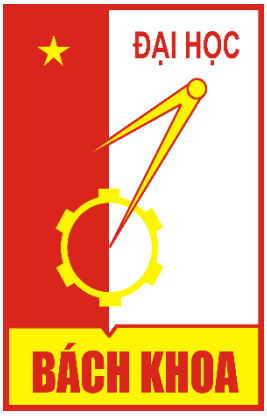
Trường Đại học Bách Khoa Hà Nội

Viện Công nghệ Thông Tin và Truyền Thông



ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

NGÀNH CÔNG NGHỆ THÔNG TIN

NGHIÊN CỨU VÀ CÀI ĐẶT THỬ NGHIỆM HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU THEO DẠNG CỘT

Sinh viên thực hiện: Đặng Việt Hưng

Người hướng dẫn: TS. Phạm Đăng Hải

Hà Nội, 12/2018

Lời cam kết

Họ và tên sinh viên: Đặng Việt Hưng MSSV : 20142139

Điện thoại liên lạc: 0396134566 Email: dangviethung096@gmail.com

Lớp: CNTT 2.2 K59 Hệ đào tạo: chính quy

Tôi – *Đặng Việt Hưng* – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS. Phạm Đăng Hải*. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

Hà Nội, ngày tháng năm

Tác giả ĐATN

Họ và tên sinh viên

Lời cảm ơn

Em xin gửi lời cảm ơn giáo viên hướng dẫn của em là TS. Phạm Đăng Hải đã giúp đỡ và hướng dẫn em trong quá trình làm đồ án tốt nghiệp.

Em cũng xin cảm ơn bạn bè và các thầy cô tại trường đại học Bách Khoa Hà Nội đã chỉ bảo hướng dẫn em giúp em tạo dựng các kiến thức nền tảng vững chắc trong nghề nghiệp sau này.

Tóm tắt

Kỷ nguyên số bắt đầu khi máy tính ra đời cùng với các thiết bị lưu trữ số. Ngày nay, thay vì việc dành cả một căn phòng cho việc lưu trữ các tài liệu giấy, với một thiết bị lưu trữ như USB có thể lưu trữ toàn bộ các dữ liệu giấy. Nhưng với việc sử lưu trữ theo dạng số lại phát sinh ra vấn đề là phải làm sao để quản lý các dữ liệu đó.

Từ các vấn đề trên, hệ quản trị CSDL đã ra đời. Khi máy tính càng trở nên phổ biến và được sử dụng rộng rãi, đặc biệt trong các doanh nghiệp, tổ chức, chính phủ v.v… Việc quản lý dữ liệu tại các đơn vị này đòi hỏi rất nhiều yêu cầu như: độ bảo mật, tốc độ xử lý, và các dữ liệu được lưu trữ có thể rất đặc thù. Do đó rất nhiều HQT CSDL đã ra đời nhằm đáp ứng các yêu cầu được đặt ra. Đặc biệt với sự xuất hiện internet, các dữ liệu được chia sẻ hàng ngày là rất lớn, việc quản lý nó ngày càng khó khăn.

Với mong muốn được học hỏi các kiến thức về CSDL và có thể thiết kế nên một HQT CSDL giúp đáp ứng các yêu cầu về lưu trữ ngày nay, em đã chọn đề tài ***"Hệ quản trị cơ sở dữ liệu theo dạng cột"*** làm đề tài tốt nghiệp của mình.

Mục lục

Danh mục hình vẽ

Danh mục bảng

Danh mục các từ viết tắt

API	Application Programming Interface Giao diện lập trình ứng dụng
CSDL	Cơ sở dữ liệu
HQT CSDL	Hệ quản trị cơ sở dữ liệu
SQL	Structured Query Language Ngôn ngữ truy vấn cấu trúc
RDBMS	Relational database management system Cơ sở dữ liệu quan hệ
IoTs	Internet of Things Vạn vật kết nối
SaaS	Software as a Service Phần mềm được coi như là một dịch vụ trong gói dành cho các đối tượng người dùng
ACID	Atomicity, Consistency, Isolation, Durability Nguyên tố, Nhất quán, Độc lập, Bền vững
MVCC	Multi-version concurrent control
IPC	Inter-Process Communication

	Kỹ thuật truyền thông giữa các tiến trình
fd	file descriptor
I/O	Input/Output Vào/Ra trong hệ thống Linux
XML	eXtensible Markup Language
OLAP	On-line analytical processing
OLTP	On-line transactional processing
JSON	JavaScript Object Notation
BSON	Binary JavaScript Object Notation

Danh mục thuật ngữ

NoSQL	Cơ sở dữ liệu phi quan hệ
Request	Thao tác yêu cầu tới data nhằm thực hiện một hành động nhất định
Consitency	Sự nhất quán trong cơ sở dữ liệu
Isolution	Sự độc lập trong cơ sở dữ liệu
Open address	Cách giải quyết đụng độ trong bảng băm

Chương 0 MỞ ĐẦU

Đặt vấn đề

Trước khi có sự xuất hiện của máy tính, việc lưu trữ các dữ liệu đều được thực hiện trên giấy, điều này mang lại nhiều sự khó khăn như: (i) tốn nhiều không gian để lưu trữ dữ liệu, (ii) thực hiện sao chép, tìm kiếm chỉnh sửa dữ liệu khó khăn, (iii) dễ bị mất mát dữ liệu.

Với sự xuất hiện của máy tính, việc lưu trữ dữ liệu trên máy tính đem lại rất nhiều lợi ích nổi bật như: (i) khả năng lưu trữ lớn chỉ với một không gian nhỏ, (ii) truy cập vào dữ liệu một cách nhanh chóng hiệu quả. Nhưng khi dữ liệu càng lớn thì việc quản lý, truy cập hoặc thực hiện các thao tác cơ bản như: (i) thêm, (ii) tìm kiếm, (iii) sửa, (iv) xóa càng trở nên khó khăn. Vì vậy từ những năm 60 của thế kỉ XX, khi mà máy tính được sử dụng nhiều trong các tổ chức, cơ quan thì các lập trình viên đã xây dựng các chương trình giúp quản lý dữ liệu. Và khái niệm "*cơ sở dữ liệu*" (CSDL), "*hệ quản trị cơ sở dữ liệu*" (HQT CSDL) được sử dụng nhằm ám chỉ các chương trình có khả năng quản lý dữ liệu. Các mô hình về việc xây dựng CSDL cũng được đưa ra như: (i) CSDL quan hệ, (ii) CSDL hướng đối tượng.

Với sự bùng nổ của internet và sự phổ cập của máy tính, các dữ liệu dễ dàng được số hóa và chúng ngày càng trở nên khổng lồ, đặc biệt với việc xuất hiện của các tập dữ liệu lớn (Big Data), điều này đặt ra các bài toàn cho việc khai thác và biến các dữ liệu này thành các tri thức giúp ích cho con người. Các "*hệ quản trị cơ sở dữ liệu*" được xây dựng theo các mô hình cũ không còn thích hợp nữa khi gây ra sự khó khăn như trong việc tìm kiếm hoặc phân tán dữ liệu. Vì vậy, nhằm đáp ứng các yêu cầu trên nhiều mô hình CSDL được ra đời. Trong đó phải kể đến các "*cơ sở dữ liệu phi quan hệ (NoSQL)*", đây được coi là CSDL thế hệ mới nhằm đáp ứng cho các bài toán về Big Data, về lưu trữ phân tán v.v…

Mục tiêu và phạm vi đề tài

Hiện nay, đã có một số "*cơ sở dữ liệu phi quan hệ*" được phát triển dựa và sử dụng rộng rãi như: (i) mongoDB, (ii) riak, (iii) Apache Cassandra, (iv) Amazon DynamoDB. Tuy nhiên hiện tại mỗi CSDL này sử dụng các cấu trúc lưu trữ là khác nhau và chúng cũng phù hợp đối với từng yêu cầu nhất định như: (i) hiệu năng cao, (ii) khả năng mở rộng, (iii) cấu trúc linh hoạt.

Tuy vậy CSDL NoSQL vẫn còn rất nhiều hạn chế như: (i) Không có được sự hỗ trợ tốt nhất do đa phần là mã nguồn mở, (ii) chưa thực sự được tin cậy đối với đa phần doanh nghiệp, (iii) hạn chế về tri thức nghiệp vụ của nhân viên kỹ thuật do đây là các công nghệ mới và thay đổi từng ngày, (iv) vấn đề với sự tương thích do chưa có được một tiêu chuẩn nhất quán.

Với mong muốn được nghiên cứu, tìm hiểu và xây dựng một trong các mô hình của CSDL NoSQL, em đã lựa chọn xây dựng một *cơ sở dữ liệu được theo dạng cột*, là một trong những mô hình sử dụng cấu trúc dữ liệu lưu trữ khác so với *cơ sở dữ liệu quan hệ* truyền thống.

Đề tài này sẽ tập trung vào việc thiết kế mô hình lưu trữ dữ liệu theo dạng cột, cung cấp các thao tác cơ bản đối với CSDL là (i) tìm kiếm, (ii) thêm, (iii) sửa, (iv) xóa. Và sử dụng cấu trúc server, client cho CSDL nhằm cung cấp khả năng đáp ứng đối với nhiều người dùng và từ đó có thể xây dựng lên CSDL phân tán.

Định hướng giải pháp

Trong các mô hình áp dụng vào CSDL NoSQL, *cơ sở dữ liệu dạng cột* sẽ giúp tăng tốc độ truy cập vào dữ liệu tại một trường nhất định giúp giảm chi phí cho việc truy cập vào các trường giá trị dư thừa trong CSDL.

Đồng thời, đề tài này sẽ sử dụng bảng băm để lưu trữ các trường giá trị nhằm tăng giảm thiểu tối đa việc tìm kiếm và thêm. Hàm băm được sử dụng là hàm băm doube hashing sử dụng open address để giải quyết vấn đề đụng độ được đề xuất bởi Donald Knuth. Chi tiết về kỹ thuật băm sẽ được giải thích chi tiết trong chương 3.

Với mô hình client-server cho database sẽ sử dụng API của Linux là epoll cho phía server nhằm đáp ứng được số lượng client nhiều và tăng tốc độ xử lý đối với các request tới từ các client.

Từ phương pháp trên, mô hình CSDL dạng cột sẽ được thiết kế và chạy thử nghiệm. Chi tiết các trường hợp thử nghiệm sẽ được trình bày tại chương 4.

CSDL tổ chức theo cột

Hiện này, rất nhiều bài toán trong thực tế cần các xử lý phân tích theo thời gian thực (OLAP – Online analytical processing) với độ phức tạp lớn và làm việc trên một tập dữ liệu có thể lên tới hàng Petabyte. Nhưng để giải quyết các bài toán đó, chỉ cần lấy ra một số dữ liệu của một trường nhất định, đối với đa phần CSDL truyền thống được tổ chức theo hàng, phải thực hiện duyệt CSDL mà chỉ lấy ra một phần dữ liệu của hàng. Điều này có dẫn tới việc đọc các dữ liệu không cần thiết và vừa làm giảm hiệu năng của CSDL.

Giải pháp được đưa ra khi ở đây là sử dụng một CSDL mà dữ liệu sẽ được phân bố theo **dạng cột** thay vì theo dạng hàng như truyền thống. Như vậy, khi thực hiện tìm kiếm mà chỉ quan tâm đến các giá trị trên một trường trong bảng, CSDL lưu trữ theo dạng cột giúp loại bỏ việc đọc các dữ liệu không cần thiết đồng thời tăng tốc độ tìm kiếm.

Sử dụng bảng băm trong việc lưu trữ

Với CSDL được tổ chức theo dạng cột tức là các giá trị trong cùng một trường trong bảng được lưu trữ thành một nhóm, thực hiện lưu trữ các giá trị theo thứ tự liên tiếp thì khi thực hiện các thao tác như tìm kiếm phải thực hiện duyệt và kiểm tra từng giá trị được lưu trong vùng nhớ của trường đó.

Để thực hiện tăng tốc trong việc tìm kiếm vùng nhớ lưu trữ các giá trị trong một trường, cần tổ chức các giá trị của một trường theo một dạng *cấu trúc dữ liệu* đặc biệt. Và cấu trúc dữ liệu được sử dụng ở đây là bảng băm. Bảng băm có nhưng ưu điểm như: (i) tốc độ tìm kiếm hiệu quả với một hàm băm thích hợp, (ii) dễ dàng triển khai với chi phí thấp.

Với việc lựa chọn bảng băm thì lại xuất hiện thêm vấn đề đó là xử lý *đụng độ* trong hàm băm. Để giải quyết vấn đề này, CSDL trong đề tài đã sử dụng double hashing với hàm băm được đề cập tới trong quyển sách *The Art of Computer Programming volume 3* của Donald Knuth. Chi tiết về việc triển khai hàm băm được mô tả trong chương 4 tại phần *thiết kế chi tiết*.

Thiết kế theo mô hình client-server

Với sự phát triển mạnh mẽ của công nghệ mạng, việc truyền tải thông tin giữa các máy tính tại các vị trí cách xa nhau không còn là rào cản. Từ đó để có thể tận dụng sức mạnh của nhiều máy tính tại các vị trí cách xa nhau, mô hình phân tán đã ra đời. Các CSDL phân tán có rất nhiều ưu điểm như: (i) tăng hiệu năng của hệ thống khi có nhiều máy chạy cùng lúc, (ii) tăng độ tin cậy vì khi một máy ngừng hoạt động đã có các máy còn lại hỗ trợ, (iii) phòng chống mất mát dữ liệu khi dữ liệu được lưu trữ ở trên nhiều máy, (iv) giảm chi phí vì sử dụng nhiều máy tính thay vì phải đầu tư một máy tính cực mạnh để đạt hiệu năng tương đương. Đối với CSDL được lưu trữ theo dạng hàng việc thiết kế theo hướng kiến trúc phân tán là khó khăn, ngược lại đối với CSDL theo dạng cột có thể đưa ra phương án phân tán hệ thống dễ dàng hơn.

Cùng với đó là sự phổ cập của máy tính và internet, các dịch vụ, chương trình thay vì phải thực hiện trên chính máy tính của người dùng thì bây giờ các máy tính server được tận dụng để tăng tốc hiệu năng, máy tính của người dùng tương đương với thiết bị đầu cuối chỉ thực hiện đặt ra yêu cầu và trả về kết quả.

HQT CSDL được thiết kế theo mô hình client-server, với phía server chịu trách nhiệm quản lý kết nối từ client, gọi đến các API của CSDL theo yêu cầu client và trả về kết quả. Phía client sẽ chịu trách nhiệm đọc các lệnh theo ngôn ngữ SQL, thông dịch nó ra thành lệnh và gửi đến server.

Vì vậy epoll API quản lý socket trên server đã được lựa chọn, ưu điểm là nó có thể quản lý số lượng kết nối lớn với hiệu năng cao. Chi tiết so sánh với các mô hình quản lý khác được đề cập tại chương 3.

Bố cục đồ án

Báo cáo đồ án tốt nghiệp này được tổ chức như sau.

Chương 0 là phần mở đầu, phần này đưa ra các mục "đặt vấn đề", "định hướng và giải pháp" và bố cục của đồ án

Chương 1 trình bày về HQT CSDL NoSQL. Định nghĩa, khái niệm, tính chất, so sánh ưu điểm nhược điểm của NoSQL, các mô hình dữ liệu của NoSQL.

Trong **chương 2**, em trình bày về thiết kế của HQT CSDL được xây dựng trong đề tài đi từ tổng quan đến chi tiết và các công nghệ được sử dụng trong. Gồm có epoll API, các hàm trong thư viện glibc, hàm băm được sử dụng.

Chương 3 đưa ra các đánh giá đối với HQT CSDL được thiết kế theo dạng cột và các kết quả đạt được của HQT CSDL được xây dựng.

Chương 4 đưa ra kết luận về HQT CSDL dạng cột, quá trình thực hiện thử nghiệm HQT CSDL dạng cột

Chương 1 HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU NoSQL

1.1 Khái niệm NoSQL

NoSQL là một thuật ngữ dùng làm tên gọi chung cho các HQT CSDL cung cấp các cơ chế lưu trữ và truy xuất dữ liệu sử dụng mô hình nhất quán lỏng lẻo hơn các HQT CSDL quan hệ truyền thống. Điều này có nghĩa là dữ liệu của các CSDL thuộc NoSQL sẽ không được xây dựng trên các bảng (table), không có các quan hệ khóa chính (primary key) – khóa ngoại (foreign key), và không sử dụng SQL làm ngôn ngữ truy vấn dữ liệu. NoSQL là CSDL phân tán được thiết kế dành cho các hệ thống cần khả năng mở rộng lưu trữ lớn, tính sẵn sàng (availability) và hiệu năng (performance) cao.

Từ NoSQL thường được hiểu và chấp nhận rộng rãi là Not only SQL (không chỉ có SQL), điều này nhằm nhấn mạnh rằng việc nó cũng hỗ trợ ngôn ngữ SQL.

Sự ra đời của các CSDL NoSQL đã tạo nên một cuộc cách mạng về công nghệ lưu trữ trong thời đại của Web 2.0, nơi mà nhu cầu về tính sẵn sàng và hiệu năng cao rất được xem trọng. Bên cạnh đó, các mạng dịch vụ cộng đồng phát triển mạnh mẽ cho phép người dùng tạo ra hàng tỷ nội dung trên web. Do đó, dữ liệu lớn rất nhanh vượt qua cả những giới hạn về phần cứng và cần giải quyết bằng bài toán phân tán.

1.2 Ứng dụng của NoSQL

1.2.1 Đáp ứng số lượng người dùng lớn (Big User)

Khi internet mới bắt đầu phổ biến, việc một ứng dụng có khoảng 1000 người dùng thì rất phổ biến, nhưng nếu nâng con số này lên 10.000 hay thậm chí 100.000 thì đó là chuyện

không tưởng. Ngày nay, hầu hết các ứng dụng mới đều được lưu trữ trên cloud và có luôn có sẵn trên Internet, nơi mà các ứng dụng này phải phục vụ cho người dùng trên toàn thế giới mọi lúc mọi nơi. Có khoảng hơn 2 tỷ người được kết nối với Internet và thời gian mà họ bỏ ra cho việc online mỗi ngày cũng tăng theo thời gian. Điều này tạo ra sự bùng nổ về số lượng người dùng đồng thời. Như vậy, sẽ không có gì lạ khi một ứng dụng có tới hàng triệu người khác nhau sử dụng mỗi ngày. Số lượng người dùng lớn kết hợp với thói quen sử dụng phong phú tạo nên nhiều mô hình dữ liệu khác nhau. Điều này dẫn đến sự cần thiết của các công nghệ CSDL có khả năng mở rộng dễ dàng. Với RDBMS, nhiều nhà phát triển ứng dụng đã gặp rất nhiều khó khăn hoặc thậm chí là không thể trong việc tạo ra các ứng dụng có khả năng mở rộng linh động trong khi vẫn duy trì được hiệu năng tốt như những gì người dùng mong đợi. Và họ đã tìm đến với các giải pháp NoSQL.

1.2.2 The Internet of Things

The Internet of Things có thể hiểu đơn giản là một mạng lưới gồm các đối tượng có khả năng kết nối Internet và tác động qua lại giữa các dịch vụ web. Thực tế cho thấy, số lượng các thiết bị có thể kết nối internet đồng thời có khả năng tạo ra dữ liệu đang ngày một gia tăng và trở nên phổ biến. Những thiết bị đang có mặt ở khắp mọi nơi và khá gần gũi với chúng ta, chúng có thể đang ở trong các nhà máy, công ty, bệnh viện, trường học, hay gia đình chúng ta, v.v… Chúng là điện thoại di động, máy tính bảng, các loại máy móc chuyên dụng, và còn rất nhiều nữa. Theo thời gian, những thiết bị này ngày càng được nâng cấp thành những thiết bị tiện dụng và thông minh hơn, một điển hình quen thuộc đó là những chiếc smart phone, tablet, v.v…

Tuy nhiên, dữ liệu từ xa thường thì nhỏ, bán cấu trúc (semi-structured), hoặc không có cấu trúc (unstructured) và được cập nhật liên tục. Điều này mang đến nhiều thách thức cho các RDBMS vốn đòi hỏi một lược đồ cố định (fixed shema) và dữ liệu có cấu trúc (structured data). Để giải quyết thách thức này, các doanh nghiệp có xu hướng đổi mới đang tin tưởng vào công nghệ NoSQL để mở rộng việc truy cập đồng thời cho hàng triệu thiết bị kết nối, lưu trữ hàng tỷ điểm dữ liệu đồng thời đáp ứng các yêu cầu về hiệu năng của cơ sở hạ tầng (infrastructure) và các hoạt động (operation) then chốt (mission-critical).

1.2.3 Big Data

Dữ liệu đang ngày một trở nên dễ dàng nắm bắt và truy cập thông qua bên thứ ba như Facebook, Google,. . . Thông tin cá nhân của người dùng, dữ liệu định vị địa lý, những nội dung do người dùng tạo là một vài ví dụ về các mảng dữ liệu không ngừng được mở rộng đã tạo nên một nguồn dữ liệu vô cùng phong phú. Không có gì phải ngạc nhiên khi các nhà phát triển tận dụng những nguồn tài nguyên sẵn có này để làm phong phú thêm những ứng dụng hiện có hay tạo ra những cái mới từ những nguồn tài nguyên đó. Việc sử dụng các nguồn dữ liệu đang làm thay đổi nhanh chóng bản chất của truyền thông, hành vi mua sắm, quảng cáo, giải trí và việc quản lý các mối quan hệ xã hội. Như vậy, một nhà phát nếu biết tận dụng nguồn dữ liệu sẵn có này một cách hợp lý vào việc phát triển ứng dụng sẽ dễ dàng thành công, ngược lại, việc nhanh chóng đi vào thất bại là điều dễ hiểu do không nắm bắt được thị hiếu của người dùng.

Việc lưu trữ và sử dụng các loại dữ liệu khác nhau tạo ra nhu cầu về loại CSDL cũng rất riêng. Tuy nhiên, các nhà phát triển muốn có một CSDL thật linh hoạt dễ dàng chứa bất kỳ kiểu dữ liệu mới nào mà họ muốn sử dụng và không bị gián đoạn bởi những thay đổi cấu trúc của nội dung dữ liệu từ nhà cung cấp dữ liệu thứ ba. Rất nhiều những dữ liệu mới là các dữ liệu không cấu trúc hoặc bán cấu trúc, do đó, các nhà phát triển cũng cần có một CSDL có khả năng lưu trữ hiệu quả. Tiếc là, việc định nghĩa một cách cứng nhắc, phương pháp tiếp cận dựa trên lược đồ (schema) được sử dụng trong các RDBMS làm cho nó không thể nhanh chóng kết hợp với các loại dữ liệu mới được, và kết quả là không phù hợp với dữ liệu không cấu trúc và bán cấu trúc.

Như vậy, với sự tăng lên nhanh chóng của dữ liệu thì việc xử lý dữ liệu lớn đóng vai trò hết sức quan trọng đặc biệt là với các loại dữ liệu không cấu trúc và bán cấu trúc. Điều này đã đặt ra nhiều thách thức cho các RDBMS truyền thống, và dễ dàng thấy rằng, với các ràng buộc về cấu trúc dữ liệu, sử dụng các lược đồ cố định thì RBDMS đã trở nên vô cùng khó khăn để xử lý lượng dữ liệu cực lớn mà đa phần là không có cấu trúc hoặc bán cấu trúc đang gia tăng hàng ngày, hàng giờ. Trong khi đó, NoSQL cung cấp mô hình dữ liệu tốt hơn làm cho đơn giản hóa việc giao tiếp giữa CSDL và ứng dụng.

1.2.4 Cloud

Trước đây, hầu hết các ứng dụng dành cho người dùng và các ứng dụng trong kinh doanh đều là các ứng dụng đơn người dùng và được cài trên một máy tính cục bộ. Các ứng dụng đa người dùng phần lớn sử dụng kiến trúc 2 tấng (2-tier) hay client-server chạy bên trong các bức tường lửa và hỗ trợ giới hạn số lượng người sử dụng.

Ngày nay, hầu hết các ứng dụng mới sử dụng kiến trúc 3 tầng, được lưu trữ và thực thi trên những đám mây, hỗ trợ một số lượng lớn người sử dụng đồng thời. Cùng với sự thay đổi này trong kiến trúc phần mềm, mô hình kinh doanh mới như là SaaS và các mô hình dựa trên quảng cáo (Advertising-based) cũng đã trở nên phổ biến hơn. Trong kiến trúc 3 lớp, ứng dụng được truy cập thông qua một trình duyệt web hoặc một ứng dụng di động được kết nối tới Internet. Trong đám mây, một cân bằng tải (load balancing) sẽ điều khiển lưu lượng truy cập đến một tầng có khả năng mở rộng theo chiều ngang (scale-out) của máy chủ web/ứng dụng mà xử lý logic của ứng dụng. Kiến trúc mở rộng theo chiều ngang tại tầng ứng dụng làm việc rất tốt. Với mỗi 10.000 (hay tùy ý) người dùng đồng thời, ta chỉ cần thêm một server khác đến tầng ứng dụng để chịu tải. Tại tầng CSDL, CSDL quan hệ từng được lựa chọn nhiều nhất, và việc sử dụng chúng ngày càng gặp nhiều vấn đề rắc rối. Bởi vì, dữ liệu của các CSDL quan hệ thường được tập trung và có xu hướng mở rộng theo chiều dọc (scale up) hơn là mở rộng theo chiều ngang (scale out). Điều này dẫn đến các CSDL rất khó phù hợp với các ứng dụng yêu cầu khả năng mở rộng động (dynamic scalability) dễ dàng. Các công nghệ NoSQL đã xây dựng với mục đích phân tán ngay từ lúc ban đầu (ground up) cộng với các kỹ thuật mở rộng theo chiều ngang, vì vậy tương thích tốt với bản chất phân tán cao của kiến trúc 3 tầng của Internet.

1.3 Các tính chất của NoSQL

1.3.1 Dữ liệu luôn sẵn có

Môi trường CSDL NoSQL thường được xây dựng với kiến trúc phân tán và nó được dự phòng cho cả chức năng và dữ liệu. Nếu một hoặc nhiều nodes bị sập, có các node khác trong hệ thống có thể tiếp tục thao tác mà không để mất dữ liệu. Điều này cho thấy khả năng

chịu lỗi của hệ thống. Bằng cách này, môi trường CSDL NoSQL có khả năng duy trì tính sẵn có của dữ liệu tại một vị trí duy nhất, thông qua các trung tâm dữ liệu và trong cloud. Khi triển khai thích hợp, các hệ CSDL NoSQL có thể cung cấp khả năng mở rộng lớn cùng với hiệu năng cao mà không bao giờ bị sập. Điều này đặc biệt có lợi trong bất kì hoạt động nâng cấp hoặc cập nhật hệ thống mà cần không phải có một CSDL ngoại tuyến.

1.3.2 Độc lập vị trí

"Độc lập vị trí" có nghĩa là khả năng đọc và ghi vào CSDL bất kể thao tác Input/Output (I/O) được xảy ra ở đâu và có bao nhiêu bản sao chép từ vị trí đó, như vậy, nó có thể sẵn có đối với những người dùng tại các vị trí khác. Chức năng này rất khó để thiết kế cho CSDL quan hệ. Một vài kĩ thuật có thể được sử dụng như kiến trúc master/salve hay database sharding (phân mảnh CSDL) đôi lúc có thể có được sự độc lập vị trí trong thao tác đọc nhưng không thể thao tác ghi lại là vấn đề khác, đặc biệt khi kích thước dữ liệu lớn. Một trong những trường hợp thể hiện tính ưu việt của đọc lập vị trí là khi có nhiều dịch vụ, bao gồm cả dịch vụ khách hàng tại nhiều khu vực địa lý khác nhau và cần lưu trữ cục bộ những thông tin này tại các khu vực đó nhằm tăng tốc độ truy cập.

1.3.3 Khả năng hỗ trợ mô hình Transaction hiện đại

Lý thuyết về transactions xuất hiện để được thay đổi trong thời đại Internet, và nó được chứng minh răng ACID transactions không còn được xem là một yêu cầu bắt buộc trong các hệ thống CSDL. Nhận định này có vẻ cực đoan khi mà toàn vẹn giao dịch là một trong những đặc trưng của mọi hệ dữ liệu, đặc biệt là những hệ thống yêu cầu thông tin trung thực và chính xác. Tuy nhiên, điều này nói đến không phải là gây nguy hiểm cho dữ liệu, mà là một cách thức mới mà các ứng dụng hiện đại đảm bảo tính nhất quán trên toàn hệ thống phân tán.

Tính nhất quán trong HQT CSDL quan hệ được thực thi thông qua các ràng buộc khóa ngoại. Để đảm bảo tính toàn vẹn (integrity) của dữ liệu, hầu hết các HQT CSDL cổ điển đều dựa trên các giao dịch (transaction). Điều này đảm bảo tính nhất quán (consistency) trong tất cả các trường hợp quản lý dữ liệu. Những đặc tính về giao dịch này được biết đến chính là ACID (Atomicity, Consistency, Isolation, Durability).

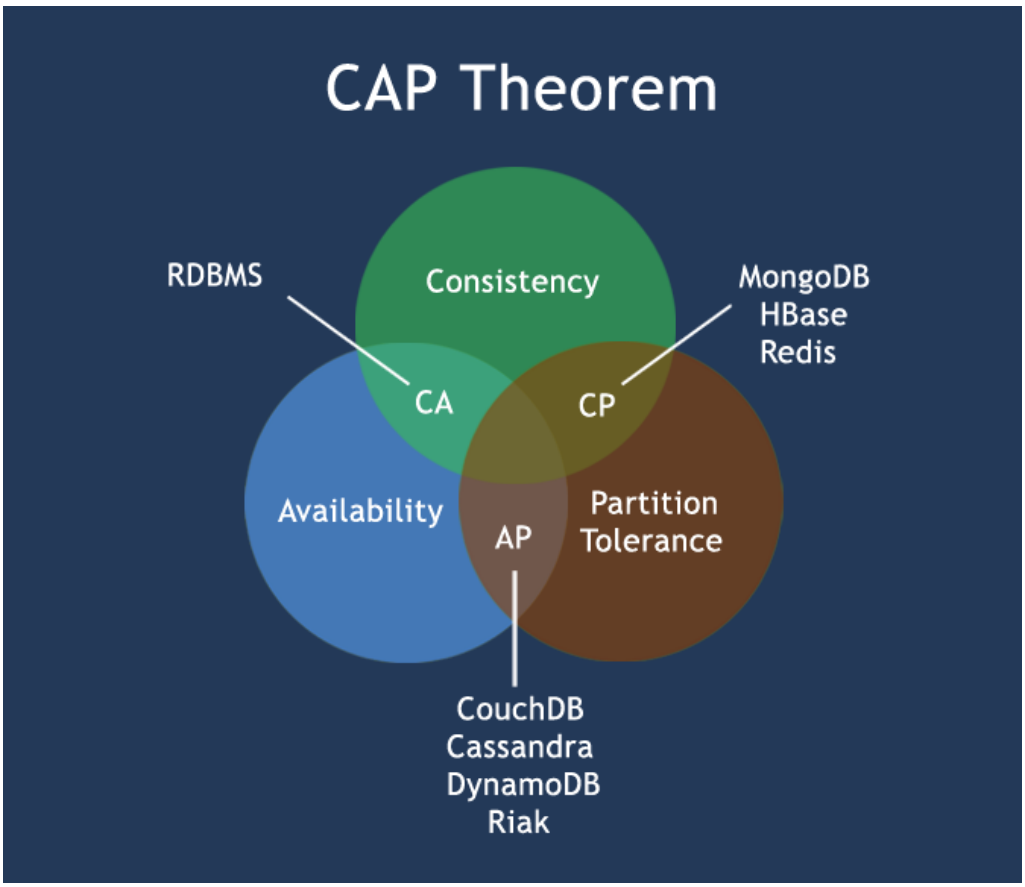
Việc mở rộng theo chiều ngang (scale out) của các hệ thống tuân thủ theo ACID đã thể hiện một số vấn đề. Những xung đột đang phát sinh giữa các khía cạnh khác nhau của tính sẵn sàng cao trong các hệ thống phân tán mà không thể giải quyết hoàn toàn được, điều này được biết với định lý CAP. Định lý CAP nói rằng trong một hệ CSDL phân tán, bạn chỉ có thể đạt được nhiều nhất hai trong ba tính chất: Consistency, Availability, Partition tolerance

Trong đó:

- *Strong Consistency* (tính nhất quán mạnh mẽ): nghĩa là cách mà hệ thống ở trong một trạng thái thống nhất sau khi thực hiện một thao tác. Một hệ thống phân tán thường được coi là nhất quán nếu sau khi cập nhật hoạt động của một vài thao tác ghi thì tất cả các thao tác đọc dữ liệu sẽ thấy được dữ liệu đã được cập nhật đó.
- *High Availability* (tính sẵn sàng cao): nghĩa là một hệ thống được thiết kế và hiện thực sao cho khi có vấn đề xảy ra với một node trong một cụm (cluster) chẳng hạn như bị hư hỏng hay được tạm ngưng để nâng cấp thì các theo tác đọc ghi dữ liệu vẫn cho phép tiếp tục thực hiện.
- *Partition Tolerance*: nghĩa là hệ thống của một cụm bất kỳ vẫn hoạt động bình thường ngay cả khi đường truyền giữa hai node bị ngắt và không liên lạc được.

1.3.4 Các mô hình dữ liệu linh hoạt

Một trong những nguyên nhân chính để cuyển sang hệ QTCSDL NoSQL là do những mô hình dữ liệu linh hoạt hơn đến từ các hệ QTCSDL NoSQL. Mô hình CSDL quan hệ dựa trên mối liên hệ giữa các bảng, trong đó các bảng được xác định bởi cấu trúc cột. Tất cả chúng được tổ chức chặt chẽ và đồng nhất trong một database schema. Vấn đề bắt đầu nảy sinh với mô hình quan hệ xung quanh khả năng mở rộng và hiệu năng khi cố gắng quản lý một lượng lớn dữ liệu. Mô hình NoSQL - thường được gọi là schema-less - có thể hỗ trợ rất nhiều những trường hợp sử dụng này. Một CSDL NoSQL có thể chấp nhận mọi loại dữ liệu - có cấu trúc, bán cấu trúc và không cấu trúc- một cách dễ dàng hơn một CSDL quan hệ, khi mà CSDL quan hệ luôn dựa vào schema được xác định sẵn. Đặc điểm này của CSDL quan hệ có thể trở thành ựự cản trở cho tính linh hoạt bởi cấu trúc được định nghĩa trước một cách cứng nhắc xác định cấu trúc và cách thức tổ chức dữ liệu của RDBMS. Trong rất nhiều các ứng dụng ngày nay có khả năng thực thi các quy định về việc sử dụng dữ liệu do chính dữ liệu đó xác định, tạo một nền tảng schema-less DB là một cách khả thi.



Hình 1. Định lý CAP

Cuối cùng là vấn đề về hiệu năng đối với mô hình RDBMS, đặc biệt là khi các "wide rows" có liên quan và thực hiện nhiều thao tác update có thể làm giảm hiệu năng hệ thống một cách nghiêm trọng. Tuy nhiên, mô hình dữ liệu NoSQL có thể dễ dàng xử lý các tình huống này và mang lại hiệu năng rất cao cho cả các thao tác đọc và ghi.

1.3.5 Kiến trúc tốt hơn

Một trong những lý do để dùng CSDL NoSQL là khi cần một kiến trúc thích hợp hơn cho các ứng dụng tương ứng. Một vài, nhưng không phải tất cả, các giải pháp NoSQL có thể cung cấp các kiến trúc hiện đại có thể giải quyết các loại ứng dụng đòi hỏi cao khả năng mở rộng, phân tán và tính sẵn sàng liên tục. Hỗ trợ trung tâm dữ liệu, và thông dụng hơn là hỗ trợ nhiều trung tâm dữ liệu là một trong những trường hợp bắt buộc phải sử dụng môi trường NoSQL.

1.4 Ưu điểm và nhược điểm của NoSQL

1.4.1 Ưu điểm

Open source: các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, đặc biết là việc sử dụng miễn phí. Những lợi ích khác là phần mềm nguồn mở có xu hướng sẽ là tin cậy hơn, an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữu độc quyền.

Khả năng mở rộng linh hoạt: NoSQL thay thế câu thần chú cũ của các nhà quản trị CSDL về "mở rộng phạm vi" với một thứ mới "mở rộng ra ngoài". Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển nhiều tải dữ liệu hơn, thì CSDL NoSQL cho phép một công ty phân tán tải qua nhiều máy chủ khi mà tải gia tăng.

Các CSDL NoSQL khác nhau cho những dự án khác nhau: MongoDB và Redis là những lựa chọn tốt cho việc lưu trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên, như một số đếm truy cập web chẳng hạn. Hadoop, một CSDL dạng tự do, phân tán làm tốt công việc lưu trữ các dữ liệu lớn như các con số thống kê thời tiết hoặc công việc phân tích nghiệp vụ.

NoSQL được các hãng lớn sử dụng: các công ty như Amazon, BBC, Facebook và Google dựa vào các CSDL NoSQL.

NoSQL phù hợp với công nghệ đám mây: NoSQL và đám mây là một sự trùng khớp tự nhiên. Các máy chủ ngày nay là không đắt và có thể dễ dàng mở rộng phạm vi được theo yêu cầu có sử dụng một dịch vụ như là Amazon EC2. Giống như tất cả công nghệ đám mây, EC2 dựa vào ảo hóa. Liên kết yếu của ảo hóa là sự thực thi của I/O, với bộ nhớ và CPU

các các kết nối mạnh. Các CSDL NoSQL hầu hết sử dụng bộ nhớ qua đĩa như là vị trí ghi đầu tiên. Vì thế ngăn ngừa được sự thực thi không ổn định của I/O. Và vì NoSQL lưu trữ dữ liệu thường thúc đẩy được tính mở rộng phạm vi theo chiều ngang thông qua việc ngăn chia, chúng có khả năng tận dụng được việc cung cấp mềm dẻo của đám mây.

1.4.2 Nhược điểm

Hỗ trợ không đồng đều cho các doanh nghiệp: trong khi các nhà cung cấp chủ chốt của các RMDBMS như SQL Server, Oracle, IBM,. . . thường đưa ra sự hỗ trợ tốt cho khách hàng thì các nhà cung cấp nguồn mở mới thành lập không thể được mong đợi sẽ cung cấp hỗ tốt hơn.

Chưa đủ "chín" cho các doanh nghiệp: dù chúng đã được triển khai tại một số công ty lớn thì các CSDL NoSQL vẫn đối mặt với một vấn đề về sự tin cậy chính với nhiều doanh nghiệp. Vấn đề lớn của NoSQL là thiếu về độ chín muồi và các vấn đề về tính không ổn định, trong khi đó tính chín muồi, hỗ trợ đầy đủ chức năng và tính ổn định của các RDBMS được thiết lập đã từ lâu.

Những hạn chế về tri thức nghiệp vụ: các CSDL NoSQL không có nhiều sự đeo bám tới các công cụ BI thường được sử dụng, trong khi những yêu cầu và phân tích hiện đại đơn giản nhất thì cũng liên quan khá nhiều tới sự tinh thông về lập trình.

Thiếu sự tinh thông: tính mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị biết công nghệ này. Như vậy sẽ rất khó khăn cho các công ty tìm người với sự tinh thông phù hợp.

Những vấn đề về tính tương thích: không giống như các CSDL quan hệ, các CSDL NoSQL chia sẻ ít theo cách thức của các tiêu chuẩn. Mỗi CSDL NoSQL có các giao diện lập trình ứng dụng API riêng của mình, các giao diện truy vấn riêng,. . . Sự thiếu hụt các tiêu chuẩn sẽ gây ra rất nhiều khó khăn khi chuyển từ một nhà cung cấp này sang một nhà cung cấp khác nếu có nhu cầu.

1.5 Mô hình nhất quán (consistency)

Một trong những yếu tố tạo nên sức mạnh cho NoSQL DB là sự thoát khỏi sự nhất quán ACID chặt chẽ. Mọi người thường có suy nghĩ rằng các loại CSDL NoSQL chỉ cung cấp weak consistency hoặc nhiều nhất là eventual consistency. Đây là một trong những hiểu lầm cơ bản với những hệ CSDL NoSQL. Các hệ thống CSDL NoSQL cung cấp một phạm vi consistency, bao gồm cả strict consistency.

Các mô hình nhất quán ảnh hưởng lớn tới sự song song hóa của CSDL - khả năng truy cập dữ liệu đồng thời từ người dùng - và tính sẵn sàng. Hiểu được cách hệ CSDL xử lý với consistency là cần thiết để xác định loại CSDL nào phù hợp với ứng dụng.

1.5.1 Các loại nhất quán

Nhất quán (Consistency) có thể bao gồm các loại sau:

Nhất quán với các người dùng khác: nếu hai người dùng cùng truy cập CSDL vào một thời điểm, họ có thể cùng thấy chung một dữ liệu không. RDBMS sẽ cố gắng đảm bảo điều đó, trong khi CSDL NoSQL thường có những lựa chọn "thoải mái" hơn. Nhất quán trong một phiên làm việc Dữ liệu có duy trì một ràng buộc logic trong bối cảnh của một phiên làm việc với CSDL không? Ví dụ: nếu chúng ta cập nhật một hàng, và sau đó đọc lại nó, chúng ta có thể thấy các thay đổi đó hay không.

Nhất quán trong một request: có phải một request đơn lẻ là nhất quán cục bộ (internally coherent)? Ví dụ chúng ta đọc tất cả các hàng trong một bảng quan hệ, chúng ta muốn đảm bảo để trạng thái của bảng là trong chính thời điểm hiện tại. Sẽ không có bất cứ sự cập nhật dữ liệu nào có thể xay ra khi chúng ta bắt đầu câu truy vấn.

Nhất quán với thực tại: có phải dữ liệu chính xác trong thời điểm hiện tại luôn được tham chiếu tới? Ví dụ, trong một giao dịch của ngân hàng, số dư tài khoản phải đạt được nhất quán trong suốt khoảng thời gian giao dịch. Tức là phải đưa ra được số dư chính xác của tài khoản theo thời gian thực trong suốt thời gian giao dịch, kể cả khi có giao dịch khác xem vào giữa giao dịch làm làm thay đổi số dư của tài khoản.

1.5.2 ACID và MVCC

RDBMS yêu cầu sự nhất quán được thực hiện bằng cách sử dụng các mẫu kiến trúc: ACID transactions và multi-version concurrency control. Chúng ta đã nói đến ACID trước đây, chúng bao gồm Atomic, Consitent, Isolated và Durable:

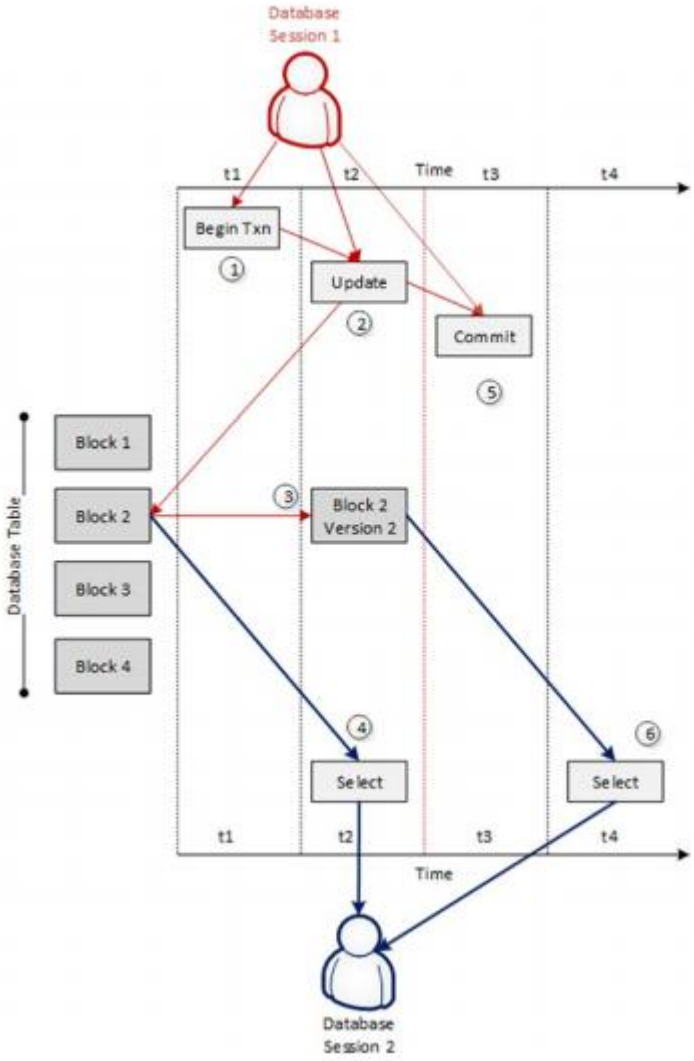
- **Atomicity** (tính nguyên tố) là một giao dịch có nhiều hoạt động thì hoặc là toàn bộ hoạt động được thực hiện thành công hoặc không hoạt động nào được thực hiện.

- **Consistency** (tính nhất quán): một giao dịch hoặc là sẽ tạo ra trạng thái mới và hợp lệ cho dữ liệu, hoặc trong trường hợp có lỗi sẽ chuyển toàn bộ dữ liệu về trạng thái trước khi thực hiện giao dịch.
- **Isolation** (tính cô lập): một giao dịch đang được thực thi phải đảm bảo được tách biệt bởi giao dịch khác.
- **Durability** (tính bền vững): dữ liệu được xác nhận sẽ được hệ thống lưu lại sao cho ngay cả trong trường hợp hỏng móc hoặc có lỗi hệ thống, dữ liệu vẫn phải được đảm bảo trong tình trạng chuẩn xác.

Cách dễ nhất để cài đặt ACID là sử dụng lock. Tuy nhiên, sử dụng lock có thể dẫn đến sự tranh chấp cao khó chấp nận và khả năng song song thấp.

Để cung cấp ACID consistency mà không cần sử dụng thừa các kóa, RDBMS cung cấp mô hình MVVC. Trong mô hình này, nhiều bản sao chép của dữ liệu được gắn nhãn với mốc thời gian hoặc thay đổi định danh để cho phép CSDL xây dựng một snapshot của CSDL tại thời một thời điểm. Bằng cách này, MVCC có thể cung cấp cho giao dịch một sự cô lập và nhất quán trong khi vẫn tối đa hóa khả năng song song.

Ví dụ, trong MVVC, nếu một bảng dữ liệu đang được cập nhật trong khoảng thời gian bắt đầu đọc bảng cho đến khi kết thúc phiên làm việc, CSDL sẽ sử dụng phiên bản cũ của dữ liệu ở trong bảng để đảm bảo rằng dữ liệu thấy được là một phiên bản nhất quán của dữ liệu. Điều này cũng có nghĩa rằng, cho đến khi giao dịch được commit, các phiên làm việc khác sẽ không nhìn thấy sự thay đổi của giao dịch mà chỉ thấy phiên bản dữ liệu cũ hơn. Những dữ liệu này cũng dùng để rollback lại giao dịch nếu chúng thực hiện không thành công. Ưu điểm lớn nhất của MVVC là giảm quá tải lock.



Hình 2. Multi-version concurrency control (MVCC)

MVVC sử dụng timestamp để xác định phiên bản của dữ liệu. Tuy nhiên phần lớn các CSDL sử dụng global transaction ID thay thế cho timestamp. Trong Oracle, nó được gọi là system change number (SCN) và trong SQL Server được gọi là transaction sequence number.

Sequence number này tự động tăng khi một transaction được bắt đầu, và nó được lưu trữ trong cấu trúc thông tin của những hàng (rows, database blocks) đã được modified. Khi có một yêu cầu truy vấn, hệ QTCSDL sẽ tìm kiếm nhữngdòngdòng có sequence number nhỏ hơn hoặc bằng so với sequence number hiện tại (khi truy vấn bắt đầu được thực thi), khi bắt gặp một dòng có sequence number cao hơn, hệ QTCSDL sẽ hiểu rằng nó phải tìm kiếm một version cũ hơn của dòng đó.

1.5.3 Two phase commit

MVVC hoạt động cùng với ACID transaction để dungung cấp sự cô lập giữa các giao dịch trong ùngùng một hệ thống. Giao dịch giữa các CSDL trong hệ thông RDBMS bằng cách sử dụng giao thức Two-phase commit.

Trong quá trình xác nhận, nếu một trong các nút của hệ thống phủ nhận giao dịch thì toàn bộ giao dịch sẽ bị hủy bỏ. Giao dịch chỉ được hoàn thành khi tất cả các nút của hệ thống xác nhận nó. Quá trình xác nhận bao gồm 2 pha là:

- Gửi yêu cầu: nút điều phối (coordinator) gửi câu truy vấn yêu cầu xác nhận giao dịch tới tất cả các nút tham gia, sau đó đợi trả lời từ các nút đó.
- Xác nhận giao dịch: khi nút điều phối nhận được trả lời đồng ý của tất cả các nút tham gia, nó sẽ gửi lại thông điệp xác nhận tới tất cả các nút tham gia một lần nữa để hoàn tất việc giao dịch, và giao dịch đó được xác nhận thành công. Trong trường hợp nút điều phối nhận được ít nhất một trả lời phủ định giao dịch từ một nút tham gia nào đó, hoặc nếu sau khoảng thời gian nhất định vẫn chưa nhận được toàn bộ trả lời từ các nút tham gia, thì nút điều phối sẽ gửi thông điệp hủy bỏ việc xác nhận tới tất cả các nút tham gia, và việc xác nhận giao dịch thất bại.

1.5.4 Các mức độ consistency

Trong các CSDL NoSQL, gần như không có hệ thống nào cung cấp sự nhất quán dữ liệu giữa nhiều câu truy vấn: ví dụ ta có thể xóa một dòng trong một bảng và thêm một dòng khác vào bảng như là một atomic operation. Tuy vậy, đối với các single-object operation, các hệ CSDL NoSQL lại cung cấp nhiều các mức độ nhất quán mà chúng ta có thể mong đợi. Một số mức độ đáng chú ý là:

Strict consistency: Một yêu cầu đọc luôn trả về giá trị mới nhất.

Causal consistency: Yêu cầu đọc có thể không trả về giá trị gần nhất, tuy nhiên nó sẽ không trả về giá trị "out of sequence". Ví dụ một phiên làm việc được tạo ra với mục đích update A,B và C thì một phiên làm việc khác chỉ có thể thấy A,B,C cùng được update hoặc chưa.

Monotonic consistency: Một phiên làm việc sẽ không bao giờ có thể thấy được dữ liệu được phục hồi từ trước thời điểm nó bắt đầu. Một khi bắt đầu đọc dữ liệu, chúng ta sẽ không bao giờ thấy các phiên bản trước đây của dữ liệu so với thời điểm bắt đầu phiên làm việc.

Eventual consistency: Hệ thống có thể không nhất quán tại một thời điểm nào đó, nhưng mỗi operation độc lập cuối cùng đều sẽ được nhất quán. Nếu tất cả các thao tác cập nhật dữ liệu đều dùng lại, hệ thống cuối cùng cũng sẽ đạt được trạng thái nhất quán.

Weak consistency: Hệ thống sẽ không đảm bảo rằng hệ thống sẽ trở nên nhất quán, ví dụ, nếu lỗi server, một thao tác cập nhật dữ liệu có thể bị hỏng.

Thông thường, các hệ CSDL NoSQL đều cài đặt strict hoặc eventual consistency trong khi RDBMS cung cấp ACID consistency.

1.5.5 Các cấp độ Isolation

Isolation level là một thuộc tính của transaction, qui định mức độ cô lập của dữ liệu mà transaction có thể truy nhập vào khi dữ liệu đó đang được cập bởi một transaction khác. Khi một transaction cập nhật dữ liệu đang diễn ra, một phần dữ liệu sẽ bị thay đổi (ví dụ một số bản ghi của bảng được sửa đổi hoặc bị xóa bỏ, một số được thêm mới), vậy các transaction hoặc truy vấn khác xảy ra đồng thời và cùng tác động vào các bản ghi đó sẽ diễn ra thế nào? Chúng sẽ phải đợi đến khi transaction đầu hoàn thành hay có thể thực hiện song song, kết quả dữ liệu nhận được là trong khi hay sau khi cập nhật? Bạn có thể điều khiển những

hành vi này thông qua việc đặt isolation level của từng transaction.

Read Uncommited: khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy nhập vào các bản ghi đang được cập nhật bởi một transaction khác và nhận được dữ liệu tại thời điểm đó mặc dù dữ liệu đó chưa được commit (uncommited data). Nếu vì lý do nào đó transaction ban đầu rollback lại những cập nhật, dữ liệu sẽ trở lại giá trị cũ. Khi đó transaction thứ hai nhận được dữ liệu sai (dirty read).

Read commited: transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong. Vì thế nó tránh được dirty read như ở mức trên.

Nó được cài đặt trong DBMS bằng lock-based concurrency control bằng cách khóa write locks (trên các dữ liệu đã được selected) cho đến khi kết thúc transaction, nhưng read locks

được giải phóng ngay sau khi lệnh SELECT được thực thi xong. Vì vậy có thể xảy ra non-repeatable read.

Repeatable read: mức isolation này hoạt động như mức read commited nhưng nâng thêm một nấc nữa bằng cách ngăn không cho transaction ghi vào dữ liệu đang được đọc bởi một transaction khác cho đến khi transaction khác đó hoàn tất.

Nó được cài đặt trong DBMS bằng cách giữ cả khóa write locks và read locks (trên các dữ liệu được select) cho đến khi kết thúc transaction, vì vậy không xảy ra non-repeatable read. Tuy nhiên có thể xảy ra trường hợp hai truy vấn trong cùng một transaction có thể trả về hai kết quả khác nhau (phantom read) do range locks không được quản lý.

Serializable: mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới.

Nó được cài đặt bằng cách giữ cả khóa read locks và write locks cho đến khi kết thúc transaction. Range locks cũng được khóa cho khi có câu truy vấn SELECT sử dụng mệnh đề clause.

Snapshot: mức độ này cũng đảm bảo độ cô lập tương đương với Serializable, nhưng nó hơi khác ở phương thức hoạt động. Khi transaction đang select các bản ghi, nó không khóa các bản ghi này lại mà tạo một bản sao (snapshot) và select trên đó. Vì vậy các transaction khác insert/update lên các bản ghi đó không gây ảnh hưởng đến transaction ban đầu. Tác dụng của nó là giảm blocking giữa các transaction mà vẫn đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên cái giá kèm theo là cần thêm bộ nhớ để lưu bản sao của các bản ghi, và phần bộ nhớ này là cần cho mỗi transaction do đó có thể tăng lên rất lớn.

Các mức isolation từ 1 – 4 kể trên tăng theo thứ tự mức độ cô lập dữ liệu, giúp tăng tính toàn vẹn dữ liệu và nhất quán của transaction. Đồng thời nó cũng tăng thời gian chờ lẫn nhau của các transaction. Khi càng lên mức cao, đòi hỏi về tính toàn vẹn dữ liệu càng cao và càng có nhiều tình huống một transaction ngăn không cho các transaction khác truy nhập vào dữ liệu mà nó đang thao tác. Do đó nó càng tăng tình trạng locking và blocking trong database (ngoại trừ với snapshot thì tăng lượng bộ nhớ cần sử dụng). Hiệu năng của hệ thống do đó bị giảm đi. Thông thường, mức isolation read commited (mức mặc định) là phù hợp trong đa số các ứng dụng. Có thể một vài chức năng quan trọng (ví dụ chức năng ở trang admin update dữ

liệu có ảnh hưởng đến toàn hệ thống) bạn cần tính toàn vẹn cao và phải chọn mức isolation cao hơn. Hoặc có những chức năng cần ưu tiên tốc độ thực hiện và có thể chấp nhận một chút dữ liệu không nhất quán, bạn có thể đặt xuống mức read uncommited. Bảng dưới đây tóm tắt các tính năng của từng mức isolation.

Bảng 1. Tính năng của từng mức Isolution

Mức Isolation	Dirty read	Non-repeatable read	Phantom read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable	No	No	No
Snapshot	No	No	No

1.6 Các mô hình dữ liệu NoSQL

1.6.1 Key-value store

Key-value stores là kiểu lưu trữ đơn giản nhất trong các loại CSDL NoSQL đồng thời nó cũng là kiểu lưu trữ cho tất cả các HQT CSDL NoSQL. Thông thường, các HQT CSDL key-value lưu trữ dữ liệu dưới dạng key (là một chuỗi duy nhất) liên kết với value có thể ở dạng chuỗi văn bản đơn giản hoặc các tập, danh sách dữ liệu phức tạp hơn. Quá trình tìm kiếm dữ liệu thường sẽ được thực hiện thông qua key, điều này dẫn đến sự hạn chế về độ chính xác.

Các API được cung cấp cho việc truy vấn dữ liệu của các CSDL NoSQL thường cũng rất đơn giản. Về cơ bản, hầu hết các các CSDL NoSQL sẽ có các API sau:

```
void put(U8bit * key, U8bit * data);

U8bit * get(U8bit * key);

Void remove(U8bit * key);
```

Students	
Key	Value
1	Name: Jean Grey DateOfBirth: 19-05-1963 IDCard: 1234567 PlaceOfOrigin: Austin Country: USA AcademicProgram_ID:1

2	Name: Scott Summers
	DateOfBirth: 12-10-1968
	IDCard: 765414A
	Supervisor: {
	Name: Emma Frost
	DateOfBirth: 1-1-1936
	IDCard: 222222
	}

Professors	
Key	Value
1	Name: Charles Xavier DateOfBirth: 13-07-1940 IDCard: 111111 PlaceOfOrigin: Mirfield Country: UK
2	Name: Emma Frost DateOfBirth: 1-1-1936 IDCard: 222222

Hình 3. Key-Value store

Tuy nhiên, việc triển khai thực tế của các CSDL key-value thường rất phức tạp do cấu trúc lưu trữ quá đơn giản.

Với kiểu lưu trữ này, ta sẽ rất dễ dàng và nhanh chóng truy xuất được thông tin của một người thông qua key, nhưng không hề đơn trong việc xử lý những dữ liệu phức tạp. Dễ thấy rằng ý tưởng của các HQT CSDL dạng key-value là đơn giản hóa việc lưu trữ dữ liệu, nghĩa là không cần quan tâm đến nội dung cần lưu trữ là gì. Nói cách khác, chúng lưu trữ thông tin mà không cần phải xác định lược đồ. Việc để biết được dữ liệu thực tế như thế nào sẽ được định nghĩa (mang tính tham khảo) ở phía client. Điều này làm cho phương pháp lưu trữ dữ liệu với key-value trở nên đơn giản hơn rất nhiều trong việc xây dựng cũng như khả năng mở rộng là cực kỳ linh động và hiệu suất cho các thao tác truy vấn dữ liệu cũng cực nhanh.

Key-value stores làm việc theo cách rất khác so với RDB. RDBs định nghĩa các cấu trúc dữ liệu được lưu trữ trong một tập các bảng chứa các trường được đã được định nghĩa cùng với kiểu dữ liệu. Chỉ định ra kiểu dữ liệu giúp RDB cho phép RDB thực hiện một số các thao tác tối ưu hóa. Ngược lại, key-value store lưu trữ tất cả dữ liệu như là một tập hợp duy nhất không có ý nghĩa riêng biệt, chúng có thể có các trường khác nhau cho mỗi bản ghi. Điều này cho phép sự linh hoạt mong muốn và tiến gần hơn với khái niệm hiện đại như lập trình hướng đối tượng. Bởi vì các giá trị tùy chọn không cần được biểu diễn bởi placeholders như trong các RDBs, vì vậy key-value stores thường sử dụng ít bộ nhớ hơn nhiều so với RDBs với cùng một CSDL, điều này dẫn đến việc nâng cao hiệu năng trong những công việc cụ thể.

Như vậy, với sự đơn giản của cách lưu trữ dạng Key – value làm cho các CSDL loại này rất phù hợp với các ứng dụng cần truy xuất nhanh và khả năng mở rộng cao, chẳng hạn như là các quản lý các phiên giao dịch (session) hoặc quản lý các thông tin về giỏ hàng vì trong trường hợp này, việc biết được ID của phiên giao dịch hoặc ID của khách hàng là điều rất cần thiết. Hay việc quản lý thông tin của sản phẩm bao gồm những thông tin cơ bản, các sản phẩm liên quan, đánh giá,. . . sẽ được lưu trữ dưới dạng key là mã sản phẩm chẳng hạn và value là các thông tin còn lại của sản phẩm cần lưu trữ. Điều này, cho phép ta truy xuất được tất cả các thông tin về một sản phẩm chỉ thông qua mã sản phẩm cực kỳ nhanh.

1.6.2 Document

CSDL Document được thiết kế để quản lý và lưu trữ dữ liệu ở dạng document. Những document này được mã hóa về các dạng chuẩn như là XML, JSON (Javascript Option Notation) hay BSON (Binary JSON). Khác với các kiểu lưu trữ dạng Key-value, giá trị của cột trong các CSDL document chứa dữ liệu bán cấu trúc (Semi-Structured Data), đặc biệt là cặp thuộc tính name (key) – value. Một cột có thể chứa hàng trăm các thuộc tính như vậy, số lượng, loại thuộc tính được lưu lại có thể khác nhau giữa các dòng. Một điểm khác nữa so với các kiểu lưu trữ dữ liệu dạng Key-value đơn giản là cả key và value đều có thể tìm kiếm trong CSDL Document.

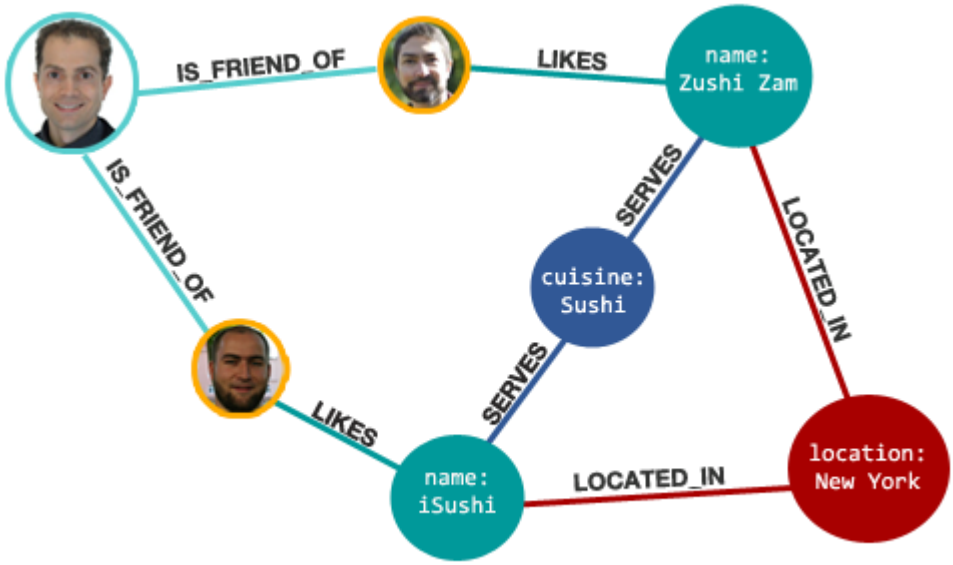
CSDL Document phù hợp cho việc lưu trữ và quản lý tập dữ liệu có kích thước lớn như là tài liệu văn bản, tin nhắn, cũng như biểu diễn một thực thể CSDL như là Product hay Customer (tài liệu khái niệm trong XML). Các CSDL tiêu biểu cho CSDL Document:

CouchDB (JSON), MongoDB (BSON),. . . đều là mà nguồn mở (open source), hướng document (ducument oriented) và có lược đồ tự do (shema free).

1.6.3 Cơ sở dữ liệu đồ thị (Graph Database)

CSDL đồ thị là một cơ sở dữ liệu dạng đồ thị sử dụng các cấu trúc đồ thị với các nút (nodes), các mối quan hệ (relationships), các thuộc tính (properties) để mô tả và lưu trữ dữ liệu.

Chúng ta có thể thực hiện những truy vấn phức tạp hơn như lọc trên các thuộc tính quan hệ, xem xét trọng số của người đó,. . CSDL đồ thị thường được sử dụng để giải quyết các vấn đề về mạng. Trong thực tế, hầu hết các trang web mạng xã hội đều sử dụng một số hình thức của CSDL đồ thị để làm những việc mà chúng ta đã biết như: kết bạn, bạn của bạn, v.v…



Hình 4. Minh họa graph database trong mạng xã hội

Một vấn đề đối với việc mở rộng CSDL đồ thị là rất khó để tìm thấy một đồ thị con độc lập, có nghĩa là rất khó để ta phân tán CSDL đồ thị thành nhiều mảnh. Có rất nhiều nỗ lực nghiên cứu cho việc này nhưng chưa có bất kỳ giải pháp nào đáng tin cậy được đưa ra.

1.6.4 Column family

Column Family được biết đến rộng rãi nhất qua sự triển khai BigTable của Google. Nhìn bề ngoài, chúng khá giống với CSDL quan hệ nhưng thực tế là hoàn toàn khác. Một số sự khác biệt dễ thấy nhất là việc lưu trữ dữ liệu theo dòng đối với các HQT CSDL quan hệ với việc

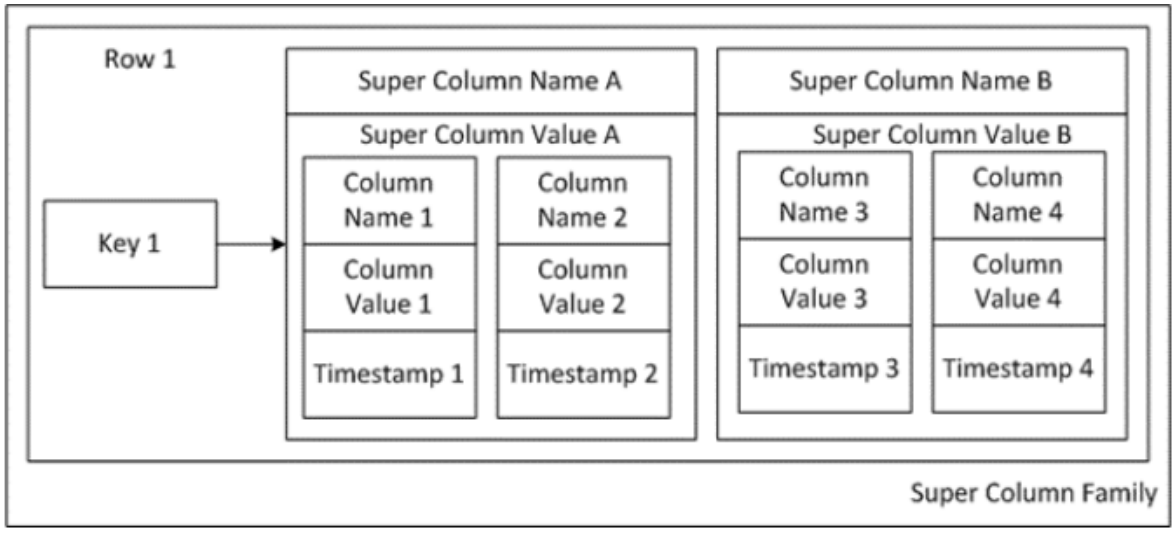
lưu trư dữ liệu theo cột của các HQT CSDL Conlumn Family. Và các tư tưởng của cả hai loại CSDL này cũng hoàn toàn khác nhau. Chúng ta không thể áp dụng cùng một giải pháp mà chúng ta đã sử dụng trong CSDL quan hệ vào CSDL Conlumn Family. Bởi vì, CSDL Column Family là các CSDL phi quan hệ.

Với các CSDL Column Family, chúng ta cần quan tâm đến các khái niệm sau:

Column family (họ cột): Một column family là cách thức dữ liệu được lưu trữ trên đĩa. Tất cả dữ liệu trong một cột sẽ được lưu trên cùng một file. Một column family có thể chứa super column hoặc column.

Super column (siêu cột): Một super column có thể được dùng như một dictionary (kiểu từ điển). Nó là một column có thể chứa những column khác (mà không phải là super column).

Column (cột): Một column là một bộ gồm tên, giá trị và dấu thời gian (thông thường chỉ quan tâm tới key-value).



Hình 5. Cấu trúc dữ liệu trong Column Family

CSDL Column Family được thiết kế để chạy trên một số lượng lớn các máy, và lưu trữ một lượng dữ liệu cực lớn. Chúng ta không thể lưu trữ một lượng lớn dữ liệu trong cơ sở dữ liệu quan hệ vì chắc chắn chúng sẽ nhanh chóng bị sụp đổ hoặc là chết rất nhanh về kích thước của dữ liệu và những truy vấn đó được các CSDL Column Family xử lý một cách dễ dàng. Các CSDL Column Family loại bỏ các khái niệm trừu tượng, những thứ làm cho nó cứng nhắt khi chạy trên một cụm máy.

CSDL được xây dựng ở trong đề tài này sử dụng một phần tư tưởng trong Column Family về mặt phân bố dữ liệu.

Chương 2 THIẾT KẾ VÀ CÀI ĐẶT

2.1 Thiết kế

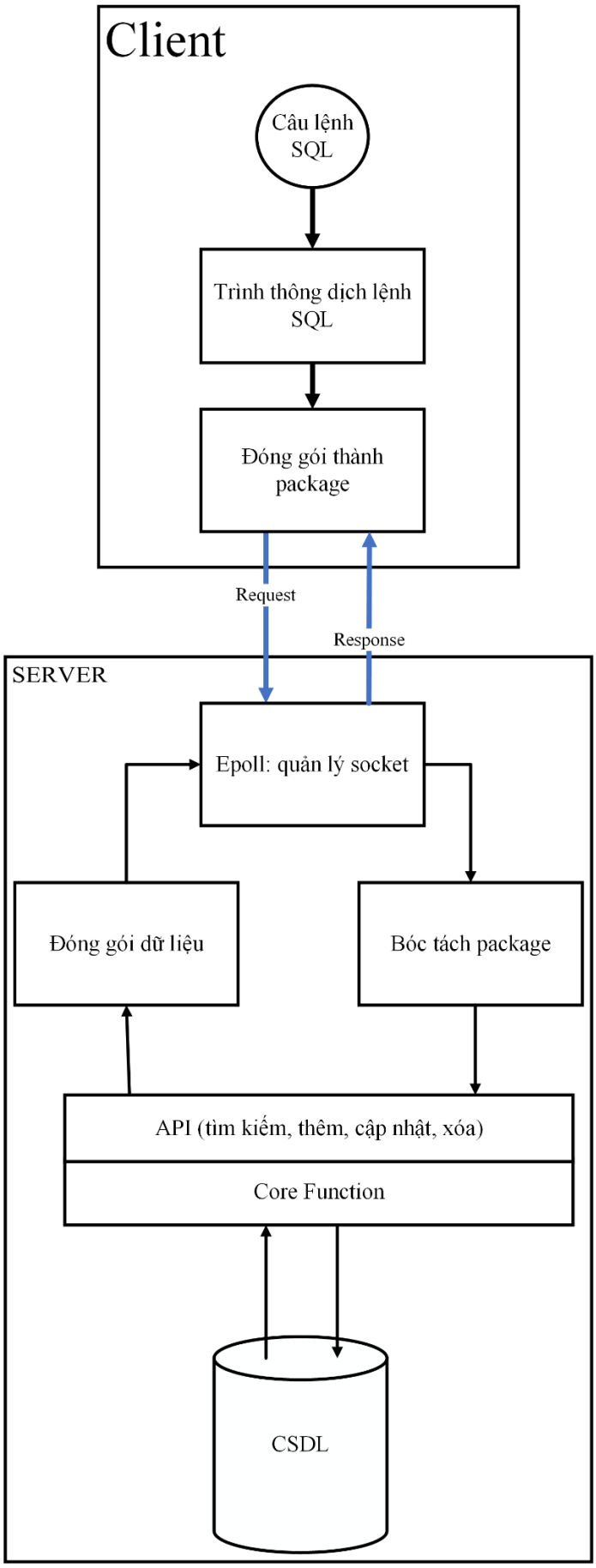
2.1.1 Thiết kế kiến trúc

2.1.1.1 Thiết kế tổng quan

HQT CSDL sẽ bao gồm 2 thành phần chính là: (i) phần cơ sở dữ liệu, (ii) phần quản lý các socket kết nối tới server.

Trong Hình 11 là các thành phần của HQT CSDL bao gồm: (i) cơ sở dữ liệu, (ii) các API để giao tiếp với CSDL, (iii) phần bóc tách bản tin sau khi nhận được từ client, (iv) phần quản lý các socket kết nối tới server sử dụng epoll API.

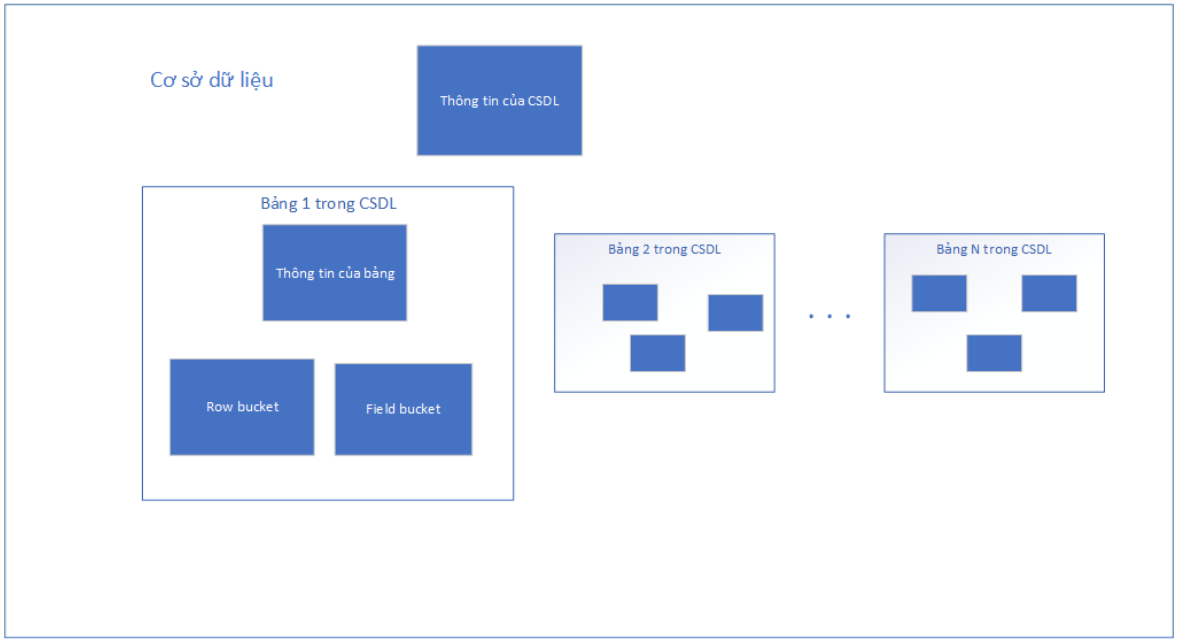
Client sẽ thực hiện các yêu cầu lên server gồm có: (i) tìm kiếm, (ii) thêm, (iii) sửa, (iv) xóa trong CSDL. Khi có resquest được gửi đến server, epoll API sẽ thông báo cho chương trình có I/O từ client tới, chương trình sẽ bóc tách bản tin được gửi và gọi đến API theo đúng yêu cầu của client. Các API sẽ giao tiếp với CSDL và thực hiện các yêu cầu rồi trả về kết quả cho client.


Client
Câu lệnh SQL
Trình thông dịch lệnh SQL
Đóng gói thành package
Request
Response
SERVER
Epoll: quản lý socket
Đóng gói dữ liệu
Bóc tách package
API (tìm kiếm, thêm, cập nhật, xóa)
Core Function
CSDL

Hình 6. Tổng quan trong HQT CSDL

2.1.1.2 Thiết kế trong CSDL

Tương tự như CSDL quan hệ thông thường, CSDL được viết ở trong đề tài bao gồm nhiều bảng (table) và mỗi bảng sẽ bao gồm nhiều trường (field). CSDL bao gồm các thành phần: (i) thông tin của CSDL, (ii) các bảng trong CSDL. Các thành phần được minh họa như hình dưới đây.



Hình 7. Các thành phần tổng quan trong CSDL

Đối với CSDL được thực hiện trong đề tài có các khái niệm sau:

Bản ghi: là một tập hợp các giá trị của các thuộc tính trong một đối tượng cần lưu trữ.

Ví dụ: muốn lưu trữ thông tin sinh viên với thuộc tính như: (i) mã số sinh viên, (ii) tên sinh viên, (iii) lớp. Một sinh viên tên Nguyễn Văn A có mã sinh viên là 1 và học tại lớp CNTT, thì một bản ghi là tập hợp các giá trị {1, Nguyễn Văn A, CNTT}.

Trường: là một thuộc tính của một đối tượng được lưu trữ.

Theo ví dụ trên thì *mã số sinh viên, tên sinh viên, lớp* là các **trường.**

Giá trị của trường: là giá trị một của thuộc tính trong đối tượng cần lưu trữ.

Theo ví dụ trên thì 1 là giá trị của trường *mã số sinh viên*, Nguyễn Văn A là giá trị của trường *tên sinh viên*, CNTT là giá trị của trường *lớp*.

Bảng: là một tập hợp các bản ghi được gộp lại với nhau vì nó lưu trữ dữ liệu về một đối tượng.

Ví dụ, khi ta muốn lưu trữ toàn bộ sinh viên trường đại học Bách Khoa Hà Nội theo 3 trường *mã số sinh viên, tên sinh viên, lớp*. Thì tập các bản ghi lưu trữ toàn bộ sinh viên Bách Khoa theo 3 trường được gọi là một bảng. Bảng này có 3 trường và số lượng bản ghi bằng số lượng sinh viên Bách Khoa.

2.1.1.3 Cách thức lưu trữ trong CSDL

Phần lưu trữ chính trong CSDL là 2 phần field bucket và row bucket.

Đối với field bucket sẽ được tổ chức thành một tập các bảng băm, với mỗi bảng băm sẽ là tập hợp các giá trị của một trường đến từ nhiều bản ghi khác nhau. Mỗi phần từ trong bảng băm sẽ bao gồm giá trị của trường (độ lớn tối đa trong CSDL 28 byte), định danh *bản ghi* của trường, độ lớn của giá trị lưu trữ.

Đối với row bucket là một bảng tham chiếu tới giá trị theo từng trường của từng bảng ghi tới field bucket.

2.1.2 Thiết kế chi tiết

2.1.2.1 Cấu trúc lữu trữ thông tin của CSDL

Hình 8. Cấu trúc lưu trữ thông tin về CSDL

Khi một cơ sở dữ liệu được mở ra hoặc khởi tạo, việc đầu tiên chương trình thực hiện là đọc các thông tin về cơ sở dữ liệu, bao gồm có 4 trường chính:

Tên cơ sở dữ liệu: trường này có độ lớn là 32 byte chứa tên CSDL.

Số lượng bảng trong CSDL: trường này có độ lơn là 4 byte, chỉ ra số bảng mà CSDL đang có.

Vị trí của các bảng trong CSDL: trường này là các vùng nhớ 8 byte liên tiếp nhằm chỉ ra địa chỉ của bảng được lưu trữ trong CSDL. (do giới hạn số lượng bảng tối đa trong một CSDL là 64 nên trường này có độ lớn là 64*8)

Vị trí cuối cùng của CSDL: trường này có độ lớn là 8 byte chỉ ra địa chỉ của điểm cuối tại CSDL.

2.1.2.2 Cấu trúc lưu trữ trong bảng

Sau khi đọc được thông tin về CSDL, chương trình sẽ dựa vào các địa chỉ lấy ra được từ trường *vị trí của bảng trong CSDL* để trỏ tới các bảng trong CSDL. Dưới đây là cấu trúc lưu trữ các bảng trong CSDL.

Hình 9. Cấu trúc lưu trữ trong một bảng

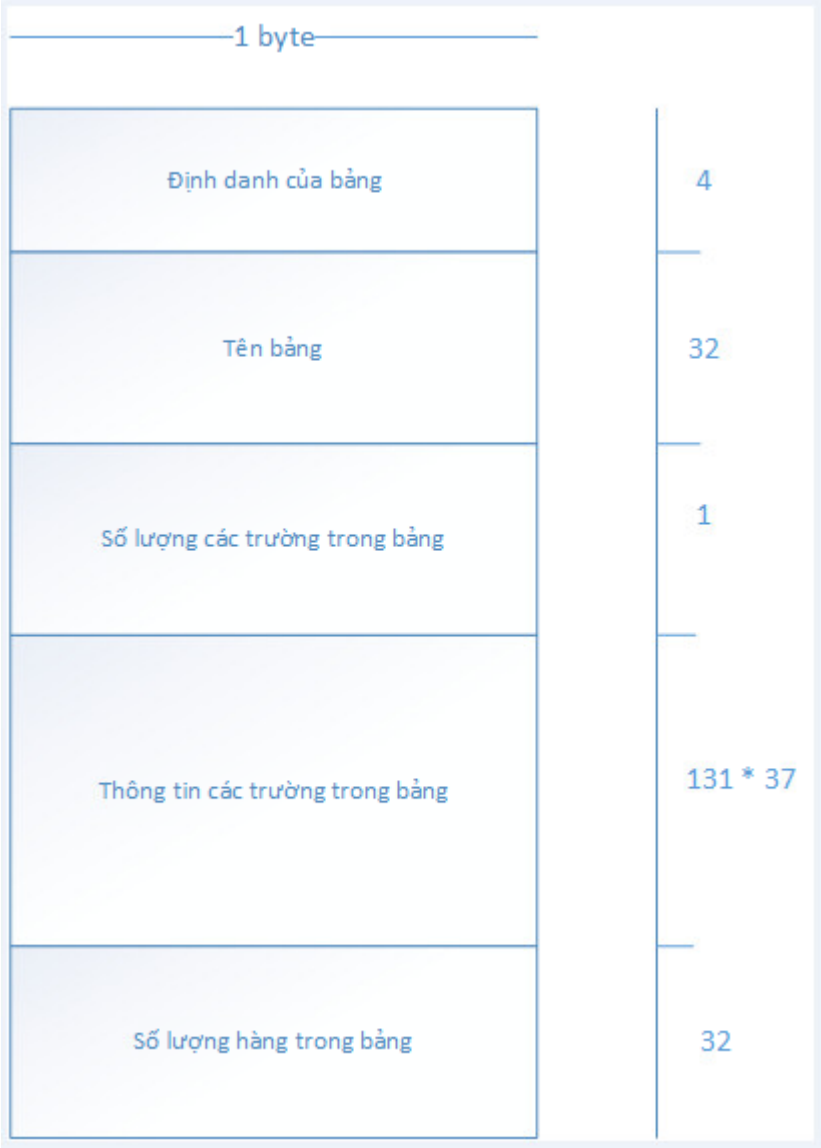
Mỗi vùng nhớ lưu trữ của một bảng sẽ gồm 3 thành phần chính là:

Thông tin về bảng: là tập hợp các thông tin của bảng như: tên, các trường, v.v…

Row bucket: vùng nhớ tương tự như một bảng trong CSDL quan hệ thông thường như thay vì lưu trữ các giá trị thì nó sẽ lưu trữ địa chỉ mà sẽ trỏ tới giá trị của các trường trong bản từng bản ghi của bảng.

Field bucket: vùng nhớ lưu trữ dữ liệu theo các dạng cột, chi tiết được mô trong hình 14.

Dưới đây là cấu trúc lưu trữ các thông tin về bảng:



Hình 10. Cấu trúc lưu trữ thông tin về bảng trong CSDL

Định danh của bảng: đây là một số nguyên duy nhất giúp định danh các bảng trong CSDL.

Tên bảng: tên của bảng trong CSDL.

Số lượng trường trong bảng: là một số nguyên dương độ lớn 1 byte chỉ ra số lượng các trường ở trong bảng.

Thông tin các trường trong bảng: là một chuỗi các vùng nhớ chỉ ra thông tin của trường như: tên trường, vị trí của trường trong Field bucket.

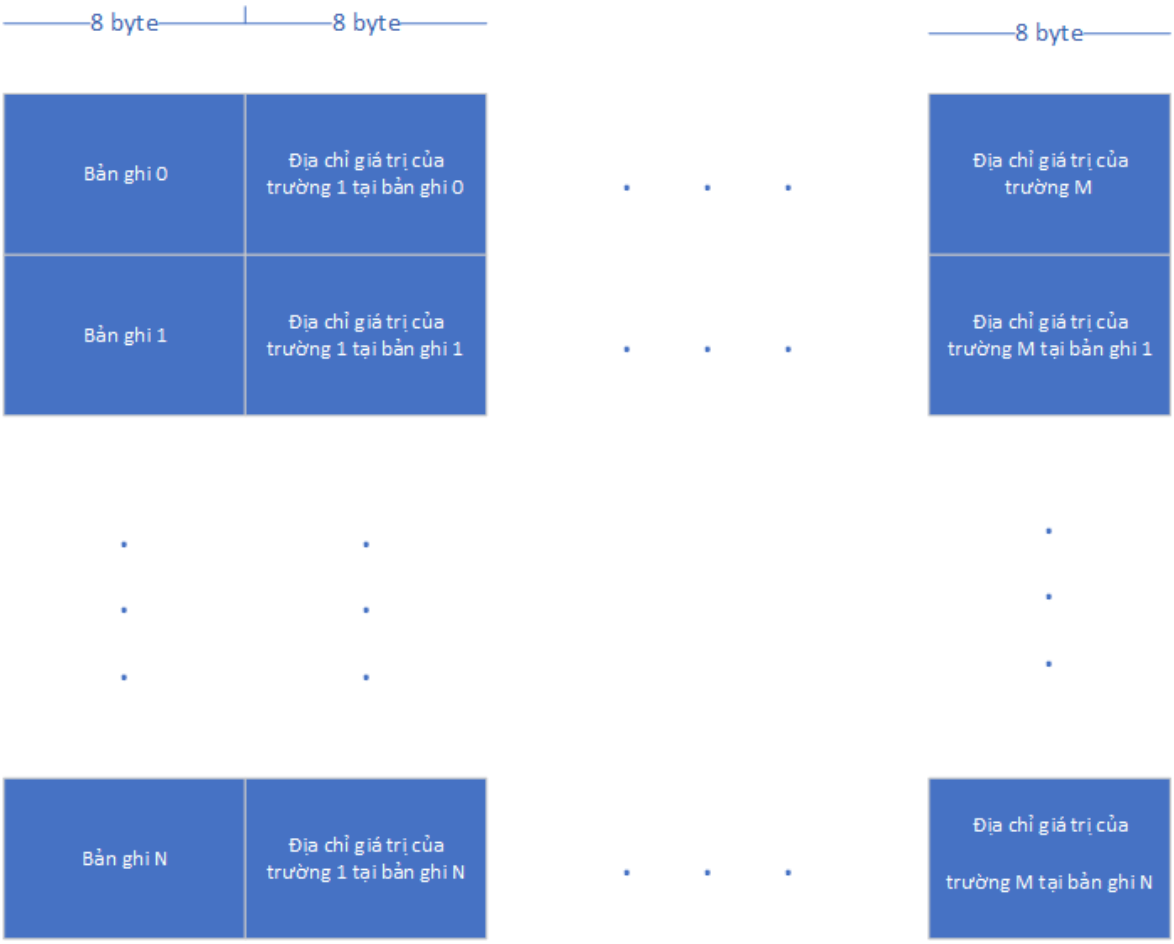
Số lượng hàng trong bảng: chỉ ra hiện tại trong bảng đang có bao nhiêu bản ghi (một hàng tương đương với một bản ghi).

Dưới đây là cấu trúc lưu trữ của field bucket:

Hình 11. Cấu trúc lưu trữ trong field bucket

Một field bucket sẽ chịu trách nhiệm lưu trữ dữ liệu. Nó gồm N bảng băm, trong đó N là số lượng các trường (thuộc tính) có trong bảng. Mỗi bảng băm tương ứng với tập các giá trị trong một trường của các bản ghi trong CSDL.

Dưới đây là cấu trúc lưu trữ của 1 row bucket:



Hình 12. Cấu trúc lưu trữ thông tin các trường theo hàng trong CSDL

Row bucket sẽ không nhận trách nhiệm lưu trữ dữ liệu, mà nó là một vùng nhớ giúp cho chương trình nhận biết vị trí của dữ liệu trong một bản ghi được lưu trữ ở đâu trong field bucket.

Row bucket là cấu trúc lưu trữ tương tự như một bảng trong CSDL quan hệ, với mỗi hàng ở trong hình 13 sẽ tương đương với một bản ghi trong CSDL bình thường. Mỗi hàng sẽ là một tập các địa chỉ trỏ tới các vùng nhớ lưu trữ giá trị trong bản ghi.

2.1.2.3 Hàm băm trong CSDL

Trong CSDL được xây dựng, các giá trị được lưu trữ theo các bảng băm tương ứng với từng trường. Khi muốn thực hiện các hành động: (i) tìm kiếm, (ii) thêm, (iii) sửa, (iv) xóa. Thì cần phải băm các giá trị đó để tìm ra được vùng nhớ lưu trữ của giá trị.

Trong bảng băm của CSDL được xây dựng sử dụng double hashing với hàm băm dựa theo đề xuất của Donald Knuth được trình bày tổng quát tại chương 3.

Cụ thể trong CSDL:

Đối với CSDL, K (key) cũng là V (value) được lưu trữ. Giả sử K có độ lớn n byte. K_i là giá trị tại byte thứ i của K. $table\_size$ là độ lớn của bảng băm.

Hàm băm thứ nhất $h_1(K)$:

Với

$$h_{val} = n \,.\, 16^n + \sum_{i=0}^{n-1} (\, 16^i . K_i)$$

$$h_1(K) = (h_{val}\ mod\ table\_size) + 1$$

Hàm băm thứ hai $h_2(K)$

Với, index là giá trị của lần băm trước nếu tiếp tục thực hiện băm.

$$index = \begin{cases} h_1(K) & nếu\ h_2(K)\ được\ thực\ hiện\ lần\ đầu \\ h_2(K)\ của\ lần\ băm\ trước & \end{cases}$$

$$h_{val2} = 1 + (h_{val}\ mod\ (table\_size - 2)\ )$$

$$h_2(K) = \begin{cases} table\_size + index - h_{val2} \\ index - h_{val2} \end{cases}$$

Giả sử khi có một giá trị A của một trường cần được thêm vào trong CSDL, bảng băm $TABLE$ của trường đó sẽ được thêm vào phần tử A này, với N là số phần tử đã được lưu trữ trong bảng băm, M là độ lớn của bảng băm. Dưới đây là các bước thực hiện:

B1: Gán $index \leftarrow h_1(K)$.

B2: Nếu $TABLE[index]$ là rỗng, đi tới **B6**. Ngoài ra nếu $KEY[index] =$ A, trả về lỗi vì đã tồn tại A trong bảng băm.

B3: Gán $index \leftarrow h_2(K)$.

B4: Nếu $TABLE[i]$ là rỗng, tới **B5**. Ngoài ra nếu $KEY[index] = K$ trả về lỗi vì đã tồn tại A trong bảng băm. Nếu không thì quay trở lại **B4**.

B5: Nếu $N = M$, kết thúc vì tràn bảng băm. Nếu không đánh dấu $TABLE[i]$ đã được sử dụng, thêm A vào trong phần tử $TABLE[i]$ của bảng băm.

Tương tự, khi ta muốn tìm kiếm phần tử có giá trị A ở trong bảng:

B1: Gán $index \leftarrow h_1(K)$.

B2: Nếu $TABLE[index\ ]$ được đánh dấu đang sử dụng, kiểm tra $KEY[index\ ] =$ A, nếu có nhảy tới B6. Nếu $TABLE[index\ ]$ không được sử dụng, trả về kết quả không tìm thấy A trong bảng băm.

B3: Gán $index \leftarrow h_2(K)$.

B4: Kiểm tra $index = h_1(K)$, kết thúc trả về kết quả không tìm thấy value.

B5: Nếu $TABLE[index]$ được đánh dấu đang sử dụng thì kiểm tra $KEY[index\ ] = K$, nếu có thực hiện B6, nếu không quay trở lại B3. Nếu $TABLE[index]$ không được sử dụng, trả về kết quả không tìm thấy value.

B6: Trả về $TABLE[index]$ là giá trị cần tìm.

2.1.3 Xây dựng ứng dụng

2.1.3.1 Thư viện và công cụ sử dụng

Bảng 2. Danh sách thư viện và công cụ sử dụng

Mục đích	**Công cụ**	**Địa chỉ URL**
IDE lập trình	Visual Studio Code	https://code.visualstudio.com/
Thư viện	glibc	https://www.gnu.org/software/libc/
Build chương trình	GNU make	https://www.gnu.org/software/make/
Quản lý source code	Git	https://git-scm.com/
	github	https://github.com/dangviethung096

2.1.3.2 Kết quả đạt được

Từ ý tưởng đặt ra ở chương 2 cùng với lý thuyết, kết hợp với các công nghệ được trình bày ở chương 3 và thiết kế trong chương 4. Em đã xây dựng mô hình HQT CSDL theo dạng cột. Trong đó HQT CSDL sử dụng mô hình client-server thực hiện các thao tác.

Mã nguồn của HQT CSDL được xây dựng là:

Phía server: https://github.com/dangviethung096/DatabaseServer

Phía client: https://github.com/dangviethung096/database_client

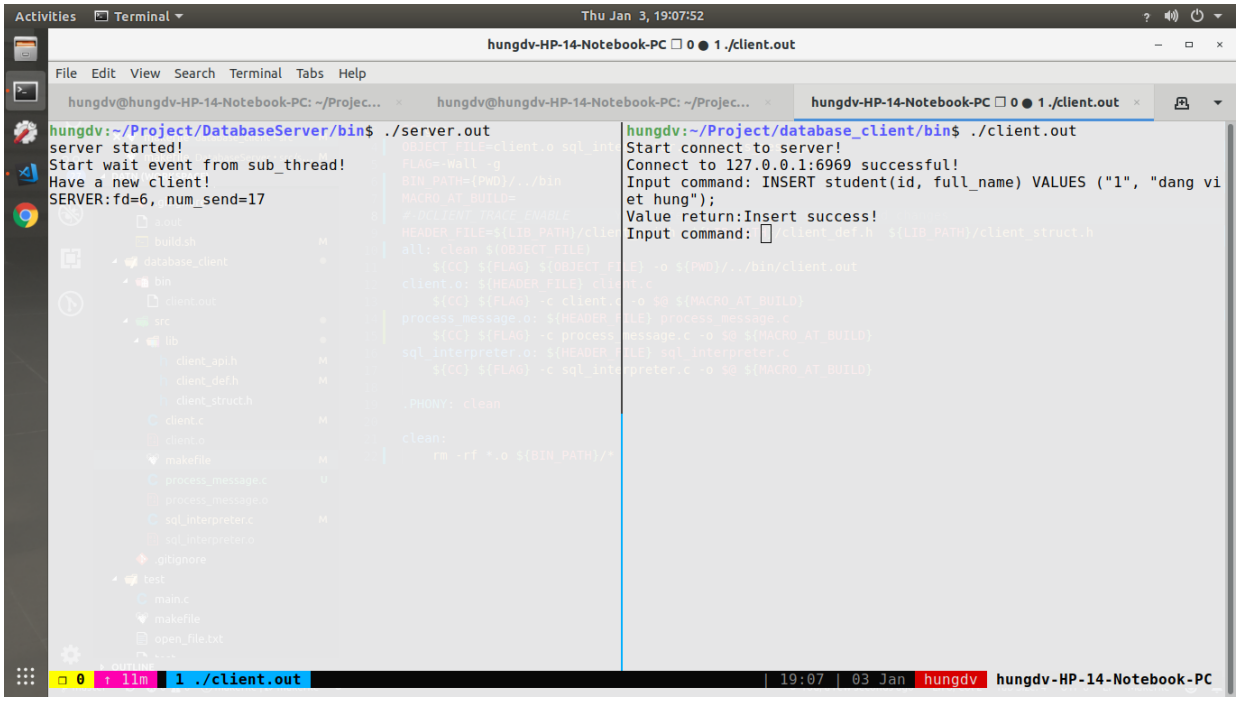
2.1.3.3 Minh hoạ các chức năng chính

Trong HQT CSDL được xây dựng, nó hỗ trợ 4 chức năng cơ bản trong việc lưu trữ đó là: (i) thêm, (ii) tìm kiếm, (iii) sửa , (iv) xóa.

Muốn thực hiện 4 chức năng này, người sử dụng sẽ kết nối tới server thông qua một địa chỉ IP được xác định từ đầu. Sau đó người dùng sẽ nhập các lệnh theo cú pháp SQL để thực hiện 4 chức năng này.

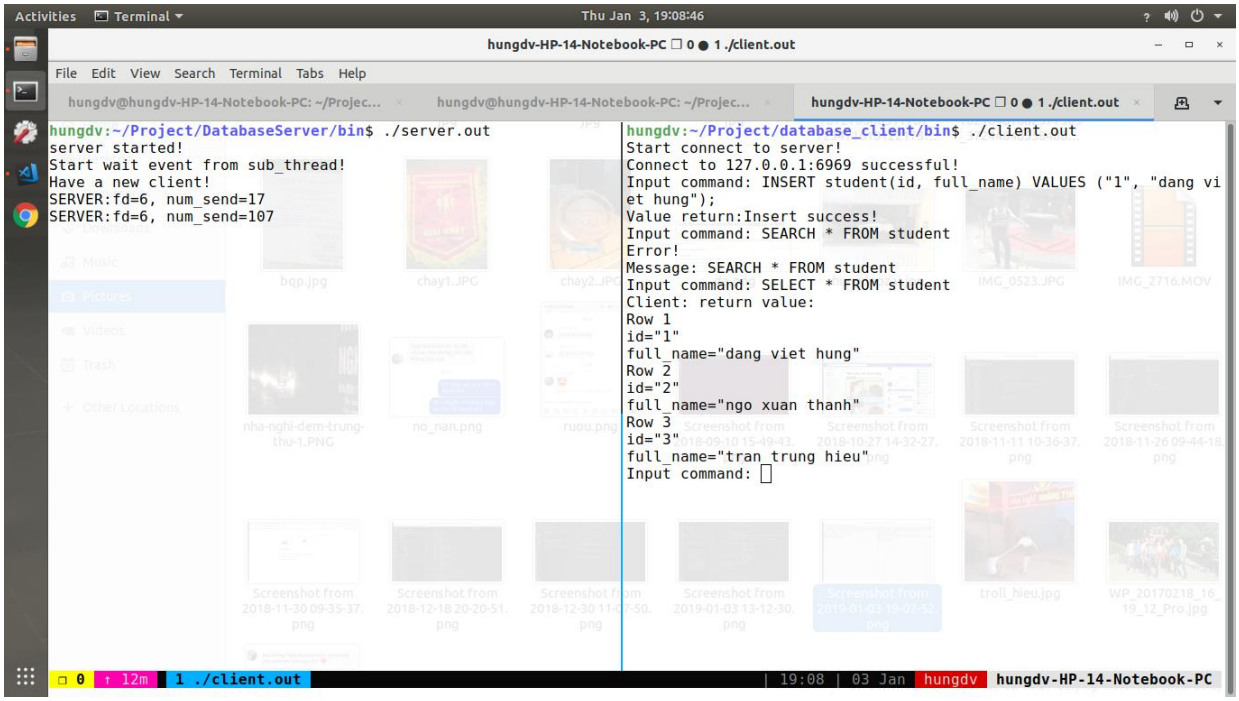
Dưới đây là hình minh họa của 4 chức năng:

Thêm

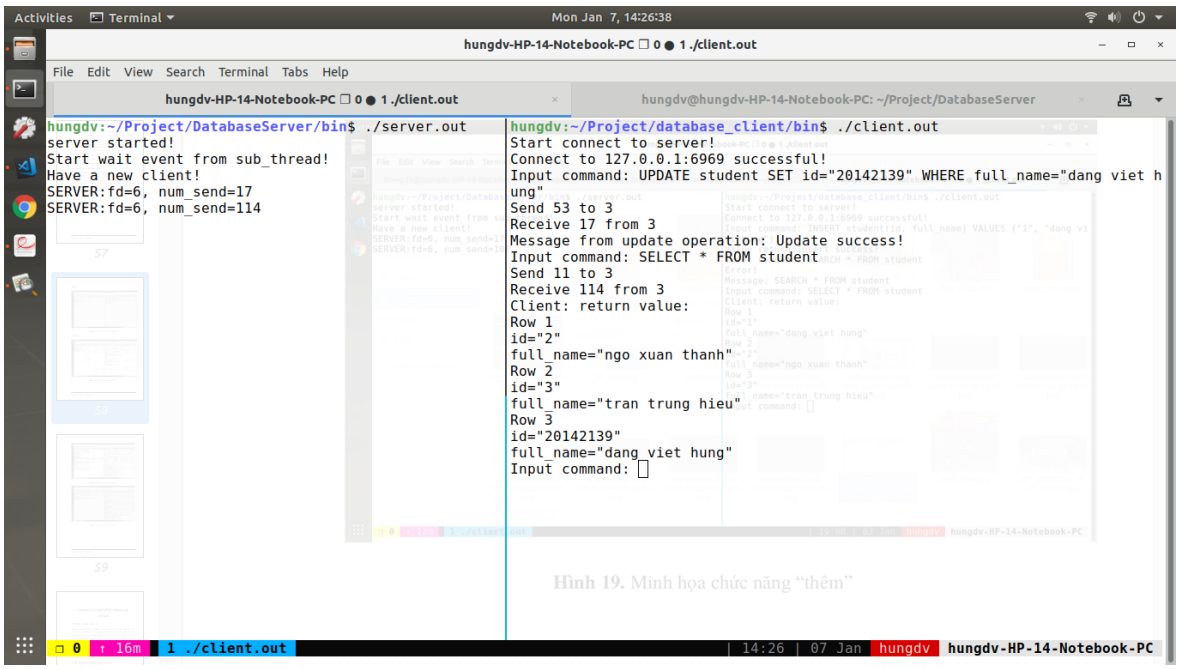


Hình 13. Minh họa chức năng "tìm kiếm"

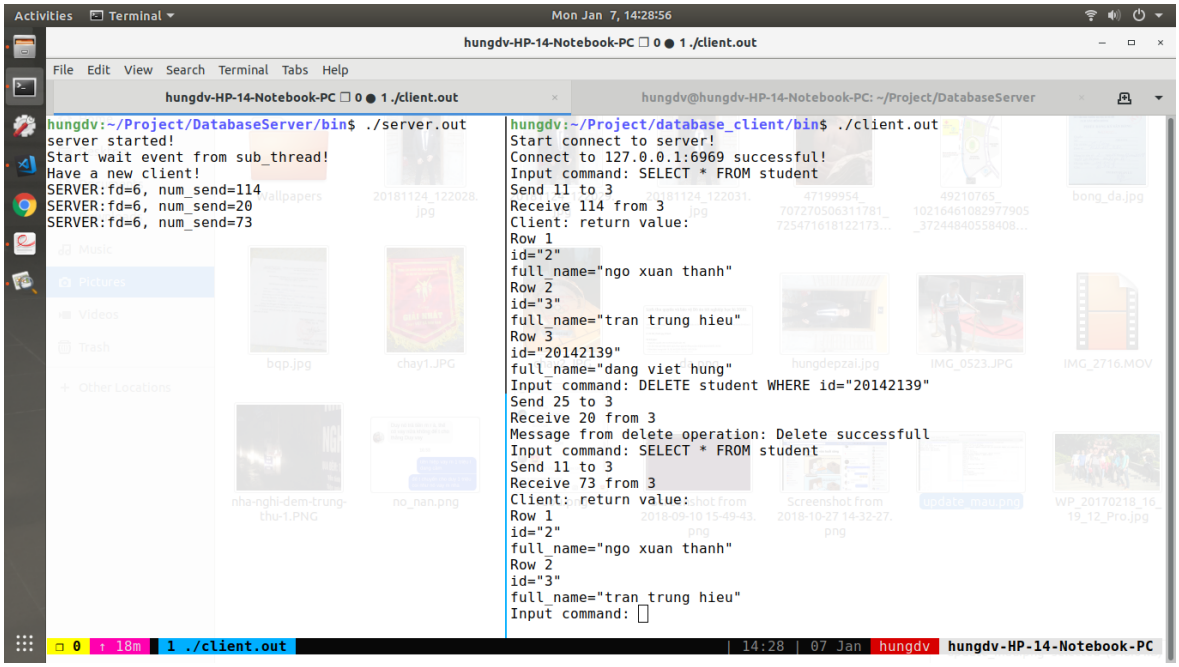
Tìm kiếm



Hình 14. Minh họa chức năng "thêm"



Hình 15. Minh họa chức năng "cập nhật"



Hình 16. Minh họa chức năng "xóa"

2.2 Công nghệ sử dụng

2.2.1 Epoll API trong linux

Để xây dựng mô hình client-server cho HQT CSDL theo dạng cột, phải sử dụng Inter-Process Communication (IPC) là kỹ thuật truyền thông giữa các tiến trình. Mà cụ thể ở đây là socket. Bản chất của socket trong hệ điều hành Linux là các file descriptors (fd), là một tài nguyên thực hiện I/O trong Linux.

Mỗi client kết nối tới server sẽ là một socket tương đương với một fd, vậy khi số lượng lớn client kết nối tới nhiều server thì ta phải quản lý số lượng lớn các fd. Epoll API ở đây được sử dụng để nhằm quản lý các fd.

2.2.1.1 Khái niệm epoll API

Epoll là một API trong được cung cấp bởi nhân Linux, xuất hiện lần đầu trong phiên bản Linux 2.6. Epoll cho phép một tiến trình có thể giám sát nhiều fd để kiểm tra các yêu cầu I/O khả dụng trên các fd. Epoll cung cấp một hiệu năng tốt hơn khi giám sát một số lượng lớn các fd.

Cấu trúc dữ liệu trung tâm của epoll API là một *epoll instance*. *epoll instance* này sẽ thực được tham chiếu bởi một open fd. Nhưng fd này không sử dụng cho việc I/O. Thay vào đó, nó phục vụ cho việc điều khiển cấu trúc dữ liệu của *epoll instance* nằm trong kernel. *epoll instance* có 2 mục đích chính là: (i) lưu trữ một danh sách các fd mà tính tiến trình muốn giám sát khi có các sự kiện I/O gọi là *danh sách giám sát (interest list),* (ii) duy trì một danh sách các fd mà xảy ra sự kiện I/O và đang đợi xử lý được gọi là *danh sách sẵn sàng (ready list).*

2.2.1.2 Các lời gọi hệ thống trong epoll API

Epoll API cung cấp 3 lời gọi hệ thống là: (i) *epoll_create,* (ii) *epoll_ctrl,* (iii) *epoll_wait.*

epoll_create: Trong đó lời gọi hệ thống *epoll_create* tạo ra một *instance epoll* mới với danh sách và trả về một file descriptor tham chiếu tới instance đó.

```
#inlucde <sys/epoll.h>
```

```
int epoll_create(int size)
```

Tham số đầu vào *size* chỉ ra số lượng fd được mong chờ sẽ giám sát trong *epoll instance*. Từ phiên bản Linux 2.6.8, tham số này đã bị bỏ qua. Hàm sẽ trả về fd tham chiếu tới *epoll instance* và khi không còn sử dụng nữa thì sử dụng hàm *close()* để đóng fd này lại.

epoll_ctl: là lời gọi hệ thống dùng cho việc thay đổi *danh sách giám sát* trong *epoll instance*.

```
#include <sys/epoll.h>

int epoll_ctl(int epfd, int op, int fd, struct epoll_event *ev);
```

Tham số *epfd* là giá trị trả về bởi *epoll_create* trỏ tới *epoll_instance* mà có *danh sách giám sát* nó quan tâm.

Tham số *fd* xác định file descriptor trong *danh sách giám sát* sẽ được thay đổi.

Tham số *op* xác định hành động sẽ được thực hiện trong *danh sách giám sát*, tham số này nhận một trong các giá trị sau:

- EPOLL_CTL_ADD: thêm file descriptor *fd* vào trong *danh sách giám sát* của *epfd*. Tập các sự kiện sẽ được giám sát được xác định ở trong bộ đệm trỏ bởi con trỏ *ev*. Nếu thực sự đã tồn tại file descriptor ở trong *danh sách giám sát*. Hàm sẽ trả về lỗi EEXIST.
- EPOLL_CTL_MOD: Thay đổi các sự kiện được giám sát của *fd* trong *danh sách giám sát*. Nếu *fd* mà không nằm trong *danh sách giám sát* thì hàm thất bại sẽ trả về lỗi ENOENT.
- EPOLL_CTL_DEL: Xóa *fd* khỏi danh sách giám sát ở trong *epfd*. Tham số *ev* sẽ được bỏ qua đối với hành động này. Nếu *fd* không nằm trong *danh sách giám sát* thì hàm sẽ thất bại và trả về lỗi ENOENT. Việc đóng file descriptor sẽ tự động xóa file đó khỏi *danh sách giám sát* của tất cả các *epoll_instance* mà nó là thành viên.

Tham số *ev* là một con trỏ mà trỏ tới cấu trúc *epoll_event*, được định nghĩa như sau:

```
struct epoll_event {
    uint32_t   events;          /* các sự kiện của epoll */
    epoll_data_t data;          /* dữ liệu người dùng */
```

```
}
```

Trường *data* trong cấu trúc *epoll_event*, được định nghĩa như sau:

```
typedef union epoll_data {
     void       *ptr; /* Con trỏ mà tới dữ liệu của người dùng */
     int        fd;   /* File descriptor */
     uint32_t   u32;  /* 32 bit integer */
     uint64_t   u64;  /* 64 bit integer */
} epoll_data_t;
```

Tham số *ev* xác định

Trường con *events* là một bit mask xác định tập các sự kiện mà chúng ta quan tâm trong giám sát *fd*. Các cơ thông dụng được sử dụng là: (i) EPOLLIN quan tâm tới các sự kiện INPUT, (ii) EPOLLOUT quan tâm tới các sự kiện OUTPUT, (iii) EPOLLERR quan tâm tới các lỗi xảy ra, (iv) EPOLLET sử dụng thông báo sự kiện edge-triggered.

Trường con *data* là một union, dùng để xác định thông tin được truyền trở lại tới hàm được gọi nếu *fd* nằm trong *danh sách sẵn sàng*.

epoll_wait: lời gọi hệ thống này sẽ trả về thông tin về các file descriptor nằm trong *danh sách sẵn sàng* từ *epoll instance*. *epoll_wait* có thể trả về nhiều file descriptor sẵn sàng.

```
#include <sys/epoll.h>
int epoll_wait(int epfd, struct epoll_event *evlist, int maxevents,
               int timeout);
```

Tham số *epfd* tham chiếu tới *epoll_instance* mà chúng ta quan tâm.

Thông tin về các fd trong *danh sách sẵn sàng* được trả về trong mảng cấu trúc *epoll_event* trỏ bởi *evlist*. Mảng *evlist* sẽ được cấp phát bởi người gọi, và số lượng các thành phần sẽ được chỉ ra trong tham số *maxevents.*

Trường con *events* sẽ trả về một mask các sự kiện mà xảy ra trong file descriptor đó. Trường con *data* trả về bất cứ giá trị nào được xác định trong *ev.data* khi nó được thêm vào.

Tham số *timeout* xác định thời gian đợi cho đến khi có phần tử trong *danh sách sẵn sàng*.

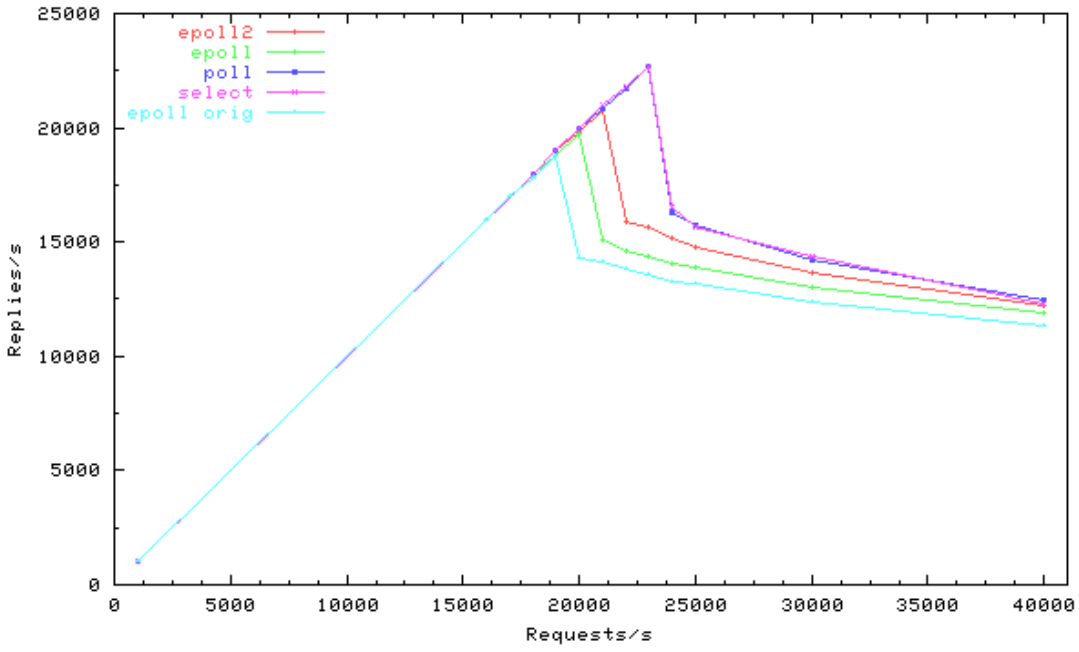
- Nếu *timeout* bằng -1, hàm sẽ đợi cho đến khi có phần tử nằm trong *danh sách sẵn sàng*.
- Nếu *timeout* bằng 0, thực hiện kiểm tra *danh sách sẵn sàng* và trả về ngay lập tức kể cả khi không có phần tử nào trong danh sách.
- Nếu *timeout* lớn hơn 0, sẽ thực hiện đợi trong *timeout* micro giây hoặc cho đến khi có phần tử trong *danh sách sẵn sàng*.

Nếu thành công *epoll_wait* sẽ trả về số lượng phần tử đã được đặt trong mảng *evlist* hoặc 0 nếu không có file descriptor trong *danh sách sẵn sàng*. Nếu lỗi, *epoll_wait()* trả về -1.

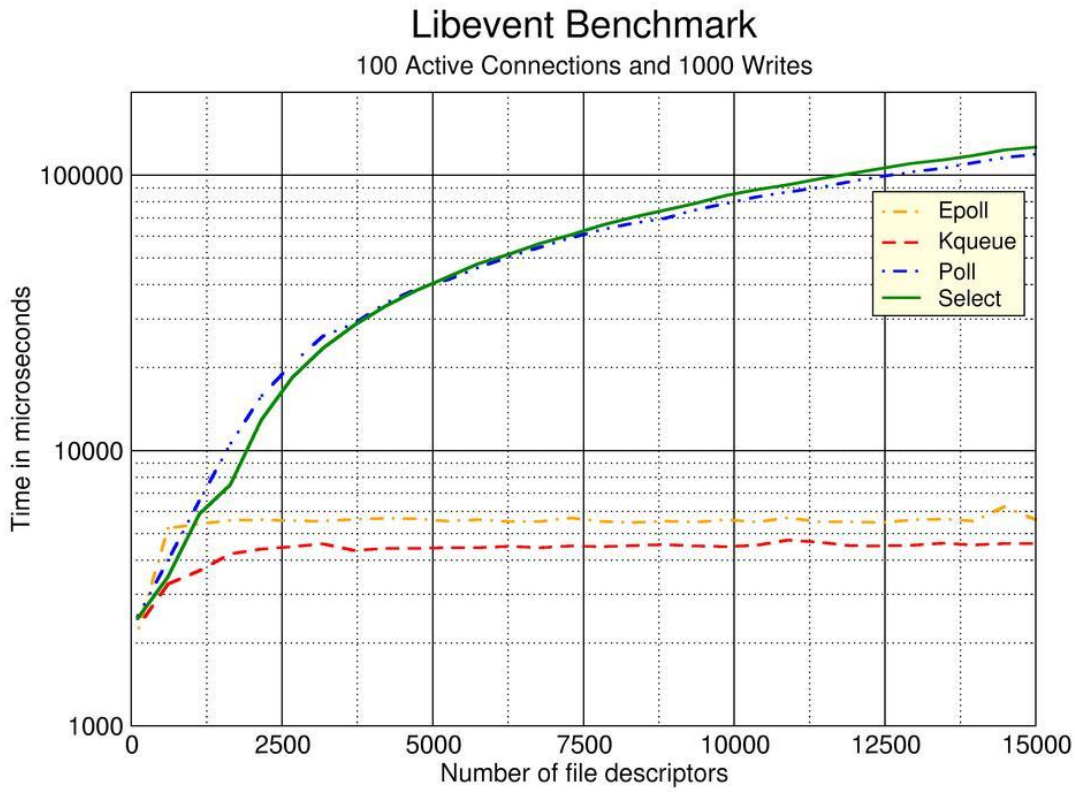
2.2.1.3 Ưu điểm và nhược điểm của epoll

Ưu điểm:

- Đạt được hiệu năng cao khi giám sát một số lượng lớn các fd
- Xác định rõ ràng file descriptor mà chúng ta muốn giám sát
- Có thể lựa chọn giữa thông báo edge-triggred và thông báo level-triggered



Hình 17. So sánh hiệu năng giữa epoll với select, poll



Hình 18. So sánh hiệu năng epol với poll, select và kqueue trong thư viện libevent

Nhược điểm:

- Nó là một API của Linux

2.2.2 Bộ thư viện glibc

Đối với CSDL được xây dựng ở trong đề tài, dữ liệu sẽ được lưu trữ ở trong file với cấu trúc được thiết kế sẽ được chỉ ra trong Chương 4. Để thực hiện các công việc vào ra trong trong file, chương trình sẽ tập trung sử dụng các hàm tron g thư viện glibc của Linux. Cụ thể đó là các hàm *open, close, read, write, lseek.*

2.2.2.1 open

Hàm open có nguyên mẫu:

```
#include <sys/types.h>
```

```
#include <sys/stat.h>

#include <fcntl.h>

int open(const char *path_name, int flags, mode_t mode);
```

open là một lời gọi hệ thống dùng để mở file xác định trong *path_name*. Nếu file không tồn tại, nó có thể tạo ra một file mới nếu cờ O_CREAT được xác định trong tham số *flags*. Giá trị trả về của hàm là một fd nếu thành công, và fd này sẽ dùng để thực hiện I/O với file.

Tham số *flags* phải bao gồm một trong các cờ sau: (i) O_RDONLY chỉ đọc file, (ii) O_WRONLY chỉ ghi file, (iii) O_RDWR thực hiện cả đọc và ghi file.

Tham số *mode* xác định các quyền của file được áp dụng khi file mới được tạo. Tham số này sẽ được dùng khi tham số *flags* có cờ O_CREAT hoặc O_TMPFILE. Nếu *flags* không có 1 trong 2 cờ này thì tham số *mode* sẽ bị bỏ qua. Tham số *mode* có các cờ sau: (i) S_IRWXU người sở hữu có quyền đọc, ghi và thực thi, (ii) S_IRUSR người sở hữu có quyền đọc, (iii) S_ IWUSR người sở hữu có quyền ghi, (iv) S_IXUSR người sở hữu có quyền thực thi, (v) S_IRWXG nhóm của người sở hữu có quyền đọc, ghi và thực thi, (vi) S_IWGRP nhóm của người sở hữu có quyền ghi, (vii) S_IXGRP nhóm của người sở hữu có quyền thực thi, (viii) S_IRWXO tất cả người dùng có quyền đọc, ghi và thực thi, (ix) S_IROTH tất cả người dùng đều có quyền đọc, (x) S_IWOTH tất cả người dùng đều có quyền ghi, (xi) S_IXOTH tất cả người dùng đều có quyền thực thi.

2.2.2.2 close

Hàm close có nguyên mẫu:

```
#inlcude <unistd.h>

int close(int fd);
```

Hàm close sẽ thực hiện đóng file descriptor lại và nếu *fd* là tham chiếu cuối cùng tới file descriptor thì các tài nguyên cấp phát cho file descriptor sẽ được giải phóng.

2.2.2.3 read

Hàm read có nguyên mẫu:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Hàm read sẽ thực hiện đọc *count* byte từ *fd* vào trong buffer mà bắt đầu từ *buf*.

2.2.2.4 write

Hàm write có nguyên mẫu:

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

Hàm write thực hiện viết *count* byte từ buffer được trỏ bởi *buf* tới file được tham chiếu bởi file descriptor *fd.*

2.2.2.5 lseek

Hàm lseek có nguyên mẫu

```
#include <sys/types.h>

#include <unistd.h>

Off_t lseek(int fd, off_t off_set, int whence);
```

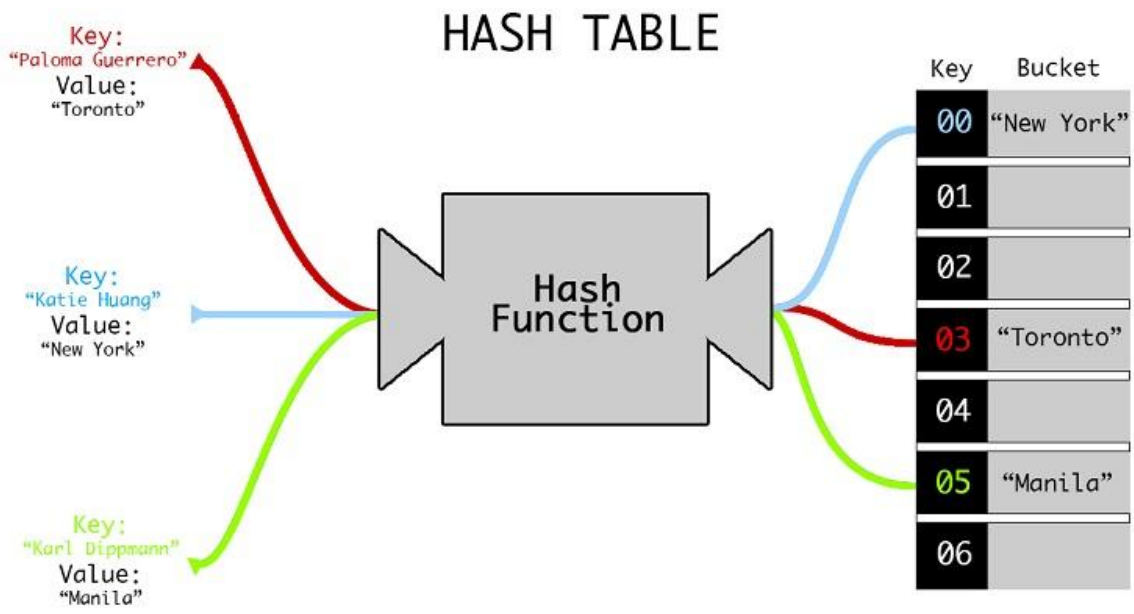
Hàm lseek thực hiện việc định lại vị trí của file được tham chiếu bởi *fd*. Vị trí mới sẽ được xác định cách *offset* byte từ *whence*. *whence* có thể nhận các giá trị: (i) SEEK_SET là vị trí đầu của file, (ii) SEEK_CUR là vị trí hiện tại của file, (iii) SEEK_END vị trí cuối cùng của file.

2.2.3 Bảng băm

Trong CSDL ở trong đề tài, bảng băm được sử dụng để lưu trữ giá trị trong CSDL, chi tiết phần lưu trữ cho trong bảng băm sẽ được đề cập tại chương 4.

2.2.3.1 Khái niệm bảng băm

Bảng băm là một CTDL thường được sử dụng như một từ điển: mỗi phần tử trong bảng băm là một cặp (khóa, giá trị). Nếu so sánh với mảng, khóa sau khi trải qua hàm băm được xem như chỉ số của mảng, còn giá trị giống như giá trị mà ta lưu tại chỉ số tương ứng. Bảng băm không như các loại từ điển thông thường - ta có thể tìm được giá trị thông qua khóa của nó.

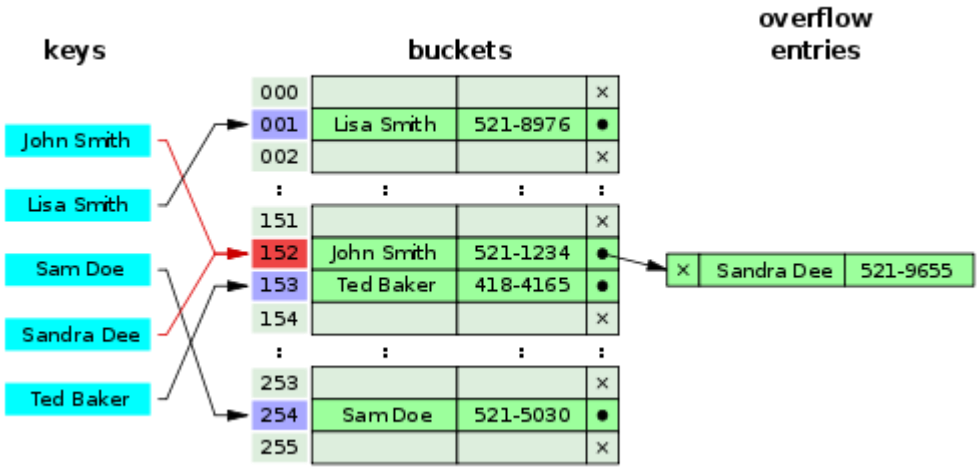


Hình 19. Bảng băm

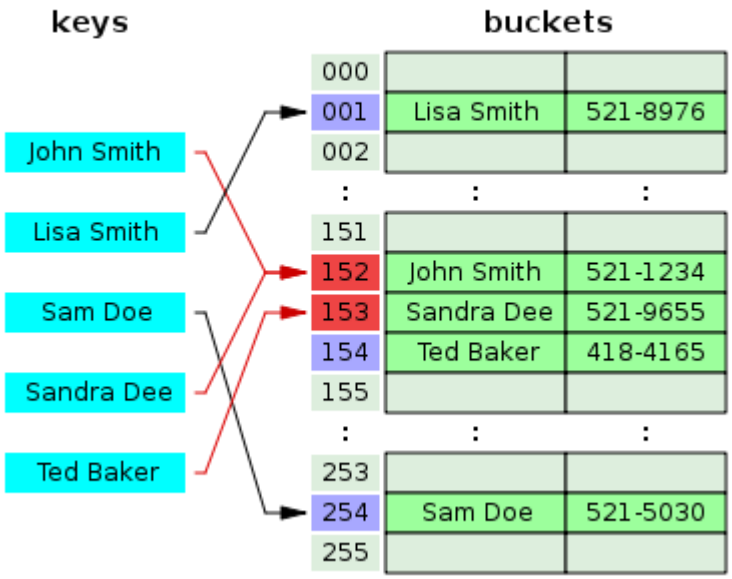
2.2.3.2 Double hashing

Việc sử dụng bảng băm gặp phải một vấn đề được gọi là *đụng độ trong bảng băm.*

Đụng độ trong bảng băm xảy ra khi với hai giá trị khóa khác nhau, sau khi trải qua hàm băm đều chỉ tới vào cùng một vị trí trong mảng. Để giải quyết vấn đề đụng độ này, thông thường có 2 cách được sử dụng là open address và sử dụng danh sách liên kết. Hình dưới là 2 ví dụ về 2 cách giải quyết đụng độ.



Hình 20. Bảng băm với danh sách liên kết



Hình 21. Bảng băm với open address

Double hashing là một cách giải quyết đụng độ trong bảng băm sử dụng open address. Nó sẽ sử dụng 2 hàm băm khác nhau, với mỗi lần băm khác nhau thì giá trị cho ra sẽ khác nhau, gọi h_1, h_2 là 2 hàm băm tại lần băm thứ i với giá trị là k trong bảng băm T, ta có công thức tổng quát của double hashing là:

$$h(i,k) = \left(h_1(k) + i \,.\, h_2(k) \right) mod \, |T|$$

2.2.3.3 Bảng băm với hàm băm đề xuất bởi Donald Knuth

Trong CSDL hàm băm được sử dụng là hàm băm được đề xuất bởi Donald Knuth được viết trong quyển sách *The Art of Computer Programming volume 3*. Hàm băm trong CSDL được thực hiện các như sau:

Bảng băm $TABLE$ với phần tử thứ i của bảng được kí hiệu $TABLE[i]$. Trong bảng lưu $KEY[i]$ là khóa của phần tử thứ i. $VALUE[i]$ là giá trị của phần tử thứ i.

Cho hàm băm $h_1(K)$ là băm ra giá trị trong khoảng từ 0 đến $M-1$ (chỉ số 0 được CSDL dùng để lưu trữ các thông tin khác của bảng băm và bảng băm sẽ được đẩy lên khoảng từ 0 đến M) và hàm băm $h_2(K)$ cũng sẽ băm ra giá trị trong khoảng từ 1 đến $M-1$ và giá trị băm được và M phải là số nguyên tố cùng nhau, vì vậy nếu M là số nguyên tố thì các giá trị trong khoảng từ 1 đến $M-1$ đều là số nguyên tố cùng nhau với M. N là số lượng các giá trị đã được chèn vào trong bảng.

Dưới đây là các bước thực hiện việc thêm một giá trị vào bảng băm, ở đây hàm băm thứ nhất chỉ sử dụng một lần, các lần băm sau hàm băm thứ hai sẽ được sử dụng để băm:

B1: Gán $i \leftarrow h_1(K)$.

B2: Nếu $TABLE[i]$ là rỗng, đi tới **B6**. Ngoài ra nếu $KEY[i] = K$, kết thúc giải thuật vì K đã có ở trong bảng.

B3: Gán $c \leftarrow h_2(K)$.

B4: Gán $i \leftarrow i - c$, nếu $i < 0$ thì gán $i \leftarrow i + M$

B5: Nếu $TABLE[i]$ là rỗng, tới **B6**. Ngoài ra nếu $KEY[i] = K$, kết thúc giải thuật vì K đã có ở trong bảng. Nếu không thì quay trở lại **B4**.

B6: Nếu $N = M - 1$, kết thúc vì tràn bảng băm. Nếu không đánh dấu $TABLE[i]$ đã được sử dụng, gán $VALUE[i] \leftarrow V$ đặt $KEY[i] \leftarrow K$.

Hàm $h_2(K)$ được đề nghị là $h_2(K) = 1 + (K \ mod \ (M-2))$.

Tương tự với các bước thực hiện tìm kiếm value trong bảng băm.

B1: Gán $i \leftarrow h_1(K)$.

B2: Nếu $TABLE[i]$ được đánh dấu đang sử dụng, kiểm tra $KEY[i] = K$, nếu có nhảy tới B7. Nếu $TABLE[i]$ không được sử dụng, trả về kết quả không tìm thấy value.

B3: Gán $c \leftarrow h_2(K)$.

B4: Gán $i \leftarrow i - c$, nếu $i < 0$ thì gán $i \leftarrow i + M$.

B5: Kiểm tra $i = h_1(K)$, kết thúc trả về kết quả không tìm thấy value.

B6: Nếu $TABLE[i]$ được đánh dấu đang sử dụng thì kiểm tra $KEY[i] = K$, nếu có thực hiện B7, nếu không quay trở lại B4. Nếu $TABLE[i]$ không được sử dụng, trả về kết quả không tìm thấy value.

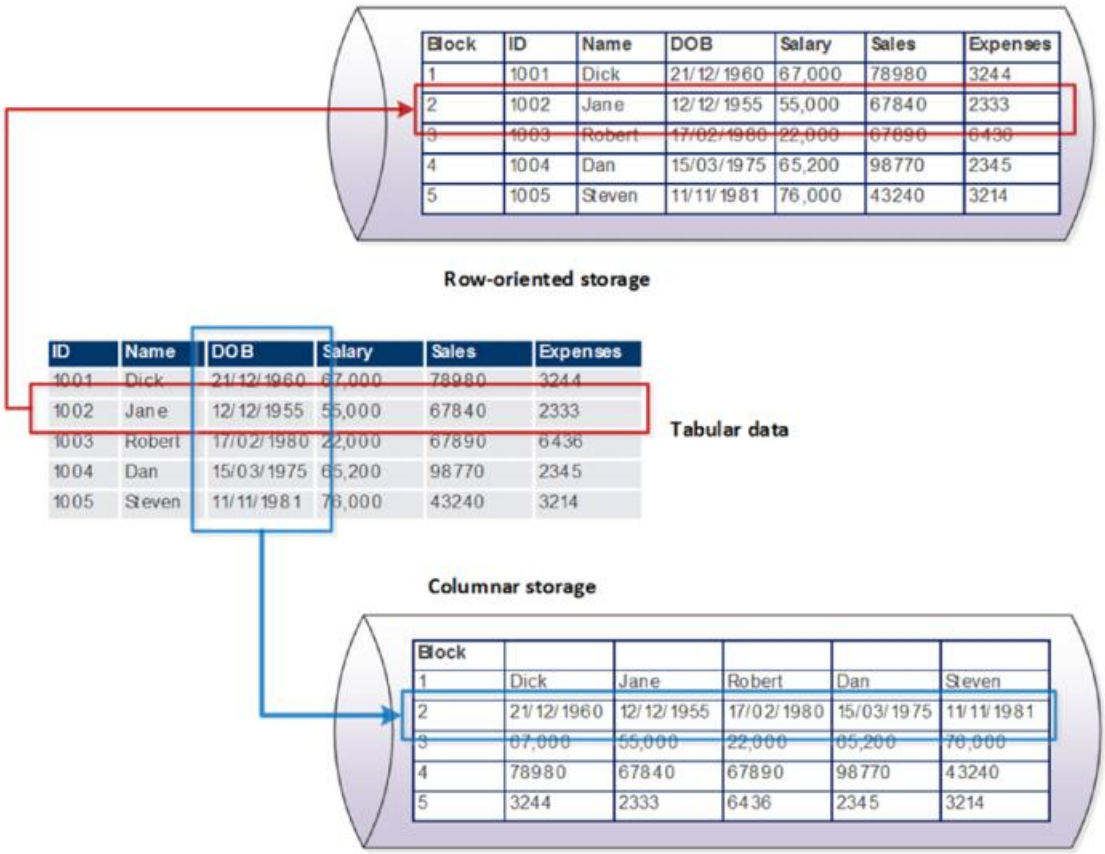
B7: Trả về $VALUE[i]$ là giá trị cần tìm.

Chi tiết về việc áp dụng bảng băm vào trong CSDL sẽ được trình bày trong chương 4.

Chương 3 ĐÁNH GIÁ

3.1 HQT CSDL dạng hàng và dạng cột

Trong kiến trúc theo dạng cột, các truy vấn thực hiện tìm kiếm tập các giá trị mà chỉ thuộc một cột sẽ được tối ưu hóa, bởi vì tập các giá trị của một cột sẽ được lưu trữ nằm trên cùng các block của ổ đĩa. Hình dưới đây minh họa cho điểm thuận lợi này của HQT CSDL theo dạng cột so với HQT CSDL theo dạng hàng, khi truy vấn cần tính tổng lương phải trả cho các nhân viện thì đối với việc lưu trữ theo dạng hàng sẽ phải truy cập vào 5 block, thay vào đó với dạng cột sẽ chỉ cần truy cập vào 1 block duy nhất [1].



Hình 22. Các lưu trữ của HQT CSDL dạng cột và dạng hàng

Trong thực tế, HQT CSDL dạng cột rất thích hợp cho các công việc OLAP (On-line analytical processing), các truy vấn mà yêu cầu phải được thực hiện trên gần như toàn bộ bản ghi có trong bảng nhưng chỉ tập trung tại một cột nhất định [4].

Và ngược lại, HQT CSDL theo dạng hàng thích hợp cho các công việc OLTP (On-line transactional processing), các truy vấn mà yêu cầu lấy hầu hết hoặc tất cả các trường trong một bản ghi. Đây cũng là nhược điểm của HQT CSDL theo dạng cột.

Và do sử dụng bảng băm trong việc lưu trữ giá trị trong cột nên đối với HQT CSDL được xây dựng sẽ tối ưu với các lệnh thêm và tìm kiếm. Lệnh xóa và cập nhật sẽ có tốc độ chậm rõ rệt do việc phải thực hiện băm lại toàn bộ bảng khi xóa hoặc cập nhật

3.2 Đánh giá kết quả đối với HQT CSDL được xây dựng

Dưới đây là một số kết quả thu được trong chạy thử nghiệm HQT CSDL theo dạng cột, toàn bộ việc chạy thử nghiệm được thực hiện ở trên máy tính với cấu hình: CPU Intel Core i5-5200U 2.20GHz x 4, RAM 4GB.

3.2.1 Thêm

Sinh ngẫu nhiên lần lượt số lượng bản ghi tương ứng trong bảng, với mỗi bản ghi gồm có 10 trường (tương đương với 10 cột).

Số lượng bản ghi	**1000**	**2000**	**3000**
Thời gian (s)	1.05	3.63	7.93

Bảng 3. Kết quả thực hiện lệnh thêm

Lý do: bởi vì các thức lưu trữ trong các cột sử dụng bảng băm với cách xử lý đụng độ là open address, vì vậy nên khi số lượng phần tử càng tiến tới giới hạn của bảng băm thì thời gian cần thiết để thêm phần tử vào trong bảng càng lâu do phải thực hiện băm nhiêu lần.

3.2.2 Tìm kiếm

Toàn bộ dữ liệu trong cơ sở dữ liệu thực hiện trong thao tác tìm kiếm được sinh ra với số lượng là 1000 bản ghi, mỗi bản ghi gồm có 10 cột, mỗi giá trị trong một cột có độ lớn tối đa là 26 byte, tối thiểu là 16 byte.

Thực hiện tìm kiếm đối với toàn bộ các bản ghi và lấy ra tất cả các trường của bản ghi:

Số lượng request	**100**	**500**	**1000**
Thời gian (s)	2.70	12.91	25.87

Bảng 4. Kết quả thực hiện lệnh tìm kiếm toàn bộ bản ghi trong bảng và lấy ra tất cả các trường

Thực hiện tìm kiếm đối với toàn bộ bản ghi nhưng chỉ lấy ra một trường:

Số lượng request	**100**	**500**	**1000**
Thời gian (s)	0.36	1.76	3.67

Bảng 5. Kết quả thực hiện lệnh tìm kiếm toàn bộ bản ghi trong bảng và chỉ lấy ra 1 trường

Thực hiện tìm kiếm đối với toàn bộ bản ghi nhưng chỉ lấy ra 3 trường trong bản ghi:

Số lượng request	**100**	**500**	**1000**
Thời gian (s)	0.79	3.95	8.30

Bảng 6. Kết quả thực hiện lệnh tìm kiếm toàn bộ bản ghi trong bảng và lấy ra 3 trường

Từ 3 bảng trên ta có thể thấy rõ lợi điểm của HQT CSDL được tổ chức theo dạng cột, tốc độ tìm kiếm đối với các request chỉ thực hiện lấy ra một số trường trong bản ghi nhanh hơn rõ rệt so với việc lấy ra toàn bộ các trường.

Chương 4 KẾT LUẬN

4.1 Kết luận

Qua việc thực hiện thiết kế và triển khai HQT CSDL theo dạng cột, em thấy mình học hỏi được nhiều kiến thức trong HQT CSDL. Nắm bắt được các vấn đề, ưu điểm, nhược điểm của các HQT CSDL với các mô hình khác nhau.

Tuy nhiên trong HQT CSDL trong đề tài còn rất nhiều thiếu sót về chức năng và chưa đạt được các tính chất cần thiết của một HQT CSDL điển hình. Khi mà nó mới chỉ hỗ trợ các hàm cơ bản trong lưu trữ là tìm kiếm, thêm, sửa, xóa (CRUD – Create, Read, Update, Delete).

4.2 Hướng phát triển

Hướng phát triển:

Hoàn thiện HQT CSDL: đối với HQT CSDL trong đề tài, cần hoàn thiện các tính năng cần thiết như

Nâng cao bảo mật: vì sử dụng theo mô hình client-server nên CSDL có thể bị tấn công, đánh cắp dữ liệu bởi các máy tính khác, vì vậy để đảm bảo vấn đề bảo mật, cần thay thế các sử dụng các socket thông thường như hiện tại bằng các socket tuân theo chuẩn bảo mật như: Secure Sockets Layer (SSL), Transport Layer Security (TLS).

Phân tán: CSDL trong đề tài sử dụng mô hình client - server, từ đó có thể xây dựng lên thành HQT CSDL phân tán với nhiều lợi ích như năng cao hiệu năng.

Tài liệu tham khảo

[1] Guy Harrison, Next Generation Databases, Apress, 2015.

[2] Michael Kerrisk, The Linux Programming Interface: A Linux and UNIX System Programming Handbook, No Starch Press, 2010.

[3] Donald E. Knuth, The Art of Computer Programming: Volume 3: Sorting and Searching (2nd Edition), Addison-Wesley Professional, 1998

[4] Florian Funke, Alfons Kemper, Thomas Neumann

Compacting Transactional Data in Hybrid OLTP&OLAP Databases

[5] https://en.wikipedia.org/wiki/Column-oriented_DBMS