**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**

🙤🙧🟍🙥🙦

A picture containing text

Description automatically generated

**ĐIỆN TOÁN ĐÁM MÂY – IS402.M21**

**BÁO CÁO SEMINAR**

**TÌM HIỂU DISTRIBUTED DATABASE - CASSANDRA**

**GIẢNG VIÊN HƯỚNG DẪN**

**HÀ LÊ HOÀI TRUNG**

**NHÓM**

ĐÀO THỊ NGÂN TRIỀU – 18521535

NGUYỄN ANH TÚ – 18521582

NGUYỄN THỊ KIM YẾN – 18521689

**TP.HCM, THÁNG 5 NĂM 2022**

MỤC LỤC

[1. Tổng quan 4](#_Toc104023591)

[1.1 Giới thiệu 4](#_Toc104023592)

[1.2 Các thành phần trong Cassandra 4](#_Toc104023593)

[1.3 Tính năng 4](#_Toc104023594)

[2. Kiến trúc và mô hình lưu trữ Apache Cassandra 5](#_Toc104023595)

[2.1 Kiến trúc 5](#_Toc104023596)

[2.1.1 Giao thức Gossip trong Cassandra 5](#_Toc104023597)

[2.1.2 Trạng thái dò thất bại và sự phục hồi 7](#_Toc104023598)

[2.1.3 Trung tâm dữ liệu (Data Centers) và Bánh răng (Racks) 8](#_Toc104023599)

[2.1.4 Rings và Tokens: 9](#_Toc104023600)

[2.1.5 Phân tán dữ liệu trong Cassandra 9](#_Toc104023601)

[2.1.6 Snitch trong Cassandra 12](#_Toc104023602)

[2.1.7 Virtual node - Node ảo trong Cassandra 12](#_Toc104023603)

[2.2 Mô hình lưu trữ dữ liệu 14](#_Toc104023604)

[2.2.1 Mô hình lưu trữ 14](#_Toc104023605)

[2.2.2 Phân vùng dữ liệu - Partitioners trong Cassandra. 19](#_Toc104023606)

[2.2.3 Cơ chế nhân bản trong Cassandra (Replication Strategie) 21](#_Toc104023607)

[2.2.4 Tính nhất quán trong Cassandra (consistency) 22](#_Toc104023608)

[2.2.4.1 Nhất quán ghi 23](#_Toc104023609)

[2.2.4.2 Mức độ nhất quán - Consistency Levels 23](#_Toc104023610)

[2.2.4.3 Đọc dữ liệu (read repair) 23](#_Toc104023611)

[2.2.4.4 Anti-Entropy Node Repair 23](#_Toc104023612)

[2.2.5 Node điều phối (coordinator node) 24](#_Toc104023613)

[2.2.6 Hinted Handoff 25](#_Toc104023614)

[2.2.7 CQL: 26](#_Toc104023615)

[2.2.7.1 Tạo Keyspace: 26](#_Toc104023616)

[2.2.7.2 Use keyspace: 28](#_Toc104023617)

[2.2.7.3 ALTER Keyspace: 28](#_Toc104023618)

[2.2.7.4 DROP Keyspace: 28](#_Toc104023619)

[2.2.7.5 Create Table: 28](#_Toc104023620)

[2.2.7.6 ALTER Table: 30](#_Toc104023621)

[2.2.7.7 DROP Table: 30](#_Toc104023622)

[2.2.7.8 Insert, Update, Delete và Where Clause 31](#_Toc104023623)

[2.3 Các hạn chế của CQL 32](#_Toc104023624)

[3. Cài đặt 33](#_Toc104023625)

[3.1 Cài đặt và thiết lập cấu hình cơ bản Cassandra trên Ubuntu 20.04 33](#_Toc104023626)

[3.2 Cài đặt Cassandra 3.11.11 trên Window 10 37](#_Toc104023627)

[4. Ưu điểm và nhược điểm của Apache Cassandra 39](#_Toc104023628)

[4.1 Ưu điểm 39](#_Toc104023629)

[4.2 Nhược điểm 40](#_Toc104023630)

[5. So sánh Apache Cassandra với các hệ quản trị cơ sở dữ liệu khác 40](#_Toc104023631)

[5.1 So sánh với SQL Server 40](#_Toc104023632)

[5.2 So sánh với MongoDB 41](#_Toc104023633)

[Bảng phân công công việc 43](#_Toc104023634)

[Tài liệu tham khảo 43](#_Toc104023635)

# Tổng quan

## Giới thiệu

* Cassandra là hệ quản trị cơ sở dữ liệu phân tán, NoSQL, mã nguồn mở, dạng key-value. Được thiết kế ban đầu tại Facebook để phát triển công nghệ kết hợp của BigTable từ Google và Dynamo của Amazon. Sử dụng kỹ thuật phân tán, nhân bản từ Dynamo và mô hình lưu trữ dữ liệu từ BigTable, Cassandra được thiết kế để đáp ứng các yêu cầu mới nổi đối với hệ thống lưu trữ có thể mở rộng, đáng tin cậy và có tính khả dụng cao.

## Các thành phần trong Cassandra

* + Nút (node): Một máy tính hoặc một máy ảo khởi chạy một Cassandra instance. Các nút trong cụm có vai trò như nhau, không có nút nào làm nút chính.
  + Rack: Nhóm một hoặc nhiều nút.
  + Datacenter: Nhóm một hoặc nhiều Rack.
  + Cụm: Bao gồm một hoặc nhiều Datacenter trong đó gồm một hay nhiều nút kết nối với nhau, dữ liệu của toàn hệ thống sẽ được tự động phân chia đến các nút trong cụm.

## Tính năng

Định lý CAP nói rằng mọi cơ sở dữ liệu phân tán chỉ có thể thỏa mãn 2 trong 3 tính chất dưới đây:

* + Tính nhất quán (Consistency), mỗi thao tác đọc đều nhận được đa phần các thao tác ghi gần nhất hoặc lỗi.
  + Tính khả dụng (Availability), mỗi request đều nhận được response.
  + Tính Phân vùng (Partition). khả năng chịu đựng của hệ thống lưu trữ đối với sự cố phân vùng mạng. Ngay cả khi một số tin nhắn bị bỏ hoặc bị trì hoãn, hệ thống vẫn tiếp tục hoạt động.
  + Tính sẵn sàng cao là ưu tiên trong các ứng dụng dựa trên web và vì mục tiêu này, Cassandra đánh đổi Tính khả dụng và Phân vùng từ đảm bảo CAP, và cung cấp một Tính nhất quán cuối cùng (Eventual Consistency)

Diagram

Description automatically generated

Định lý CAP

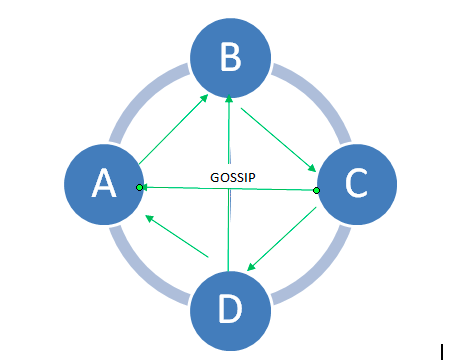
# Kiến trúc và mô hình lưu trữ Apache Cassandra

## Kiến trúc

* Thiết kế của Cassandra là thiết kế phân tán dựa trên kiến trúc mạng ngang hàng (Peer - to - Peer) tất cả các node máy chủ trong hệ thống đều có vai trò như nhau và không có node máy chủ nào đóng vai trò là máy chủ trung tâm (master), giảm thiểu sự cố của máy chủ này có thể kéo theo đánh sập hoàn toàn hệ thống như các kiến trúc master-slave truyền thống.
* Các node máy chủ của Cassandra là độc lập và tham gia vào kết nối với các node máy chủ khác trong hệ thống. Mỗi node đều có thể xử lý các thao tác ghi và đọc dữ liệu, không phân biệt là dữ liệu được lưu trữ một cách vật lý trên máy chủ nào trong hệ thống.
* Khi một node trong hệ thống bị sự cố và dừng hoạt động, các thao tác đọc ghi dữ liệu có thể được xử lý bởi các node khác trong hệ thống. Quá trình này hoàn toàn trong suốt với ứng dụng cho phép ẩn đi sự cố của máy chủ hệ thống đối với các ứng dụng.
* Trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Cơ chế này được thực thi trong quá trình đọc dữ liệu (read repair) thay vì đồng bộ ngay trong thao tác ghi dữ liệu, điều này cho phép tăng hiệu năng cho thao tác ghi dữ liệu.

### Giao thức Gossip trong Cassandra

* Mỗi khi cụm Cassandra bổ sung hoặc loại bỏ một node ra khỏi cụm, dữ liệu trong cụm sẽ phải được phân bố lại. Khi bổ sung một node, node đó sẽ lấy đi 1 phần dữ liệu của các node, khi một node bị loại khỏi cụm, dữ liệu của node đó sẽ phải được lưu trữ đều trên các node khác.
* Trong Cassandra, các node giao tiếp với nhau thông qua giao thức Gossip. Gossip là một giao thức dùng để cập nhật thông tin về trạng thái của các node khác đang tham gia vào cluster. Đây là một giao thức liên lạc ngang hàng dạng peer-to-peer trong đó mỗi node trao đổi định kỳ thông tin trạng thái của chúng với các node khác mà chúng có liên kết.

[](https://laptrinh.vn/uploads/images/gallery/2021-06/gossip.png)

Giao thức Gossip

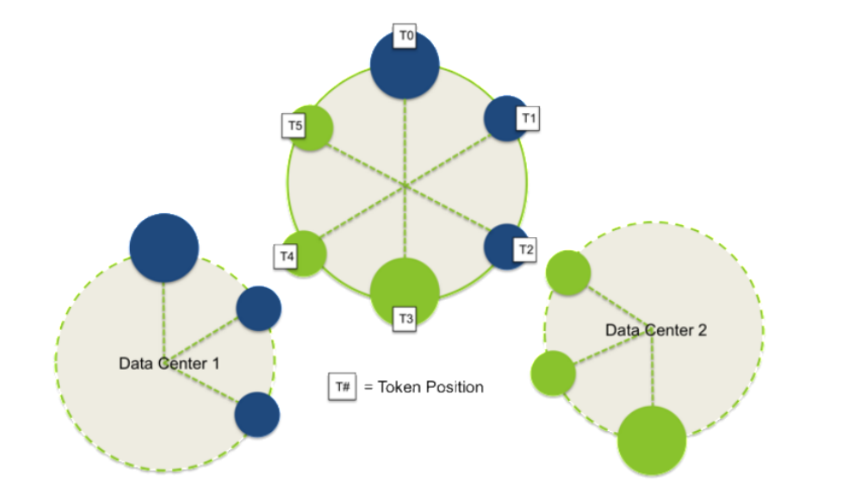
* Tiến trình gossip chạy mỗi giây và trao đổi thông tin với nhiều nhất là ba node khác trong cluster. Các node trao đổi thông tin về chính chúng và cả thông tin với các node mà chúng đã trao đổi, bằng cách này toàn bộ những node có thể nhanh chóng hiểu được trạng thái của tất cả các node còn lại trong cluster. Một gói tin gossip bao gồm cả version đi kèm với nó, như thế trong mỗi lần trao đổi gossip, các thông tin cũ sẽ bị ghi đè bởi thông tin mới nhất ở một số node.
* Khi một node được khởi động, nó sẽ xem file cấu hình cassandra.yaml để xác định tên cluster chứa nó và các node khác trong cluster được cấu hình trong file, được biết với tên là seed node. Để ngăn chặn sự gián đoạn trong truyền thông gossip, tất cả các node trong cluster phải có cùng 1 danh sách các seed node được liệt kê trong file cấu hình. Bởi vì, phần lớn các xung đột được sinh ra khi 1 node được khởi động.
* Mặc định, 1 node sẽ phải nhớ những node mà nó đã từng gossip kể cả khi khởi động lại và seed node sẽ không có mục đích nào khác ngoài việc cập nhật 1 node mới khi nó tham gia vào cluster. Tức là, khi một node tham gia vào cluster, nó sẽ liên lạc với các seed node để cập nhật trạng thái của tất cả các node khác trong cluster.
* Trong những cluster có nhiều data center, danh sách seed node nên chứa ít nhất một seed node trên mỗi data center, nếu không thì khi có 1 node mới tham gia vào cluster, thì nó sẽ liên lạc với một seed node nằm trên data center khác. Cũng không nên để mọi node đều là seed node vì nó sẽ làm giảm hiệu năng của gossip và gây khó duy trì. Việc tối ưu gossip là không quan trọng như khuyến khích, nên sử dụng một danh sách nhỏ các seed node, thông thường 3 seed node trên một data center.

### Trạng thái dò thất bại và sự phục hồi

* Dò thất bại là thông qua trạng thái của tin đồn (gossip), từ 1 nút xác định xem các nút khác trong hệ thống đang online hay offline. Thông tin dò thất bại cũng được sử dụng trong Cassandra để tránh định tuyến các yêu cầu từ máy khách đến các nút không thể truy cập được.
* Trong quá trình truyền tải các bản tin từ các nút khác cả trực tiếp (các nút giao tiếp trực tiếp đến nó) và gián tiếp (thông tin có được khi nghe qua 2 nút, 3 nút ...), Cassandra sử dụng một cơ chế tính toán một ngưỡng cho mỗi nút dựa vào điều kiện mạng, khối lượng công việc, hoặc các điều kiện khác mà có thể ảnh hưởng đến quá trình truyền tải. Trong quá trình trao đổi tin đồn, mỗi nút duy trì một cửa sổ trượt báo các thông tin tin đồn từ các nút khác trong cluster, Trong Cassandra, cấu hình phi\_convict\_threshold điều chỉnh độ nhạy cho việc dò thất bại. Các giá trị mặc định là fine cho hầu hết các tình huống, nhưng DataStax đề nghị tăng đến 12 cho Amazon EC2 do tắc nghẽn mạng thường xuyên xảy ra trên nền tảng đó.
* Node có thể thất bại do nhiều nguyên nhân khác nhau như thất bại phần cứng, mất mạng, ... Để chính thức thay đổi nút thành viên trong một cluster, các quản trị viên sử dụng tiện ích nodetool để thêm hoặc loại bỏ các nút trong một cụm Cassandra.
* Khi một node trở lại trực tuyến sau khi không hoạt động, nó có thể bỏ lỡ việc sao chép các dữ liệu mà nó duy trì. Một khi quá trình dò thất bại đánh dấu một nút là offline, nếu như hinted handoff được kích hoạt thì quá trình ghi nhớ được thực hiện bởi các bản sao khác. Tuy nhiên, nó có thể xảy ra tình huống việc ghi bỏ lỡ giữa khoảng thời gian của một nút thực sự offline cho tới khi nó bị phát hiện là offline. Hoặc nếu một nút không hoạt dộng lâu hơn max\_hint\_window\_in\_ms (mặc định là một giờ), gợi ý sẽ không còn được lưu lại. Vì lý do đó, tốt nhất là thường xuyên chạy nodetool sửa chữa tất cả các nút để đảm bảo chúng toàn vẹn dữ liệu, và chạy repair sau khi hồi phục một nút đã offline trong một thời gian dài.

### Trung tâm dữ liệu (Data Centers) và Bánh răng (Racks)

* Trong triển khai trung tâm đa dữ liệu, vị trí bản sao được tính cho mỗi trung tâm dữ liệu dựa vào chính sách NetworkTopologyStrategy. Trong mỗi trung tâm dữ liệu (hoặc nhóm nhân bản), bản sao đầu tiên cho 1 hàng cụ thể được xác định bởi giá trị thẻ bài gán cho một nút. Các bản sao trong cùng một trung tâm dữ liệu được xác định bằng việc dịch chuyển vòng theo chiều kim đồng hộ cho đến khi nó tìm được nút đầu tiên trong bánh răng (Rack) khác.
* Nếu ta không tính toán thẻ phân vùng để đảm bảo dữ liệu được phân bố đều cho mỗi trung tâm dữ liệu, bạn có thể gặp phải tình trạng phân phối dữ liệu không đồng đều trong mỗi trung tâm dữ liệu.



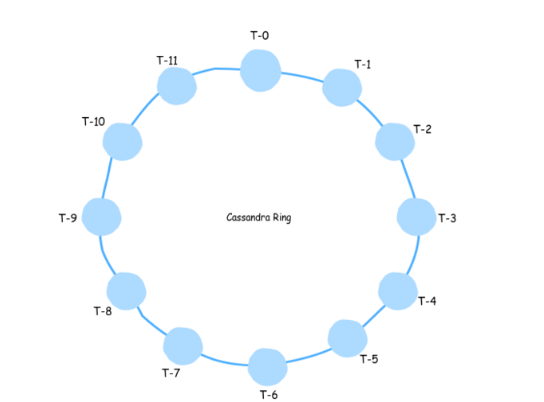
* Mục đích là để đảm bảo rằng các nút tại mỗi trung tâm dữ liệu đều được phân chia thẻ bài trên phạm vi tổng thể. Nếu không, ta có thể gặp tình trạng các nút trong mỗi trung tâm dữ liệu sở hữu một số lượng không cân xứng các khóa hàng. Mỗi trung tâm dữ liệu phải được phân chia một cách độc lập, tuy nhiên việc gán thẻ bài trong phạm vi 1 cụm không được xung đột với nhau (mỗi node phải có 1 thẻ bài duy nhất).

### Rings và Tokens:

Trong Cassandra, dữ liệu được quản lý bởi một cluster được đại diện như một không gian dữ liệu hay một ring. Vòng tròn (ring) được chia tương ứng với phạm vi là số lượng các node, mỗi node quản lý một hoặc nhiều vùng của dữ liệu. Trước khi một node có thể tham gia vòng nó được gắn một giá trị token (thẻ bài). Token xác định vị trí của node trên ring và phạm vi dữ liệu mà nó quản lý.

### Phân tán dữ liệu trong Cassandra

* Cassandra sử dụng cơ chế hàm băm nhất quán phân tán (Distributed consistent hashing) để tổ chức các node máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo hàm băm nhất quán. Mỗi vòng tròn được coi là một Datacenter.

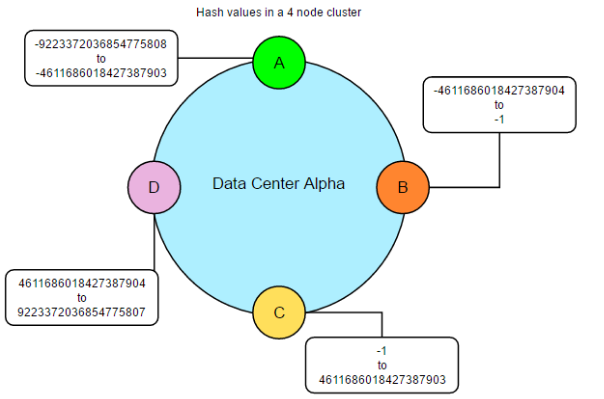
[](https://laptrinh.vn/uploads/images/gallery/2021-06/cassandra-ring.png)

*Các node trong Cassandra*

* Các node trong một cụm Cassandra sẽ được phân bố trên một vòng tròn gọi là ring (Hình trên). Mỗi node sẽ được gán với 1 giá trị key, Cassandra dùng 127bit để tạo ra key này.
* Mỗi node trong ring sẽ quản lý một phạm vi giá trị của các key. Phạm vi của key được xác định trải đều từ giá trị của chính node đó nắm giữ, đi ngược lại chiều kim đồng hồ cho đến khi gặp node đầu tiên thì dừng lại. Đối chiếu lên hình, ta sẽ thấy rằng phạm vi các key mà node T-1 quản lý nằm trong vùng (T-0; T-1]. Khi một bản ghi được ghi vào cụm Cassandra. Trường khóa của bản ghi đó sẽ được đi qua một hàm băm nhất quán, trả về một giá trị key 127bit, giá trị key này nằm trong vùng kiểm soát của node nào thì bản ghi đó sẽ được ghi vào node đấy.
* Ví dụ ta có giá trị trên các trường name được băm ra như bảng sau:

|  |  |
| --- | --- |
| **Partition Key** | **Hash value** |
| Jim | -2245452657672322382 |
| Carol | 7723358928203680754 |
| Johnny | -6756552657672322382 |
| Suzy | 1168658928203680754 |

Và ta có một cụm Cassandra với 4 node được phân bố trên ring như sau:

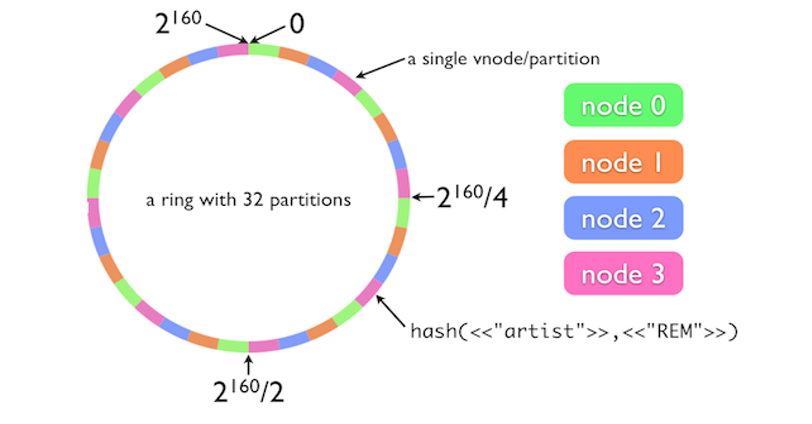
* [](https://laptrinh.vn/uploads/images/gallery/2021-06/phan-bo-du-lieu-trong-cassandra.png)Với Cassandra, chúng ta có hai cách để partition dữ liệu (xác định vị trí của từng node trong ring):
* Random partitioning: Đây là chiến lược mặc định và được đề xuất của Cassandra, vị trí của các node được xác định hoàn toàn thông qua hàm băm MD5. Phạm vi khóa nằm trong khoảng từ 0 tới 2127- 1
* Ordered partitioning: Đây là chiến lược đảm bảo các node được sắp xếp theo thứ tự và phạm vi key mà mỗi node sở hữu là như nhau.
* Với chiến lược partition thứ nhất, nếu như các giá trị băm xuất ra giúp cho việc đặt các node trong vòng phù hợp thì tất cả các bản ghi sẽ được phân bố đều trên toàn cụm. Việc thêm hay bớt mỗi node ra khỏi cụm cũng dễ dàng hơn do không phải phân bố lại vị trí các node khác.
* Với chiến lược partition thứ hai, khi mà các node được phân bố đều vả phạm vi quản lý key là như nhau, nhưng điều đó lại mang lại nhược điểm: Khó cân bằng trong cụm. Mỗi khi thêm hay bớt một node khỏi cụm, người quản trị sẽ phải tự tái cân bằng lại cụm một cách thủ công để đảm bảo các node phân bố đều. Nếu dữ liệu được ghi tuần tự, có thể xảy ra trường hợp hàng loạt dữ liệu được ghi vào một node. Gây mất cân bằng trong cụm.
* **Nhận xét:** Với cả hai chiến lược partition trên, vẫn có những nhược điểm, khi số lượng node trong vòng quá ít, hoặc các node phân bố không đều theo giá trị băm của các bản ghi đưa vào, rất dễ đưa đến hiện tượng mất cân bằng, quá tải trong cụm. Ngoài ra, khi thêm hay xóa một node khỏi vòng, thì sẽ phải mất công tái cân bằng lại cụm.

### Snitch trong Cassandra

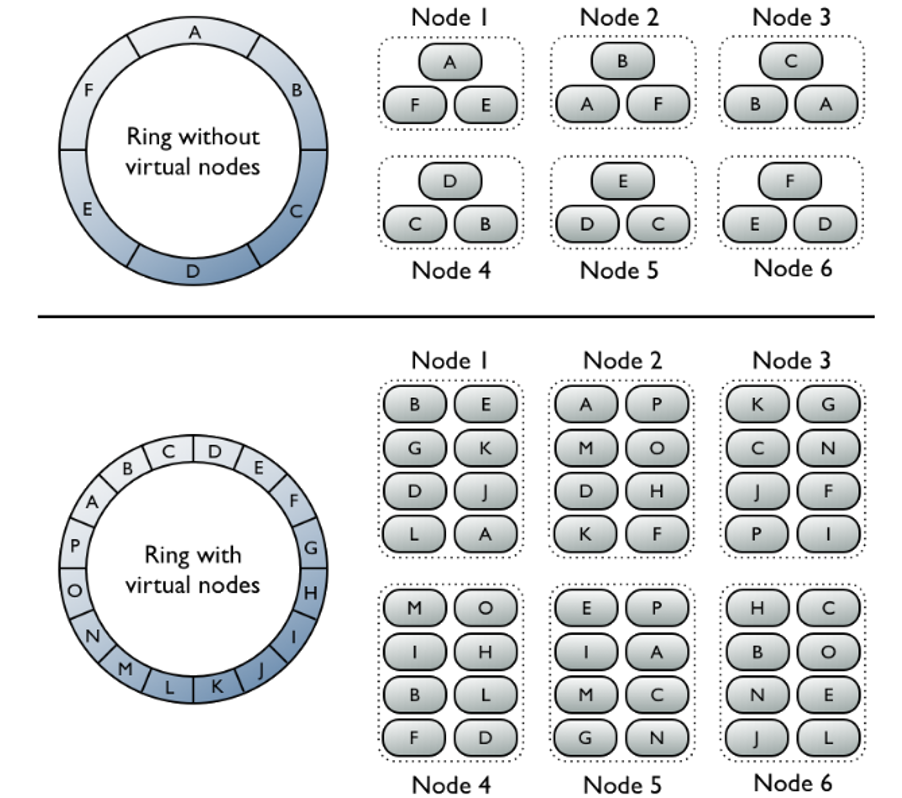
* Snitches đơn giản là một protocol sử dụng để mapping IP với Racks và Datacenter, áp dụng các snitches khác nhau thì dữ liệu sẽ được lưu trữ tại các điểm khác nhau trên cluster, hay hình dung đơn giản hơn snitches giúp ta thiết kế sơ đồ để lưu trữ dữ liệu (sở đồ mạng máy tính).
* Có một lưu ý quan trọng là tất cả các nodes trong một cluster thì được áp dụng cùng một snitche, nếu muốn thay đổi snitche cho cluster thì sửa đổi tên snitche cần áp dụng trong file cấu hình và sau đó restart toàn bộ cluster.
* Thông tin cấu hình snitches được lưu trong file cassandra.yaml. Các snitches được sử dụng trong Cassandra: SimpleSnitch: Lợi thế của sử dụng SimpleSnitch là không cần hiểu sâu về cách cài đặt, cấu hình Cassandra, SimpleSnitch không đòi hỏi các thông tin về thiết lập data center hay rack. Áp dụng loại Snitch này tốt khi triển khai Cassandra trên một máy đơn lẻ, khi thiết lập SimpleSnitch cần thiết lập replication\_factor = # đối với strategy\_options:
* CREATE KEYSPACE IF NOT EXISTS demo WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : # };
* Ngoài ra còn có các kiểu snitches dưới đây:
* Dynamic Snitching
* RackInferringSnitch
* PropertyFileSnitch
* GossipingPropertyFileSnitch
* Ec2Snitch
* Ec2MultiRegionSnitch
* GoogleCloudSnitch
* CloudstackSnitch

### Virtual node - Node ảo trong Cassandra

* Để giải quyết vấn đề tái cân bằng lại cụm, Cassandra đề ra một giải pháp đó là sử dụng node ảo. Node ảo giống như một thành phần của vòng tròn trong hệ thống, nhưng bản chất node ảo chỉ là ánh xạ của một node vật lý đến một địa chỉ khác trong vòng. Khi dữ liệu đi vào vùng quản lý của node ảo, nó sẽ được đưa về lưu trữ tại node vật lý của node ảo đó.
* Mỗi node vật lý khi tham gia vào vòng sẽ được gán một vị trí của chính node đó và gán thêm một số lượng các vị trí khác (được coi như là node ảo của node đó). Cassandra cấu hình mặc định mỗi một node tham gia vòng sẽ được gán 256 node ảo trong vòng.

[](https://laptrinh.vn/uploads/images/gallery/2021-06/vitural-node.png)

* Hình trên thể hiện một vòng trong có 4 node vật lý, mỗi node được gán thêm 7 node ảo, như vậy tổng cộng trên vòng tròn sẽ có 32 phân vùng key. Khi việc phân tán đều các node ảo ra khắp vòng, số lượng node tăng lên khiến cho các phân vùng key bé lại, việc phân vùng key bé lại mang ý nghĩa rất lớn trong việc phân bổ dữ liệu của cụm Cassandra, việc phân vùng nhỏ lại và các node sát nhau hơn đưa hệ thống càng gần đến với việc tất cả dữ liệu sẽ được phân bổ đều khắp các node, xác suất dữ liệu được đưa vào các node là cân bằng nhau khi mà trên một khoảng key nhỏ ta có đầy đủ các node ảo hoặc node vật lý. Trường hợp hoàn hảo nhất là các node vật lý đều có thành phần hiện diện của mình đều khắp trên vòng.

[](https://laptrinh.vn/uploads/images/gallery/2021-06/virtual-node-2.png)

Cân bằng của cụm khi có và không có node ảo

* Ring không có virtual node: mỗi node được gán với một single token mà thể hiện vị trí trong ring. Mỗi node lưu trữ dữ liệu được ánh xạ bởi partition key đến token value bên trong mỗi range bởi node phía trước gán giá trị cho nó. Mỗi node cũng chứa bản sao của mỗi hàng từ node khác trong cluster. Ví dụ như Range E có các bản sao trên node 5, 6, 1. Đặc biệt một node sở hữu chính xác một partition range trong ring space.
* Ring có Vnode: Trong cluster, vnode sẽ ngẫu nhiên chọn rac các phân vùng của dữ liệu và lưu trữ trên các node. Vị trí của một row được xác định bởi giá trị băm của partion key trong nhiều partition range nhỏ hơn thuộc về mỗi node.

## Mô hình lưu trữ dữ liệu

### Mô hình lưu trữ

* Apache Cassandra lưu trữ dữ liệu trong các bảng, với mỗi bảng bao gồm các hàng và cột. CQL (Cassandra Query Language) được sử dụng để truy vấn dữ liệu, được lưu trữ trong bảng. Mô hình dữ liệu Apache Cassandra dựa trên và được tối ưu hóa để truy vấn. Cassandra không hỗ trợ mô hình dữ liệu quan hệ dành cho cơ sở dữ liệu quan hệ.
* Dữ liệu được lưu trữ trong DB của Cassandra thuộc dạng Key value store (KVS).
* Cấu trúc dữ liệu Cassandra cơ bản:
* Column: là một cặp key/value
* Row: là vùng chứa cho các cột được tham chiếu bởi khóa chính
* Table: là một vùng chứa cho các hàng
* Keyspace: là vùng chứa các bảng
* Cluster: là một vùng chứa cho các Keyspace kéo dài một hoặc nhiều nút

Graphical user interface, table

Description automatically generated

Cassandra table

A screenshot of a computer

Description automatically generated with medium confidence

Cấu trúc dữ liệu Cassandra cơ bản

* Cluster: Cơ sở dữ liệu Cassandra được thiết kế đặc biệt để phân phối trên một số máy hoạt động cùng nhau xuất hiện như một phiên bản duy nhất cho người dùng cuối. Vì vậy, cấu trúc ngoài cùng trong Cassandra là cụm (cluster), đôi khi được gọi là vòng (ring), vì Cassandra gán dữ liệu cho các nút trong cụm (cluster) bằng cách sắp xếp chúng trong một vòng (ring).
* Table: Table là một vùng chứa cho một tập hợp các hàng (row) có thứ tự, mỗi hàng trong số đó chính là một tập hợp các cột (column) được sắp xếp theo thứ tự. Thứ tự được xác định bởi các cột, được xác định là các key.
* Row: Khi ghi dữ liệu vào bảng trong Cassandra, ta chỉ định giá trị cho một hoặc nhiều cột. Tập hợp các giá trị đó được gọi là một hàng (row). Ít nhất một trong các giá trị ta chỉ định phải là khóa chính (primary key) đóng vai trò là mã định danh duy nhất cho hàng đó.
* Columns:

Columns (cột) là đơn vị cơ bản nhất của cấu trúc dữ liệu trong mô hình dữ liệu Cassandra. Cột là một tập hợp dữ liệu (bộ 3) gồm tên cột, giá trị, và mốc thời gian.

* Column Family: Dùng để mô tả cấu trúc của dữ liệu. Mỗi một Column family có nhiều rows và mỗi row lại có nhiều columns theo thứ tự nhất định (khác với Relational Database, người dùng có thể tự do thêm column vào bất kỳ lúc nào và các row không nhất thiết phải có cùng các columns). Thông thường mỗi Keyspace thường có ít nhất một hoặc nhiều column families.
* Keyspace: Keyspace là một kho chứa các Column Family. Tất cả dữ liệu của Cassandra được đặt trong Keyspace nên có thể tạo một Keyspace duy nhất cho tất các các Column Family. Nhưng ta hoàn toàn có thể tạo nhiều Keyspace cho mỗi dự án khác nhau. Tất cả dữ liệu trong Cassandra sẽ đặt trong một keyspace. Nó có thể được xem như là một cơ sở dữ liệu trong RDBMS, nó là một tập hợp các bảng. Trong trường hợp của Cassandra, một keyspace là một tập hợp hệ thống cột.
* Nguyên tắc thiết kế database:
  + Đối với các hệ quản trị NoSQL, việc thiết kế database sẽ khác nhau cho từng loại. Có loại sẽ ưu tiên cho khả năng đọc, có loại sẽ ưu tiên việc ghi, có loại cần thiết kế để ghi – đọc là tương đương với nhau, tùy theo tính chất lưu trữ và cách database hoạt động. Cassandra thuộc loại lưu trữ ưu tiên ghi, vì database này không hỗ trợ các hàm tính toán mà chỉ cho phép các truy vấn và câu lệnh CRUD dơn giản, trong khi chi phí ghi dữ liệu của database này rất rẻ so với việc truy vấn. Nguyên tác thiết kế database trên Cassandra là tối ưu để lấy nhiều dữ liệu nhất có thể trong 1 lần truy vấn.
  + Ngoài ra, chúng ta cần tối đa hóa dữ liệu trùng lặp. Mình biết điều này sai sai khi mình nói xong, nhưng không nhầm đâu. Dung lương ổ đĩa không đắt hơn bộ xử lý CPU hay giá tiền cho hoạt động IO thông thường. Vì Cassandra là một database phân tán, bạn sẽ muốn việc sao chép dữ liệu xảy ra liên tục để có thể truy xuất dữ liệu ngay lập tức và không có điểm lỗi nào trong hệ thống.
  + Trong Cassandra, dữ liệu được lan truyền đến các nút khác nhau dựa trên các khóa phân vùng là phần đầu tiên của khóa chính. Vì vậy, để dữ liệu có thể lan truyền đều trong mỗi nút của cụm, hãy cố gắng chọn số nguyên làm khóa chính.
  + Hãy thiết lập số lương phân vùng để đọc một cách vừa phải. Phân vùng là một nhóm các bản ghi có cùng khóa phân vùng. Khi một truy vấn được đưa ra, Cassandra sẽ thu thập dữ liệu từ các nút khác nhau của các phân vùng khác nhau. Nếu có nhiều phân vùng, thì tất cả các phân vùng này cần được truy cập để thu thập dữ liệu truy vấn. Tất nhiên, điều đó không có nghĩa là không nên tạo phân vùng, chỉ là bạn nên tối ưu việc tạo phân vùng cho phù hợp để có được một hiệu suất tốt nhất.
  + Xét 1 ví dụ sau trong việc tổ chức database: Ta có yêu cầu lưu trữ thông tin bài hát trên Cassandra.
  + Đây là cách đầu tiên, về cơ bản, đây là cách tổ chức thường thấy trong RDMS

Create table MusicPlaylist

(

SongId int,

SongName text,

Year int,

Singer text,

Primary key(SongId, SongName)

);

* + Trong ví dụ này, chúng ta thiết lập SongId là Patition Key, SongName là Cluster Column. Dữ liệu sẽ được nhóm lại trên cơ sở của SongName, chỉ 1 phân vùng được tạo ra với SongId. Việc truy xuất dữ liệu trong trường hợp dữ liệu phình to sẽ rất tệ dù bạn đã tuân thủ hạn chế việc tạo phân vùng. Thực ra, cách làm tối ưu hơn là dự phòng trường hợp dữ liệu quá nhiều và ta nên tìm cách tách phân vùng khi dữ liệu đủ lớn.
  + Hãy xem cách tạo table thứ 2:

Create table MusicPlaylist

(

SongId int,

SongName text,

Year int,

Singer text,

Primary key((SongId, Year), SongName)

);

* + Trong cách này, chúng ta có SongId và Year được thiết lập là Patition Key, SongName là Cluster Column. Lúc này các phân vùng sẽ được tạo ra dựa trên SongId và Year, tức là qua từng năm, chúng ta sẽ thiết lập được 1 phân vùng mới. Hệ thống này đảm bảo sự hiệu quả trong truy vấn dữ liệu.

### Phân vùng dữ liệu - Partitioners trong Cassandra.

Partitioner là việc bạn quyết định việc dữ liệu được phân tán như thế nào trên các node trong cluster (bao gồm cả các bản sao). Trong Cassandra, dữ liệu được quản lý bởi một cluster được đại diện như một không gian dữ liệu hay một ring. Vòng tròn (ring) được chia tương ứng với phạm vi là số lượng các node, mỗi node quản lý một hoặc nhiều vùng của dữ liệu. Trước khi một node có thể tham gia vòng nó được gắn một giá trị token (thẻ bài). Token xác định vị trí của node trên ring và phạm vi dữ liệu mà nó quản lý.

Application cần chỉ rõ giá trị nằm trong khoảng giữa các token và Cassandra sử dụng nó để điều hướng request tới node chứa dữ liệu đích. Cassandra phân vùng dữ liệu trên cluster sử dụng consistent hashing. Trong consistent hashing, phạm vi output trả về bởi hash function được chia ra trên một ring. Consistent hashing cho phép việc phân tán dữ liệu trên các cluster mà giảm tối thiểu việc tái cấu trúc lại khi có một hoặc nhiều node được thêm vào hoặc xóa đi trong cluster. Consistent hashing partitions data dựa trên partition key.

Ví dụ dữ liệu với trường name được chọn là partition key. Các giá trị trên trường name được băm ra và sắp xếp theo thứ tự như bảng dưới đây:

Table

Description automatically generated

Sau đó mỗi node trên cluster sẽ quản lý một range dữ liệu dựa trên các giá trị băm được từ partition key.

Diagram

Description automatically generated

Diagram

Description automatically generated

Như hình trên chúng ta xem xét một cụm đơn giản gồm 4 nút, nơi tất cả các dữ liệu được quản lý bởi 1 cụm được đánh số trong khoảng từ 0 đến 100. Mỗi nút được gán một thẻ bài đại diện cho một điểm trong phạm vi này. Trong ví dụ này, các thẻ có giá trị là 0, 25, 50, và 75. Nút đầu tiên, với token 0, chịu trách nhiệm về phạm vi gói (75-0). Nút với thẻ bài thấp nhất cũng chấp nhận khóa hàng ít hơn so với các mã thẻ bài thấp nhất và nhiều hơn với các mã thẻ bài cao nhất.

Partitioner quyết định nơi lưu trữ dữ liệu. Về cơ bản Partitioner chỉ là các hàm băm, sử dụng thuật toán băm đối với các Rowkey để xác định các token, dựa trên giá trị các token mà dữ liệu được xác định sẽ lưu tại node nào trên cluster. Cassandra cung cấp 2 loại Partitioner: ByteOrderedPartitioner: đây là partitioner được Cassandra support sớm nhất. Theo phương pháp này khóa được lưu trữ theo thứ tự các raw byte thay vì chuyển đổi chúng sang các chuỗi mã hóa. Tokens được tính bằng cách nhìn vào các giá trị thực tế của dữ liệu, sử dụng hệ thập lục phân cho các ký tự đầu trong khóa. Ưu điểm của phương pháp này là khi đã biết được cấu trúc dữ liệu, ta có thể tìm kiếm dữ liệu rất nhanh, ví dụ khi tìm kiếm tên của một khách hàng là "Lan", ta biết được phân vùng của node chứa các khách hàng có ký tự đầu tiên là "L" và truy cập thẳng vào đó mà không phải tìm qua các node khác. (đối với mỗi node thì được chia thành từng khoảng một để lưu trữ dữ liệu. ví dụ ta có thết lập node 1 lưu trữ các ký tự A -> F, node 2 lưu trữ các ký tự G -> K và node 3 lưu trữ các kỹ tự K -> Z). Nhược điểm của phương pháp này chính là khi dữ liệu trên một tập nào đó quá lớn, ví dụ một tập khách hàng có chữ cái đầu của tên nằm trong khoảng từ A->F quá lớn, còn các chữ cái khác thì không nhiều, khi đó node 1 sẽ chịu nhiều tác động (dễ rơi vào tình trạng bottomneck) trong khi các node khác lại không thể hiện nhiều vài trò của nó trên hệ thống, đây còn gọi là hiện thượng mất cân bằng (not balancing). Chính vì lý do này mà trong tài liệu có khuyến cáo không nên sử dụng loại phân vùng này.

### Cơ chế nhân bản trong Cassandra (Replication Strategie)

Thực ra đây là cách xác định nhân bản dữ liệu trên cluster. Ban đầu khi xây dựng hệ thống, người dùng phải xác định được chiến lược sao lưu dữ liệu trên một hay nhiều máy, xác định mức độ High avaibility (tính sẵn sàng cao) để đưa ra được chiến lược nhân bản dữ liệu phù hợp. Cassandra cung cấp hai cách nhân bản dữ liệu khác nhau: nhân bản dữ liệu trên một máy (SimpleStrategy) hoặc phân tán dữ liệu trên nhiều máy (NetworkTopologyStrategy) trong cùng cluster. SimpleStrategy: được sử dụng trên một máy đơn lẻ hoặc database được lưu trên một data center và cũng là giá trị mặc định khi tạo keyspace. Đối với trường hợp Cassandra được cài trên một máy thì không cần thảo luận nhiều, vì tất cả dữ liệu đều nằm trên một máy, trong trường hợp cassandra được cài trên cluster là một datacenter, thì phiên bản dữ liệu đầu tiên được xác định bởi việc lựa chọn các Partitioner, từ phiên bản thứ hai trở đi, dữ liệu sẽ được phân bố trên các node tiếp theo, dọc theo chiều kim đồng hồ tính từ phiên bản đầu tiên. Ví dụ tạo một keyspace demo với SimpleStrategy:

CREATE KEYSPACE IF NOT EXISTS demo WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 3 };

NetworkTopologyStrategy: Khi chúng ta có kế hoạch nhân bản dữ liệu trên nhiều data center trong một cluster thì NetworkTopologyStrategy là một sự lựa chọn hợp lý. Strategy này sẽ xác định bao nhiêu bản sao sẽ được lưu trữ trên mỗi data center. Ở đây việc dữ liệu được chỉ định dữ liệu được lưu trên một hay nhiều rack trong cùng một data center hoặc trên nhiều data center là phụ thuộc vào từng bài toán. Đối với các bài toán cần ưu tiên lấy nhanh dữ liệu, độ trễ thấp thì người ta ưu tiên lưu dữ liệu trên các rack khác nhau của cùng một data center (khi đó dữ liệu không phải lấy từ các dải mạng khác nhau), trái lại để đáp ứng hệ thống có tính sẵn sàng cao, giảm tối thiểu rủi do cho hệ thống (ví dụ, các rủi ro như mất điện, lỗi mạng ....) thì người ta lưu dữ liệu trên các data center khác nhau.

Cách nhân bản dữ liệu là dùng Snitches - Sniches đơn giản là một protocol sử dụng để mapping IP với Racks và Datacenter, áp dụng các snitches khác nhau thì dữ liệu sẽ được lưu trữ tại các điểm khác nhau trên cluster, hay hình dung đơn giản hơn snitches giúp ta thiết kế sơ đồ để lưu trữ dữ liệu (sở đồ mạng máy tính). Có một lưu ý quan trọng là tất cả các nodes trong một cluster thì được áp dụng cùng một snitche, nếu muốn thay đổi snitche cho cluster thì sửa đổi tên snitche cần áp dụng trong file cấu hình và sau đó restart toàn bộ cluster. Thông tin cấu hình snitches được lưu trong file cassandra.yaml. Các snitches được sử dụng trong Cassandra: SimpleSnitch: Lợi thế của sử dụng SimpleSnitch là không cần hiểu sâu về cách cài đặt, cấu hình Cassandra, SimpleSnitch không đòi hỏi các thông tin về thiết lập data center or rack.

### Tính nhất quán trong Cassandra (consistency)

Trong Cassandra, consistency refer tới cách để update, đồng bộ row dữ liệu lên tất cả các phiên bản của nó. Cassandra mở rộng khái niệm eventual consistency bởi việc offer tunable consistency. Cho mỗi tác vụ đọc ghi, client sẽ quyết định loại consistency bằng việc cấu hình trong code.

Thêm vào cơ chế nhất quán, Cassandra còn hỗ trợ một số built-in của cơ chế sửa chửa để đảm bảo dữ liệu vẫn còn consistent qua nhiều bản sao. Cấp độ consistency trong Cassandra có thể được set trên mỗi câu truy vấn đọc ghi. Cái này cho phép nhà phát triển ứng dụng điều chỉnh consistency trên mỗi câu truy ván dựa vào yêu cầu của họ là nhận được thời gian đáp ứng nhanh hay kết quả trả về chính xác. Cassandra offer 1 số cấp độ nhất quán cho cả đọc và ghi.

#### Nhất quán ghi

Khi thực hiện ghi trong Cassandra, consistency level đặc tả có bao nhiêu bản sao được viết thành công trước khi trả về kết quả cho client. Theo những consistency level được cung cấp, ANY là consistency thấp nhất (nhưng độ sẵn sàng cao nhất). QUORUM ở khoảng giữa đảm bảo nhất quán tốt nhất nhưng vẫn còn chịu 1 vài cấp độ lỗi.

#### Mức độ nhất quán - Consistency Levels

Chọn consistency level để đọc ghi bao gồm việc quyết định yêu cầu của bạn giữa kết quả nhất quán và độ trễ đọc ghi. Nếu độ trễ đọc ghi là ưu tiên hàng đầu, xem xét consistency level ONE. Nếu nhất quán là ưu tiên hàng đầu, bạn có thể đảm bảo rằng đọc ghi sẽ luôn phản ánh tác vụ ghi gần nhất bằng cách dùng công thức

(nodes\_written + nodes\_read) > replication\_factor

#### Đọc dữ liệu (read repair)

Cho tác vụ read, có 2 loại read request mà một coordinator có thể gửi cho một bản sao a direct read request and a background read repair request. Số bản sao tiếp xúc trực tiếp với read request phụ thuộc vào consistent level đặc tả bởi client. Background read repair request được gửi cho những bản sao còn lại không nhận được read request để đảm bảo rằng dữ liệu đọc thường xuyên vẫn còn nhất quán. Coordinator sẽ so sánh dữ liệu từ tất cả các bản sao còn lại có sở hữu row request. Và nếu chúng không nhất quán thì nó sẽ có một tác vụ ghi để cập nhật lại những giá trị đã lỗi thời sử dụng giá trị được cập nhật sau cùng. Read repair có thể được cấu hình cho mỗi column family và được enable mặc định.

#### Anti-Entropy Node Repair

Cho những dữ liệu không được đọc thường xuyên hoặc dữ liệu update trên một node đã down, nodetool repair (ám chỉ anti-entropy node repair) đảm bảo rằng tất cả dữ liệu trên bản sao được tạo nhất quán. Node tool repair phài được chạy định kì tại mỗi node để duy trì sự nhất quán dữ liệu.

### Node điều phối (coordinator node)

Để thỏa mãn tính sẵn sàng và liên tục trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc là phiên bản cũ, không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Đồng thời với việc này Cassandra tiến hành thao tác sửa lỗi đọc (read repair) là tiến trình ngầm để cập nhật trạng thái mới nhất cho tất cả các máy chủ lưu trữ nhân bản của dữ liệu. Cassandra tổ chức các nút máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo bảng hàm băm nhất quán (Distributed consistent hashing). Nếu mỗi dữ liệu của Cassandra được sao lưu trên N nút, khi một khóa k được quyết định sẽ lưu vào một nút nào đó, nút đó sẽ được coi là nút điều phối. Nút điều phối có nhiệm vụ phân phối bản ghi đấy cho N-1 nút còn lại theo nguyên tắc: từ nút điều phối, đi theo chiều kim đồng hồ, dữ liệu sẽ được ghi lên 2 nút tiếp theo được gặp.

Diagram

Description automatically generated

Hình trên mô tả khi khóa k được xác định là sẽ ghi vào nút B, nút B sẽ đóng vai trò điều phối, luân chuyển khóa đấy cho 2 nút tiếp theo là nút C và nút D. như vậy, nút D sẽ lưu trữ các khóa nằm trong vùng (A; D]. Danh sách các khóa trong vùng này được gọi là danh sách liên kết của nút D. Việc đưa các giá trị của khóa k sang các nút khác áp dụng cho tất cả các tác vụ ghi, cập nhật hay xóa. Vì việc việc quyết định số lượng nút được luân chuyển ngay lập tức mỗi khi có tác vụ ghi diễn ra ảnh hưởng trực tiếp đến mức độ nhất quán của hệ thống. Trong cấu hình của Cassandra Apache ta có một chỉ số "replicatioon\_factor" và "w". Chỉ số "replication\_factor sẽ" được cài đặt ngay khi khởi tạo một key\_space, đó là số lượng nút trong vòng sẽ được dùng để sao lưu dữ liệu. Chỉ số "w" khi cấu hình Cassandra là số lượng nút trả về kết quả khi thực hiện tác vụ ghi bắt buộc để tác vụ đấy được coi là thành công. Xét hình trên, khi ta đặt replication\_factor = 3 và w = 2, khi khóa k được ghi vào thì cần phải có ít nhất 2 nút trong 3 nút B, C, D phản hồi lại ghi thành công thì tác vụ đấy mới được coi là thành công. Việc cài đặt chỉ số "w" cho ta thấy mức độ chi phí ta có thể bỏ ra để đảm bảo tính nhất quán của dữ liệu ngay lập tức. Việc nhân bản dữ liệu cũng ảnh hưởng đến mức độ nhất quán của hệ thống. Mức độ nhất quán xét trên cả 2 phương diện đó là đọc và ghi dữ liệu. Để duy trì mức độ nhất quán của dữ liệu, Cassandra cung cấp cho người dùng nhiều mức độ nhất quán của các tác vụ đọc và ghi. Từ mức độ cao nhất đến thấp nhất, ta có thể điều chỉnh mức nhất quán dựa vào hai tham số cấu hình là "w" và "r" cùng với chỉ số "replication\_factor". Trong đó, "w" là số nút trả về khi ghi thành công, "r" là số nút trả về khi đọc thành công. Nếu như tính nhất quán là sự ưu tiên, ta có thể đặt "w" và "r" sao cho đảm bảo.

Logo

Description automatically generated

Và nên đảm bảo "w" hoặc "r" luôn nhỏ hơn replication\_factor để cho được độ trễ tốt hơn. Giả sử như replication\_factor = 3, vậy 2 giá trị của "w" và "r" tốt nhất sẽ là 2. Nghĩa là mỗi khi đọc và ghi dữ liệu, cần ít nhất 2 nút trả về giá trị thì tác vụ đó coi là thành công. Và khi tác vụ đọc hoặc ghi thực hiện, sẽ luôn đảm bảo sẽ được thực hiện trên dữ liệu mới nhất mà tác vụ trước đó đã thực hiện.

### Hinted Handoff

Hinted handoff là một tính năng nổi bật của Cassandra để giảm thời gian restore một node bị hỏng trở nên nhất quán. Nó cũng có thể được dùng để sẵn sang viết cho những ứng dụng không chịu được lỗi do ghi sai nhưng chịu được lỗi do đọc không nhất quán. Khi một tác vụ ghi được tạo, Cassandra nỗ lực viết vào tất cả các bản sao cho những row bị ảnh hưởng. Nếu một bản sao bị down tại thời điểm ghi, một bản sao sống tương ứng sẽ lưu trữ một hint. Hint bao gồm thông tin về vị trí cũng như dữ liệu được viết.

Nếu tất cả các bản sao của những row bị tác động đều down. Nó vẫn có thể viết thành công nếu cùng consistent level ANY. Dưới ngữ cảnh này, dữ liệu được ghi và dữ liệu hint được lưu trữ trên node coordinator nhưng sẽ không sẵn sang để đọc cho đến khi hint ghi vào bản sao thật sự sở hữu row đó. ANY hỗ trợ absolute write availability. Cái này không đảm bảo dữ liệu được viết sẽ sẵng sang được đọc. ANY cũng có thể tăng khả năng load cho cluster, khi coordinator phải lưu trữ tạm những row bất cứ khi nào bản sao không sẵn sang để ghi

Mặc định hint chỉ lưu một giờ trước khi chúng bị drop. Nếu tất cả các bản sao bị down tại thời điểm ghi và chúng vẫn còn down dài hơn max\_hint\_window\_in\_ms thì có thể có khả năng mất dữ liệu

Hints không thể cấu hình consistent level khác ANY. Ví dụ nếu là ONE và tất cả các bản sao cho dòng được viết đều down thì việc ghi đều thất bại bất kể là hint được ghi hay không.

### CQL:

#### Tạo Keyspace:

Một keyspace được tạo bằng câu lệnh CREATE KEYSPACE:

create\_keyspace\_statement::= CREATE KEYSPACE [ IF NOT EXISTS ] keyspace\_name WITH options

Ví dụ:

create keyspace project\_keyspace with replication = {'class': 'SimpleStrategy', 'replication\_factor': '1'} AND durable\_writes = 'true';

Các options được hỗ trợ là:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tên** | **Loại** | **Bắt buộc** | **Mặc định** | **Mô tả** |
| replication | Map | Có | n/a | Sao chép strategy và các tùy chọn để sử dụng cho keyspace |
| durable\_writes | simple | Không | true | Có sử dụng commit log cho các bản cập nhật trên keyspace này hay không |

replication là bắt buộc và phải chứa tùy chọn con 'class' xác định lớp chiến lược nhân bản mong muốn. Phần còn lại của các tùy chọn phụ phụ thuộc vào chiến lược sao chép nào được sử dụng. Theo mặc định, Cassandra hỗ trợ các giá trị 'class' sau:

* SimpleStrategy:

Một chiến lược đơn giản xác định yếu tố sao chép để dữ liệu được trải rộng trên toàn bộ cụm. Đây thường không phải là một lựa chọn khôn ngoan cho quá trình sản xuất, vì nó không tôn trọng bố cục data center và có thể dẫn đến độ trễ truy vấn rất khác nhau. Để sản xuất, hãy sử dụng NetworkTopologyStrategy. SimpleStrategy hỗ trợ một đối số bắt buộc duy nhất:

|  |  |  |
| --- | --- | --- |
| Sub-option | Loại | Mô tả |
| 'replication\_factor' | Int | Số lượng bản sao cần lưu trữ trên mỗi phạm vi |

* NetworkTopologyStrategy:

Chiến lược sao chép đặt hệ số sao chép độc lập cho mỗi data center. Phần còn lại của các tùy chọn phụ là các cặp khóa-giá trị, với khóa được đặt thành tên data center và giá trị của nó được đặt thành yếu tố sao chép liên quan. Tùy chọn:

| **sub-option** | **Loại** | **Mô tả** | **'<datacenter>'** |
| --- | --- | --- | --- |
| int | Số lượng bản sao cần lưu trữ trên mỗi phạm vi trong data center được cung cấp. | 'replication\_factor' | int |

#### Use keyspace:

Câu lệnh USE thay đổi keyspace hiện tại thành keyspace được chỉ định:

use\_statement::= USE keyspace\_name

Ví dụ:

use project\_keyspace;

#### ALTER Keyspace:

Câu lệnh ALTER KEYSPACE sửa đổi các tùy chọn của một keyspace:

alter\_keyspace\_statement::= ALTER KEYSPACE [ IF EXISTS ] keyspace\_name WITH options

Ví dụ:

Alter keyspace project\_keyspace with replication = {'class': 'SimpleStrategy', 'replication\_factor': '4'} ;

#### DROP Keyspace:

Việc loại bỏ một keyspace được thực hiện với câu lệnh DROP KEYSPACE:

drop\_keyspace\_statement::= DROP KEYSPACE [ IF EXISTS ] keyspace\_name

Ví dụ:

Drop keyspace project\_keyspace;

#### Create Table:

Tạo một bảng mới sử dụng câu lệnh CREATE TABLE

* Cú pháp create table trong Cassandra:

Text, letter

Description automatically generated

Mệnh đề Create

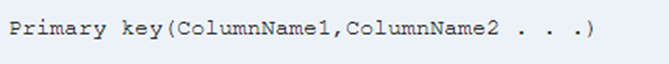
* + - Một Table sẽ được gán với 1 Key\_space cụ thể. Trong table sẽ khai báo Column Name và kiểu dữ liệu của Column đó (tương tự với các hệ RDMS).
    - Primary key là bắt buộc trong Cassandra, phụ trách chia phân vùng cho Table
    - Cassandra có 2 dạng Primary key là Single và Compound
      * Single:



Primary key

Với khai báo này, khóa chính và khóa phân vùng thuộc cùng 1 Column. Dữ liệu sẽ được phân vùng dựa vào Column này và được lan truyền tới các nút trên cơ sở khóa phân vùng.

* + - * Compound Primary Key:



Khai báo Primary key

Với khai báo này, khóa chính là Column đầu tiên được khai báo, Column thứ 2 là khóa phân cụm. Dữ liệu sẽ được phân vùng dựa trên khóa chính và phân cụm dựa trên khóa phân cụm. Quá trình phân cụm là quá trình phân bố dữ liệu trên 1 phân vùng.

* + - * Compound Partition Key:



Khai báo Partition key

Với khai báo này, khóa phân vùng là Column 1 và 2. Cách khai báo này sẽ áp dụng khi người dùng có quá nhiều dữ liệu trên 1 phân vùng, dữ liệu sẽ được tự động được chia theo khóa phân vùng vào nhiều vùng khác nhau. Column 3 sẽ đóng vai trò là khóa phân cụm trong trường hợp này.

#### ALTER Table:

Thay đổi bảng hiện có sử dụng câu lệnh ALTER TABLE

* + Cú pháp thay đổi một table trong Cassandra:

Text

Description automatically generated with low confidence

Mệnh đề Alter

Cần chỉ định chính xác Table trong KeySpace sẽ được thay đổi thông qua câu lệnh Alter

#### DROP Table:

Xóa bảng sử dụng câu lệnh DROP TABLE

Logo

Description automatically generated

Mệnh đề Drop

Dữ liệu sẽ được sao lưu trước khi thực hiện câu lệnh. Các câu truy vấn sau đó đến table này của keyspace đều trả về lỗi không tồn tại.

* + Truncate table cho phép xóa hoàn toàn dữ liệu trong 1 table nhưng không xóa table:



Mệnh đề Truncate

Dữ liệu được sao lưu tự động trước khi câu lệnh được thực thi. Mọi truy vấn sau đó đến table này sẽ trả về 0(rows) data.

#### Insert, Update, Delete và Where Clause

* + Insert Into:
    - Như mọi database thông thường, Cassandra hỗ trợ câu lệnh insert dữ liệu vào database. Tuy nhiên cấu trúc lệnh sẽ có hơi khác so với bình thường.
    - Đây là cấu trúc của 1 câu truy vấn Insert trong Cassandra:

Graphical user interface, text

Description automatically generated

Mệnh đề Insert

* + - Keyspace Name được tạo khi người dùng tạo 1 database mới trong Cassandra
    - Table Name là tên table mà người dung cần Insert dữ liệu, phải được tạo trong Key\_space
    - Thứ tự Insert dữ liệu sẽ tương ứng với Column Name trong câu lệnh.
    - Cassandra không tồn tại lệnh Upsert. Khi chạy lệnh Insert, Cassandra sẽ kiểm tra xem khóa chính đó đã tồn tại hay chưa. Nếu chưa, đây là 1 lệnh insert thông thường, nếu đã tồn tại thì row đó sẽ được cập nhật giá trị mới
  + Update:

Text, letter

Description automatically generated

Mệnh đề Update

* + - Nếu câu truy vấn gặp lỗi, Cassandra sẽ trả về chi tiết lỗi, ngược lại sẽ không trả về gì và mặc định truy vấn thành công.
    - Mệnh đề Set cho biết dữ liệu được thay đổi và giá trị của nó
    - Mệnh đề Where sẽ lọc các row cần được thay đổi trong database.
  + Delete:

Graphical user interface, text

Description automatically generated with medium confidence

Mệnh đề Delete

Tùy thuộc vào Where, Cassandra sẽ xóa 1 hoặc nhiều hàng khỏi database.

## 2.3 Các hạn chế của CQL

* + CQL không hỗ trợ các truy vấn tổng hợp như min, max, avg
  + CQL không hỗ trợ truy vấn group, having
  + CQL không hỗ trợ các mệnh đề Joins
  + CQL không hỗ trợ mệnh đề OR
  + CQL không hỗ trợ các truy vấn có kí tự đại diện
  + CQL không hỗ trợ các truy vấn Union, Intersection
  + Không thể lọc các cột trong bảng nếu bạn không set index
  + Truy vấn < và > chỉ được hỗ trợ trên các cột phân cụm (các cột đóng vai trò là khóa phân cụm).

-> Các vấn đề này dẫn tới Cassandra là 1 database ưu tiên ghi, với nguyên tắc thiết kế cần đáp ứng nhu cầu truy vấn nhiều dữ liệu nhất trong 1 lần thực thi và dữ liệu truy vấn cần gần nhất so với nhu cầu.

# Cài đặt

## Cài đặt và thiết lập cấu hình cơ bản Cassandra trên Ubuntu 20.04

* Cassandra từ bản 4.0 đã ngừng hỗ trợ Windows vì thiếu mức độ thử nghiệm của cộng đồng. Hiện tại chúng ta có thể cài đặt và thiết lập database này theo 3 cách chính là Docker, Tarball binary file và Package Installation. Hôm nay nhóm sẽ demo cách tiếp cận dễ nhất là cài đặt qua Tarball binary file.
* Đầu tiên chúng ta cần thiết lập môi trường để Cassandra có thể khởi chạy. Java 8 là yêu cầu bắt buộc vì gói cài đặt của Cassandra không tích hợp sẵn môi trường khởi chạy.
* Để cài đặt Java trên Ubuntu, sự dụng lệnh:

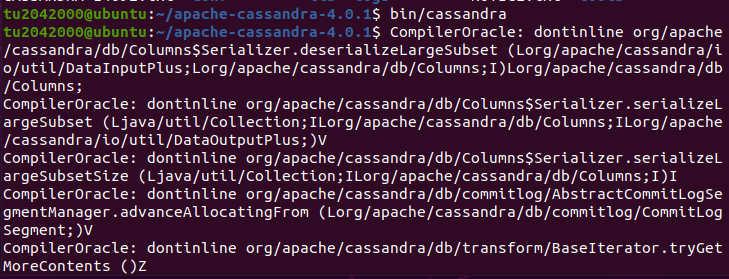
Sudo apt install java

* Kiểm tra việc cái đặt có thành công hay không, ta sử dụng lệnh:

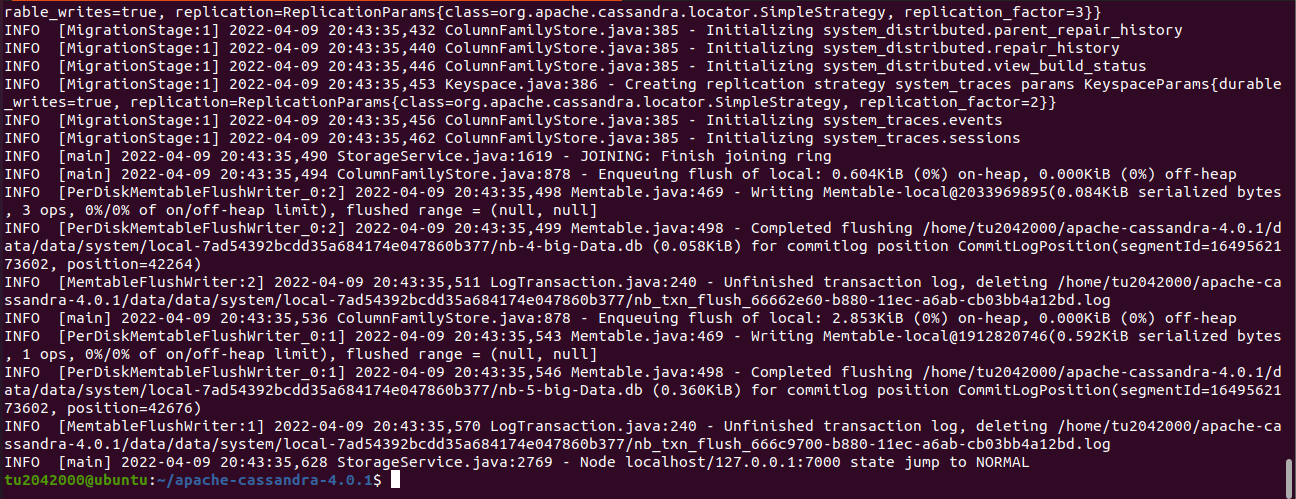
Java -version

* Tiếp theo ta cần download bản cài đặt Cassandra. Truy cập và tải bản cài đặt 4.0.1 tại đây: http://archive.apache.org/dist/cassandra/
* Giải nén file vừa download ra một vị trí khác trong máy và bật terminal lên trong thư mục vừa giải nén được.
* Sử dụng lệnh để khởi chạy Cassandra theo thiết lập mặc định:

bin/cassandra



Khởi chạy Cassandra



Khởi chạy cassandra

* Kiểm tra tình trạng database:

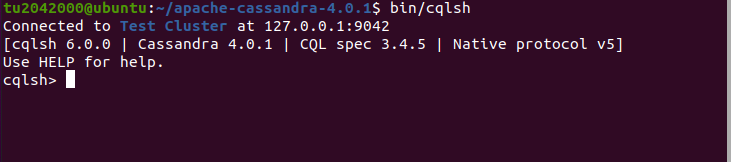
bin/nodetool status



Kiểm tra tình trạng database

* Sử dụng CQLSH để truy vấn dữ liệu:

Bin/cqlsh



Bắt đầu truy vấn dữ liệu

* Tạo một keyspace có tên là project\_keyspace, trong đó SimpleStrategy có nghĩa là nhân bản dữ liệu trên một máy. Sau đó tạo bảng có tên là hotels\_by\_pol với các thuộc tính như bên dưới. Sau đó insert dữ liệu vào bảng

A computer screen capture

Description automatically generated with low confidence

Tạo bảng mới

* Lấy ra toàn bộ dữ liệu trong bảng vừa tạo, ta được như bên dưới

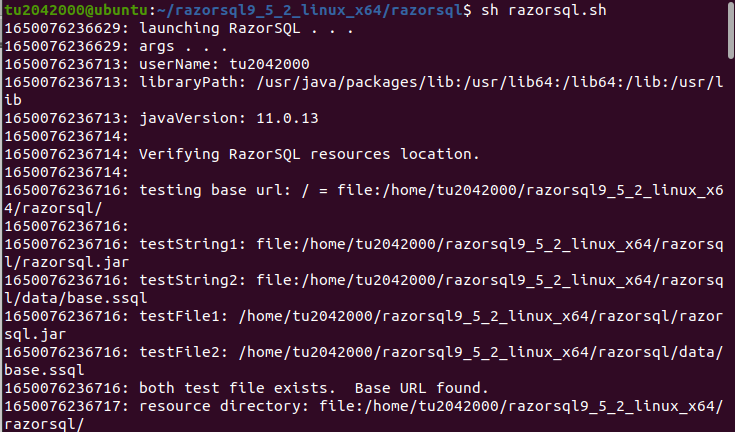
Text

Description automatically generated with medium confidence

Lấy dữ liệu trong bảng

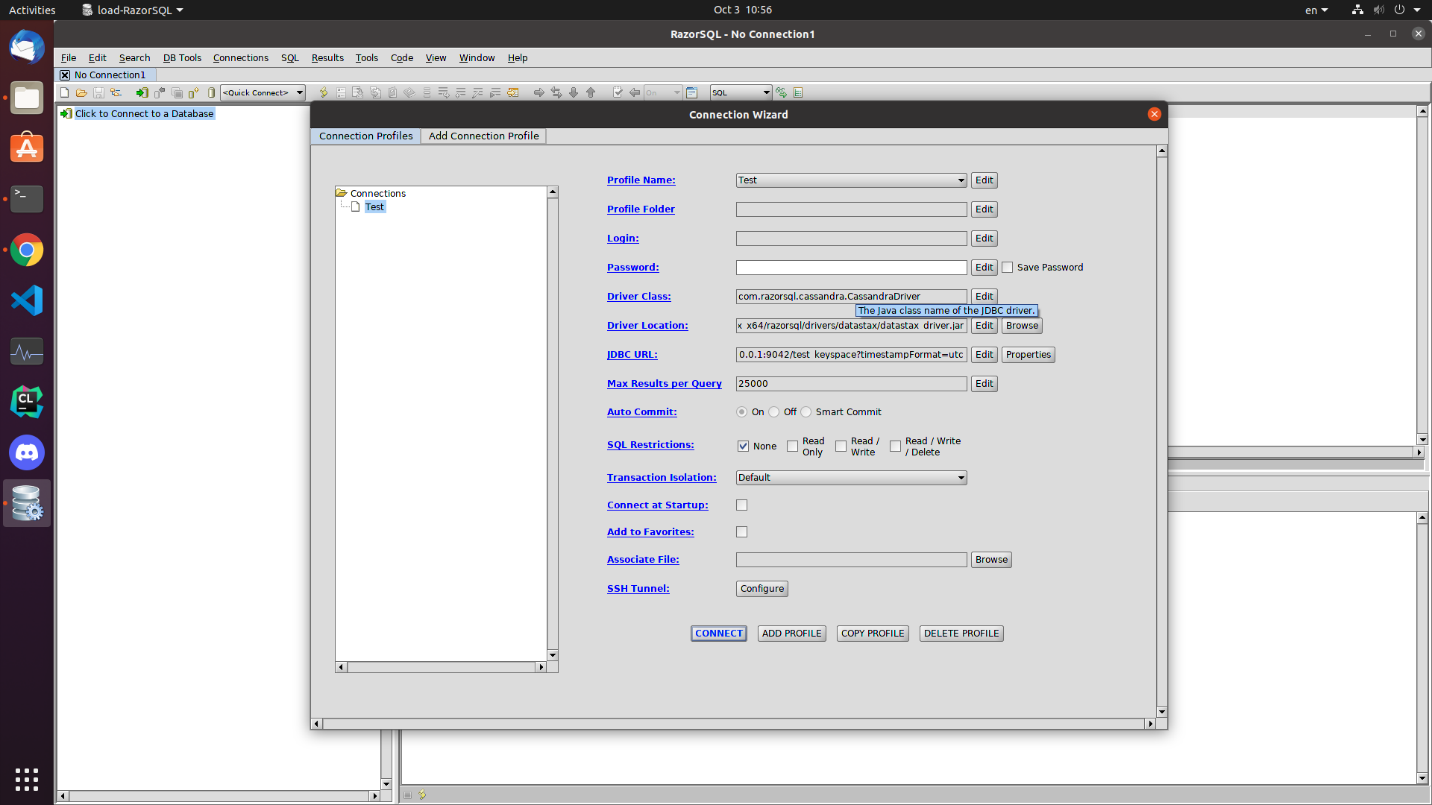
* Để dễ sử dụng hơn, chúng ta có thể dùng tool Razor SQL để truy vấn, chỉ cần tải file cài đặt tại đây: <https://razorsql.com/download_linux.html>. Sau đó giải nén và mở terminal tại thư mục vừa giải nén, chạy lệnh sau

Sh razorsql.sh



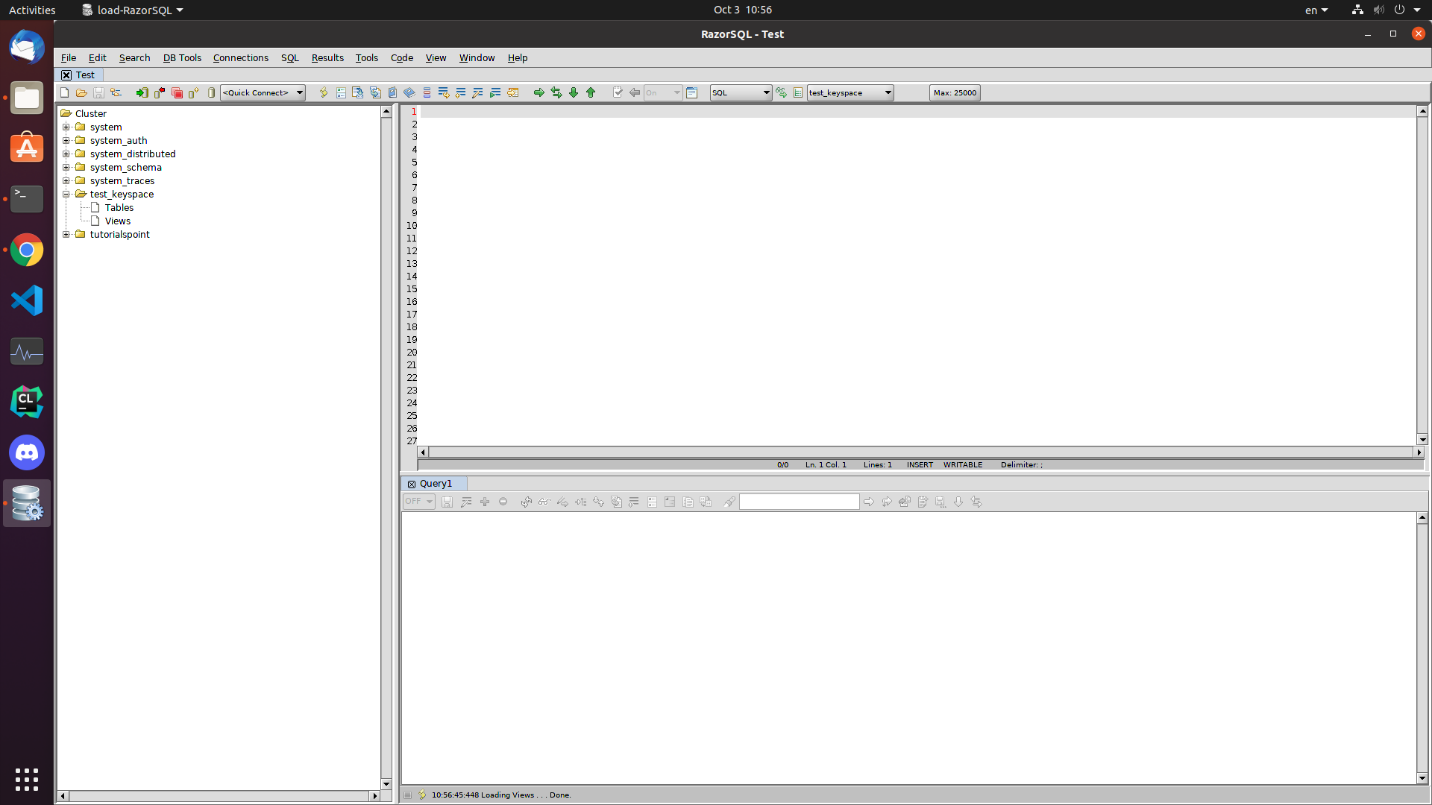
Khởi chạy RazorSQL

* Cấu hình Razor SQL để truy cập Cassandra



Cấu hình RazorSQL

* Giao diện sử dụng



Giao diện RazorSQL

## Cài đặt Cassandra 3.11.11 trên Window 10

* Yêu cầu: JDK 1.8.0\_241 và thấp hơn, Python 2.7

Text

Description automatically generated

Phiên bản Python và Java cần thiết

* Tải phiên bản apache-cassandra-3.11.11 từ website của Cassandra và tạo biến môi trường:

Graphical user interface, text, application

Description automatically generated

Biến môi trường

* Bên trong thư mục bin của Cassandra, copy các file từ Python sang.
* Trên cmd Admin, vào file Cassanda, thư mục bin, chạy lệnh cassandra.bat -f

Text

Description automatically generated

Khởi chạy Cassandra



Khởi chạy thành công

* Mở một cmd khác, vào thư mục bin của Cassandra, chạy lệnh cqlsh

Graphical user interface, text

Description automatically generated

cqlsh

# Ưu điểm và nhược điểm của Apache Cassandra

## Ưu điểm

* Kiến trúc ngang hàng: các nút trong Cassandra có vai trò tương tự nhau, đều có thể đảm nhận việc đọc ghi dữ liệu. Các nút giao tiếp với nhau bằng giao tiếp Gossip, là một giao thức dùng để cập nhật thông tin về trạng thái của các nút khác đang tham gia vào cụm. Đây là một giao thức liên lạc dạng peer-to-peer, trong đó mỗi nút trao đổi định kỳ thông tin trạng thái của chúng với các nút mà chúng có liên kết. Tiến trình gossip chạy mỗi giây và trao đổi thông tin với nhiều nhất là ba nút khác trong cụm. Các nút trao đổi thông tin về chính chúng và cả thông tin với các nút mà chúng đã trao đổi. Bằng cách này, toàn bộ nút có thể nhanh chóng hiểu được trạng thái của tất cả các nút còn lại trong cụm.

Chart

Description automatically generated with low confidence

Kiến trúc ngang hàng

* Mô hình định hướng cột: Cassandra lưu trữ các cột dựa trên các tên cột, dẫn đến việc cắt rất nhanh. Không giống như các cơ sở dữ liệu truyền thống, trong đó tên cột chỉ bao gồm siêu dữ liệu, trong tên cột Cassandra cũng có thể bao gồm dữ liệu thực tế. Vì vậy, các hàng trong Cassandra có thể chứa số lượng cột khác nhau.
* Khả năng mở rộng đàn hồi: có thể dễ dàng mở rộng hoặc thu nhỏ cụm trong Cassandra mà không làm cụm bị xáo trộn. Có nghĩa là không cần phải khởi động lại cụm khi cần mở rộng hoặc thu nhỏ cụm. Không có thời gian chết hoặc bất kỳ thời gian tạm dừng nào trong quá trình mở rộng quy mô. Do đó, thông lượng đọc và ghi tăng đồng thời mà không có độ trễ.

Map

Description automatically generated

Khả năng chịu lỗi

* Khả năng chịu lỗi (Fault Tolerance): dữ liệu sẽ được sao chép thành nhiều bản trên các server. Nếu 1 server nào đó bị hỏng thì vẫn có thể truy xuất dữ liệu từ các server khác.
* Tính sẵn sàng cao: khi thực hiện tác vụ đọc/ghi, Cassandra có thể thực hiện trên bản sao gần nhất hoặc trên tất cả các bản sao. Điều này phụ thuộc vào thông số ConsitencyLevel do người dùng thiết lập.

## Nhược điểm

* Cassandra không hỗ trợ nhiều cho việc tính toán trên storage, nó không hỗ trợ các hàm sum, group, join, max, min và bất kì hàm nào khác mà developer muốn sử dụng để tính toán dữ liệu khi query.
* Vì là dữ liệu phân tán, dữ liệu được lan truyền trên nhiều máy, nên khi có 1 lỗi trong cơ sở dữ liệu thì lỗi này sẽ lan truyền ra toàn bộ các máy trên hệ thống.

# So sánh Apache Cassandra với các hệ quản trị cơ sở dữ liệu khác

## So sánh với SQL Server

|  |  |  |
| --- | --- | --- |
|  | **Apache Cassandra** | **SQL Server** |
| **Khái niệm** | Cassandra là phần mềm trung gian NoSQL, một cơ sở dữ liệu phân tán kết hợp mô hình dữ liệu của Google Bigtable với thiết kế hệ thống phân tán như bản sao của Amazon Dynamo. | SQL Server là một hệ quản trị dữ liệu quan hệ sử dụng câu lệnh SQL để trao đổi dữ liệu giữa máy cài SQL Server và máy Client. |
| **Điểm quan trọng** | Mở rộng quy mô, tính khả dụng cao | Tính nhất quán |
| **Performance** | Mở rộng quy mô bằng cách sắp xếp các máy chủ cạnh nhau | Chia tỷ lệ hoặc chia nhỏ dữ liệu theo chiều ngang |
| **Query** | Querry với một Key khóa đơn giản | Query bằng SQL |
| **Tính nhất quán** | Linh động, lỏng lẻo | Mạnh mẽ |
| **Data model** | Nhiều mô hình như mô hình định hướng cột, khóa và giá trị thuần túy | Mô hình quan hệ |

## So sánh với MongoDB

* Giống nhau:
* Là cơ sở dữ liệu NoSQL
* Không thay thế cho RDBMS và cũng không phải là cơ sở dữ liệu ACID
* Khác nhau:

|  |  |  |
| --- | --- | --- |
|  | **Apache Cassandra** | **MongoDB** |
| **Mô hình dữ liệu** | Hướng cột | Hướng đối tượng, hướng dữ liệu. Mô hình dữ liệu này có thể dễ dàng thể hiện bất kỳ cấu trúc dữ liệu nào. Dữ liệu có thể có các thuộc tính và có thể được lồng vào nhau ở nhiều cấp độ. |
| **Nút master** | Có nhiều nút chính trong một cụm. Nếu một nút gặp sự cố, nút khác sẽ thế chỗ, cụm sẽ không bị ảnh hưởng. Do đó, cụm luôn luôn khả dụng. | Chỉ có 1 nút master ở 1 cụm. Nút master này kiểm soát một số nút slave. Nếu nút master gặp sự cố, một trong các nút slave sẽ được chọn làm nút master. Quá trình này mất khoảng 10 đến 30 giây. Trong thời gian trì hoãn này, cụm bị ngừng hoạt động và không thể nhận vào bất kỳ dữ liệu hoặc thao tác nào. |
| **Khả năng mở rộng** | Có nhiều nút chính. Dữ liệu sẽ được nhập vào các nút khác từ các nút chính này. Do đó, cụm càng có nhiều nút chính thì khả năng mở rộng sẽ tốt hơn. | Chỉ có một nút master và nó chỉ chấp nhận đầu vào và tất cả các nút còn lại được sử dụng như một đầu ra. Do đó, nếu dữ liệu phải được ghi ở các nút slave thì nó phải đi qua nút master. |
| **Ngôn ngữ truy vấn** | Có ngôn ngữ truy vấn riêng là CQL | JSON |

# Bảng phân công công việc

|  |  |
| --- | --- |
| **Tên** | **Công việc** |
| Đào Thị Ngân Triều | Tìm hiểu tổng quan Cassandra, kiến trúc và mô hình lưu trữ của Cassandra, ngôn ngữ truy vấn CQL |
| Nguyễn Anh Tú | Tìm hiểu điểm mạnh của Cassandra, so sánh Cassandra với MongoDB, demo |
| Nguyễn Thị Kim Yến | Tìm hiểu mô hình lưu trữ dữ liệu và điểm yếu của Cassandra, so sánh Cassandra với SQL Server |

# Tài liệu tham khảo

[1] “Apache Cassandra Documentation”. <https://cassandra.apache.org/_/index.html>.

[2] “Cassandra Performance: The Most Comprehensive Overview You’ll Ever See ”. <https://www.scnsoft.com/blog/cassandra-performance>.

[3] “About Cassandra”. https://kunal14053.medium.com/about-cassandra-1423223945b3.