SAMSUNG

Samsung Innovation Campus

Khoá học Big Data

Together for Tomorrow!
Enabling People
Education for Future Generations

Chương 5.

Phân tích Big Data

Khoá học Big Data

Mô tả chương

Mục tiêu:

- ✓ Chúng ta sẽ tìm hiểu cách sử dụng các công cụ khác nhau để phân tích dữ liệu trong nền tảng Hadoop.
  - SQL là ngôn ngữ của truy vấn. Chúng ta sẽ đạt được sự hiểu biết toàn diện về ngôn ngữ này.
  - Trong phân tích dữ liệu, có nhiều cách khác nhau mà chúng tôi muốn phân tích dữ liệu.
  - Truyền dữ liệu thời gian thực, dữ liệu lịch sử chế độ hàng loạt và các truy vấn đặc biệt tương tác
  - Những trường hợp sử dụng này thường yêu cầu các công cụ khác nhau để xử lý chúng. Chúng ta sẽ bắt đầu với chế độ hàng loạt và chế độ tương tác trong chương này.
- ✓ Chúng tôi sẽ hiểu rõ hơn về hai kiến trúc chính để xử lý phân tích đồng thời dữ liệu lịch sử và thời gian thực
  - Chúng ta sẽ thấy tại sao việc phân tích dữ liệu lịch sử và thời gian thực cùng nhau lại quan trọng.
  - Kiến trúc Lambda phát triển và triển khai hai hệ thống riêng biệt để xử lý chúng trong khi Kappa cố gắng khắc phục khó khăn khi phát triển và duy trì hai hệ thống riêng biệt.

Nội dung:

1. Giới thiệu về SQL
2. Phân tích cơ bản
3. Phân tích nâng cao
4. Kiến trúc phân tích dữ liệu truyền tr

Bài 1.

Giới thiệu về SQL

Phân tích Big Data

Bài 1

Giới thiệu về SQL

DB (CSDL) là gì?

- Một bộ dữ liệu
- Một tập hợp dữ liệu liên quan đến logic được tổ chức hữu cơ
- Một tập hợp dữ liệu chung mà nhiều ứng dụng có thể lưu trữ và vận hành thông tin tích hợp
- Một tập hợp thông tin được tích hợp và quản lý để sử dụng chung
- Đặc trưng
  - Khả năng truy cập thời gian thực – xử lý thời gian thực
  - Tiến hóa liên tục
  - Chia sẻ đồng thời
  - Tham chiếu nội dung – được tham chiếu theo giá trị dữ liệu

DBMS (Hệ thống quản lý CSDL) là gì?

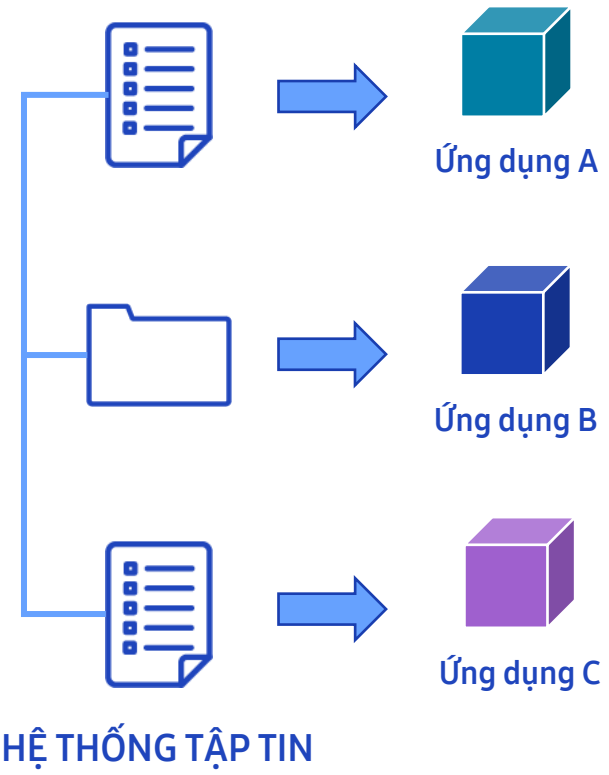
- DBMS, một hệ thống quản lý cơ sở dữ liệu, là một chương trình được tạo thành từ nhiều chương trình khác nhau giúp dễ dàng tạo và quản lý cơ sở dữ liệu.
- Oracle, MySQL, SQLServer, PostgreSQL, MariaDB...
- Vì hầu hết các DB được tạo và vận hành thông qua DBMS, DB và DBMS đôi khi được sử dụng với cùng một nghĩa, nhưng hai cách diễn đạt khác nhau rõ ràng, nhưng nhìn chung có thể chấp nhận chúng như nhau

Các loại hệ thống cơ sở dữ liệu

- Hệ thống tập tin
- DB phân cấp
- Cơ sở hạ tầng mạng
- DB quan hệ
- DB hướng đối tượng
- NoSQL

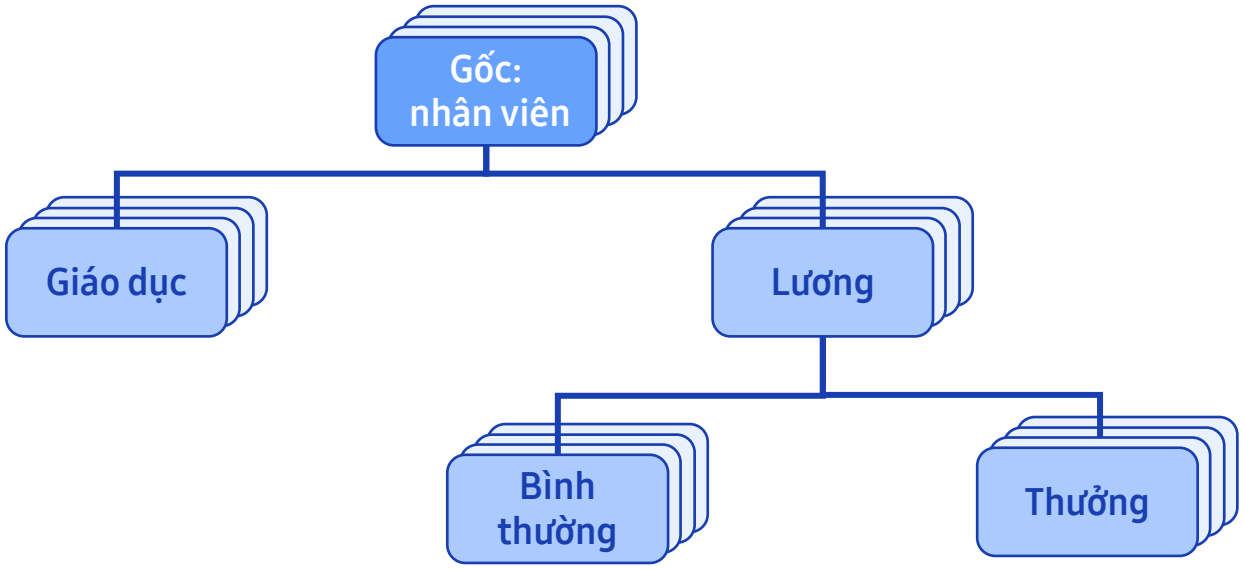
HỆ THỐNG TẬP TIN

- Để xử lý dữ liệu, chúng được sắp xếp theo một quy tắc nhất định và được lưu dưới dạng tệp trên đĩa.
  - Cơ sở dữ liệu ban đầu bắt đầu với hệ thống tập tin
  - Một hệ thống tệp về cơ bản là một cách sắp xếp các tệp trên phương tiện lưu trữ, chẳng hạn như đĩa cứng.
  - Khi các bản ghi được lưu trữ trong tệp được đọc từng cái một, quá trình xử lý cần thiết được lặp lại cho đến khi bản ghi đích được hiển thị.
  - Dữ liệu dư thừa có thể có trong một hệ thống tệp.
- Để xử lý dữ liệu tồn tại trong hệ thống tệp, con trỏ phải được sử dụng.
- Các chức năng truy vấn phức tạp hoặc đặc biệt không được hỗ trợ vì quá trình xử lý đơn vị bản ghi phải được lặp lại.



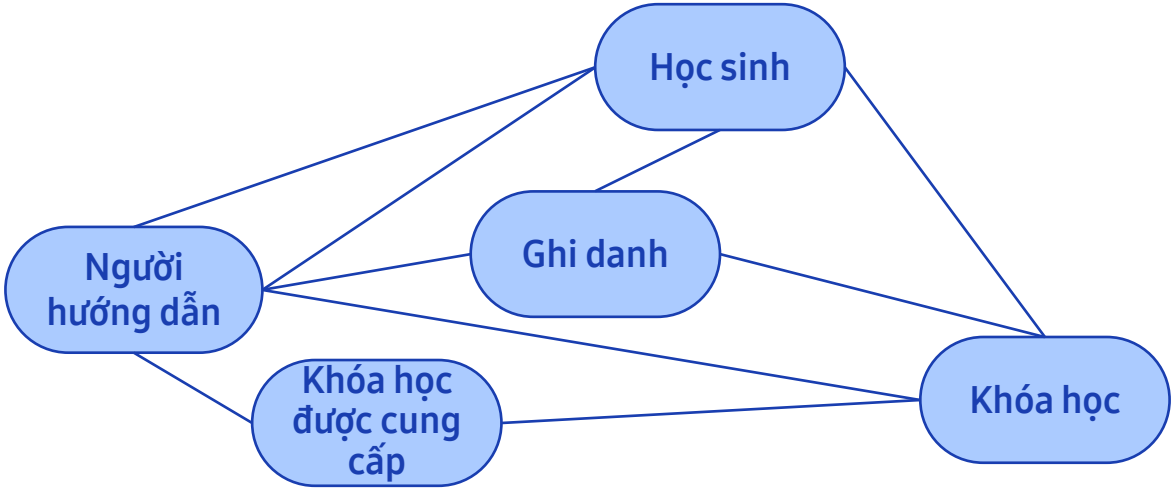
CSDL phân cấp

- Dữ liệu được phân cấp và cấu trúc trong các mối quan hệ cấp dưới và cấp dưới
  - Ưu điểm: Tốc độ truy cập dữ liệu nhanh và có thể dễ dàng dự đoán được việc sử dụng dữ liệu.
  - Nhược điểm: Không dễ để điều chỉnh quy trình thay đổi sau khi cài đặt ban đầu vì nó bao gồm mối quan hệ cấp dưới.



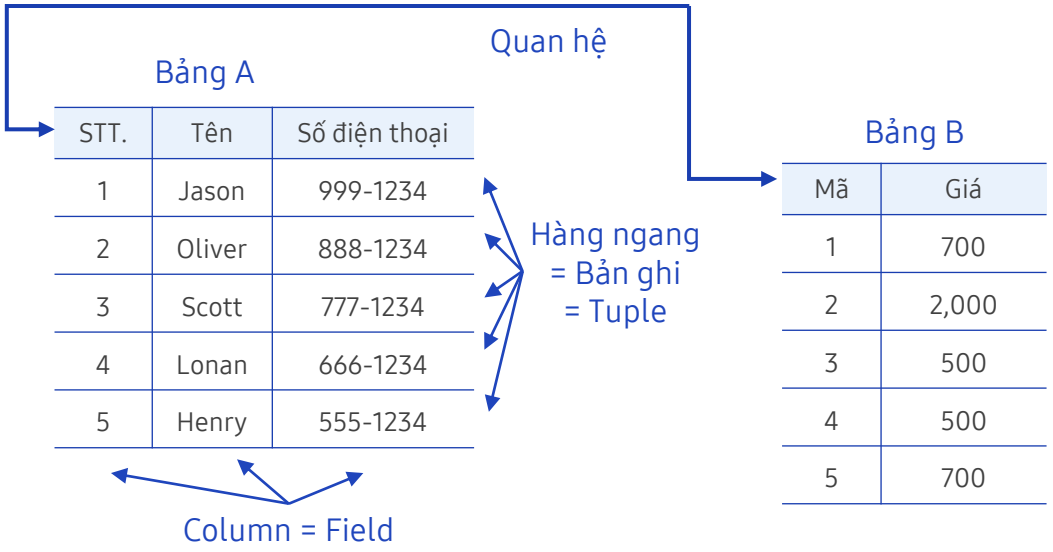
Mạng cơ sở dữ liệu

- Một mô hình dữ liệu thể hiện một cách logic cấu trúc dữ liệu dưới dạng một nút trên mạng và một hệ thống trong đó mỗi nút được định cấu hình trong một mối quan hệ bình đẳng (ở đây, nút đề cập đến dữ liệu, không phải hệ thống)
  - Ưu điểm: Giải quyết được nhược điểm của CSDL phân cấp. Nó có thể chứa các mối quan hệ khác nhau giữa các thực thể dữ liệu.
  - Nhược điểm: Xây dựng và thiết kế phức tạp, và cuối cùng là phụ thuộc vào dữ liệu



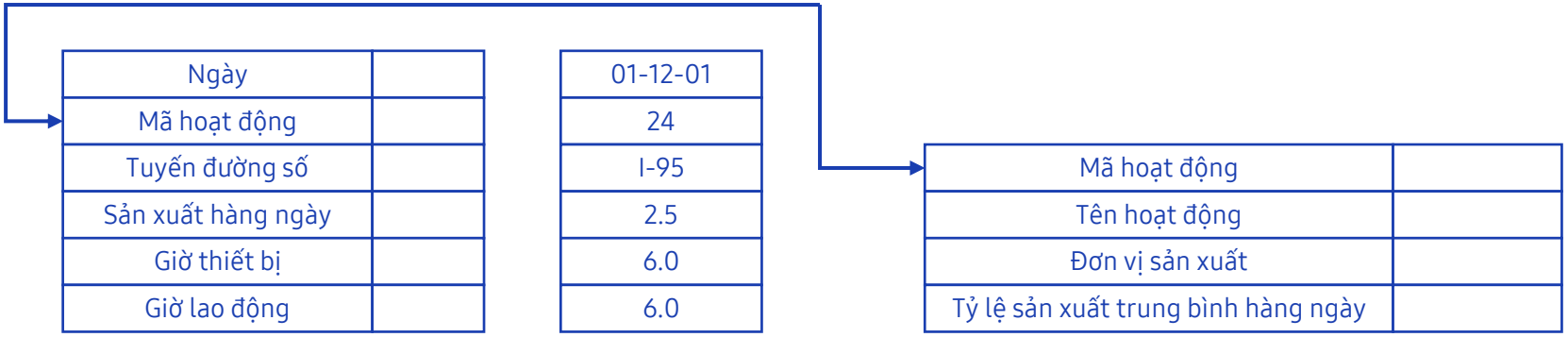
CSDL quan hệ

- Một cấu trúc trong đó các mối quan hệ toán học và logic được hình thành dưới dạng một bảng và một số cột trong bảng được sao chép với các bảng khác để xác định mối tương quan giữa mỗi bảng.
  - Ưu điểm: Khả năng thích ứng cao với những thay đổi trong công việc, dễ dàng sử dụng để thay đổi công việc và bảo trì thuận tiện. Do đó, năng suất cũng được cải thiện.
  - Nhược điểm: Tải hệ thống cao do sử dụng nhiều tài nguyên hơn các DBMS khác.



CSDL hướng đối tượng

- Nó được thiết kế để khắc phục những hạn chế cơ bản của việc xử lý dữ liệu đa phương tiện một cách trơn tru và chỉ các kiểu dữ liệu kiểu nghiệp vụ của RDBMS.
- Cơ sở dữ liệu đối tượng khác với cơ sở dữ liệu quan hệ được định hướng theo bảng. Cơ sở dữ liệu quan hệ đối tượng là sự kết hợp của cả hai cách tiếp cận.
  - UniSQL, Lưu trữ Đối tượng



Đối tượng 1: Báo cáo bảo trì

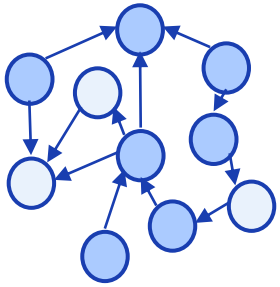
Đối tượng 1: Phiên bản

Đối tượng 2: Báo cáo bảo trì

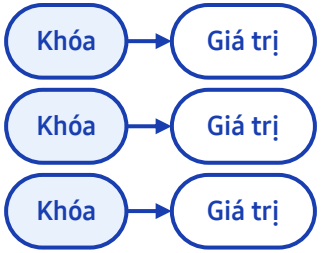
NoSQL

- Cơ sở dữ liệu NoSQL cung cấp cơ chế lưu trữ và truy xuất dữ liệu được mô hình hóa theo các phương tiện khác với quan hệ dạng bảng được sử dụng trong cơ sở dữ liệu quan hệ.
- Cấu trúc dữ liệu được sử dụng bởi cơ sở dữ liệu NoSQL (ví dụ: cặp khóa-giá trị, cột rộng, biểu đồ hoặc tài liệu) khác với cấu trúc dữ liệu được sử dụng theo mặc định trong cơ sở dữ liệu quan hệ, giúp một số thao tác nhanh hơn trong NoSQL.

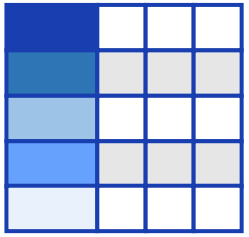
Tài liệu



Đồ thị



Khóa-Giá trị



Cột rộng

Ưu điểm của RDBMS

- Cơ sở dữ liệu SQL (quan hệ) có mô hình quản lý và lưu trữ dữ liệu trưởng thành.
- Cơ sở dữ liệu SQL hỗ trợ khái niệm dạng xem cho phép người dùng chỉ xem dữ liệu mà họ được phép xem.
- Cơ sở dữ liệu SQL hỗ trợ thủ tục lưu trữ SQL cho phép các nhà phát triển cơ sở dữ liệu triển khai một phần logic nghiệp vụ vào cơ sở dữ liệu.
- Cơ sở dữ liệu SQL có mô hình bảo mật tốt hơn so với cơ sở dữ liệu NoSQL.

Các tính năng của RDBMS (CSDL quan hệ) (1/2)

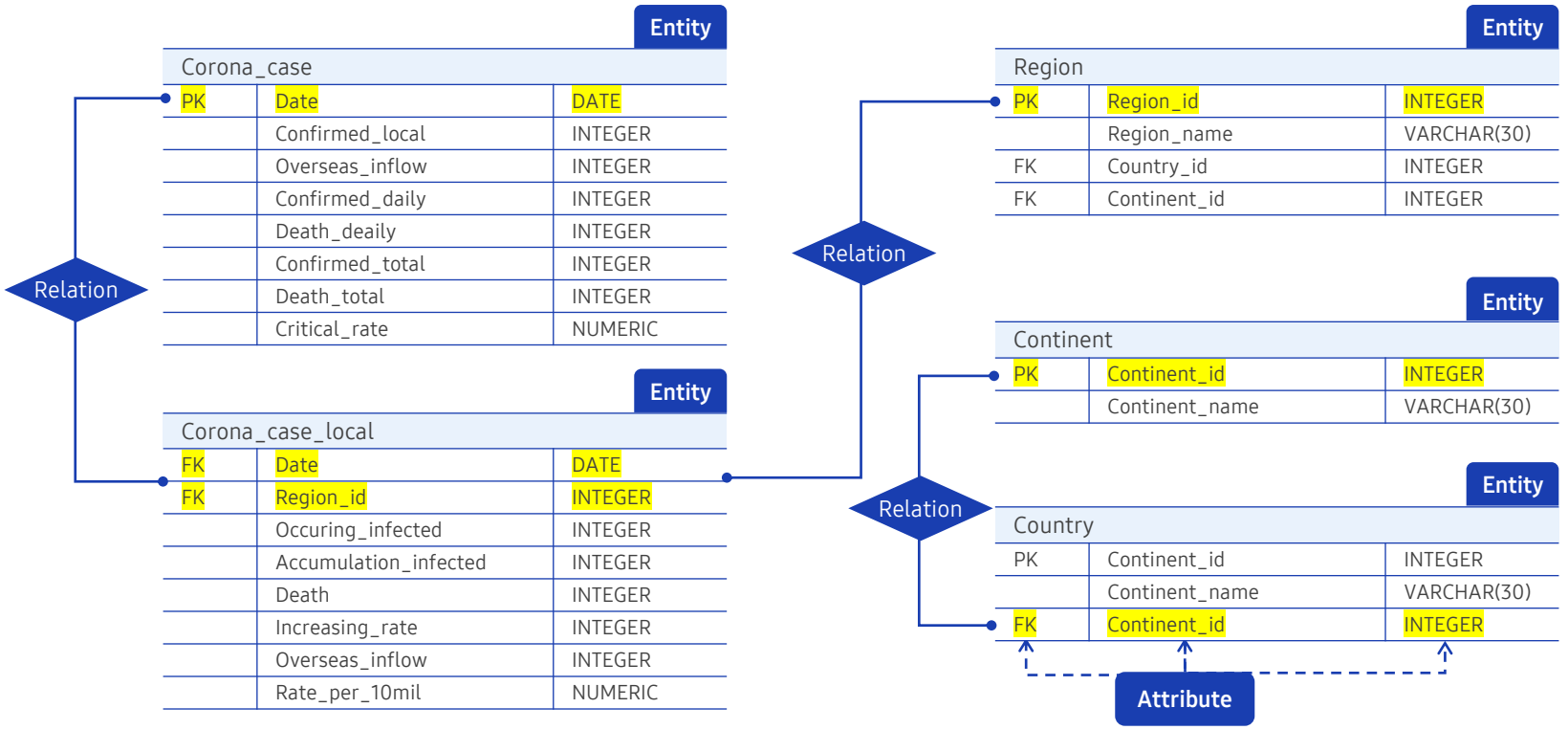
- Hỗ trợ giao dịch, ACID (Nguyên tử, Tính nhất quán, Cách ly, Độ bền)
- Biểu diễn tất cả dữ liệu dưới dạng bảng hai chiều
  - Bảng là một bài lưu trữ dữ liệu cơ bản gồm hàng (bản ghi, bộ) và cột (trường, mục).
  - Tập hợp các bảng có liên quan với nhau
  - Mô hình Mối quan hệ Thực thể (ER) của bản thiết kế cơ sở dữ liệu
  - Theo mô hình ER, cơ sở dữ liệu được tạo và cơ sở dữ liệu bao gồm một hoặc nhiều bảng.
- Quan hệ
  - Tìm kiếm dữ liệu mong muốn bằng cách kết hợp nhiều bảng

Các tính năng của RDBMS (CSDL quan hệ) (2/2)

- DataBase - cơ sở dữ liệu là một loại lưu trữ dữ liệu.
  - Một kho lưu trữ thông tin được sắp xếp thành các hình đơn giản, thông thường
  - Giống như một bảng trong Excel, nó bao gồm các bảng và mỗi bảng bao gồm các hàng và cột.
  - Mỗi hàng được gọi là một bản ghi và các bản ghi được tạo thành từ một số phần thông tin, trong đó các phần là các cột.
- Management System là phần mềm cho phép bạn chèn (insert), tìm kiếm (select), sửa đổi (update), và xóa (delete) các bản ghi trong CSDL.
  - Khả năng xử lý dữ liệu
  - Nhiều DBMS thực hiện điều này bằng cách hỗ trợ SQL (Ngôn ngữ truy vấn có cấu trúc).

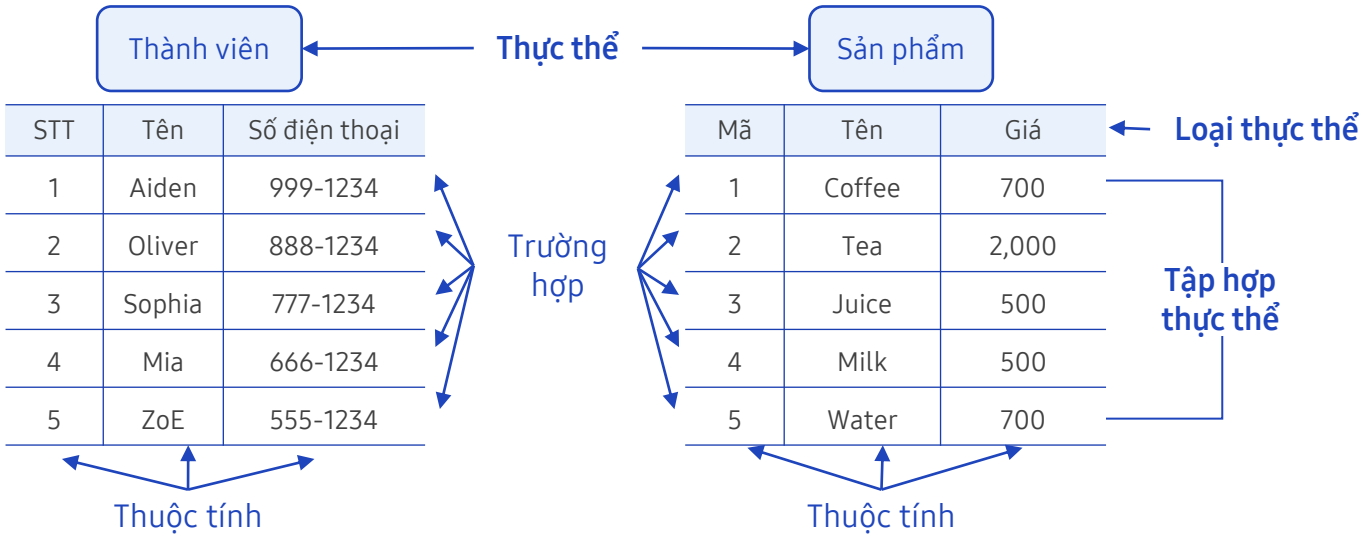
Thiết kế cơ sở dữ liệu (1/4)

- Mô hình quan hệ thực thể
  - Sơ đồ ER được tạo dựa trên ba khái niệm cơ bản: Thực thể (Enity), Thuộc tính (Attributes) và Quan hệ (Relations)



Thiết kế cơ sở dữ liệu (2/4)

- Thực thể có nghĩa là một tập hợp dữ liệu
  - Đó là dữ liệu phải được lưu trữ và quản lý.
  - Nó đề cập đến một khái niệm, địa điểm, sự kiện, vv
  - Nó đề cập đến một đối tượng hữu hình hoặc vô hình.



Thiết kế cơ sở dữ liệu (3/4)

- Thuộc tính
  - Được định nghĩa là đơn vị dữ liệu nhỏ nhất không còn được phân tách về mặt ngữ nghĩa để được quản lý dưới dạng một phiên bản
  - Các thuộc tính không còn tách rời về mặt ngữ nghĩa mà là các phần tử mô tả các thực thể và thành phần của các thể hiện

Ví dụ) Số học sinh, tên, địa chỉ

Tên xe, màu sắc, năm sản xuất, dung tích, kiểu dáng...

Tên	Màu	Năm	Di chyển	Mẫu
Porsche	red	2021	2981	911

Power unit

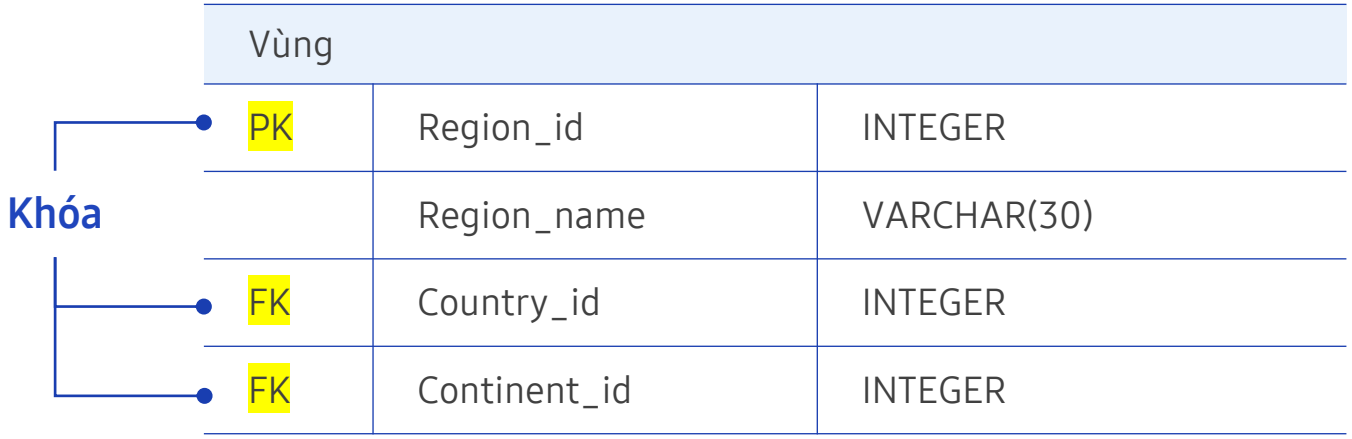
Number of cylinders	6
Bore	91.0 mm
Stroke	76.4 mm
Displacement	2,981 cc
Power (kW)	288 kW
Power (PS)	392 PS
RPM point maximum power	6,500 r/min
Maximum engine speed	7,500 r/min
Max. torque	45.9 kg·m
RPM range maximum torque	1,950 - 5,000 r/min

Thiết kế cơ sở dữ liệu (4/4)

- Mối quan hệ
  - Bảng hai chiều, được biểu thị bằng các hàng và cột
  - Một quan hệ được sử dụng để lưu trữ thông tin về một thực thể được biểu diễn trong cơ sở dữ liệu.
  - Mỗi quan hệ có một tên duy nhất.
  - Nói chung, các quan hệ không có sự tương ứng một-một với hệ thống tệp.
- Hàng = bản ghi = tuple
  - Một tập hợp các giá trị liên quan đến một thể hiện cụ thể của thực thể được đại diện bởi mối quan hệ.
  - Nghĩa là, một bộ đại diện cho một thể hiện của thực thể được đại diện bởi quan hệ.
- Cột = thuộc tính
  - Một cột có tên liên quan
  - Độ: Số cột/số thuộc tính trong một quan hệ
  - Thứ tự tối thiểu của một quan hệ hợp lệ phải có 1 -> ít nhất một thuộc tính. Thứ tự không thay đổi thường xuyên.

Khóa DBMS (1/4)

- Khóa
  - Xác định duy nhất một cái gì đó
  - Một thuộc tính hoặc tập hợp các thuộc tính được sử dụng để xác định một bộ dữ liệu cụ thể.



Vùng		
PK	Region_id	INTEGER
	Region_name	VARCHAR(30)
FK	Country_id	INTEGER
FK	Continent_id	INTEGER

* PK – Khóa chính, FK – Khóa ngoại

Khóa DBMS (2/4) – Khóa siêu và ứng viên

- Siêu khóa
  - Siêu khóa là một thuộc tính hoặc tập hợp các thuộc tính xác định duy nhất một bộ dữ liệu.
  - Nghĩa là, nếu một bộ có thể được xác định, thì tất cả đều là siêu khóa.
- Một khóa ứng viên (hoặc siêu khóa tối thiểu)
  - Là một siêu khóa không thể rút gọn thành một siêu khóa đơn giản hơn bằng cách loại bỏ một thuộc tính

Khách hàng

STT	Tên	Số xã hội	Địa chỉ	Số điện thoại
1	Aiden	810101-1111111	Seoul, Korea	000-5000-0001
2	Oliver	900101-2222222	London, England	000-6000-0001
3	Mia	830101-2333333	Paris, France	000-7000-0001
4	ZoE	820101-1444444	LA, US	000-8000-0001

Khóa DBMS (3/4) – Khóa chính (PK)

- Khóa chính là khóa chọn một trong các khóa ứng cử viên để đại diện cho nó.
- Cân nhắc khi chọn khóa chính
  - Nó phải có một giá trị duy nhất xác định bộ trong quan hệ.
  - Giá trị NULL không được phép.
  - Giá trị khóa không nên thay đổi.
  - Nó nên có càng ít thuộc tính càng tốt.
  - Sẽ không có vấn đề gì trong việc sử dụng khóa trong tương lai.

Khóa DBMS (4/4) – Khóa ngoại (FK)

- Khóa ngoại là một thuộc tính đề cập đến khóa chính của một quan hệ khác.
  - Khóa ngoại đề cập đến khóa chính của một quan hệ khác để thể hiện mối quan hệ giữa các quan hệ, đây là một đặc điểm của mô hình dữ liệu quan hệ.
- Đặc điểm của khóa ngoại
  - Biểu diễn mối quan hệ giữa các quan hệ trong một mô hình dữ liệu quan hệ.
  - Một thuộc tính đề cập đến khóa chính của một quan hệ khác.
  - Cả miền được tham chiếu (khóa ngoại) và miền được tham chiếu (khóa chính) đều có cùng miền
  - Giá trị NULL và giá trị trùng lặp được cho phép.
  - Cũng có thể có khóa ngoại tham chiếu đến khóa chính của chính nó.
  - Khóa ngoại có thể là một phần của khóa chính

Chuẩn hóa cơ sở dữ liệu

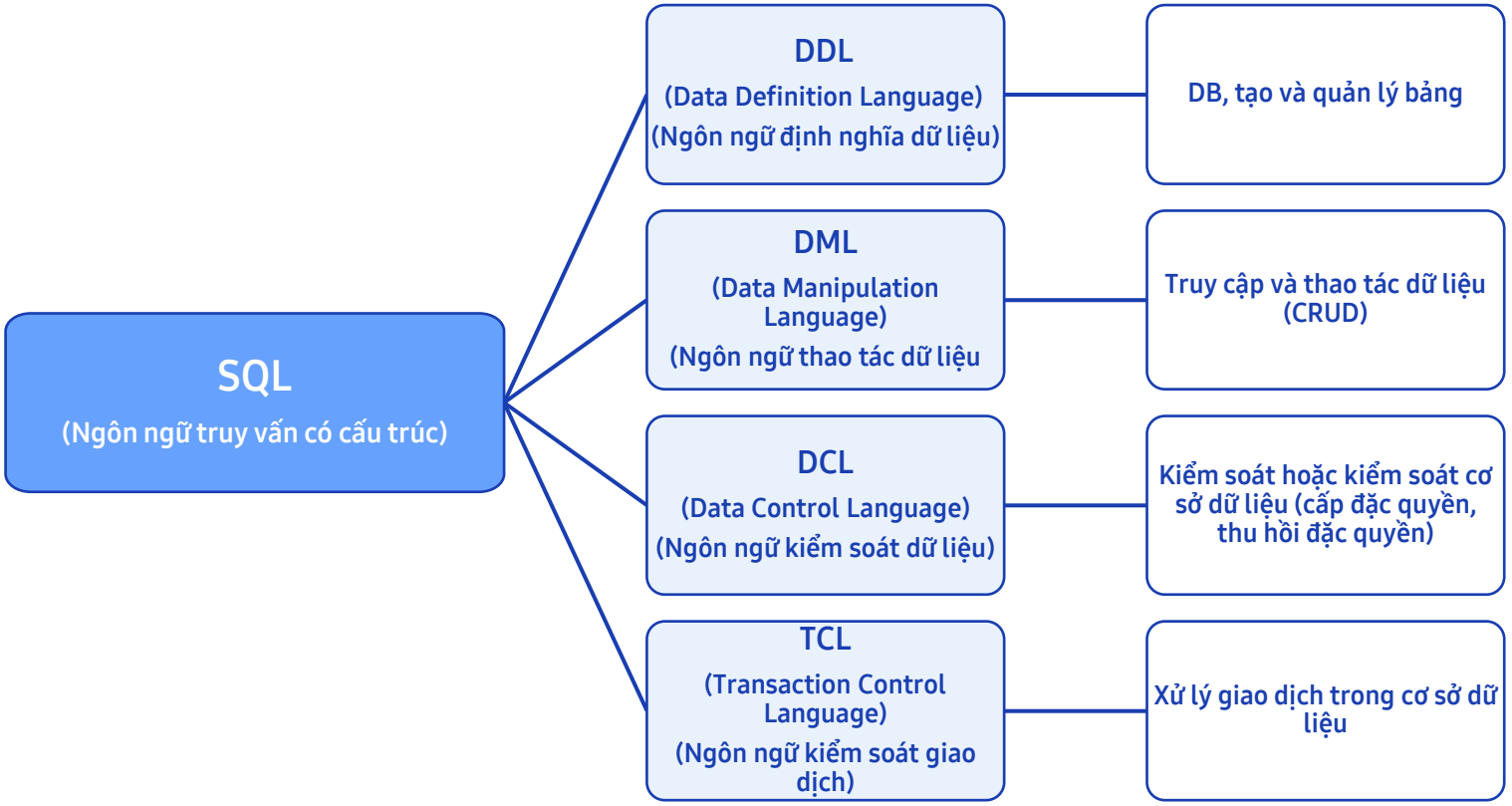
- Chuẩn hóa là để giảm dư thừa dữ liệu và cải thiện tính toàn vẹn của dữ liệu
- Các kỹ thuật thiết kế cơ sở dữ liệu giúp giảm dư thừa dữ liệu và loại bỏ các thuộc tính không mong muốn như chèn, cập nhật và xóa các điểm bất thường
  - Tách một bảng lớn thành các bảng nhỏ hơn và kết nối chúng bằng các mối quan hệ
  - Mục đích của chuẩn hóa là loại bỏ dữ liệu dư thừa (lặp lại) và đảm bảo rằng dữ liệu được lưu trữ một cách hợp lý.

    Ví dụ) 1NF, 2NF, 3NF, BCNF
- Hầu hết các hệ thống cơ sở dữ liệu là cơ sở dữ liệu được chuẩn hóa cho đến dạng chuẩn thứ ba.
- Khóa chính được xác định duy nhất là một bản ghi trong bảng và không thể rỗng.
- Khóa ngoại giúp liên kết các bảng và tham chiếu đến khóa chính

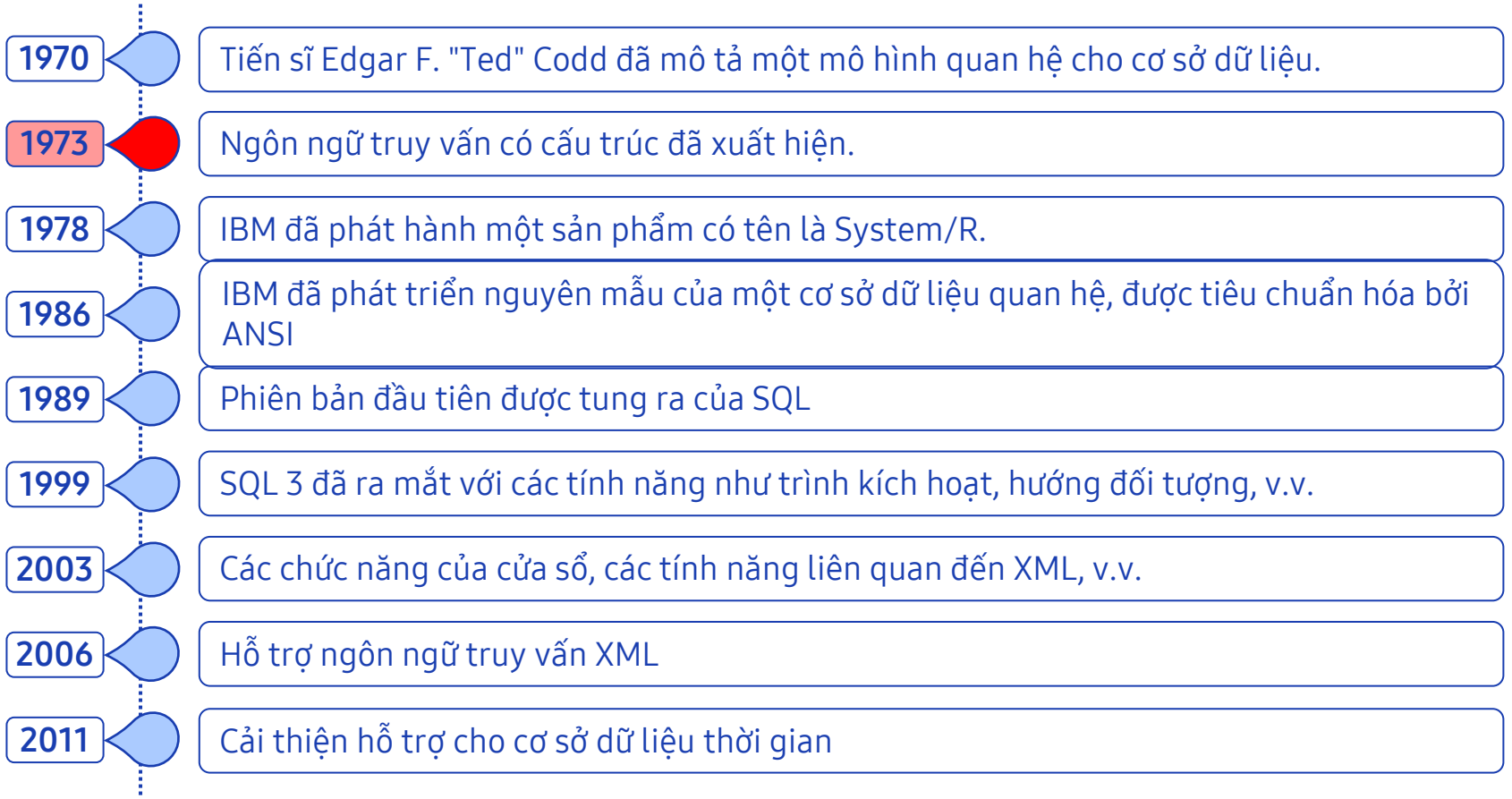
SQL là gì

- **S**tructured **Q**uery **L**anguage: Ngôn ngữ truy vấn có cấu trúc
- Một ngôn ngữ để quản lý dữ liệu trong các hệ thống cơ sở dữ liệu quan hệ.
- Chèn, tìm kiếm, cập nhật và xóa các bản ghi cơ sở dữ liệu
- Năng suất cao
- tính di động
- SQL tiêu chuẩn – Các lệnh được viết bằng SQL tiêu chuẩn có thể được thực thi theo cách tương tự trong các DBMS khác mà không cần sửa đổi các lệnh được viết một lần.

Các loại lệnh SQL



Lịch sử của SQL



SQL được sử dụng để làm gì?

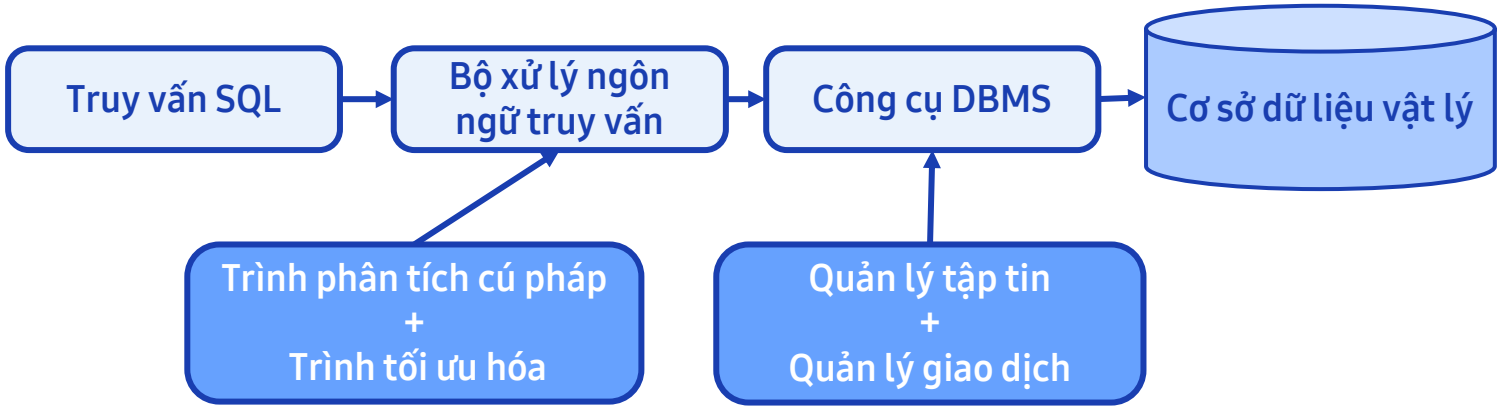
- Giúp người dùng truy cập dữ liệu trong hệ thống RDBMS
- Giúp giải thích dữ liệu
- Nó xác định dữ liệu trong cơ sở dữ liệu và thao tác dữ liệu cụ thể.
- Tạo và xóa cơ sở dữ liệu và bảng bằng cách sử dụng các lệnh SQL
- SQL sử dụng các chức năng trong cơ sở dữ liệu, tạo dạng xem và cung cấp các thủ tục được lưu trữ
- Đặt quyền trên bảng, thủ tục và dạng xem

lệnh SQL

Lệnh	Mô tả
CREATE	Định nghĩa lược đồ cấu trúc cơ sở dữ liệu
INSERT	Chèn dữ liệu vào hàng của bảng
UPDATE	Cập nhật dữ liệu trong cơ sở dữ liệu
DELETE	Xóa một hoặc nhiều hàng khỏi bảng
SELECT	Chọn thuộc tính dựa trên điều kiện được mô tả bởi mệnh đề WHERE
DROP	Xóa bảng và cơ sở dữ liệu

Quy trình SQL

- Các thành phần chính của quy trình SQL
  - Công cụ truy vấn SQL
  - Công cụ tối ưu hóa
  - Bộ điều phối truy vấn
  - Công cụ truy vấn cổ điển



Các phần tử ngôn ngữ SQL

Ngôn ngữ định nghĩa dữ liệu

- Trong ngữ cảnh của SQL, định nghĩa dữ liệu hoặc ngôn ngữ mô tả dữ liệu (DDL) là một cú pháp để tạo và sửa đổi các đối tượng cơ sở dữ liệu như bảng, chỉ mục và người dùng
- Các câu lệnh DDL tương tự như ngôn ngữ lập trình máy tính để xác định cấu trúc dữ liệu, đặc biệt là lược đồ cơ sở dữ liệu.
- Các ví dụ phổ biến về câu lệnh DDL bao gồm **CREATE**, **ALTER** và **DROP**

Cách tạo/xóa Cơ sở dữ liệu

- Bằng cách thực hiện một truy vấn SQL đơn giản trong PostgreSQL
  - Sử dụng ứng dụng pgAdmin4 cho các lệnh
- Cú pháp **CREATE DATABASE**

```
CREATE DATABASE IF NOT EXISTS db_name;
```

  - Use the command CREATE SCHEMA instead of Database for schema
- Cú pháp **DROP DATABASE**
  - Chỉ chủ sở hữu cơ sở dữ liệu hoặc siêu người dùng mới có thể xóa

```
DROP DATABASE db_name;
```

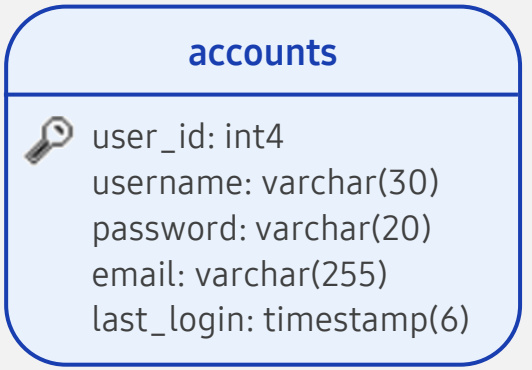
Cách tạo Bảng (1/2)

- Cú pháp **CREATE TABLE**

```
CREATE TABLE IF NOT EXISTS accounts;
```

- Ví dụ

```
CREATE TABLE IF NOT EXISTS accounts (
    user_id serial PRIMARY KEY,
    username VARCHAR (30) UNIQUE NOT NULL,
    password VARCHAR (20) NOT NULL,
    email VARCHAR (50) UNIQUE NOT NULL,
    last_login          timestamp
);
```



Cách tạo Bảng (2/2)

- PostgreSQL bao gồm các ràng buộc cột sau:
  - **NOT NULL** – Đảm bảo rằng các giá trị trong một cột không thể NULL
  - **UNIQUE** – Đảm bảo các giá trị trong một cột là duy nhất trên các hàng trong cùng một bảng
  - **PRIMARY KEY**
    - Cột khóa chính xác định duy nhất các hàng trong bảng
    - Một bảng có thể có một và chỉ một khóa chính
    - Ràng buộc khóa chính cho phép bạn xác định khóa chính của bảng
  - **CHECK** – Ràng buộc KIỂM TRA đảm bảo dữ liệu phải thỏa mãn biểu thức Boolean
  - **FOREIGN KEY**
    - Đảm bảo các giá trị trong một cột hoặc một nhóm cột từ một bảng tồn tại trong một cột hoặc một nhóm cột trong một bảng khác
    - Khác với khóa chính, một bảng có thể có nhiều khóa ngoại

Thay đổi cấu trúc bảng

- Thay đổi lược đồ (Thuộc tính của trường) của Bảng
  - Thay đổi cho cột

```
ALTER TABLE test ADD COLUMN abc BOOLEAN;
```

```
ALTER TABLE test DROP COLUMN abc;
```

```
ALTER TABLE test RENAME COLUMN abc TO new_abc;
```

  - Thay đổi cho bảng

```
ALTER TABLE test RENAME TO t_table;
```

CREATE TABLE AS SELECT (CTAS): TẠO BẢNG NHƯ CHỌN

- Cách tạo bảng thông qua kết quả cà ri sử dụng câu lệnh Select
  - Tạo bảng new_table dưới dạng câu lệnh chọn
  - Khi sao chép dữ liệu

```
CREATE TABLE table_name
AS SELECT * FROM table_name_to_copy;
```

  - Không sao chép dữ liệu mà chỉ sao chép cấu trúc bảng

```
CREATE TABLE table_name
AS SELECT * FROM table_name_to_copy
WHERE 1=2;
```

Các loại dữ liệu

- Kiểu dữ liệu cơ sở dữ liệu đề cập đến định dạng lưu trữ dữ liệu có thể chứa một loại hoặc phạm vi giá trị riêng biệt.
  - Khi các chương trình máy tính lưu trữ dữ liệu trong các biến, mỗi biến phải được chỉ định một kiểu dữ liệu riêng biệt.
- Một số kiểu dữ liệu phổ biến như sau: số nguyên, ký tự, chuỗi, số dấu phẩy động và mảng.
  - Các loại dữ liệu cụ thể hơn như sau: định dạng varchar (ký tự biến), giá trị Boolean, ngày tháng và dấu thời gian.
- Các kiểu dữ liệu cơ sở dữ liệu phổ biến
  - Số nguyên
  - Chuỗi ký tự
  - Chuỗi
  - Số điểm nổi
  - Mảng
  - Varchar
  - Boolean
  - Ngày giờ

Kiểu dữ liệu số

Loại	Mô tả
TINYINT()	-128 đến 127 bình thường 0 đến 255 UNSIGNED.
SMALLINT()	-32768 đến 32767 bình thường. 0 đến 65535 UNSIGNED.
MEDIUMINT()	-8388608 đến 8388607 bình thường. 0 đến 16777215 UNSIGNED.
INT()	-2147483648 đến 2147483647 bình thường. 0 đến 4294967295 UNSIGNED.
BIGINT()	-9223372036854775808 thành 9223372036854758077 bình thường. 0 đến 18446744073709551615 UNSIGNED.
FLOAT	Một số gần đúng nhỏ với dấu thập phân động.
DOUBLE(,)	Một số lớn với dấu thập phân động.
DECIMAL(,)	A DOUBLE được lưu dưới dạng chuỗi, cho phép dấu thập phân cố định. Lựa chọn để lưu trữ các giá trị tiền tệ.

Các kiểu dữ liệu chuỗi (String data)

Loại	Mô tả
CHAR()	Một phần cố định dài từ 0 đến 255 ký tự.
VARCHAR()	Phần biến có độ dài từ 0 đến 255 ký tự.
TINYTEXT	Một chuỗi có độ dài tối đa là 255 ký tự.
TEXT	Một chuỗi có độ dài tối đa là 65535 ký tự.
BLOB	Một chuỗi có độ dài tối đa là 65535 ký tự.
MEDIUMTEXT	Một chuỗi có độ dài tối đa là 16777215 ký tự.
MEDIUMBLOB	Một chuỗi có độ dài tối đa là 16777215 ký tự.
LONGTEXT	Một chuỗi có độ dài tối đa là 4294967295 ký tự.

Kiểu dữ liệu Ngày & Giờ

Loại	Mô tả
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

Các kiểu dữ liệu chính của MariaDB

Loại	List
Các loại chuỗi	varchar(n), char(n), text, string, binary(n)
Các loại số	tinyint, smallint, int, bigint
Các kiểu dấu chấm động	float, double, decimal(m,n)
Loại ngày/giờ	date, time, datetime, timestamp
Các kiểu Boolean	boolean

Bài 1

Giới thiệu về SQL

1.1. Tạo bảng bằng DDL

1.2. Quản lý dữ liệu bảng với DML

1.3. Truy vấn dữ liệu bằng SQL

Ngôn ngữ thao tác dữ liệu

- DML là Ngôn ngữ thao tác dữ liệu được sử dụng để tự thao tác dữ liệu
  - Ví dụ: chèn, cập nhật, xóa là các lệnh trong SQL
- Nó được sử dụng để thêm, truy xuất hoặc cập nhật dữ liệu
- Nó thêm hoặc cập nhật hàng của bảng. Các hàng này được gọi là tuple
- Lệnh CƠ BẢN hiện diện trong DML là **UPDATE, INSERT, SELECT, DELETE,** v.v.

CRUD (Create, Read, Update, Delete) (Tạo, Đọc, Cập nhật, Xóa)

- Các thao tác CRUD có nghĩa là
  - C- Create có nghĩa là "Chèn dữ liệu"
  - R- Read có nghĩa là "Chọn dữ liệu"
  - U- Update có nghĩa là "Cập nhật dữ liệu"
  - D- Delete có nghĩa là "Xóa dữ liệu"

- UPSERT - cập nhật có điều kiện khi cố chèn
  - Nếu khóa tồn tại khi chèn
  - Ví dụ

```
INSERT INTO table_name(column_1) values(value_1) ON CONFLICT target action;
```

UPDATE (CẬP NHẬT)

- Câu lệnh **UPDATE** PostgreSQL cho phép bạn sửa đổi dữ liệu trong một bảng
- Khi tham chiếu nội dung của một bảng khác trong quá trình **UPDATE**
  - Ví dụ

```
UPDATE target_table a SET a.column_1 = expression FROM ref_table b
WHERE a.column_1 = b.column_1;
```

- Sửa đổi dữ liệu hiện có trong một bảng
  - Ví dụ

```
UPDATE table_name SET column_1 = value1, column_2 = value2 WHERE condition;
```

DELETE (XÓA)

- Câu lệnh DELETE của PostgreSQL cho phép bạn xóa một hoặc nhiều hàng khỏi một bảng.
  - Khi xóa dữ liệu cụ thể khỏi bảng
  - Xóa dữ liệu thỏa mãn điều kiện
  - Ví dụ

```
DELETE FROM target_table a WHERE conditional expression;
```

INSERT (CHÈN)

- Chèn - chèn dữ liệu vào bảng
  - Chỉ nhập các cột cụ thể
  - Ví dụ

```
INSERT INTO table_name ( column1, column2 ) values ( value1, value2 );
```

- Khi nhập theo thứ tự các cột có trong bảng, có thể bỏ cột
  - Ví dụ

```
INSERT INTO table_name values ( value1, value2, value3, ... );
```

SELECT (CHỌN) (1/2)

- **SELECT** là một từ khóa SQL cho cơ sở dữ liệu biết bạn muốn truy xuất dữ liệu (bộ dữ liệu)
  - Ví dụ

```
SELECT columns_list FROM table_name;
```

  - PostgreSQL đánh giá mệnh đề FROM trước mệnh đề SELECT trong câu lệnh SELECT.
- Câu lệnh **SELECT** có các mệnh đề sau:
  - Chọn các hàng riêng biệt bằng toán tử **DISTINCT**.
  - Sắp xếp các hàng bằng mệnh đề **ORDER BY**.
  - Lọc các hàng bằng mệnh đề **WHERE**.
  - Chọn một tập hợp con các hàng từ một bảng bằng mệnh đề **LIMIT** hoặc **FETCH**.
  - Nhóm các hàng thành các nhóm sử dụng mệnh đề **GROUP BY**.
  - Lọc các nhóm sử dụng mệnh đề **HAVING**.

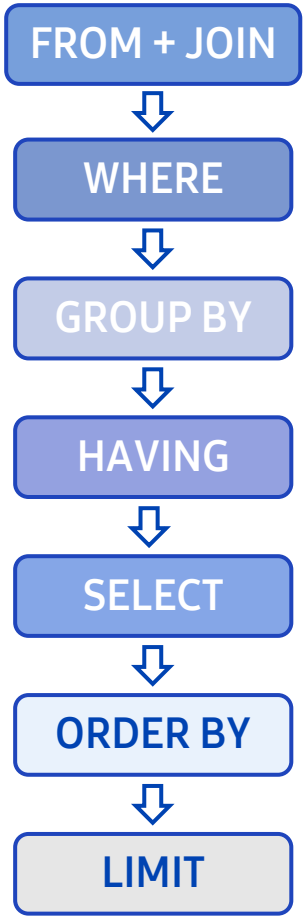
SELECT (CHỌN) (2/2)

Cú pháp

```
SELECT [DISTINCT|ALL] { * | fieldExpression [AS newName]}
FROM tableName [alias]
[WHERE condition]
[GROUP BY fieldName(s)]
[HAVING condition]
ORDER BY fieldName(s)
```

Thứ tự thực hiện SELECT

- 1. **FROM** và **JOIN** - bảng
- 2. **WHERE** - điều kiện
- 3. **GROUP BY** – cột hoặc biểu thức (trong đó mệnh đề được trích xuất nhóm dữ liệu)
- 4. **HAVING** – nhóm điều kiện
- 5. **SELECT** - tên cột
- 6. **DISTINCT**
- 7. **ORDER BY** – sắp xếp
- 8. **LIMIT / OFFSET**



Mệnh đề WHERE

- Ví dụ

```
SELECT * FROM `movies` WHERE `category_id` = 2 AND `year_released` = 2008;
```

```
SELECT * FROM `movies` WHERE `category_id` = 1 OR `category_id` = 2;
```

```
SELECT * FROM `movies` WHERE `membership_number` IN (1,2,3);
```

```
SELECT * FROM `movies` WHERE `membership_number` NOT IN (1,2,3);
```

```
SELECT * FROM `movies` WHERE `category_id`<> 1;
```

Bài 1

Giới thiệu về SQL

1.1. Tạo bảng bằng DDL

1.2. Quản lý dữ liệu bảng với DML

1.3. Truy vấn dữ liệu bằng SQL

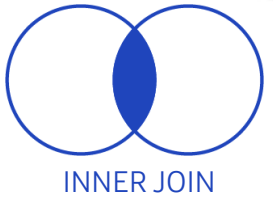
JOIN

- Join được sử dụng để kết hợp các cột từ một (tự tham gia) hoặc nhiều bảng dựa trên giá trị của các cột chung giữa các bảng có liên quan.
  - Các cột phổ biến thường là các cột khóa chính của bảng đầu tiên và các cột khóa ngoại của bảng thứ hai.
- Nó đề cập đến việc xuất dữ liệu bằng cách liên kết hoặc kết hợp hai hoặc nhiều bảng.
- PostgreSQL hỗ trợ phép nối bên trong, phép nối trái, phép nối phải, phép nối ngoài đầy đủ, phép nối chéo, phép nối tự nhiên và một loại phép nối đặc biệt gọi là tự nối.

a_id [PK] số nguyên	**animal** Ký tự khác nhau(50)
10	Sư tử
20	Voi
30	Hổ
40	Ngựa

b_id [PK] số nguyên	**animal** ký tự khác nhau(50)
10	Sư tử
20	Khỉ
30	Ngựa
50	Voi

Inner JOIN (phép nối trong)



- Inner JOIN
  - Phương pháp này được sử dụng khi các giá trị cột giữa hai bảng khớp chính xác với nhau.
  - Điều kiện của JOIN được mô tả trong mệnh đề WHERE và toán tử = được sử dụng.
  - Ví dụ

```
SELECT  a_id, a_animal, b_id, b_animal
FROM  set_a
INNER JOIN set_b
ON a_animal = b_animal;
```

a_id số nguyên	**a_animal** ksy tự thay đổi(50)	**b_id** số nguyên	**b_animal** ký tự thay đổi(50)
1	Sư tử	10	Sư tử
2	Voi	50	Voi
7	Ngựa	30	Ngựa

Outer JOIN – LEFT Outer Join (Phép nối ngoài trái)

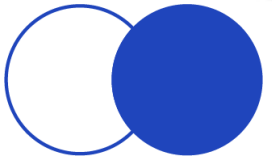

LEFT Outer Join

- Nối ngoài được sử dụng để trả về kết quả bằng cách kết hợp các hàng từ hai hoặc nhiều bảng
- Trả về mọi hàng từ một bảng đã chỉ định, ngay cả khi điều kiện nối
- **LEFT** Outer Join
  - Dữ liệu tương ứng với bảng bên trái được đánh dấu được đọc trước, sau đó dữ liệu mục tiêu JOIN được đọc từ bảng bên phải
  - NULL nếu không có dữ liệu thỏa mãn trong bảng bên phải
  - Ví dụ

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
LEFT JOIN set_b
ON a_animal = b_animal;
```

a_id số nguyên	**a_animal** ký tự thay đổi (50)	**b_id** số nguyên	**b_animal** ký tự thay đổi (50)
1	Sư tử	10	Sư tử
2	Voi	50	Voi
3	Hổ	[null]	[null]
7	Ngựa	30	Ngựa

Outer JOIN – RIGHT Outer Join (Phép nối ngoài phải)


RIGHT Outer Join

- **RIGHT** Outer Join
  - Khi thực hiện phép nối, dữ liệu tương ứng trước tiên được đọc từ bảng bên phải được đánh dấu trước, sau đó dữ liệu đích JOIN được đọc từ bảng bên trái
  - NULL nếu không có dữ liệu thỏa mãn trong bảng bên trái
  - Ví dụ

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
RIGHT JOIN set_b
ON a_animal = b_animal;
```

a_id integer	a_animal character varying (50)	b_id integer	b_animal character varying (50)
1	Sư tử	10	Sư tử
[null]	[null]	20	Khỉ
7	Ngựa	30	Ngựa
2	Voi	50	Voi

Outer JOIN – FULL Outer Join

FULL Outer Join

- **FULL** Outer Join
  - Khi thực hiện nối, tất cả dữ liệu trong bảng bên trái và bên phải được đọc và kết quả được tạo bằng cách nối
  - Ví dụ

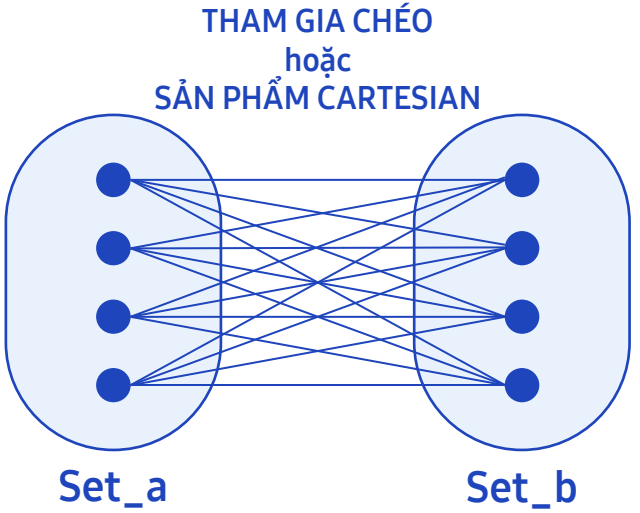
```
SELECT  a_id, a_animal, b_id, b_animal
FROM  set_a
FULL JOIN set_b
ON a_animal = b_animal;
```

a_id số nguyên	**a_animal** ký tự thay đổi (50)	**b_id** số nguyên	**b_animal** ký tự thay đổi (50)
1	Sư tử	10	Sư tử
2	Voi	50	Voi
3	Hổ	[null]	[null]
7	Ngựa	30	Ngựa
[null]	[null]	20	Khỉ

Cross JOIN (THAM GIA chéo)

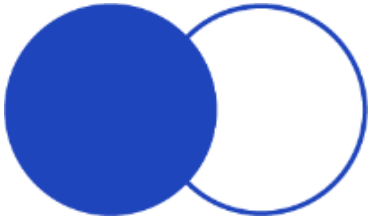
- Phép nối chéo là một kiểu phép nối trả về tích Descartes của các hàng từ các bảng trong phép nối.
- Nói cách khác, nó kết hợp từng hàng từ bảng đầu tiên với từng hàng từ bảng thứ hai.
- Để có kết quả CROSS JOIN cho hai bảng, M * N dữ liệu kết hợp của cả hai bộ.
- Tải trọng lớn (Thận trọng)
  - Ví dụ

```
SELECT a_id, a_animal, b_id, b_animal
FROM set_a
CROSS JOIN set_b
```

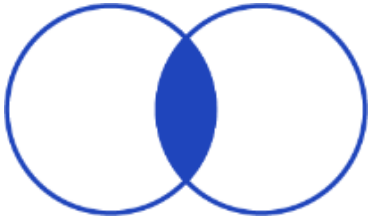


JOIN Operations (Phép nối)

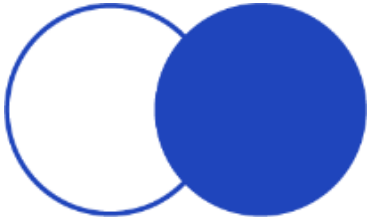
LEFT JOIN b ON a.key = b.key



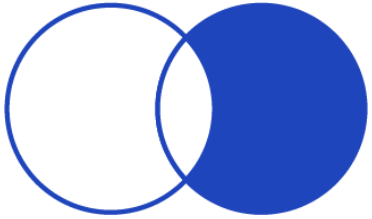
INNER JOIN b ON a.key = b.key



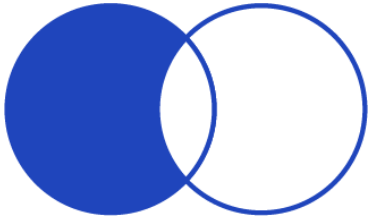
RIGHT JOIN b ON a.key = b.key



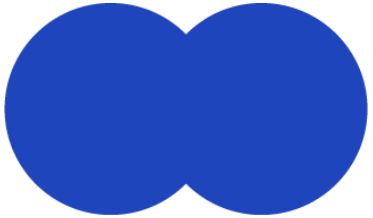
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL



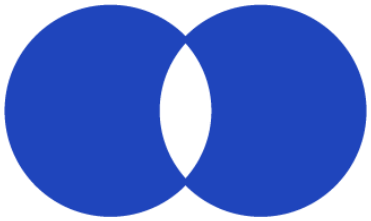
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL



FULL JOIN b ON a.key = b.key



FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL



Toán tử so sánh và logic

- Để hình thành điều kiện trong mệnh đề **WHERE**, bạn sử dụng các toán tử so sánh và logic

Toán tử	Mô tả
=	Bằng
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng
<>or!=	không bằng
AND	Toán tử logic AND
OR	Toán tử logic OR

- Sự ưu tiên
  - NOT > AND > OR
  - dấu ngoặc đơn nếu cần thiết

LIKE & Wildcards

- Like và Wildcards các công cụ mạnh mẽ giúp tìm kiếm dữ liệu khớp với các mẫu phức tạp.
- Có một wildcard bao gồm tỷ lệ phần trăm, dấu gạch dưới và danh sách xếp hạng (không được MySQL hỗ trợ) trong số những ký tự khác
- Wildcard phần trăm được sử dụng để khớp với bất kỳ số lượng ký tự nào bắt đầu từ số không (0) trở lên.
- Wildcard gạch dưới được sử dụng để khớp với chính xác một ký tự.
- Từ khóa Escape
  - Từ khóa ESCAPE được sử dụng để loại trừ các ký tự khớp mẫu khi các ký tự như tỷ lệ phần trăm (%) và dấu gạch dưới (_) tạo thành một phần của dữ liệu.
  - Ví dụ Để tìm kiếm "67% Guilty"

```
SELECT * FROM movies WHERE title LIKE '67#%%' ESCAPE '#';
```

```
SELECT * FROM movies WHERE title LIKE '67=%%' ESCAPE '=';
```

Regular Expressions (REGEXP): Biểu thức chính quy

- Biểu thức chính quy tìm kiếm dữ liệu phù hợp với tiêu chí phức tạp
- So với ký tự đại diện, biểu thức chính quy truy xuất dữ liệu phù hợp với tiêu chí phức tạp hơn nhiều
  - Cú pháp

```
SELECT statement ... WHERE fieldname REGEXP 'pattern';
```

  - Ví dụ

```
SELECT * FROM `movies` WHERE `title` REGEXP '^ [abcd]';
```

BETWEEN & IN

- **BETWEEN** - toán tử xuất ra một tập hợp nằm trong một phạm vi cụ thể
  - Biểu thức BETWEEN giá trị 1 AND giá trị 2

```
SELECT * FROM price_table WHERE price BETWEEN 200 AND 280;
```

```
SELECT * FROM price_table WHERE price >= 200 AND price  <= 280;
```

- **IN** - một toán tử xác định xem một tập hợp hoặc danh sách cụ thể có tồn tại trong một tập hợp cụ thể (cột hoặc danh sách) hay không.
  - Sử dụng toán tử IN trong mệnh đề **WHERE** để kiểm tra xem một giá trị có khớp với bất kỳ giá trị nào trong danh sách giá trị hay không.
- Value **IN** (value1, value2, ...)
  - Toán tử để kiểm tra xem giá trị value1 và value2 có tồn tại trong tập hợp cột không

GROUP BY

- fMệnh đề **GROUP BY** của PostgreSQL được sử dụng để phân chia các hàng được trả về bởi câu lệnh **SELECT** thành các nhóm khác nhau.
- Thực hiện chức năng tổng hợp cho từng nhóm
- Nếu bạn tạo một nhóm có nhiều cột, hãy liệt kê chúng với dấu “,”
- Viết ngay sau mệnh đề **FROM** và **WHERE**
  - Cú pháp

```
SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING condition];
```

  - Ví dụ

```
SELECT col_1, sum(col_2) FROM table GROUP BY col_1
```

- Câu lệnh **SELECT** được sử dụng trong mệnh đề **GROUP BY** chỉ có thể chứa tên cột, hàm tổng hợp, hằng số và biểu thức.
- Mệnh đề có SQL được sử dụng để giới hạn kết quả trả về bởi mệnh đề **GROUP BY**.

Aggregate functions

- Các hàm tổng hợp thực hiện phép tính trên một tập hợp các hàng và trả về một hàng.
  - **AVG()** – trả về giá trị trung bình.
  - **COUNT()** – trả về số lượng giá trị.
  - **MAX()** – trả về giá trị lớn nhất.
  - **MIN()** – trả về giá trị nhỏ nhất.
  - **SUM()** – trả về tổng của tất cả hoặc các giá trị riêng biệt.

Hàm Single-row

- Nó có thể được sử dụng trong các mệnh đề **SELECT**, **WHERE**, **ORDER BY**.
- Nó hoạt động trên từng hàng riêng lẻ để thao tác các giá trị dữ liệu và trả về kết quả hoạt động cho từng hàng.
- Ngay cả khi nhiều đối số được nhập vào, chỉ có một kết quả được trả về.
- Hàm single row có thể là hàm ký tự, hàm số, hàm ngày tháng và hàm chuyển đổi.
- Lưu ý rằng các chức năng này được sử dụng để thao tác các mục dữ liệu.

Hàm Single-row

- Các hàm Chuyển đổi trường hợp - Chấp nhận đầu vào ký tự và trả về một giá trị ký tự
  - Các chức năng trong danh mục là **UPPER**, **LOWER** và **INITCAP**
    - **Hàm UPPER** chuyển đổi một chuỗi thành chữ hoa
    - **Hàm LOWER** chuyển đổi một chuỗi thành chữ thường
    - **Hàm INITCAP** chỉ chuyển đổi các chữ cái đầu tiên của chuỗi thành chữ hoa
- Hàm ký tự - Chấp nhận đầu vào ký tự và trả về số hoặc giá trị ký tự.
  - Các hàm trong danh mục là **CONCAT**, **LENGTH**, **SUBSTR**, **INSTR**, **LPAD**, **RPAD**, **TRIM** và **REPLACE**
    - Hàm **CONCAT** nối hai giá trị chuỗi
    - Hàm **LENGTH** trả về độ dài của chuỗi đầu vào
    - Hàm **SUBSTR** trả về một phần của chuỗi từ điểm bắt đầu cho trước đến điểm kết thúc
    - Hàm **INSTR** trả về vị trí số của một ký tự hoặc một chuỗi trong một chuỗi đã cho.
    - Các hàm **LPAD** và **RPAD** đệm chuỗi đã cho tối đa một độ dài cụ thể với một ký tự nhất định.
    - Hàm **TRIM** cắt đầu vào chuỗi từ đầu hoặc cuối.
    - Hàm **REPLACE** thay thế các ký tự từ chuỗi đầu vào bằng một ký tự đã cho.

Hàm SUBSTRING( )

- Hàm chuỗi con PostgreSQL được sử dụng để trích xuất một chuỗi chứa một số ký tự cụ thể từ một vị trí cụ thể của một chuỗi đã cho.
  - Trích xuất số lượng ký tự cụ thể từ vị trí cụ thể
  - Cú pháp

```
SUBSTRING(string [from starting_position] [for length])
```

  - Ví dụ

```
SELECT substring('jsjeong' for 2);
```

▷ Kết quả là “js”

```
SELECT substring('jsjeong' from 3 for 7);
```

▷ Kết quả là ”jeong”

SUBQUERY (Truy vấn con)

- Câu lệnh SELECT được lồng trong một câu lệnh SQL
- Một truy vấn trong đó một truy vấn khác được lồng trong một truy vấn
- **JOIN** vs. **SUBQUERY**
  - **JOIN** : Tập hợp dữ liệu từ nhiều bảng.
  - **SUBQUERY**: Xử lý nhiều câu lệnh sql thành một (tất cả DML đều có thể)
  - Ví dụ

```
SELECT select_list
FROM table condition or View
WHERE condition operator (SELECT select_list
FROM table
WHERE condition);
```

[Lab1]
Bắt đầu với MariaDB

[Lab2]
Làm việc với bảng SQL

[Lab3]
Làm việc với bảng

[Lab4]
Làm việc với truy vấn

Bài 2.

Phân tích cơ bản

Phân tích Big Data

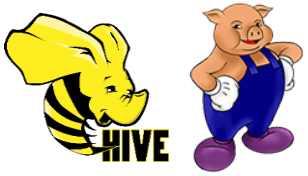
Bài 2

Phân tích cơ bản

Phân tích và xử lý dữ liệu

Hadoop MapReduce và Spark

- ✓ Xử lý và phân tích dữ liệu với Hadoop
- ✓ Công cụ xử lý dữ liệu đẹp nhưng quá khó đối với người dùng bình thường



Pig và Hive

- ✓ Tóm tắt cấp cao hơn để xử lý dữ liệu chung

Impala

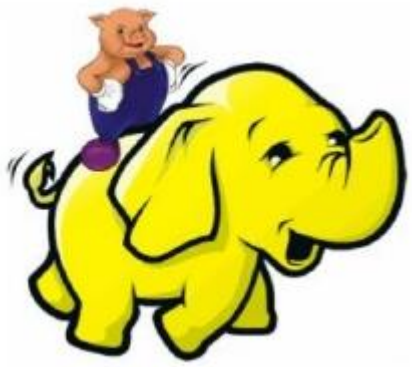
- ✓ Phân tích tương tác với công cụ thực thi

Apache Pig / Hive / Impala

- Pig
  - Thay thế cho việc viết mã java để xử lý Big data.
  - Cung cấp xử lý dữ liệu cấp cao
  - Đặc biệt giỏi tiền xử lý dữ liệu cho ETL với dữ liệu phi cấu trúc.
- Apache Hive
  - Một sự trừu tượng hóa khác trên Hadoop với SQL
  - Hive sử dụng ngôn ngữ giống như SQL, HiveQL
- Apache Impala
  - Một công cụ SQL song song lớn trên cụm Hadoop
  - Sử dụng impala SQL như HiveQL
  - Ngôn ngữ truy vấn cấp cao

Pig là gì?

- Nó là một công cụ để thực thi các chương trình trên Hadoop
- Pig là một giải pháp thay thế cho việc viết mã MapReduce cấp thấp.
- Nó cung cấp một ngôn ngữ, Pig Latin, để xác định các chương trình này
- Trường hợp sử dụng
  - Lấy mẫu dữ liệu
  - Được sử dụng để xử lý trích xuất, chuyển đổi và tải (ETL)
  - Trích xuất dữ liệu có giá trị từ nhật ký



Tại sao lại là Pig?

- Pig (Lợn) ăn bất cứ thứ gì
  - Pig có thể xử lý bất kỳ dữ liệu nào, có cấu trúc hoặc không cấu trúc
- Pig (Lợn) sống ở bất cứ đâu
  - Pig có thể chạy trên mọi khung xử lý dữ liệu song song
- Ngôn ngữ luồng dữ liệu là ngôn ngữ cấp cao
- Một ngôn ngữ thủ tục và nó phù hợp với mô hình đường ống
- Xử lý dữ liệu có cấu trúc, phi cấu trúc và bán cấu trúc
- Không cần biên dịch, khi thực thi toán tử pig được chuyển đổi nội bộ thành Công việc MapReduce

Pig VS MapReduce

Sự khác biệt chính giữa Apache Pig và MapReduce

Apache Pig	MapReduce
Apache Pig là một ngôn ngữ luồng dữ liệu.	MapReduce là một mô hình xử lý dữ liệu.
Nó là một ngôn ngữ cấp cao.	MapReduce ở mức thấp và cứng nhắc.
Thực hiện thao tác Join trong Apache Pig khá đơn giản.	MapReduce khá khó thực hiện thao tác Join giữa các tập dữ liệu.
Bất kỳ lập trình viên mới làm quen nào có Kiến thức cơ bản về SQL đều có thể làm việc thuận tiện với Apache Pig.	Tiếp xúc với Java là phải làm việc với MapReduce.
Apache Pig uses multi-query approach, Thereby reducing the length of the codes to a great extent.	MapReduce sẽ yêu cầu số dòng gấp gần 20 lần để thực hiện cùng một tác vụ.
Không cần biên dịch. Khi thực thi, mọi toán tử Apache Pig được chuyển đổi nội bộ thành công việc MapReduce.	Các công việc MapReduce có một quá trình biên dịch dài.

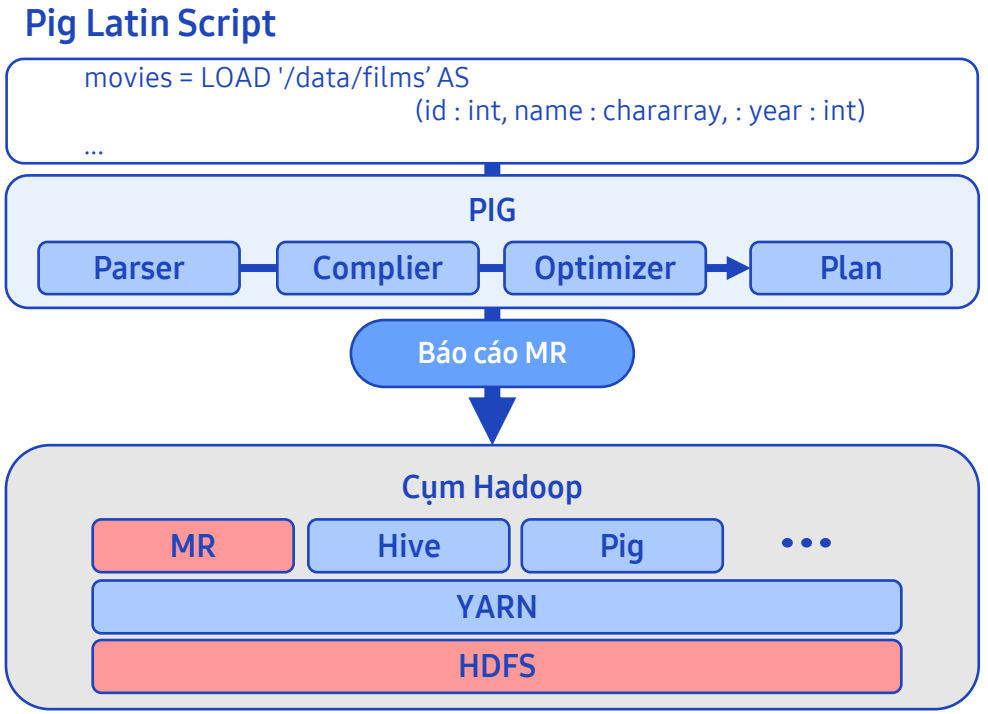
Pig VS SQL

Sự khác biệt chính giữa Apache Pig và SQL.

Pig	SQL
Pig Latin là một ngôn ngữ thủ tục.	SQL là ngôn ngữ khai báo.
Trong Apache Pig, lược đồ là tùy chọn. Chúng ta có thể lưu trữ dữ liệu bằng cách thiết kế lược đồ (các giá trị được lưu trữ dưới dạng $01, $02, v.v.)	Lược đồ là bắt buộc trong SQL.
Mô hình dữ liệu trong Apache Pig là quan hệ lồng nhau	Mô hình dữ liệu được sử dụng trong SQL là quan hệ phẳng.
Nó cung cấp cơ hội hạn chế để tối ưu hóa Truy vấn.	Có nhiều cơ hội hơn để tối ưu hóa truy vấn là SQL.

Thành phần và kiến trúc Pig

- Pig Latin – ngôn ngữ luồng dữ liệu
- Grunt – shell tương tác, nơi bạn có thể nhập các câu lệnh Latin lợn
- Công cụ thông dịch và thực thi Pig



Grunt Shell

- Grunt shell là shell tương tác của Pig.
- Các grunt shell của Apache Pig chủ yếu được sử dụng để viết các chữ viết bằng tiếng Latinh của Pig.
- Tập lệnh pig có thể được thực thi với grunt shell là shell gốc do Apache pig cung cấp để thực thi các truy vấn pig.
- Hữu ích cho việc kiểm tra dữ liệu đặc biệt
- Quá trình thực thi bị trì hoãn cho đến khi yêu cầu đầu ra
- Cách bắt đầu và thoát

```
$ pig
grunt> sh ls
grunt> fs –ls
grunt> quit;
```

Các tập lệnh pig

- Tập lệnh Pig chỉ đơn giản là mã Pig Latin được lưu trữ trong tệp văn bản (phần mở rộng .pig)
  - Viết tất cả các câu lệnh Pig Latin cần thiết trong một tệp.
  - Viết tất cả các lệnh và câu lệnh Pig Latin trong một tệp duy nhất và lưu dưới dạng tệp . tệp heo.
  - Thực thi tập lệnh Apache Pig (sales_report.pig)

```
$ pig
grunt› run sales_report.pig;
grunt› quit;

$ pig sales_report.pig
```

Pig Latin (1/5)

- Pig Latin là ngôn ngữ luồng dữ liệu
  - Luồng dữ liệu được thể hiện dưới dạng một chuỗi các câu lệnh
- Chú thích
  - Một dòng: –
  - Đa dòng: /* */
- Định danh
  - Định danh là tên được gán cho các trường và cấu trúc dữ liệu khác
  - Mã định danh phải luôn bắt đầu bằng một chữ cái
  - Từ khóa (bằng văn bản màu xanh) không phân biệt chữ hoa chữ thường
  - Định danh và đường dẫn (bằng văn bản màu đỏ) phân biệt chữ hoa chữ thường

```
allsales = LOAD ‘sales’ AS (name, price) ;
bigsales = FILTER allsales BY price > 999 ;
STORE bigsales INTO ‘myreport’ ;
```

Pig Latin (2/5)

Toán tử

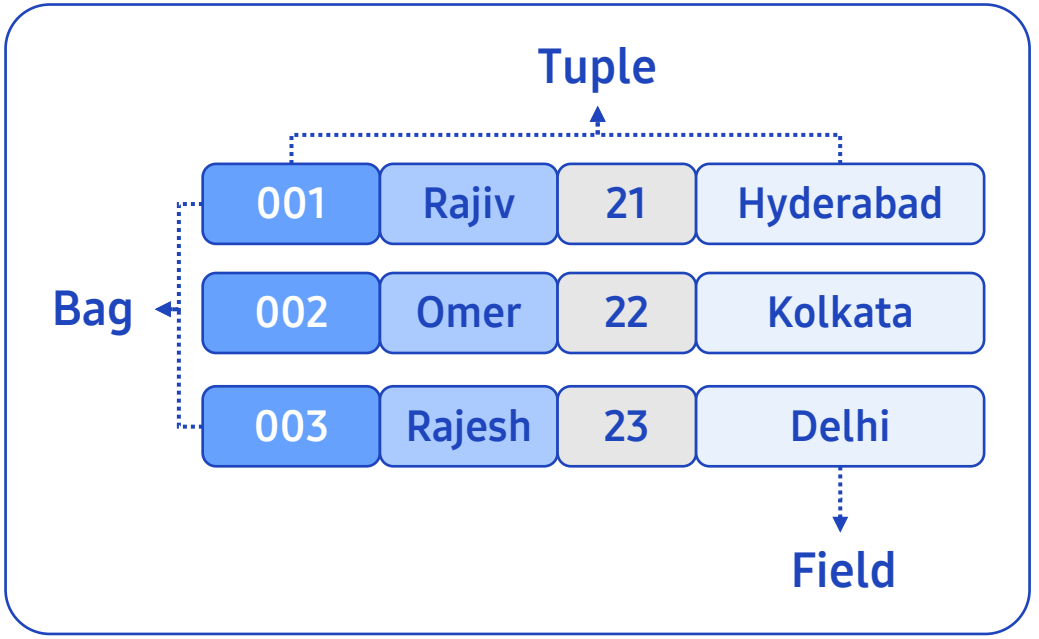
Môn số học	So sánh	Null	Boolean
+	==	IS NULL	AND
-	! =	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

Pig Latin (3/5)

Loai dữ liệu	Mô tả	Ví dụ
int	Số nguyên 32 bit đã ký	10
long	Số nguyên 64 bit đã ký	Dữ liệu: 10L hoặc 10l Hiển thị: 10L
float	Dấu chấm động 32-bit	Dữ liệu: 10,5F hoặc 10,5f hoặc 10,5e2f hoặc 10,5E2F Hiển thị: 10,5F hoặc 1050,0F
double	Dấu chấm động 64-bit	Dữ liệu: 10,5 hoặc 10,5e2 hoặc 10,5E2 Hiển thị: 10.5 hoặc 1050.0
chararray	Mảng ký tự (chuỗi) ở định dạng Unicode UTF-8	hello world
bytearray	Mảng byte (blob)	
boolean	boolean	đúng/sai (không phân biệt chữ hoa chữ thường)
datetime	Datetime	1970-01-01 T00:00:00.000+00:00

Pig Latin (4/5)

- Tuples
  - Một bộ sưu tập các giá trị được gọi là một tuple
- Bags (Túi)
  - Một tập hợp các tuple được gọi là một bag
  - Các tuple trong một bag không được sắp xếp theo mặc định
- Fields (Trường)
  - Một phần tử của dữ liệu được gọi là một trường



Pig Latin (5/5)

- Một quan hệ là kết quả của một bước xử lý, đơn giản là một túi có tên được gán (bí danh)
- Tên được đặt cho một quan hệ được gọi là bí danh
- Ví dụ: result_2cols là bí danh:

```
grunt> allsales = LOAD 'sales.csv' USING PigStorage(',') AS (name, price);
grunt> describe all_sales;
grunt> dump all_sales;
```

LOAD (Tải)

- Tải dữ liệu
- Chức năng tải mặc định của Pig được gọi là PigStorage
  - Tên hàm ẩn khi gọi LOAD
  - PigStorage giả định định dạng văn bản với các cột được phân tách bằng tab
- Đường dẫn cũng có thể tham chiếu đến một thư mục (tải đệ quy tất cả các tệp trong thư mục đó)
  - Ví dụ

```
grunt> allsales = LOAD 'sales' AS (name, price);
grunt> allsales = LOAD 'sales';
grunt> allsales = LOAD 'sales_200[5-9]' AS (name, price);
grunt> allsales = LOAD 'sales.csv' USING PigStorage(',') AS (name, price);
```

PigStorage

- Hàm load phân tích một dòng đầu vào thành các trường bằng cách sử dụng dấu phân cách ký tự
  - Dấu phân cách mặc định là tab
- Sử dụng định dạng tệp văn bản được phân tách
- Dấu phân cách mặc định (Tab) có thể được thay đổi thành (,)
  - Ví dụ

```
grunt> result_2cols = load ‘mydata.log’ AS (name, grade)
grunt> Describe result_2cols;
Result_2cols: {name: bytearray, grade: bytearray}
grunt> Dump
```

- Pigstorage cũng có thể được áp dụng khi sử dụng store() để lưu kết quả

DUMP & STORE

- Lệnh thực thi mã hiện tại
- **DUMP** : gửi dữ liệu ra màn hình
- **STORE** : gửi đầu ra vào đĩa (HDFS)
  - Dấu phân cách trường cũng có giá trị mặc định (tab)
  - Cú pháp

```
STORE relation_name INTO ' required_directory_path ' [USING function];
```

  - Ví dụ

```
STORE bigsales INTO ‘myreport’;
```

```
STORE bigsales INTO ‘myreport’ USING PigStorage(‘,’);
```

DESCRIBE (Mô tả)

- Hiển thị cấu trúc của dữ liệu, bao gồm tên và loại

```
grunt> result_2cols = load ‘mydata.log’ as (name, grade)
grunt> Describe result_2cols;
Result_2cols: {name: bytearray, grade: bytearray}
```

- Xác định lược đồ
  - Xác định kiểu dữ liệu cho quan hệ được tạo.
  - Điều này cho phép bạn xác định loại mong muốn cho từng trường khác với loại mặc định.

```
grunt> result_2cols = load ‘mydata.log’ as (name: chararray,  grade: int)
grunt> Describe result_2cols;
Result_2cols: {name: chararray, grade: int}
```

FILTER (Lọc)

- Trích xuất các tuple phù hợp với tiêu chí đã chỉ định
- Ví dụ

```
A = FILTER salaries BY (age > 50 AND salary < 60000) OR zip == '95103'
```

Lương → A?

Giới tính	Tuổi	Lương	Zip
M	66	41000.00	95103
M	58	76000.00	57701
F	40	95000.00	95102
M	45	60000.00	95105
F	28	55000.00	95103

FOREACH...GENERATE

- Toán tử làm việc trên từng bản ghi trong tập dữ liệu (như trong “cho từng bản ghi”).
- Kết quả của **FOREACH** là một tuple mới, thường là với một lược đồ khác.

```
A = FOREACH salaries GENERATE age, salary;
```

Lương

Giới tính	Tuổi	Lương	Zip
M	66	41000.00	95103
M	58	76000.00	57701
F	40	95000.00	95102
M	45	60000.00	95105
F	28	55000.00	95103

A

Tuổi	Lương
66	41000.00
58	76000.00
40	95000.00
45	60000.00
28	55000.00

FOREACH...GENERATE

- Để tạo các trường
  - Tạo một trường mới dựa trên giá

```
T = FOREACH salaries GENERATE salary * 0.07;
```

  - Có thể đặt tên cho các lĩnh vực như vậy

```
T = FOREACH salaries GENERATE salary * 0.07 as tax;
```

  - Chỉ định loại dữ liệu

```
T = FOREACH salaries GENERATE salary * 0.07 as tax: float;
```

DISTINCT / LIMIT

- Loại bỏ các bản ghi trùng lặp trong một bag

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> unique_2cols = DISTINCT result_2cols;
```

- LIMIT
  - Giảm số lượng bản ghi đầu ra
  - Cú pháp

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

  - Ví dụ

```
grunt> unique_2cols = DISTINCT result_2cols;
grunt> limit10_2cols = LIMIT unique_2cols 10;
```

ORDER BY

- Được sử dụng để hiển thị nội dung của một quan hệ theo thứ tự được sắp xếp dựa trên một hoặc nhiều trường
- Thứ tự bản ghi là ngẫu nhiên trừ khi được chỉ định với **ORDER BY**
- Dưới đây là ví dụ về toán tử **ORDER BY** (TOP 5)
  - Sắp xếp tăng dần theo cấp (ASC: thứ tự tăng dần, DESC: thứ tự giảm dần)

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> sorted_2cols = ORDER result_2cols BY grade DESC;
grunt> top_five = LIMIT sorted_2cols 5;
```

GROUP

- Nhóm dữ liệu trong một hoặc nhiều quan hệ dựa trên một khóa cụ thể
  - Có thể nhóm ngay cả đối với nhiều khóa
  - Ví dụ

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> group_2cols_by_name = GROUP result_2cols BY name;
```

- **GROUP All**
  - Bạn có thể nhóm một mối quan hệ theo tất cả các cột
  - Ví dụ

```
grunt> result_2cols = load 'mydata.log' as (name: chararray, grade: int)
grunt> group_all = GROUP result_2cols All;
```

UNION

- Dùng để hợp nhất hai quan hệ có cùng cấu trúc
  - Cú pháp

```
grunt> Relation_name3 = UNION Relation_name1, Relation_name2;
```

  - Ví dụ

```
grunt> data_june = load 'mydata_jun.log' as (name: chararray, grade: int)
grunt> data_july = load 'mydata_jul.log' as (name: chararray,  grade: int)
grunt> data_union = UNION data_june, data_july;
```

Built-in Functions (Hàm tích hợp sẵn)

- Các hàm toán học – ABS(số tiền), SUM(số tiền), AVERAGE(số tiền), MINIMUM(số tiền), MAX(số tiền)
- Hàm chuỗi – CONCAT(a, b), REPLACE(a, '-', '/')
- Chức năng ngày và giờ

Mô tả hàm	Yêu cầu ví dụ	Dữ liệu nhập	Dữ liệu xuất
Convert to uppercase	UPPER(quốc gia)	UK	UK
Remove leading/trailing spaces	TRIM(tên)	Bob	Bob
Return a random number	RANDOM()		0.4816132 6652569
Round to closest whole number	ROUND(giá)	37.19	37
Return chars between two positions	SUBSTRING(tên, 0, 2)	Alice	Al

Câu hỏi ôn tập

[Câu hỏi]

Câu hỏi 1

- Một số thành phần chính của Pig là gì?

Câu hỏi 2

- Sự khác biệt giữa chế độ MapReduce và chế độ Cục bộ là gì?

Câu hỏi 3

- Nếu bạn không chỉ định loại dữ liệu được tải, Pig sẽ sử dụng loại dữ liệu nào làm loại mặc định?

[Lab5]
Sử dụng Pig để xử lý ETL (Tiền xử lý)

Bài 2

Phân tích cơ bản

2.1. Tiền xử lý dữ liệu và phân tích dữ liệu cơ bản với Apache Pig

2.2. Truy vấn cơ bản với Apache Hive và Impala

HIVE là gì?

- Một hệ thống truy vấn và quản lý dữ liệu có cấu trúc được xây dựng trên Hadoop
  - Sử dụng Map-Reduce để thực hiện
  - HDFS để lưu trữ - nhưng bất kỳ hệ thống nào triển khai API Hadoop FS
- Các nguyên tắc xây dựng chính:
  - Dữ liệu có cấu trúc với các loại dữ liệu phong phú (cấu trúc, danh sách và bản đồ)
  - Truy vấn trực tiếp dữ liệu từ các định dạng khác nhau (văn bản/nhị phân) và định dạng tệp (Phẳng/Tuần tự)
  - SQL như một công cụ lập trình quen thuộc và để phân tích tiêu chuẩn
  - Cho phép các tập lệnh nhúng để có khả năng mở rộng và cho các ứng dụng không chuẩn
  - Siêu dữ liệu phong phú để cho phép khám phá dữ liệu và tối ưu hóa

Hive cung cấp những gì?

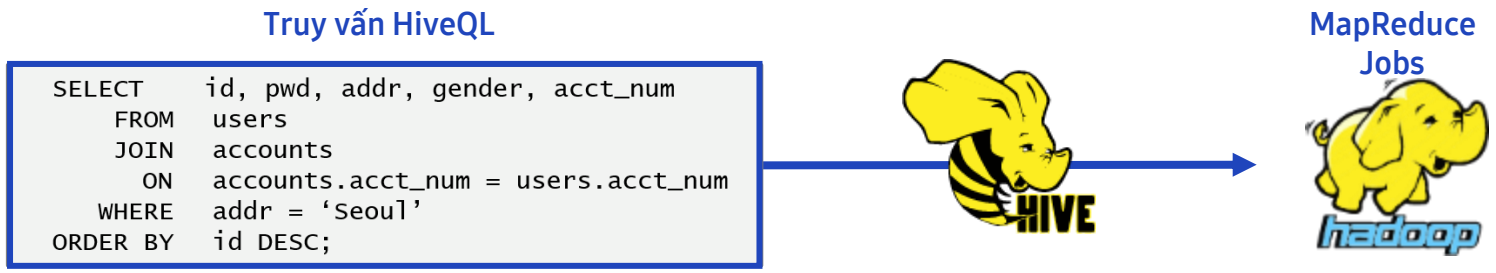
- Hive cho phép người dùng truy vấn dữ liệu bằng HiveQL, một ngôn ngữ rất giống với SQL tiêu chuẩn
- Hive biến các truy vấn HiveQL thành MapReduce tiêu chuẩn
  - Tự động chạy công việc và hiển thị kết quả
- Hive KHÔNG phải là một RDBMS
  - Kết quả của các truy vấn được chuyển đổi thành công việc MapReduce và có thể mất khá nhiều thời gian để hoàn thành
  - Bạn không thể sửa đổi dữ liệu ở cấp hàng

Tại sao lại là Hive?

- MapReduce rất khó lập trình.
- Các nhà phân tích của Facebook là chuyên gia SQL
- Được phát triển để tận dụng khả năng xử lý dữ liệu khổng lồ của Hadoop và kinh nghiệm nội bộ của chuyên gia SQL hiện có.

Đặc điểm của Apache Hive

- Một công cụ cấp cao sử dụng cú pháp giống như SQL để truy vấn HDFS
- Khi xử lý Hadoop, nó được thay đổi thành Map-Reduce thực tế và được thực thi.
- Sử dụng công cụ Map-Reduce và SPARK
- Dịch vụ xử lý/phân tích dữ liệu do Facebook phát triển
  - Được sử dụng để truy cập dữ liệu HDFS bằng cú pháp giống như SQL
  - Kho dữ liệu trên nền tảng Hadoop
  - Chuẩn bị dữ liệu hàng loạt / ETL



Đặc điểm của từng phiên bản cụ thể

- Hive 1.0
  - Nó dần dần được phát triển bắt đầu với phiên bản 0.10 vào năm 2012 và phiên bản 1.0 được phát hành vào tháng 2 năm 2015.
  - Xử lý **MapReduce** bằng SQL
  - Biểu diễn logic của dữ liệu tệp
  - Nhằm mục đích xử lý hàng loạt Big Data

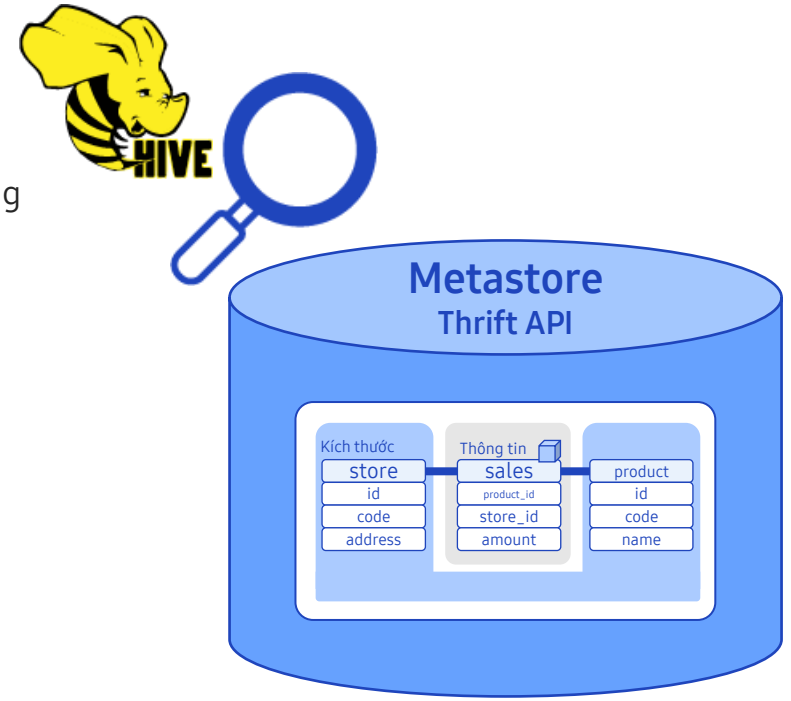
- Hive 2.0
  - Vào tháng 2 năm 2016, phiên bản 2.0 đã được phát hành bằng cách cải thiện Hive 1.0. Với sự ra đời của LLAP và công cụ thực thi mặc định được đổi thành **TEZ**, hiệu suất đã được cải thiện.
  - Thêm cấu trúc Live Long and Process (LLAP)
  - Hỗ trợ Spark nâng cao

Đặc điểm của từng phiên bản cụ thể

- Hive 3.0
  - Vào tháng 5 năm 2018, phiên bản 3.0 đã được phát hành. Nó đã được sửa đổi để **loại bỏ công cụ MapReduce** và Hive CLI và xử lý tác vụ **bằng công cụ TEZ và Beeline.**
  - Quản lý khối lượng công việc bằng vai trò
  - Tăng cường xử lý giao dịch
  - Thêm chế độ xem cụ thể hóa
  - Kết quả truy vấn bộ đệm hoạt động nhanh hơn
  - Thêm cơ sở dữ liệu quản lý thông tin bảng

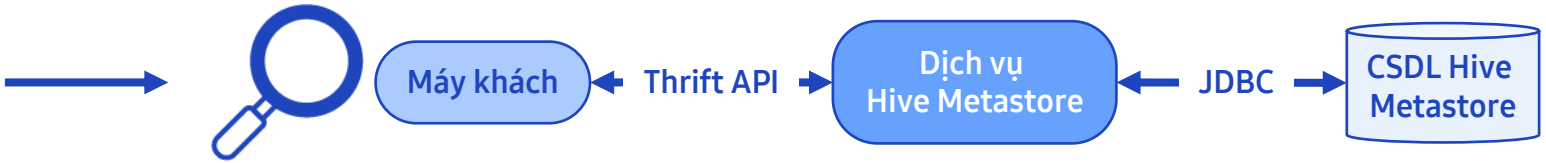
Apache Hive Metastore

- Hive Metastore
  - Lược đồ dữ liệu
  - Các bảng và trường cơ sở dữ liệu
  - Loại trường
  - Thông tin vị trí dữ liệu
  - Lược đồ dữ liệu được quản lý trong cơ sở dữ liệu riêng



Lấy dữ liệu từ Apache Hive

- Truy xuất dữ liệu từ Hive theo hai bước
  - Bước đầu tiên: kiểm tra cấu trúc dữ liệu trong Metastore
    Truy xuất thông tin trường, loại và vị trí trong bảng



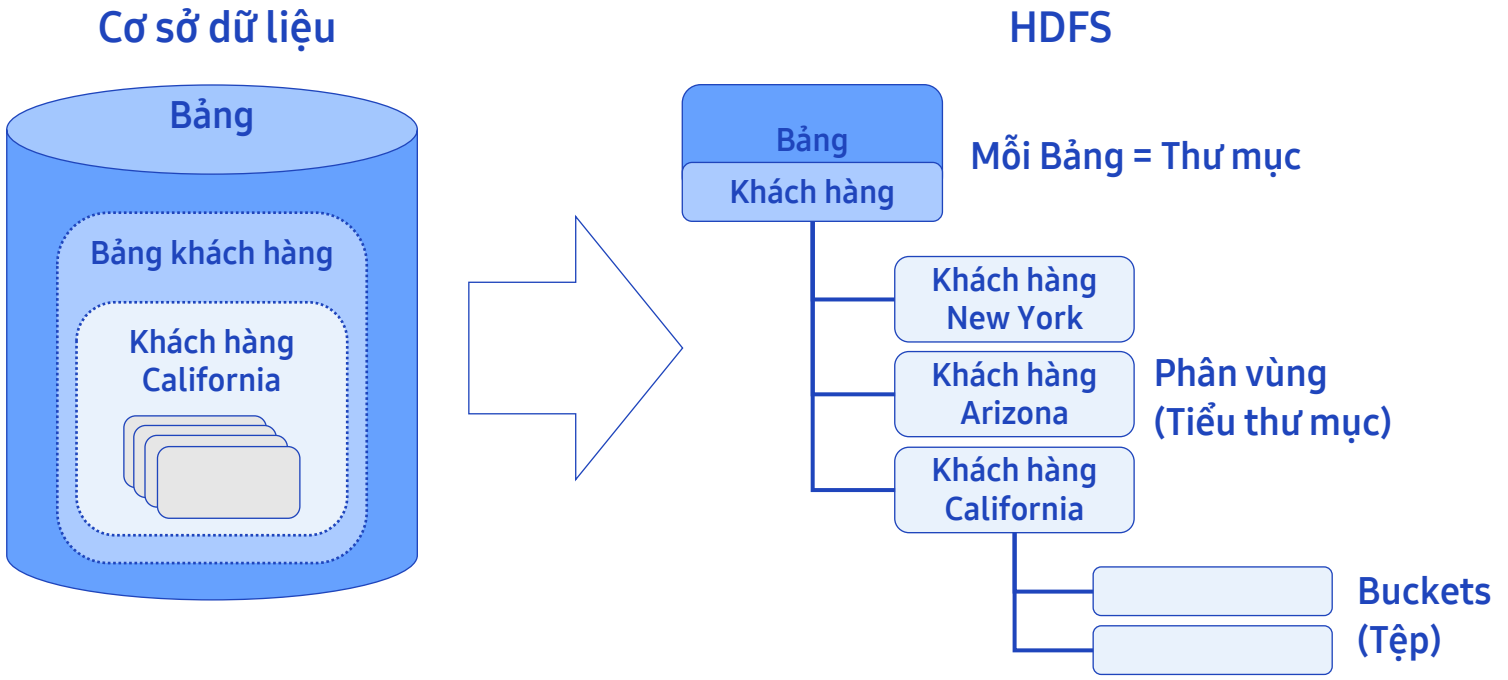
  - Bước thứ hai: dữ liệu thực tế là từ HDFS
    Truy xuất dữ liệu thực trong HDFS

Cách thức hoạt động của Hive (1/3)

- Bảng trong Hive là một thư mục trong HDFS
  - Tất cả các tệp trong thư mục là nội dung của Bảng
  - Thông tin lược đồ và cách các hàng và cột được phân tách được lưu trữ trong Hive Metastore
- Các thư mục con trong thư mục “Bảng” là các phân vùng do người dùng xác định
- Siêu dữ liệu (cấu trúc bảng và vị trí cho dữ liệu) được lưu trữ trong RDBMS
- Các bảng được tạo dưới dạng được quản lý hoặc bên ngoài
  - Quản lý: nếu bảng bị hủy, dữ liệu HDFS sẽ bị xóa
  - Bên ngoài: bảng bị hủy, chỉ lược đồ bảng bị xóa. Dữ liệu không bị xóa

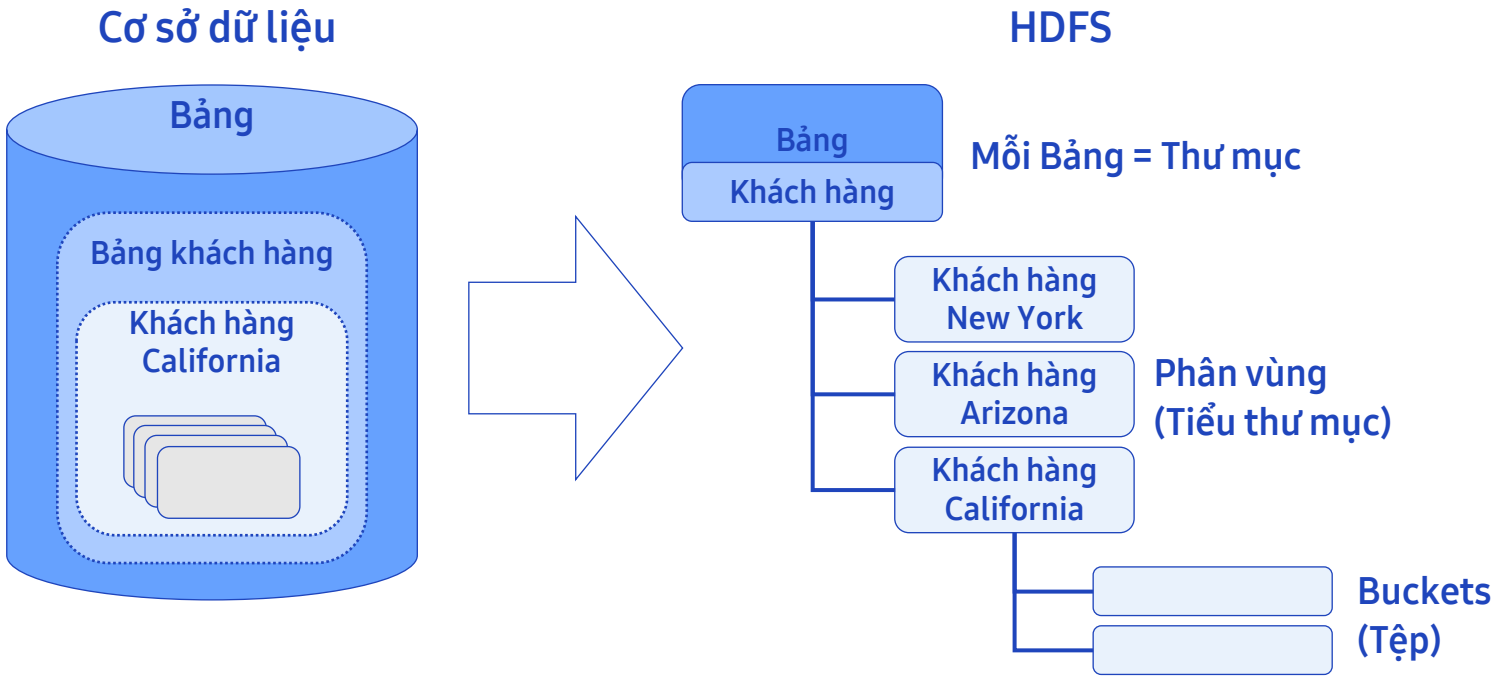
Cách thức hoạt động của Hive (2/3)

- Mô hình dữ liệu Hive và HDFS



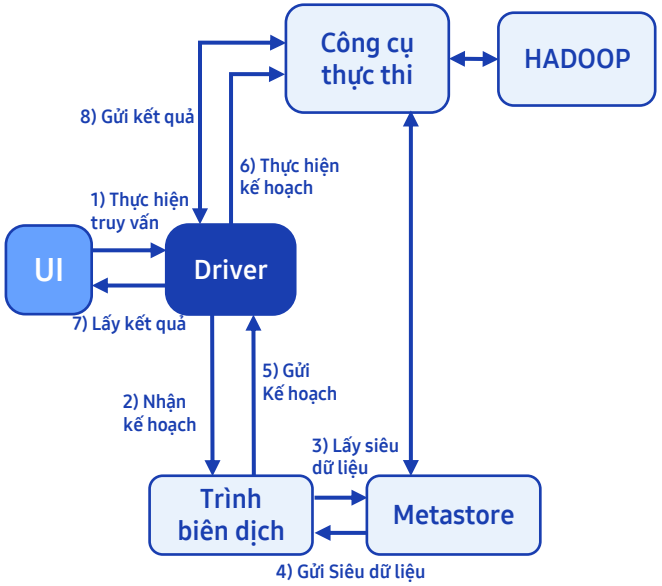
Cách thức hoạt động của Hive (3/3)

- Hive sử dụng metastore để xác định lược đồ dữ liệu và vị trí
- Bản thân truy vấn hoạt động trên dữ liệu được lưu trữ trong hệ thống tệp (thường là HDFS)



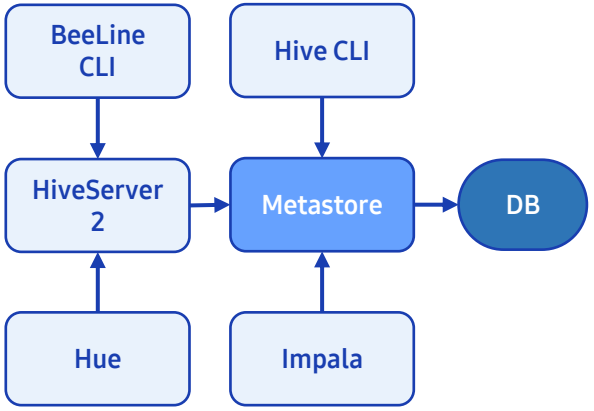
Thành phần HIVE

- UI
  - Giao diện người dùng thông qua đó người dùng gửi truy vấn và các hành động khác cho hệ thống
  - CLI, Beeline, JDBC, v.v.
- Driver
  - Nhận đầu vào truy vấn và vận hành quy trình
  - Triển khai phiên người dùng và cung cấp API giao diện JDBC/ODBC
- Compiler (Trình biên dịch)
  - Đề cập đến metastore để phân tích cú pháp truy vấn và tạo kế hoạch thực hiện
- Metastore
  - Lưu trữ thông tin về db, bảng và phân vùng
- Công cụ thực thi
  - Thực hiện kế hoạch thực hiện được tạo bởi trình biên dịch



Dịch vụ Hive

- metastore
  - Dịch vụ metastore có cơ sở dữ liệu vật lý lưu trữ cấu trúc dữ liệu HDFS. Metastore có ba chế độ thực hiện.
  - Nhúng: Chế độ sử dụng Derby DB mà không cần định cấu hình cơ sở dữ liệu riêng. Chỉ một người dùng có thể truy cập tại một thời điểm
  - Cục bộ: Nó có một cơ sở dữ liệu riêng nhưng chạy trên cùng một JVM với trình điều khiển tổ ong.
  - Từ xa: Chế độ có cơ sở dữ liệu riêng và hoạt động độc lập trong một JVM riêng
- Hiveserver2 (hiveserver2)
  - HiveServer2 cung cấp các dịch vụ liên kết với các máy khách được phát triển bằng các ngôn ngữ khác.
- beeline
  - Đây là giao diện dòng lệnh của Hive hoạt động ở chế độ nhúng giống như CLI thông thường hoặc truy cập quy trình HiveServer2 bằng JDBC.



HIVE CLI

- Giao diện dòng lệnh Hive (CLI) là công cụ cơ bản nhất để chạy truy vấn Hive

Tùy chọn Hive CLI	Mô tả
hive –e 'select a.col from tab1 a'	Chạy truy vấn
hive –S –e 'select a.col from tab1 a'	Chạy chế độ im lặng truy vấn
hive –e 'select a.col from tab1 a' –hiveconf hive.root.logger=DEBUG, console	Đặt biến cấu hình hive
hive –i initialize.sql	Sử dụng tập lệnh khởi tạo
hive –f script.sql	Chạy tập lệnh không tương tác

- Shell tương tác
  - Ví dụ

```
--hive execution
$ hive
hive> select * from product;

Show the result
```

Beeline

- Beeline là một công cụ để thực hiện truy vấn bằng cách kết nối với Hiveserver2 dựa trên SQLLine.
  - Kết nối với Hive Server 2 bằng JDBC.
- Có hai cách để kết nối với Hive Server 2
  - Nhập chuỗi kết nối làm tham số cho beeline

```
$ beeline –u jdbc:hive2://localhost:10000 –n jsjeong –p password
jdbc:hive2://localhost:10000> select * from product;
jdbc:hive2://localhost:10000>!quit
```

  - Nhập lệnh kết nối với **!connect** từ beeline shell

```
$ beeline
beeline> !connect jdbc:hive2://localhost:10000 –jsjeong password
```

Apache Impala là gì? (1/2)

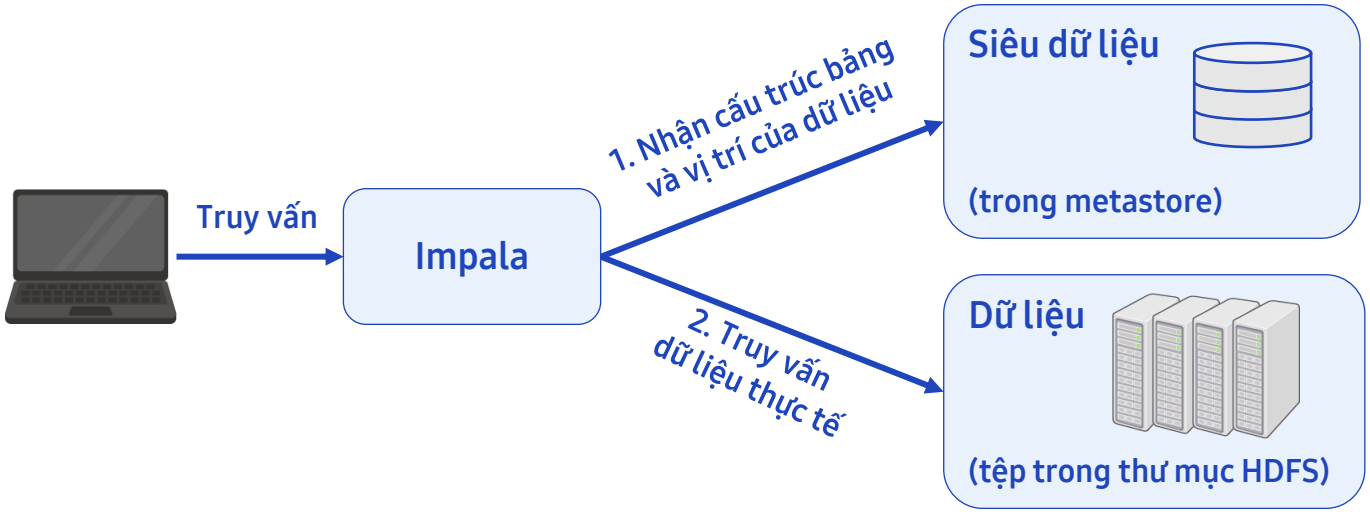
- Công cụ xử lý dữ liệu và phân tích do Cloudera phát triển
- Giống như Hive, dữ liệu HDFS có thể được truy cập thông qua các câu lệnh Truy vấn giống như SQL.
- Không giống như Hive, nó không sử dụng MR hoặc Spark làm Công cụ thực thi.
  - Nó có Công cụ truy vấn riêng và thực thi Truy vấn thông qua Impala Daemon
  - Nhanh hơn khoảng 10 đến 50 lần so với Hive
  - Khả năng truy cập đồng thời vào nhiều máy khách được cung cấp hiệu quả hơn so với Hive
- Chia sẻ Metastore với Hive
  - Các bảng được tạo trong Hive có thể được sử dụng
  - Tương tự, các bảng được tạo trong Impala có thể được sử dụng trong Hive
- Chủ yếu được sử dụng cho Truy vấn tương tác hoặc phân tích dữ liệu

Apache Impala là gì? (2/2)

- Impala là một công cụ SQL hiệu suất cao cho lượng dữ liệu khổng lồ
  - Xử lý song song lớn (MPP)
- Impala có thể truy vấn dữ liệu được lưu trữ trong HBase, Kudu, S3, ADLS thay vì HDFS
- Đọc và ghi dữ liệu ở các định dạng tệp Hadoop phổ biến
- Trực tiếp thực hiện truy vấn trên cụm
- Ngôn ngữ truy vấn Impala : Impala SQL
  - Nó tương thích với SQL và HiveQL tiêu chuẩn, nhưng có một số khác biệt
  - HiveQL và Impala SQL rất giống nhau

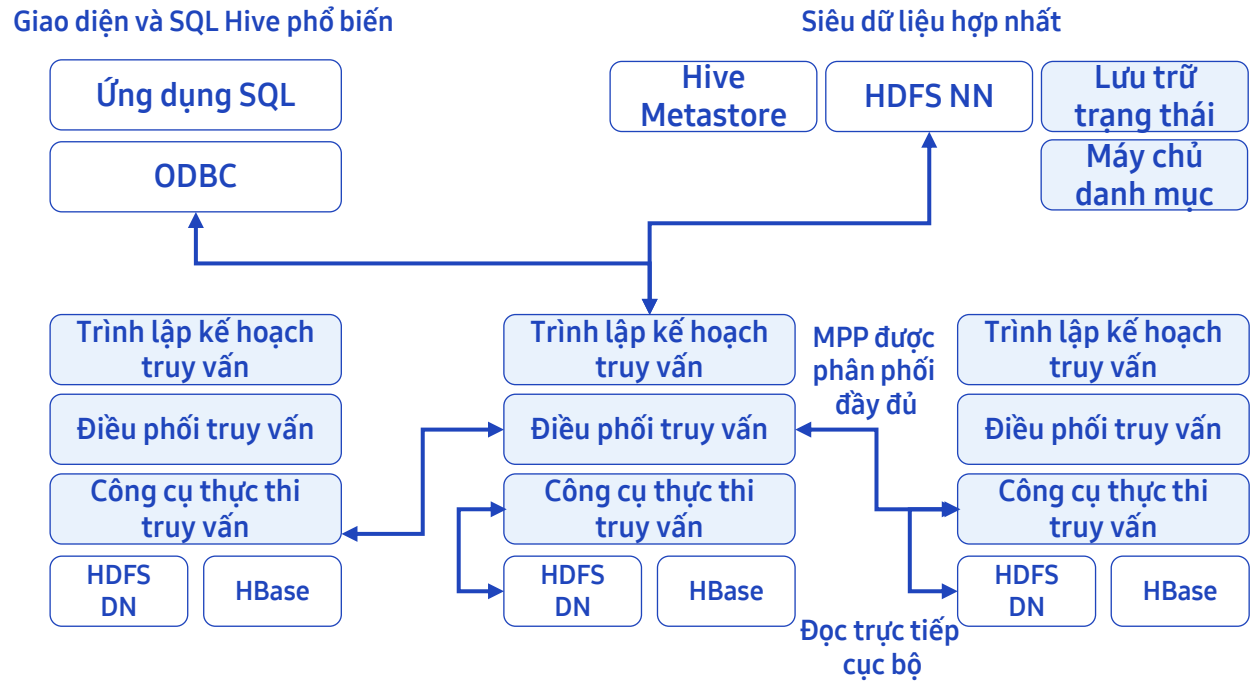
Dữ liệu truy vấn Impala

- Đầu tiên, Impala sử dụng metastore để kiểm tra lược đồ dữ liệu và vị trí cho bảng
- Sau đó, Truy vấn dữ liệu bảng thực tế trong HDFS hoặc bộ lưu trữ khác



Kiến trúc Apache Impala

- Impala với HDFS



Thành phần của Apache Impala (1/2)

- Trình nền Impala
  - Nói chung, có một (hoặc nhiều hơn một) worker node trong cụm.
  - Thực hiện truy vấn nhận được từ máy khách và gửi kết quả
  - Máy khách: Hue Web UI, Impala-shell, JDBC/ODBC, v.v.
- Các yếu tố cải thiện hiệu suất
  - Làm việc với HDFS DataNode mà trình nền thuộc về
  - Sử dụng bộ đệm siêu dữ liệu cục bộ
  - Cân bằng tải giữa các trình nền Impala trong một cụm

Thành phần của Apache Impala (2/2)

- Lưu trữ trạng thái Impala
  - Một kho lưu trữ trạng thái trên mỗi cụm
  - Liên tục kiểm tra trạng thái của trình nền Impala
- Máy chủ danh mục Impala
  - Một máy chủ danh mục trên mỗi cụm
  - Những thay đổi trong kết quả thực thi Truy vấn trong trình nền Impala được phản ánh trong toàn cụm
  - Các thay đổi có thể được phản ánh thủ công thông qua các lệnh INVALIDATE METADATA hoặc REFRESH
  - Lệnh trên là bắt buộc khi thay đổi siêu dữ liệu bên ngoài Impala (Cập nhật bảng trong Ví dụ: Hive)

Apache Impala – Impala Shell

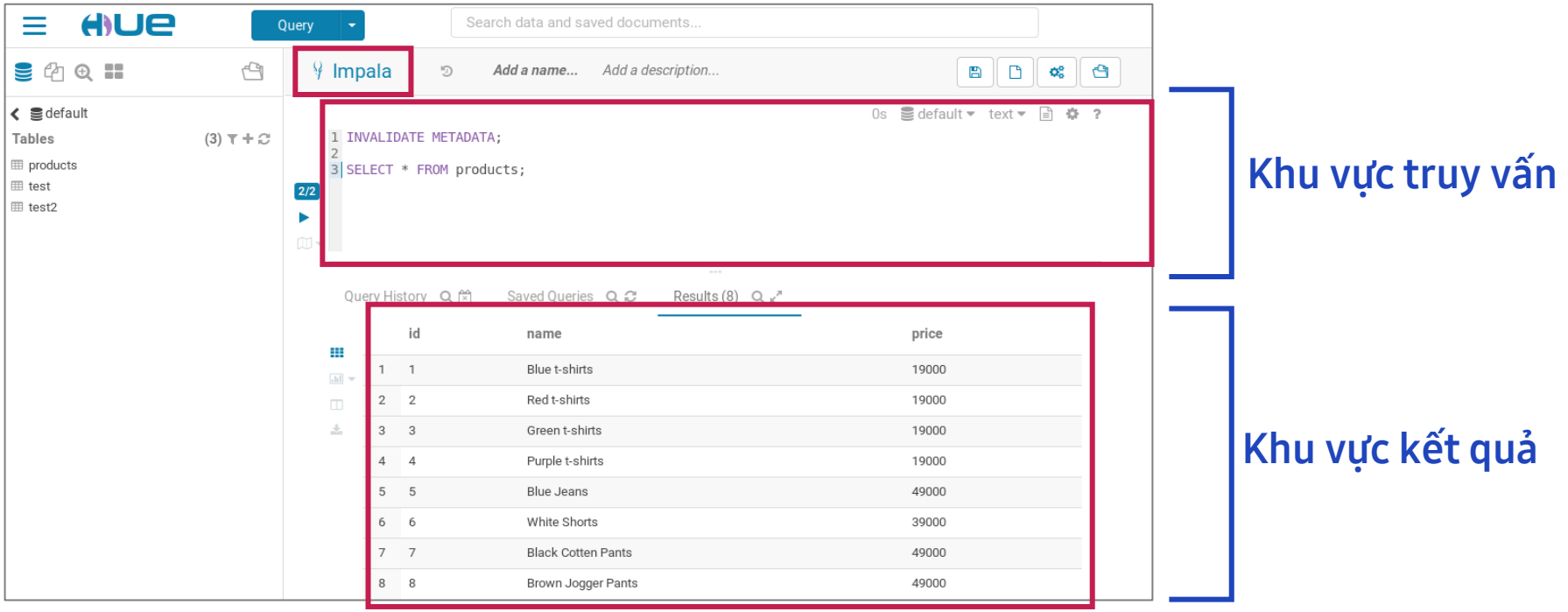
- Sử dụng Impala Shell
  - Cung cấp ứng dụng khách Impala-shell dựa trên CLI như Hive Beeline
  - Có sẵn trên các nút đã cài đặt daemon Impala

```
impala@90d98a18078c:/opt/impala/bin$ impala-shell
Starting Impala Shell without Kerberos authentication
Opened TCP connection to 90d98a18078c:21000
Connected to 90d98a18078c:21000
Server version: impalad version 3.4.0-RELEASE RELEASE (build Cloud not obtain git hash)
***********************************************************************
Welcome to the Impala shell.
(Impala Shell v3.4.0-RELEASE (9f1c31c) built on Fri Apr 24 14:10:19 PDT 2020)

To see more tips, run the TIP command.
***********************************************************************
[90d98a18078c:21000] default>
```

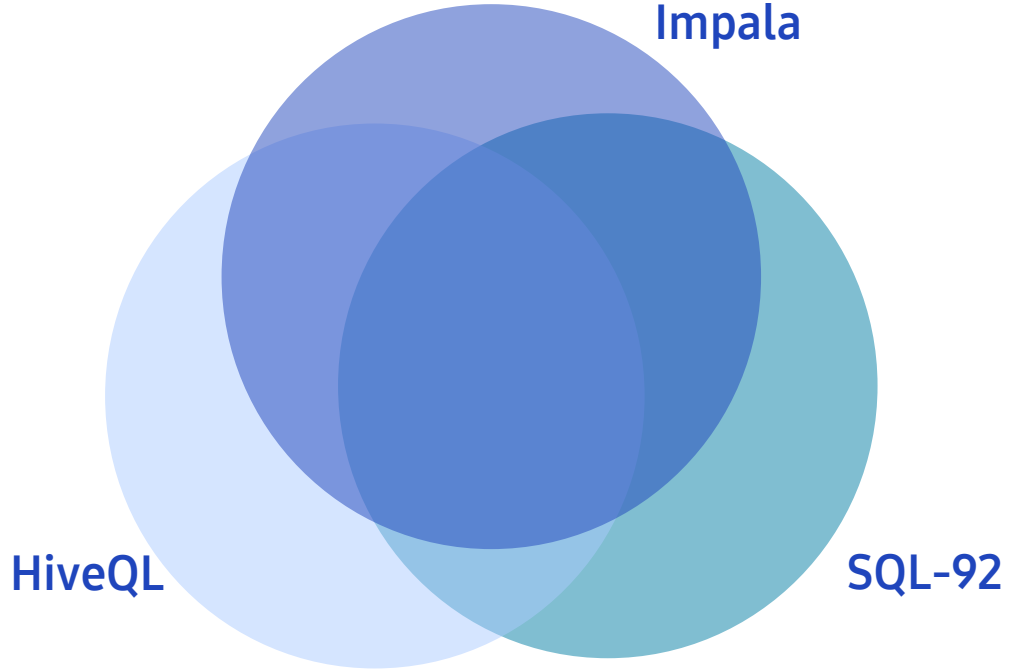
Apache Impala – Trình chỉnh sửa HUE

- Sử dụng trình chỉnh sửa HUE



Ngôn ngữ truy vấn cho Hive, Impala

- Hive : HiveQL
- Impala: Impala SQL



Chúng tôi có thể sử dụng ngôn ngữ truy vấn gần như giống nhau cho Hive và Impala

Hive QL

- Từ khóa Hive không phân biệt chữ hoa chữ thường
- Các câu lệnh được kết thúc bằng dấu chấm phẩy
  - Một tuyên bố kéo dài nhiều dòng của tôi
- Nhận xét bắt đầu bằng “–”
  - Chỉ được hỗ trợ trong tập lệnh Hive (*.hql, *.sql)
  - Không có bình luận nhiều dòng

Cơ sở dữ liệu trong HDFS

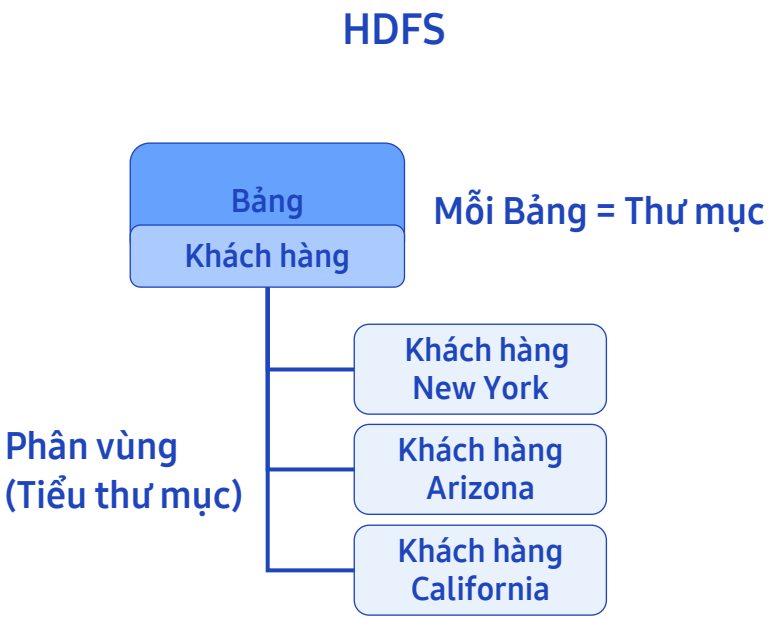
- "default" là tên của cơ sở dữ liệu mặc định
- Hive và Impala thường sử dụng /user/hive/warehouse để lưu trữ dữ liệu của nó
- Hive và Impala hỗ trợ nhiều cơ sở dữ liệu
  - Hữu ích cho tổ chức và ủy quyền
- Tạo cơ sở dữ liệu mới sẽ tạo một thư mục mới trong thư mục mặc định
- Cơ sở dữ liệu "mặc định" không có thư mục riêng
  - Đường dẫn mặc định: /user/hive/warehouse/tablename

```
$hdfs dfs – ls /user/hive/warehouse

Found 2 items
drwxrwxrwx – hive supergroup 0 2020-08-10 02:35 /user/hive/warehouse/accounting.db
drwxrwxrwx – hive supergroup 0 2020-10-26 04:55 /user/hive/warehouse/orders
```

Bảng

- Dữ liệu cho các bảng Hive và Impala được lưu trữ trên HDFS
  - Mỗi bảng ánh xạ tới một thư mục
- Thư mục của bảng có thể chứa nhiều tệp
  - Nhiều lần giảm sẽ tạo nhiều tệp với một thư mục
  - Thư mục con không được phép, trừ khi bảng được phân vùng
- Phân vùng là thư mục con
- Cơ sở dữ liệu Metastore được sử dụng để ánh xạ lược đồ cơ sở dữ liệu tới thư mục và tệp HDFS.
  - Giúp ánh xạ dữ liệu thô trong HDFS tới các cột được đặt tên theo loại cụ thể



Quản lý cơ sở dữ liệu

- Khi tạo một cơ sở dữ liệu mới trong hive:
  - Một DB mới được tạo trong thư mục con /user/hive/warehouse/new_dbname

```
CREATE DATABASE <DB_NAME>;
```

```
SHOW DATABASES;
```

```
USE DATABASE <DB_NAME>;
```

```
DROP DATABASE <DB_NAME>;
```

Loại dữ liệu

- Các kiểu dữ liệu nguyên thủy
  - Kiểu số

Loại	Mô tả
INT	Số nguyên có dấu 4 byte
BIGINT	Số nguyên có dấu 8 byte
SMALLINT	Số nguyên có dấu 2 byte
TINYINT	Số nguyên có dấu 1 byte
FLOAT	Dấu chấm động chính xác đơn 4 byte
DOUBLE	Dấu chấm động chính xác kép 8 byte
DECIMAL	số có 38 chữ số
NUMERIC	Tương tự như DECIMAL

Loại dữ liệu

- Các kiểu dữ liệu nguyên thủy
  - Date/time, String, Boolean

Loại	Mô tả
STRING	VARCHAR() VÀ CHAR() cũng khả dụng
BINARY	Dữ liệu nhị phân
TIMESTAMP	Số nguyên, số thực hoặc chuỗi
BOOLEAN	Boolean đúng hay sai

- Các loại phức tạp
  - Mảng
  - Map
  - Cấu trúc (STRUCT)
  - Liên hiệp (UNION)

TẠO Bảng (1/5)

- Bảng được quản lý / bên ngoài
  - Cú pháp

```
› CREATE [EXTERNAL] TABLE [IF NOT EXISTS]
  [<DB_NAME>.]<TBL NAME>
  [(<COL_NAME> DATA_TYPE, <COL_NAME> DATA_TYPE, ...)]
  [PARTITIONED BY (<COL_NAME> DATA_TYPE)]
  [ROW FORMAT <ROW_FORMAT>]
  [[STORED AS <FLIE_FORMAT>]
  | STORED BY '<storage.handler.class.name>' [WITH SERDEPROPERTIES (...)]]
  [LOCATION <HDFS_PATH>]
  [TBLPROPERTIES (<property_name>=<property_value>, ...)]
  [AS <SELECT_STMT>];
```

TẠO Bảng (2/5)

- Tạo bảng Hive sử dụng products.file
  - Bảng được quản lý Hive : sản phẩm
  - Lược đồ bảng

	Cột	Loại dữ liệu		Cột	Loại dữ liệu
1	Id	Int	3	Price	int
2	name	string			

  - Create Hive table ‘products’

```
CREATE TABLE products (
    id int,
    name string,
    price int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE
LOCATION ‘/user/jsjeong/products’;
```

TẠO Bảng (3/5)

- Tạo bảng Hive sử dụng products.file
  - Bảng ngoài Hive : sản phẩm
  - Lược đồ bảng:

	Cột	Loại dữ liệu		Cột	Loại dữ liệu
1	Id	Int	3	Price	int
2	name	string			

  - Tạo bảng Hive 'sản phẩm'

```
CREATE EXTERNAL TABLE AB_NYC_2019 (
    id int,
    name string,
    price int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE
LOCATION ‘/user/jsjeong/products’;
```

TẠO Bảng (4/5)

- Tạo Bảng có chú thích, thuộc tính bảng và vị trí thay thếExample

```
CREATE TABLE IF NOT EXISTS mydb.employees (
    name STRING COMMENT 'Employee name’,
    salary FLOAT  COMMENT 'Employee salary’,
    subordinates ARRAY<STRING> COMMENT 'Names of subordinates’,
    deductions MAP<STRING, FLOAT> COMMENT 'Keys are deductions names, values are percentages’,
    address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT> COMMENT 'Home address’
)
COMMENT 'Description of the table’
TBLPROPERTIES ('creator'='me', 'created_at'='2021-07-02 10:00:00', ...)
LOCATION '/user/hive/warehouse/mydb.db/employees';
```

TẠO Bảng (5/5)

- Sao chép lược đồ của một bảng.
- Bạn cũng có thể thay đổi vị trí tạo bảng. Không có thuộc tính nào khác có thể được sửa đổi và tuân theo bảng gốc.
  - Ví dụ

```
CREATE TABLE IF NOT EXISTS mydb.employees2
LIKE mydb.employees
LOCATION /my/preferred/location;
```

  - Hiển thị danh sách các bảng từ cơ sở dữ liệu hiện tại hoặc cơ sở dữ liệu khác

```
hive> USE mydb;
hive> SHOWTABLES;
employees
table1
table2
```

=

```
hive> USE default;
hive> SHOWTABLES IN mydb;
employees
table1
table2
```

DESCRIBE Table (MÔ TẢ Bảng)

- Hiển thị thông tin bảng.

```
> DESCRIBE <TBL_NAME>  // (or) DESC <TBL_NAME>
```

```
+-------------+-------------+-------------+
| col_name    | data_type   | comment     |
+-------------+-------------+-------------+
| id          | int         |             |
| name        | string      |             |
| price       | int         |             |
+-------------+-------------+-------------+
```

<Mô tả sản phẩm>

- Hiển thị chi tiết thông tin bảng

```
> DESCRIBE FORMATTED <TBL_NAME>  // (or) DESC FORMATTED  <TBL_NAME>
```

Bảng DROP & Bảng ALTER

- ‘sản phẩm’ bảng drop
  - Trong trường hợp bảng bên ngoài, ngay cả khi bảng bị xóa, thư mục trong HDFS vẫn không bị xóa

```
DROP TABLE products;
```

- ‘sản phẩm’ bảng alter

```
ALTER TABLE products RENAME products_2019;
ALTER TABLE products CHANGE id prod_id int;
ALTER TABLE products ADD COLUMNS(last_modified date);
```

  - Xóa hoặc thay thế các cột

```
ALTER TABLE products REPLACE COLUMNS (
 hours_mins_secs  INT COMMENT 'hour, minute, seconds from timestamp',
 severity STRING COMMENT 'The message severity'
 message STRING COMMENT 'The rest of the message’
);
```

Bảng ALTER

- Các lệnh bảng thay đổi khác nhau cho Phân vùng (Partition)
  - Thêm phân vùng vào bảng

```
ALTER TABLE log_messages ADD IF NOT EXISTS
PARTITION (year = 2021, month = 1, day = 1) LOCATION '/logs/2021/01/01'
PARTITION (year = 2021, month = 1, day = 2) LOCATION '/logs/2021/01/02'
```

  - Thay đổi vị trí của một phân vùng

```
ALTER TABLE log_messages PARTITION(year = 2021, month = 7, day = 2)
SET LOCATION   's3n://ourbucket/logs/2021/07/02';
```

  - Xóa một phân vùng

```
ALTER TABLE log_messages DROP IF EXISTS
PARTITION(year = 2021, month = 7, day = 2);
```

Truy vấn dữ liệu từ Table

- Câu lệnh SELECT truy xuất các bản ghi từ các bảng
  - Chỉ định các cột riêng lẻ
  - Tên cột không phân biệt chữ hoa chữ thường.

```
SELECT column1, column2, column FROM table_name;
```

  - Tất cả các cột cho “*”

```
SELECT * FROM table_name;
```

Câu hỏi ôn tập

[Câu hỏi]

Câu hỏi 1

- Bạn sử dụng lệnh nào để chuyển sang cơ sở dữ liệu khác?

Câu hỏi 2

- Khi bạn khởi động Impala hoặc Hive lần đầu tiên, tên của cơ sở dữ liệu đã chọn là gì?

Câu hỏi 3

- Siêu dữ liệu về một bảng (chẳng hạn như thuộc tính của bảng và tên cột) được lưu trữ ở đâu?

Câu hỏi 4

- Dữ liệu cho các bảng được phân vùng được lưu trữ ở đâu?

[Lab6]
Chạy các truy vấn cơ bản với Hive QL

Bài 3.

Phân tích nâng cao

Phân tích Big Data

Bài 3

Phân tích nâng cao

Chọn định dạng tệp

- Hive và Impala hỗ trợ nhiều định dạng tệp khác nhau để lưu trữ dữ liệu
  - Cú pháp

```
CREATE TABLE DB_name.table_name (col_name DATATYPE,....)
ROW FORMAT DELIMITED FIELDS TERMINATED BY char
STORED AS supported_format;
```

- Định dạng tệp được hỗ trợ
  - Tệp văn bản
  - Tệp tuần tự
  - AVRO
  - Parquet
  - Orc (chỉ Hive)

Định dạng tệp

- TEXTFILE (Tệp văn bản)
  - Thông thường, các giá trị được phân tách bằng dấu phẩy hoặc tab (CSV, TSV)
- SEQUENCEFILE (Tệp tuần tự)
  - Tệp phẳng bao gồm các cặp khóa/giá trị nhị phân
  - Không nén
  - Bản ghi được nén - chỉ các giá trị được nén
- Parquet – một định dạng lưu trữ cột
  - Có sẵn cho mọi thành phần trong hệ sinh thái Hadoop, bất kể khung xử lý dữ liệu, mô hình dữ liệu hay ngôn ngữ lập trình.
  - Kết hợp tốt nhất với impala
- ORC – Optimized Row Columnar (Cột hàng được tối ưu hóa)
  - Được thiết kế để khắc phục những hạn chế của các định dạng tệp khác
  - Sử dụng tệp ORC cải thiện hiệu suất

Sử dụng định dạng Parquet

- Tạo bảng bên ngoài bằng định dạng sàn gỗ
- Khi bạn sử dụng **LƯU TRỮ DẠNG PARQUET**, Impala và Hive sẽ tự động đảm nhận việc thiết lập cách mã hóa các bản ghi trong các tệp Parquet.

```
CREATE EXTERNAL TABLE authors_parquet
(
    first_name string,
    last_name string
)
STORED AS PARQUET
```

```
0: jdbc:hive2://> create external table authors_parquet  (
. . . . . .>    f_name string,
. . . . . .>    l_name string)
. . . . . .> STORED AS PARQUET;
OK
No rows affected (0.086 seconds)
```

Sử dụng định dạng Parquet trong Impala

- Tạo một bảng mới để truy cập tệp Parquet hiện có trong HDFS
  - Trong Impala, sử dụng **LIKE PARQUET** để tạo bảng bằng lược đồ của tệp parquet hiện có.

```
CREATE EXTERNAL TABLE author_parquet
    LIKE PARQUET '/home/student/authors_parquet/data1.parquet'
    STORED AS PARQUET
    LOCATION '/home/student/authors_parquet/';
```

Truy vấn HiveQL Select

- Ngữ nghĩa HiveQL

SELECT, LOAD INSERT từ truy vấn
Biểu thức trong WHERE và HAVING
GROUP BY, ORDER BY, SORT BY
Các truy vấn phụ trong mệnh đề FROM
GROUP BY, ORDER BY
UNION
LEFT, RIGHT và FULL INNER/OUTER JOIN
CROSS JOIN, LEFT SEMI JOIN
Hàm cửa sổ (OVER, RANK, v.v.)
INTERSECT, EXCEPT, UNION, DISTINDT
Truy vấn phụ trong WHERE (IN, NOT IN, EXISTS/NOT EXISTS)

Truy vấn dữ liệu từ Table

- Câu lệnh **SELECT** truy xuất các bản ghi từ các bảng
  - Chỉ định các cột riêng lẻ
  - Tên cột không phân biệt chữ hoa chữ thường

```
SELECT column1, column2, columnN FROM table_name;
```

  - Tất cả các cột cho “*”

```
SELECT * FROM table_name;
```

Sắp xếp và phân biệt truy vấn

- Sử dụng **DISTINCT** để loại bỏ trùng lặp

```
SELECT DISTINCT name FROM authors;
```

- **ORDER BY** sắp xếp kết quả
  - Thứ tự mặc định tăng dần (ASC)
  - **DESC**: thứ tự giảm dần

```
SELECT id, first_name, last_name, email FROM authors
ORDER BY last_name;
```

Giới hạn kết quả truy vấn

- Mệnh đề **LIMIT** đặt số hàng tối đa được trả về

```
SELECT id, first_name, last_name  FROM authors
ORDER BY last_name, first_name DESC LIMIT 10;
```

- Bí danh của bảng
  - Bí danh bảng có thể giúp đơn giản hóa các truy vấn phức tạp

```
SELECT a.first_name, a.last_name, a.email, p.title
FROM authors AS a JOIN posts AS p
ON (a.id = p.author_id)
WHERE a.email LIKE '%.net' OR
a.Email LIKE '%.org' LIMIT 10;
```

id	first_name	last_name
4572	Willow	Abbott
3574	Vanessa	Abbott
6071	Turner	Abbott
2923	Tiara	Abbott
6503	Stanley	Abbott
4169	Sincere	Abbott
1780	Melany	Abbott
3310	Mario	Abbott
9290	Mario	Abbott
4829	Magdalena	Abbott

10 hàng được chọn (19,667 giây)

Lọc kết quả với WHERE

- Các mệnh đề **WHERE** giới hạn các hàng đối với các tiêu chí phù hợp đã chỉ định

```
SELECT * FROM authors WHERE id=9290;
```

- Toán tử **IN**

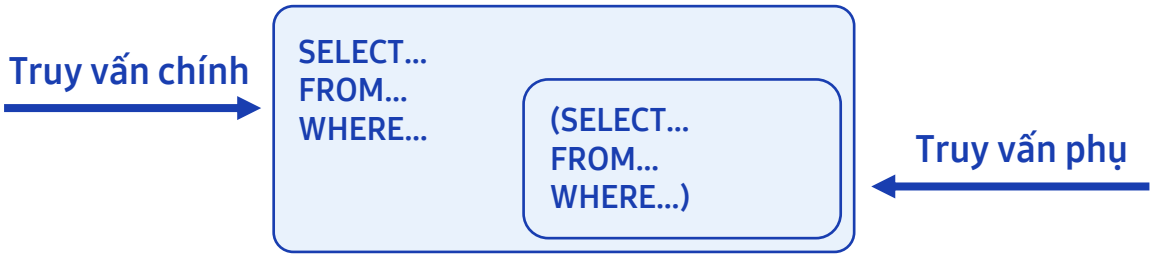
```
SELECT * FROM authors
WHERE first_name IN ("Mario", "Malany", "Dorthy", "Tiara", "Willow");
```

- **LIKE** và **AND**, **OR**

```
SELECT * FROM authors
WHERE first_name LIKE 'Do%'
AND (birthdate = '1980-04-20’ OR birthdate = '1983-02-12');
```

Các truy vấn phụ trong mệnh đề FROM

- Truy vấn phụ là một câu lệnh SQL khác có trong một câu lệnh SQL
- Các truy vấn phụ hỗ trợ Hive và Impala trong mệnh đề FROM
  - Truy vấn phụ được sử dụng trong mệnh đề FROM được gọi là dạng xem nội tuyến (dạng xem động)
  - Truy vấn phụ phải là name(sub_name)




```
SELECT order_item
FROM (SELECT  order_item, COUNT(*) AS cnt
FROM order_table GROUP BY order_item) big_order
WHERE cnt >=10;
```

Các truy vấn phụ trong mệnh đề WHERE

- Hive và Impala hỗ trợ truy vấn phụ trong mệnh đề WHERE

```
SELECT u.first_name, u.last_name, login_time FROM users u
WHERE u.id IN ( SELECT c.uid FROM checkin c );
```

bảng kiểm tra

bảng người dùng

Truy vấn và chèn dữ liệu (1/2)

- Nhập dữ liệu và truy vấn là các hoạt động đọc và ghi dữ liệu bằng thông tin meta của bảng Hive
- Thông tin meta của bảng lưu trữ vị trí của tệp thực tế và định dạng của dữ liệu
- Đối với nhập dữ liệu (**INSERT**),
  - Có hai cách để ghi dữ liệu vào bảng bằng câu lệnh **INSERT** và sao chép tệp vào vị trí lưu trữ của bảng và đến LOCATION được chỉ định khi tạo bảng
  - Một bảng phân vùng bao gồm một thư mục được chia thành các phân vùng và sao chép các tệp
- Đối với truy vấn (**SELECT**),
  - Một truy vấn (**SELECT**) đọc một tệp tại vị trí của LOCATION trong bảng

Truy vấn và chèn dữ liệu (2/2)

- Các phương pháp Insert và Select bao gồm
  - Phương pháp đọc một tệp và ghi vào một bảng
  - Phương pháp đọc dữ liệu từ một bảng và ghi vào một bảng khác
  - Phương pháp đọc một bảng và ghi nó vào một thư mục được chỉ định
- Từ tập tin đến bảng
  - Ghi vào bảng bằng lệnh LOAD
  - Sao chép tệp vào LOCATION của bảng
- Từ bảng đến bảng
  - Câu lệnh **INSERT**
  - Câu lệnh **CREATE TABLE AS SELECT**
- Bảng tới thư mục
  - Câu lệnh **INSERT DIRECTORY**

Tải dữ liệu

- Ghi vào bảng bằng lệnh **LOAD**
  - Thực thi lệnh trên Hive và Impala CLI
  - Nguồn dữ liệu có thể là tệp hoặc thư mục
  - Cú pháp

```
LOAD DATA [LOCAL] INPATH source [OVERWRITE] INTO TABLE table_name [PARTITION
(partcol1=value1, partcol2=value2 ...)]
```

  - Ví dụ

```
LOAD DATA INPATH '/user/student/data/product.txt' INTO table product;
```

- Sử dụng lệnh hdfs dfs **–mv** để tải dữ liệu
  - Trong shell, nó có thể chạy bằng lệnh hdfs dfs

```
$ hdfs dfs –mv product.txt /user/hive/warehouse/product/
```

Sao chép tệp vào LOCATION của bảng (1/2)

- Phương thức chỉ định thông tin **LOCATION** của bảng có thể được chỉ định trong quá trình **CREATE**
  - Chỉ cần sao chép hoặc di chuyển dữ liệu đến vị trí này
  - Không thể chỉ định một vị trí cục bộ,
  - Phải sử dụng hệ thống tệp chia sẻ tệp mà Hadoop có thể truy cập, chẳng hạn như HDFS hoặc S3

```
CREATE TABLE product (
            id integer,
            name string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/data';
```

Sao chép tệp vào LOCATION của bảng (2/2)

- Sử dụng lệnh **Alter** để thay đổi vị trí

```
hive > create table product (
     > id integer,
     > name string
     > )
     > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.089 seconds
hive > desc product;
OK
id                      int
name                    string
Time taken: 0.058 seconds, Fetched: 2 row(s)
hive > select * from product;
OK
Time taken: 0.206 seconds
hive > ALTER TABLE product SET location '/user/student/data';
OK
Time taken: 0.132 seconds
hive >
     >
     >
     > select * from product;
OK
1       hadoop
2       hive
3       impala
4       pig
5       sqoop
6       flume
7       kafka
8       nifi
9       hdfs
10      yarn
11      spark
12      flink
Time taken: 0.161 seconds, Fetched: 12 row(s)
hive >
```

```
[student@localhost ch5]$ hdfs dfs -put product.txt data
[student@localhost ch5]$ hdfs dfs -ls data
Found 1 items
-rw-r--r--   1 student row  95 2021-07-25 11:59 data/product.txt
[student@localhost ch5]$ hdfs dfs -cat data/product.txt
1       hadoop
2       hive
3       impala
4       pig
5       sqoop
6       flume
7       kafka
8       nifi
9       hdfs
10      yarn
11      spark
12      flink
[student@localhost ch5]$
```

Truy vấn dữ liệu xuất ra Bảng

- Câu lệnh **INSERT**
  - Là một phương thức nhập dữ liệu cơ bản, dữ liệu từ một bảng hoặc dạng xem được nhập vào một bảng khác
  - Cú pháp cơ bản của câu lệnh **INSERT** như sau

```
INSERT OVERWRITE TABLE tablename [PARTITION (partcol1=value1, partcol2=value2 ...) [IF NOT EXISTS]]
select_statement FROM statement;
```

  - Cú pháp thêm hàng mà không xóa dữ liệu hiện có cho bảng đích

```
INSERT INTO TABLE tablename [PARTITION (partcol1=value1, partcol2=value2 ...)] select_statement
FROM statement;
```

  - Ví dụ

```
INSERT INTO TABLE new_product
SELECT * FROM product
WHERE  id > 10 and name = 'spark';
```

CREATE TABLE AS SELECT

- Câu lệnh CTAS điền dữ liệu trong khi tạo bảng
  - Được gọi là **CREATE TABLE AS SELECT** (CTAS)

```
CREATE TABLE new_product
ROW FORMAT DELIMITED FIELDS TERMINATED
BY ‘,’
AS SELECT name, id
FROM product
WHERE id > 5;
```

```
0:  jbdc:hive2://> select * from
new_product;
OK
```

new_product.name	new_product.id
hdfs	6
yarn	7
sqoop	8
flume	9
kafka	10
nifi	11
hive	12
pig	13
kudu	14
hbase	15

```
Đã chọn 10 hàng (0,213 giây)
```

Ghi dữ liệu xuất vào HDFS trong Hive (1/3)

- Nó đọc dữ liệu từ bảng và xuất một tệp đến vị trí đã chỉ định
  - Đặt cách lưu dữ liệu bằng **ROW FORMAT**
  - Cú pháp cơ bản

```
INSERT OVERWRITE [LOCAL] DIRECTORY directory [ROW FORMAT row_format] [STORED AS file_format]
SELECT ... FROM ...
```

  - Cú pháp của nhiều lần chèn

```
FROM from_statement
INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select_statement 1
[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select_statement2] ... row_format : DELIMITED
[FIELDS TERMINATED BY char
```

Ghi dữ liệu xuất vào HDFS trong Hive (2/3)

- Hive cũng lưu đầu ra vào một thư mục trong HDFS

```
0: jbdc:hive://> insert overwrite directory '/user/student/product/'
. . . . . .> row format delimited fields terminated by '/t'
. . . . . .> select * from new_product
. . . . . .> where id >= 10;
```

- Hiển thị kết quả trong /user/student/product

```
[student@localhost ~]$ hdfs dfs -ls product
Found 1 items
-rw-r--r--   1 student student          49 2021-07-25 18:42 product/000000_0
[student@localhost ~]$ hdfs dfs -cat product/000000_0
kafka   10
nifi    11
hive    12
pig     13
kude    14
hbase   15
```

Ghi dữ liệu xuất vào HDFS trong Hive (3/3)

- Truy vấn sau đây tạo ra kết quả tương tự cho sản phẩm

```
FROM product p
INSERT OVERWRITE DIRECTORY 'product_from'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT name, id
WHERE id >= 10;
```

```
[student@localhost ~]$ hdfs dfs -ls product
Found 1 items
-rw-r--r--   1 student student          49 2021-07-25 18:42 product/000000_0
[student@localhost ~]$ hdfs dfs -cat product/000000_0
kafka   10
nifi    11
hive    12
pig     13
kude    14
hbase   15
```

Lưu vào nhiều thư mục

- Trích xuất dữ liệu ra nhiều tệp
  - Kết quả là hai thư mục trong HDFS

```
FROM product p
INSERT OVERWRITE DIRECTORY 'product_name'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT name
WHERE id < 10
INSERT OVERWRITE DIRECTORY 'product_id'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
SELECT count(id), id
WHERE id >= 10
GROUP BY id;
```

  - Hive có thể tạo hai kết quả thư mục bằng một truy vấn duy nhất, hiệu quả hơn so với sử dụng nhiều truy vấn.

Tạo và xóa View (Dạng xem)

- Tạo view tại thời điểm thực hiện câu lệnh **SELECT**
  - Cú pháp

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...)]
[COMMENT table_comment] AS SELECT ...
```

- View là bảng lưu trữ tạm thời và truy vấn đã lưu trên bảng hoặc tập hợp bảng

```
CREATE VIEW author_100 AS
SELECT * FROM authors
WHERE id < 101;

SELECT * FROM author_100;
```

- XÓA VIEW

```
DROP VIEW author_100;
```

Quản lý View (1/2)

- Liệt kê một view

```
SHOW TABLES;
```

- Xem truy vấn cơ bản của view

```
DESCRIBE FORMATTED author_100;
```

- Để hiển thị một câu lệnh để tạo view

```
SHOW CREATE TABLE author_100;
```

```
0:  jdbc:hive2://> show create table author 100;
OK
+----------------------------------------------------+
|                    createtab_stmt                  |
+----------------------------------------------------+
| CREATE VIEW 'author_100' AS select 'authors'.'id', 'authors'.'first_name', 'authors'. 'last_name',  |
| 'authors'.'email', where 'authors'.'id' < 101      |
+----------------------------------------------------+
```

Quản lý View (2/2)

- Thay đổi truy vấn cơ bản

```
ALTER VIEW author_100
AS select id, first_name, email, birthdate from authors;
```

- Để đổi tên một view

```
ALTER VIEW author_100
RENAME TO author_hundred;
```

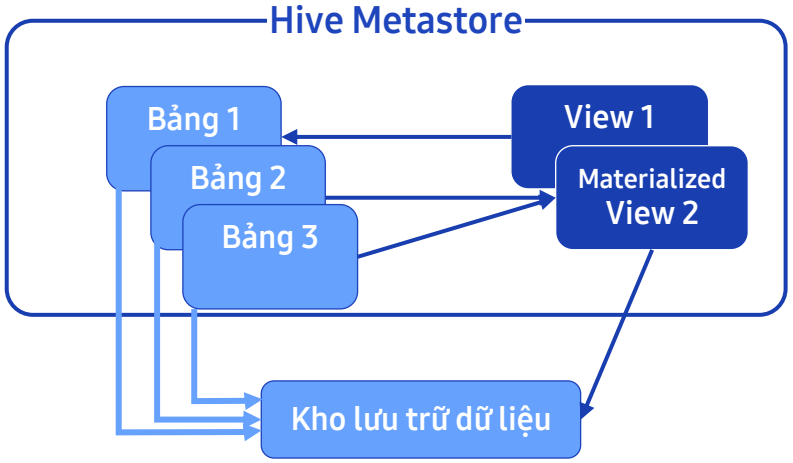
- Để xóa view

```
DROP VIEW author_hundred;
```

Materialized View trong Hive

- Materialized view duy trì kết quả để truy vấn không chạy mỗi lần
  - Có sẵn trong Hive 3.0
- Để cho phép viết lại truy vấn bằng cách sử dụng các Materialized view, cần có thuộc tính toàn cầu:

```
SET hive.materializedview.rewriting=true;.
```



Bài 3

Phân tích tương tác

3.1. Quản lý dữ liệu Hive và Impala

3.2. Dữ liệu phức tạp và phân tích dữ liệu quan hệ

Phân vùng bảng (1/4)

- Phân vùng lưu trữ dữ liệu được tách thành các thư mục
  - Các bảng dựa trên tệp như Hive về cơ bản đọc tất cả thông tin hàng trong bảng, vì vậy chúng chậm lại khi có nhiều dữ liệu
  - Một cột phân vùng có thể được sử dụng giống như một cột trong điều kiện where, do đó, nó làm giảm dữ liệu đọc lúc đầu và cải thiện tốc độ xử lý
- Cú pháp cơ bản để tạo phân vùng cho bảng là

```
CREATE TABLE author hundred (
            column type,
            ...
)
PARTITIONED BY  (column type)
ROW FORMAT DELIMITED FIEDLS TERMINATED BY '\t';
```

Phân vùng bảng (2/4)

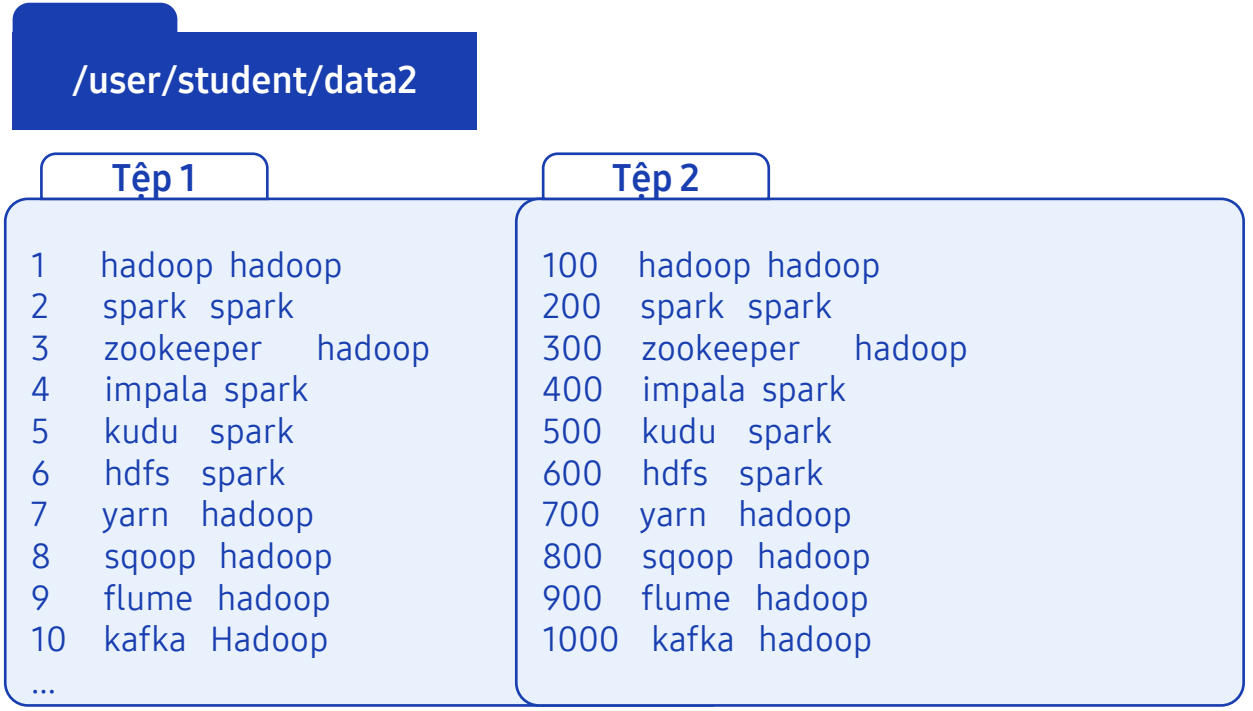
- product_category là một bảng không được phân vùng

```
CREATE TABLE product_category (
            id integer,
            name string,
            category string

)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/data2';
```

Phân vùng bảng (3/4)

- Các tệp của Product_category được lưu trữ trong một thư mục duy nhất, "/user/student/data2"
- Tất cả các tệp được quét để truy vấn



Phân vùng bảng (4/4)

- Các bảng được phân vùng lưu trữ dữ liệu trong các thư mục con

```
CREATE TABLE product_by_category (
 id integer,
 name string
)
PARTITIONED BY (category string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/product_by_category';
```

```
hive> desc product_by_category;
OK
id          int
name        string
category    string

# Partition Information
# col_name              data_type               comment
category                string
Time taken: 0.446 seconds, Fetched: 7 row(s)
```

Loại phân vùng (1/4)

- Có hai loại phân vùng: phân vùng động và phân vùng tĩnh
  - Phân vùng tĩnh: Dữ liệu đầu vào sử dụng phân vùng tĩnh được nhập bằng cách chuyển thông tin phân vùng dưới dạng giá trị cố định cho câu lệnh INSERT
  - Phân vùng động: Dữ liệu đầu vào sử dụng phân vùng động được nhập bằng cách chuyển một cột để truy vấn thông tin phân vùng tới câu lệnh INSERT
- Phân vùng tĩnh
  - Khi tải dữ liệu, bạn chỉ định phân vùng nào sẽ lưu trữ dữ liệu đó
  - Ví dụ

```
ALTER TABLE product_by_category
ADD PARTITION (category='hadoop');

INSERT OVERWRITE table product_by_category
 PARTITION(category='hadoop')
 SELECT id, name from product_category
 WHERE category = 'hadoop';
```

Loại phân vùng (2/4)

- Nếu lệnh slide trước được thực thi, thư mục sau được tạo trong hdfs và dữ liệu được nhập
  - Thư mục HDFS: /user/student/product_by_category/category=hadoop/

```
[student@localhost ~] hdfs dfs -cat product_by_category/category=Hadoop/000000_0
1       Hadoop
3       zookeeper
7       yarn
8       sqoop
9       flume
10      kafka
11      nifi
12      hive
100     Hadoop
300     zookeeper
700     yarn
800     sqoop
900     flume
10000   kafka
[student@localhost ~]
```

```
0: jbdc:hive2://> select * from product_by_category;
OK
```

product_by_category.id	product_by_category.name	product_by_category.category
1	hadoop	hadoop
3	zookeeper	hadoop
7	yarn	hadoop
8	sqoop	hadoop
9	flume	hadoop
10	kafka	hadoop
11	nifi	hadoop
12	hive	hadoop
100	hadoop	hadoop
300	zookeeper	hadoop
700	yarn	hadoop
800	sqoop	hadoop
900	flume	hadoop
1000	kafka	hadoop

Loại phân vùng (3/4)

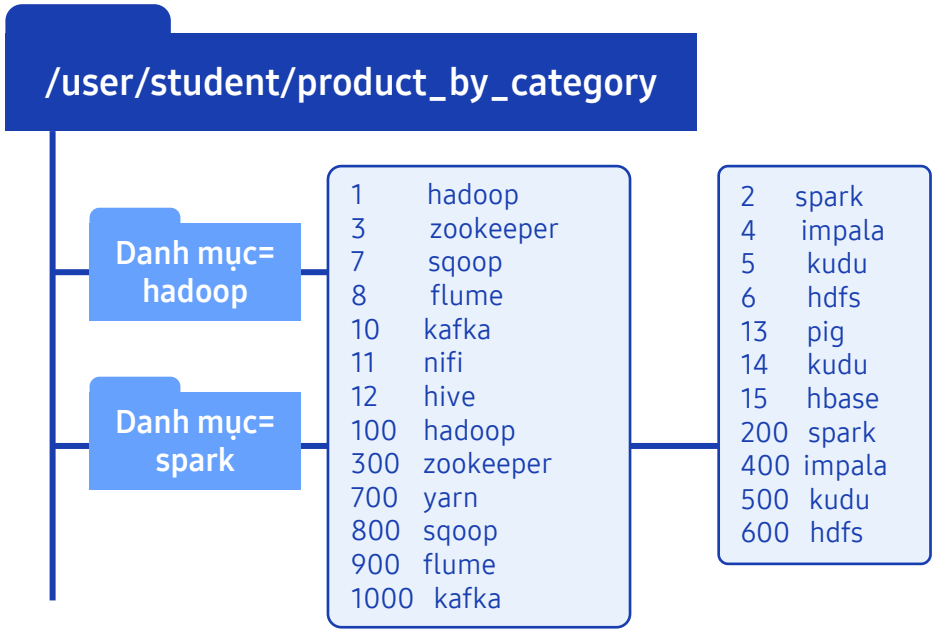
- Phân vùng động
  - Sử dụng câu lệnh INSERT để tải dữ liệu
  - Cột phân vùng phải được chỉ định cuối cùng trong cột SELECT
  - Hive không khuyến nghị chỉ sử dụng phân vùng động theo mặc định
  - Đặt hive.exec.dynamic.partition.mode=nonstrict
  - Ví dụ

```
SET hive.exec.dynamic.partition.mode=nostrict;

INSERT OVERWRITE TABLE product_by_category
PATRITION(category)
SELECT name, id, category from product_category;
```

Loại phân vùng (4/4)

- Nếu lệnh slide trước được thực thi, thư mục sau được tạo trong hdfs và dữ liệu được nhập
  - Thư mục HDFS: /user/student/product_by_category/category=hadoop/
  - Thư mục HDFS: /user/student/product_by_category/category=spark/




```
0: jdbc:hive2://> select * from product_by_category;
OK
```

product_by_category.id	product_by_category.name	product_by_category.category
1	hadoop	hadoop
3	zookeeper	hadoop
7	yarn	hadoop
8	sqoop	hadoop
9	flume	hadoop
10	kafka	hadoop
11	nifi	hadoop
12	hive	hadoop
100	hadoop	hadoop
300	zookeeper	hadoop
700	yarn	hadoop
800	sqoop	hadoop
900	flume	hadoop
1000	kafka	hadoop
2	spark	spark
4	impala	spark
5	kudu	spark
6	hdfs	spark
13	pig	spark
14	kudu	spark
15	hbase	spark
200	spark	spark
400	impala	spark
500	kudu	spark
600	hdfs	spark

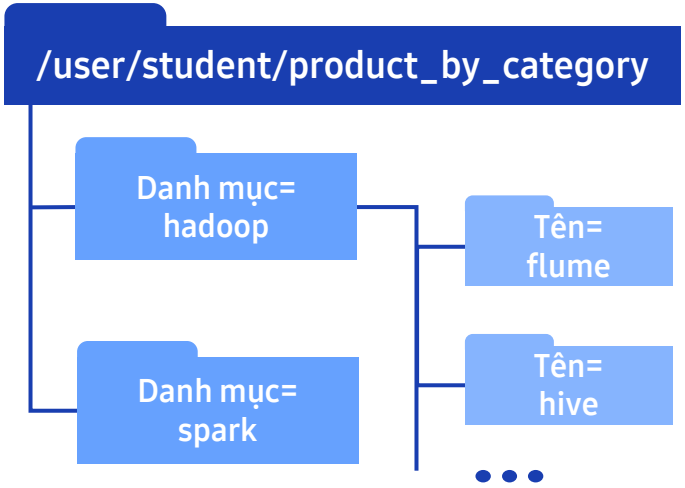
Phân vùng lồng nhau

- Khi bảng được phân vùng bằng nhiều cột, Hive sẽ tạo các thư mục con lồng nhau dựa trên thứ tự của các cột phân vùng

```
CREATE TABLE product_by_category (id int)
PARTITIONED BY (category string, name string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/student/product_by_category';
```

```
0: jbdc:hive2://> desc product_by_category;
OK
```

col_name	data_type	comment
id	int	
category	string	
name	string	
	NULL	NULL
# Partition Information	NULL	NULL
# col_name	data_type	comment
category	string	
name	string	



Thuộc tính Hive trong các phân vùng

- Có giới hạn về số lần tạo phân vùng động vì phân vùng động chậm và có thể tạo một số lượng lớn phân vùng
- Khi tạo nhiều phân vùng hơn cài đặt mặc định, các cài đặt sau sẽ được thực hiện
  - đặt hive.exec.dynamic.partition.mode=strict;
  - số phân vùng động
  - số phân vùng động được tạo trên mỗi node

```
SET hive.exec.dynamic.partition.mode=strict;
SET hive.exec.max.dynamic.partitions=1000;
SET hive.exec.max.dynamic.partitions.pernode=100;
```

Quản lý phân vùng (1/2)

- Để hiển thị các phân vùng hiện tại trong bảng

```
SHOW PARTITIONS product_by_category;
```

```
0: jbdc:hive2://> show partitions product_by_category;
OK
+-----------------------+
|       partition       |
+-----------------------+
| category=hadoop       |
| category=nosql        |
| category=spark        |
+-----------------------+
3 rows selected (1.591 seconds)
```

- Sửa đổi/Xóa phân vùng
  - Để sửa đổi hoặc xóa một phân vùng, hãy sử dụng câu lệnh ALTER
- ADD (thêm) phân vùng mới

```
ALTER TABLE product_by_category ADD PARTITION (category='nosql');
```

Quản lý phân vùng (2/2)

- Sửa đổi LOCATION của phân vùng

```
ALTER TABLE product_by_category PARTITION
(category='nosql') SET LOCATION '/user/student/data3';
```

- Xóa bỏ sự chia cắt

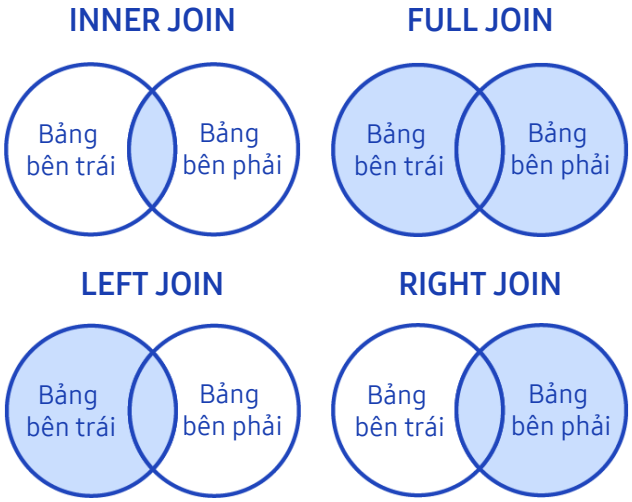
```
ALTER TABLE product_by_category DROP PARTITION (category='nosql');
```

- Xóa phạm vi phân vùng, phạm vi có thể bị xóa bằng toán tử so sánh

```
ALTER TABLE product_by_category DROP PARTITION (category < 'nosql');
```

Joins (nối)

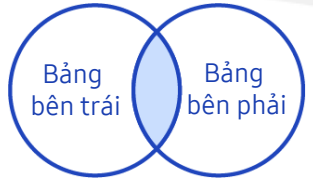
- Để kết hợp các bản ghi từ hai hoặc nhiều bảng trong cơ sở dữ liệu, chúng tôi sử dụng mệnh đề JOIN
- Hive và Impala hỗ trợ một số kiểu thao tác kết nối
  - Về cơ bản có 4 kiểu Join
    INNER JOIN
    LEFT OUTER JOIN
    RIGHT OUTER JOIN
    FULL OUTER JOIN



- Cú pháp **JOIN**

```
table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference join_condition
  | table_reference LEFT SEMI JOIN table_reference join_condition
  | table_reference CROSS JOIN table_reference [join_condition]
```

Inner Join (Phép nối trong)



- **INNER JOIN** bao gồm các bản ghi có giá trị khóa phù hợp giữa Bảng A và Bảng B

```
SELECT c.name, c.capital, c.population, c1.cname FROM country c
INNER JOIN continent_info c1 ON c.continent_id = c1.id;
```

Country table

name	capital	population	Co_id
B_country	Africa	2083850	5
E_country	Asia	963089	1
F_country	Europe	671580	3
G_country	Oceania	823494	3
K_country	North America	1266720	2
D_country	South America	5256570	4

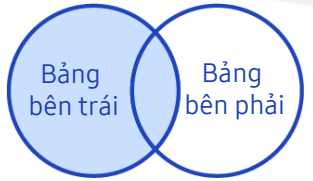
Continent_info Table

id	cname
1	Africa
2	Asia
3	Europe
4	Oceania
5	North America
6	South America

<Kết quả>

c.name	c.capital	c.population	c1.cname
B_Country	B_City	2083850	North America
E_Country	E_City	963089	Africa
F_Country	F_City	671580	Europe
G_Country	G_City	823494	Europe
K_Country	K_City	1266720	Asia
D_Country	D_City	5256570	Oceania
A_Country	A_City	3253651	South America

Left Outer Join (Phép nối ngoài trái)



- **LEFT JOIN** trả về tất cả các giá trị từ bảng bên trái, cộng với các giá trị phù hợp từ bảng bên phải hoặc NULL trong trường hợp không có vị từ **JOIN** phù hợp

```
SELECT * FROM a LEFT OUTER JOIN b ON a.sid = b.cid;
```

Bảng A

sid	count	p_name
1	10	hadoop
3	20	spark
5	30	nosql
7	100	spark
9	3	NULL

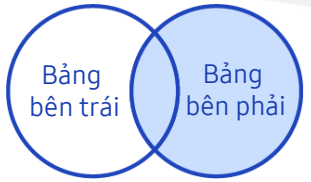
Bảng B

id	cid	service
1	1	hdfs
2	1	yarn
3	3	kudu
4	3	NULL
5	5	NULL
7	7	impala

<Kết quả>

a.sid	a.node_count	a.platform_name	b.id	b.cid	b.service
1	10	hadoop	1	1	hdfs
1	10	hadoop	2	1	yarn
3	20	spark	3	3	kudu
3	20	spark	4	3	
5	50	nosql	5	5	
7	100	spark	7	7	impala
9	3		NULL	NULL	NULL

Right Outer Join (Phép nối ngoài phải)



- **RIGHT JOIN** trả về tất cả các giá trị từ bảng bên phải, cộng với các giá trị phù hợp từ bảng bên trái hoặc NULL trong trường hợp không có vị từ nối phù hợp

```
SELECT * FROM a RIGHT OUTER JOIN b ON a.sid = b.cid;
```

Bảng A

sid	count	p_name
1	10	hadoop
3	20	spark
5	30	nosql
7	100	spark
9	3	NULL

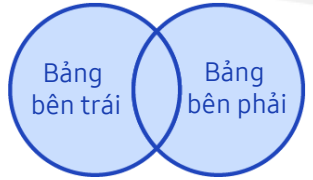
Bảng B

id	cid	service
1	1	hdfs
2	1	yarn
3	3	kudu
4	3	NULL
5	5	NULL
7	7	impala

<Kết quả>

a.sid	a.node_count	a.platform_name	b.id	b.cid	b.service
1	10	hadoop	1	1	hdfs
1	10	hadoop	2	1	yarn
3	20	spark	3	3	kudu
3	20	spark	4	3	
5	50	nosql	5	5	
7	100	spark	7	7	impala

Full Outer Join



- HiveQL **FULL OUTER JOIN** kết hợp các bản ghi của cả hai bảng bên ngoài bên trái và bên phải đáp ứng điều kiện JOIN

```
SELECT * FROM a FULL OUTER JOIN b ON a.sid = b.cid;
```

Bảng A

sid	count	p_name
1	10	hadoop
3	20	spark
5	30	nosql
7	100	spark
9	3	NULL

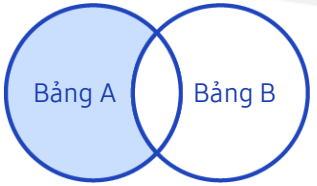
Bảng B

id	cid	service
1	1	hdfs
2	1	yarn
3	3	kudu
4	3	NULL
5	5	NULL
7	7	impala

‹Kết quả›

a.sid	a.node_count	a.platform_name	b.id	b.cid	b.service
1	10	hadoop	1	1	hdfs
1	10	hadoop	2	1	yarn
3	20	spark	3	3	kudu
3	20	spark	4	3	
5	50	nosql	5	5	
7	100	spark	7	7	impala
9	3		NULL	NULL	NULL

Left Semi Join



- Một nửa nối bên trái chỉ trả về các bản ghi từ bảng bên trái

```
SELECT * FROM a LEFT SEMI JOIN b ON a.sid = b.cid;
```

Bảng A

sid	count	p_name
1	10	hadoop
3	20	spark
5	30	nosql
7	100	spark
9	3	NULL

Bảng B

id	cid	service
1	1	hdfs
2	1	yarn
3	3	kudu
4	3	NULL
5	5	NULL
7	7	impala

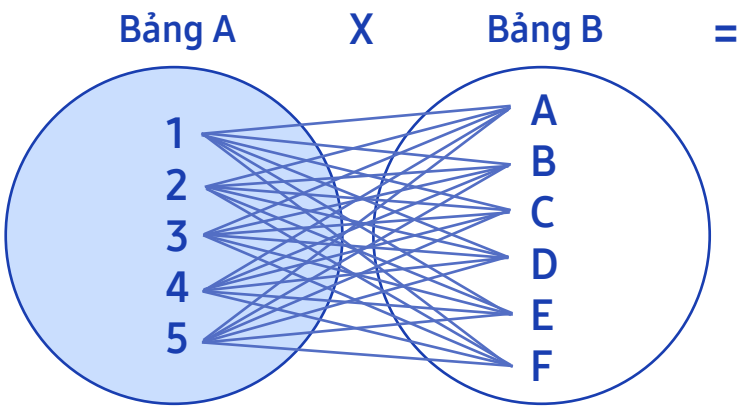
<Kết quả>

a.sid	a.node_count	a.platform_name
1	10	hadoop
3	20	spark
5	50	nosql
7	100	spark

Cross Join (Phép nối chéo)

- **CROSS JOIN** là một cách nối nhiều bảng trong đó tất cả các hàng hoặc bộ từ một bảng được ghép nối với các hàng và bộ từ một bảng khác

```
SELECT * FROM a CROSS JOIN b;
```



a.s id	a.node_count	a.platform_name	b.id	b.cid	b.service
1	10	hadoop	1	1	hdfs
1	10	hadoop	2	1	yarn
1	10	hadoop	3	3	kudu
1	10	hadoop	4	3	
1	10	hadoop	5	5	
1	10	hadoop	7	7	impala
3	20	spark	1	1	hdfs
3	20	spark	2	1	yarn
3	20	spark	3	3	kudu
3	20	spark	4	3	
3	20	spark	5	5	
3	20	spark	7	7	impala
5	50	nosql	1	1	hdfs
5	50	nosql	2	1	yarn
5	50	nosql	3	3	kudu
5	50	nosql	4	3	
5	50	nosql	5	5	
5	50	nosql	7	7	impala
7	100	spark	1	1	hdfs
7	100	spark	2	1	yarn
7	100	spark	3	3	kudu
7	100	spark	4	3	
7	100	spark	5	5	
7	100	spark	7	7	impala
9	3		1	1	hdfs
9	3		2	1	yarn
9	3		3	3	kudu
9	3		4	3	
9	3		5	5	
9	3		7	7	impala

30 rows selected (37.354 seconds)

Built-In Functions (Hàm tích hợp sẵn) (1/7)

- Hive và Impala cung cấp các hàm tương tự như các hàm tích hợp sẵn do SQL cung cấp.
  - Hầu hết đều giống SQL, nhưng một số dành riêng cho impala và HiveQL
  - Tên hàm không phân biệt chữ hoa chữ thường
- Danh sách tất cả các hàm và xem thông tin về một hàm cho HiveQL

```
hive> SHOW functions;
      ......
      --display the function list
hive> DESC function abs
```

```
0: jbdc:hive2://> desc function abs;
OK
+-----------------------------------+
|             tab_name              |
+-----------------------------------+
| abs(x) – returns the absolute value |
| of x                              |
+-----------------------------------+
1 row selected (0.038 seconds)
```

Built-In Functions (Hàm tích hợp sẵn) (2/7)

- Hàm toán học
  - Thực hiện phép tính số – **round(a,b), ceil(a) abs(a), rand(), sqrt(a), floor(a)**
- hàm ngày tháng (date)
  - Các hàm liên quan đến ngày chủ yếu được sử dụng để thay đổi định dạng đầu ra của ngày – **current_timestamp(), unix_timestamp(a), to_date(a)**
- Hàm chuỗi (string)
  - Có nhiều loại hàm khác nhau thao tác với chuỗi - **concat(a, b), concat_ws(a, array,<string>), substring(a, b, c) , upper(a), lower(a), trim(a)**

```
0: jbdc:hive2://> select concat_ws('@', 'student', 'samsung.com') as
e_mail;
OK
+----------------------+
|        e_mail        |
+----------------------+
| student@samsung.com  |
+----------------------+

1 row selected (0.143 seconds)
0: jbdc:hive2://> select upper('samsung') as upper;
OK
+-----------+
|   upper   |
+-----------+
| SAMSUNG   |
+-----------+

1 row selected (0.167 seconds)
0: jbdc:hive2://> select trim('    s t u d e n t    ') as trim;
OK
+-------------+
|    trim     |
+-------------+
| s t u d e n t |
+-------------+

1 row selected (0.141 seconds)
```

Built-In Functions (Hàm tích hợp sẵn) (3/7)

- Hàm cast (Hàm đổi kiểu dữ liệu)
  - Sử dụng cast để thay đổi loại của một giá trị nhất định – cast(expr as <type>)
- Hàm điều kiện
  - Các hàm có điều kiện có thể được sử dụng để thay đổi đầu ra của một giá trị dựa trên một điều kiện– isnull(), coalesce(), if()

```
0:  jbdc:hive2://> select if(10 > 1, 'a', 'b');
OK
┌─────┐
│ _c0 │
├─────┤
│ a   │
└─────┘
1 row selected (0.139 seconds)
0: jbdc:hive2://> select if(10 < 1, 'a', 'b');
OK
┌─────┐
│ _c0 │
├─────┤
│ b   │
└─────┘
1 row selected (0.143 seconds)
```

True: nếu câu điều kiện đúng
False: nếu điều kiện sai

Trả về giá trị khác null đầu tiên theo thứ tự

```
0:  jbdc:hive2://>select coalesce(null, '1',
'2');
OK
┌─────┐
│ _c0 │
├─────┤
│ 1   │
└─────┘
1 row selected (0.53 seconds)
0: jbdc:hive2://> select coalesce(null, null,
'2');
OK
┌─────┐
│ _c0 │
├─────┤
│ 2   │
└─────┘
1 row selected (0.206 seconds)
0:  jbdc:hive2://> select coalesce(10, '1', '2');
OK
┌─────┐
│ _c0 │
├─────┤
│ 10  │
└─────┘
```

Toán tử tích hợp (4/7)

Toán tử quan hệ

Toán tử	Toán hạng	Toán tử	Toán hạng
A = B	tất cả các loại ban đầu	A IS NULL	tất cả các loại
A != B	tất cả các loại ban đầu	A IS NOT NULL	tất cả các loại
A < B	tất cả các loại ban đầu	A <= B	tất cả các loại ban đầu
A > B	tất cả các loại ban đầu	A >= B	tất cả các loại ban đầu

Toán tử tích hợp (5/7)

Toán tử số học

Toán tử	Toán hạng	Mô tả
A+B	Tất cả các loại số	Đưa ra kết quả của việc thêm A và B
A-B	Tất cả các loại số	Đưa ra kết quả trừ B từ A
A*B	Tất cả các loại số	Đưa ra kết quả của phép nhân A và B
A/B	Tất cả các loại số	Đưa ra kết quả chia B từ A
A%B	Tất cả các loại số	Đưa ra lời nhắc kết quả chia B từ A
A&B	Tất cả các loại số	Đưa ra kết quả của bitwise AND của A và B
A | B	Tất cả các loại số	Đưa ra kết quả của OR theo bit của A và B
~A	Tất cả các loại số	Đưa ra kết quả của bitwise NOT của A

Toán tử tích hợp (6/7)

Toán tử logic

Toán tử	Toán hạng	Mô tả
A AND B	Boolean	TRUE nếu cả A và B đều TRUE, ngược lại là FALSE
A && B	Boolean	Tương tự như A AND B
A OR B	Boolean	TRUE nếu A hoặc B hoặc cả hai đều TRUE, nếu không thì FALSE
A || B	Boolean	Tương tự như A OR B

Toán tử tích hợp (7/7)

Toán tử kiểu phức hợp

Toán tử	Toán hạng	Mô tả
A[n]	A là một Mảng và n là một số nguyên	trả về phần tử thứ n trong mảng A. Phần tử đầu tiên có chỉ số 0, ví dụ: nếu A là một mảng bao gồm ['spark', 'hadoop'] thì A[0] trả về 'spark' và A[1] trả về 'hadoop'
M[key]	M là Map<K, V> và key có kiểu K	trả về giá trị tương ứng với khóa trong bản đồ chẳng hạn, nếu M là bản đồ bao gồm {'h' -> 'hadoop', 's' -> 'spark', 'n' -> 'nosql'} thì M ['n'] trả về 'nosql'
S.x	S là một cấu trúc	trả về trường x của S, ví dụ: for struct hadoop {int har, string doo} hadoop.har trả về số nguyên được lưu trữ trong trường har của cấu trúc

Sử dụng Cột MAP (1/2)

- Tất cả các khóa MAP phải là một loại dữ liệu và tất cả các giá trị phải là một loại dữ liệu
  - MAP<KEY-TYPE,VALUE-TYPE>

```
CREATE TABLE product_map (
  id int,
  platform string,
  detail_info map<string, string>
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ':';
DESC product_map;
```

```
+---------------+---------------------+---------+
|    col_name   |      data_type      | comment |
+---------------+---------------------+---------+
| id            | int                 |         |
| platform      | string              |         |
| detail_info   | map<string, string> |         |
+---------------+---------------------+---------+
```

Sử dụng Cột MAP (2/2)

- MAP tương tự như ARRAY, ngoại trừ thay vì sử dụng số để truy cập các giá trị, chúng ta sử dụng nhãn

```
SELECT id, platform, detail_info['node'] AS number_of_node, detail_info['location'] AS location
FROM product_map;
```

id	platform	number_of_node	location
1	hadoop	10	Seoul
5	spark	20	Busan
7	nosql	30	Jeju

Kết quả

chọn ALL *

product_map.id	product_map.platform	product_map.detail_info
1	hadoop	{"node":"10", "name":"Hdfs ", "location":"Seoul"}
5	spark	{"node":"20", "name":"Kafka", "location":"Busan"}
7	nosql	{"node":"30","name":"Hbase", "location":"Jeju"}

Sử dụng các cột STRUCT (1/2)

- STRUCT lưu trữ một số trường được đặt tên cố định
  - Mỗi trường có thể có một kiểu dữ liệu khác nhau

```
CREATE TABLE product_complex (
  id integer,
  name string
  detail STRUCT<node:int, name:string, location:string>
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY ':';
DESC product_complex;
```

col_name	data_type	comment
id	int	
platform	string	
detail_info	struct<node:int,name:string,location:string>	

Sử dụng các cột STRUCT (2/2)

- Loại Struct có tên và loại

```
SELECT id, platform, detail_info.name, detail_info.node FROM product_complex;
```

id	platform	name	node
1	hadoop	Hdfs	10
5	spark	Kafka	20
7	nosql	HBase	30

Kết quả

Chọn *

product_complex.id	product_complex.platform	product_complex.detail_info
1	hadoop	{"node":"10","name":"Hdfs ","location":"Seoul"}
5	spark	{"node":"20","name":"Kafka","location":"Busan"}
7	nosql	{"node":"30","name":"Hbase","location":"Jeju"}

Tập hợp và phân chia cửa sổ

- Tập hợp (Aggregate) và phân chia cửa sổ (Windowing) (hàm phân tích) đều sử dụng các giá trị trên nhiều hàng
- Hàm tập hợp
  - Hàm tập hợp trong Hive có thể được sử dụng có hoặc không có hàm **GROUP BY**
  - Hàm Hive Aggregate là hàm tích hợp được sử dụng nhiều nhất
- Hàm phân chia cửa sổ
  - Tính năng tạo cửa sổ cho phép các hàm tạo cửa sổ trên tập dữ liệu để thực hiện thao tác tổng hợp như COUNT, AVG, MIN, MAX
  - Các hàm phân tích khác như LEAD, LAG, FIRST_VALUE và LAST_VALUE

Hàm tập hợp tích hợp

- Danh sách hàm
  - **COUNT (*):** Đếm tất cả các hàng
  - **COUNT(column):** Đếm tất cả các hàng có trường không phải NULL
  - **COUNT(cột riêng biệt):** Đếm tất cả các hàng có trường là duy nhất và không phải NULL
  - **MAX():** Trả về giá trị lớn nhất
  - **MIN():** Trả về giá trị nhỏ nhất
  - **SUM():** Cộng tất cả các giá trị được cung cấp và trả về kết quả
  - **AVG():** Trả về giá trị trung bình của tất cả các giá trị được cung cấp

Windows (Cửa sổ)

- Một cửa sổ xác định một tập hợp các hàng trong một bảng
- OVER (đặc tả cửa sổ) chỉ định các cửa sổ sẽ áp dụng chức năng tổng hợp hoặc cửa sổ

Dịch vụ PATRION BY

id	name	service	cost	rank
10	HUE	HADOOP	3200	1
9	IMPALA	HADOOP	5400	1
8	KUDU	HADOOP	5000	1
2	YARN	HADOOP	1500	1
1	HDFS	HADOOP	1800	1
11	MONGODB	NoSQL	5000	1
4	Cassandra	NoSQL	1300	1
3	HBASE	NoSQL	2000	1
7	AWS	PUB-CLOUD	8000	1
6	AZURE	PUB-CLOUD	5000	1
12	KAFKA	SPARK	8000	1
5	SPARKSQL	SPARK	1500	1

Cửa sổ

Cửa sổ

Cửa sổ

Hàm Windowing

- Cú pháp của truy vấn với cửa sổ:

```
SELECT <columns_name>, <aggregate>(column_name) OVER (<windowing specification>)
FROM <table_name>;
```

- column_name – tên cột của bảng
- Tập hợp – Bất kỳ (các) hàm tổng hợp nào như COUNT, AVG, MIN, MAX
- Thông số kỹ thuật cửa sổ – Nó bao gồm những điều sau đây:
  - **PARTITON BY** – Lấy (các) cột của bảng làm tham chiếu
  - **ORDER BY** – Đã chỉ định Thứ tự của (các) cột Tăng dần hoặc Giảm dần
  - Khung – Đã chỉ định ranh giới của khung theo chỉ số và giá trị kết thúc. Ranh giới có thể là một loại **RANGE** hoặc **ROW**, theo sau là **PRECEDING**, **FOLLOWING** và bất kỳ giá trị nào

RANK Functions (Hàm xếp hạng) (1/3)

ROW_NUMBER

- Một trong những chức năng phân tích, chỉ định số duy nhất cho mỗi hàng dựa trên giá trị cột được sử dụng trong OVER

RANK

- Hàm phân tích tích hợp được sử dụng để xếp hạng bản ghi trong một nhóm hàng

DENSE_RANK

- Thứ hạng của giá trị hiện tại trong cửa sổ (với thứ hạng liên tiếp)

RANK Functions (Hàm xếp hạng) (2/3)

RANK

```
SELECT name, service, cost,
RANK() OVER(ORDER BY cost DESC) AS costRank
FROM Projects;
```

name	company	cost	costrank
KAFKA	SPARK	8000	1
AWS	PUB-CLOUD	8000	1
IMPALA	HADOOP	5400	3
MONGODB	NoSQL	5000	4
KUDU	HADOOP	5000	4
AZURE	PUB-CLOUD	5000	4
HUE	HADOOP	3200	7
HBASE	NoSQL	2000	8
HDFS	HADOOP	1800	9
SPARKSQL	SPARK	1500	10
YARN	HADOOP	1500	10
Cassandra	NoSQL	1300	12

RANK Functions (Hàm xếp hạng) (3/3)

- **ROW_NUMBER** vs **DENSE_RANK** vs **RANK**

```
SELECT name, service, cost, RANK() OVER(ORDER BY cost DESC) AS rank,
DENSE_RANK() OVER(ORDER BY cost DESC) AS dense_rank,
ROW_NUMBER() OVER(ORDER BY cost DESC) AS row_number FROM projects;
```

name	company	cost	rank	dense_rank	row_number
KAFKA	SPARK	8000	1	1	1
AWS	PUB-CLOUD	8000	1	1	2
IMPALA	HADOOP	5400	3	2	3
MONGODB	NoSQL	5000	4	3	4
KUDU	HADOOP	5000	4	3	5
AZURE	PUB-CLOUD	5000	4	3	6
HUE	HADOOP	3200	7	4	7
HBASE	NoSQL	2000	8	5	8
HDFS	HADOOP	1800	9	6	9
SPARKSQL	SPARK	1500	10	7	10
YARN	HADOOP	1500	10	7	11
Cassandra	NoSQL	1300	12	8	12

Hàm RANK và ROW_NUMBER

- Xếp hạng của từng dự án theo chi phí, trong mỗi dịch vụ là gì?

```
SELECT name, service, cost, rank() OVER(PATRITION BY service ORDER BY cost) AS Rank,
row_number() OVER(PATRITION BY service ORDER BY cost) AS row_number
FROM projects;
```

name	company	cost	rank	row_number
YARN	HADOOP	1500	1	1
HUE	HADOOP	3200	2	2
HDFS	HADOOP	3200	2	3
KUDU	HADOOP	5000	4	4
IMPALA	HADOOP	5400	5	5
Cassandra	NoSQL	1300	1	1
HBASE	NoSQL	2000	2	2
MONGODB	NoSQL	5000	3	3
AWS	PUB-CLOUD	5000	1	1
AZURE	PUB-CLOUD	5000	1	2
KAFKA	SPARK	8000	1	1
SPARKSQL	SPARK	8000	1	2

Các hàm windowing khác

- **LEAD**
  - Số lượng hàng để dẫn có thể được chỉ định tùy chọn. Nếu số lượng hàng dẫn đầu không được chỉ định, thì dẫn đầu là một hàng
  - Trả về null khi vị trí dẫn đầu của hàng hiện tại vượt quá cuối cửa sổ
- **LAG**
  - Số lượng hàng trễ có thể được chỉ định tùy ý. Nếu số lượng hàng trễ không được chỉ định, thì độ trễ là một hàng
  - Trả về null khi độ trễ cho hàng hiện tại kéo dài trước khi bắt đầu cửa sổ
- **FIRST_VALUE**
  - Điều này có nhiều nhất hai tham số. Tham số đầu tiên là cột mà bạn muốn có giá trị đầu tiên, tham số thứ hai (không bắt buộc) phải là một giá trị boolean, giá trị này là false theo mặc định. Nếu được đặt thành đúng, nó sẽ bỏ qua các giá trị null
- **LAST_VALUE**
  - Điều này có nhiều nhất hai tham số. Tham số đầu tiên là cột mà bạn muốn có giá trị cuối cùng, tham số thứ hai (không bắt buộc) phải là một giá trị boolean, giá trị này là false theo mặc định. Nếu được đặt thành đúng, nó sẽ bỏ qua các giá trị null

So sánh với Apache Hive và Impala

- So sánh Hive và Impala với CSDL quan hệ

[Thảo luận]

Đặc điểm	Hive	Impala	RDBMS
Ngôn ngữ truy vấn	Hive QL	Impala SQL	SQL(full)
Cập nhật	Không*	Không*	Có
Xóa	Không*	Không*	Có
Giao dịch	Không*	Không*	Có
Hỗ trợ chỉ mục	Có giới hạn	Có giới hạn	Extensive
Độ trễ	Cao	Thấp	Rất thấp
Kích thước dữ liệu	Petabytes	Petabytes	Terabytes
Chi phí lưu trữ	Rất thấp	Thấp	Rất cao

Câu hỏi ôn tập

[Câu hỏi]

Câu hỏi 1

- Một số lý do khiến bạn không thay thế cơ sở dữ liệu quan hệ điển hình bằng Hive là gì?

Câu hỏi 2

- Hai lợi ích mà Hive và Hadoop có đối với các hệ thống kho dữ liệu điển hình là gì?

Câu hỏi 3

- Khi bạn truy vấn một bảng trong Hive, dữ liệu đang được truy vấn nằm ở đâu?

Câu hỏi 4

- Sự khác biệt giữa bảng bên ngoài và bảng được quản lý là gì?

Biểu thức chính quy

[Thảo luận]

Một biểu thức chính quy (regex) khớp với một mẫu trong văn bản.

Một điều đáng chú ý đối với những người biết regex từ một ngôn ngữ khác Java (ví dụ: perl hoặc awk) là dấu gạch chéo ngược là một ký tự đặc biệt trong Java và phải được thoát bằng dấu gạch chéo ngược khác trong biểu thức chính quy.

Trường hợp

- look
- \\d
- \\d{5}
- \\d\\s\\w+
- \\w{5,9}
- .?\\.
- .*\\.

[Lab7]
Xử lý bảng phân vùng cho hiệu suất

[Lab8]
Làm việc với kiểu dữ liệu phức tạp

[Lab9]
Truy vấn phức tạp

Bài 4.

Kiến trúc phân tích truyền dữ liệu

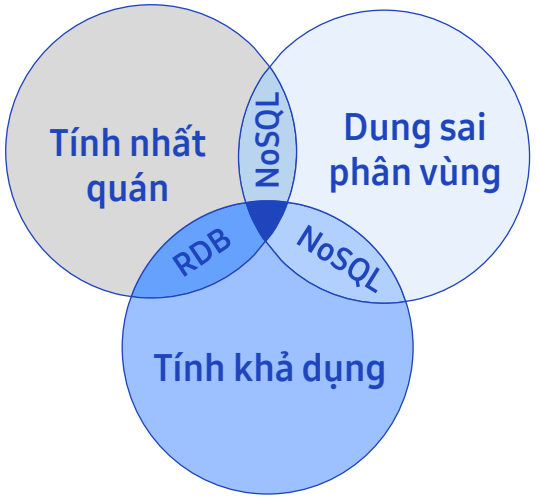
Phân tích Big Data

Bài 4

Kiến trúc phân tích truyền dữ liệu

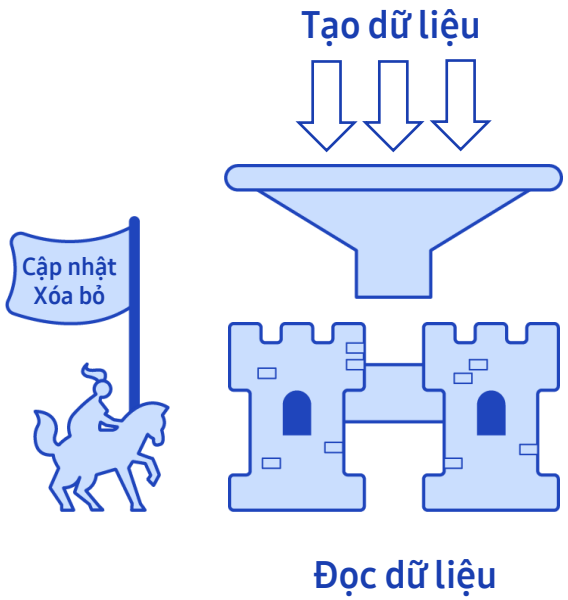
Các hạn chế của định lý CAP

- Xem lại định lý CAP
  - Về tính nhất quán, tính khả dụng và dung sai phân vùng, chỉ có 2 tính năng có thể được đưa vào bất kỳ hệ thống cơ sở dữ liệu nào
- Phân vùng là một tính năng PHẢI để xử lý quy mô lớn
- Tính nhất quán trên tính khả dụng
  - Máy chủ không thể truy cập được cho đến khi tất cả đều nhất quán
  - Trong các hệ thống hiện đại, các máy chủ có thể được phân phối trên toàn cầu
  - Bản thân việc cố gắng ghi đệm có thể dẫn đến sự không nhất quán và hơn nữa, rất khó thực hiện
- Tính khả dụng hơn tính nhất quán
  - Cơ sở dữ liệu nhất quán cuối cùng
  - Những người dùng khác nhau có thể đọc dữ liệu khác nhau và gây ra sự không nhất quán khi họ viết
  - Làm "đọc sửa" rất vất vả và khó khăn



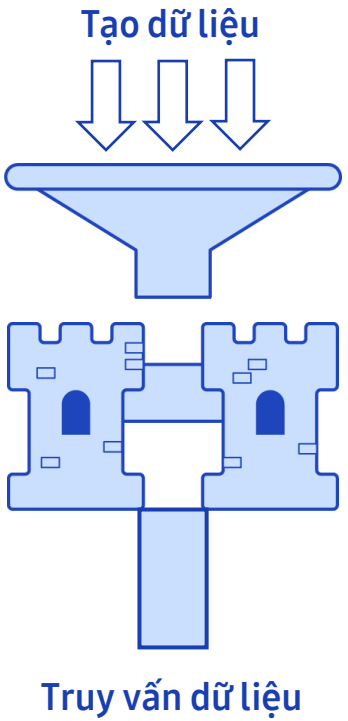
Nguồn gốc của Kiến trúc Lambda (1/2)

- Trở lại năm 2011, khi Big Data và Hadoop đang được triển khai, Nathan Marz đã viết một dòng tweet về một cách khả thi để đánh bại định lý CAP
- Nói một cách đơn giản, phương pháp luận của anh ấy là xem tất cả các truy vấn khi thực hiện một số loại chức năng trên tất cả dữ liệu
  - Truy vấn = Hàm (Tất cả dữ liệu)
- Sau đó, anh ấy phân loại lại dữ liệu để có mối liên hệ nào đó với thời gian
  - Địa chỉ của Sally là Chicago - chỉ hợp lệ trong một khoảng thời gian nhất định
  - Địa chỉ hiện tại mới của Sally là Los Angeles - hiện tại hợp lệ
  - Anh ấy nói rằng việc Sally chuyển đến Los Angeles không thay đổi sự thật rằng cô ấy đã sống ở Chicago.
- Dữ liệu bất biến
  - Dữ liệu phải là bất biến và không được ghi đè để có được bức tranh hoàn chỉnh
  - Các hoạt động CRUD (Tạo, Đọc, Cập nhật và Xóa) hiện chỉ là CR



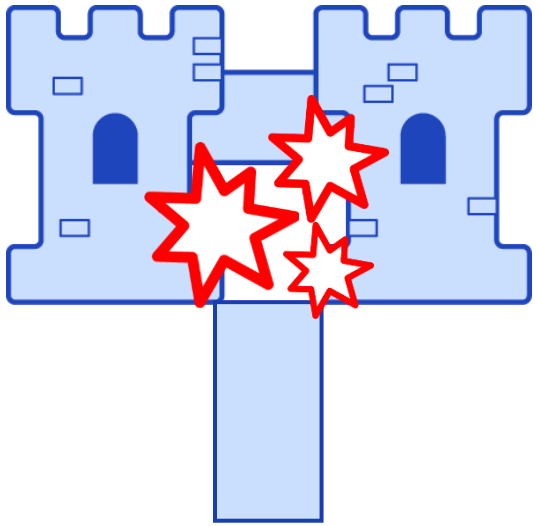
Nguồn gốc của Kiến trúc Lambda (2/2)

- Ông Marz định nghĩa truy vấn là
  - Truy vấn = Hàm (Tất cả dữ liệu)
- Nếu không có vấn đề về độ trễ, tất cả các hoạt động truy vấn bao gồm tập hợp và tham gia có thể được xem như đơn giản là thực hiện các chức năng cần thiết trên toàn bộ tập dữ liệu
- Bằng cách xác định dữ liệu là bất biến và truy vấn là các hàm được thực hiện trên toàn bộ tập dữ liệu, sự phức tạp của định lý CAP trở nên khá đơn giản
- Tính nhất quán trên tính khả dụng
  - Hoàn thành nhanh hơn nhiều vì chỉ tạo dữ liệu mới
- Tính khả dụng hơn tính nhất quán
  - Các truy vấn có thể tạo ra kết quả không chính xác nhưng một khi cuối cùng nhất quán, nó sẽ tạo ra kết quả chính xác.
  - Vì dữ liệu là bất biến, chúng tôi không còn gặp sự cố "đọc sửa chữa" nữa



Định lý CAP đã bị đánh bại?

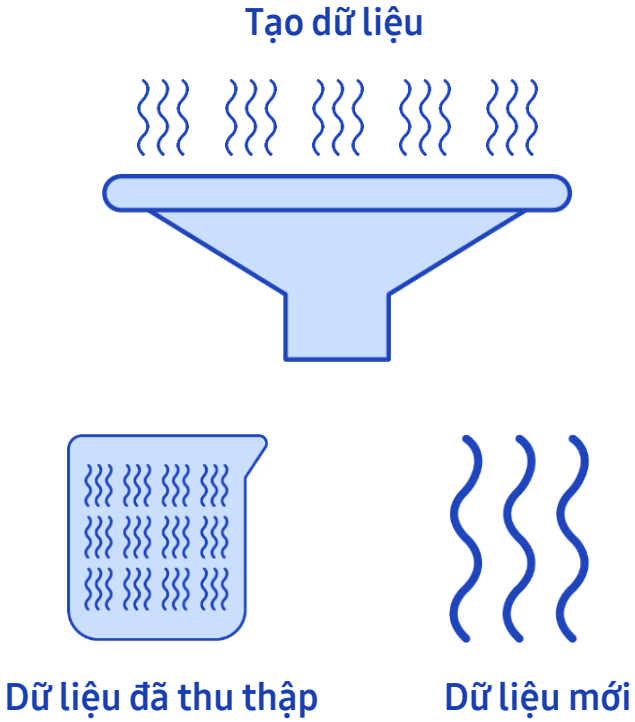
- Tuy nhiên, việc thực hiện truy vấn trên toàn bộ tập dữ liệu mỗi lần mà không phải lo lắng về độ trễ là không thực tế và cũng không khả thi!



Truy vấn dữ liệu

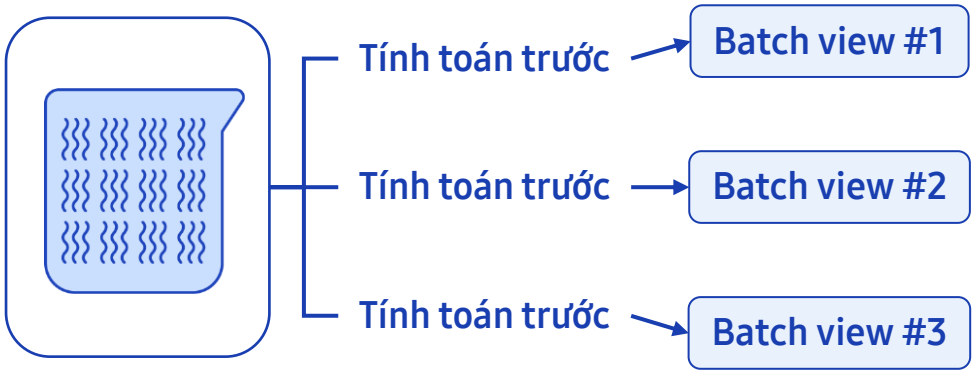
Chúng ta có thể thực sự gần nhau không?

- Không thể áp dụng hàm trên toàn bộ tập dữ liệu mỗi khi chúng ta truy vấn nó
- Người bạn cũ của chúng ta - chia những vấn đề lớn thành những vấn đề nhỏ hơn
  - Tập dữ liệu bất biến có thể được chia thành hai tập dữ liệu khác nhau tại bất kỳ thời điểm nào
  - Có những dữ liệu đã được thu thập cho đến nay. . .
  - Và bất kỳ dữ liệu mới nào đã được tạo kể từ đó
- Chia nhỏ vấn đề cho phép chúng ta tiếp cận giải pháp mong muốn
  - Giờ đây, chúng tôi có thể làm việc với từng bộ riêng biệt để đáp ứng các yêu cầu về độ trễ



Lớp hàng loạt

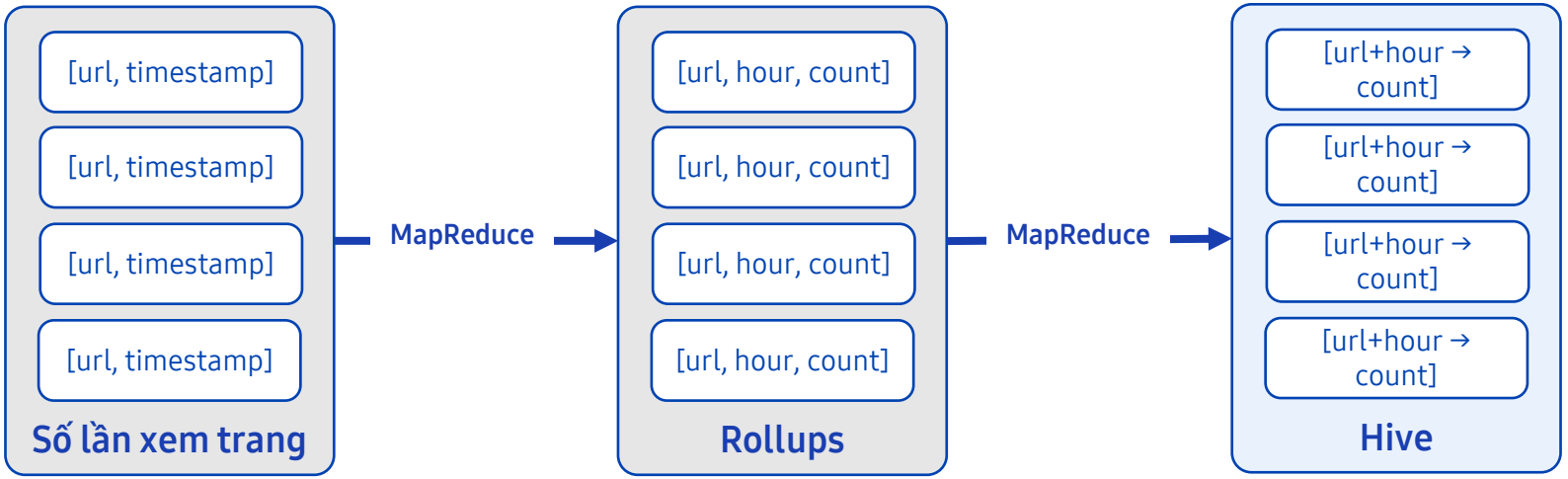
- Nới lỏng các yêu cầu một chút
  - Truy vấn sẽ chính xác trong một vài *delta*
- Chúng tôi không muốn lưu từng luồng dữ liệu đến trong thời gian thực
  - Quá kém hiệu quả
  - Cho phép dữ liệu phát trực tuyến mới chồng chất cho *delta* và sau đó tổng hợp thành dữ liệu đã thu thập
- Tạo kết quả được tính toán trước trên tập dữ liệu đã thu thập sau mỗi lần cập nhật *delta*
- Yêu cầu:
  - Hệ thống lưu trữ có thể lưu trữ một tập dữ liệu lớn và không ngừng phát triển
  - Có thể tính toán các chức năng trên tập dữ liệu theo cách có thể mở rộng



Quy trình làm việc trước khi tính toán

Ví dụ về tiền tính toán lớp hàng loạt

- Xây dựng ứng dụng để phân tích luồng nhấp chuột và truy cập web
- Theo dõi lượt xem trang trên một trang web
- Có thể truy vấn số lượt xem trang trong bất kỳ khoảng thời gian nào
  - Cung cấp mức độ chi tiết trong vòng một giờ



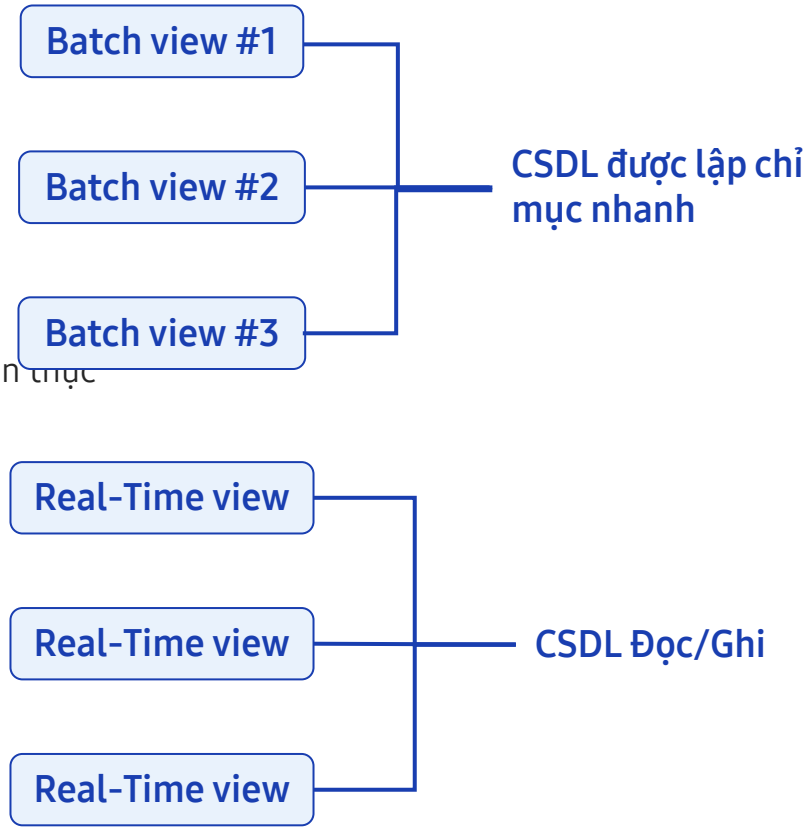
Ví dụ về quy trình làm việc hàng loạt

Lớp tốc độ hoặc Lớp thời gian thực

- Chia nhỏ vấn đề
  - Lớp lô đã lấy bất kỳ dữ liệu nào cũ hơn *delta*
  - Điều này cho phép lớp tốc độ xử lý một tập dữ liệu nhỏ hơn nhiều
  - Lớp tốc độ chỉ cần xử lý một giá trị dữ liệu *delta*
- Yêu cầu
  - Một công cụ xử lý luồng

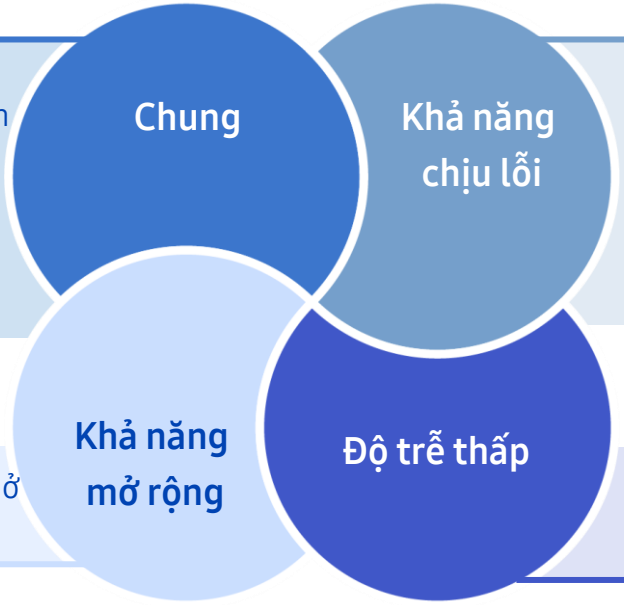
Lớp phục vụ

- Từ lớp hàng loạt:
  - Lập chỉ mục kết quả tính toán trước từ Lớp hàng loạt
  - Các kết quả phải được truy cập nhanh chóng bằng các ứng dụng
  - Cơ sở dữ liệu tra cứu được lập chỉ mục nhanh
- Từ lớp tốc độ:
  - Cập nhật gia tăng chế độ xem thời gian thực
  - Cơ sở dữ liệu Đọc/Ghi để cập nhật thông tin trạng thái theo thời gian thực



Kiến trúc Lambda là gì?

- Kiến trúc Lambda xác định một phương pháp xử lý mà không cần cam kết với các công cụ triển khai thực tế
- Kiến trúc sư được tự do sửa đổi và sử dụng các công cụ khác nhau phù hợp nhất với trường hợp sử dụng của họ



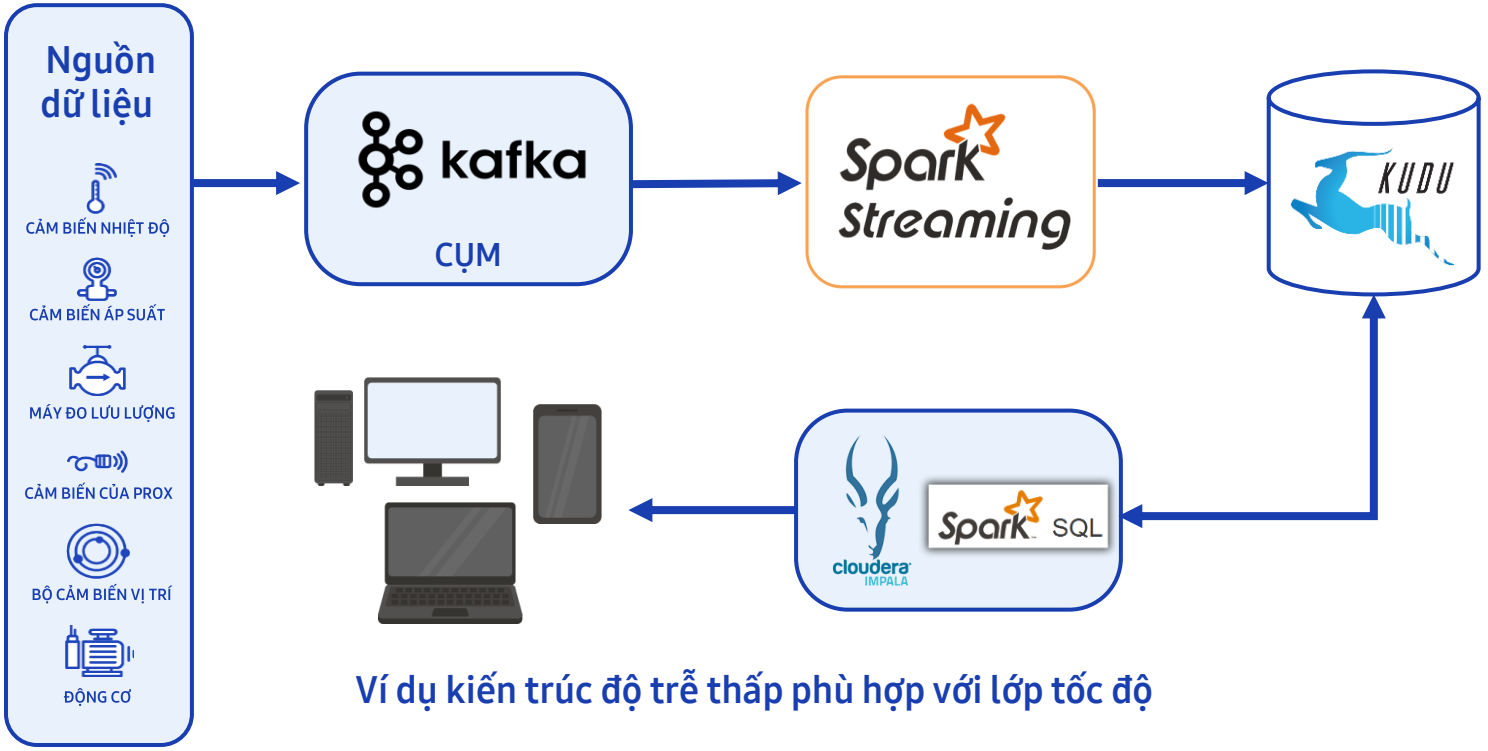
- Phục hồi từ lỗi phần cứng
- Phục hồi dễ dàng từ những sai lầm của con người

- Kiến trúc mở rộng thay vì kiến trúc mở rộng

- Cung cấp kết quả truy vấn trong thời gian thực
- Cho phép truy vấn đặc biệt

Kiến trúc phân tích độ trễ thấp

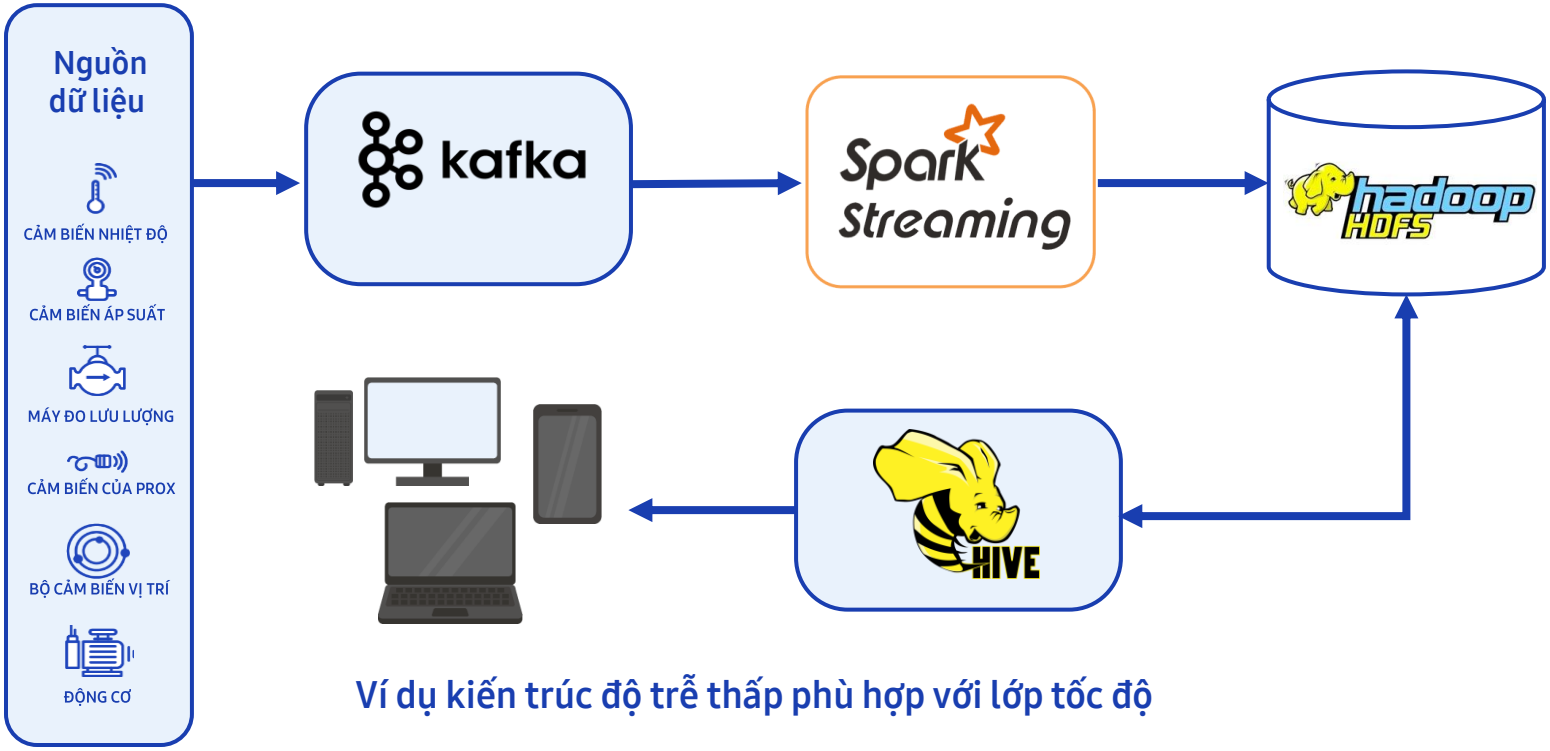
- Ví dụ kiến trúc độ trễ thấp phù hợp với lớp tốc độ



Ví dụ kiến trúc độ trễ thấp phù hợp với lớp tốc độ

Kiến trúc phân tích thông lượng cao

- Ví dụ về kiến trúc thông lượng cao phù hợp với lớp hàng loạt



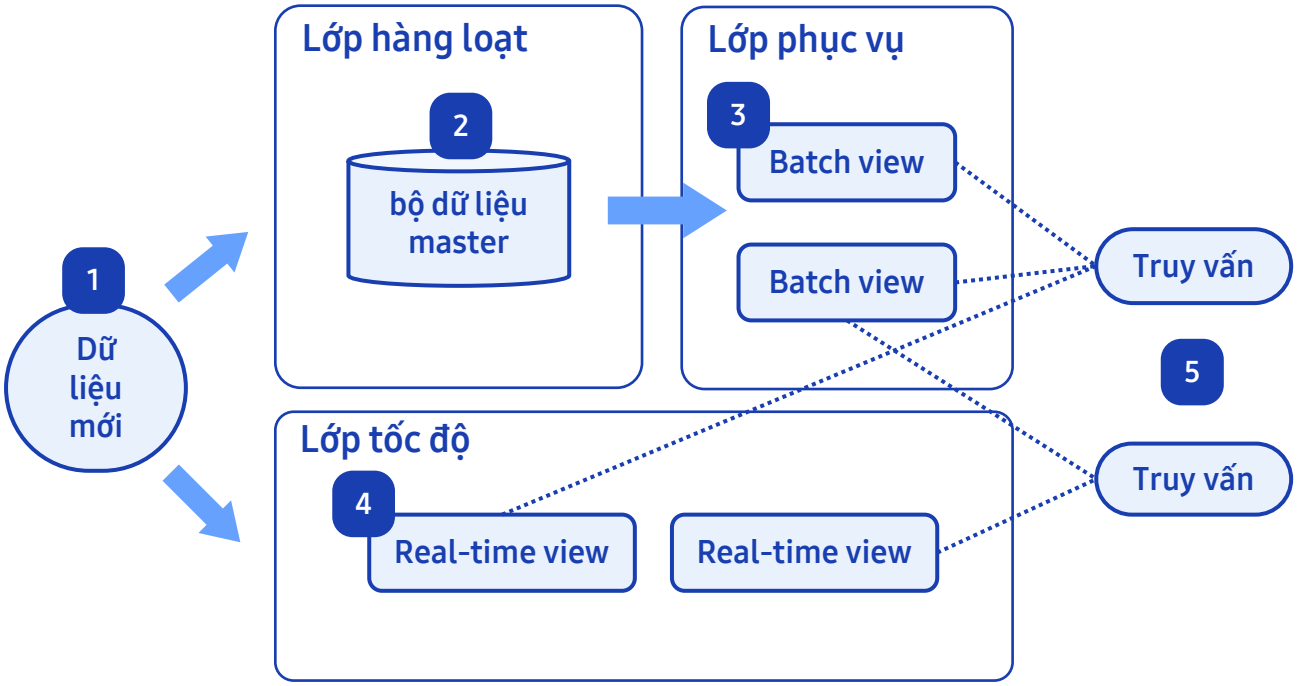
Ví dụ kiến trúc độ trễ thấp phù hợp với lớp tốc độ

Kiến trúc Lambda có khả năng chịu lỗi

- Khả năng chịu lỗi của con người và máy móc
- Lỗi chu kỳ phát triển
  - Lỗi mã hóa và phát triển
  - Sửa lỗi và chạy lại các truy vấn tính toán trước lớp hàng loạt
- Lỗi nhập dữ liệu con người
  - Xóa dữ liệu xấu và tính toán lại các truy vấn trước
- Hadoop HDFS và YARN/MapReduce có khả năng chịu lỗi tích hợp
  - HDFS ngăn ngừa mất dữ liệu không mong muốn
  - YARN tính toán lại các công việc khi cần thiết

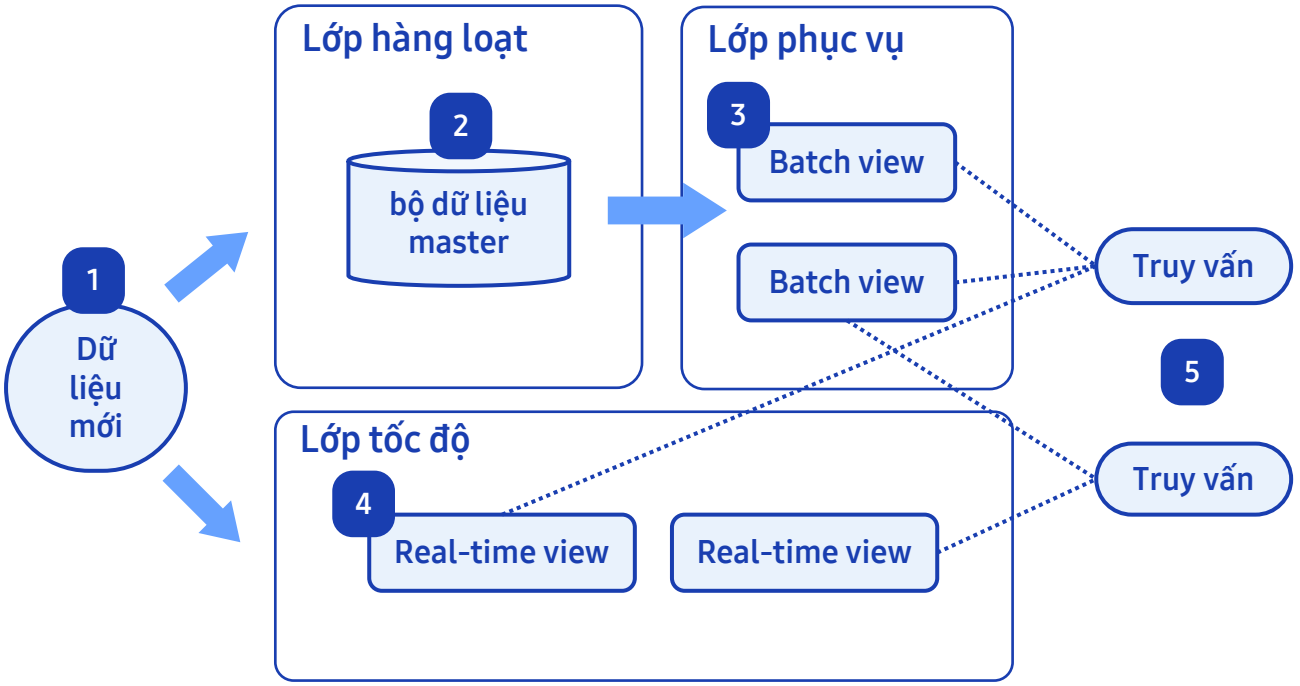
Các bước kiến trúc Lambda (1/5)

1. Tất cả **dữ liệu** vào hệ thống đều được gửi đến cả lớp lô và lớp tốc độ để xử lý



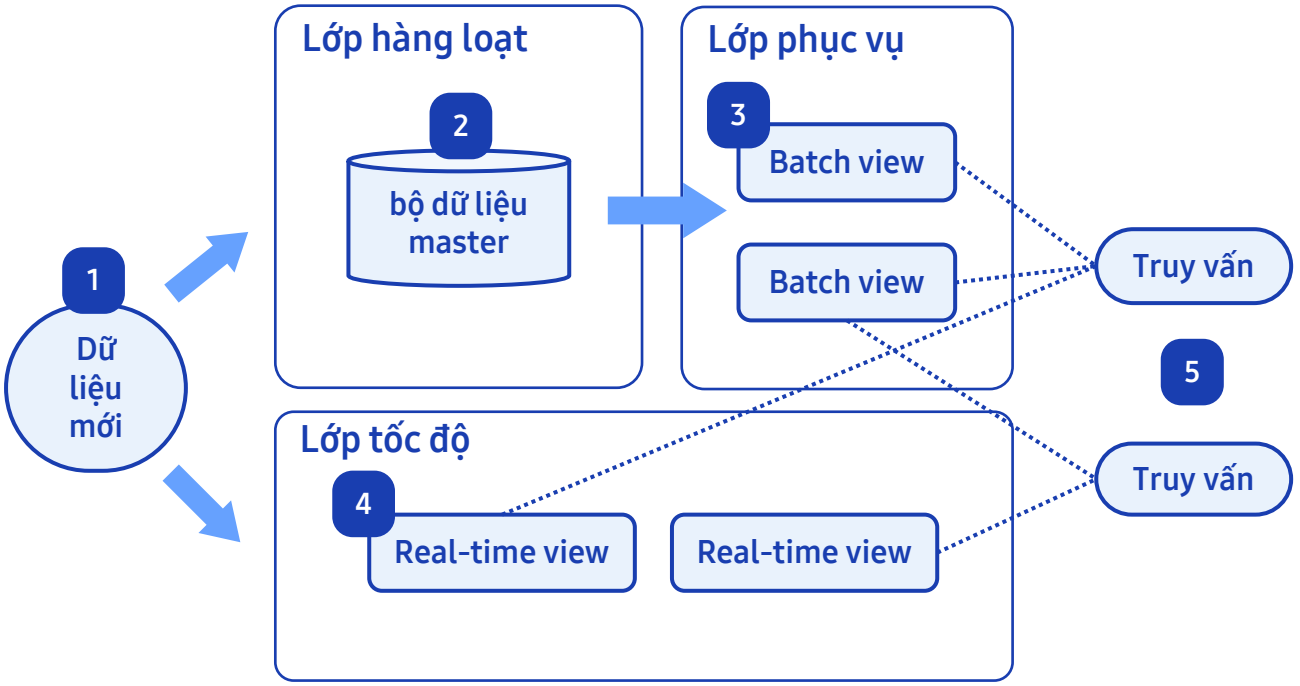
Các bước kiến trúc Lambda (2/5)

- 2. **Lớp hàng loạt** có hai chức năng: (i) quản lý tập dữ liệu chính (một tập hợp dữ liệu thô không thể thay đổi, chỉ nối thêm) và (ii) để tính toán trước các chế độ xem lô



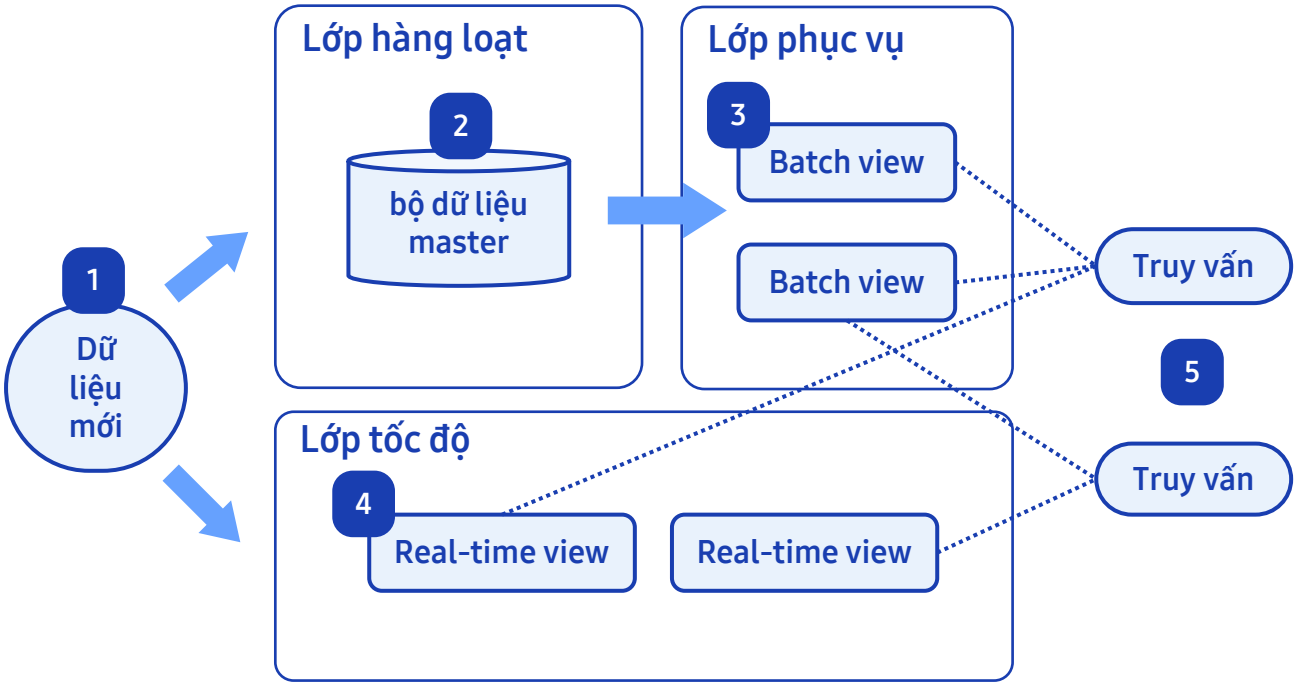
Các bước kiến trúc Lambda (3/5)

- 3. **Lớp phân phối lập chỉ mục** các chế độ xem hàng loạt để chúng có thể được truy vấn theo cách đặc biệt, có độ trễ thấp



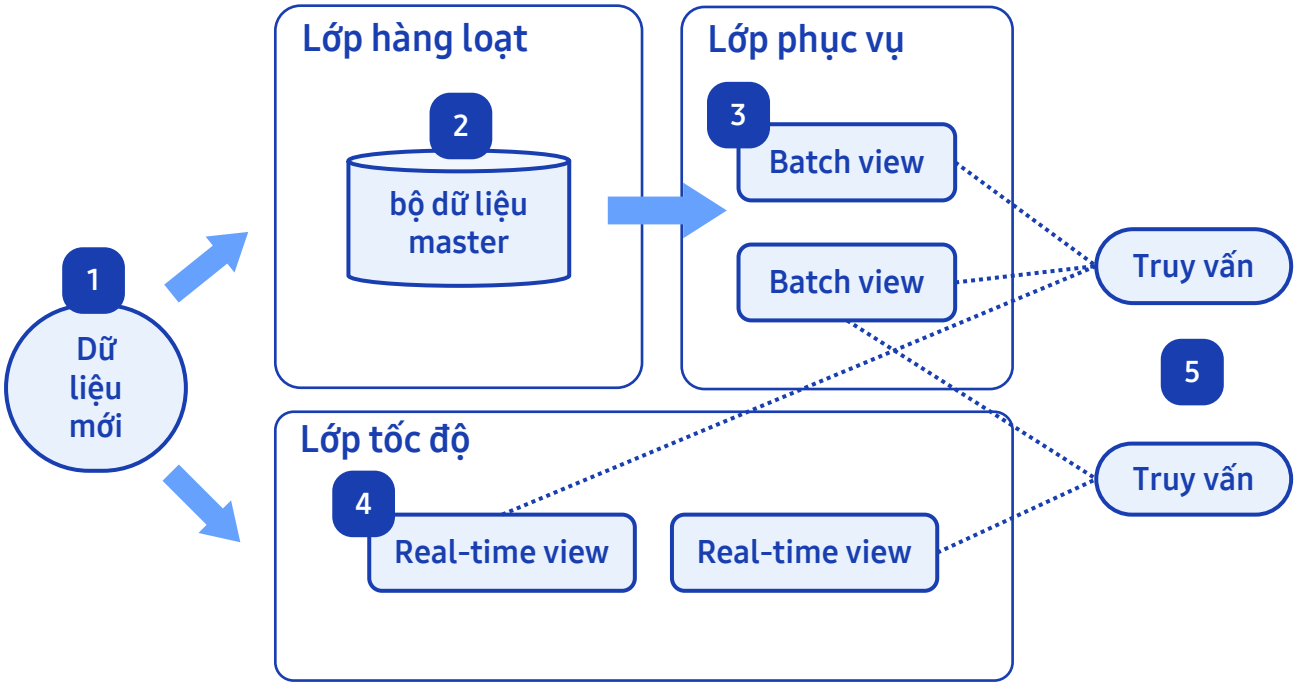
Các bước kiến trúc Lambda (4/5)

- 4. **Lớp tốc độ** bù cho độ trễ cao của các bản cập nhật đối với lớp phân phát và chỉ xử lý dữ liệu gần đây



Các bước kiến trúc Lambda (5/5)

- 5. Mọi **truy vấn** đến có thể được trả lời bằng cách hợp nhất kết quả từ chế độ xem hàng loạt và chế độ xem thời gian thực



Bài 4

Kiến trúc phân tích truyền dữ liệu

The Good, The Bad and The Ugly

- Kiến trúc Lambda, The Good, The Bad and The Ugly
- The Good
  - Dựa trên kho dữ liệu bất biến
  - Làm nổi bật và cung cấp một phương pháp để xử lý lại dữ liệu
- The Bad
  - Lập trình trong một nền tảng phân tán rất phức tạp
  - Duy trì hai hệ thống khác nhau và phát triển mã cho từng hệ thống là cực kỳ khó khăn
- The Ugly
  - Độ phức tạp vận hành của việc triển khai và duy trì Kiến trúc Lambda quá khó
  - Tại sao hệ thống xử lý luồng không thể được cải thiện để xử lý toàn bộ vấn đề

Nguồn gốc của Kiến trúc Kappa

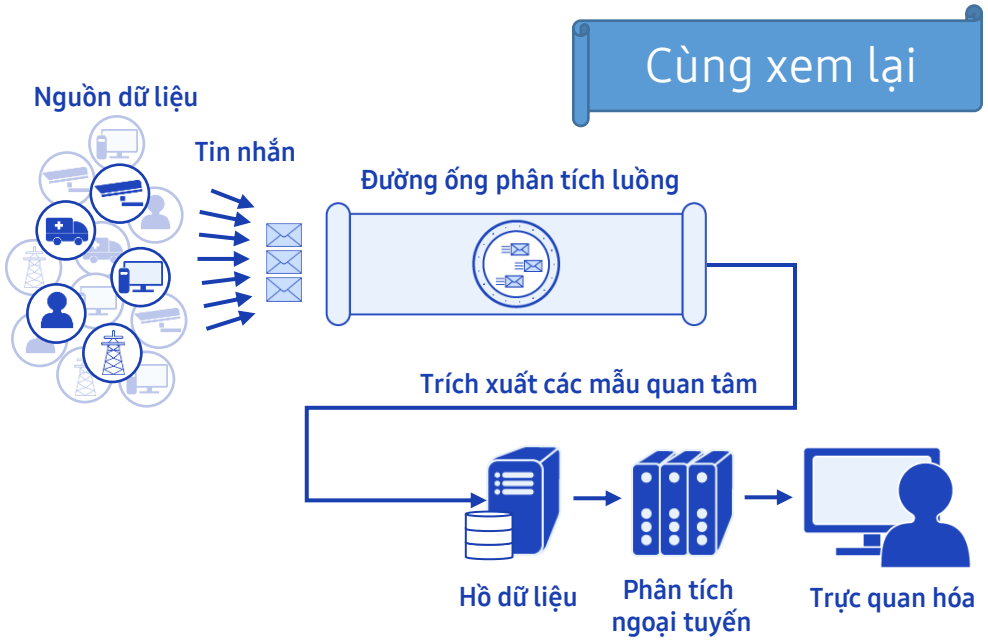
- Ý tưởng về Kiến trúc Kappa ban đầu được đề xuất bởi Jay Kreps khi làm việc tại LinkedIn
  - Jay Kreps là đồng tác giả của Apache Kafka
- Kiến trúc Kappa cố gắng giải quyết khó khăn cốt lõi của Kiến trúc Lambda
  - Phát triển và duy trì mã cho hai hệ thống phân tán phức tạp
- Bị ảnh hưởng nặng nề bởi dự án Kafka được phát triển tại LinkedIn
  - Kafka được những người tạo ra nó mô tả là "nhật ký cam kết phân tán"
  - Được thiết kế để lưu giữ hồ sơ giao dịch lâu bền
  - Tất cả các bản ghi là bất biến và có dấu thời gian liên quan

Một cách tiếp cận mới với cơ sở dữ liệu

- Cơ sở dữ liệu là toàn cầu, được chia sẻ và có trạng thái có thể thay đổi
  - Cập nhật trạng thái có nghĩa là chúng ta đã mất thông tin mãi mãi
- Một mô hình mới là coi cơ sở dữ liệu là một tập hợp các sự kiện bất biến ngày càng tăng
  - Sự thật ngày càng tăng đang truyền dữ liệu
  - Những sự thật bất biến được chèn vào như vậy được đóng dấu thời gian
  - Chúng có thể được truy vấn liên quan đến các thời điểm nhất định dưới dạng dữ kiện
- Nhật ký cam kết phân tán của các giao dịch như Kafka có thể đóng vai trò là cơ sở để lưu trữ bộ sưu tập các sự kiện bất biến ngày càng tăng
  - Mỗi nhật ký cam kết của một giao dịch đại diện cho một giao dịch cơ sở dữ liệu
- Xử lý các luồng sự kiện, trong thời gian thực, khi chúng đến

Truyền dữ liệu theo thời gian thực - Nguyên tắc cơ bản

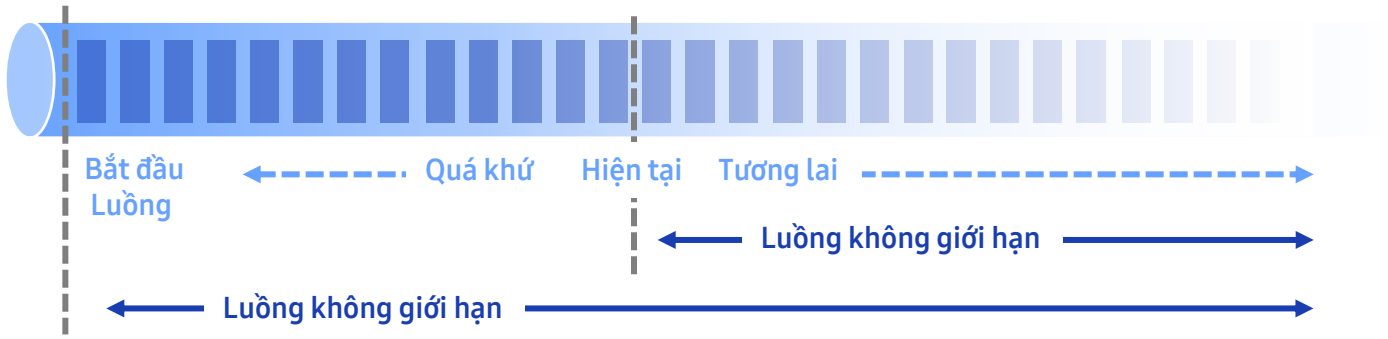
- Thu thập và nhập dữ liệu từ các nguồn dữ liệu tạo dữ liệu theo luồng liên tục
  - Ví dụ: Dữ liệu cảm biến, điện thoại thông minh, nhật ký trang web, thiết bị thông minh, ô tô tự lái, v.v.
  - Xử lý các đợt dữ liệu mà không làm mất thông tin
- Lưu trữ và phân loại dữ liệu để sử dụng lâu dài
- Xử lý dữ liệu theo thời gian thực
  - Trích xuất, chuyển đổi và làm sạch dữ liệu
  - Truy vấn dữ liệu trong thời gian thực
  - Sản xuất báo cáo và trực quan hóa



Luồng dữ liệu không giới hạn

- Luồng dữ liệu không giới hạn có một số điểm bắt đầu
  - Có thể rất xa trong quá khứ và không có bất kỳ hậu quả nào trong việc xử lý dữ liệu đến hiện tại
- Luồng dữ liệu đến không có kết thúc xác định
  - Luồng dữ liệu đến dự kiến sẽ không kết thúc bất cứ lúc nào
- Luồng không giới hạn phải được nhập theo thời gian thực và theo thứ tự cụ thể
  - Không thể đợi cho đến khi toàn bộ luồng được đọc - có thể không bao giờ kết thúc
  - Các sự kiện phải được nhập và xử lý tại thời điểm xảy ra sự kiện

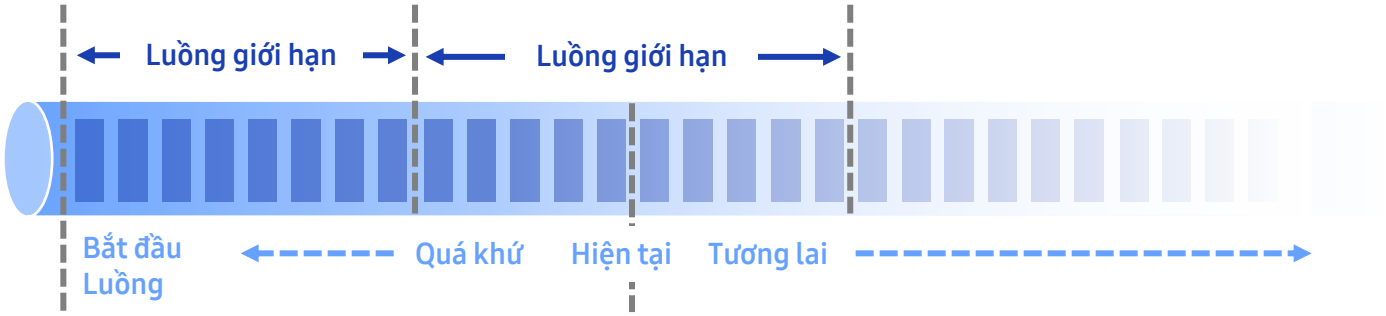
Cùng xem lại



Luồng dữ liệu giới hạn

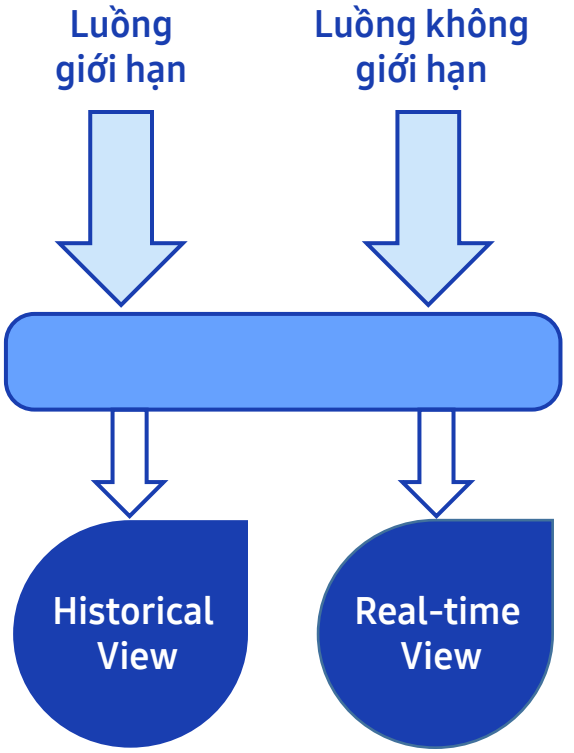
- Luồng dữ liệu bị chặn có điểm bắt đầu và điểm kết thúc
- Có thể nhập và lưu trữ toàn bộ tập dữ liệu trước khi xử lý
- Nhập và xử lý các sự kiện theo thứ tự không nghiêm ngặt
  - Dữ liệu có thể được sắp xếp lại để phù hợp với truy vấn mong muốn vì nó đã được lưu

Cùng xem lại



Truyền dữ liệu trên Kiến trúc Lambda

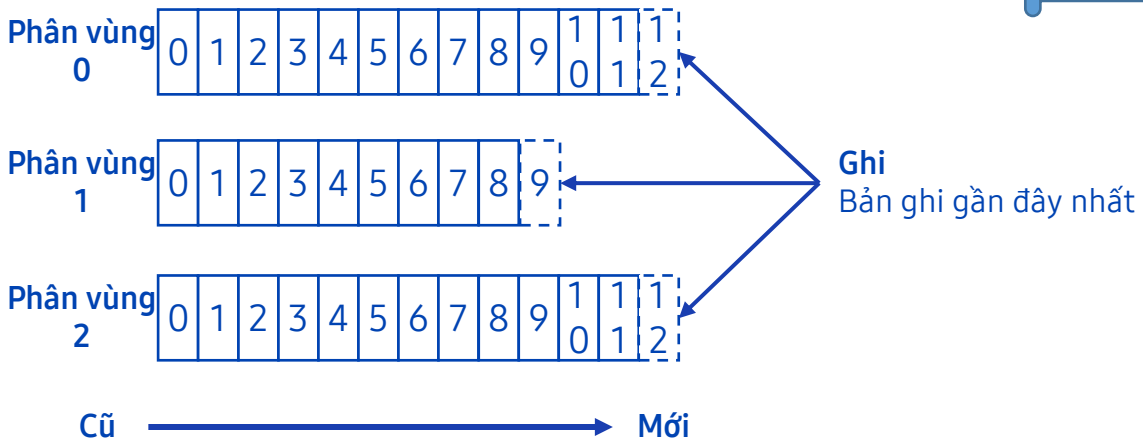
- Kiến trúc Lambda tiếp cận các luồng dữ liệu có giới hạn và không giới hạn theo cùng một cách
  - Cả hai đều là các luồng dữ kiện bất biến có thể được xử lý
- Các luồng dữ liệu có giới hạn có điểm bắt đầu sẽ chứa các sự kiện lịch sử bất biến
- Luồng dữ liệu không giới hạn sẽ chứa các sự kiện bất biến hiện tại
- Xử lý cả hai luồng dữ liệu trên cùng một công cụ xử lý với cùng một mã
  - Cung cấp cả chế độ xem hàng loạt và chế độ xem tốc độ trên cùng một động cơ



Chủ đề Apache Kafka dưới dạng Nhật ký cam kết

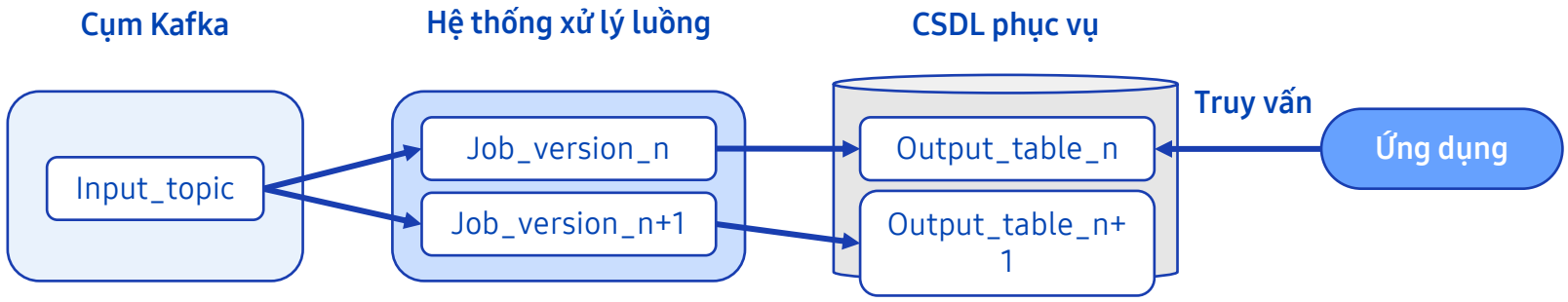
- Bộ lưu trữ vĩnh viễn
  - Các tin nhắn được lưu trữ theo thứ tự chúng được gửi và không thay đổi
  - Thời gian lưu giữ có thể được đặt cho từng chủ đề
  - Đọc tin nhắn từ đầu hoặc mới nhất

Cùng xem lại



Kappa trên Kafka - Từng Bước

- Các bước sau đây minh họa phương pháp cho một ứng dụng sử dụng một công cụ xử lý nhật ký duy nhất để tạo các chế độ xem hàng loạt và tốc độ cùng một lúc
  - Đặt thời gian lưu giữ của Kafka thành nhật ký dữ liệu đầy đủ mong muốn sẽ được xử lý lại
  - Để bắt đầu quá trình xử lý lại, hãy bắt đầu phiên bản thứ hai của công việc xử lý luồng của bạn và đặt nó để đọc từ đầu dữ liệu được giữ lại và chuyển đầu ra của nó sang một bảng đầu ra mới
  - Khi phiên bản thứ hai này được cập nhật và đọc các giao dịch mới nhất, hãy chuyển ứng dụng sang đọc từ bảng đầu ra mới
  - Dừng phiên bản cũ của công việc xử lý luồng của bạn và xóa bảng đầu ra cũ



[Lab10]
Tạo chế độ xem hàng loạt

[Lab11]
Tạo chế độ xem tốc độ

SAMSUNG