưu điểm:**
  * Hỗ trợ đa dạng loại index tối ưu truy vấn mạnh cho nhiều kiểu truy vấn phức tạp và nghiệp vụ logic cao.
  * Hiệu quả cao với các truy vấn kết hợp nhiều điều kiện và `JOIN`.
- **Nhược điểm:**
  * Chi phí ghi cao do phải cập nhật nhiều index.
  * Việc thiết kế index đòi hỏi hiểu rõ quy trình, tránh gây dư thừa.

3.2.3.2 MongoDB Nhóm áp dụng các cơ chế indexing phù hợp với mô hình document trên Collection `Products`.

◦ **Giải pháp:**

– Tạo **Single Field Index** trên `category` để tăng tốc truy vấn lọc theo danh mục.

– Sử dụng **Compound Index** trên (`category`, `createdAt`) nhằm tối ưu truy vấn lọc kết hợp sắp xếp.

– Sử dụng **Text Index** cho trường `name` để hỗ trợ tìm kiếm sản phẩm theo từ khóa.

◦ **Đánh giá:**

– **Ưu điểm:**
  * Tạo và sử dụng index đơn giản, phù hợp với các truy vấn đọc có tần suất cao.
  * Hiệu năng tốt với các truy vấn tập trung trên một Collection.
  * Linh hoạt với dữ liệu không có schema cố định.

– **Nhược điểm:**
  * cơ chế index ít đa dạng hơn so với PostgreSQL, dẫn đến hạn chế với các truy vấn phức tạp.
  * Hiệu quả giảm khi cần xử lý các logic truy vấn phức tạp hoặc các Collection có nhiều quan hệ.

3.2.4 Kết luận

PostgreSQL thể hiện ưu thế trong các truy vấn phức tạp nhờ đa dạng cơ chế index và bộ tối ưu truy vấn mạnh, phù hợp với các nghiệp vụ yêu cầu tính chính xác và logic xử lý cao. Ngược lại, MongoDB phù hợp hơn với các truy vấn đọc đơn giản, tần suất lớn trên một Collection nhờ mô hình document linh hoạt. Việc kết hợp PostgreSQL và MongoDB trong một kiến trúc Hybrid giúp tận dụng ưu điểm của cả hai hệ quản trị, đáp ứng hiệu quả các yêu cầu phức tạp.

3.3 Query Processing

3.3.1 Tổng quan về Xử lý truy vấn

Xử lý truy vấn là một quy trình phức tạp bao gồm nhiều bước nhằm chuyển đổi một câu truy vấn cấp cao thành một chiến lược thực thi hiệu quả để truy xuất dữ liệu từ cơ sở dữ liệu.

Một câu truy vấn trong ngôn ngữ cấp cao như SQL mang tính *khai báo* (declarative), nghĩa là nó chỉ định *những gì* người dùng muốn lấy ra chứ không mô tả *cách thức* để truy xuất dữ liệu. Do đó, DBMS cần xác định một chiến lược thực thi tối ưu hoặc gần tối ưu nhất nhằm đảm bảo hiệu năng truy vấn.

Quy trình xử lý truy vấn

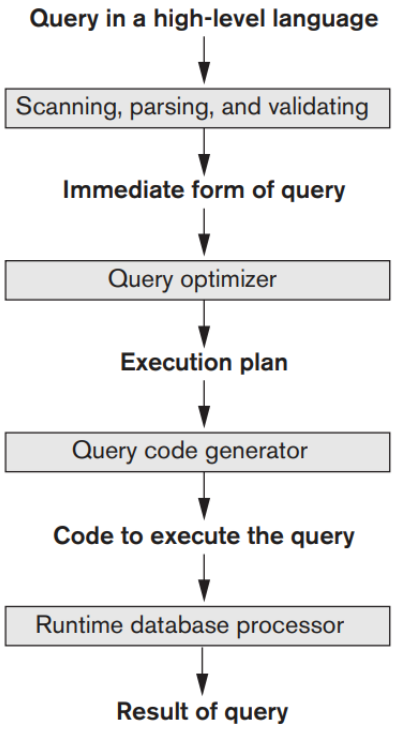

Figure 7: Quy trình xử lý truy vấn trong hầu hết các hệ quản trị cơ sở dữ liệu

Quy trình xử lý truy vấn trong DBMS thường bao gồm các giai đoạn sau:

1. **Quét, phân tích cú pháp và kiểm tra hợp lệ (Scanning, Parsing, and Validation)**: Giai đoạn này nhận diện các đơn vị từ vựng (tokens) như tên quan hệ, tên thuộc tính và các từ khóa SQL. Sau đó, hệ thống kiểm tra cú pháp nhằm đảm bảo truy vấn tuân thủ các quy tắc ngữ pháp của ngôn ngữ truy vấn, đồng thời kiểm tra tính hợp lệ về mặt ngữ nghĩa.
2. **Biểu diễn nội bộ (Internal Representation)**: Sau khi phân tích, truy vấn được chuyển đổi thành một cấu trúc dữ liệu nội bộ, phổ biến nhất là cây truy vấn (query tree) hoặc đồ thị truy vấn (query graph). Cây truy vấn tương ứng với một biểu thức đại số quan hệ mở rộng. Trong thực tế, truy vấn SQL thường được phân rã thành các khối truy vấn (query blocks), là đơn vị cơ bản để dịch sang các toán tử đại số và thực hiện tối ưu hóa.
3. **Tối ưu hóa truy vấn (Query Optimization)**: Đây là giai đoạn quan trọng nhất trong xử lý truy vấn, nơi DBMS lựa chọn chiến lược thực thi phù hợp. Quá trình

này được thực hiện bởi mô-đun *Query Optimizer*. Hai kỹ thuật chính được sử dụng bao gồm:

- *Tối ưu hóa dựa trên quy tắc (Heuristic Optimization)*, sử dụng các quy tắc kinh nghiệm để biến đổi cây truy vấn ban đầu thành một cây tương đương nhưng hiệu quả hơn.
- *Tối ưu hóa dựa trên chi phí (Cost-based Optimization)*, dựa trên việc ước lượng chi phí thực thi của các chiến lược khác nhau và lựa chọn chiến lược có chi phí thấp nhất.

4. **Sinh mã thực thi (Code Generation)**: Sau khi kế hoạch thực thi tối ưu được lựa chọn, bộ sinh mã truy vấn (Query Code Generator) sẽ tạo ra mã thực thi tương ứng. Mã này bao gồm các lời gọi đến các thuật toán vật lý cụ thể để thực hiện các phép toán cơ sở dữ liệu.
5. **Thực thi (Runtime Execution)**: Bộ xử lý cơ sở dữ liệu tại thời điểm chạy (Runtime Database Processor) sẽ thực thi mã được sinh ra, ở chế độ biên dịch hoặc thông dịch, để tạo ra kết quả truy vấn. Nếu xảy ra lỗi trong quá trình thực thi, hệ thống sẽ phát sinh thông báo lỗi tương ứng.

Các thuật toán xử lý phép toán đại số quan hệ

Để xử lý truy vấn, các hệ quản trị cơ sở dữ liệu sử dụng các thuật toán vật lý tương ứng với từng phép toán trong đại số quan hệ.

1. **Sắp xếp ngoài (External Sorting)**: Sắp xếp là một thuật toán cơ bản, cần thiết đối với các truy vấn có mệnh đề `ORDER BY`, loại bỏ trùng lặp (`DISTINCT`), hoặc hỗ trợ các thuật toán kết như *sort-merge join*. Đối với các tập dữ liệu lớn không vừa trong bộ nhớ chính, DBMS thường sử dụng thuật toán *external sort-merge*.
2. **Phép chọn (Selection)**: Phép chọn là thao tác tìm kiếm các bản ghi thỏa mãn một điều kiện nhất định. Các phương pháp phổ biến bao gồm quét tuyến tính (linear scan), tìm kiếm nhị phân (binary search) khi dữ liệu đã được sắp xếp, hoặc sử dụng chỉ mục (index scan) như chỉ mục sơ cấp, chỉ mục băm hoặc chỉ mục thứ cấp (B^+-tree). Các phương pháp dựa trên chỉ mục đặc biệt hiệu quả đối với truy vấn đẳng thức và truy vấn phạm vi.
3. **Phép kết (Join)**: Phép kết là một trong những thao tác tốn chi phí nhất trong xử lý truy vấn. Các chiến lược phổ biến bao gồm:
   - *Nested-loop join*: Với mỗi bản ghi trong bảng ngoài, hệ thống quét toàn bộ bảng trong để tìm các cặp khớp.
   - *Index-based nested-loop join*: Sử dụng chỉ mục trên thuộc tính kết của bảng trong để giảm chi phí quét.
   - *Sort-merge join*: Hiệu quả khi cả hai bảng đã được sắp xếp theo thuộc tính kết.
   - *Partition hash join*: Phân hoạch hai bảng bằng cùng một hàm băm, sau đó thực hiện kết nối trên các phân hoạch tương ứng.

Tối ưu hóa truy vấn

Tối ưu hóa truy vấn chủ yếu dựa trên hai kỹ thuật chính:

1. **Tối ưu hóa dựa trên quy tắc (Heuristic Optimization)**: Phương pháp này sử dụng các quy tắc kinh nghiệm để biến đổi cây truy vấn ban đầu thành một cây tương đương nhưng hiệu quả hơn. Một số đặc điểm chính bao gồm:

- Thực hiện các phép `SELECT` và `PROJECT` càng sớm càng tốt nhằm giảm kích thước dữ liệu trung gian.
- Cây truy vấn (Query Tree) biểu diễn các phép toán, trong đó các nút lá là các quan hệ đầu vào và các nút trong là các phép toán như σ, π, $\bowtie$.
- Áp dụng các quy tắc biến đổi để đẩy các phép lọc xuống thấp nhất có thể trong cây truy vấn.

2. **Tối ưu hóa dựa trên chi phí (Cost-based Optimization)**: Phương pháp này ước lượng chi phí thực thi của các chiến lược khác nhau và lựa chọn chiến lược có chi phí thấp nhất.
   - Thành phần chi phí bao gồm chi phí truy cập đĩa (I/O), chi phí tính toán CPU, chi phí lưu trữ và chi phí truyền dữ liệu (đối với hệ phân tán).
   - Bộ tối ưu hóa sử dụng thông tin thống kê từ *system catalog* như số lượng bản ghi, kích thước bản ghi, số lượng khối và hệ số phân khối.
   - Độ chọn lọc (selectivity) được sử dụng để ước lượng kích thước các kết quả trung gian, thường dựa trên histogram.

Do số lượng thứ tự thực hiện phép kết (join ordering) có thể rất lớn ($n!$), các bộ tối ưu hóa thường giới hạn không gian tìm kiếm bằng cách ưu tiên các cấu trúc *left-deep tree*. Trong cấu trúc này, toán hạng bên phải của mỗi phép kết luôn là một quan hệ cơ sở, thuận lợi cho việc thực thi theo đường ống (pipelining).

Đánh giá và thực thi truy vấn

Đánh giá truy vấn (Query Evaluation) là quá trình chuyển đổi một kế hoạch thực thi truy vấn thành các hoạt động cụ thể để truy xuất dữ liệu. Hai chiến lược chính được sử dụng bao gồm:

1. **Materialized Evaluation**: Mỗi phép toán trong cây truy vấn được thực thi tuần tự và kết quả trung gian được lưu trữ tạm thời trên đĩa trước khi chuyển sang phép toán tiếp theo. Cách tiếp cận này gây ra chi phí I/O đáng kể do phải ghi và đọc lại các kết quả trung gian.
2. **Pipelining Evaluation**: Các phép toán được kết nối thành một luồng xử lý liên tục. Ngay khi một bộ kết quả được tạo ra, nó được chuyển trực tiếp sang phép toán tiếp theo mà không cần lưu trữ trung gian. Cách tiếp cận này giúp giảm chi phí I/O và cho phép hệ thống trả về kết quả sớm hơn. Trong thực tế, các phép toán vật lý thường được cài đặt dưới dạng các *iterator*.

3.3.2 Xử lý truy vấn trong PostgreSQL

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ tuân thủ chuẩn SQL, sử dụng pipeline xử lý truy vấn nhiều giai đoạn kết hợp giữa phân tích cú pháp, tối ưu hóa dựa trên chi phí và cơ chế thực thi hiệu quả. Quá trình xử lý truy vấn trong PostgreSQL bao gồm các bước chính sau:

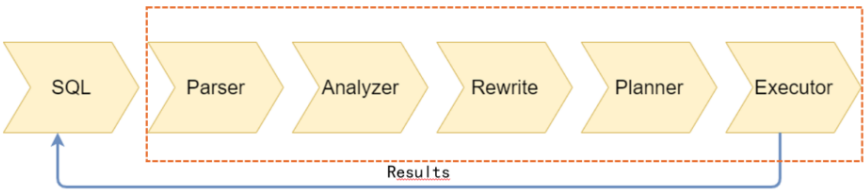

Figure 8: Quy trình xử lý truy vấn trong PostgreSQL

1. **Parsing**: Truy vấn SQL đầu vào được phân tích cú pháp nhằm kiểm tra tính hợp lệ về mặt ngữ pháp và cấu trúc. PostgreSQL chuyển truy vấn thành cây cú pháp (parse tree), phản ánh cấu trúc logic của câu lệnh SQL, bao gồm các mệnh đề `SELECT`, `FROM`, `WHERE`, `JOIN` và `GROUP BY`.

2. **Analysis và Query Rewrite**: Ở giai đoạn này, PostgreSQL thực hiện phân tích ngữ nghĩa bằng cách xác minh sự tồn tại của các đối tượng cơ sở dữ liệu như bảng, cột, view và kiểm tra kiểu dữ liệu cũng như quyền truy cập. Đồng thời, bộ rewrite áp dụng các luật chuyển đổi truy vấn, bao gồm mở rộng view, biến đổi các biểu thức con như `IN`, `EXISTS` và áp dụng chính sách bảo mật ở mức hàng (Row Level Security).

3. **Query Optimization (Planner)**: Bộ tối ưu hóa của PostgreSQL sử dụng mô hình tối ưu hóa dựa trên chi phí (Cost-Based Optimizer). Dựa trên các thống kê như số lượng bản ghi, phân bố dữ liệu và độ chọn lọc của chỉ mục, hệ thống sinh ra nhiều kế hoạch thực thi khả thi. Mỗi kế hoạch được ước lượng chi phí I/O và CPU, từ đó lựa chọn kế hoạch có chi phí thấp nhất, quyết định thứ tự join, thuật toán join (Nested Loop, Hash Join, Merge Join) và phương pháp truy xuất dữ liệu.

4. **Execution Engine**: Kế hoạch thực thi được chuyển cho bộ máy thực thi, nơi các toán tử được xử lý theo mô hình iterator (Volcano model). Các phép toán như quét dữ liệu, lọc, nối, sắp xếp và tổng hợp được thực hiện theo đơn vị bộ (tuple-at-a-time). PostgreSQL cũng hỗ trợ thực thi song song để tận dụng tài nguyên đa lõi.

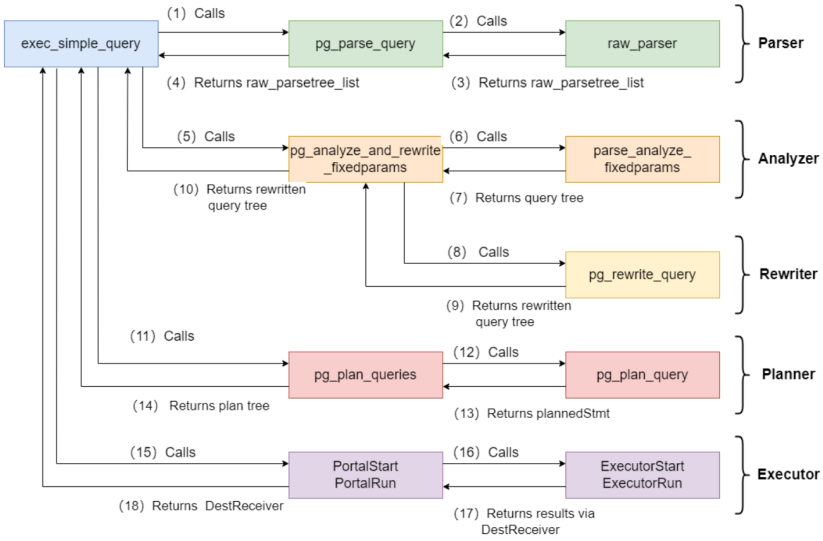

Figure 9: Vòng đời xử lý truy vấn trong PostgreSQL

3.3.3 Xử lý truy vấn trong MongoDB

Quy trình xử lý truy vấn trong MongoDB được thiết kế nhằm đảm bảo hệ thống luôn lựa chọn kế hoạch thực thi hiệu quả nhất, ngay cả khi khối lượng dữ liệu hoặc phân bố dữ liệu thay đổi theo thời gian.

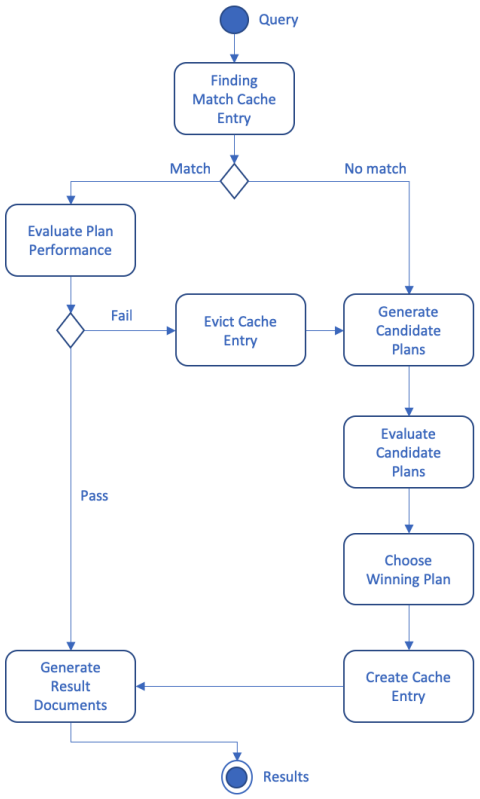

Figure 10: Quy trình xử lý truy vấn trong MongoDB

1. **Phân tích Truy vấn (Query Parsing)**: Mọi quy trình đều bắt đầu từ một **Query**. Truy vấn MongoDB được biểu diễn dưới dạng BSON và có thể bao gồm các toán tử như `$match`, `$and`, `$or`, `$gt`. Hệ thống tiến hành phân tích cú pháp truy vấn nhằm

xác định tập collection mục tiêu, các điều kiện lọc dữ liệu và các yêu cầu truy xuất, xử lý dữ liệu liên quan.

2. **Kiểm tra Bộ nhớ đệm Kế hoạch (Finding Match Cache Entry)**

   Trước khi sinh ra kế hoạch thực thi mới, MongoDB kiểm tra *plan cache* để xác định xem một truy vấn có cùng hình dạng (query shape) đã từng được xử lý hay chưa.

   - **Match (Có sẵn):**
     - MongoDB sử dụng kế hoạch trong cache và tiến hành *Evaluate Plan Performance* để đánh giá hiệu năng thực tế.
     - **Pass:** Nếu kế hoạch vẫn đạt hiệu năng tốt, hệ thống tiếp tục sử dụng kế hoạch này và chuyển sang bước sinh kết quả.
     - **Fail:** Nếu hiệu năng suy giảm (ví dụ do phân bố dữ liệu thay đổi), kế hoạch sẽ bị *Evict Cache Entry* và hệ thống tiến hành lập kế hoạch lại.
   - **No Match (Chưa có):** MongoDB chuyển trực tiếp sang giai đoạn lập kế hoạch mới.

3. **Lập Kế hoạch và Tối ưu hóa Khi Thực thi (Query Planning & Runtime Optimization)**

   Khi không tồn tại kế hoạch phù hợp trong cache, MongoDB thực hiện:

   - **Generate Candidate Plans:** Sinh ra nhiều kế hoạch thực thi ứng viên dựa trên các chỉ mục (indexes) hiện có, bao gồm index scan, collection scan hoặc các chiến lược kết hợp.
   - **Evaluate Candidate Plans:** Các kế hoạch ứng viên được chạy thử trong một khoảng thời gian ngắn (trial execution) nhằm đo lường các chỉ số thực tế như số lượng tài liệu được quét và thời gian xử lý.

4. **Lựa chọn và Lưu Kế hoạch (Plan Selection & Caching)**

   Sau quá trình đánh giá, MongoDB thực hiện:

   - **Choose Winning Plan:** Kế hoạch có hiệu năng tốt nhất được chọn làm kế hoạch chiến thắng.
   - **Create Cache Entry:** Kế hoạch chiến thắng được lưu vào plan cache để tái sử dụng cho các truy vấn tương tự trong tương lai.

5. **Thực thi và Trả Kết quả (Execution Engine & Cursor)**

   MongoDB thực thi truy vấn dựa trên kế hoạch đã được lựa chọn:

   - **Generate Result Documents:** Truy vấn được xử lý theo mô hình *document-at-a-time*, trong đó mỗi tài liệu BSON được xử lý như một đơn vị độc lập.
   - **Results:** Kết quả được trả về cho client thông qua *cursor*, cho phép truy xuất dữ liệu từng phần, giúp tối ưu bộ nhớ cho cả phía server và client.

Đối với các truy vấn nâng cao, MongoDB hỗ trợ:

- **Aggregation Pipeline:** Dữ liệu được xử lý qua chuỗi các stage như lọc, nhóm, sắp xếp và tính toán.
- **Distributed Query (Scatter–Gather):** Trong môi trường sharded cluster, truy vấn được phân tán đến nhiều node (scatter) và kết quả được tổng hợp lại (gather) trước khi trả về cho người dùng.

3.4 Transaction

3.5 Concurrency Control

3.5.1 Cơ sở lý thuyết

Khả năng điều khiển đồng thời (Concurrency Control) quyết định hiệu suất và tính đúng đắn của dữ liệu trong môi trường đa người dùng.

- **PostgreSQL (MVCC):** PostgreSQL giải quyết bài toán đồng thời bằng cơ chế *Multi-Version Concurrency Control* (MVCC). Thay vì khóa dữ liệu khi đọc, hệ thống cung cấp cho mỗi transaction một bản chụp (snapshot) dữ liệu tại thời điểm truy vấn. Nguyên tắc cốt lõi là "Readers don't block writers, and writers don't block readers" (Người đọc không chặn người ghi và ngược lại). PostgreSQL hỗ trợ đầy đủ các mức cô lập giao dịch chuẩn SQL và cung cấp khóa tường minh (`Explicit Locking`) cho các trường hợp cần kiểm soát tranh chấp gay gắt [**?**].
- **MongoDB (Document-Level Locking):** MongoDB sử dụng cơ chế khóa đa mức độ (*Multiple-granularity locking*). Với storage engine mặc định là **WiredTiger**, MongoDB hỗ trợ khóa ở cấp độ tài liệu (*Document-level locking*). Điều này có nghĩa là nhiều thao tác ghi có thể diễn ra đồng thời trên cùng một Collection miễn là chúng tác động vào các tài liệu khác nhau. MongoDB cũng sử dụng các khóa ý định (*Intent Locks*) để tối ưu hóa việc quản lý tài nguyên [**?**].

3.5.2 Kịch bản thử nghiệm (Test Scenario)

Nhóm thiết lập kịch bản **Flash Sale** để kiểm tra tính nhất quán dữ liệu (Data Consistency):

- **Tài nguyên:** Sản phẩm có `Stock = 1`.
- **Tải (Load):** 100 yêu cầu mua hàng (Request) được gửi đồng thời.
- **Điều kiện đạt:** Chỉ 1 đơn hàng thành công, tồn kho về 0, không có hiện tượng *Race Condition*.

3.5.3 Giải pháp và Đánh giá

3.5.3.1 PostgreSQL (Pessimistic Locking) Để đảm bảo tính đúng đắn, nhóm sử dụng câu lệnh `SELECT ... FOR UPDATE`.

- **Cơ chế:** Transaction đầu tiên sẽ khóa dòng dữ liệu trong bảng `Inventory`. 99 transaction còn lại sẽ bị đưa vào trạng thái chờ (Wait Queue) cho đến khi transaction đầu tiên Commit hoặc Rollback.
- **Ưu điểm:** Đảm bảo tính nhất quán tuyệt đối (Strong Consistency), ngăn chặn hoàn toàn hiện tượng *Lost Update*.
- **Nhược điểm:** Thông lượng (Throughput) giảm do tắc nghẽn khóa (Lock contention).

3.5.3.2 MongoDB (Atomic Operations) Nhóm sử dụng tính năng cập nhật nguyên tử trên document.

- **Cơ chế:** Sử dụng toán tử `findOneAndUpdate` với điều kiện `{ $gt: 0 }`. WiredTiger engine sẽ thực hiện khóa document, kiểm tra điều kiện và cập nhật giá trị trong một thao tác duy nhất (Atomic).

- **Ưu điểm:** Hiệu năng xử lý cao nhờ cơ chế khóa mịn (Fine-grained locking) của WiredTiger.
- **Nhược điểm:** Khó khăn hơn trong việc quản lý giao dịch phức tạp liên quan đến nhiều Collection so với RDBMS (dù đã có hỗ trợ Multi-document Transactions từ v4.0).

3.5.4 Kết luận

Kiến trúc Hybrid là sự lựa chọn tối ưu: sử dụng **PostgreSQL** cho các giao dịch tài chính/tồn kho cần sự an toàn tuyệt đối của MVCC và Locking; sử dụng **MongoDB** cho các luồng dữ liệu log/review cần tận dụng tốc độ của Document-level locking.

4 DEMO ỨNG DỤNG

5 KẾT LUẬN

References