BÀI BÁO CÁO LÝ THUYẾT
MODULE DATABASE THINKING – ZTF 2023

Người thực hiện: Nguyễn Đại Nghĩa (nghiand)

1. **<u>Tìm hiểu về transaction, ACID trong transaction khi lưu trữ dữ liệu quan hệ (ví dụ MySQL)</u>**

a/ Transaction: là một tiến trình xử lý có xác định điểm đầu và điểm cuối, được chia nhỏ thành các operations, tiến trình được thực thi một cách tuần tự và độc lập các operation đó theo nguyên tắc hoặc **tất cả đều thành công** hoặc **một operation thất bại thì toàn bộ tiến trình thất bại**.

- Nếu tất cả các operation đều thành công tới bước cuối cùng, ta sẽ thực hiện **commit** để xác nhận toàn bộ quá trình đã thành công và lưu dữ liệu tương ứng
- Nếu tại một operation bị thất bại, tất cả sự thay đổi từ operation đó trở về trước sẽ được đảo ngược (**rollback**) về trạng thái trước đó, đảm bảo tính toàn vẹn và đúng đắn của dữ liệu

b/ Transaction có 4 tính chất, được viết tắt thành ACID, bao gồm Atomicity, Consitency, Isolation, và Durability:

- Atomicity (Tính nguyên tử): Đảm bảo rằng một trasaction được thực hiện hoặc không được thực hiện hoàn toàn. Nếu một phần của transaction không thể hoàn thành, tất cả các thay đổi sẽ bị hủy và cơ sở dữ liệu sẽ được đưa về trạng thái ban đầu

- Consistency (Tính nhất quán): Đảm bảo rằng dữ liệu luôn ở trạng thái hợp lệ sau khi một transaction được thực hiện. Điều này đảm bảo rằng các ràng buộc và quy tắc của cơ sở dữ liệu không bị vi phạm

- Isolation (Tính cô lập): Đảm bảo rằng các thay đổi được thực hiện bởi một transaction không ảnh hưởng đến các transaction khác đang diễn ra đồng thời

- Durability (Tính bền vững): Đảm bảo rằng các thay đổi đã được áp dụng trong một transaction sẽ được lưu trữ vĩnh viễn và không bị mất ngay cả khi xảy ra lỗi hệ thống hoặc máy chủ.

2. Tìm hiểu về indexing trong cơ sở dữ liệu quan hệ MySQL

Trong cơ sở dữ liệu quan hệ như MySQL, indexing là quá trình **tạo ra một cấu trúc dữ liệu phụ** để **cải thiện hiệu suất của các truy vấn đối với bảng dữ liệu lớn**. Indexing giúp cơ sở dữ liệu nhanh chóng định vị và truy cập vào các hàng dữ liệu dựa trên các điều kiện truy vấn, thay vì phải quét qua toàn bộ bảng.

- Cách hoạt động: Khi tạo một index trên một cột, MySQL tạo ra một cấu trúc dữ liệu bổ sung (index) dựa trên giá trị của cột đó. Index này cho phép MySQL nhanh chóng định vị các hàng dữ liệu phù hợp với điều kiện truy vấn.
  Indexing có thể thực hiện trên 1 cột của bảng, hoặc kết hợp giữa các cột, phụ thuộc vào các truy vấn thực tế có mức độ sử dụng nhiều.

  Ví dụ: Giả sử bảng user có nhiều cột, trong đó các cột age và cột address thường xuyên được truy vấn, ta có thể có các giải pháp đánh index như sau:
  - Indexing trên cột age:
    `CREATE INDEX idx_age ON user(age);`
  - Indexing trên cột address:
    `CREATE INDEX idx_address ON user(address);`
  - Indexing hỗn hợp trên cả hai cột này:
    `CREATE INDEX idx_age_addr ON user(address);`

- Loại index: MySQL hỗ trợ nhiều loại index, bao gồm index BTREE, HASH, FULLTEXT, và SPATIAL. Index BTREE là loại index phổ biến nhất và được sử dụng mặc định trong hầu hết các trường hợp.

- Ảnh hưởng lên hiệu suất: Indexing có thể **cải thiện hiệu suất của các truy vấn SELECT** bằng cách giảm thời gian tìm kiếm và sắp xếp dữ liệu. Tuy nhiên, việc tạo index cũng có thể ảnh hưởng đến hiệu suất của các truy vấn INSERT, UPDATE và DELETE, do MySQL phải duy trì index khi có thay đổi dữ liệu. Nói một cách khác, indexing thực hiện đánh đổi độ phức tạp của thao tác cập nhật (INSERT, UPDATE, DELETE) để đổi lấy tốc độ query dữ liệu trên các bảng dữ liệu lớn.

3. Tìm hiểu về MySQL Master-Slave Replication

Trong một hệ thống triển khai cơ sở dữ liệu MySQL, kiến trúc master-slave thường được sử dụng để cải thiện hiệu suất, khả năng mở rộng và đảm bảo sự sẵn sàng của hệ thống. Dưới đây là một số điểm cần biết về kiến trúc master-slave:

- Master Server:
    - Master server là nơi chứa bản gốc của dữ liệu và là nơi chịu trách nhiệm cho các thao tác ghi (INSERT, UPDATE, DELETE).
    - Các thay đổi dữ liệu được thực hiện trên master server và sau đó được sao chép đến các slave server.
- Slave Server:
    - Slave server là các bản sao của master server, nơi dữ liệu được sao chép từ master để đảm bảo sự nhất quán.
    - Slave server thường được sử dụng cho các thao tác đọc (SELECT), giảm tải cho master và cải thiện hiệu suất.
    - Dữ liệu trên slave server không được phép chỉnh sửa trực tiếp.
- Cấu hình Replication:
    - Replication: Cấu hình replication trên master server để sao chép dữ liệu đến slave server. Cấu hình này bao gồm:
        - Xác định master server và slave server.
        - Cấu hình các thông số như server ID, giao thức sao chép (ví dụ: binlog), và quyền truy cập.
    - Replication Start: Khởi động quá trình sao chép dữ liệu từ master server đến slave server.
    - Replication Management: Đảm bảo rằng quá trình sao chép diễn ra một cách nhất quán và không bị gián đoạn. Kiểm tra logs và các cảnh báo để phát hiện và xử lý các vấn đề.
- Sử dụng trong deployment:
    - Đối với các ứng dụng có yêu cầu cao về đọc và ghi dữ liệu, master-slave replication giúp tăng tốc độ truy xuất dữ liệu và cải thiện khả năng mở rộng.
    - Đối với các ứng dụng có yêu cầu cao về sự sẵn sàng và sao lưu, slave server có thể được sử dụng để tạo ra bản sao dự phòng của dữ liệu.

Để đảm bảo các tính chất của master-slave replication, cần chú ý:

- Sao chép dữ liệu giữa master và slave là bất đồng bộ, vì vậy có thể có một khoảng thời gian ngắn giữa khi dữ liệu được ghi vào master và khi nó được sao chép đến slave.
- Đảm bảo rằng mạng và hệ thống của bạn đủ mạnh để xử lý lưu lượng sao chép dữ liệu giữa master và slave một cách hiệu quả.

4. Sự khác nhau giữa cơ sở dữ liệu SQL và NoSQL

SQL	NoSQL
MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite	MongoDB, Cassandra, Redis
Tính cấu trúc: SQL sử dụng mô hình cấu trúc cho dữ liệu, trong đó mỗi bảng có các cột được xác định trước và có quan hệ với nhau thông qua khóa ngoại. Phù hợp cho lưu các dữ liệu có tính cấu trúc, mỗi thông tin về **đối tượng cùng loại có số lượng các thuộc tính giống nhau và đồng nhất**, **giữa các loại đối tượng có mối quan hệ** với nhau thông qua một hoặc một số thuộc tính.	**Tính linh hoạt:** NoSQL cho phép lưu trữ dữ liệu mà không cần định rõ cấu trúc trước, giúp linh hoạt khi thêm hoặc thay đổi dữ liệu. Phù hợp lưu trữ dữ liệu **không có cấu trúc rõ ràng** hoặc **cấu trúc không đồng nhất**
Sử dụng SQL trong truy vấn dữ liệu	Không sử dụng SQL, mà sử dụng các API và phương thức khác
SQL thường phát triển theo mô hình cơ sở dữ liệu quan hệ (RDBMS) có thể **khó mở rộng khi cần thêm nhiều node hoặc khi dữ liệu tăng lên đáng kể**.	NoSQL thường có **khả năng mở rộng tốt hơn**, cho phép mở rộng dữ liệu dễ dàng bằng cách thêm các node hoặc tăng hiệu suất mà không cần thay đổi cấu trúc.
Tóm lại: SQL thích hợp cho các ứng dụng yêu cầu dữ liệu **có cấu trúc và có quan hệ phức tạp** giữa các đối tượng.	Tóm lại: NoSQL thích hợp cho các ứng dụng yêu cầu **tính linh hoạt cao và khả năng mở rộng tốt**, đặc biệt là khi xử lý dữ liệu phi cấu trúc và có nhu cầu lưu trữ dữ liệu lớn và đa dạng.

5. **Trong trường hợp write nhiều và read nhiều thì điều gì sẽ có thể xảy ra và giải quyết như thế nào?**

Một số vấn đề xảy ra khi quá trình write và read nhiều, đặc biệt là các truy vấn xảy ra ở các thời điểm gần nhau bao gồm:

- Thời gian thực thi và phản hồi yêu cầu truy vấn lâu, liên quan tới:
  - Kích thước của các bảng dữ liệu
  - Các lệnh join giữa các bảng làm tăng độ phức tạp tìm kiếm
  - Nhiều yêu cầu đồng thời vượt quá khả năng xử lý của DBMS
- Tính đúng đắn của dữ liệu bị ảnh hưởng, bị sai về logic, do các lệnh cập nhật, đặc biệt là lệnh write thực hiện bất đồng bộ.
  Ví dụ: Thực hiện rút tiền và cập nhật số dư, giả sử logic của quá trình này bao gồm: kiểm tra số dư hiện tại, so sánh số dư và số tiền rút, cho phép rút hoặc không, sau đó cập nhật lại số dư. Nếu có từ 2 truy vấn này trở lên được thực hiện bất đồng bộ mà không có cơ chế bảo vệ, dữ liệu số dư trả về cho từng truy vấn sẽ được hiểu lầm là phù hợp và cả hai hành động rút tiền sẽ được thực thi, dẫn đến tổng số tiền được rút có thể lớn hơn số dư vốn có, và số dư cuối cùng trong database trở nên âm.

Một số hướng giải quyết:

- Để tăng tốc độ đọc dữ liệu:
  - **Partition database**: phân chia các bảng dữ liệu lớn, gồm phân chia theo chiều dọc (chia thành các bảng nhỏ với số lượng các cột ít hơn) hoặc phân chia theo chiều ngang (chia bảng lớn thành các bảng nhỏ hơn với số cột giữ nguyên nhưng số dòng ít hơn, có thể liên quan tới khu vực địa lý, khoảng thời gian, nhóm đối tượng khách hàng,…)
  - **Sử dụng cache**: bản chất nội dung của database thường sẽ được lưu trữ trên các đĩa cứng và quản lý bằng hệ quản trị cơ sở dữ liệu (DBMS). Tuy nhiên nếu số lượng lệnh đọc dữ liệu nhiều, tùy vào yêu cầu thực tế và loại dữ liệu, loại thuộc tính nào thường được truy vấn, ta sẽ lưu lại dữ liệu trên cache (bản chất là RAM, có tốc độ truy xuất nhanh hơn ổ cứng) để tăng tốc. Việc này sẽ phải đánh đổi bằng việc cung cấp thêm tài nguyên phần cứng, do đó tùy vào nghiệp vụ ta có thể cài đặt giải thuật để cung cấp thời gian sống cho dữ liệu trên cache phù hợp, để thu hồi và cấp mới cho các dữ liệu khác.
- Đảm bảo tính nhất quán, đúng đắn cho dữ liệu khi có nhiều yêu cầu ghi vào cơ sở dữ liệu:
  - Phân tích: Việc ghi dữ liệu bất đồng bộ sẽ bị sai nếu các lệnh ghi này cùng tác động lên 1 dòng, hoặc tác động lên các dòng của các bảng có mối quan hệ với nhau (các instances khác nhau của cùng một core service).
    Trên mô hình hiện tại (của team), yêu cầu ghi xuất phát từ các clients, đi qua api-gateway, sau đó đến các middleware, sau đó đến các core services, và cuối cùng đến cơ sở dữ liệu (được quản lý bởi DBMS).
    Nếu tất cả các bước đều được thực hiện bất đồng bộ mà không có cơ chế bảo vệ thì việc sai sót sẽ xảy ra, do đó nguyên lý áp dụng sẽ là chọn một vị trí trên các luồng đi của dữ liệu được mô tả bên trên để áp dụng một cơ chế khóa / cơ chế loại trừ lẫn nhau (mutex), để tại đó, việc ghi dữ liệu phải được thực hiện

tuần tự. Việc tuần tự này có thể sẽ gây nghẽn cổ chai, ảnh hưởng tới tính phân tán của microservices, do đó ta sẽ lựa chọn vị trí có khả năng xử lý nhanh nhất để đặt vào ngay trước đó một hệ thống cho phép mutex.
(Mutex (binary semaphore, chỉ có giá trị 0 và 1, chỉ có phương thức Up và Down): Khi có các yêu cầu muốn thay đổi dữ liệu cần được bảo vệ, yêu cầu này sẽ kiểm tra mutex có đang phải 1 không, nếu 1 nghĩa là sẽ được thực thi và ngay lập tức Down để đưa mutex về 0. Khi thực thi xong (tức dữ liệu ghi hoàn tất), mutex sẽ được Up thành 1. Ở một yêu cầu xử lý khác, nếu gặp mutex bằng 0 thì xử lý này sẽ phải đợi (sleep), đợi đến khi mutex là 1 thì sẽ được đánh thức (wake up) và thực thi. Áp dụng tương tự cho đến khi tất cả các yêu cầu ghi dữ liệu được thực thi xong).

- Kết quả tìm hiểu: Lựa chọn vị trí đặt mutex là trên đường đi của yêu cầu ghi dữ liệu từ các core-services đến database. Các yêu cầu ghi dữ liệu bất đồng bộ sẽ được "tuần tự hóa" đến database bởi mutex, do đó sẽ đảm bảo tính đúng đắn cho dữ liệu.
    - redis-mutex: https://github.com/kenn/redis-mutex
    - Sau khi sử dụng giải pháp này, các yêu cầu ghi dữ liệu từ các instances của core service cùng loại sẽ nhận được quyền thực thi hoặc đợi cho đến khi được phép thực thi, một cách tuần tự