Discussion 06/03

Discussion 14-1

Consider the task of transferring $100 from account *A* to account *B*. How would you define the transaction(s) for the task? Why?

Options:

1. as one single transaction.
2. as two separate transactions:
   - one for subtracting $100 from *A* and
   - another for adding $100 to *B*.

1 번을 선택해야 하지 않을까? 이유는 2 번을 선택할 경우 A 과정을 commit 후에 B 연산을 처리하는 도중 에러가 발생해 rollback 해야 할 경우 문제가 생김.

Discussion 14-2

Consider the task of increasing every employee's salary by 5% for a company with 500 employees. How would you define the transaction(s) for the task? Why?

Options:

1. as one single transaction.
2. as ten separate transactions:
   one for every 50 employees.
3. as 500 separate transactions:
   one for each employee.

3 번이 좋지 않을까? 1 번과 2 번은 에러가 발생하면 너무 많은 연산을 rollback 해야 하므로 비효율적.

=〉 근데 트랜잭션 개수가 많아질수록 오버헤드가 커짐. 너무 많아도 안 좋다. Fail 이 없다면 옵션 1 도 괜찮음.

Discussion 14-3

Can you think of an interesting situation where a concurrent execution of two (correct) transactions produces an incorrect result? Provide the *schedule* of the situation.

A 에 2000 원이 있을 때 500 원을 넣고 2500 원을 뺄 경우

T1

Read A

A = A + 500

T2

Read A

A = A - 2500

Write A

T1

Write A

T1 의 결과가 쓰이기 전에 T2 에서 Write 가 실행되면 A 계좌가 -500 이 됨. Invalid.

Discussion 14-4

Consider the following situation.

① A user makes a reservation over the Web.

② The *database system crashes* just *after* the reservation transaction commits and before sending the result to the application server.

What should the DB system and the application server do to provide a consistent service?

What if the crash occurred *before* the transaction committed?

어플리케이션 서버와 DBMS 각각에 로그나 상태 등이 저장되어 있을 텐데, 데이터베이스를 복구할 때 로그를 읽어보고 트랜잭션이 commit 되어 있는데 application 서버의 상태에 반영이 되어있지 않으면 해당 트랜잭션 결과를 application 서버로 전송해준다.

트랜잭션이 commit 되기 전에 crash 가 발생하면 해당 트랜잭션을 rollback 해서 abort 상태로 만든 뒤 유저에게 오류 발생 알림을 띄워야 할 것 같다.

Discussion 14-5

Justify the following statement:

Concurrent execution of transactions is …

- *more important when data must be fetched from (slow) disk or when transactions are long, and*
- *less important when data are in memory and transactions are short.*

속도 측면에서 봤을 때 첫 번째 경우에서 동시 실행을 할 수 없다면 각각의 트랜잭션을 처리하는 데 시간이 너무 많이 걸리기 때문에? 두 번째는 각각의 트랜잭션의 수행 시간이 짧기 때문에 덜 중요하다.