**Data Engineer with AWS**

1. **Data modelling**

* What is data modelling?

Data modelling (mô hình hóa dữ liệu) là quá trình tạo ra một bản vẽ hay biểu đồ giúp ta hiểu các thông tin trong dữ liệu và mối quan hệ giữa chúng một cách dễ dàng hơn. Ví dụ, khi ta muốn tạo ra một trang web bán hàng, ta cần một mô hình hóa dữ liệu để biết được thông tin nào cần phải lưu trữ (ví dụ như thông tin sản phẩm, giá cả, thông tin khách hàng) và quan hệ giữa các thông tin đó như thế nào (ví dụ như mỗi khách hàng có thể mua nhiều sản phẩm, mỗi sản phẩm có thể có nhiều màu sắc khác nhau). Mô hình hóa dữ liệu giúp cho việc quản lý dữ liệu trên trang web được dễ dàng và hiệu quả hơn.

* What is the Fault-Tolerance?

Fault-tolerance là khả năng của một hệ thống để tiếp tục hoạt động đúng đắn và cung cấp dịch vụ mà không bị ảnh hưởng quá nhiều bởi các lỗi hoặc sự cố xảy ra. Trong một hệ thống fault-tolerant, các thành phần có thể tự động phát hiện, chẩn đoán và khắc phục lỗi để duy trì tính khả dụng và tính đáng tin cậy của hệ thống. Hệ thống fault-tolerant thường có nhiều thành phần phụ trợ, bao gồm các bộ đệm, bộ nhớ đệm, các bản sao dự phòng của dữ liệu và các thành phần khác để đảm bảo rằng các sự cố không làm gián đoạn hoạt động của hệ thống. Hệ thống fault-tolerant được sử dụng trong nhiều lĩnh vực, bao gồm các hệ thống máy tính, mạng, thiết bị điện tử và các ứng dụng phần mềm quan trọng.

* What is ACID transaction?

An ACID transaction is a type of transaction in computer programming that ensures reliability and consistency when making changes to a database. ACID stands for Atomicity, Consistency, Isolation, and Durability.

* Atomicity means that a transaction is treated as a single unit of work, which either succeeds completely or fails completely. In other words, if something goes wrong during the transaction, all changes made during that transaction are rolled back, ensuring that the database is left in a consistent state.
* Consistency means that the database is always in a valid state before and after a transaction. This ensures that any changes made during a transaction won't leave the database in an invalid state.
* Isolation means that each transaction is executed independently of other transactions, so that the changes made by one transaction won't affect the outcome of other transactions.
* Durability means that once a transaction is completed, the changes made during that transaction are permanent and won't be lost, even in the event of power failures, system crashes, or other unforeseen circumstances.
* The benefit of using ACID transactions is that they provide a reliable and consistent way to update databases. By ensuring that transactions are atomic, consistent, isolated, and durable, ACID transactions help prevent data inconsistencies and ensure the reliability of the data in the database.
* When not use a Relational Database?
* **Have large amounts of data:** Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. You are limited by how much you can scale and how much data you can store on one machine. You cannot add more machines like you can in NoSQL databases.
* **Need to be able to store different data type formats:** Relational databases are not designed to handle unstructured data.
* **Need high throughput -- fast reads:** While ACID transactions bring benefits, they also slow down the process of reading and writing data. If you need very fast reads and writes, using a relational database may not suit your needs.
* **Need a flexible schema:** Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
* **Need high availability:** The fact that relational databases are not distributed (and even when they are, they have a coordinator/worker architecture), they have a single point of failure. When that database goes down, a fail-over to a backup system occurs and takes time.
* **Need horizontal scalability:** Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data.
* When to Use a NoSQL Database?
* **Need to be able to store different data type formats:** NoSQL was also created to handle different data configurations: structured, semi-structured, and unstructured data. JSON, XML documents can all be handled easily with NoSQL.
* **Large amounts of data:** Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. NoSQL databases were created to be able to be horizontally scalable. The more servers/systems you add to the database the more data that can be hosted with high availability and low latency (fast reads and writes).
* **Need horizontal scalability:** Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data
* **Need high throughput:** While ACID transactions bring benefits they also slow down the process of reading and writing data. If you need very fast reads and writes using a relational database may not suit your needs.
* **Need a flexible schema:** Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
* **Need high availability:** Relational databases have a single point of failure. When that database goes down, a failover to a backup system must happen and takes time.
* When NOT to use NoSQL Database?
* **When you have a small dataset**: NoSQL databases were made for big datasets not small datasets and while it works it wasn’t created for that.
* **When you need ACID Transactions:** If you need a consistent database with ACID transactions, then most NoSQL databases will not be able to serve this need. NoSQL database are eventually consistent and do not provide ACID transactions. However, there are exceptions to it. Some non-relational databases like MongoDB can support ACID transactions.
* **When you need the ability to do JOINS across tables:** NoSQL does not allow the ability to do JOINS. This is not allowed as this will result in full table scans.
* **If you want to be able to do aggregations and analytics**
* **If you have changing business requirements:** Ad-hoc queries are possible but difficult as the data model was done to fix particular queries
* **If your queries are not available and you need the flexibility:** You need your queries in advance. If those are not available or you will need to be able to have flexibility on how you query your data you might need to stick with a relational database.

**Remember:**

NoSQL databases and Relational Databases do not replace each other for all tasks.

Both do different tasks extremely well, and should be utilized for the use cases they fit best.

* Note for PosgreSQL (Apache Cassandra):
* KEYSPACE is DATABASE
* In Apache Cassandra, queries that involve non-primary key columns are generally discouraged because they can be slow and resource-intensive.

To run the query anyway, you can add the ALLOW FILTERING directive to the end of the query. This will enable Cassandra to execute the query, but it may result in slower performance and higher resource usage.

* Difference between OLAP and OLTP?
* **Online Analytical Processing (OLAP):**

Databases optimized for these workloads allow for complex analytical and ad hoc queries, including aggregations. These types of databases are optimized for reads.

* **Online Transactional Processing (OLTP):**

Databases optimized for these workloads allow for less complex queries in large volume. The types of queries for these databases are read, insert, update, and delete.

* The key to remember the difference between OLAP and OLTP is analytics (A) vs transactions (T). If you want to get the price of a shoe then you are using OLTP (this has very little or no aggregations). If you want to know the total stock of shoes a particular store sold, then this requires using OLAP (since this will require aggregations).
* Normal Form (NF)

Normalization là quá trình tổ chức dữ liệu trong cơ sở dữ liệu sao cho giảm thiểu sự trùng lặp và phụ thuộc. Có nhiều cấp độ normalization khác nhau, mỗi cấp độ có riêng các quy tắc và tiêu chí. Các cấp độ normalization thường được sử dụng là 1NF, 2NF và 3NF.

* First Normal Form (1NF): Đây là cấp độ normalization cơ bản nhất. Nó yêu cầu tất cả dữ liệu trong bảng phải là atomic, có nghĩa là mỗi ô chỉ có thể chứa một giá trị duy nhất. Nó cũng yêu cầu rằng mỗi cột trong bảng phải có một tên duy nhất và thứ tự các hàng và cột trong bảng không quan trọng.
* Second Normal Form (2NF): Để đạt được 2NF, một bảng phải trước tiên phải ở trong 1NF. Nó cũng yêu cầu rằng mọi cột không phải là khóa trong bảng phải phụ thuộc vào khóa chính theo cách hàm số. Nó có nghĩa là một cột không phải là khóa phải được xác định duy nhất bởi khóa chính và không phải bởi bất kỳ cột không phải là khóa nào khác.
* Third Normal Form (3NF): Để đạt được 3NF, một bảng phải trước tiên phải ở trong 2NF. Nó cũng yêu cầu rằng mọi cột không phải là khóa trong bảng phải độc lập với tất cả các cột không phải là khóa khác. Điều này có nghĩa là không được có sự phụ thuộc chéo, trong đó một cột không phải là khóa phụ thuộc vào một cột không phải là khóa khác trong cùng bảng.

Sự khác biệt chính giữa ba cấp độ normalization này là các tiêu chí phải đáp ứng để một bảng được coi là đã được normalized. 1NF đảm bảo rằng mỗi ô trong bảng chứa một giá trị duy nhất, 2NF đảm bảo rằng các cột không phải là khóa được xác định duy nhất bởi khóa chính và 3NF đảm bảo rằng các cột không phải là khóa độc lập với nhau. Khi tiến đến các cấp độ normalization cao hơn, cơ sở dữ liệu kết quả trở nên hiệu quả hơn và dễ bảo trì hơn.

* Denormalization?

Denormalization trong cơ sở dữ liệu là quá trình thêm dữ liệu trùng lặp có chủ ý vào một hoặc nhiều bảng để cải thiện hiệu suất khi truy xuất dữ liệu. Điều này thường được thực hiện trong các kho dữ liệu hoặc hệ thống yêu cầu hiệu suất cao cho các hoạt động đọc dữ liệu. Bằng cách sao chép dữ liệu trên các bảng, nó loại bỏ nhu cầu thực hiện các liên kết phức tạp và cho phép các truy vấn nhanh hơn. Tuy nhiên, denormalization cũng có thể dẫn đến các không nhất quán dữ liệu nếu các dữ liệu trùng lặp không được quản lý đúng cách, và có thể đòi hỏi nhiều công sức hơn để duy trì và cập nhật cơ sở dữ liệu.

* Compare Denormalization and Normalization
* **Normalization** là việc cố gắng tăng tính toàn vẹn của dữ liệu bằng cách giảm số lượng bản sao của dữ liệu. Dữ liệu cần được thêm hoặc cập nhật sẽ được thực hiện ở ít nơi nhất có thể.
* **Denormalization** là việc cố gắng tăng hiệu suất bằng cách giảm số lượng liên kết giữa các bảng (vì các liên kết có thể chậm). Tính toàn vẹn dữ liệu sẽ bị ảnh hưởng một chút, vì sẽ có nhiều bản sao của dữ liệu (để giảm số lượng LIÊN KẾT).
* Fact Table and Dimension Table?
* **Fact table** là một bảng trung tâm chứa dữ liệu liên quan đến các quy trình kinh doanh, sự kiện hoặc giao dịch. Thông thường, nó chứa dữ liệu số học, cộng dồn như doanh thu bán hàng hoặc số lượng sản phẩm được bán và được kết nối với một hoặc nhiều bảng chiều (Dimension table) thông qua các khóa ngoại.
* **Dimension table** chứa dữ liệu mô tả cung cấp ngữ cảnh cho dữ liệu trong bảng sự kiện. Thông thường, chúng có một hoặc nhiều cột làm khóa ngoại trong bảng sự kiện. Bảng chiều có thể cung cấp thông tin về thời gian, địa điểm, sản phẩm, khách hàng hoặc bất kỳ chiều nào khác liên quan đến kinh doanh.

Sự khác biệt chính giữa bảng sự kiện và bảng chiều là bảng sự kiện chứa dữ liệu định lượng có thể phân tích, trong khi bảng chiều chứa dữ liệu mô tả cung cấp ngữ cảnh cho phân tích. Bảng sự kiện thường lớn, có nhiều hàng và ít cột, trong khi bảng chiều có xu hướng có ít hàng hơn nhưng nhiều cột hơn. Bảng sự kiện cũng thường được cập nhật thường xuyên hơn so với bảng chiều, mà có xu hướng tương đối ổn định.

Tóm lại, bảng sự kiện chứa dữ liệu định lượng liên quan đến các sự kiện kinh doanh và được kết nối với bảng chiều, chứa dữ liệu mô tả cung cấp ngữ cảnh. Bảng sự kiện có xu hướng lớn và được cập nhật thường xuyên, trong khi bảng chiều có xu hướng nhỏ hơn và ổn định hơn.

* Pros and Con of Star Schema:

|  |  |
| --- | --- |
| **Benefits** | **Drawbacks** |
| Đơn giản: Mô hình Star Schema dễ hiểu và sử dụng, ngay cả đối với người dùng không chuyên. Nó cung cấp một cách đơn giản, trực quan để biểu diễn dữ liệu phức tạp. | Trùng lặp: Mô hình Star Schema có thể bị trùng lặp vì nó lưu trữ cùng một dữ liệu ở nhiều nơi khác nhau. Điều này có thể dẫn đến yêu cầu lưu trữ tăng và có thể truy xuất dữ liệu chậm hơn. |
| Hiệu suất truy vấn: Mô hình Star Schema cung cấp hiệu suất truy vấn tuyệt vời nhờ cấu trúc không chuẩn hóa của nó. Nó loại bỏ nhu cầu cho các phép kết nối phức tạp, có thể chậm và tốn tài nguyên. | Chất lượng dữ liệu: Mô hình Star Schema có thể ảnh hưởng đến chất lượng dữ liệu khi nó lưu trữ dữ liệu không chuẩn hóa. Điều này có thể dẫn đến không nhất quán và lỗi dữ liệu. |
| Khả năng mở rộng: Mô hình Star Schema có tính mở rộng cao và có thể xử lý lượng dữ liệu lớn. Nó cho phép dễ dàng phân vùng và phân phối dữ liệu trên nhiều máy chủ. | Giới hạn các trường hợp sử dụng: Mô hình Star Schema không phù hợp cho tất cả các mô hình dữ liệu và có thể không phải là lựa chọn tốt nhất cho cấu trúc dữ liệu phân cấp phức tạp. |
| Tính linh hoạt: Mô hình Star Schema cho phép dễ dàng thêm các chiều và đơn vị đo mới mà không ảnh hưởng đến cấu trúc dữ liệu hiện có. Điều này làm cho nó trở thành một giải pháp linh hoạt cho các yêu cầu kinh doanh thay đổi. | Bảo trì: Mô hình Star Schema đòi hỏi bảo trì định kỳ để đảm bảo rằng dữ liệu chính xác và cập nhật. Điều này có thể tốn thời gian và tài nguyên. |

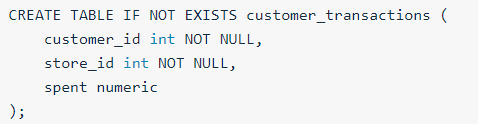
Tóm lại, mặc dù mô hình Star Schema có nhiều lợi ích như tính đơn giản, hiệu suất truy vấn tốt, khả năng mở rộng và linh hoạt, nhưng nó cũng có một số nhược điểm như sự trùng lặp, ảnh hưởng đến tính toàn vẹn dữ liệu, giới hạn các trường hợp sử dụng và yêu cầu bảo trì. Quan trọng là phải xem xét những yếu tố này khi chọn phương pháp mô hình hóa dữ liệu phù hợp với nhu cầu kinh doanh.

* Pros and Con of Snowflake Schema:

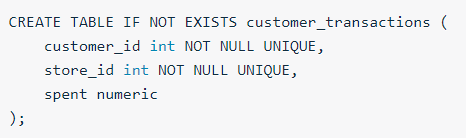
|  |  |
| --- | --- |
| **Benefits** | **Drawbacks** |
| Giảm sự trùng lặp dữ liệu: Vì các bảng chiều được chuẩn hóa, có ít sự trùng lặp dữ liệu hơn so với mô hình Star Schema, điều này có thể tiết kiệm không gian lưu trữ và cải thiện hiệu suất truy vấn. | Phức tạp hóa truy vấn: Cấu trúc chuẩn hóa của schema có thể làm cho các truy vấn phức tạp hơn, vì chúng yêu cầu nhiều liên kết giữa nhiều bảng. |
| Cải thiện chất lượng dữ liệu: Chuẩn hóa có thể giúp loại bỏ không nhất quán và bất thường trong dữ liệu, dẫn đến cải thiện chất lượng và độ chính xác của dữ liệu. | Hiệu suất truy vấn thấp hơn: Do số lượng liên kết tăng lên, hiệu suất truy vấn có thể chậm hơn so với mô hình Star Schema, đặc biệt là với các tập dữ liệu lớn. |
| Độ linh hoạt cao hơn: Mô hình Snowflake Schema cho phép thêm hoặc sửa đổi các chiều và thuộc tính một cách linh hoạt hơn mà không ảnh hưởng đến các phần khác của schema. | Yêu cầu lưu trữ cao hơn: Chuẩn hóa dẫn đến một số lượng lớn các bảng nhỏ hơn, điều này có thể dẫn đến yêu cầu lưu trữ cao hơn so với một schema không được chuẩn hóa. |
| Hỗ trợ tốt hơn cho các cấu trúc phân cấp phức tạp: Mô hình Snowflake Schema phù hợp hơn cho việc mô hình hóa các mối quan hệ phân cấp phức tạp giữa các chiều. | Quản lý dữ liệu phức tạp hơn: Việc duy trì một Snowflake Schema có thể phức tạp hơn so với mô hình Star Schema, vì nó yêu cầu quản lý một số lượng lớn các bảng và các mối quan hệ giữa chúng. |

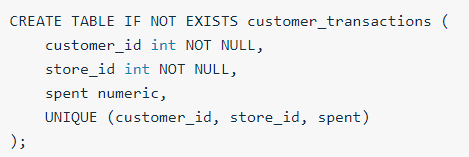
Tổng thể, mô hình Snowflake có thể có lợi trong việc xử lý các hệ thống dữ liệu phức tạp và cải thiện chất lượng dữ liệu, nhưng nó có thể đến với các trade-offs về hiệu suất truy vấn, yêu cầu lưu trữ và phức tạp trong quản lý dữ liệu.

* PostgreSQL Data Definition and Constraints
* NOT NULL

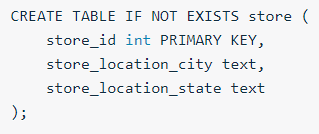


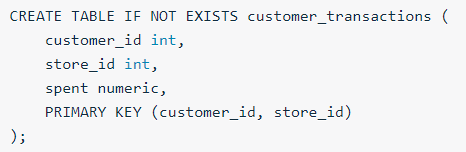
* UNIQUE



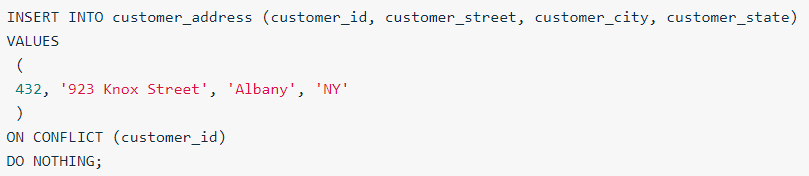


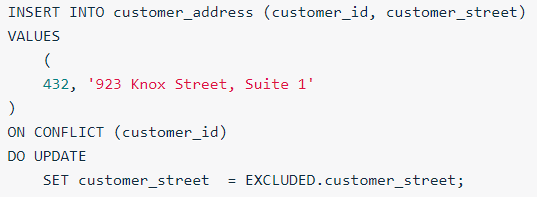
* PRIMARY KEY





* UPSERT: *ON CONFLICT = WHEN MATCHED* in databricks





* Apache Cassandra?

Benefits:

* Scalability: Kiến trúc phân tán của Cassandra cho phép nó mở rộng theo chiều ngang và xử lý lượng dữ liệu lớn.
* High availability and fault tolerance: Sao chép và phân phối dữ liệu của Cassandra đảm bảo tính sẵn có cao và độ bền cao.
* Flexible data model: Mô hình dữ liệu linh hoạt của Cassandra cho phép tạo động các cột mới, làm cho nó phù hợp cho các trường hợp sử dụng yêu cầu các thay đổi cấu trúc nhanh chóng.
* High write throughput: Đường ghi tối ưu hóa của Cassandra cho phép nó xử lý lưu lượng ghi lớn.

Drawbacks:

* Complexity: Kiến trúc phân tán và tùy chọn cấu hình của Cassandra có thể làm cho nó phức tạp để thiết lập và bảo trì.
* Query complexity: Mô hình dữ liệu của Cassandra được tối ưu hóa cho các khối lượng công việc ghi và có thể làm cho truy vấn trở nên phức tạp.
* Eventual consistency: Mô hình nhất quán của cuối cùng của Cassandra có thể dẫn đến việc trả về dữ liệu cũ trong một số trường hợp.
* No transaction support: Cassandra không hỗ trợ giao dịch, làm cho việc duy trì tính nhất quán của dữ liệu trở nên khó khăn hơn.

Nhìn chung, Apache Cassandra là một hệ thống cơ sở dữ liệu NoSQL có khả năng mở rộng cao, độ bền cao và rất phù hợp cho các trường hợp sử dụng yêu cầu tính sẵn có cao và khả năng mở rộng. Tuy nhiên, nó có thể phức tạp để thiết lập và bảo trì, và mô hình dữ liệu và mô hình nhất quán của nó có thể không phù hợp cho tất cả các trường hợp sử dụng.

* What is the CAP theorem?

Định lý CAP là một khái niệm trong tính toán phân tán, khẳng định rằng không thể đồng thời cung cấp ba đảm bảo sau đây cho một hệ thống phân tán:

* Consistency: Mọi hoạt động đọc đều nhận được ghi mới nhất hoặc thông báo lỗi.
* Availability: Mọi nút không bị lỗi trong hệ thống trả về phản hồi cho tất cả các yêu cầu đọc và ghi trong khoảng thời gian hợp lý.
* Partition tolerance: Hệ thống tiếp tục hoạt động mặc dù có mất tin nhắn tùy ý hoặc sự cố về một số thành phần.

Nói cách khác, định lý CAP cho rằng trong một hệ thống phân tán, nếu có phân vùng mạng (tức là một lỗi liên lạc giữa các nút), hệ thống phải chọn giữa tính nhất quán và khả dụng. Nó có thể tiếp tục hoạt động chỉ với một tập hợp con của các nút có sẵn và duy trì tính nhất quán hoặc cho phép tất cả các nút đều khả dụng nhưng có thể có dữ liệu không nhất quán.

Định lý CAP thường được sử dụng như là một hướng dẫn để thiết kế các hệ thống phân tán và giúp kỹ sư đưa ra các sự lựa chọn giữa tính nhất quán, khả dụng và khả năng chịu lỗi phân tán dựa trên các yêu cầu cụ thể của trường hợp sử dụng của họ.

* What is the PK in Apache Cassandra?
* **Primary Key** (khóa chính): là một định danh duy nhất cho mỗi hàng trong bảng. Nó bao gồm một hoặc nhiều cột duy nhất xác định một hàng và được tạo thành từ hai phần: khóa phân vùng và cột nhóm.
* **Partition Key** (khóa phân vùng): là phần đầu tiên của khóa chính và xác định nút nào sẽ lưu trữ dữ liệu trong cụm. Nó chịu trách nhiệm phân vùng dữ liệu và đảm bảo rằng các hàng có cùng khóa phân vùng được lưu trữ cùng với nhau trên cùng một nút. Khóa phân vùng được sử dụng để xác định nút sẽ xử lý các yêu cầu đọc và ghi cho hàng đó.
* **Clustering Column** (các cột nhóm): là phần thứ hai của khóa chính và được sử dụng để sắp xếp dữ liệu trong một phân vùng. Chúng xác định thứ tự lưu trữ dữ liệu trong một phân vùng và được sử dụng để truy xuất dữ liệu theo đúng thứ tự khi thực hiện một truy vấn.

Tóm lại, khóa phân vùng xác định nút nào trong cụm sẽ lưu trữ dữ liệu, trong khi các cột nhóm được sử dụng để sắp xếp và truy xuất dữ liệu trong một phân vùng.

1. Cloud Data Warehouse

* D
* Fd
* F
* D
* f
* D
* Sd
* S
* D
* S
* d

1. Sds
2. Ds
3. Ds
4. D
5. Sd
6. S
7. D
8. S
9. d