

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN

---o0o---



BÁO CÁO ĐỒ ÁN LÝ THUYẾT:
TÌM HIỂU HỆ QUẢN TRỊ CSDL
CASSANDRA

MÔN HỌC: HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Nhóm sinh viên: **Nguyễn Quang Phú – 19127507**
 Lâm Hoàng Phúc – 19127512
 Võ Đình Phúc - 19127515

Lớp: **19HTTT1**

TP. HỒ CHÍ MINH – THÁNG 12/2021

MỤC LỤC

1. GIỚI THIỆU CASSANDRA	4
1.1. Khái niệm	4
1.2. Đặc điểm	4
1.2.1. Tính phân tán	4
1.2.2. Tính mềm dẻo	4
1.2.3. Tính sẵn sàng cao	4
1.2.4. Tính chịu lỗi cao	4
1.2.5. Tính hướng cột	4
1.2.6. Hiệu năng cao	4
1.3. Các tính năng hỗ trợ	5
1.3.1. Giao tác	5
1.3.2. Cơ chế ghi theo lô	6
1.3.3. Materialized views	7
1.3.4. Trigger	8
1.3.5. Cơ chế phục hồi dữ liệu	9
1.3.6. Cơ chế bảo mật	9
1.3.7. Một số ưu và nhược điểm của các tính năng	12
2. KIẾN TRÚC TRONG CASSANDRA	14
2.1. Khái niệm	14
2.2. Giao thức truyền thông giữa các node (Gossip)	15
2.3. Phân phối và nhân rộng dữ liệu	16
2.4. Node ảo	17
2.5. Partitioner (phân vùng)	18
3. MÔ HÌNH DỮ LIỆU CASSANDRA	19
3.1. Khái niệm	19
3.2. Keyspace	19
3.3. Column family	20
3.4. Column	21
3.5. Expiring column	22

3.6. Counter column (bộ đếm)	22
3.7. Super column (siêu cột)	23
3.8. Kiểu dữ liệu.....	23
3.8.1. Các kiểu dữ liệu cơ bản.....	23
3.8.2. Các kiểu dữ liệu tập hợp.....	26
3.8.3. Các kiểu dữ liệu tùy biến	27
3.9. Primary key	27
3.10. Primary index	29
3.11. Secondary index.....	29
4. CÁCH TRIỂN KHAI/CÀI ĐẶT HỆ QUẢN TRỊ CASSANDRA	31
4.1. Yêu cầu cài đặt	31
4.2. Cài đặt trên Linux.....	31
4.3. Cài đặt trên Windows	35
4.4. Cài đặt trên MacOS	38
4.5. Triển khai.....	39
5. HỆ THỐNG PHÙ HỢP.....	40
6. ỨNG DỤNG MINH HỌA.....	42
6.1. Mô tả.....	42
6.2. Các hệ chức năng	43
6.3. Thiết kế CSDL.....	44
6.3.1. Mô hình ER.....	44
6.3.2. Quy trình làm việc.....	44
6.3.3. Mô hình thiết kế logic	45
6.3.4. Mô hình thiết kế vật lý	45
6.3.5. Kịch bản demo các tính năng hỗ trợ	46
7. BẢNG PHÂN CÔNG VÀ ĐÁNH GIÁ	47
8. TÀI LIỆU THAM KHẢO.....	48

1. GIỚI THIỆU CASSANDRA

1.1. Khái niệm

Hệ quản trị Cassandra là hệ cơ sở dữ liệu NoSQL được phát triển bởi facebook vào năm 2007, sau đó được tặng cho quỹ Apache vào năm 2010 và trở thành một dự án hàng đầu của Apache. Đây là sự kết hợp của Google Bigtable và Amazon DynamoDB, ngôn ngữ phát triển là Java.

Cassandra là hệ quản trị cơ sở dữ liệu phân tán, với dữ liệu được lưu trên nhiều node ở nhiều máy khác nhau theo mô hình P2P (peer-to-peer), hiệu năng cũng tăng theo số node (tức là số node càng nhiều thì xử lý được càng nhiều request).

1.2. Đặc điểm

1.2.1. Tính phân tán

Khả năng chia dữ liệu thành nhiều phần đặt trên nhiều node khác nhau trong khi người dùng vẫn thấy đây là một khối thống nhất.

1.2.2. Tính mềm dẻo

Hệ thống có thể thêm hoặc bớt số node đi tùy theo số request nhận được, để đảm bảo hệ thống vẫn hoạt động ổn định.

1.2.3. Tính sẵn sàng cao

Hệ thống chia dữ liệu thành nhiều bản và lưu trên nhiều node khác nhau, điều này đảm bảo khả năng đáp ứng ngay lập tức khi người dùng thực hiện thao tác đọc hay ghi bằng cách thực hiện trên bản sao gần nhất hoặc trên tất cả bản ghi phụ (tùy theo consistency level người dùng thiết lập).

1.2.4. Tính chịu lỗi cao

Do dữ liệu được lưu trên nhiều node, nên khi một node xảy ra lỗi hỏng hóc, thì vẫn có thể thực hiện truy vấn trên một node khác.

1.2.5. Tính hướng cột

Không các hệ quản trị cơ sở dữ liệu quan hệ, bạn phải định nghĩa trước các cột trong một bảng. Đối với Cassandra thì bạn không cần làm điều đó bạn có thể thêm cột.

1.2.6. Hiệu năng cao

Các nhà phát triển muốn phát triển Cassandra để tận dụng được khả năng của nhiều máy đa lỗi, và Cassandra đã chứng minh sự đáng tin cậy một cách xuất sắc khi nói đến một bộ dữ liệu lớn.

Do đó, Cassandra được tin tưởng sử dụng trong các hệ thống dữ liệu không lỗi, bên cạnh đó họ còn được đảm bảo về tính an toàn dữ liệu.

1.3. Các tính năng hỗ trợ

1.3.1. Giao tác

Cassandra không sử dụng giao tác ACID với cơ chế rollback hay khóa, tuy nhiên thay vào đó, nó cung cấp các giao tác nguyên tử, độc lập và bền vững với độ nhất quán đồng nhất hoặc có thể thay đổi được, cho phép người dùng quyết độ nhất quán của giao tác mà họ mong muốn mạnh như thế nào.

Là một hệ quản trị cơ sở dữ liệu không quan hệ, Cassandra không hỗ trợ các tính năng khóa hoặc hay join, và vì thế nên chúng cũng không thể hỗ trợ tính nhất quán theo nghĩa của ACID. Ví dụ, khi chúng ta chuyển tiền từ một tài khoản ngân hàng này sang một tài khoản ngân hàng khác, tổng tiền trên các tài khoản sẽ không thay đổi. Cassandra hỗ trợ tính nguyên tử và cô lập trên dòng, tuy nhiên đánh đổi chúng trong giao dịch để có được khả năng sẵn sàng và và hiệu suất ghi cao.

Tuy không hỗ trợ transaction, nhưng Cassandra có hỗ trợ tính năng lightweight transaction (giao tác nhẹ), và nó được sử dụng phổ biến trong thao tác INSERT dữ liệu, và dữ liệu này phải là duy nhất, chẳng hạn như xác định một nhân viên trong công ty.

Các câu lệnh INSERT và UPDATE thường sử dụng IF để hỗ trợ cho lightweight transaction, được viết đến như là Compare and Set. Tuy nhiên, chúng ta không nên sử dụng nó một cách tùy tiện, bởi vì độ trễ của một thao tác có thể tăng lên 4 lần do chuyển đi khứ hồi giữa các CAS.

Cassandra có hỗ trợ các điều kiện bất bình đẳng trong lightweight transaction như: >, <, >=, <=, != và IN, bạn có thể sử dụng những toán tử này trong mệnh đề WHERE.

Chúng ta cần lưu ý việc sử dụng IF NOT EXIST trong một lần INSERT, nhân thời gian sẽ được tạo bởi lightweight transaction vì vậy việc sử dụng TIMESTAMP là bị cấm.

Ví dụ: Chúng ta chèn một nhân viên vào bảng nhân viên:

```
INSERT INTO keyspace.employee (emp_ID, emp_name,
emp_exp) VALUE (1, 'Vo Dinh Phuc', 2)
IF NOT EXISTS;
```

Câu lệnh trên sẽ thực hiện INSERT nhân viên này vào nếu trong bảng chưa có nhân viên nào trùng với nhân viên này. Nếu có nhân viên trùng, bảng sẽ UPDATE lại nhân viên đó với dữ liệu vừa INSERT.

1.3.2. Cơ chế ghi theo lô

Nhiều câu lệnh INSERT, UPDATE, DELETE có thể thực thi trong duy nhất một câu lệnh bằng cách nhóm chúng lại trong một câu lệnh BATCH.

Cú pháp:

```
batch_statement ::= BEGIN [ UNLOGGED | COUNTER ]
BATCH [ USING update_parameter (AND update_parameter)* ]
      modification_statement ( ';' modification_statement )*
APPLY BATCH
modification_statement ::= insert_statement | update_statement |
delete_statement
```

Ví dụ:

```
BEGIN BATCH
  INSERT INTO cart_items (item_id, cart_id, item_name)
  VALUES (1, 1, 'Book');
  UPDATE cart SET quantily = quantily +1
  WHERE cart_id = 1;
APPLY BATCH;
```

Câu lệnh BATCH nhóm nhiều câu lệnh thay đổi dữ liệu lại với nhau (thêm/xóa/sửa) lại với nhau thành một câu lệnh đơn. Một số mục đích chính của câu lệnh này là:

- Nó giúp lưu lại đường đi vòng quanh giữa máy chủ và máy khách (và đôi khi giữa nhiều server và bản sao) khi thực hiện nhiều update.
- Các cập nhật trong một BATCH thuộc một partition được cho trước được thực hiện độc lập.
- Mặc định, tất cả các thao tác trong một BATCH được thực hiện dưới dạng log để bảo đảm rằng tất cả được hoàn thành cuối cùng hoặc không có thao tác nào được thực hiện.

Chú ý rằng:

- Một câu lệnh BATCH chỉ có thể chứa các câu lệnh INSERT, UPDATE, DELETE.
- Batch không hoàn toàn đóng vai trò như một transaction trong SQL.
- Nếu nhãn thời gian không được xác định cho mỗi thao tác, sau đó tất cả thao tác sẽ được ấn định cùng một nhãn thời gian.
- Một câu lệnh LOGGED BATCH trong một phân vùng đơn sẽ được chuyển thành một UNLOGGED BATCH để tối ưu hóa.

UNLOGGED batch:

- Mặc định, Cassandra sử dụng log batch để đảm bảo tất cả thao tác trong một batch có thể được hoàn thành cùng nhau hoặc không có cái nào được hoàn thành.
- Tuy nhiên, đối với một LOGGED BATCH kéo dài trong nhiều phân vùng thì sẽ gây vấn đề về hiệu suất, nếu bạn muốn tránh vấn đề này thì bạn có thể chỉ định cho câu lệnh BATCH thành UNLOGGED thay vì mặc định, và khi đó một BATCH bị thất bại có thể chỉ được áp dụng một phần.

1.3.3. Materialized views

Cú pháp:

```
create_materialized_view_statement ::= CREATE MATERIALIZED  
VIEW [ IF NOT EXISTS ] view_name  
AS select_statement  
PRIMARY KEY (primary_key)  
WITH table_options
```

Ví dụ:

```
CREATE MATERIALIZED VIEW items_by_name  
AS SELECT * FROM items_by_id  
PRIMARY KEY (item_name,item_id)
```

Câu lệnh trên giúp chúng ta tạo ra một materialized view. Mỗi view như vậy là một tập hợp các dòng tương ứng với các dòng được chọn trong table được chỉ định trong câu lệnh trên. Một materialized view không thể được cập nhật trực tiếp thay vào đó nó được cập nhật khi bảng được tạo view thực hiện cập nhật.

Nếu tạo một view đã tồn tại sẽ trả về lỗi vì vậy để tránh việc này thì chúng ta dùng IF NOT EXISTS.

Tạo một materialized view có 3 phần chính:

- Câu lệnh select để chọn ra những cột trong view.
- Khóa chính, khóa chính này cần chứa tất cả các thuộc tính của khóa chính trong bảng cơ sở, và nó chỉ được chỉ định thêm một thuộc tính khác không phải thuộc tính trong bảng cơ sở.
- Option cho view.

1.3.4. Trigger

Có thể tạo trigger trong Cassandra bằng câu lệnh CREATE TRIGGER

Cú pháp:

```
create_trigger_statement ::= CREATE TRIGGER [ IF NOT EXISTS  
] trigger_name  
ON table_name  
USING string
```


Ví dụ:

```
CREATE TRIGGER myTrigger ON myTable  
USING 'org.apache.Cassandra.triggers.InvertedIndex';
```

Có thể xóa trigger bằng câu lệnh DROP TRIGGER:

```
drop_trigger_statement ::= DROP TRIGGER [ IF EXISTS ]  
trigger_name ON table_name
```

Ví dụ:

```
DROP TRIGGER myTrigger ON myTable;
```

1.3.5. Cơ chế phục hồi dữ liệu

Cassandra sao lưu dữ liệu bằng cách tạo ra các bản sao lưu nhanh của tất cả dữ liệu trên đĩa (SSTable files) được lưu trong thư mục dữ liệu. Bạn có thể, tạo một bản sao lưu nhanh của tất cả keyspace, một keyspace hoặc trên một bảng đơn trong khi hệ thống đang trực tuyến.

Việc sử dụng công ssh song song, bạn có thể tạo một bản ghi nhanh trên toàn bộ cluster. Điều này cung cấp một bản sao lưu nhất quán cuối cùng. Mặc dù không có node nào được đảm bảo được nhất quán với các node sao của nó tại thời điểm nó được tạo bản sao, một bản sao được khôi phục sẽ tiếp tục sự nhất quán bằng cách sử dụng cơ chế nhất quán được tích hợp sẵn trong Cassandra.

Sau khi snapshot được thực hiện, bạn có thể kích hoạt bản sao gai tăng trong mỗi node để sao lưu dữ liệu đã thay đổi kể từ lần cuối tạo bản sao: mỗi lần bản ghi nhớ được chuyển vào đĩa và SSTable được tạo, một đường liên kết cứng được sao chép hoặc sao lưu vào trong thư mục con của thư mục dữ liệu. Các bản ghi được nén sẽ tạo những đường liên kết cứng bởi vì snapshot_before_compaction tạo ra một tập hợp những liên kết cứng trước khi mỗi bản nén có thể được sử dụng để tái tạo bất kỳ SSTable nén nào.

1.3.6. Cơ chế bảo mật

Cassandra cung cấp các tính năng bảo mật sau:

- Xác thực dựa trên rolename và mật khẩu: Việc xác thực Cassandra là dựa trên vai trò và lưu trữ nội bộ bên trong các

bảng trong hệ thống Cassandra. Người quản trị có thể tạo, alter, xóa hoặc liệt kê ra các vai trò bằng cách sử dụng lệnh CQL, với mật khẩu tương ứng. Các vai trò có thể được tạo với các phân quyền superuser, non-superuser và login. Chúng thực nội bộ được sử dụng để truy cập keyspace và table trong Cassandra, bằng cách sử dụng cqlsh và Devcenter để chúng thực kết nối với cụm trong Cassandra và sstableloader để tải SSTables.

- Ủy quyền dựa trên quản lý quyền đối tượng: ủy quyền cung cấp các quyền truy cập đến cụm Cassandra dựa trên chứng thực vai trò. Người ủy quyền có thể cấp quyền cho truy cập đến cả database hoặc hạn chế một số quyền riêng biệt đến một số bảng nhất định. Các vai trò có thể cung cấp ủy quyền để ủy quyền cho các vai trò khác. Các vai trò có thể được cung cấp từ các vai trò khác. Câu lệnh CQL GRANT và REVOKE được sử dụng để quản lý ủy quyền.

Đăng nhập trong Cassandra

Cú pháp:

```
# /CassandraPath/bin/cqlsh <localhost or server Ip> -u Cassandra -p  
Cassandra
```

Cassandra: username và password mặc định

Tạo User

Cú pháp:

```
CREATE USER 'user_name' WITH PASSWORD 'password'  
NOSUPERUSER/SUPERUSER;
```

Giải thích cú pháp	Ý nghĩa
CREATE USER	Tạo user
username/password	Nhập username và password
NOSUPERUSER/SUPERUSER	User thông thường hay là admin

Trường hợp gặp lỗi :

InvalidRequest: Error from server: code=2200 [Invalid query] message="remoteuser already exists"

Tức là user muốn tạo đã tồn tại, chúng ta có thể thay thế bằng lệnh:

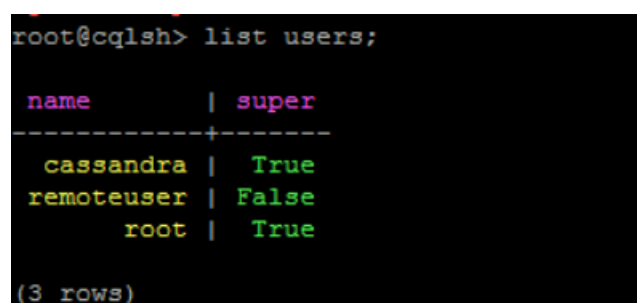
```
CREATE USER IF NOT EXISTS user_name WITH PASSWORD 'password';
```

Ví dụ:

```
CREATE USER IF NOT EXISTS remoteuser WITH PASSWORD 'Admin@123';
```

Kiểm tra User sau khi tạo

```
Cassandra@cqlsh> list users;
```



The screenshot shows the output of the 'list users;' command in CQLSH. It displays a table with two columns: 'name' and 'super'. The table contains three rows: 'cassandra' with 'True', 'remoteuser' with 'False', and 'root' with 'True'. Below the table, it indicates '(3 rows)'.

name	super
cassandra	True
remoteuser	False
root	True

(3 rows)

Cấp quyền cho User

Cú pháp:

```
GRANT permission_name ON resource TO user_name
```

Trong đó, permission_name bao gồm:

Permission	CQL Statement
ALL	All statements
ALTER	ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX
AUTHORIZE	GRANT, REVOKE
CREATE	CREATE KEYSPACE, CREATE TABLE
DROP	DROP KEYSPACE, DROP TABLE
MODIFY	INSERT, DELETE, UPDATE, TRUNCATE

Permission	CQL Statement
SELECT	SELECT

Resource bao gồm:

- ALL KEYSPACES
- KEYSPACE keyspace_name
- TABLE keyspace_name.table

Ví dụ:

```
GRANT ALL ON ALL KEYSPACES TO remoteuser;
```

Kiểm tra cấp quyền cho remoteuser:

```
root@cqlsh> list ALL ON ALL KEYSPACES;
```

role	username	resource	permission
remoteuser	remoteuser	<all keyspaces>	CREATE
remoteuser	remoteuser	<all keyspaces>	ALTER
remoteuser	remoteuser	<all keyspaces>	DROP
remoteuser	remoteuser	<all keyspaces>	SELECT
remoteuser	remoteuser	<all keyspaces>	MODIFY
remoteuser	remoteuser	<all keyspaces>	AUTHORIZE

1.3.7. Một số ưu và nhược điểm của các tính năng

Tính năng	Ưu điểm	Khuyết điểm
Lightweight trans	Đôi khi việc insert hay update một dòng dữ liệu cần duy nhất, điều này yêu cầu đọc trước ghi. Việc đọc trước khi ghi có ý nghĩa về mặt hiệu suất và loại vấn đề này được giải quyết bằng lightweight trans, vì lightweight transaction hỗ trợ mệnh đề if trong câu insert và update.	Bạn không thể sử dụng nhiều điều kiện để sửa đổi nhiều phân vùng. Không thể sử dụng LWT cho bảng sử dụng counter. Khi sử dụng timestamp do người dùng cung cấp, bạn phải đảm bảo rằng timestamp được gán bởi người điều phối giao dịch, nếu không sẽ không thể đảm bảo tính nhất quán.
Cơ chế ghi theo lô	Batch statement hỗ trợ nhóm nhiều câu lệnh sửa đổi vào một câu lệnh đơn, điều này giúp cho tất cả các hoạt	Batch không hoàn toàn giống transaction trong sql. Nếu một nhãn thời gian không được xác định cho mỗi thao tác thì tất cả thao tác sẽ có sẽ có chung một nhãn thời gian (được

	động được đảm bảo hoàn thành đến cuối cùng.	<p>tạo tự động hoặc được cung cấp ở batch level). Bởi vì thủ tục giải quyết xung đột trong trường hợp ràng buộc nhân thời gian, các thao tác có thể được thực thi khác thứ tự được đề cập trong batch statement cho nên để ràng buộc thứ tự cho một thao tác cụ thể bạn phải chỉ định nhân thời gian cho mỗi giao tác. Batch statement chỉ chứa câu lệnh update insert delete.</p> <p>Mặc định Cassandra sử dụng batch log để đảm bảo tất cả giao tác được hoàn thành một lúc. Điều này sẽ dẫn đến các lỗi về hiệu suất.</p> <p>Sẽ gây ra lỗi về hiệu suất nếu một câu lệnh batch trải dài trên nhiều partition.</p>
Material view	Rất thích hợp cho tổ hợp dữ liệu lớn.	<p>Không thể update trực tiếp mà phải update qua bản gốc.</p> <p>Khi tạo một material view đã tồn tại sẽ trả về lỗi nếu không sử dụng if not exists.</p> <p>Khóa chính của material view phải bao gồm tất cả khóa chính của bảng cơ sở và chỉ được thêm nhiều nhất một column vào khóa chính.</p>
Trigger	Giúp phát hiện ra các sửa đổi trên các dữ liệu như SET, COLLECTION, ... và dữ liệu nguyên thủy	Logic của trigger không được viết trực tiếp CQL mà được viết bằng Java ở bên ngoài database.
Cơ chế phục hồi dữ liệu	<p>Lưu dữ liệu lâu dài.</p> <p>Có thể khôi phục dữ liệu nếu dữ liệu trong bảng bị mất do lỗi node, lỗi partition hoặc mạng.</p>	<p>Cassandra cần không gian lớn để lưu trữ những bản sao lưu của mình.</p> <p>Cơ chế đọc sửa chữa trong Cassandra có thể xung đột với các bảng backup.</p>
Secondary Index	Second index thường được sử dụng để truy vấn một	Second index khó sử dụng và thường dẫn đến nhiều lỗi performance. Bảng

	bảng với một cột không thường xuyên được truy vấn.	index được lưu trong mỗi node trong một cluster, nên trong một câu truy vấn bao gồm secondary index có thể nhanh chóng dẫn đến lỗi hiệu suất nghiêm trọng nếu nhiều node được truy vấn. Nó chỉ nên được sử dụng cho các cột có ít dữ liệu.
--	--	---

2. KIẾN TRÚC TRONG CASSANDRA

2.1. Khái niệm

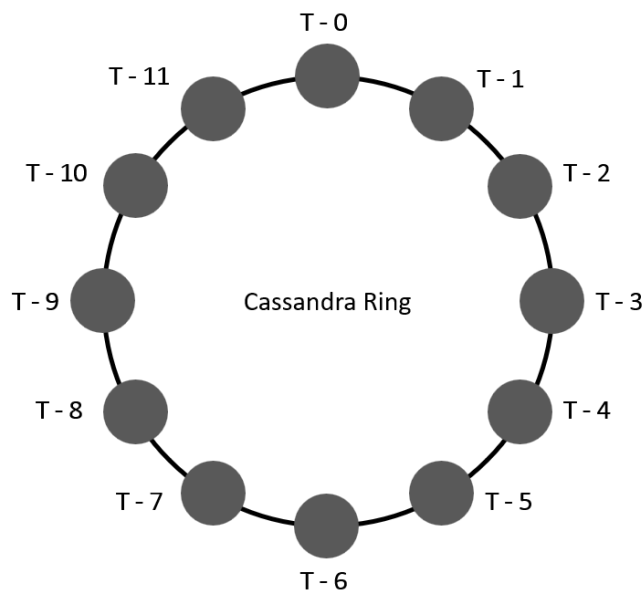
Cassandra thiết kế hệ thống phân tán trên hàng ngàn máy chủ mà không có bất kỳ điểm chết nào, như đã nói ở phần khái niệm thì Cassandra thiết kế theo mô hình ngang hàng (peer-to-peer), dữ liệu giữa các node trong một cụm được phân phối cho nhau, Mỗi nút thường xuyên trao đổi thông tin trạng thái về chính nó và các nút khác trên toàn cụm bằng cách sử dụng giao thức truyền thông gossip ngang hàng chính vì vậy:

- Tất cả các node trong một cụm có vai trò như nhau, mỗi node trong cụm độc lập với nhau và kết nối với các node khác.
- Mỗi node trong cụm đều có thể thực hiện đọc ghi đối với tất cả dữ liệu được lưu ở bất kỳ đâu trong cụm.
- Chính vì các node độc lập và có vai trò như nhau nên khi một node trong cụm gặp sự cố, thì thao tác đọc ghi có thể được thực hiện bởi các node khác.

Cassandra lưu trữ dữ liệu hàng được phân vùng, các hàng được tổ chức bên trong các bảng, với một khóa chính bắt buộc. Kiến trúc của Cassandra cho phép bất kỳ authorized user nào cũng có thể truy cập bất kỳ vào một node bất kỳ trong một data center bất kỳ, sử dụng CQL. CQL khá giống với SQL và được sử dụng trên bảng.

Cassandra sử dụng cơ chế băm nhất quán phân tán (Distributed consistent hashing) để tổ chức các node trong cụm hình tròn, và dữ liệu được phân tán theo hình tròn này. Mỗi vòng tròn được xem là một Data

center, và mỗi node trong cụm sẽ được phân bố trên một vị trí của vòng tròn gọi là ring:



2.2. Giao thức truyền thông giữa các node (Gossip)

Gossip là một giao thức giao tiếp ngang hàng, các node trao đổi thông với nhau định kỳ về trạng thái của chúng và các node xung quanh mà chúng liên kết, và quá trình này chạy liên tục mỗi giây, và trao đổi thông tin với tối đa 3 node trong 1 cụm. Các node trao đổi thông tin của chúng và thông tin của những node mà chúng đã trao đổi trước đó, từ đó tất cả các node sẽ có được thông tin của tất cả các node khác một cách nhanh chóng trong một cụm. Mỗi Gossip có một phiên bản để liên kết với chính nó, chính vì vậy trong quá trình trao đổi, thông cũ hơn sẽ được ghi đè bởi trạng thái mới nhất.

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 khi một node sẽ phải nhớ tất cả các node mà chúng từng giao tiếp gossip kể cả khi khởi động lại và seed node có nhiệm vụ cập nhập một node mới khi tham gia vào một cụm, điều này có nghĩa nếu một node

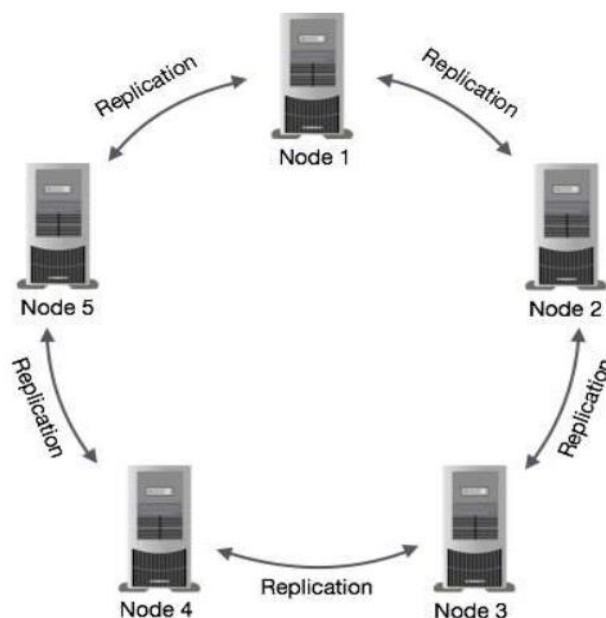
mới nếu một node nào đó muốn tham gia vào cụm thì chúng phải liên hệ với seed node để được cập nhật tất cả các node trong cụm.

Mỗi data center nên chứa ít nhất một seed node, vì nếu không có seed node khi một node tham gia vào cụm, chúng sẽ phải liên hệ với một seed node khác từ data center khác, tuy nhiên cũng không nên đặt tất cả các node thành seed node vì nó sẽ gây giảm hiệu năng của gossip và gây khó duy trì, việc tối ưu seed node không quan trọng như khuyến khích chỉ nên sử dụng danh sách seed node 3 đến 4 node trên một data center.

2.3. Phân phối và nhân rộng dữ liệu

Trong Cassandra, phân phối và nhân rộng dữ liệu luôn đi cùng nhau, dữ liệu được lưu trữ trên bảng và được xác định bởi một khóa chính, và điều này cũng xác định rằng dữ liệu được lưu trên node nào. Replicas là bản sao của các hàng, khi dữ liệu được ghi lần đầu tiên thì chúng cũng được xem là một bản sao.

Mỗi node trong một cụm của Cassandra hoạt động như một bản sao cho một phần dữ liệu nào đó, và khi phát hiện ra một giá trị bị lỗi thời (cũ) thì Cassandra sẽ thực hiện đọc và trả về giá trị gần đây nhất, sau đó Cassandra thực hiện repair read để cập nhật lại giá trị cũ. Việc giao tiếp và phát hiện lỗi sử dụng giao thức Gossip được đề cập ở trên. Sau đây là hình ảnh minh họa cho việc Cassandra sử dụng bản sao để đảm bảo không có bất kỳ điểm chết:

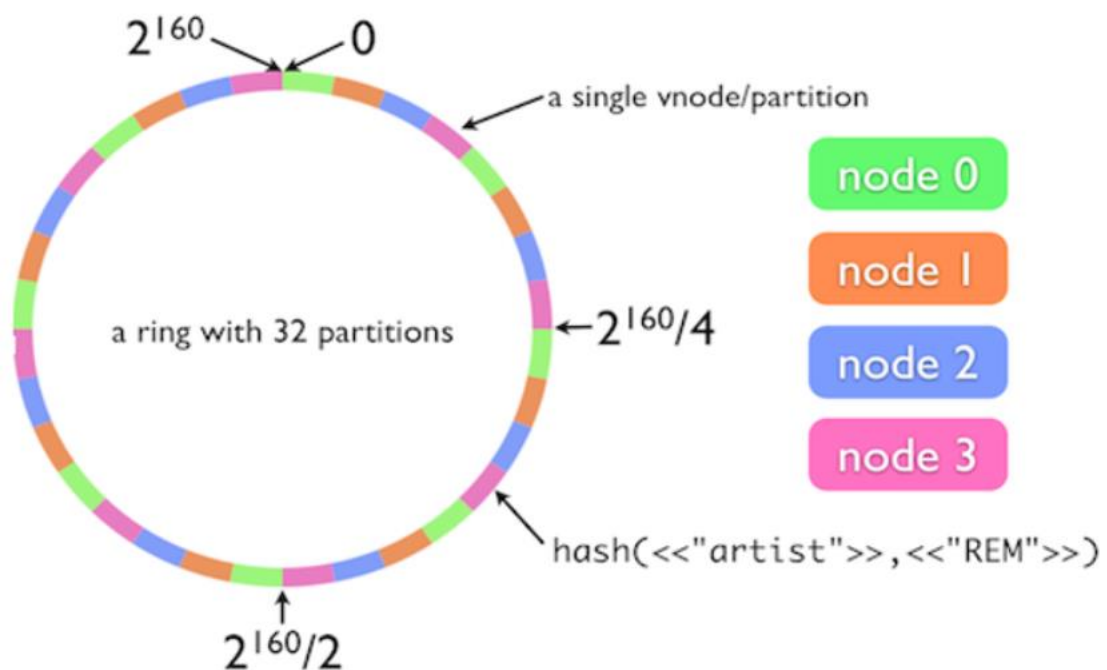


Hai chiến lược sao lưu có sẵn:

- SimpleStrategy: sử dụng duy nhất cho một datacenter đơn và một giá. Nếu bạn muốn sử dụng nhiều hơn một datacenter sử dụng NetworkTopologyStrategy.
- NetworkTopologyStrategy: được đề xuất cao cho hầu hết các triển khai bởi vì nó dễ hơn nhiều để mở rộng nhiều datacenter khi có yêu cầu mở rộng trong tương lai.

2.4. Node ảo

Node ảo giúp giải quyết vấn đề tái cân bằng cụm trong Cassandra. Node ảo giống như một thành phần của vòng trong hệ thống, tuy nhiên thực tế chúng chỉ là ánh xạ của một node vật lý trên vòng, và khi dữ liệu đi vào node ảo thì chúng sẽ được chuyển lại lưu vào vùng nhớ của node vật lý của node ảo này. Mỗi node vật lý khi được thêm vào vòng sẽ mặc định được gán vào một vị trí trong vòng, cùng với đó là một số lượng node ảo nhất định. Mặc định Cassandra sẽ gán 256 node ảo cho một node vật lý khi tham gia vào cụm.



Hình trên thể hiện rằng, trong vòng có 4 node vật lý, và mỗi node được gán thêm 7 node ảo tạo thành một vòng với 32 phân vùng key. Khi phân tán đều tất cả các node ảo ra khắp vòng, số lượng node tăng lên đồng nghĩa với việc phân vùng key bé lại, việc này mang một ý nghĩa rất lớn, khi phân vùng key bé lại khiến cho các node sát nhau hơn, đưa hệ thống gần hơn với việc tất cả dữ liệu được phân bổ đều trên tất cả các node, xác

suất dữ liệu được đưa vào là cân bằng nhau khi trên một vùng key nhỏ ta có được đầy đủ các node ảo hoặc node vật lý. Và trường hợp hoàn hảo nhất là tất cả các node vật lý đều có đại diện trên cả vòng.

2.5. Partitioner (phân vùng)

Partitioner xác định cách dữ liệu được phân phối vào các node trên cụm (bao gồm cả các bản sao). Về cơ bản, một partitioner là một hàm cho việc lấy token đại diện cho một hàng từ khóa phân vùng của nó, thông thường là hashing. Mỗi dòng dữ liệu sau đó được phân phối vào cụm bằng giá trị của token.

Cả Murmur3Partitioner và RandomPartitioner đều sử dụng để gán các phần dữ liệu bằng nhau vào mỗi node và phân phối dữ liệu từ tất cả các bảng trong vòng hoặc nhóm khác, như keyspace. Điều này đúng ngay cả khi bảng sử dụng những khóa phân vùng khác nhau, như username hoặc nhãn thời gian. Hơn nữa, yêu cầu đọc và ghi vào cụm cũng được phân phối đồng đều và việc cân bằng tải cũng được đơn giản bởi vì mỗi phần của phạm vi băm nhận được số lượng hàng trung bình bằng nhau.

Điểm khác nhau chính của hai phân vùng này là cách chúng tạo ra giá trị băm token. RandomPartitioner sử dụng băm mật mã mất nhiều thời gian hơn Murmur3Partitioner.

Cassandra không thực sự cần một băm mật mã, nên việc sử dụng Murmur3Partitioner sẽ giúp cải thiện hiệu suất lên 3 đến 5 lần.

Cassandra đưa ra các phân vùng sau có thể được cài đặt trong file Cassandra.yaml:

- Murmur3Partitioner (Mặc định): phân phối dữ liệu đồng nhất trên cụm dựa trên giá trị băm MurmurHash.
- RandomPartitioner: Phân phối dữ liệu đồng nhất vào cụm dựa trên giá trị băm MD5.
- ByteOrderPartitioner: giữ thứ tự phân phối dữ liệu theo bằng byte khóa.

Murmur3Partitioner là chiến lược phân vùng mặc định từ phiên bản Cassandra 1.2 và là lựa chọn hợp lý cho hầu hết các trường hợp. Tuy nhiên, các phân vùng không tương thích và dữ liệu được chia với một phân vùng không dễ để chuyển sang một kỹ thuật phân vùng khác.

3. MÔ HÌNH DỮ LIỆU CASSANDRA

3.1. Khái niệm

Trong Cassandra, mô hình dữ liệu (data model) được thiết kế dựa trên mục đích truy vấn dữ liệu (query-driven). Các mẫu dữ liệu và các truy vấn của ứng dụng được xác định cấu trúc và tổ chức dữ liệu mà sau đó được dùng để thiết kế các bảng dữ liệu.

Không giống như mô hình dữ liệu quan hệ thường lấy dữ liệu bằng cách kết nhiều bảng, điều này không hỗ trợ bởi Cassandra nên các cột cần được nhóm lại với nhau trong một bảng. Mặc dù mỗi câu truy vấn chỉ hoạt động trên một bảng nhưng dữ liệu có thể được nhân bản ở nhiều bảng khác hay còn gọi là quá trình phi chuẩn (denormalization). Nhân bản dữ liệu (data duplication) và tần suất ghi cao sẽ làm tăng hiệu suất (performance) của việc đọc dữ liệu.

Mô hình dữ liệu trong Cassandra bao gồm các keyspace ở tầng cao nhất.

Mô hình dữ liệu Cassandra tuân theo quy tắc hệ thống cột (the column family).

- Column Family: Tương tự với RDBMS, column family là một bảng, nơi chứa các cột dữ liệu. Nó là một tập hợp dữ liệu chứa các cặp “key – value”. Trong đó “key” được ánh xạ đến một giá trị gồm tập hợp các cột, mỗi cặp “key – value” là một hàng.
- Cột là một tập hợp dữ liệu gồm tên cột, giá trị, và mốc thời gian.
- Siêu cột một cột đặc biệt lưu trữ bản đồ (map) của các cột. Nó giúp lưu trữ cũng như truy xuất dữ liệu dễ dàng và nhanh chóng hơn.

3.2. Keyspace

Keyspace dùng để nhóm các column family với nhau. Các keyspace là các vật chứa của dữ liệu, giống như schema và database trong mô hình dữ liệu quan hệ. Một keyspace bao gồm nhiều bảng. Một số tính năng của keyspace:

- Cần được định nghĩa trước khi tạo bảng (không có keyspace mặc định).
- Một keyspace có thể bao gồm nhiều bảng, một bảng chỉ thuộc một keyspace.

- Bản sao được chỉ định ở tầng keyspace.
- Cần chỉ định hệ số nhân bản (số lượng bản sao) trong quá trình tạo keyspace. Tuy nhiên, hệ số nhân bản có thể được điều chỉnh sau.

Trong Cassandra, thuộc tính cơ bản cái mà bạn có thể đặt làm keyspace bao gồm:

- Replication factor: replication factor quy định số lượng các nút mà thực thi như là bản sao của một hàng trong dữ liệu. Nếu có 3 replication factor và 3 node trong ring sẽ có các bản sao của mỗi hàng. Replication factor về bản chất cho phép chúng ta quyết định chi phí muốn bỏ ra để đạt được tính nhất quán nhiều hơn. Có nghĩa là mức độ nhất quán trong việc đọc và ghi dữ liệu là dựa trên Replication factor.
- Replica placement strategy: Replica placement chỉ ra cách mà các bảng ghi sẽ được sắp xếp trên ring.
- Column families: Một keyspace là một nơi một hoặc nhiều column family. Một column family có thể coi như là một bảng ở trong mô hình quan hệ, và bao hàm cho tập hợp các dòng. Mỗi dòng bao hàm một cột.

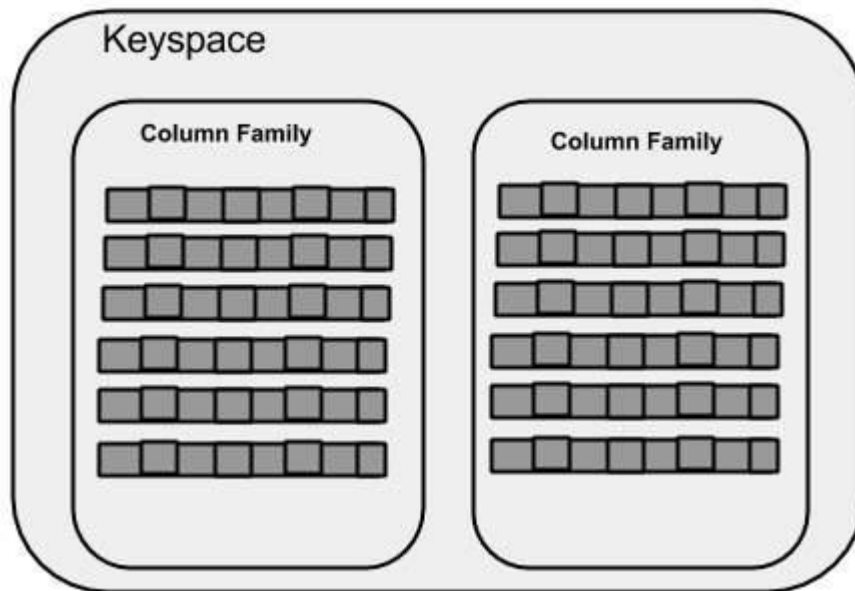
Ví dụ:

```
CREATE KEYSPACE DemoDB
WITH REPLICATION = {
    'class' : 'SimpleStrategy',
    'replication_factor': 5
};
USE DemoDB;
```

Giải thích: Tạo một keyspace có tên là DemoDB, nhân bản thành 5 bản sao.

3.3. Column family

“Gia đình cột” trong Cassandra là một tập hợp các dòng, bao gồm các cột có thứ tự. Thể hiện cấu trúc của dữ liệu được lưu trữ. Các column family sẽ được chứa trong Keyspace. Mỗi Keyspace sẽ có ít nhất một Column family.



Cột là đơn vị cơ bản (basic unit) của cấu trúc dữ liệu trong Cassandra. Có 3 giá trị được lưu trữ trong một cột: Khóa hoặc tên cột, nhãn thời gian và giá trị. Có thể thêm hoặc xóa cột tùy theo yêu cầu của người dùng. Trong khi đó các column family không thể thay đổi do được xác định trước đó. Mỗi hàng trong một column family không yêu cầu phải có cùng số cột.

Đặc biệt, trong Cassandra không có khóa ngoại, không có phép kết bảng.

Mỗi column family nên được thiết kế để chứa một kiểu dữ liệu đơn. Có 2 loại column family trong Cassandra là static và dynamic:

- Static column family dùng cho một tập tĩnh. Ví dụ column family lưu trữ dữ liệu người dùng có một số thông tin như tên người dùng, address, email, phone,
- Dynamic column family thể hiện lợi ích của khả năng của Cassandra. Thay vì việc định nghĩa metadata cho những cột đơn. Dynamic column family định nghĩa thông tin cho tên cột và giá trị cột (comparator và validator). Nhưng tên cột và giá trị của nó được thay đổi bởi ứng dụng khi cột được thêm vào.

3.4. Column

Cột là phần tử nhỏ nhất trong Cassandra và là một tập hợp dữ liệu gồm tên cột, giá trị, và mốc thời gian.

Cột phải có tên và tên là nhãn tĩnh hoặc động tùy thuộc bởi ứng dụng. Có thể đặt index lên tên cột nhưng việc đặt index không hỗ trợ câu truy vấn

truy xuất dữ liệu có thứ tự như dữ liệu chuỗi thời gian. Trường hợp này đặt secondary lên timestamp column sẽ không hiệu quả bởi không thể sắp xếp theo thứ tự với secondary index.

Cassandra dùng timestamp để quyết định thời điểm update cột sau cùng. Nếu một người dùng update lên cùng một cột đồng thời, update gần nhất sẽ được sử dụng.

Ví dụ:

```
{
    name: "country",
    value: "vietnam",
    timestamp: 123456
}
```

3.5. Expiring column

Một cột sẽ có thời gian hết hạn của nó (TTL). Khi insert một cột, người dùng có thể đặc tả một TTL (tính bằng giây). Những cột quá hạn sẽ đánh dấu bị xóa. Để gia hạn thêm thời gian, ta chỉ cần insert lại cột đó với TTL mới. Trong Cassandra insert và update là một.

3.6. Counter column (bộ đếm)

Counter column (bộ đếm) là một cột đặc biệt được sử dụng để lưu trữ một số nguyên được thay đổi theo thời gian khi có một sự kiện hoặc một tiến trình xảy ra. Bộ đếm hữu ích cho nhiều mô hình dữ liệu, chẳng hạn như:

- Theo dõi số lượt xem trang web nhận được trên trang web của công ty.
- Theo dõi số lượng trò chơi đã chơi trực tuyến hoặc số lượng người chơi đã tham gia một trò chơi trực tuyến.

Một số lưu ý khi sử dụng bộ đếm:

- Không thể lập chỉ mục hoặc xóa cột bộ đếm

- Để tải dữ liệu vào cột bộ đếm hoặc để tăng hoặc giảm giá trị của bộ đếm, hãy sử dụng lệnh UPDATE. Không sử dụng TIMESTAMP hoặc TTL khi cập nhật cột bộ đếm.
- Để tạo bảng có một hoặc nhiều cột bộ đếm, hãy sử dụng CREATE TABLE để xác định bộ đếm và cột không bộ đếm.

3.7. Super column (siêu cột)

Là một cột đặc biệt lưu trữ bản đồ (map) của các cột. Hiểu đơn giản là column có value là một danh sách có column và super column không có timestamp. Nó giúp lưu trữ cũng như truy xuất dữ liệu dễ dàng và nhanh chóng hơn.

Ví dụ:

```
{
  name: "address",
  value: {
    street:
      {
        name: "street",
        value: "227 Nguyen Van Cu",
        timestamp: 123456
      }
    city:
      {
        name: "city",
        value: "Ho Chi Minh",
        timestamp: 123456
      }
  }
}
```

3.8. Kiểu dữ liệu

3.8.1. Các kiểu dữ liệu cơ bản

- **ascii**: Biểu diễn cho một chuỗi ký tự ASCII. Việc chèn bất kỳ ký tự không phải ASCII vào một cột của kiểu dữ liệu này sẽ cho ra một lỗi.

- **bigint**: Đại diện cho số nguyên có dấu dài 64-bit. Được sử dụng để lưu trữ những con số dài. Nó nên được sử dụng chỉ khi chúng ta chắc chắn cần các số dài vì nó chiếm nhiều không gian bộ nhớ hơn so với int.
- **blob**: dùng để lưu trữ các byte tùy ý. Nó ở hệ thập lục phân, và bất kỳ dữ liệu nào không cần xác nhận có thể được lưu trữ trong trường này.
- **boolean**: Lưu trữ true hoặc false.
- **counter**: Đại diện cho một số nguyên dài 64-bit, nhưng giá trị của cột này không thể thiết lập. Chỉ có hai hoạt động trên cột này, tăng và giảm. Trong một bảng với một cột counter, chỉ có các kiểu counter và khóa chính là được phép. Không có câu lệnh INSERT được phép trong một bảng với các column counter; chỉ UPDATE có thể được sử dụng.

Ví dụ:

```
CREATE TABLE website_tracker (
    id int PRIMARY KEY,
    url text,
    visitor_count counter
);
UPDATE website_tracker
SET visitor_count = visitor_count + 1 WHERE id = 1;
SELECT * FROM website_tracker;
id | url | count
----+-----+-----
1 | a.com | 1
(1 rows)
```

- **date**: Đại diện cho một giá trị ngày mà không có một giá trị giờ. Cassandra mã hóa giống như một giá trị số nguyên kể từ epoch.

Ngày có thể được biểu diễn như là chuỗi trong định dạng yyyy-mm-dd.

- **decimal**: Đại diện cho một biến-giá trị thập phân chính xác. Dùng tốt nhất trong việc lưu trữ tiền tệ hoặc các giá trị tài chính.
- **double**: Lưu trữ một giá trị dấu chấm động dài 64-bit.
- **float**: Lưu trữ một giá trị dấu chấm động 32-bit.
- **inet**: Biểu diễn cho một chuỗi địa chỉ IP trong định dạng của IPv4 hoặc IPv6.
- **int**: Biểu diễn cho một số nguyên có dấu dài 32-bit. Sử dụng chủ yếu để lưu trữ các giá trị số nguyên.
- **smallint**: Biểu diễn cho một số nguyên 2 byte (16-bit). Có thể được ưu tiên hơn so với kiểu int để lưu trữ các giá trị số nguyên nhỏ để tiết kiệm không gian lưu trữ.
- **text**: Biểu diễn cho một chuỗi mã hoá UTF-8. Nên được sử dụng khi chúng ta muốn lưu trữ các ký tự không phải mã ASCII.
- **time**: Biểu diễn cho một giá trị thời gian. Đại diện như một chuỗi ở dạng 01:02:03.123 và lưu trữ số nguyên có dấu dài 64-bit đại diện cho số nano giây trôi qua kể từ nửa đêm.
- **timestamp**: Lưu trữ cả thành phần ngày và giờ với độ chính xác milli giây. Có thể được biểu diễn dưới dạng văn bản ở định dạng 2016-12-01 01:02:03.123.
- **tinyint**: Biểu diễn cho một số nguyên 1 byte (8 bit). Có thể được ưu tiên hơn so với kiểu int hoặc smallint để lưu trữ các giá trị số nguyên nhỏ để tiết kiệm không gian lưu trữ.
- **timeuuid**: Lưu trữ phiên bản 1 UUID.
- **uuid**: UUID ở định dạng chuẩn. Đây là một giá trị lớn hơn so với timeuuid.
- **varchar**: Tương tự như văn bản. Cả hai có thể được sử dụng để thay thế cho nhau.
- **variant**: Một giá trị số nguyên với độ chính xác tùy ý. Nó được khuyến nghị nên sử dụng một kiểu dữ liệu với độ chính xác cần thiết.

3.8.2. Các kiểu dữ liệu tập hợp

- **set**: Kiểu này lưu trữ một bộ sưu tập các giá trị. Các giá trị được lưu trữ không có thứ tự, nhưng CQLSH sẽ trả về dữ liệu đã được sắp xếp.

Ví dụ:

```
ALTER TABLE website_tracker ADD tagsSet set<text>;
UPDATE website_tracker
SET tagsSet = {'tag1'} WHERE id = 1;
SELECT tagsSet FROM website_tracker WHERE id = 1;
tagsSet
-----
{'tag1'}

UPDATE website_tracker SET tagsSet = tagsSet + {'gat2'}
WHERE id = 1;
SELECT tagsSet FROM website_tracker WHERE id = 1;
tagsSet
-----
{'gat2', 'tag1'}
```

- **list**: Một danh sách cũng lưu trữ một bộ sưu tập các giá trị nhưng lưu trữ chúng theo kiểu đã được sắp xếp, mặc định sắp theo thứ tự chèn vào.

Ví dụ:

```
ALTER TABLE website_tracker ADD tagsList list<text>;
UPDATE website_tracker
SET tagsList = ['tag1'] WHERE id = 1;
SELECT tagsList FROM website_tracker WHERE id = 1;
tagsList
-----
['tag1']

UPDATE website_tracker
SET tagsList = tagsList + ['gat2'] WHERE id = 1;

SELECT tagsList FROM website_tracker WHERE id = 1;
```

```
tagsList
```

```
-----
```

```
['tag1', 'tag2']
```

- **map**: Một bản đồ chứa một bộ sưu tập của các cặp khóa-giá trị. Đây có thể là bất cứ điều gì ngoại trừ kiểu counter.

Ví dụ:

```
ALTER TABLE website_tracker ADD tagsMap map<text, text>;
```

```
UPDATE website_tracker
```

```
SET tagsMap = {'tag1': 'Tag One'} WHERE id = 1;
```

```
SELECT tagsMap FROM website_tracker WHERE id = 1;
```

```
tagsMap
```

```
-----
```

```
{'tag1': 'Tag One'}
```

```
UPDATE website_tracker
```

```
SET tagsMap['tag2'] = 'Tag Two' WHERE id = 1;
```

```
SELECT tagsMap FROM website_tracker WHERE id = 1;
```

```
tagsMap
```

```
-----
```

```
{'tag1': 'Tag One', 'tag2': 'Tag Two'}
```

3.8.3. Các kiểu dữ liệu tùy biến

Chúng ta có thể định nghĩa các kiểu dữ liệu của riêng mình trong Cassandra.

Ví dụ: Chúng ta muốn lưu trữ địa chỉ đăng ký của trang web.

```
CREATE TYPE address (
```

```
... street text,
```

```
... city text,
```

```
... state text);
```

```
ALTER TABLE website_tracker ADD reg_address address;
```

3.9. Primary key

Primary key còn được gọi là Partition key (Khóa phân vùng).

Khi đề cập tới Primary key người ta thường đề cập tới một số thuật ngữ: partition key, composite/compound key, clustering key. Trong đó:

- Partition key chịu trách nhiệm phân bổ dữ liệu qua các node (server).
- Clustering key chịu trách nhiệm sắp xếp dữ liệu theo Partition key.
- Primary key tương đương với Partition key trong trường hợp key được tạo nên bởi một field.
- Composite/Compound key là key được tạo nên bởi nhiều hơn một field.

Ví dụ 1:

```
CREATE TABLE items_by_id (  
    item_id BIGINT PRIMARY KEY,  
    item_name TEXT,  
    item_description TEXT,  
    item_price DECIMAL  
);
```

Trong trường hợp trên thì Primary key chính là Partition key.

Ví dụ 2:

```
CREATE TABLE carts_by_user (  
    user_id BIGINT,  
    cart_id BIGINT,  
    state BOOLEAN,  
    total DECIMAL,  
    PRIMARY KEY ((user_id), cart_id)  
) WITH CLUSTERING ORDER BY (cart_id ASC);
```

Trong trường hợp này thì chúng ta có 1 Composite primary key là (user_id, cart_id), trong đó user_id là partition key và cart_id là clustering key.

3.10. Primary index

Trong Cassandra, primary index cho column family là index của row key. Mỗi node sẽ bảo trì index này cho dữ liệu nó quản lý. Những row được thiết kế cho node bởi việc cấu hình partitioner và cấu hình replica placement strategy. Primary index trong Cassandra cho phép tìm kiếm dòng bởi khóa của nó. Khi mỗi node biết range khóa mà mỗi node quản lý, row được yêu cầu có thể cấp phát hiệu quả bằng cách duyệt qua index dòng chỉ trên relevant replicas.

Với randomly partitioner row keys (mặc định trong Cassandra) row key được phân bố bởi MD5 hash và không thể scanned thứ tự giống index B-tree truyền thống. Sử dụng Ordered partitioner cho phép truy vấn một khoảng các row key. Nhưng không khuyến khích sử dụng partitioner này bởi vì khó bảo trì việc phân tán dữ liệu thông qua các node.

3.11. Secondary index

Trong cơ chế của Cassandra, đối với câu lệnh SELECT chúng ta có thể sử dụng mệnh đề WHERE để lọc dữ liệu, tuy nhiên để sử dụng được mệnh đề WHERE thì điều kiện khá ngặt nghèo.

Ví dụ: Chúng ta có bảng đường đua như sau:

```
CREATE TABLE keyspace.rank_by_year_and_name (  
    race_year int,  
    race_name text,  
    cyclist_name text,  
    rank int,  
    PRIMARY KEY ((race_year, race_name), rank)  
);
```

Với partition key là race_year và race_name, clustering key là rank. Và ở bảng trên khi chúng ta muốn liệt kê ra thông tin của cuộc đua thì ít nhất trong mệnh đề WHERE phải chứa 2 thuộc tính của partition key,
Ví dụ:

```
SELECT * FROM rank_by_year_and_name  
WHERE race_year=? AND race_name=?
```

Thế nhưng, giả sử khi chúng ta muốn xem thông tin của những cuộc đua trong 1 năm nhất định. Nếu chúng ta dùng câu truy vấn theo logic thông thường như trong SQL sau:

```
SELECT * FROM rank_by_year_and_name
WHERE race_year =?
```

Câu truy vấn trên sẽ báo lỗi, bởi vì trong mệnh đề WHERE không đề cập đến hai thuộc tính của partition key.

Đây là lúc chúng ta nên sử dụng index, index cho phép chúng ta lọc dữ liệu với điều kiện where với những thuộc tính có thể không nằm trong partition key.

Giả sử, chúng ta muốn thực hiện câu lệnh SELECT bên trên hoạt động đúng, chúng ta cần tạo một index cho nó.

Cú pháp của câu lệnh tạo index như sau:

```
CREATE INDEX [ IF NOT EXISTS ] index_name
ON [keyspace_name.]table_name
([ (KEYS | FULL)] column_name)
(ENTRIES column_name);
```

Giải pháp đối với câu lệnh SELECT bên trên:

```
CREATE INDEX ON keyspace.rank_by_year_and_name (race_year)
SELECT * FROM rank_by_year_and_name
WHERE race_year =?
```

Bây giờ giá trị trả về sẽ chính xác không gặp lỗi.

Tóm lại, để truy cập dữ liệu nhanh và hiệu quả ứng với điều kiện cho trước trên một cột nào đó, và cột này không nằm trong partition key thì chúng ta nên sử dụng index. Nó được sử dụng cho hầu hết các mục đích như: collection, collection column, static column và bất kỳ cột nào ngoại trừ counter column.

4. CÁCH TRIỂN KHAI/CÀI ĐẶT HỆ QUẢN TRỊ CASSANDRA

4.1. Yêu cầu cài đặt

- Cài đặt phiên bản mới nhất của Java 8, Oracle Java Standard Edition 8 hoặc OpenJDK 8
- Để sử dụng cqlsh, cần cài đặt phiên bản mới nhất của python2.7 hoặc python3.6+

4.2. Cài đặt trên Linux

Cassandra được hỗ trợ mạnh mẽ trên các bản phân phối Linux sau:

- Ubuntu trên đa số các phiên bản LTS từ 16.04 đến 18.04.
- CentOS và RedHat Enterprise Linux (RHEL) bao gồm 6.6 đến 7.7.
- Amazon Linux AMIs.
- Debian phiên bản 8 và 9.
- SUSE Enterprise Linux 12.

Để cài đặt Cassandra trên các phiên bản phân phối của linux sẽ có 3 phương pháp chính:

- Sử dụng Docker image.
- Sử dụng file nhị phân Tarball.
- Sử dụng các Package installation (RPM, YUM).

Nếu bạn đã từng sử dụng docker thì phương pháp đầu tiên là cách dễ nhất để cài đặt Cassandra. Bạn chỉ cần cài đặt Docker trên linux, tải image của Cassandra và chạy với một vài dòng lệnh.

Tuy vậy với nhiều người, cài đặt với file nhị phân Tarball cũng là một cách dễ. Tarball sẽ giải nén toàn bộ nội dung vào một địa điểm trong filesystem ở dạng nhị phân và định nghĩa những file trong các thư mục ở địa điểm đó. Điểm đặc biệt của cách thức này là ta không cần quyền của root để cài đặt Cassandra trên các bản phân phối linux.

Cách cuối là sử dụng Package installation thì bạn cần quyền root để cài đặt. Đây là cách tối ưu nhất khi hoàn thành sản phẩm. Bạn cài đặt cấu trúc RPM trên CentOS và các bản phân phối linux phát triển trên RHEL nếu bạn muốn cài đặt Cassandra với YUM. Cài đặt cấu trúc Debian trên

Ubuntu và các bản phân phối linux phát triển trên debian nếu bạn muốn cài đặt Cassandra với APT.

- Cài đặt với Docker image:

- Bạn cần tải docker image của Cassandra ở phiên bản mới nhất:

```
docker pull cassandra:latest
```

- Sau đó chạy Cassandra:

```
docker run --name cass_cluster cassandra:latest
```

--name là tên bạn muốn đặt cho Cassandra cluster được tạo ra

- Ngoài ra để sử dụng cqlsh của Cassandra node vừa tạo ra:

```
docker exec -it cass_cluster cqlsh
```

- Cài đặt với file nhị phân Tarball:

- Đầu tiên bạn cần tải về file nhị phân tarball phiên bản mới nhất từ trang chủ của Cassandra:

```
$ curl -OL http://apache.mirror.digitalpacific.com.au/cassandra/4.0.0/apache-cassandra-4.0.0-bin.tar.gz
```

- Sau đó giải nén file:

```
$ tar xzvf apache-cassandra-4.0.0-bin.tar.gz
```

Các file sẽ được giải nén vào thư mục apache-Cassandra-4.0.0/


```
<tarball_installation>/
  bin/           1
  conf/          2
  data/          3
  doc/
  interface/
  javadoc/
  lib/
  logs/          4
  pylib/
  tools/         5
```

Trong đó:

- 1 là thư mục xử lý các lệnh để chạy Cassandra, cqlsh, nodetool, và SSTable tools.
- 2 là thư mục chứa file Cassandra.yaml là file định nghĩa.
- 3 là thư mục chứa commit, hint và SSTables logs.
- 4 là thư mục chứa system và debug logs.
- 5 là thư mục chứa stress tool.

- Để chạy Cassandra:

```
$ cd apache-cassandra-4.0.0/ && bin/cassandra
```

- Để chạy cqlsh:

```
$ bin/cqlsh
```

- Để kiểm tra trạng thái của Cassandra:

```
$ bin/nodetool status
```

- Cài đặt với Package installation:

- APT:

- Bạn cần thêm apache Cassandra repo vào thư file source.list của hệ thống:

```
$ echo "deb http://www.apache.org/dist/cassandra/debian 40x main" | sudo  
tee -a /etc/apt/sources.list.d/cassandra.sources.list  
deb http://www.apache.org/dist/cassandra/debian 40x main
```

- Sau đó thêm các key của apache Cassandra repo vào các key tin cậy trên hệ thống:

```
$ curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key ad  
d -
```

- Sau đó update để hệ thống cập nhật thư viện:

```
$ sudo apt-get update
```

- Sau đó cài đặt Cassandra:

```
$ sudo apt-get install cassandra
```

Cassandra sẽ tự khởi động sau khi cài đặt.

- Để kiểm tra trạng thái:

```
$ nodetool status
```

- Để sử dụng cqlsh:

```
$ cqlsh
```

○ YUM:

- Bạn cần thêm apache Cassandra repo vào thư file repo của hệ thống:

```
[cassandra]  
name=Apache Cassandra  
baseurl=https://downloads.apache.org/cassandra/redhat/40x/  
gpgcheck=1  
repo_gpgcheck=1  
gpgkey=https://downloads.apache.org/cassandra/KEYS
```

- Sau đó update để hệ thống cập nhật thư viện:

```
$ sudo yum update
```

- Cài đặt Cassandra:

```
$ sudo yum install cassandra
```

- Khởi động Cassandra:

```
$ sudo service cassandra start
```

- Để kiểm tra trạng thái:

```
$ nodetool status
```

- Để sử dụng cqlsh:

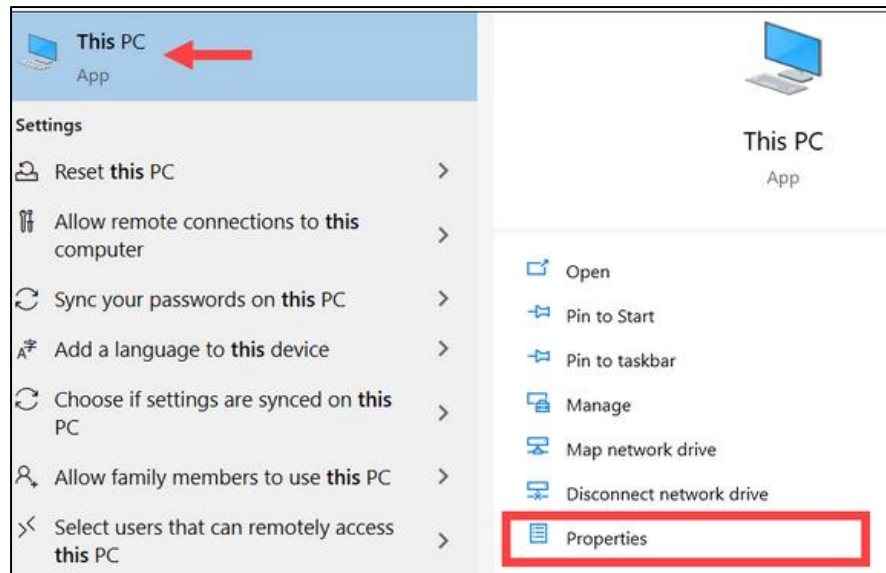
```
$ cqlsh
```

4.3. Cài đặt trên Windows

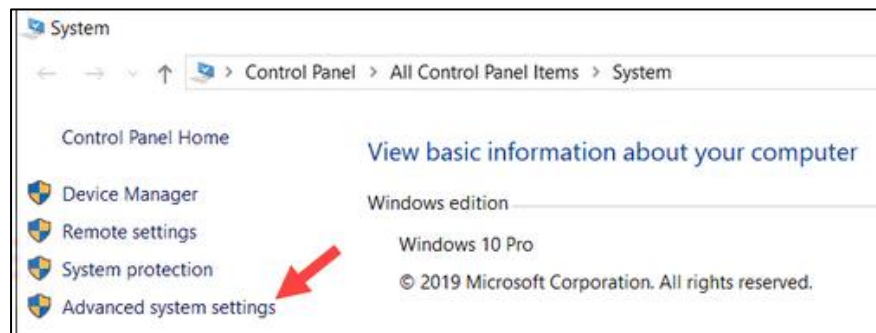
- Tài file Tarball Cassandra version mới nhất từ trang chủ:



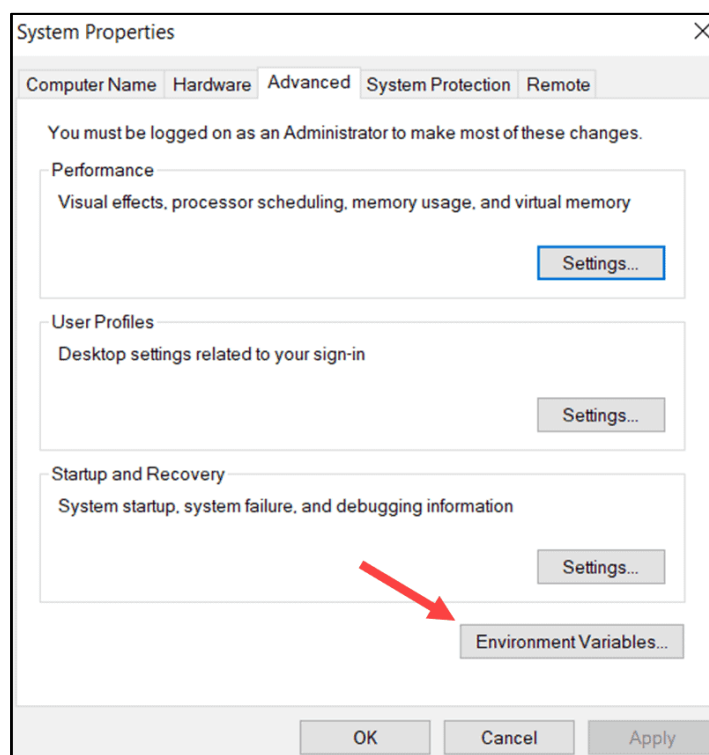
- Giải nén thành thư mục apache-Cassandra-4.0.1/
- Sau đó định nghĩa các biến môi trường:
 - Vào PC chọn phần Properties:



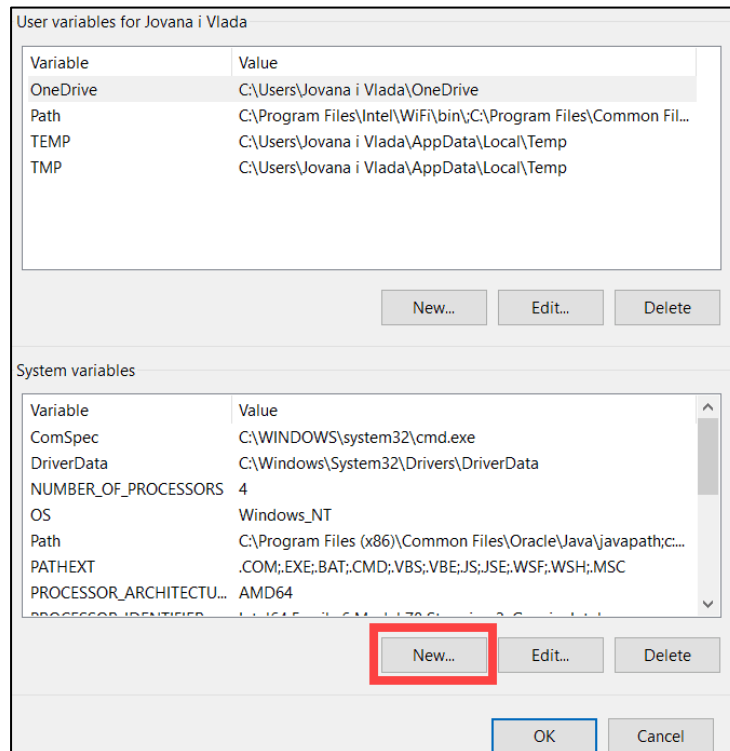
- Sau đó chọn phần **Advanced system settings**



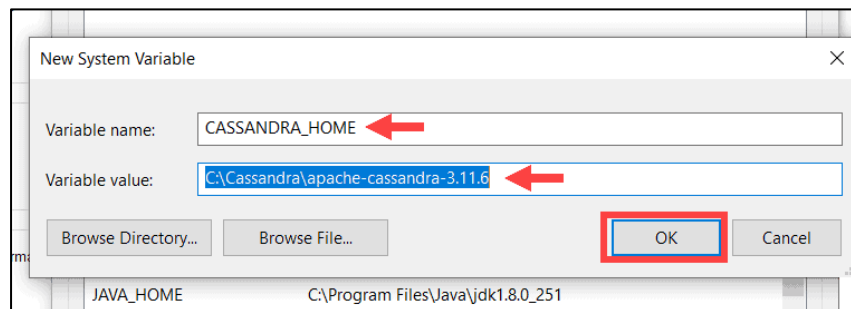
- Click vào nút **Environment Variables...**



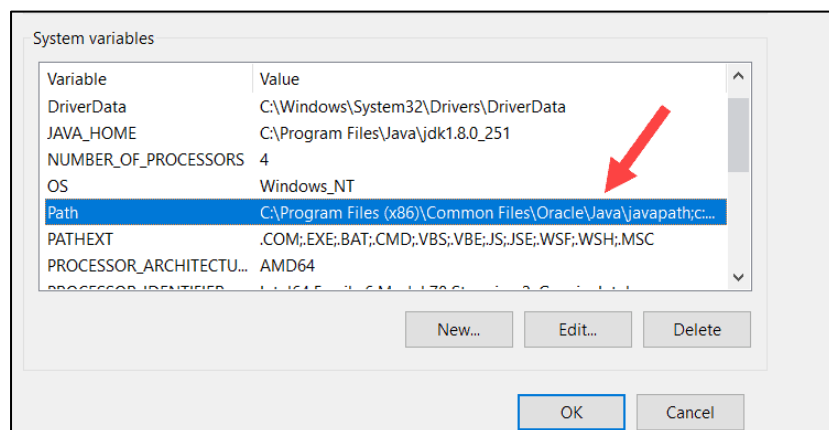
- Click vào nút **New** để tạo một đường dẫn mới:



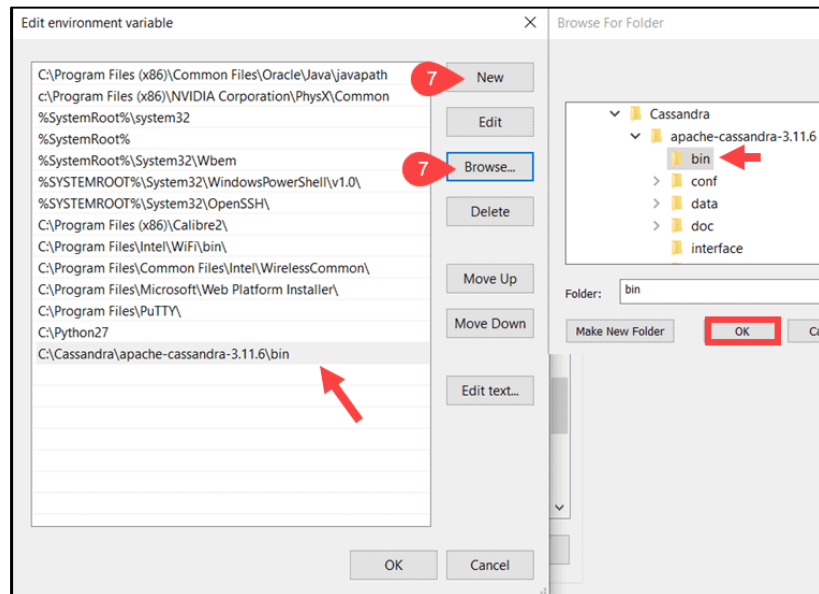
- Điền **Cassandra_HOME** cho tên biến, và giá trị của biến là đường dẫn vào folder chứa Cassandra đã tạo ở các bước đầu:



- Sau đó click đúp vào biến **Path**:



- Click vào button **New** sau đó là button **Browse**. Chọn đường dẫn tới thư mục bin trong thư mục Cassandra đã tạo ra ở các bước đầu:



- Sau đó bạn có thể chạy Cassandra bằng CMD:

```
C:\Windows\System32\cmd.exe - cassandra
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Cassandra\apache-cassandra-3.11.6\bin>cassandra
WARNING! Powershell script execution unavailable.
Please use 'powershell Set-ExecutionPolicy Unrestricted'
on this user-account to run cassandra with fully featured
functionality on this platform.
Starting with legacy startup options
Starting Cassandra Server
INFO [main] 2020-05-15 16:28:39.036 YamlConfigurationLoader: java:89 - Config
```

- Chạy cqlsh:

```
C:\Windows\System32\cmd.exe - cqlsh
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Cassandra\apache-cassandra-3.11.6\bin>cqlsh
WARNING: console codepage must be set to cp65001 to support utf
If you experience encoding problems, change your console codepage

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native proto
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab
cqlsh>
```

4.4. Cài đặt trên MacOS

Có thể cài đặt Cassandra trên macos bằng cách sử dụng phương pháp Package installation.

- Sử dụng brew của MacOS để cài đặt:

```
brew install cassandra
```

- Sau đó chạy Cassandra:

```
cassandra -f
```

- Sử dụng cqlsh:

```
cqlsh
```

4.5. Triển khai

Để triển tạo một cơ sở dữ liệu trong Cassandra đầu tiên chúng ta cần tạo một keyspace với cú pháp:

```
CREATE KEYSPACE [IF NOT EXISTS] keyspace_name  
WITH REPLICATION = {replication_map}  
[AND DURABLE_WRITES = [true|false] ;
```

Ví dụ:

```
CREATE KEYSPACE IF NOT EXISTS App_data  
WITH REPLICATION = {'class' : 'NetworkTopologyStrategy',  
'datacenter1' : 3, 'datacenter2': 2};
```

Tiếp theo, chúng ta tạo một bảng trong Cassandra với cú pháp:

```
CREATE TABLE [IF NOT EXISTS] keyspace_name.table_name  
{  
    [column] [data Type],  
    .....,  
    Primary key (([columns as partition key]),[columns as  
cluster key])  
};
```

Để insert một dòng dữ liệu trong Cassandra:

```
INSERT INTO KeyspaceName.TableName (ColumnName1,  
(Column1Value, Column2Value, Column3Value . . .);
```

Update trong Cassandra:

```
UPDATE KeyspaceName.TableName  
SET ColumnName1 = new Column1Value,  
ColumnName2=new Column2Value,  
ColumnName3=new Column3Value,  
.  
.  
.  
WHERE ColumnName = ColumnValue;
```

Delete trong Cassandra:

```
DELETE FROM KeyspaceName.TableName  
WHERE ColumnName1 = ColumnValue;
```

5. HỆ THỐNG PHÙ HỢP

Cassandra thường được sử dụng để lưu trữ dạng dữ liệu theo thời gian như các phân tích/logs/dữ liệu số lượng lớn được thu về từ các bộ cảm biến sensor. Ứng dụng kiểu đó có số lượng dữ liệu cần ghi rất lớn và ít phải đọc và dữ liệu là không có mối quan hệ (non-relational).

Cassandra cung cấp giải pháp cho các vấn đề như cần có một hệ thống ghi rất nặng và một hệ thống phản hồi nhanh trên dữ liệu được lưu trữ. Xem xét trường hợp sử dụng phân tích Web, nơi lưu trữ dữ liệu nhật ký như để đếm số lần truy cập mỗi giờ, theo trình duyệt, theo IP, v.v. theo cách thức thời gian thực. Các tính năng đặc trưng của Cassandra:

- Kiến trúc có thể mở rộng rộng rãi: Cassandra có một thiết kế không có tổng thể, trong đó tất cả các nút đều ở cùng một cấp (peer-to-peer), mang lại sự đơn giản trong hoạt động và dễ dàng mở rộng quy mô.

- Kiến trúc Masterless (vô chủ): Dữ liệu có thể được ghi và đọc trên bất kỳ nút nào. Vì thế khi gặp lỗi, nó có nhiều điểm có thể gặp lỗi chứ không phải một điểm duy nhất.
- Hiệu suất quy mô tuyến tính (Scale linearly): Khi nhiều nút được thêm vào, hiệu suất của Cassandra sẽ tăng lên (việc sử dụng phần cứng trên nút được giảm).
- Không có điểm lỗi duy nhất: Cassandra sao chép dữ liệu trên các nút khác nhau để đảm bảo không có điểm lỗi duy nhất.
- Phát hiện lỗi và khôi phục: Các nút bị lỗi có thể dễ dàng được khôi phục và phục hồi.
- Mô hình dữ liệu linh hoạt và động: Hỗ trợ các kiểu dữ liệu với tính năng ghi và đọc nhanh.
- Bảo vệ dữ liệu: Dữ liệu được bảo vệ với thiết kế nhật ký cam kết và xây dựng trong bảo mật như cơ chế sao lưu và khôi phục.
- Tính nhất quán dữ liệu có thể điều chỉnh: Hỗ trợ tính nhất quán dữ liệu mạnh mẽ trong kiến trúc phân tán.
- Nhân rộng nhiều trung tâm dữ liệu: Cassandra cung cấp tính năng sao chép dữ liệu trên nhiều trung tâm dữ liệu.
- Nén dữ liệu: Cassandra có thể nén tới 80% dữ liệu mà không cần bất kỳ chi phí nào.
- Ngôn ngữ truy vấn Cassandra: Cassandra cung cấp ngôn ngữ truy vấn tương tự như ngôn ngữ SQL. Nó rất dễ dàng cho các nhà phát triển cơ sở dữ liệu quan hệ chuyển từ cơ sở dữ liệu quan hệ sang Cassandra.

Từ các tính năng trên cho thấy, Cassandra rất thích hợp để sử dụng thực tế bởi tính khả dụng, khả năng chịu lỗi cao, tự do kiểm soát nhất quán, mô hình dữ liệu rất phong phú. Kiến trúc của Cassandra không có SPOF (Single Point of Failure - một thành phần hệ thống quan trọng với khả năng ngừng hoạt động hệ thống trong quá trình chuyển đổi dự phòng), toàn bộ hệ thống sẽ không bị ngừng lại do một phần nào đó gặp lỗi, service vẫn tiếp tục vận hành. Cassandra hỗ trợ rất nhiều ngôn ngữ dưới dạng client code như: C++, Java, Python, PHP,....

Dưới đây là một số trường hợp sử dụng mà Cassandra nên được ưu tiên:

Nhắn tin

Cassandra là một cơ sở dữ liệu tuyệt vời cho các công ty cung cấp điện thoại di động và dịch vụ nhắn tin. Những công ty này có một lượng dữ liệu khổng lồ, vì vậy Cassandra là tốt nhất cho họ. Một số ứng dụng mạng xã hội sử dụng Cassandra như Facebook, Instagram, Twitter,

Ứng dụng Internet of things (IOT)

Cassandra là một cơ sở dữ liệu tuyệt vời cho các ứng dụng mà dữ liệu đang đến với tốc độ rất cao từ các thiết bị hoặc cảm biến khác nhau.

Danh mục sản phẩm và ứng dụng bán lẻ

Cassandra được nhiều nhà bán lẻ sử dụng để bảo vệ giỏ hàng lâu bền và nhập và xuất danh mục sản phẩm nhanh chóng.

Công cụ đề xuất và phân tích phương tiện truyền thông xã hội

Cassandra là một cơ sở dữ liệu tuyệt vời cho nhiều công ty trực tuyến và nhà cung cấp phương tiện truyền thông xã hội để phân tích và giới thiệu cho khách hàng của họ.

6. ỨNG DỤNG MINH HỌA

6.1. Mô tả

- Hệ thống quản lý giỏ hàng online. Nhóm em chọn ứng dụng này, vì đối với giỏ hàng của khách hàng cần được bảo vệ lâu bền, và đối với những hệ thống mua bán lớn cần một lượng lưu trữ giỏ hàng lớn, và các thao tác thêm vào giỏ hàng chủ yếu là thao tác ghi nên nhóm em quyết định chọn hệ quản trị Cassandra cho hệ thống này để đảm bảo tốc độ ghi cao khi người dùng muốn thêm một sản phẩm vào giỏ hàng.
- Hệ thống cho phép lưu trữ:
 - Người dùng: có id, tên, số điện thoại và email
 - Mỗi user có một hoặc nhiều giỏ hàng trong hệ thống, mỗi giỏ hàng có id, trạng thái và tổng tiền.

- Mỗi giỏ hàng chứa nhiều sản phẩm, một sản phẩm có thể thuộc nhiều giỏ hàng, có thời gian thêm vào giỏ và số lượng.
- Mỗi sản phẩm có id, tên, mô tả sản phẩm, loại sản phẩm và giá sản phẩm.
- Mỗi loại sản phẩm có id, tên loại, mô tả.

6.2. Các hệ chức năng

Khách hàng:

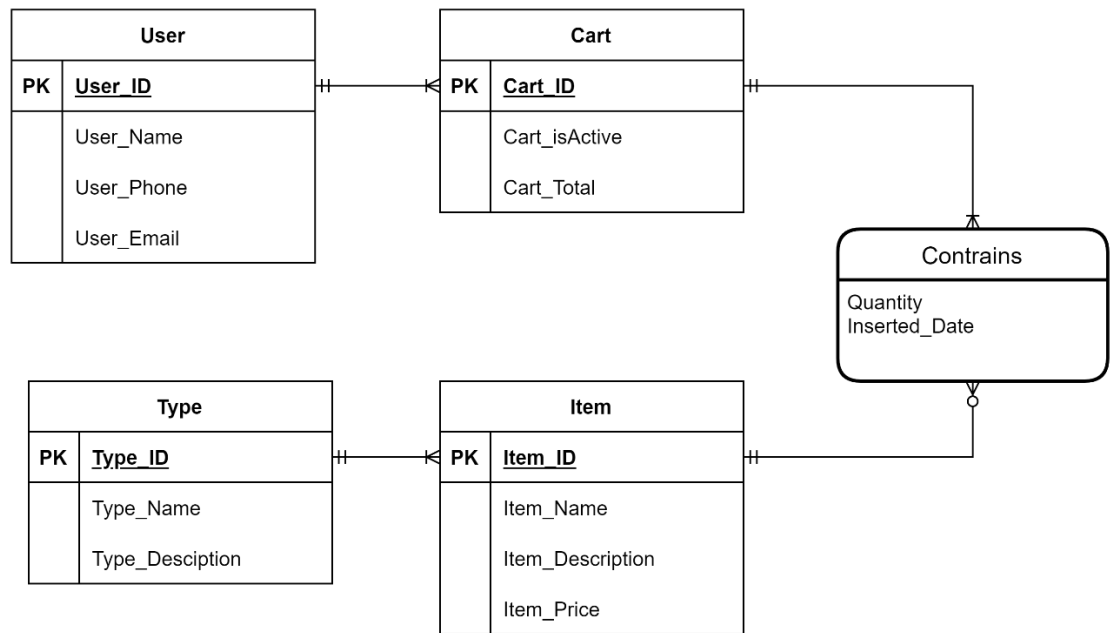
- Khách hàng xem tất cả giỏ hàng của bản thân.
Lưu thông tin giỏ hàng của khách hàng, với partition key là mã khách hàng.
- Khách hàng có thể xem tất cả sản phẩm của một giỏ hàng.
Lưu thông tin của các sản phẩm trong một giỏ hàng, với partition key là mã giỏ hàng.
- Khách hàng có thể tìm một sản phẩm bằng ID hoặc tên của sản phẩm.
Lưu thông tin của sản phẩm với partition key là mã sản phẩm/tên sản phẩm.
- Khách hàng có thể xem loại sản phẩm theo tên hoặc ID của loại sản phẩm.
Lưu thông tin loại sản phẩm, với partition key là ID hoặc tên của mã loại.
- Khách hàng xem danh sách sản phẩm theo loại.
Lưu thông tin của sản phẩm với partition key là mã loại sản phẩm.
- Khách hàng có thể thêm một sản phẩm vào giỏ hàng.
- Khách hàng có thể lưu một giỏ hàng, hoạt kích hoạt một giỏ hàng.

Chủ shop:

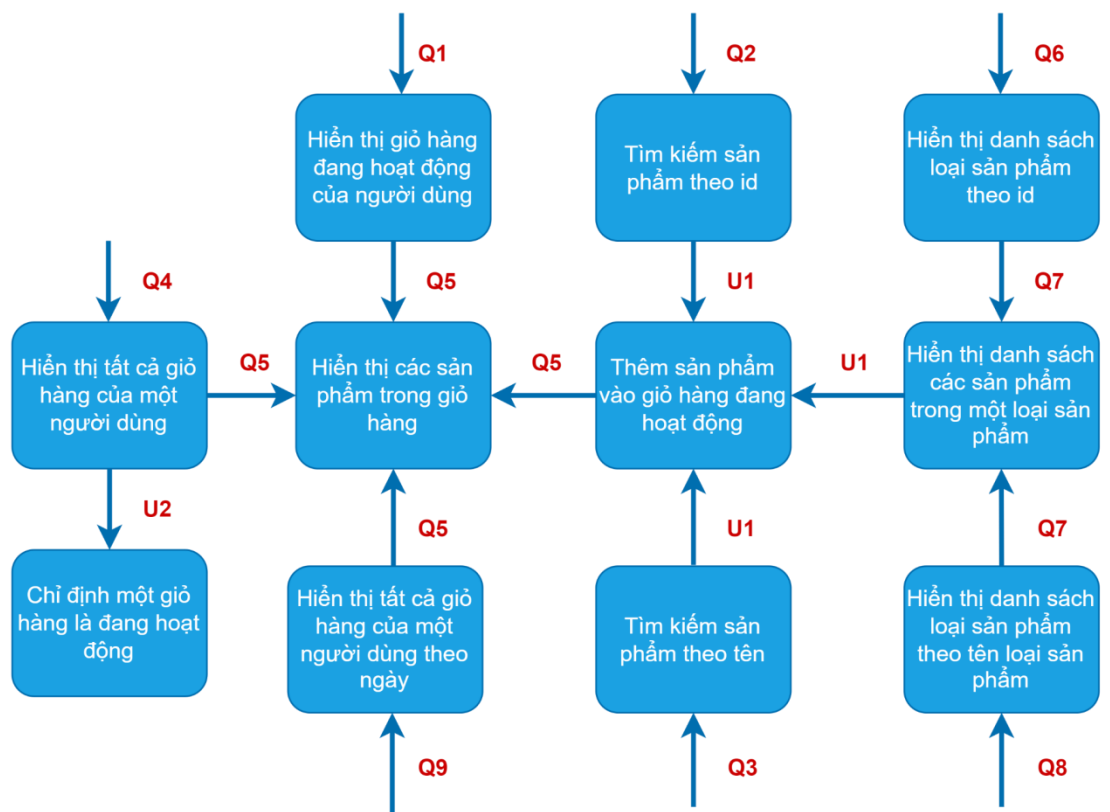
- Chủ shop có thể thêm loại sản phẩm vào danh sách loại sản phẩm của shop.
- Chủ shop có thể thêm sản phẩm vào danh sách sản phẩm của shop.

6.3. Thiết kế CSDL

6.3.1. Mô hình ER



6.3.2. Quy trình làm việc



Q1: Hiển thị giỏ hàng của người dùng.

Q2: Tìm kiếm sản phẩm theo id sản phẩm.

Q3: Tìm kiếm sản phẩm theo tên sản phẩm.

Q4: Hiển thị tất cả giỏ hàng của một người dùng.

Q5: Hiển thị các sản phẩm trong giỏ hàng.

Q6: Hiển thị danh sách loại sản phẩm theo id loại sản phẩm.

Q7: Hiển thị danh sách các sản phẩm trong một loại sản phẩm.

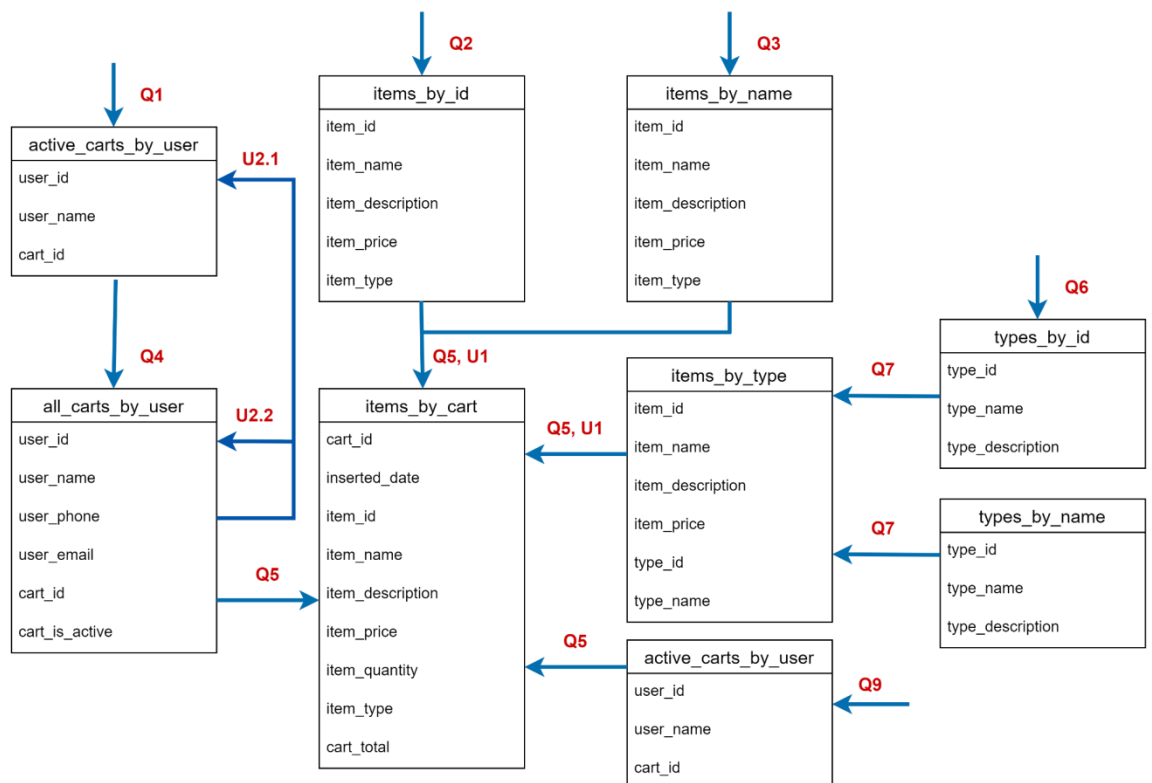
Q8: Hiển thị danh sách loại sản phẩm theo tên loại sản phẩm.

Q9: Hiển thị tất cả giỏ hàng của một người dùng theo ngày.

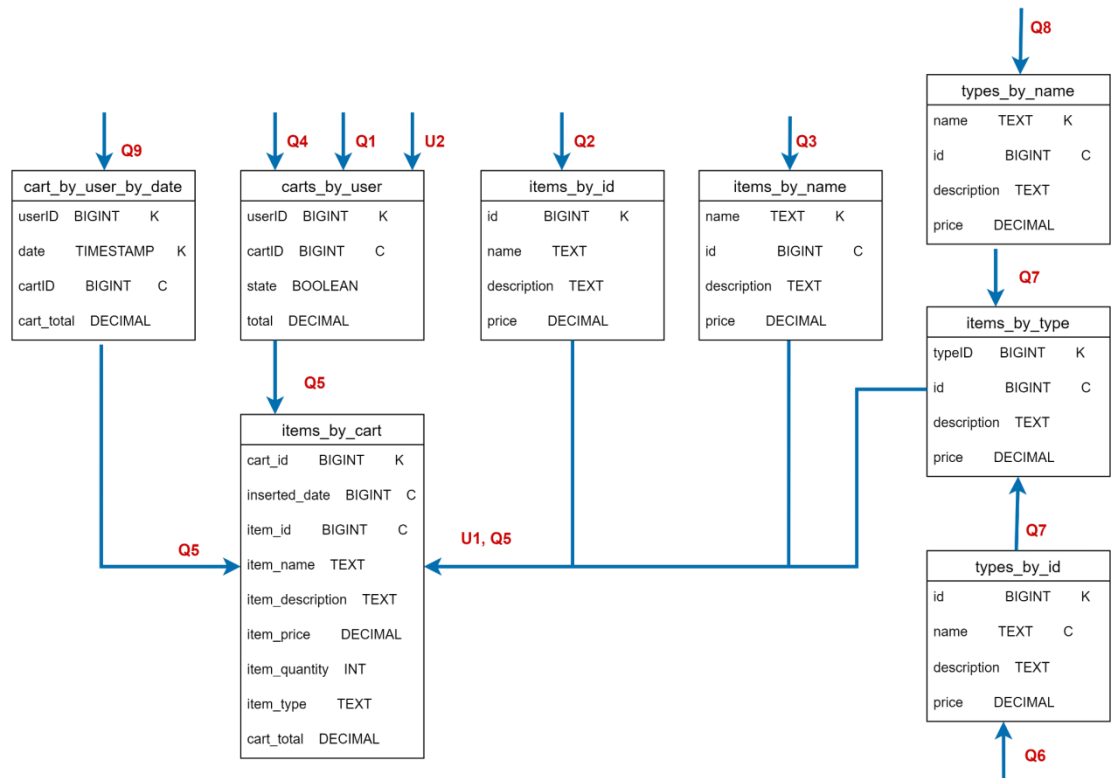
U1: Thêm sản phẩm vào giỏ hàng đang hoạt động.

U2: Chỉ định một giỏ hàng là đang hoạt động.

6.3.3. Mô hình thiết kế logic



6.3.4. Mô hình thiết kế vật lý



6.3.5. Kịch bản demo các tính năng hỗ trợ

Kịch bản demo chính cho ứng dụng quản lý giỏ hàng:

- Khách hàng có thể xem danh sách giỏ hàng đang hoạt động, sau đó xem các sản phẩm trong giỏ hàng.
- Khách hàng có thể tìm kiếm sản phẩm theo id của sản phẩm, sau đó có thể thêm sản phẩm vào giỏ hàng đang hoạt động.
- Khách hàng tìm kiếm sản phẩm theo tên của sản phẩm, sau đó có thể thêm sản phẩm vào giỏ hàng đang hoạt động.
- Khách hàng có thể xem loại sản phẩm theo ID, sau đó xem danh sách sản phẩm của một loại sản phẩm, sau đó khách hàng có thể thêm sản phẩm vào giỏ hàng đang hoạt động.
- Khách hàng có thể xem loại sản phẩm theo tên loại, sau đó xem danh sách sản phẩm của một loại sản phẩm, sau đó khách hàng có thể thêm sản phẩm vào giỏ hàng đang hoạt động.
- Khách hàng có thể chọn những giỏ hàng đã lưu, sau đó chỉ định một giỏ hàng sang trạng thái đang hoạt động, khách hàng cũng có thể xem danh sách các sản phẩm trong giỏ hàng đã lưu.

Link video demo và script demo:

https://drive.google.com/drive/folders/1yoYb9_RK4HFhxcW9nXVmy2MOFzdFU3r1?usp=sharing

7. BẢNG PHÂN CÔNG VÀ ĐÁNH GIÁ

STT	Công việc	Chi tiết công việc	Phân công	Hoàn thành
1	Tìm hiểu tổng quát, đặc điểm, tính năng hỗ trợ	Tổng quát về Cassandra và đặc điểm của HQT CSDL này	19127515	100%
		Các tính năng hỗ trợ: giao tác, trigger, materialized views	19127507	100%
		Các tính năng hỗ trợ: Cơ chế ghi theo lô, cơ chế phục hồi dữ liệu, cơ chế bảo mật	19127512	100%
2	Mô hình dữ liệu	Khái niệm, keyspace, column family, column	19127507	100%
		Expiring column, counter column, super column	19127512	100%
		Kiểu dữ liệu, primary key, index	19127515	100%
3	Triển khai và cài đặt	Linux	19127512	100%
		Window	19127515	100%
		MasOS	19127507	100%
4	Phân tích hệ thống phù hợp	Cuộc họp qua Zoom	Cả nhóm	100%
5	Demo, đánh giá các tính năng	Cuộc họp qua Zoom	Cả nhóm	100%
		Quay video demo	19127515	100%
6	Hoàn thành báo cáo		19127507	100%
7	Nộp bài		19127515	100%

MSSV	Họ và tên	Mức độ đóng góp
19127515	Võ Đình Phúc	34%
19127507	Nguyễn Quang Phú	33%
19127512	Lâm Hoàng Phúc	33%

8. TÀI LIỆU THAM KHẢO

- Nguồn tham khảo chính: <https://Cassandra.apache.org/>
- Một số nguồn khác:
 - Tổng quan về Cassandra: <https://viblo.asia/p/kien-truc-mang-Cassandra-RQqKLxp6K7z>
 - https://www.tutorialspoint.com/Cassandra/Cassandra_architecture.html
 - Cơ chế phục hồi dữ liệu: <https://docs.datastax.com/en/Cassandra-oss/2.2/Cassandra/operations/opsAboutSnapshots.html>
 - Cơ chế bảo mật: <https://docs.datastax.com/en/Cassandra-oss/3.0/Cassandra/configuration/secureIntro.html>
 - Secondary index: <https://www.geeksforgeeks.org/concept-of-indexing-in-apache-Cassandra/>
 - Mô hình dữ liệu: <https://itfromzero.com/database/Cassandra/tao-user-cap-quyen-trong-Cassandra.html>
 - Hệ thống phù hợp: <https://blog.pythian.com/Cassandra-use-cases/>
 - <https://hoidapit.com/bigdata/huong-dan-ve-Cassandra-cho-nguoi-moi-bat-dau-hoc-trong-3-ngay-8m8qntq>