**1. DATA SOURCE (Hệ thống nguồn dữ liệu)**

**1.1 Different Types of Source Systems (Các loại hệ thống nguồn dữ liệu)**

**Định nghĩa:**

Source systems là nơi dữ liệu được tạo ra hoặc lưu trữ ban đầu, trước khi được trích xuất để đưa vào hệ thống phân tích (Data Warehouse, Data Lake, v.v.)

**Các loại chính bao gồm:**

* **Relational Databases** (Cơ sở dữ liệu quan hệ)
* **NoSQL Databases**
* **Object Storage** (Lưu trữ đối tượng)
* **Logs** (Nhật ký hệ thống)
* **Streaming Systems** (Dữ liệu thời gian thực)

## **1.2 Relational Databases (RDBMS)**

### **Định nghĩa:**

Là hệ thống lưu trữ dữ liệu có cấu trúc, dữ liệu được tổ chức trong các bảng (tables), bao gồm các hàng (rows) và cột (columns). Các bảng có thể liên kết với nhau thông qua **primary key** và **foreign key**.

### **Ví dụ cụ thể:**

**Bảng Customers:**

A screenshot of a phone

Description automatically generated

**Bảng** Orders:

A close-up of a computer screen

Description automatically generated

→ **CustomerID** trong Orders là **foreign key** tham chiếu đến **primary key** trong Customers.

### **Tính chất: ACID**

#### A – Atomicity (Tính nguyên tố):

Mỗi transaction (giao dịch) là một đơn vị không thể chia nhỏ. Nếu một phần của transaction thất bại, toàn bộ bị hủy.

📌 *Ví dụ:* Nếu bạn chuyển tiền từ A → B, hệ thống sẽ trừ tiền A và cộng vào B. Nếu lỗi ở bước cộng, thì việc trừ cũng sẽ được rollback.

#### C – Consistency (Tính nhất quán):

Transaction phải đưa database từ trạng thái hợp lệ này sang trạng thái hợp lệ khác, luôn tuân theo ràng buộc dữ liệu.

📌 *Ví dụ:* Nếu một bảng yêu cầu CustomerID là duy nhất, thì bạn không thể thêm hai khách hàng có cùng ID.

#### I – Isolation (Tính độc lập):

Các transaction chạy đồng thời không được ảnh hưởng lẫn nhau.

📌 *Ví dụ:* Hai người cùng cập nhật điểm cho một sinh viên, nhưng mỗi người không thấy được cập nhật của người kia cho đến khi hoàn tất.

#### D – Durability (Tính bền vững):

Khi một transaction được commit, dữ liệu sẽ được lưu trữ vĩnh viễn, kể cả khi mất điện hoặc máy chủ gặp sự cố.

📌 *Ví dụ:* Sau khi bạn mua hàng thành công, dù mất điện, đơn hàng vẫn được lưu trữ trong hệ thống.

**Ra đời: Đầu những năm 1970 (Edgar F. Codd)**

**Phổ biến: Rộng rãi từ những năm 1980 đến nay**

**Ưu điểm:**

* Đảm bảo tính toàn vẹn dữ liệu (ACID)
* Chuẩn hóa cao
* Truy vấn mạnh mẽ bằng **SQL**

**Nhược điểm:**

* Khó scale ngang (horizontally)
* Không phù hợp với big data hoặc semi-structured/unstructured data

**Khi nào dùng:**

* Dữ liệu có cấu trúc rõ ràng, quan hệ phức tạp
* Cần đảm bảo toàn vẹn dữ liệu

**Hệ quản trị phổ biến:**

* MySQL, PostgreSQL, SQL Server, Oracle

**1.3 NoSQL Databases**

**Định nghĩa:**

Cơ sở dữ liệu phi quan hệ, cho phép lưu trữ dữ liệu linh hoạt hơn, không cần schema chặt chẽ, hỗ trợ dữ liệu semi-structured hoặc unstructured.

**Hệ thống lưu trữ phân tán trong noSQL**

Là cách mà noSQL chia nhỏ dữ liệu ra nhiều máy chủ khác nhau để lưu trữ và xử lý song song

Mục tiêu: tăng khả năng mở rộng, chịu lỗi và hiệu suất cao.

**Cách hoạt động của hệ thống phân tán**

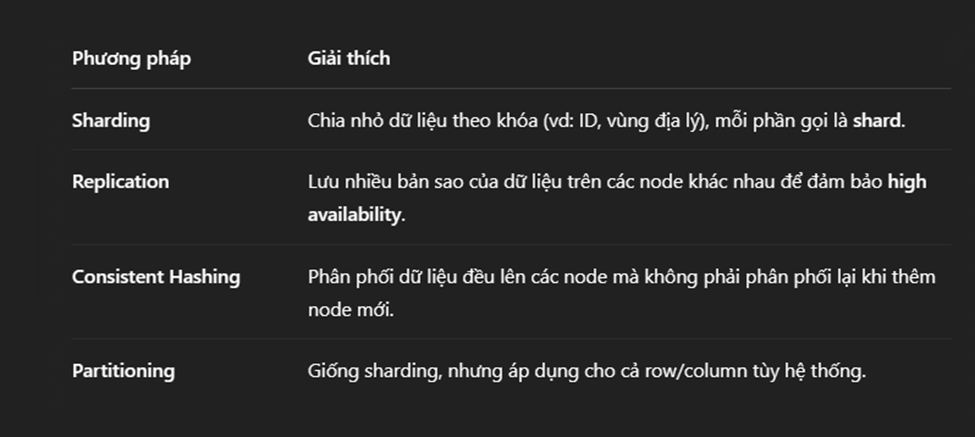
Ví dụ hệ thống Cassandra:

· Có 5 server (node A, B, C, D, E).

· Khi bạn lưu 1 dữ liệu (ví dụ: "user123"), hệ thống sẽ hash key → ra giá trị (vd: hash("user123") = 205) → nó sẽ lưu trên node C.

· Thêm nữa: hệ thống thường lưu bản sao (replica) trên các node khác (để dự phòng khi 1 node hư).

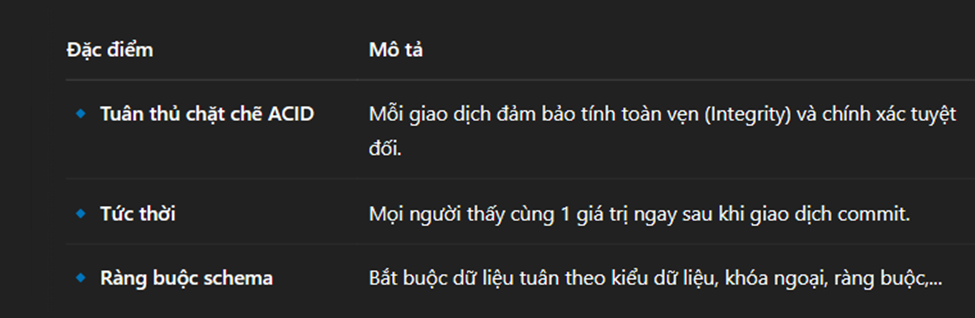
**Methods cho phân tán dữ liệu:**

****

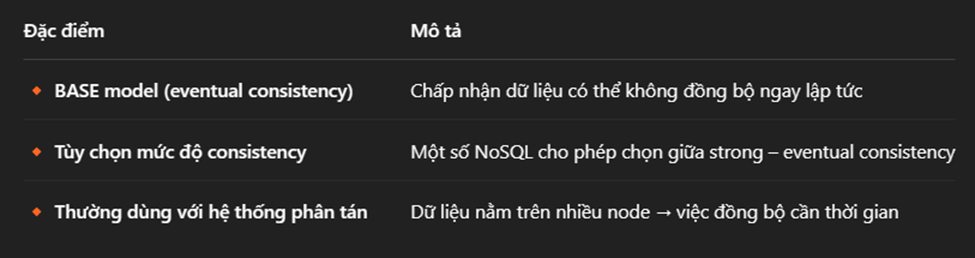
**e) Consistency:**

Khi write dữ liệu thành công, thì mọi node khác phải thấy cùng một giá trị

Trong Relational DB:

****

Trong NoSQL:

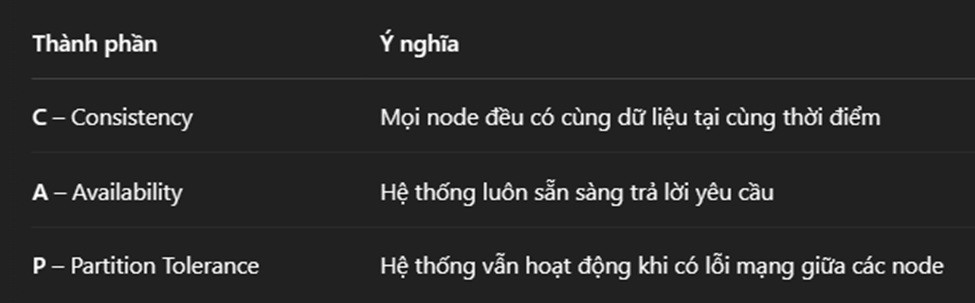
****

**So sánh**

****

**CAP THEOREM**

Một hệ thống phân tán không thể đảm bảo cả 3 yếu tố sau cùng lúc:

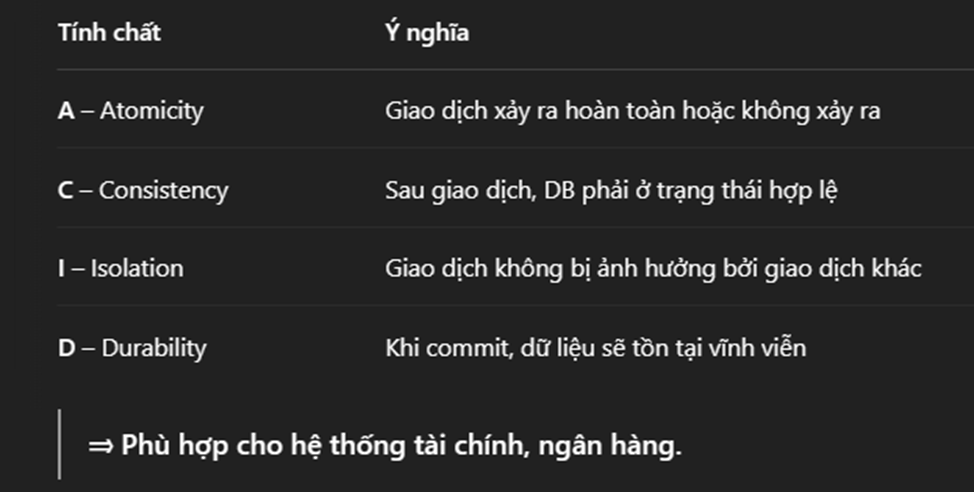


Ví dụ:

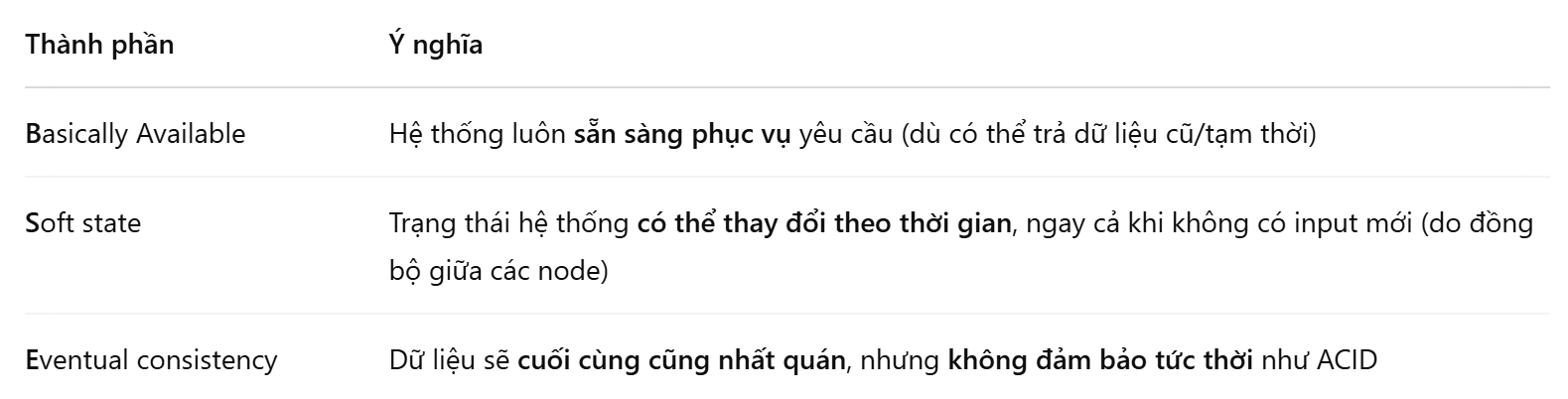


Dưa vào CAP THEOREM, ta có được 2 tính chất có trong Relational DB , noSQL

\* ACID( Trong Relational DB)



**Tính chất BASE:**

****

Types of NoSQL:

A screenshot of a computer

Description automatically generated

Ví dụ (MongoDB - Document):

A close-up of a computer screen

Description automatically generated

**Ra đời: 1998 bởi Carlo Strozzi, cùng thời với Big Data**

**Phổ biến: Tăng mạnh từ 2010s**

**Ưu điểm:**

* Linh hoạt schema
* Dễ scale ngang
* Hiệu quả với dữ liệu lớn và phân tán

**Nhược điểm:**

* Không đảm bảo ACID chặt chẽ
* Cần kỹ năng xử lý phức tạp hơn

**Dùng khi:**

* Dữ liệu bán cấu trúc hoặc phi cấu trúc
* Hệ thống phân tán lớn

**1.4 Object Storage**

**Định nghĩa:**

Lưu trữ dữ liệu dưới dạng “đối tượng” chứa nội dung, metadata và ID duy nhất.

**Ví dụ: Amazon S3, Google Cloud Storage, Azure Blob**

📁 File: sales-2023.json  
🧾 Metadata: size=2MB, created=2023-04-01  
🆔 Object ID: auto-generated hash

**Ra đời: Cuối 2000s**

**Ưu điểm:**

* Tối ưu lưu trữ dữ liệu lớn, không có cấu trúc
* Scale gần như vô hạn

**Nhược điểm:**

* Không hỗ trợ truy vấn như RDBMS
* Độ trễ cao hơn

**Dùng khi:**

* Lưu trữ logs, ảnh, video, file lớn

## **1.5 Logs**

### **Định nghĩa:**

Dữ liệu sinh ra từ hệ thống, thường để ghi lại hoạt động hoặc lỗi.

### **Ví dụ:**



**Ưu điểm:**

* Ghi nhận lịch sử hành vi, sự kiện
* Dễ tích hợp vào các hệ thống giám sát

**Nhược điểm:**

* Dữ liệu thô, cần xử lý thêm
* Dễ bị tràn nếu không quản lý

**Dùng khi:**

* Phân tích hành vi người dùng, phát hiện lỗi hệ thống

**1.6 Streaming Systems**

**Định nghĩa:**

Nguồn dữ liệu phát sinh liên tục theo thời gian thực.

**Ví dụ: Kafka, Pulsar, Kinesis, MQTT**

📍 Dữ liệu click, cảm biến IoT, giao dịch tài chính realtime

**Ưu điểm:**

* Phân tích thời gian thực
* Phản ứng nhanh với sự kiện

**Nhược điểm:**

* Khó kiểm soát độ trễ
* Hạ tầng phức tạp

**Dùng khi:**

* Hệ thống cần xử lý ngay lập tức: Fintech, e-commerce, securit

**2. Data ingestion:**

### 1. **Xử Lý Dữ Liệu Theo Lô (ETL và ELT)**

**Định nghĩa**: Xử lý dữ liệu theo lô (batch processing) đề cập đến việc xử lý dữ liệu theo các nhóm hoặc lô dữ liệu trong các khoảng thời gian đã lên lịch. Quá trình này thu thập dữ liệu trong một khoảng thời gian và xử lý tất cả dữ liệu đó một lần. ETL (Extract, Transform, Load) và ELT (Extract, Load, Transform) là hai phương thức xử lý theo lô phổ biến dùng để tích hợp và chuẩn bị dữ liệu cho phân tích.

* **ETL (Extract, Transform, Load)**: Quá trình ETL trích xuất dữ liệu từ các nguồn khác nhau, chuyển đổi nó thành định dạng mong muốn và sau đó tải vào hệ thống đích (ví dụ: kho dữ liệu).
* **ELT (Extract, Load, Transform)**: Ngược lại, trong ELT, dữ liệu được trích xuất, tải vào hệ thống đích và sau đó thực hiện các phép biến đổi ngay trong hệ thống đó.

**Ví dụ**:

* **ETL**: Một công ty bán lẻ trích xuất dữ liệu bán hàng từ cơ sở dữ liệu giao dịch, chuyển đổi nó thành định dạng phù hợp để báo cáo và tải vào kho dữ liệu để phân tích.
* **ELT**: Một nền tảng mạng xã hội trích xuất dữ liệu người dùng thô và tải vào cơ sở dữ liệu đám mây. Các phép biến đổi (ví dụ: làm sạch, tổng hợp) sau đó được thực hiện trong cơ sở dữ liệu.

**Ưu điểm**:

* **ETL**: Thích hợp cho các dữ liệu yêu cầu biến đổi phức tạp trước khi tải vào hệ thống đích, mang lại sự kiểm soát tốt hơn trong quá trình biến đổi dữ liệu.
* **ELT**: Thích hợp cho các bộ dữ liệu lớn khi hệ thống đích có đủ tài nguyên xử lý các phép biến đổi trong quá trình hoặc sau khi tải dữ liệu.

**Nhược điểm**:

* **ETL**: Có thể chậm và tiêu tốn tài nguyên do quá trình biến đổi dữ liệu diễn ra trước khi tải vào hệ thống.
* **ELT**: Không thích hợp khi dữ liệu có vấn đề về chất lượng cần được xử lý sớm, vì quá trình biến đổi xảy ra sau khi tải vào hệ thống.

**Khi nào nên sử dụng**:

* **ETL**: Khi cần biến đổi dữ liệu phức tạp trước khi tải vào hệ thống đích.
* **ELT**: Khi xử lý các bộ dữ liệu lớn và hệ thống đích có đủ tài nguyên để xử lý biến đổi trong quá trình tải dữ liệu.

**Xu hướng phát triển**: Khi các kho dữ liệu đám mây (như Google BigQuery, AWS Redshift, Snowflake) trở nên phổ biến, ELT đã trở nên phổ biến hơn vì các hệ thống này có thể xử lý các phép biến đổi quy mô lớn với độ trễ thấp. Tuy nhiên, ETL vẫn có giá trị trong việc xử lý dữ liệu phức tạp.

### 2. **Hệ Thống Streaming (Message Queue, Event Streaming Platform Platform)**

**Định nghĩa**: Hệ thống streaming được thiết kế để xử lý dữ liệu liên tục trong thời gian thực, trái ngược với xử lý theo lô. Message Queue và Event Streaming Platform platform là hai thành phần quan trọng của hệ thống streaming.

* **Queue message**: Là cơ chế giao tiếp để truyền tải các message giữa các thành phần khác nhau trong hệ thống, đảm bảo message được xử lý bất đồng bộ và theo thứ tự.
* **Event Streaming Platform platform**: Các nền tảng như Apache Kafka cho phép các hệ thống xuất bản, lưu trữ và tiêu thụ các dòng sự kiện trong thời gian thực, thường sử dụng mô hình xuất bản-đăng ký.

**Ví dụ**:

* **Queue message**: Một hệ thống dịch vụ tài chính sử dụng Message Queue(ví dụ: RabbitMQ hoặc Amazon SQS) để xử lý các giao dịch bất đồng bộ.
* **Event Streaming Platform**: Một ứng dụng chia sẻ xe có thể sử dụng Event Streaming Platform platform (ví dụ: Apache Kafka) để theo dõi vị trí tài xế theo thời gian thực và gửi thông báo cho hành khách.

**Ưu điểm**:

* **Queue message**: Đảm bảo message được truyền tải đáng tin cậy, tách biệt giữa các hệ thống sản xuất và tiêu thụ, và dễ dàng mở rộng các hệ thống bất đồng bộ.
* **Event Streaming Platform**: Cho phép xử lý sự kiện theo thời gian thực, cung cấp thông tin tức thì, giám sát và các hành động dựa trên sự kiện (ví dụ: phân tích thời gian thực, phát hiện bất thường).

**Nhược điểm**:

* **Queue message**: Có thể gây độ trễ nếu hệ thống bị quá tải. Cần quản lý message cẩn thận để tránh tắc nghẽn.
* **Event Streaming Platform**: Yêu cầu cơ sở hạ tầng phức tạp để quản lý xử lý dòng dữ liệu và lưu trữ sự kiện ở quy mô lớn.

**Khi nào nên sử dụng**:

* **Queue message**: Khi cần giao tiếp bất đồng bộ giữa các hệ thống và xử lý các tác vụ nền (ví dụ: xử lý công việc nền, các hàng đợi tác vụ).
* **Event Streaming Platform**: Khi cần xử lý dữ liệu theo thời gian thực, kiến trúc dựa trên sự kiện hoặc giám sát và phân tích cần được cập nhật liên tục khi dữ liệu liên tục vào.

**Công cụ sử dụng**:

* **Message Queue**: RabbitMQ, Apache ActiveMQ, Amazon SQS.
* **Event Streaming Platform**: Apache Kafka, Apache Pulsar, AWS Kinesis.

**Xu hướng phát triển**: Các kiến trúc dựa trên sự kiện và xử lý dữ liệu theo thời gian thực đang trở nên phổ biến, đặc biệt trong các ngành như thương mại điện tử, tài chính và Internet of Things (IoT). Kafka và các nền tảng tương tự ngày càng được tích hợp vào các kiến trúc microservices để cho phép các hệ thống thời gian thực có khả năng chịu tải cao, có độ tin cậy cao.