Transaction trong java là cái vẹo gì? Tại sao phải sử dụng Transaction? ACID transactions?

G’day everyone!   
“**Em có biết gì về Transaction không?**” Chắc hẳn các bạn đã từng ít nhiều gặp câu hỏi này khi phỏng vấn xin việc, nghe quen quen đấy nhỉ ☺. Thật đáng tiếc khi phải tự mình thốt ra “Em không biết ạ!” hoặc “Em có nghe qua rồi nhưng….” :v.

Mình cũng từng trải qua và biết rất rõ cảm giác lúc đó, hối tiếc có, hụt hẫng cũng có nên hôm nay mình có mặt ở đây để tổng hợp lại kiến thức mình đã thu thập được. Dù cũng chỉ dừng lại ở mặt lý thuyết nhưng mong phần nào sẽ giúp được các bạn hiểu đượt đôi chút “**nó**” là cái gì. Let’s go!!

**Công Dụng**

- Trước tiên hãy đi vào ví dụ “huyền thoại” được lưu truyền từ nhiều bài viết:

+ Hôm nay là ngày bạn phải thực hiện nghĩa vụ chuyển tiền cho Vợ (kế toán giải ngân) theo hàng tháng

+ Bạn đã thực hiện chuyển tiền 2 tỉ (nhiều nhiều mới sót) và tài khoản của bạn đã bị trừ đi số tiền tương ứng

+ Nhưng có một lỗi nào đó mà tài khoản của vợ không nhận được

* Bạn bị mang tiếng trễ hạn và giấu quỹ đen.

- Đó là hậu quả thực tế của việc không áp dụng Transaction. Nếu áp dụng, Transaction sẽ có 2 hành động (operations):

+ Tài khoản của bạn bị trừ 2 tỉ

+ Tài khoản của vợ được cộng thêm 2 tỉ

- Nếu một trong 2 hành động trên bị lỗi thì Transaction sẽ bị hủy và tài khoản của bạn sẽ không bị thay đổi, quá an toàn đúng không. Trính vì ràng buộc trên nên Transaction được ứng dụng nhiều trong các hành động yêu cầu độ tin cậy, đảm bảo tính chính xác dữ liệu…

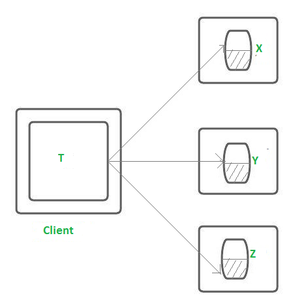
* Nhóm các hoạt động (với DB) của một tác vụ
* Nếu có lỗi xảy ra trong Transaction thì tất cả các tác vụ sẽ thực hiện sẽ thất bại
* Đảm bảo tính toàn vẹn của phiên làm việc với cớ sở dữ liệu

=> Đó là 3 công dụng chính của Transaction!

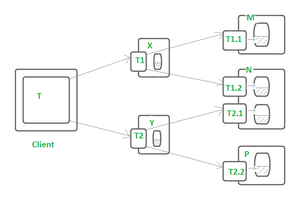
**Các loại Transaction**

- Transaction được chia thành 2 loại chính:

* Flat Transaction: Thực thi các operation là tuần tự từ trái sang phải hoặc từ trên xuống dưới



* Nested Transaction: Các operation lồng nhau, việc thực thi các operation dựa theo nguyên tắc từ trong ra ngoài



**ACID Transaction**

Nghe hoang mang ha! Nhưng đừng lo vì nó chỉ đơn giản là viết tắt các tính chất trong Transaction. Ta sẽ đi vào từng chữ cái ACID.

- Atomicity - Tính đơn vị

* Transaction sẽ xác định ranh giới một cách rõ ràng điểm đầu – điểm cuối là gì, tức là điểm bắt đầu và điểm kết thúc ấy. Có ý nghĩa đơn giản là nếu một thành phần nào đó trong transaction thực thi lỗi thì đồng nghĩa với việc không có gì xảy ra tức không có gì thay đổi về mặt dữ liệu.

- Consistency - Tính nhất quán

* Dữ liệu nhất quán từ thời điểm bắt đầu đến thời điểm kết thúc
* Đảm bảo Db thay đổi chính xác theo trạng thái mà Transaction thực hiện

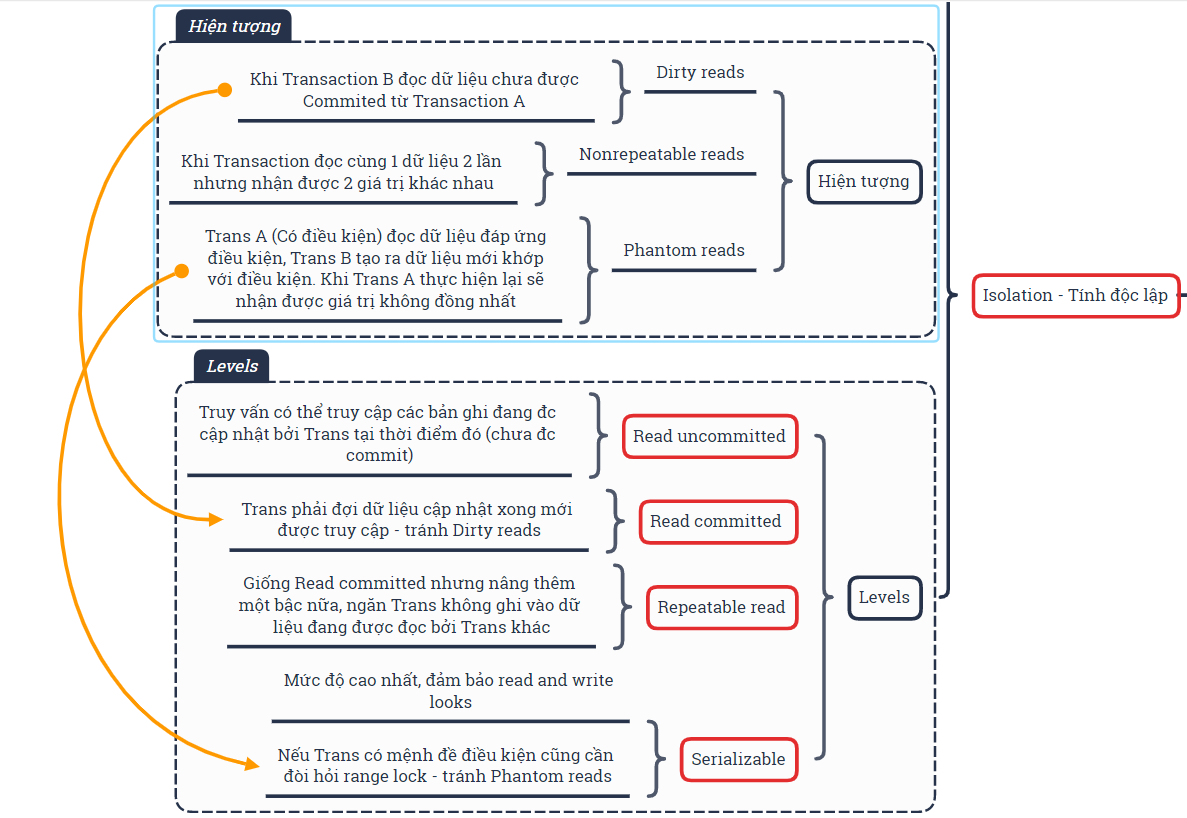
- Durability - Tính bền vững (tạm thời bỏ qua chữ I trong ACID, mình sẽ nói ở phần cuối)

* Dữ liệu của Transaction sau khi thực thi được cố định, không có chuyện có thể chuyển lại
* Hiểu đơn giản là nếu đã truyền tiền đi rồi thì bạn sẽ mất 2 tỉ trong tài khoản chứ không có chuyện 2 tỉ của bạn vẫn giữ nguyên.

- Isolation - Tính độc lập

* Các transaction có khả năng hoạt động một cách độc lập và không liên quan đến nhau.

Tạm thời các bạn cứ quan sát trong mô hình sau:



- Isolation sẽ được xác định qua 3 hiện tượng sau:

* Dirty reads: VD – Bạn cập nhật dữ liệu mới lên tài khoản cá nhân trong cty, sếp của bạn nhìn vào số liệu bạn đã cập nhật để báo lên cấp trên. Trong một ngày đẹp trời nào đó, bạn chủ quan cập nhật dữ liệu mà không kiểm tra lại. Dữ liệu không được commit thành công và trở thành dữ liệu cũ => sếp chửi!
* Nonrepeatable reads: VD – Bạn cập nhật dữ liệu lên tài khoản cty, sếp bạn xem và tổng hợp và báo cáo lên cấp trên. Bạn nhận ra là đã cập nhật sai và quyết định xóa đi để nhập lại. Vô tình sếp bạn nhìn lại và nhận ra các giá trị đã bị thay đổi trong khi đã báo cáo lên cấp trên rồi => sếp chửi!
* Phantom reads: VD – Sếp bạn tổng hợp dữ liệu để đáp ứng một số yêu cầu của cấp trên. Bạn cũng táy tổng hợp một bản dữ liệu khác muốn giúp sếp một tay và nộp lên. Sếp nhìn vào 2 bản tổng hợp, nhận ra có sự khác nhau và phân vân không biết nên nộp cái nào chính xác hơn => sếp chửi!

Như vậy, để tránh các hiện tượng trên để dẫn đến việc bạn bị sếp chửi thì cần phải khóa dữ liệu, không cho những tiến trình khác đụng vào dữ liệu khi Transaction đang làm việc. Có 3 loại khóa khác nhau:

* Write locks
* Read locks.
* Rang locks

Các levels sau đây có thể chi ra các mức độ khoác khác nhau:

* Read uncommitted: Vẫn có thể đọc được các bản ghi đang được câp nhật của các Transaction tại thời điểm đó
* Read committed: Phải đợi dữ liệu của Transaction đó được cập nhật xong => Khắc phục được hiện tượng Dirty reads
* Repeatable read: Giống Read uncommitted nhưng cao hơn một bậc ngăn Transaction không ghi vào dữ liệu đang được đọc bởi Transaction khác
* Serializable: Mực độ cao nhất, nếu Trans có mệnh đề điều kiện cũng cần đòi hỏi range lock - tránh Phantom reads

**Kết:**

**-** Đó là những kiến thức cơ bản về Transaction mình thu thập được sau khi đã tham khảo được qua khá nhiều nguồn và liên hệ đến các ví dụ dễ hiểu nhất có thể. Mong các bạn có thể dựa vào để khỏi lúng túng khi phỏng vấn nhé!

- Điều quan trọng cuối cùng là mong các bạn có thể giữ vững tinh thần lạc quan và sức khỏe trong mùa dịch. Stay fabulous! See ya!

References:

* <https://www.geeksforgeeks.org/flat-nested-distributed-transactions/>
* https://www.baeldung.com/transaction-configuration-with-jpa-and-spring