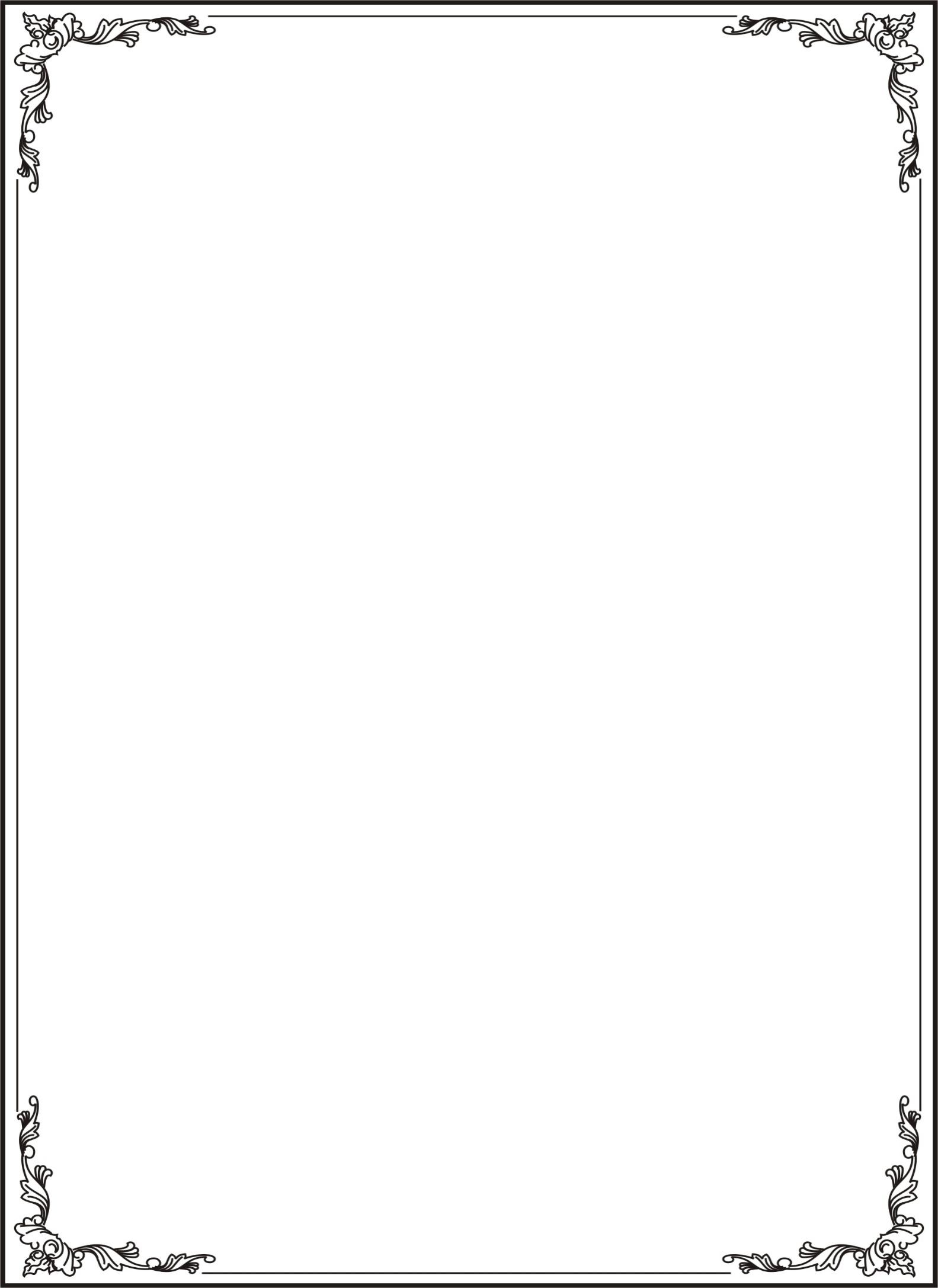
**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

🙠🙟🕮🙝🙢



**TIỂU LUẬN CUỐI KỲ**

**MÔN HỌC: BIG DATA ESSENTIALS**

**ĐỀ TÀI TIỂU LUẬN**

**TÌM HIỂU CƠ SỞ DỮ LIỆU PHI QUAN HỆ. TÌM HIỂU, CÀI ĐẶT VÀ SỬ DỤNG RIAK**

**GVHD: ThS. Lê Thị Minh Châu**

**Mã học phần: BDES333877**

**NHÓM SINH VIÊN THỰC HIỆN:**

1. **Trần Nguyễn Trí Đạt – 21110162**
2. **Phạm Tấn Huy – 21110190**
3. **Lê Nguyễn Trí Nhân – 21110841**
4. **Nguyễn Trần Huỳnh Lê – 21110229**

**Thành phố Hồ Chí Minh, Tháng 05 năm 2024**

**DANH SÁCH THÀNH VIÊN THAM GIA VIẾT TIỂU LUẬN**

**HỌC KÌ II, NĂM HỌC 2023–2024**

**Mã học phần: BDES333877**

**Tên đề tài:** Tìm hiểu cơ sở dữ liệu phi quan hệ. Tìm hiểu, cài đặt và sử dụng riak

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **HỌ VÀ TÊN SINH VIÊN** | **MÃ SỐ SINH VIÊN** | **TỈ LỆ %**  **HOÀN THÀNH** |
| **1** | Trần Nguyễn Trí Đạt | 21110162 | 100% |
| **2** | Phạm Tấn Huy | 21110190 | 100% |
| **3** | Lê Nguyễn Trí Nhân | 21110841 | 100% |
| **4** | Nguyễn Trần Huỳnh Lê | 21110229 | 100% |

***Ghi chú:***

Tỷ lệ %: Mức độ phần trăm hoàn thành của từng sinh viên tham gia.

**Trưởng nhóm:** Trần Nguyễn Trí Đạt

**Nhận xét của giảng viên:**

*Tp. Hồ Chí Minh – Tháng 05 năm 2024*

**PHÂN CHIA CÔNG VIỆC**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên** | **MSSV** | **Công việc** | **Hoàn thành** |
| Trần Nguyễn Trí Đạt | 21110162 | * Tìm hiểu ứng dụng của Riak, tìm hiểu về kiểu dữ liệu Maps trong Riak. * Thực hiện demo Create (tạo, thêm dữ liệu), Update (sửa dữ liệu trên Riak) * Thực hiện demo truy vấn dữ liệu * Thực hiện demo mô hình hóa đối tượng | 100% |
| Phạm Tấn Huy | 21110190 | * Tìm hiểu khái niệm, ưu nhược điểm của Riak, tìm hiểu về kiểu dữ liệu Sets của Riak. * Thực hiện demo Delete (xóa dữ liệu) * Thực hiện demo truy vấn dữ liệu bằng secondary indexes | 100% |
| Lê Nguyễn Trí Nhân | 21110841 | * Tìm hiểu ưu, nhược điểm của NoSQL và so sánh với CSDL quan hệ, tìm hiểu về kiểu dữ liệu HyperLogLogs. * Tìm hiểu cấu hình Hadoop cluster window * Thực hiện demo Update (sửa dữ liệu trên Riak) | 100% |
| Nguyễn Trần Huỳnh Lê | 21110229 | * Tìm hiểu về lịch sử, khái niệm, đặc điểm của cở sở dữ liệu phi quan hệ, tìm hiểu về kiểu dữ liệu Counters của Riak. * Thực hiện demo Read (đọc và truy vấn dữ liệu) | 100% |

**LỜI NÓI ĐẦU**

Trong thời đại ngày nay, công nghệ thông tin đã trở thành một phần không thể thiếu trong mọi lĩnh vực của cuộc sống. Với hệ thống máy tính, chúng ta có thể thực hiện rất nhiều công việc một cách hiệu quả, tiết kiệm thời gian, công sức và tiền bạc. Sự phát triển vượt bậc của Internet đã tạo ra một lượng thông tin khổng lồ, và việc này có thể được coi là tăng theo cấp số nhân. Theo một nghiên cứu, lượng tri thức của nhân loại được dự đoán sẽ tăng gấp đôi mỗi năm, và trong bối cảnh này, việc tìm kiếm và khai thác thông tin trở nên vô cùng quan trọng, mang lại nhiều lợi ích về khoa học và kinh tế.

Dữ liệu lớn đã trở thành một thách thức lớn đối với các tổ chức và doanh nghiệp. Việc lưu trữ, quản lý và phân tích dữ liệu lớn đòi hỏi các giải pháp mạnh mẽ và hiệu quả để giải quyết vấn đề này. Trong ngữ cảnh này, Hadoop là một trong những công nghệ phổ biến nhất được sử dụng để xử lý dữ liệu lớn. Với khả năng lưu trữ và xử lý dữ liệu lớn, Hadoop được áp dụng rộng rãi trong nhiều lĩnh vực như tài chính, y tế, giáo dục và công nghệ.

Tuy nhiên, việc triển khai và quản lý một cluster Hadoop trên hệ điều hành Windows vẫn là một thách thức. Mặc dù Hadoop thường được triển khai trên các hệ điều hành dựa trên Unix như Linux, nhưng với nhu cầu của nhiều tổ chức sử dụng Windows, việc triển khai trên nền tảng này trở nên quan trọng. Ngoài ra, Riak là một hệ thống cơ sở dữ liệu phân tán mạnh mẽ, thường được sử dụng để lưu trữ và quản lý dữ liệu lớn. Mặc dù Ubuntu thường được sử dụng làm hệ điều hành cho các hệ thống phân tán như Riak, nhưng việc cài đặt và quản lý Riak trên một môi trường hệ điều hành khác như Windows hoặc Ubuntu cũng là một phần quan trọng của quá trình nghiên cứu và triển khai.

Do đó, việc tìm hiểu và thực hiện cài đặt cluster Hadoop trên hệ điều hành Windows, cùng với việc khám phá và thực hiện thao tác CRUD trên cơ sở dữ liệu Riak trên Ubuntu, đều đem lại cái nhìn tổng quan về khả năng tích hợp và hiệu suất của các giải pháp này trong môi trường phân tán đa dạng.

**MỤC LỤC**

[DANH MỤC TỪ VIẾT TẮT 5](#_Toc166101782)

[1. PHẦN MỞ ĐẦU 6](#_Toc166101783)

[**1.1. Mục đích đề tài** 6](#_Toc166101784)

[**1.2. Yêu cầu cần thực hiện** 6](#_Toc166101785)

[**1.3. Phạm vi đề tài** 6](#_Toc166101786)

[2. CƠ SỞ LÝ THUYẾT 7](#_Toc166101787)

[**2.1. Cơ sở dữ liệu phi quan hệ (NoSQL)** 7](#_Toc166101788)

[**2.1.1. Lịch sử, khái niệm, đặc điểm cửa cơ sở dữ liệu phi quan hệ** 7](#_Toc166101789)

[**2.1.2. Ưu nhược điểm của NoSQL** 8](#_Toc166101790)

[**2.1.3. So sánh SQL và NoSQL** 9](#_Toc166101791)

[**2.2. Cơ sở dữ liệu Riak** 12](#_Toc166101792)

[**2.2.1. Khái niệm, đặc điểm, lưu trữ dữ liệu của Riak** 12](#_Toc166101793)

[**2.2.2. Ưu nhược điểm của Riak** 13](#_Toc166101794)

[**2.2.3. Các kiểu dữ liệu của Riak** 14](#_Toc166101795)

[**2.2.4. Ứng dụng của Riak** 16](#_Toc166101796)

[3. CÀI ĐẶT VÀ SỬ DỤNG RIAK 20](#_Toc166101797)

[**3.1. Cài đặt Python trong ubuntu** 20](#_Toc166101798)

[**3.2. Cài đặt Riak trong ubuntu** 20](#_Toc166101799)

[**3.3. Sử dụng Riak single node trên python** 22](#_Toc166101800)

[**3.3.1. Create (tạo bucket trên Riak)** 22](#_Toc166101801)

[**3.3.2. Read (đọc, truy vấn dữ liệu)** 26](#_Toc166101802)

[**3.3.3. Update (sửa dữ liệu)** 28](#_Toc166101803)

[**3.3.4. Delete (xóa dữ liệu)** 30](#_Toc166101804)

[4. CẤU HÌNH HADOOP CLUSTER TRÊN WINDOW 33](#_Toc166101805)

[**4.1. Cài đặt JDK bản 1.8 (bắt buộc):** 33](#_Toc166101806)

[**4.2. Thiết lập biến môi trường cho Java JDK** 34](#_Toc166101807)

[**4.3. Tải Hadoop và giải nén vào ổ C** 36](#_Toc166101808)

[**4.4. Thiết lập biến môi trường cho Hadoop** 36](#_Toc166101809)

[**4.5. Cấu hình các tập tin cho Hadoop** 37](#_Toc166101810)

[**4.5.1. Máy master** 37](#_Toc166101811)

[**4.5.2. Cấu hình máy slave** 46](#_Toc166101812)

[**4.5.3. Kết quả sau khi start cả máy master và máy slave** 55](#_Toc166101813)

[5. KẾT LUẬN 58](#_Toc166101814)

[**5.1. Các kết quả đạt được** 58](#_Toc166101815)

[**5.2. Ưu điểm và hạn chế của đề tài** 58](#_Toc166101816)

[**5.3. Định hướng phát triển** 58](#_Toc166101817)

[TÀI LIỆU THAM KHẢO 59](#_Toc166101818)

# **DANH MỤC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **Ký hiệu chữ viết tắt** | **Chữ viết đầy đủ** |
| 1 | CSDL | Cơ sở dữ liệu |
| 2 | LAN | Local Area Network: mạng cục bộ |
| 3 | NoSQL | Cơ sở dữ liệu phi quan hệ |
| 4 | SQL | Cơ sở dữ liệu quan hệ |
| 5 | CRUD | Create, Read, Update, Delete |
| 6 | HDFS | Hadoop Distributed File System |
| 7 | API | application programming interface: giao diện lập trình ứng dụng |
| 8 | Riak kv | Riak key value |
| 9 | MB | Megabyte |
| 10 | JSON | JavaScript Object Notation |

## **PHẦN MỞ ĐẦU**

### **Mục đích đề tài**

Đề tài này nhằm mục đích nghiên cứu và thực hiện việc triển khai một cluster Hadoop trên hệ điều hành Windows và khám phá các thao tác CRUD trên cơ sở dữ liệu phân tán Riak trên Ubuntu. Mục tiêu chính là hiểu rõ quá trình cài đặt và quản lý cluster Hadoop trên Windows cũng như tích hợp và thao tác với cơ sở dữ liệu Riak trên Ubuntu.

### **Yêu cầu cần thực hiện**

Nghiên cứu về kiến trúc và cách thức hoạt động của Hadoop, đặc biệt là trong môi trường Windows.

Thực hiện cài đặt và cấu hình một cluster Hadoop trên hệ điều hành Windows, bao gồm cài đặt các thành phần như Hadoop Distributed File System (HDFS), YARN và MapReduce.

Tìm hiểu về cơ sở dữ liệu Riak và quá trình triển khai trên Ubuntu.

Phân tích và thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên Riak, bao gồm tạo, đọc, cập nhật và xóa dữ liệu, từ một môi trường phân tán.

### **Phạm vi đề tài**

Phạm vi của đề tài sẽ tập trung vào việc cài đặt, cấu hình và thực hiện thao tác kết nói các node khác nhau trên cluster Hadoop trên hệ điều hành Windows, cũng như cài đặt và triển khai cơ sở dữ liệu Riak trên Ubuntu. Nghiên cứu sẽ phản ánh khả năng tích hợp của hai công nghệ này trong một môi trường phân tán đa dạng. Tuy nhiên, đề tài sẽ không đi sâu vào việc tối ưu hóa hiệu suất hoặc xử lý dữ liệu lớn.

## **CƠ SỞ LÝ THUYẾT**

### **Cơ sở dữ liệu phi quan hệ (NoSQL)**

#### **Lịch sử, khái niệm, đặc điểm cửa cơ sở dữ liệu phi quan hệ**

1. **Lịch sử cơ sở dữ liệu phi quan hệ**

Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên gọi chung cho các lightweight open source relational database (cơ sở dữ liệu quan hệ nguồn mở nhỏ) nhưng không sử dụng SQL cho truy vấn. Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL trong một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc). Đây là 2 đặc tính quan trọng nhất.

1. **Khái niệm cơ sở dữ liệu phi quan hệ**

NoSQL là một hệ thống quản lý dữ liệu không quan hệ – non-relational Data Management System sở hữu lược đồ linh hoạt. Đặc trưng của cơ sở dữ liệu NoSQL chính là dễ dàng mở rộng khi cần thiết. Bởi thế, việc sử dụng thường được ứng dụng cho những kho dữ liệu phân tán với khả năng lưu trữ lượng dữ liệu lớn, hay những ứng dụng web dựa trên thời gian thực.

1. **Đặc điểm cơ sở dữ liệu phi quan hệ**

Các đặc điểm của NoSQL:

* Phi quan hệ: không có ràng buộc nào cho việc nhất quán dữ liệu.
* Mô hình lưu trữ phân tán các tập tin hoặc dữ liệu ra nhiều máy khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mềm.
* NoSQL lưu trữ dữ liệu của mình theo dạng cặp giá trị “key – value”. Sử dụng số lượng lớn các node để lưu trữ thông tin.
* Tính nhất quán không theo thời gian thực: Sau mỗi thay đổi CSDL, không cần tác động ngay đến tất cả các CSDL liên quan mà được lan truyền theo thời gian.
* Mô hình dữ liệu và truy vấn linh hoạt.
* Triển khai đơn giản, dễ nâng cấp và mở rộng.

#### **Ưu nhược điểm của NoSQL**

1. **Ưu điểm:**

* Hiệu suất cao: CSDL NoSQL được tối ưu hóa cho các mô hình dữ liệu cụ thể và các mẫu truy cập. Các vấn đề liên quan tới tới dữ liệu lớn (truy xuất dữ liệu từ nhiều nguồn khác nhau).
* Mở rộng tốt: trong môi trường phân tán NoSQL thường được thiết kế để mở rộng trong hệ thống quan hệ bằng cách sử dụng các cụm phần cứng đuược phân phối thay vì tăng quy mô bằng việc bổ sung máy chủ mạnh và tốn kém
* Tích hợp dữ liệu không đồng nhất: có thể lưu trữ và truy vấn dữ liệu không có cấu trúc hoặc cấu trúc khác nhau mà không cần định rõ cấu trúc trước.
* Linh Hoạt: sơ đồ linh hoạt giúp quá trình phát triển nhanh chóng, cũng đảm bảo ở khả năng lặp lại cao hơn rất nhiều. Chính mô hình dữ liệu có khả năng linh hoạt cao sẽ biến cơ sở dữ liệu NoSQL trở thành sự lựa chọn hoàn hảo cho những dữ liệu ở dạng không được tổ chức thành cấu trúc, hay có cấu trúc chưa thực sự hoàn chỉnh. khả năng thích ứng và mở rộng với các mô hình dữ liệu và yêu cầu kinh doanh khác nhau.
* Chức năng cao: CSDL NoSQL cung cấp các API và kiêu dữ liệu cực kỳ thiết thực được xây dựng riêng cho từng mô hình dữ liệu trong từng mô hình tương ứng.Nó đem lại tính ứng dụng cao hơn rất nhiều trong quá trình sử dụng.

1. **Nhược điểm:**

* Quản lý dữ liệu: việc quản lý dữ liệu của NoSQL phức tạp hơn nhiều so với SQL. Đặc biệt NoSQL rất khó cài đặt và để quản lý hàng ngày cũng rất tốn thời gian.
* Sao lưu dữ liệu:sao lưu là một điểm yếu lớn đối với một số cơ sở dữ liệu NoSQL. Nó không có cách tiếp cận để làm sao lưu dữ liệu một cách nhất quán.
* Thiếu tính nhất quán: NoSQL đánh đổi sự nhất quán để ưu tiên tốc độ, hiệu suất hiệu quả hơn.
* Trọng tâm hẹp: Cơ sở dữ liệu NoSQL có trọng tâm rất hẹp vì nó chủ yếu được thiết kế để lưu trữ nhưng nó cung cấp rất ít chức năng.
* Mã nguồn mở: NoSQL là cơ sở dữ liệu mã nguồn mở và không có tiêu chuẩn đáng tin cậy cho NoSQL được nêu ra.
* Không có lược đồ: Ngay cả khi bạn lấy dữ liệu ở dạng tự do, bạn hầu như luôn cần áp đặt các ràng buộc để làm cho nó hữu ích. Với NoSQL, trách nhiệm sẽ được chuyển từ cơ sở dữ liệu sang nhà phát triển, lập trình ứng dụng.
* Kỹ năng NoSQL: Một hạn chế khác đối với NoSQL là người sử dụng có thể sẽ thiếu các kỹ năng chuyên môn ở mức tương đối vì hệ thống này còn khá mới và không phải ai cũng biết sử dụng nó một cách thành thạo.

#### **So sánh SQL và NoSQL**

|  |  |  |
| --- | --- | --- |
|  | **SQL** | **NoSQL** |
| **Định nghĩa** | Cơ sở dữ liệu SQL chủ yếu được gọi là RDBMS hoặc Cơ sở dữ liệu quan hệ | Cơ sở dữ liệu NoSQL chủ yếu được gọi là cơ sở dữ liệu không liên quan hoặc phân tán |
| **Design for** | RDBMS truyền thống sử dụng cú pháp và truy vấn SQL để phân tích và lấy dữ liệu để có thêm thông tin chi tiết. Chúng được sử dụng cho các hệ thống OLAP. | Hệ thống cơ sở dữ liệu NoSQL bao gồm nhiều loại công nghệ cơ sở dữ liệu khác nhau. Các cơ sở dữ liệu này được phát triển để đáp ứng nhu cầu trình bày cho sự phát triển của ứng dụng hiện đại. |
| **Ngôn ngữ Query** | Structured query language (SQL) | Không có ngôn ngữ query |
| **Type** | SQL databases là cơ sở dữ liệu dựa trên bảng | NoSQL databases có thể dựa trên tài liệu, cặp khóa-giá trị, cơ sở dữ liệu biểu đồ |
| **Schema** | SQL databases có lược đồ được xác định trước | NoSQL databases sử dụng lược đồ động cho dữ liệu phi cấu trúc. |
| **Khả năng mở rộng** | SQL databases có thể mở rộng theo chiều dọc | NoSQL databases có thể mở rộng theo chiều ngang |
| **Phù hợp cho** | Đây là 1 lựa chọn lý tưởng cho môi trường truy vấn phức tạp | Không phù hợp với truy vấn phức tạp |
| **Lưu trữ dữ liệu phân cấp** | SQL databases không thích hợp cho việc lưu trữ dữ liệu phân cấp. | Phù hợp hơn cho kho lưu trữ dữ liệu phân cấp vì nó hỗ trợ phương thức cặp khóa-giá trị. |
| **Variations** | Một loại có biến thể nhỏ | Nhiều loại khác nhau bao gồm các kho khóa-giá trị, cơ sở dữ liệu tài liệu và cơ sở dữ liệu đồ thị. |
| **Năm phát triển** | Nó được phát triển vào những năm 1970 để giải quyết các vấn đề với lưu trữ tệp phẳng | Được phát triển vào cuối những năm 2000 để khắc phục các vấn đề và hạn chế của SQL databases. |
| **Open-source** | Một sự kết hợp của mã nguồn mở như Postgres & MySQL, và thương mại như Oracle Database. | Open-source |
| **Tính nhất quán** | Nó phải được cấu hình cho sự nhất quán chặt chẽ. | Nó phụ thuộc vào DBMS như một số cung cấp tính nhất quán mạnh mẽ như MongoDB, trong khi những người khác cung cấp chỉ cung cấp sự nhất quán cuối cùng, như Cassandra. |
| **Được sử dụng tốt nhất cho** | RDBMS database là tùy chọn thích hợp để giải quyết các vấn đề về ACID. | NoSQL được sử dụng tốt nhất để giải quyết các vấn đề về tính khả dụng của dữ liệu |
| **Tầm quan trọng** | Nó nên được sử dụng khi hiệu lực dữ liệu là siêu quan trọng | Sử dụng khi nó quan trọng hơn để có dữ liệu nhanh hơn dữ liệu chính xác |
| **Lựa chọn tốt nhất** | Khi bạn cần hỗ trợ truy vấn động | Sử dụng khi bạn cần mở rộng quy mô dựa trên yêu cầu thay đổi |
| **Hardware** | Specialized DB hardware (Oracle Exadata, etc.) | Commodity hardware |
| **Network** | Highly available network (Infiniband, Fabric Path, etc.) | Commodity network (Ethernet, etc.) |
| **Loại lưu trữ** | Highly Available Storage (SAN, RAID, etc.) | Commodity drives storage (standard HDDs, JBOD) |
| **Tính năng tốt nhất** | Hỗ trợ đa nền tảng, Bảo mật và miễn phí | Dễ sử dụng, hiệu suất cao và công cụ linh hoạt. |
| **Mô hình ACID và BASE** | ACID (Atomicity, nhất quán, cách ly và độ bền) là một chuẩn cho RDBMS | Cơ bản (Về cơ bản có sẵn, trạng thái mềm, phù hợp cuối cùng) là một mô hình của nhiều hệ thống NoSQL |
| **Performance** | SQL hoạt động tốt và nhanh thì việc desgin tốt là cực kì quan trọng và ngược lại. | Nhanh hơn SQL NoSQL thì denormalized cho phép bạn lấy được tất cả thông tin về một item cụ thể với các codition mà không cần JOIN liên quan hoặc truy vấn SQL phức tạp. |
| **Kết luận** | Dự án đã có yêu cầu dữ liệu rõ ràng xác định quan hệ logic có thể được xác định trước. | Phù hợp với những dự án yêu cầu dữ liệu không liên quan, khó xác định, đơn giản mềm dẻo khi đang phát triển |

**Bảng 1: so sánh giữa SQL và NoSQL**

### **Cơ sở dữ liệu Riak**

#### **Khái niệm, đặc điểm, lưu trữ dữ liệu của Riak**

1. **Khái niệm của Riak**

Riak là một loại cơ sở dữ liệu phân tán được xây dựng bởi Basho Technology để có thể tối đa hóa quá trình chuyển dữ liệu tới người dùng qua nhiều server. Riak được chia làm 2 phần chính: Riak server và Riak client. Khi Riak client có thể kết nối tới một server là có thể truy cập dữ liệu.

Với Riak chúng ta có thể xây dựng hệ thống theo hai mô hình chính:

* Eventually consistent system: Dữ liệu mong muốn nên được lưu trữ để có thể sẵn sàng trong hầu hết các kịch bản xảy ra sự cố mặc dù dữ liệu này có thể không phải là dữ liệu được cập nhật mới nhất.
* Strongly consistent system : với mô hình hệ thống này, quá trình đọc dữ liệu sẽ trả về dữ liệu gần như là mới nhất nhưng một vài nút sẽ có thể không sẵn sàng để ghi dữ liệu này vào.

1. **Đặc điểm của Riak**

Một số đặc điểm của Riak:

* Availability: Riak có thể ghi và đọc dữ liệu từ nhiều server và đề nghị dữ liệu sẵn sàng trong các trường hợp phần cứng hoặc network gặp sự cố
* Operational simplicity :Dễ dàng để thêm mới một riak cluster mà không ảnh hưởng tới toàn bộ hệ thống
* Scalibility: Riak tự động phân tán dữ liệu xung quanh các cluster và yield với hiệu năng tăng dần theo khối lượng
* Masterless: Request nếu không được giữ trong một serrver cụ thể nào đó thì có nghĩa là nó không tồn tại

1. **Lưu trữ dữ liệu của Riak**

Trong Riak, dữ liệu được tự động lưu phân tán trên các node sử dụng phương pháp consistent hashing nghĩa là chia nhỏ dữ liệu và phân bố đều trên các node. Consistent hashing đảm bảo dữ liệu được phân tán đều trong nhóm các node và các node có thể thêm vào tự động, tối thiểu quá quá trình sắp xếp lại dữ liệu. Điều này đã làm giảm đi đáng kể những điểm nóng trong cơ sở dữ liệu, dễ dàng tăng quy mô của hệ thống.

Riak lưu trữ dữ liệu sử dụng mô hình Object {key: value}. Các object thuộc cùng một namspace sẽ được nhóm lại với nhau thành các bucket.

Khi ghi một key mới vào Riak, đối tượng bucket/key sẽ được chia thành các phần nhỏ. Giá trị trả về sẽ được gán vào 160-bit integer và 160-bit này sẽ được tổ chức thành vòng tròn (ring) để có thể quyết định server vật lý nào mà dữ liệu sẽ được lưu trữ. Ở đây, Riak sẽ chia tổng dung lượng thành nhiều phân mảng nhỏ có dung lượng bằng nhau (thông thường sẽ chia làm 64 phần). Mỗi phân mảnh sẽ được nhận một khoảng giá trị nằm trên vòng tròn và có trách nhiệm lưu trữ cho tất cả các key/bucket. Node ảo ( vnode) là quá trình quản lý mỗi phần của những phân mảnh nằm trên các nút vật lý khác nhau. Nói một cách đơn giản rằng, mỗi một server vật lý sẽ sở hữu nhiều vnode

Quá trình chia nhỏ dữ liệu cũng như chia sẻ trách nhiệm lưu trữ các key trên nhiều node đảm bảo rằng dữ liệu trên Riak sẽ được phân tán đều. Khi một server vật lý mới được thêm vào, thiết bị này sẽ phải tính toán lại lượng dữ liệu hiện giờ đang sở hữu và sẽ yêu cầu phân mảng dữ liệu cho tới khi lượng dữ liệu này được cân bằng với các thiết bị khác. Để giúp đỡ quá trình này, Riak có sử dụng giao thức trao đổi (gossip protocol) và định hướng dữ liệu (guide for routing request).

#### **Ưu nhược điểm của Riak**

**Ưu điểm của Riak:**

* Xử lý dữ liệu không thay đổi (Immutable data): Riak giải quyết tốt các vấn đề xung đột dữ liệu trong các đối tượng không thay đổi, đồng thời giảm thiểu các yêu cầu xử lý đồng thời và tương tác với hệ thống bên ngoài.
* Đối tượng nhỏ (Small objects): Riak được thiết kế cho việc lưu trữ các đối tượng nhỏ như JSON, log file, sensor data, HTML file, giúp tăng cường hiệu suất.
* Độc lập (Independent objects): Riak phù hợp với các đối tượng độc lập mà không có sự phụ thuộc vào các đối tượng khác.
* Sử dụng key tự nhiên (Objects with "natural" keys): Riak khuyến khích việc sử dụng các key tự nhiên để tăng cường hiệu suất tìm kiếm.

**Nhược điểm của Riak:**

* Giới hạn dung lượng (Size limitations): Riak không thích hợp cho việc lưu trữ các đối tượng với dung lượng lớn hơn 1-2 MB, gây ra hiệu suất thấp.
* Cấu trúc phụ thuộc phức tạp (Complex dependency structures): Riak không phù hợp cho các đối tượng có cấu trúc phụ thuộc phức tạp, khiến việc quản lý trở nên khó khăn.

#### **Các kiểu dữ liệu của Riak**

1. **Kiểu dữ liệu Counters:**

Riak Counter là một tính năng trong Riak KV (Riak Key-Value Store) cho phép bạn lưu trữ và cập nhật các giá trị số nguyên phân tán. Điều này hữu ích khi bạn cần theo dõi các bước đếm, ví dụ như số lượt xem của một bài viết trên blog, số lượt click trên một nút, hoặc bất kỳ loại đếm nào khác mà bạn muốn theo dõi trong một môi trường phân tán.

Điểm đặc biệt của Riak Counter là khả năng cập nhật giá trị của nó một cách phân tán mà không cần đồng bộ hóa trên một nút duy nhất. Điều này là do Riak sử dụng các kỹ thuật phân tán như CRDT (Conflict-free Replicated Data Types) để đảm bảo tính nhất quán của dữ liệu trên nhiều nút.

Riak Counter là một loại CRDT (convergent replicated data type) mà cuối cùng sẽ hội tụ đến tổng số đúng. Bạn chỉ cần tăng giá trị của counter bằng một số nguyên, và bất kỳ xung đột nào có thể sẽ được giải quyết tự động bởi Riak.

1. **Kiểu dữ liệu HyperLogLogs**

Kiểu dữ liệu HyperLogLogs của Riak KV là một cơ chế lưu trữ dữ liệu đặc biệt được sử dụng để ước lượng số lượng các phần tử duy nhất trong một tập hợp lớn mà không cần lưu trữ tất cả các phần tử đó. HyperLogLogs hoạt động bằng cách sử dụng một cấu trúc dữ liệu xấp xỉ để đánh giá số lượng các phần tử duy nhất.

Cách hoạt động cơ bản của HyperLogLogs là thông qua việc sử dụng các hàm băm không đụng độ (non-colliding hash functions) để chia các giá trị đầu vào thành các "buckets" (thùng). Mỗi bucket được sử dụng để lưu trữ thông tin về các giá trị hash duy nhất mà nó nhận được. Khi có một giá trị mới được thêm vào, HyperLogLogs sẽ so sánh giá trị hash của nó với các giá trị đã được lưu trữ trong các bucket và cập nhật hoặc bỏ qua nó dựa trên kết quả của so sánh.

Tính chất chính của HyperLogLogs là khả năng ước lượng số lượng phần tử duy nhất trong một tập hợp lớn mà không cần lưu trữ tất cả các phần tử đó. Điều này làm cho HyperLogLogs trở thành một công cụ hữu ích trong việc theo dõi sự đa dạng và phân bố của dữ liệu mà không tốn kém nhiều tài nguyên.

1. **Kiểu dữ liệu Sets**

Sets là một tập hợp các giá trị nhị phân duy nhất, như các chuỗi. Tất cả các giá trị trong một tập hợp đều là duy nhất. Ví dụ, nếu bạn cố gắng thêm phần tử "shovel" vào một tập hợp đã chứa "shovel", thì thao tác đó sẽ bị bỏ qua bởi Riak KV. Các tập hợp có thể được sử dụng độc lập hoặc được nhúng trong một bản đồ.

*Một số ví dụ về việc sử dụng tập hợp:*

* Lưu trữ UUID của các bạn bè của người dùng trong một ứng dụng mạng xã hội.
* Lưu trữ các mặt hàng trong một giỏ hàng mua sắm điện tử.

*Các thao tác trên Sets:*

Sets phụ thuộc vào bốn phép toán cơ bản: thêm một phần tử, loại bỏ một phần tử, thêm nhiều phần tử hoặc loại bỏ nhiều phần tử.

1. **Kiểu dữ liệu Maps**

Trong Riak, kiểu dữ liệu “maps” là một loại dữ liệu phân tán rất linh hoạt, vì nó cho phép các kiểu dữ liệu khác nhau được nhúng bên trong nó, kể cả chính nó (maps bên trong maps). Điều này giúp tạo ra các kiểu dữ liệu phức tạp, tùy chỉnh từ một số khối xây dựng cơ bản. “Maps” trong Riak có thể chứa các kiểu dữ liệu như counters, sets, và registers, và cho phép thực hiện các thao tác phức tạp hơn so với việc chỉ lưu trữ và truy xuất các cặp khóa/giá trị đơn giản

Maps là lựa chọn tốt nhất cho dữ liệu phức tạp, đa chiều. Mã giả dưới đây lấy cảm hứng từ JSON để minh họa cách một bài đăng trên tweet có thể được cấu trúc dưới dạng một map:

|  |
| --- |
| Map tweet {  Counter: numberOfRetweets,  Register: username,  Register: tweetContent,  Flag: favorited?,  Map: userInfo  } |

*Các thao tác trên maps:*

Có thể thực hiện 2 loại thao tác trên maps:

* Các thao tác được thực hiện trực tiếp trên bản đồ, bao gồm thêm trường vào và xóa trường khỏi bản đồ (ví dụ với mã giả nêu trên có thể thêm flag hoặc xóa counter)
* Các thao tác được thực hiện trên các Kiểu dữ liệu được lồng trong maps, ví dụ: tăng bộ đếm trong maps hoặc đặt cờ thành enable. Các thao tác đó hoạt động giống như các thao tác cụ thể cho kiểu dữ liệu đó.

#### **Ứng dụng của Riak**

Riak là một công nghệ lưu trữ dữ liệu linh hoạt có khả năng giải quyết nhiều vấn đề khác nhau theo cách có thể mở rộng. Sau đây là một số ứng dụng riak đã được sử dụng trong sản xuất cho các dự án lớn và nhỏ:

*Đọc/ghi cao, ứng dụng đơn giản:*

* Lưu trữ phiên: Riak ban đầu được tạo ra để làm nơi lưu trữ phiên làm việc có khả năng mở rộng cao. Đây là một trường hợp sử dụng lý tưởng cho Riak, vì nó luôn hoạt động hiệu quả và dự đoán được khi được sử dụng như một cơ sở dữ liệu kiểu khóa/giá trị. Vì ID người dùng và phiên làm việc thường được lưu trong cookie hoặc đã biết khi tìm kiếm, Riak có thể xử lý những yêu cầu này với độ trễ thấp có thể dự đoán được. Sự không phân biệt loại nội dung của Riak cũng không hạn chế giá trị, vì vậy dữ liệu phiên có thể được mã hóa theo nhiều cách và có thể phát triển mà không cần thay đổi cấu trúc dữ liệu bởi quản trị viên.
* Phục vụ quảng cáo: Riak thường là lựa chọn tốt để phân phát nội dung quảng cáo tới nhiều người dùng web và thiết bị di động khác nhau cùng lúc với độ trễ thấp. Nội dung thuộc loại này, ví dụ: hình ảnh hoặc văn bản, có thể được lưu trữ trong Riak bằng cách sử dụng nội dung duy nhất do ứng dụng hoặc Riak tạo ra. Ví dụ: có thể tạo khóa dựa trên chiến dịch hoặc ID công ty để dễ dàng truy xuất.
* Dữ liệu nhật ký: Trường hợp sử dụng phổ biến của Riak là lưu trữ lượng lớn dữ liệu nhật ký, để phân tích bằng MapReduce hoặc dưới dạng hệ thống lưu trữ được sử dụng cùng với cụm phân tích thứ cấp được sử dụng để thực hiện các tác vụ phân tích nâng cao hơn. Để lưu trữ dữ liệu nhật ký, bạn có thể sử dụng một nhóm có tên logs(chỉ để đưa ra ví dụ) và sử dụng một giá trị duy nhất, chẳng hạn như ngày, cho khóa. Các tệp nhật ký khi đó sẽ là các giá trị được liên kết với mỗi khóa duy nhất.
* Dữ liệu cảm biến: Thiết kế có thể mở rộng của Riak giúp nó trở nên hữu ích đối với các tập dữ liệu, như dữ liệu cảm biến, có quy mô nhanh chóng và chịu tải đọc/ghi lớn. Nhiều cảm biến thu thập và gửi dữ liệu trong một khoảng thời gian nhất định. Một cách để mô hình hóa điều này trong Riak là sử dụng một nhóm cho mỗi thiết bị cảm biến và sử dụng khoảng thời gian làm khóa duy nhất (tức là ngày hoặc sự kết hợp giữa ngày và giờ), sau đó lưu trữ dữ liệu cập nhật làm giá trị.

*Quản lý nội dung, ứng dụng xã hội:*

* Tài khoản người dùng: Tài khoản người dùng có xu hướng dựa vào các mô hình dữ liệu khá đơn giản. Một cách lưu trữ dữ liệu tài khoản người dùng trong Riak là lưu trữ dữ liệu của mỗi người dùng dưới dạng đối tượng JSON trong một nhóm có tên users (hoặc bất cứ thứ gì). Khóa cho các đối tượng dữ liệu người dùng có thể được xây dựng bằng cách xem xét các ứng dụng cụ thể. Ví dụ: nếu ứng dụng của liên quan đến thông tin đăng nhập của người dùng, chiến lược đơn giản và hiệu quả nhất sẽ là sử dụng tên người dùng đăng nhập làm khóa đối tượng. Tên người dùng có thể được trích xuất khi đăng nhập và yêu cầu đọc có thể được thực hiện trên khóa tương ứng.
* Cài đặt và tùy chọn người dùng: Đối với dữ liệu liên quan đến tài khoản người dùng đơn giản và thường xuyên được đọc nhưng hiếm khi thay đổi (chẳng hạn như cài đặt quyền riêng tư hoặc tùy chọn chủ đề), hãy cân nhắc việc lưu trữ dữ liệu đó trong chính đối tượng người dùng. Một mẫu phổ biến khác là tạo một loại đối tượng cài đặt người dùng đồng hành, với các khóa dựa trên ID người dùng để dễ dàng truy xuất một lần đọc.
* Sự kiện và dòng thời gian của người dùng: thực hiện mô hình hóa dữ liệu người dùng phức tạp hoặc cụ thể hơn. Một ví dụ phổ biến là lưu trữ dữ liệu để tạo ra một bảng tin mạng xã hội. Để tạo ra bảng tin người dùng, có thể sử dụng một bucket bảng tin trong Riak và hình thành các khóa dựa trên một ID người dùng duy nhất. Lưu trữ thông tin bảng tin như giá trị, ví dụ: một danh sách các ID cập nhật trạng thái có thể sau đó được sử dụng để truy xuất thông tin đầy đủ từ một bucket, hoặc có thể chứa thông tin cập nhật trạng thái đầy đủ. Nếu muốn lưu trữ thêm dữ liệu, chẳng hạn như một dấu thời gian, danh mục hoặc danh sách thuộc tính, tjif có thể chuyển đổi danh sách đó thành một mảng các băm chứa thông tin bổ sung này.
* Bài viết, bài đăng trên blog và nội dung khác: Cách đơn giản nhất để lập mô hình các bài đăng trên blog, bài viết hoặc nội dung tương tự là sử dụng một nhóm trong Riak với một số thuộc tính duy nhất để phân chia nội dung một cách hợp lý, chẳng hạn như blogs hoặc articles. Các khóa có thể được tạo từ các mã định danh duy nhất cho các bài đăng, có thể là tiêu đề của mỗi bài viết, sự kết hợp giữa tiêu đề và dữ liệu/thời gian, một số nguyên có thể được sử dụng như một phần của chuỗi URL, v.v.

## **CÀI ĐẶT VÀ SỬ DỤNG RIAK**

### **Cài đặt Python trong ubuntu**

**Bước 1: Cài đặt các gói thư viện cần thiết:**

* Cài đặt các gói thư viện và công cụ cần thiết cho hệ thống Linux:

|  |
| --- |
| # apt-get install build-essential checkinstall  # apt-get install libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev |

**Bước 2: Tải xuống Python 3.5.3:**

* Tải xuống kho lưu trữ mã nguồn Python từ trang web tải xuống chính thức của nó bằng lệnh sau:

|  |
| --- |
| # cd /usr/src  # wget https://www.python.org/ftp/python/3.5.3/Python-3.5.3.tgz |

* Giải nén kho lưu trữ đã tải xuống trên hệ thống

|  |
| --- |
| # tar xzf Python-3.5.3.tgz |

**Bước 3: Cài đặt Python trên máy ubuntu:**

|  |
| --- |
| # cd Python-3.5.3  # ./configure --enable-optimizations  # make altinstall |

* Lệnh make altinstall được sử dụng để tránh việc thay thế tệp nhị phân Python mặc định tại /usr/bin/python.

**Bước 4: Kiểm tra phiên bản python:**

|  |
| --- |
| # python3.5 -V |

### **Cài đặt Riak trong ubuntu**

# apt-get install libpam0g-dev

A screen shot of a computer

Description automatically generated

# apt-get install openssl

A screen shot of a computer

Description automatically generated

# wget https://files.tiot.jp/riak/kv/3.2/3.2.0/ubuntu/jammy64/riak\_3.2.0-OTP25\_amd64.deb

A screenshot of a computer program

Description automatically generated

# dpkg -i riak\_3.2.0-OTP25\_amd64.deb

A screenshot of a computer

Description automatically generated

### **Sử dụng Riak single node trên python**

#### **Create (tạo bucket trên Riak)**

|  |
| --- |
| **Create.py** |
| import riak  dataCN =[  {"MACN": "HCM",  "TENCN": "Sai Gon"},  { "MACN": "GL",  "TENCN": "Gia Lai"},  {"MACN": "LD",  "TENCN": "Lam Dong"}]  dataSP =[  {"MASP": "VR01",  "TENSP": "Varto",  "HANGSX": "Honda",  "HP": 150,  "NAMSX": 2020,  "GIA": 53000000,  "MACN": "SG"},  {"MASP": "AB01",  "TENSP": "Air Blade",  "HANGSX": "Honda",  "HP": 150,  "NAMSX": 2021,  "GIA": 56000000,  "MACN": "GL"},  {"MASP": "BMW01",  "TENSP": "SH1000RR",  "HANGSX": "BMW",  "HP": 1000,  "NAMSX": 2016,  "GIA": 600000000,  "MACN": "LD"}]  #Tao ket noi den node  client= riak.RiakClient()  bucketCN=client.bucket('CHINHANH')  bucketSP=client.bucket('SANPHAM')  #Insert du lieu CHINHANH  for cn in dataCN:  cn\_riak= bucketCN.new(cn.get('MACN'), data=cn)  cn\_riak.store()  #Insert du lieu SANPHAM  for sp in dataSP:  sp\_riak= bucketSP.new(sp.get('MASP'), data=sp)  sp\_riak= sp\_riak.add\_index('gia\_int', sp.get('GIA')) # Secondary Indexes  sp\_riak.store() |
| **Printall.py** |
| import riak  client = riak.RiakClient()  bucketCN = client.bucket('CHINHANH')  bucketSP = client.bucket('SANPHAM')  print("Chi Nhanh")  for key in bucketCN.get\_keys():  obj = bucketCN.get(key)  print(obj.data)  print("San Pham")  for key in bucketSP.get\_keys():  obj = bucketSP.get(key)  print(obj.data) |

**Giải thích:**

* import riak: Đoạn mã này import module riak, được sử dụng để kết nối và tương tác với cơ sở dữ liệu Riak.
* dataCN và dataSP: Đây là danh sách các từ điển chứa dữ liệu cho các chi nhánh (dataCN) và sản phẩm (dataSP). Mỗi từ điển đại diện cho một bản ghi và có các trường như "MACN" (Mã Chi Nhánh), "TENCN" (Tên Chi Nhánh), "MASP" (Mã Sản Phẩm), "TENSP" (Tên Sản Phẩm), vv.
* Tạo kết nối đến Riak node: client = riak.RiakClient() Đoạn mã này tạo một đối tượng RiakClient, đại diện cho kết nối tới một node của Riak. Các hoạt động về sau sẽ được thực hiện thông qua kết nối này.
* bucketCN = client.bucket('CHINHANH') và bucketSP = client.bucket('SANPHAM'): Đoạn mã này tạo hai đối tượng RiakBucket, mỗi đối tượng đại diện cho một bucket trong cơ sở dữ liệu Riak. Bucket là một bộ sưu tập các đối tượng dữ liệu có cùng một loại.
* Lặp qua danh sách dataCN và dataSP để chèn dữ liệu vào Riak:
  + for cn in dataCN: và for sp in dataSP: bắt đầu hai vòng lặp lần lượt qua danh sách chi nhánh và danh sách sản phẩm.
  + cn\_riak = bucketCN.new(cn.get('MACN'), data=cn): Đoạn mã này tạo một đối tượng RiakObject mới trong bucket CHINHANH với khóa (key) là giá trị của trường "MACN" trong từ điển chi nhánh hiện tại và dữ liệu (data) là từ điển chi nhánh đó.
  + cn\_riak.store(): Đoạn mã này lưu trữ đối tượng RiakObject vào cơ sở dữ liệu Riak.
  + Tương tự, đối với sản phẩm, sp\_riak = bucketSP.new(sp.get('MASP'), data=sp) tạo một đối tượng RiakObject mới trong bucket SANPHAM với khóa là giá trị của trường "MASP" trong từ điển sản phẩm hiện tại và dữ liệu là từ điển sản phẩm đó. sp\_riak.add\_index('gia\_int', sp.get('GIA')) thêm một chỉ số phụ (secondary index) với tên 'gia\_int' và giá trị là giá sản phẩm.
* Cuối cùng, sp\_riak.store() lưu trữ đối tượng RiakObject vào cơ sở dữ liệu Riak.
* Với file printall.py: lặp vòng lặp lấy key của bucketCN và bucketSP để in ra màn hình dữ liệu của 2 bảng

**Kết quả:**

* Tạo file create.py bằng lệnh nano create.py

**A screenshot of a computer

Description automatically generated**

* Kết quả thu được

**A screenshot of a computer program

Description automatically generated**

#### **Read (đọc dữ liệu)**

|  |
| --- |
| **Test1.py** |
| from riak import RiakClient  client = RiakClient()  bucketSP = client.bucket('SANPHAM')  print('San pham co gia tri lon hon 55000000:')  for key in bucketSP.get\_keys():  sanpham = bucketSP.get(key)  if sanpham.exists:  gia = sanpham.data.get('GIA')  if gia is not None and gia > 55000000:  print(sanpham.data) |
| Test2.py |
| from riak import RiakClient  client = RiakClient()  bucketSP = client.bucket('SANPHAM')  print('San pham co chi nhanh khac LAM Dong:')  for key in bucketSP.get\_keys():  sanpham = bucketSP.get(key)  if sanpham.exists:  chinhanh = sanpham.data.get('MACN')  if chinhanh is not None and chinhanh != 'LD':  print(sanpham.data) |

**Giải thích:**

**test1.py**

* Đầu tiên sẽ import một class có tên RiakClient từ thư viện riak from riak import RiakClient:
* Tiếp theo tạo một đối tượng client từ lớp RiakClient: client = RiakCilent() :
* Tiếp theo tạo một bucket có tên SANPHAM trong cơ sỡ dữ liệu riak : bucketSP = client.bucket('SANPHAM')
* print('san phan co gia tri lon hon 50000000'): In ra màn hình dòng lệnh San pham co gia tri lon hon 55000000 .
* for key in bucketSP.get\_keys():Tạo một vòng lặp lặp tất cả các key trong Bucket SANPHAM để lất dữ liệu tương ứng.
* sanpham = bucketSP.get(key):Đọc và lấy dữ liệu từ bucket SANPHAM dựa trên key hiện tại trong vòng lặp.
* if sanpham.exists: Kiểm tra xem dữ liệu đã tồn tại trong bucket hay không. Nếu tồn tại, tiếp tục xử lý.
* gia=sanpham.data.get.('GIA'): Lấy giá trị của trường dữ liệu có tên là ‘GIA’ từ đối tượng sanpham.
* if gia is not None and gia > 55000000: print(sanpham.data): Kiểm tra xem giá có giá trị và lớn hơn 55000000 hay không, nếu có In dữ liệu của sản phẩm lên màn hình.

**Test2.py**

* Đầu tiên sẽ tạo một class có tên RiakClient từ thư viện riak from riak import RiakClient:
* Tiếp theo tạo một đối tượng client từ lớp RiakClient: client = RiakCilent() :
* Tiếp theo tạo một bucket có tên SANPHAM trong cơ sỡ dữ liệu riak : bucketSP = client.bucket('SANPHAM')
* print(Cac san pham co chi nhanh khac LAM Dong'): In ra màn hình dòng lệnh Cac san pham co chi nhanh khac LAM Dong.
* for key in bucketSP.get\_keys():Tạo một vòng lặp lặp tất cả các key trong Bucket SANPHAM để lất dữ liệu tương ứng.
* chinhanh = bucketSP.get(key):Đọc và lấy dữ liệu từ bucket SANPHAM dựa trên key hiện tại trong vòng lặp.
* if sanpham.exists:Kiểm tra xem dữ liệu đã tồn tại trong bucket hay không. Nếu tồn tại, tiếp tục xử lý.
* chinhanh=sanpham.data.get.('MCN): Lấy giá trị của trường dữ liệu có tên là ‘MCN’ từ đối tượng sanpham.
* if chinhanh is not None and chinhanh != ‘LD’: print(sanpham.data): Kiểm tra xem chi nhánh có giá trị và khác mã chi nhanh là LD hay không, nếu có In dữ liệu của sản phẩm lên màn hình.

**Kết quả:**

* Sản phẩm có giá trị lớn hơn 55000000:

A black screen with white text

Description automatically generated

* Các sản phẩm có chi nhánh khác Lâm Đồng

A black screen with white text

Description automatically generated

#### **Update (sửa dữ liệu)**

|  |
| --- |
| **update.py** |
| import riak  # Tao ket noi den node Riak  client = riak.RiakClient()  bucketCN = client.bucket('CHINHANH')  bucketSP = client.bucket('SANPHAM')  # Update san pham co ma san pham BMW01  bmw01\_riak = bucketSP.get('BMW01')  bmw01\_riak.data['NAMSX'] = 2024  bmw01\_riak.store()  # Them chi nhanh  new\_branch\_data = {"MACN": "BH", "TENCN": "Bien Hoa"}  new\_branch\_riak = bucketCN.new(new\_branch\_data['MACN'], data=new\_branch\_data)  new\_branch\_riak.store()  # Them san pham tu chi nhanh moi vao co so du lieu  new\_product\_data = {"MASP": "BHSP01", "TENSP": "Bien Hoa Special", "HANGSX": "Bien Hoa Motors", "HP": 200, "NAMSX": 2023, "GIA": 65000000, "MACN": "BH"}  new\_product\_riak = bucketSP.new(new\_product\_data['MASP'], data=new\_product\_data)  new\_product\_riak = new\_product\_riak.add\_index('gia\_int', new\_product\_data['GIA']) # Secondary Indexes  new\_product\_riak.store() |

**Giải thích:**

* client = riak.RiakClient(): Tạo một kết nối tới node Riak bằng cách tạo một đối tượng RiakClient.
* bucketCN = client.bucket('CHINHANH'): Tạo một bucket Riak có tên 'CHINHANH' để lưu trữ thông tin về các chi nhánh.
* bucketSP = client.bucket('SANPHAM'): Tạo một bucket Riak có tên 'SANPHAM' để lưu trữ thông tin về các sản phẩm.
* bmw01\_riak = bucketSP.get('BMW01'): Lấy thông tin của sản phẩm có mã sản phẩm là 'BMW01' từ bucket 'SANPHAM'.
* bmw01\_riak.data['NAMSX'] = 2024: Cập nhật năm sản xuất của sản phẩm 'BMW01' thành 2024.
* bmw01\_riak.store(): Lưu trữ thông tin cập nhật của sản phẩm 'BMW01' vào cơ sở dữ liệu Riak.
* new\_branch\_riak = bucketCN.new(new\_branch\_data['MACN'], data=new\_branch\_data): Tạo một chi nhánh mới với mã chi nhánh 'BH' và thông tin chi nhánh tương ứng, sau đó lưu vào bucket 'CHINHANH'.
* new\_branch\_riak.store(): Lưu trữ thông tin chi nhánh mới vào cơ sở dữ liệu Riak.
* new\_product\_riak = bucketSP.new(new\_product\_data['MASP'], data=new\_product\_data): Tạo một sản phẩm mới với mã sản phẩm 'BHSP01' và thông tin sản phẩm tương ứng, sau đó lưu vào bucket 'SANPHAM'.
* new\_product\_riak = new\_product\_riak.add\_index('gia\_int', new\_product\_data['GIA']): Thêm một chỉ mục phụ (secondary index) có tên 'gia\_int' với giá sản phẩm tương ứng.
* new\_product\_riak.store(): Lưu trữ thông tin sản phẩm mới vào cơ sở dữ liệu Riak.

**Kết quả:**

**A screenshot of a computer

Description automatically generated**

#### **Delete (xóa dữ liệu)**

|  |
| --- |
| **del.py** |
| from riak import RiakClient  #Tao ket noi den node  client= RiakClient()  #bucket  bucketCN = client.bucket('CHINHANH')  bucketSP = client.bucket('SANPHAM')  #Doc du lieu  sanpham1 = bucketSP.get('VR01')  sanpham1.delete() |
| **Read.py** |
| from riak import RiakClient  #Tao ket noi den node  client= RiakClient()  #bucket  bucketCN = client.bucket('CHINHANH')  bucketSP = client.bucket('SANPHAM')  #Doc du lieu  sanpham1 = bucketSP.get('VR01')  print(sanpham1.data) |

**Giải thích:**

* from riak import RiakClient: import class RiakClient từ module riak. Class này cho phép tạo và quản lý kết nối tới cơ sở dữ liệu Riak.
* client = RiakClient(): Dòng này tạo một đối tượng RiakClient mới và lưu trữ trong biến client. Đối tượng này sẽ được sử dụng để thực hiện các thao tác với cơ sở dữ liệu Riak.
* bucketCN = client.bucket('CHINHANH'): tạo một "bucket" mới có tên là 'CHINHANH'. Riak sử dụng các bucket để lưu trữ dữ liệu. Mỗi bucket tương ứng với một nhóm các đối tượng dữ liệu.
* bucketSP = client.bucket('SANPHAM'): dòng này tạo một bucket mới có tên là 'SANPHAM'.
* sanpham1 = bucketSP.get('VR01'): Dòng này thực hiện việc lấy dữ liệu từ bucket 'SANPHAM' với khóa là 'VR01' và lưu trữ trong biến sanpham1. Cụ thể, nó lấy một đối tượng dữ liệu có khóa là 'VR01' từ bucket 'SANPHAM'.
* sanpham1.delete(): Dòng này thực hiện việc xóa đối tượng dữ liệu được lấy từ bucket 'SANPHAM' với khóa là 'VR01'. Sau khi dòng này được thực hiện, dữ liệu tương ứng sẽ không còn tồn tại trong cơ sở dữ liệu Riak nữa.
* print(sanpham1.data): Dòng này đọc sản phẩm có mã là “VR01”

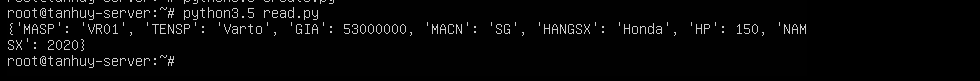
**Kết quả:**

* Tất cả các key trong 2 bucket:

A screenshot of a computer

Description automatically generated

* Đọc sản phẩm có key là “VR01” cũng là sản phẩm cần xóa



* Chạy file del.py và xem lại các 2 bucket đã thấy sản phẩm ‘VR01’ biến mất

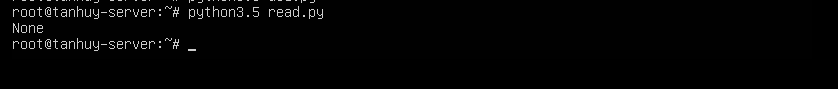
A black screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

* Đọc lại sản phẩm ‘VR01’ thì kết quả là none



#### **Querying (truy vấn dữ liệu)**

1. **Same Keys - Different Buckets**

|  |
| --- |
| **Create\_data\_querying.py** |
| import riak  # Creating Data  customer = {  'customer\_id': 1,  'name': "John Smith",  'address': "123 Main Street",  'city': "Columbus",  'state': "Ohio",  'zip': "43210",  'phone': "+1-614-555-5555",  'created\_date': "2013-10-01 14:30:26"  }  orders = [  {  'order\_id': 1,  'customer\_id': 1,  'salesperson\_id': 9000,  'items': [  {  'item\_id': "TCV37GIT4NJ",  'title': "USB 3.0 Coffee Warmer",  'price': 15.99  },  {  'item\_id': "PEG10BBF2PP",  'title': "eTablet Pro, 24GB, Grey",  'price': 399.99  }  ],  'total': 415.98,  'order\_date': "2013-10-01 14:42:26"  },  {  'order\_id': 2,  'customer\_id': 1,  'salesperson\_id': 9001,  'items': [  {  'item\_id': "OAX19XWN0QP",  'title': "GoSlo Digital Camera",  'price': 359.99  }  ],  'total': 359.99,  'order\_date': "2013-10-15 16:43:16"  },  {  'order\_id': 3,  'customer\_id': 1,  'salesperson\_id': 9000,  'items': [  {  'item\_id': "WYK12EPU5EZ",  'title': "Call of Battle: Goats - Gamesphere 4",  'price': 69.99  },  {  'item\_id': "TJB84HAA8OA",  'title': "Bricko Building Blocks",  'price': 4.99  }  ],  'total': 74.98,  'order\_date': "2013-11-03 17:45:28"  }]  order\_summary = {  'customer\_id': 1,  'summaries': [  {  'order\_id': 1,  'total': 415.98,  'order\_date': "2013-10-01 14:42:26"  },  {  'order\_id': 2,  'total': 359.99,  'order\_date': "2013-10-15 16:43:16"  },  {  'order\_id': 3,  'total': 74.98,  'order\_date': "2013-11-03 17:45:28"  }  ]  }  # Starting Client  client = riak.RiakClient(pb\_port=8087, protocol='pbc')  # Creating Buckets  customer\_bucket = client.bucket('Customers')  order\_bucket = client.bucket('Orders')  order\_summary\_bucket = client.bucket('OrderSummaries')  # Storing Data  cr = customer\_bucket.new(str(customer['customer\_id']),  data=customer)  cr.store()  for order in orders:  order\_riak = order\_bucket.new(str(order['order\_id']),  data=order)  order\_riak.store()  os = order\_summary\_bucket.new(str(order\_summary['customer\_id']),  data=order\_summary)  os.store() |
| **Querying.py** |
| import riak  client = riak.RiakClient(pb\_port=8087, protocol='pbc')  # Lay cac buckets  customer\_bucket = client.bucket('Customers')  order\_summary\_bucket = client.bucket('OrderSummaries')  # Lay du lieu khach hang va them tom tat don hang  customer = customer\_bucket.get('1').data  customer['order\_summary'] = order\_summary\_bucket.get('1').data  # In ra du lieu khach hang da cap nhat  print(customer) |

**Giải thích:**

**Dữ liệu khách hàng và đơn hàng:**

* customer: Đối tượng chứa thông tin khách hàng như ID, tên, địa chỉ, thành phố, tiểu bang, mã zip, số điện thoại và ngày tạo.
* orders: Danh sách chứa các đối tượng đơn hàng. Mỗi đơn hàng bao gồm thông tin về ID đơn hàng, ID khách hàng, ID nhân viên bán hàng, các mặt hàng trong đơn hàng, tổng giá trị và ngày đặt hàng.
* order\_summary: Đối tượng chứa tổng hợp các đơn hàng của một khách hàng, bao gồm ID khách hàng, danh sách các bản tóm tắt đơn hàng (ID đơn hàng, tổng giá trị, ngày đặt hàng).

**Khởi tạo khách hàng:**

* client = riak.RiakClient(pb\_port=8087, protocol='pbc'): Khởi tạo một kết nối tới cơ sở dữ liệu Riak, sử dụng cổng 8087 và giao thức pbc.
* Tạo các bucket:
* customer\_bucket = client.bucket('Customers'): Tạo bucket để lưu trữ thông tin khách hàng.
* order\_bucket = client.bucket('Orders'): Tạo bucket để lưu trữ thông tin đơn hàng.
* order\_summary\_bucket = client.bucket('OrderSummaries'): Tạo bucket để lưu trữ thông tin tóm tắt đơn hàng.

**Lưu trữ dữ liệu:**

* cr = customer\_bucket.new(str(customer['customer\_id']), data=customer): Tạo một đối tượng mới trong bucket Customers với khóa là ID khách hàng và dữ liệu là đối tượng customer.
* cr.store(): Lưu đối tượng khách hàng vào bucket.
* for order in orders: Lặp qua từng đơn hàng trong danh sách orders.
* order\_riak = order\_bucket.new(str(order['order\_id']), data=order): Tạo một đối tượng mới trong bucket Orders với khóa là ID đơn hàng và dữ liệu là đối tượng order.
* order\_riak.store(): Lưu từng đối tượng đơn hàng vào bucket.
* os = order\_summary\_bucket.new(str(order\_summary['customer\_id']), data=order\_summary): Tạo một đối tượng mới trong bucket OrderSummaries với khóa là ID khách hàng và dữ liệu là đối tượng order\_summary.
* os.store(): Lưu đối tượng tóm tắt đơn hàng vào bucket.

**Lấy dữ liệu khách hàng và thêm tóm tắt đơn hàng:**

* customer = customer\_bucket.get('1').data: Lấy dữ liệu của khách hàng với ID là '1' từ bucket Customers. Dữ liệu này được lưu vào biến customer.
* customer['order\_summary'] = order\_summary\_bucket.get('1').data: Lấy dữ liệu tóm tắt đơn hàng với ID là '1' từ bucket OrderSummaries và thêm dữ liệu này vào đối tượng customer với khóa order\_summary.
* In ra dữ liệu khách hàng đã cập nhật:
* print(customer): In ra đối tượng customer đã được cập nhật với thông tin tóm tắt đơn hàng.

Trong cơ sở dữ liệu Riak, các bucket hoạt động như các không gian tên (namespace) riêng biệt. Điều này có nghĩa là có thể sử dụng cùng một khóa (key) trong nhiều bucket khác nhau mà không gây ra xung đột hay ghi đè dữ liệu.

* Khóa '1' được sử dụng để lưu trữ thông tin khách hàng trong bucket Customers.
* Cùng khóa '1' cũng được sử dụng để lưu trữ thông tin tóm tắt đơn hàng trong bucket OrderSummaries.

Do các bucket là các không gian tên riêng biệt, việc sử dụng cùng một khóa '1' trong hai bucket này không gây ra bất kỳ xung đột nào. Điều này cho phép tổ chức dữ liệu một cách logic và có cấu trúc hơn.

Với cách này, có thể dễ dàng quản lý và truy cập dữ liệu liên quan đến cùng một khách hàng hoặc cùng một đơn hàng từ các bucket khác nhau mà không cần phải lo lắng về việc ghi đè hoặc nhầm lẫn dữ liệu.

**Kết quả:**

**A screen shot of a computer

Description automatically generated**

1. **Secondary Indexes**

|  |
| --- |
| **Index4.py** |
| from riak import RiakClient, RiakObject  client= RiakClient()  bucket = client.bucket\_type('default').bucket('users')  bucket = client.bucket('users')  obj = RiakObject(client, bucket, 'john\_smith')  obj.content\_type = 'text/plain'  obj.data = '...user data...'  obj.add\_index('twitter\_bin', 'jsmith123')  obj.add\_index('email\_bin', 'jsmith@riak.com')  obj.store() |
| **Index5.py** |
| from riak import RiakClient, RiakObject  client= RiakClient()  bucket = client.bucket('users') # equivalent to client.bucket\_type('default').bucket('users')  a=bucket.get\_index('twitter\_bin', 'jsmith123').results  print(a) |

**Giải thích:**

* **Index4.py:**
* from riak import RiakClient, RiakObject: Dòng này import hai lớp cần thiết từ thư viện Riak, đó là RiakClient (để tạo kết nối tới Riak cluster) và RiakObject (để tạo, lưu trữ và truy xuất các đối tượng trong Riak).
* client = RiakClient(): Tạo một đối tượng RiakClient để kết nối tới cơ sở dữ liệu Riak. Mặc định, đối tượng này sẽ kết nối tới một cụm Riak chạy trên cùng một máy với mã Python.
* bucket = client.bucket\_type('default').bucket('users'): Tạo một đối tượng bucket (ngăn chứa) có tên là 'users' trong loại bucket mặc định ('default'). Riak sử dụng các bucket để tổ chức và lưu trữ dữ liệu.
* bucket = client.bucket('users'): Một lần nữa, gán đối tượng bucket 'users' cho biến bucket. Dòng này trùng lặp với dòng trước và không cần thiết.
* obj = RiakObject(client, bucket, 'john\_smith'): Tạo một đối tượng RiakObject có tên là 'john\_smith', thuộc bucket đã được tạo ở trên. Đối tượng này sẽ đại diện cho dữ liệu của người dùng có tên là John Smith.
* obj.content\_type = 'text/plain': Thiết lập kiểu nội dung của đối tượng là 'text/plain'. Điều này cho biết rằng nội dung của đối tượng là văn bản thuần túy.
* obj.data = '...user data...': Gán dữ liệu người dùng vào đối tượng RiakObject. Trong ví dụ này, dữ liệu của người dùng được đại diện bằng chuỗi '...user data...'.
* obj.add\_index('twitter\_bin', 'jsmith123'): Thêm một chỉ mục (index) có tên 'twitter\_bin' với giá trị 'jsmith123'. Điều này cho phép bạn truy vấn và lọc dữ liệu dựa trên chỉ mục này.
* obj.add\_index('email\_bin', 'jsmith@riak.com'): Tương tự như trên, dòng này thêm một chỉ mục có tên 'email\_bin' với giá trị 'jsmith@riak.com'.
* obj.store(): Lưu trữ đối tượng RiakObject vào cơ sở dữ liệu Riak. Sau dòng này, đối tượng 'john\_smith' với thông tin người dùng, chỉ mục và dữ liệu của nó sẽ được lưu trữ trong bucket 'users'.
* **Index5.py:**
* from riak import RiakClient, RiakObject: Dòng này import hai lớp cần thiết từ thư viện Riak, RiakClient và RiakObject, giống như trong ví dụ trước.
* - client = RiakClient(): Tạo một đối tượng RiakClient để kết nối tới cơ sở dữ liệu Riak, tương tự như trong ví dụ trước.
* - bucket = client.bucket('users'): Tạo một đối tượng bucket có tên là 'users'. Đối tượng này tương tự như trong ví dụ trước, nhưng ở đây không sử dụng bucket\_type, mặc định Riak sẽ sử dụng loại bucket là 'default'.
* - a = bucket.get\_index('twitter\_bin', 'jsmith123').results: Thực hiện truy vấn trên chỉ mục 'twitter\_bin' với giá trị là 'jsmith123'. Hàm get\_index trả về tất cả các đối tượng trong bucket mà có chỉ mục 'twitter\_bin' bằng 'jsmith123'. Kết quả của truy vấn được lưu trong biến a.
* - print(a): In ra kết quả của truy vấn. Biến a chứa danh sách các đối tượng mà có chỉ mục 'twitter\_bin' là 'jsmith123'.

Secondary indexes (2i) trong Riak cho phép gắn nhãn cho các đối tượng được lưu trữ trong Riak, tại thời điểm ghi, với một hoặc nhiều giá trị có thể truy vấn được. Các giá trị đó sau đó có thể được sử dụng để tìm nhiều đối tượng trong Riak. Nếu lưu trữ dữ liệu người dùng, ví dụ, có thể gắn thẻ cho mỗi đối tượng liên quan đến người dùng đó với tên người dùng hoặc đánh dấu duy nhất khác. Một khi đã gắn thẻ, có thể tìm thấy tất cả các đối tượng trong một bucket Riak chia sẻ cùng một thẻ đó. Chỉ mục phụ có thể là một chuỗi hoặc nhị phân, chẳng hạn như sensor\_1\_data hoặc admin\_user hoặc click\_event, hoặc một số nguyên, chẳng hạn như 99 hoặc 141121.

**Kết quả:**

A screen shot of a computer

Description automatically generated

#### **Object modeling (mô hình hóa đối tượng)**

|  |
| --- |
| **Object\_modeling.py** |
| from datetime import datetime  import string  import riak  marleen = {'user\_name': 'marleenmgr',  'full\_name': 'Marleen Manager',  'email': 'marleen.manager@basho.com'}  joe = {'user\_name': 'joeuser',  'full\_name': 'Joe User',  'email': 'joe.user@basho.com'}  msg = {'sender': marleen['user\_name'],  'recipient': joe['user\_name'],  'created': datetime.utcnow().isoformat(),  'text': 'Welcome to the company!'}  class UserRepository:  BUCKET = 'Users'  def \_\_init\_\_(self, client):  self.client = client  def save(self, user):  riak\_obj = self.client.bucket(self.BUCKET).get(user['user\_name'])  riak\_obj.data = user  return riak\_obj.store()  def get(self, user\_name):  riak\_obj = self.client.bucket(self.BUCKET).get(user\_name)  return riak\_obj.data  class MsgRepository:  BUCKET = 'Msgs'  def \_\_init\_\_(self, client):  self.client = client  def save(self, msg):  msgs = self.client.bucket(self.BUCKET)  key = self.\_generate\_key(msg)  riak\_obj = msgs.get(key)  if not riak\_obj.exists:  riak\_obj.data = msg  riak\_obj.store(if\_none\_match=True)  return riak\_obj  def get(self, key):  riak\_obj = self.client.bucket(self.BUCKET).get(key)  return riak\_obj.data  def \_generate\_key(self, msg):  return msg['sender'] + '\_' + msg['created']  class TimelineRepository:  BUCKET = 'Timelines'  SENT = 'Sent'  INBOX = 'Inbox'  def \_\_init\_\_(self, client):  self.client = client  self.msg\_repo = MsgRepository(client)  def post\_message(self, msg):  # Save the canonical copy  saved\_message = self.msg\_repo.save(msg)  msg\_key = saved\_message.key  # Post to sender's Sent timeline  self.\_add\_to\_timeline(msg, self.SENT, msg\_key)  # Post to recipient's Inbox timeline  self.\_add\_to\_timeline(msg, self.INBOX, msg\_key)  def get\_timeline(self, owner, msg\_type, date):  key = self.\_generate\_key(owner, msg\_type, date)  riak\_obj = self.client.bucket(self.BUCKET).get(key)  return riak\_obj.data  def \_add\_to\_timeline(self, msg, msg\_type, msg\_key):  timeline\_key = self.\_generate\_key\_from\_msg(msg, msg\_type)  riak\_obj = self.client.bucket(self.BUCKET).get(timeline\_key)  if riak\_obj.exists:  riak\_obj = self.\_add\_to\_existing\_timeline(riak\_obj,  msg\_key)  else:  riak\_obj = self.\_create\_new\_timeline(riak\_obj,  msg, msg\_type,  msg\_key)  return riak\_obj.store()  def \_create\_new\_timeline(self, riak\_obj, msg, msg\_type, msg\_key):  owner = self.\_get\_owner(msg, msg\_type)  new\_timeline = {'owner': owner,  'msg\_type': msg\_type,  'msgs': [msg\_key]}  riak\_obj.data = new\_timeline  return riak\_obj  def \_add\_to\_existing\_timeline(self, riak\_obj, msg\_key):  riak\_obj.data['msgs'].append(msg\_key)  return riak\_obj  def \_get\_owner(self, msg, msg\_type):  if msg\_type == self.INBOX:  return msg['recipient']  else:  return msg['sender']  def \_generate\_key\_from\_msg(self, msg, msg\_type):  owner = self.\_get\_owner(msg, msg\_type)  return self.\_generate\_key(owner, msg\_type, msg['created'])  def \_generate\_key(self, owner, msg\_type, datetimestr):  dateString = datetimestr.split('T',1)[0]  return owner + '\_' + msg\_type + '\_' + dateString  # Setup our repositories  client = riak.RiakClient(pb\_port=8087, protocol='pbc')  userRepo = UserRepository(client)  msgsRepo = MsgRepository(client)  timelineRepo = TimelineRepository(client)  # Save users  userRepo.save(marleen)  userRepo.save(joe)  # Post msg to timelines  timelineRepo.post\_message(msg)  # Get Joe's inbox for today, get first message  joes\_inbox\_today = timelineRepo.get\_timeline(  joe['user\_name'],  TimelineRepository.INBOX,  datetime.utcnow().isoformat())  joes\_first\_message = msgsRepo.get(joes\_inbox\_today['msgs'][0])  print('From: {0}\nMsg : {1}\n\n'.format(  joes\_first\_message['sender'],  joes\_first\_message['text'])) |

**Giải thích:**

**Tạo dữ liệu mẫu:**

* marleen và joe: Thông tin của hai người dùng.
* msg: Thông điệp từ Marleen gửi cho Joe.

**Định nghĩa các lớp để làm việc với dữ liệu trong Riak:**

* **UserRepository: Lớp này dùng để lưu và lấy thông tin người dùng từ bucket 'Users'.**
* \_\_init\_\_(self, client): Khởi tạo với đối tượng client.
* save(self, user): Lưu thông tin người dùng vào bucket.
* get(self, user\_name): Lấy thông tin người dùng từ bucket.
* **MsgRepository: Lớp này dùng để lưu và lấy thông điệp từ bucket 'Msgs'.**
* \_\_init\_\_(self, client): Khởi tạo với đối tượng client.
* save(self, msg): Lưu thông điệp vào bucket, tạo khóa duy nhất cho mỗi thông điệp.
* get(self, key): Lấy thông điệp từ bucket.
* \_generate\_key(self, msg): Tạo khóa duy nhất cho thông điệp từ người gửi và thời gian tạo.
* **TimelineRepository: Lớp này dùng để quản lý dòng thời gian của người dùng (inbox và sent).**
* \_\_init\_\_(self, client): Khởi tạo với đối tượng client và đối tượng MsgRepository.
* post\_message(self, msg): Đăng thông điệp vào dòng thời gian của người gửi và người nhận.
* get\_timeline(self, owner, msg\_type, date): Lấy dòng thời gian của người dùng theo loại (sent/inbox) và ngày.
* \_add\_to\_timeline(self, msg, msg\_type, msg\_key): Thêm thông điệp vào dòng thời gian.
* \_create\_new\_timeline(self, riak\_obj, msg, msg\_type, msg\_key): Tạo dòng thời gian mới.
* \_add\_to\_existing\_timeline(self, riak\_obj, msg\_key): Thêm thông điệp vào dòng thời gian đã tồn tại.
* \_get\_owner(self, msg, msg\_type): Lấy chủ sở hữu của dòng thời gian (người gửi hoặc người nhận).
* \_generate\_key\_from\_msg(self, msg, msg\_type): Tạo khóa cho dòng thời gian từ thông điệp.
* \_generate\_key(self, owner, msg\_type, datetimestr): Tạo khóa cho dòng thời gian từ chủ sở hữu, loại và ngày.
* **Thiết lập các repository và lưu dữ liệu:**
* Tạo kết nối đến Riak: client = riak.RiakClient(pb\_port=8087, protocol='pbc')
* Tạo các đối tượng repository: userRepo, msgsRepo, timelineRepo.
* Lưu thông tin người dùng: userRepo.save(marleen), userRepo.save(joe).
* Đăng thông điệp vào dòng thời gian: timelineRepo.post\_message(msg).
* **Lấy và in ra thông điệp đầu tiên trong inbox của Joe:**
* Lấy dòng thời gian inbox của Joe cho hôm nay: joes\_inbox\_today.
* Lấy thông điệp đầu tiên trong dòng thời gian: joes\_first\_message.
* In ra thông tin của thông điệp: print('From: {0}\nMsg : {1}\n\n'.format(joes\_first\_message['sender'], joes\_first\_message['text'])).

Tóm lại, đoạn mã này tạo các đối tượng người dùng, lưu thông tin vào cơ sở dữ liệu Riak, đăng thông điệp vào dòng thời gian của người gửi và người nhận, và sau đó truy xuất và in ra thông điệp từ dòng thời gian của người nhận.

**Kết quả:**

**A black screen with white text

Description automatically generated**

## **CẤU HÌNH HADOOP CLUSTER TRÊN WINDOW**

### **Cài đặt JDK bản 1.8 (bắt buộc):**

Hadoop sử dụng JDK 1.8

Ta vào link sau: <https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>

A screenshot of a computer

Description automatically generated

**Tải và cài đặt vào: "C:\install\jdk1.8"**

### **Thiết lập biến môi trường cho Java JDK**

Cần cấu hình biến môi trường JAVA\_HOME cho Java JDK

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

### **Tải Hadoop và giải nén vào ổ C**

**A close up of numbers

Description automatically generated**

Đổi tên thành hadoop-Master để phân biệt(không cần làm theo)

### **Thiết lập biến môi trường cho Hadoop**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**A computer screen with white text

Description automatically generated**

### **Cấu hình các tập tin cho Hadoop**

#### **Máy master**

- Cấu hình **core-site.xml**như dưới đây:

A screenshot of a computer program

Description automatically generated

- Cấu hình **mapred-site.xml**như dưới đây:  
A screenshot of a computer program

Description automatically generated

- Cấu hình **mapred-site.xml**như dưới đây:

* Tạo thư mục “data” trong: “C:\hadoop-Master”
* Tạo thư mục con “datanode” trong: “C:\hadoop-Master\data\datanode”
* Tạo thư mục con “namenode” trong: ”C:\hadoop-Master\data\namenode”

**A screenshot of a computer

Description automatically generated**

* Sau đó cấu hình **hdfs-site.xml**như sau:

A screenshot of a computer code

Description automatically generated

* Cấu hình **yarn-site.xml**như dưới đây:

A computer screen shot of a computer code

Description automatically generated

* Cấu hình **hadoop-env.cmd**:

**A black and white text

Description automatically generated**

Cập nhật các Hadoop Configurations

Vì file bin của hadoop khi tải về chưa được cấu hình nên ta phải tải

* **Tải**[**https://github.com/s911415/apache-hadoop-3.1.0-winutils**](https://github.com/s911415/apache-hadoop-3.1.0-winutils)
* Tải về giải nén ra thấy thư mục bin ở bên trong
* Chép đè thư mục bin này trong thư mục bin của C:\hadoop-3.3.6\bin
* Sau đó format lại **namenode và datanode**:
* **hdfs namenode –format**

**A screen shot of a computer

Description automatically generated**

* **hdfs datanode -format**

**A screen shot of a computer

Description automatically generated**

* Sau khi format ta có kết quả 1 thư mục hadoop-3.3.6

A screenshot of a computer

Description automatically generated

* **Tiếp theo sao chép file:**

*"C:hadoop-Master\share\hadoop\yarn\timelineservice\hadoop-yarn-server-timelineservice-3.3.6.jar"*

**vào** *"C:hadoop-Master\share\hadoop\yarn\hadoop-yarn-server-timelineservice-3.3.6.jar*"

Cấu hình file workers (C:\hadoop-Master\etc\hadoop\workers)

Đây là tên các máy thực hiện

A screenshot of a computer

Description automatically generated

Ta sẽ câu hình file hosts để thêm các datanode khác vào cụm

Mở notepad ở chế độ admin  
Mở file hosts (C:\Windows\System32\drivers\etc\hosts)

A screenshot of a computer program

Description automatically generated

Để lấy được các ip trên ta thực hiện các bước sau

* Kết nối với mạng cá nhân
* Mở setting của Network & internet
* Mở Mobile hostpot để biến máy của mình thành máy local

A screenshot of a computer

Description automatically generated

* Tiếp theo mở cmd và gõ ipconfig

A screen shot of a computer

Description automatically generated

Đây là ip của namenode(Ta sẽ sưử dụng ip ở Ipv4 Addreses của Wireless LAN adapter Local Area Connection\* 1:)

Sau khi thực hiện các cấu hình sau thì ta sẽ start các node để kiểm tra

Mở cmd ở chế độ admin và gõ lệnh: **Start-all.cmd**

A screenshot of a computer screen

Description automatically generated

* Slave: http://localhost:9864

A screenshot of a computer

Description automatically generated

* Master: http://localhost:9870

A screenshot of a computer

Description automatically generated

* Resource Manager: http://localhost:8088

A screenshot of a computer

Description automatically generated

Đây là kết quả sau khi chạy 1 mình máy master

#### **Cấu hình máy slave**

Cấu hình ban đầu và set biến môi trường tương tự máy master

Trong thư mục **C:/Hadoop-3.3.6/etc/hadoop** lần lượt chỉnh sửa các file:

* core-site.xml
* mapred-site.xml
* hdfs-site.xml
* yarn-site.xml
* hadoop-env.cmd

**Cấu hình core-site.xml như dưới đây:**

|  |
| --- |
| **<configuration>**  **<property>**  **<name>fs.defaultFS</name>**  **<value>hdfs://** **hadoopMaster:9000</value>**  **</property>**  **</configuration>** |

**A screenshot of a computer

Description automatically generated**

**Cấu hình mapred-site.xml như dưới đây:**

|  |
| --- |
| **<configuration>**  **<property>**  **<name>mapreduce.framework.name</name>**  **<value>yarn</value>**  **</property>**  **</configuration>** |

**A screenshot of a computer code

Description automatically generated**

**Cấu hình hdfs-site.xml như dưới đây:**

Tạo thư mục “data” trong “C:/Hadoop-3.3.6”

Tạo thư mục con “datanode” trong “C:/Hadoop-3.3.6/data”

Tạo thư mục con “namenode” trong “C:/Hadoop-3.3.6/data”

**A screenshot of a computer

Description automatically generated**

**Sau đó cấu hình** **hdfs-site.xml như sau**:

|  |
| --- |
| **<configuration>**  **<property>**  **<name>dfs.replication</name>**  **<value>4</value>**  **</property>**  **<property>**  **<name>dfs.namenode.name.dir</name>**  **<value>/hadoop-3.3.6/data/namenode</value>**  **</property>**  **<property>**  **<name>dfs.datanode.data.dir</name>**  **<value>/hadoop-3.3.6/data/datanode</value>**  **</property>**  **</configuration>** |

**A screenshot of a computer code

Description automatically generated**

**Cấu hình yarn-site.xml như dưới đây:**

|  |
| --- |
| **<configuration>**  **<property>**  **<name>yarn.nodemanager.aux-services</name>**  **<value>mapreduce\_shuffle</value>**  **</property>**  **<property>**  **<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>**  **<value>org.apache.hadoop.mapred.ShuffleHandler</value>**  **</property>**  **</configuration>** |

**A computer screen shot of a computer code

Description automatically generated**

**Cấu hình** **hadoop-env.cmd**:

* set JAVA\_HOME=%JAVA\_HOME%

sửa %JAVA\_HOME% thành đường dẫn cài JDK trong ổ C:

* set JAVA\_HOME= C:\PROGRA~1\Java\jdk1.8.0\_201

**A screenshot of a computer program

Description automatically generated**

**Cập nhật các Hadoop Configurations**

Tải <https://github.com/s911415/apache-hadoop-3.1.0-winutils>

Tải về giải nén ra thấy thư mục **bin** ở bên trong

Chép đè thư mục bin này trong thư mục bin của C:\hadoop-3.3.6\bin

Sau đó format lại **namenode và datanode**: mở command line lên, gõ lệnh sau:

* **hdfs namenode –format**

A screenshot of a computer program

Description automatically generated

* **hdfs datanode -format**

A screen shot of a computer

Description automatically generated

**\*Tiếp theo sao chép file:**

“C:/hadoop-3.3.5/share/hadoop/yarn/timelineservice/ hadoop-yarn-server-timelineservice-3.3.5.jar”

vào “C:/hadoop-3.3.5/share/hadoop/yarn/hadoop-yarn-server-timelineservice-3.3.5.jar”

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Cấu hình file hosts**

Địa chỉ file hosts

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

192.168.137.1 hadoopMaster: là Default Gateway của máy master

192.168.137.43 TanHuy-slave : là IPv4 Address của máy slave

Để chạy cần kết nối chung mạng và dùng ipconfig để xem

#### **Kết quả sau khi start cả máy master và máy slave**

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

## **KẾT LUẬN**

### **Các kết quả đạt được**

* Tìm hiểu được cách tạo và hiểu hơn về hadoop trên windows
* Triển khai thành công một cluster Hadoop trên hệ điều hành Windows, kết nối được các máy tính khác nhau.
* Tìm hiểu được Riak
* Thực hiện các thao tác CRUD trên cơ sở dữ liệu Riak trên Ubuntu, từ việc tạo, đọc, cập nhật đến xóa dữ liệu.

### **Ưu điểm và hạn chế của đề tài**

**Ưu điểm:**

* Đề tài mở ra cơ hội nghiên cứu và thực hành về việc triển khai cluster Hadoop trên môi trường Windows, một lĩnh vực đang phát triển và cần thiết cho nhiều tổ chức sử dụng Windows trong môi trường công nghiệp.
* Phân tích và thực hiện các thao tác CRUD trên cơ sở dữ liệu Riak, giúp hiểu rõ hơn về tính linh hoạt và khả năng mở rộng của Riak trong môi trường phân tán.

**Hạn chế:**

* Phạm vi đề tài có thể hạn chế đối với những nghiên cứu sâu hơn về hiệu suất và tối ưu hóa của Hadoop trên Windows và Riak trên Ubuntu.
* Giới hạn về cơ sở kiến thức và kinh nghiệm của người thực hiện có thể ảnh hưởng đến kết quả nghiên cứu và đánh giá.
* Riak chỉ dừng lại ở Single node, chưa tìm hiểu sâu hơn

### **Định hướng phát triển**

* Tìm hiểu nhiều hơn về cơ sở dữ liệu phi quan hệ và ứng dụng được vào các đề tài thực tế
* Tìm hiểu sâu hơn về Riak và ứng dụng của nó trong các trường hợp thực tế, ứng dụng vào các đề tài lớn nhỏ khác nhau

# **TÀI LIỆU THAM KHẢO**

1. Riak KV documentation. (N.d.). Retrieved from <https://docs.riak.com/riak/kv/latest/index.html>
2. NoSQL là gì?. (N.d.). Retrieved from https://aws.amazon.com/vi/nosql/
3. NoSQL. (n.d.). Retrieved from <https://vi.wikipedia.org/wiki/NoSQL#:~:text=C%C6%A1%20s%E1%BB%9F%20d%E1%BB%AF%20li%E1%BB%87u%20NoSQL,d%E1%BB%AF%20li%E1%BB%87u%20ki%E1%BB%83u%20quan%20h%E1%BB%87>.
4. HAI NGUYEN (2022) NOSQL là gì? Một số ưu điểm và nhược điểm cần biết về nosql. Retrieved from <https://chiasekinang.com/nosql-la-gi-mot-so-uu-diem-va-nhuoc-diem-can-biet-ve-nosql/>
5. Linh, N. T. N. (2024). Những điểm khác biệt giữa SQL và NoSQL. Retrieved from <https://viblo.asia/p/nhung-diem-khac-biet-giua-sql-va-nosql-gGJ59b4rKX2>
6. To, T. D. (2024). NOSQL DATABASE – RIAK. Retrieved from https://viblo.asia/p/nosql-database-riak-oaKYMNz4M83E