Transaction

트랜잭션은 하나의 논리적 작업 단위를 구성하는 일련의 연산들의 집합.

한 계좌에서 10만 원을 인출하여 다른 계좌로 10만 원 입금하는 이체 작업은 전체 작업이 정상적으로 완료되거나, 만약 정상적으로 처리될 수 없는 경우에는 아무 것도 실행되지 않은 처음 상태로 되돌려져야 한다. 이러한 트랜잭션은 다양한 데이터 항목들을 액세스하고 갱신하는 프로그램 수행의 단위가 된다. 흔히 트랜잭션은 ACID 성질이라고 하는 다음의 네 가지 성질로 설명된다.

- Atomicity(원자성): 이체 과정 중에 트랜잭션이 실패하게 되어 예금이 사라지는 경우가 발생해서는 안 되기 때문에 DBMS는 완료되지 않은 트랜잭션의 중간 상태를 데이터베이스에 반영해서는 안 된다. 즉, 트랜잭션의 모든 연산들이 정상적으로 수행 완료되거나 아니면 전혀 어떠한 연산도 수행되지 않은 상태를 보장해야 한다. atomicity는 쉽게 'all or nothing' 특성으로 설명된다.

- Consistency(일관성): 고립된 트랜잭션의 수행이 데이터베이스의 일관성을 보존해야 한다. 즉, 성공적으로 수행된 트랜잭션은 정당한 데이터들만을 데이터베이스에 반영해야 한다. 트랜잭션의 수행을 데이터베이스 상태 간의 전이(transition)로 봤을 때, 트랜잭션 수행 전후의 데이터베이스 상태는 각각 일관성이 보장되는 서로 다른 상태가 된다. 트랜잭션 수행이 보존해야 할 일관성은 기본 키, 외래 키 제약과 같은 명시적인 무결성 제약 조건들뿐만 아니라, 자금 이체 예에서 두 계좌 잔고의 합은 이체 전후가 같아야 한다는 사항과 같은 비명시적인 일관성 조건들도 있다.

- Isolation(독립성): 여러 트랜잭션이 동시에 수행되더라도 각각의 트랜잭션은 다른 트랜잭션의 수행에 영향을 받지 않고 독립적으로 수행되어야 한다. 즉, 한 트랜잭션의 중간 결과가 다른 트랜잭션에게는 숨겨져야 한다는 의미인데, 이러한 isolation 성질이 보장되지 않으면 트랜잭션이 원래 상태로 되돌아갈 수 없게 된다. Isolation 성질을 보장할 수 있는 가장 쉬운 방법은 모든 트랜잭션을 순차적으로 수행하는 것이다. 하지만 병렬적 수행의 장점을 얻기 위해서 DBMS는 병렬적으로 수행하면서도 일렬(serial) 수행과 같은 결과를 보장할 수 있는 방식을 제공하고 있다.

- Durability(지속성): 트랜잭션이 성공적으로 완료되어 커밋되고 나면, 해당 트랜잭션에 의한 모든 변경은 향후에 어떤 소프트웨어나 하드웨어 장애가 발생되더라도 보존되어야 한다.

Undo

오퍼레이션 수행 중에 수정된 페이지들이 버퍼 관리자의 버퍼 교체 알고리즘에 따라서 디스크에 출력될 수 있다. 버퍼 교체는 전적으로 버퍼의 상태에 따라서 결정되며, 일관성 관점에서 봤을 때는 임의의 방식으로 일어나게 된다. 즉 아직 완료되지 않은 트랜잭션이 수정한 페이지들도 디스크에 출력될 수 있으므로, 만약 해당 트랜잭션이 어떤 이유든 정상적으로 종료될 수 없게 되면 트랜잭션이 변경한 페이지들은 원상 복구되어야 한다.

이러한 복구를 UNDO라고 한다. 만약 버퍼 관리자가 트랜잭션 종료 전에는 어떤 경우에도 수정된 페이지들을 디스크에 쓰지 않는다면, UNDO 오퍼레이션은 메모리 버퍼에 대해서만 이루어지면 되는 식으로 매우 간단해질 수 있다. 이 부분은 매력적이지만 이 정책은 매우 큰 크기의 메모리 버퍼가 필요하다는 문제점을 가지고 있다. 수정된 페이지를 디스크에 쓰는 시점을 기준으로 다음과 같은 두 개의 정책으로 나누어 볼 수 있다.

- STEAL: 수정된 페이지를 언제든지 디스크에 쓸 수 있는 정책
- ¬STEAL: 수정된 페이지들을 최소한 트랜잭션 종료 시점(EOT, End of Transaction)까지는 버퍼에 유지하는 정책

STEAL 정책은 수정된 페이지가 어떠한 시점에도 디스크에 써질 수 있기 때문에 필연적으로 UNDO 로깅과 복구를 수반하는데, 거의 모든 DBMS가 채택하는 버퍼 관리 정책이다.

Redo

이제는 UNDO 복구의 반대 개념인 REDO 복구에 대해서 알아볼 것인데, 앞서 설명한 바와 같이 커밋한 트랜잭션의 수정은 어떤 경우에도 유지(durability)되어야 한다. 이미 커밋한 트랜잭션의 수정을 재반영하는 복구 작업을 REDO 복구라고 하는데, REDO 복구 역시 UNDO 복구와 마찬가지로 버퍼 관리 정책에 영향을 받는다. 트랜잭션이 종료되는 시점에 해당 트랜잭션이 수정한 페이지들을 디스크에도 쓸 것인가 여부로 두 가지 정책이 구분된다.

- FORCE: 수정했던 모든 페이지를 트랜잭션 커밋 시점에 디스크에 반영하는 정책
- ¬FORCE: 수정했던 페이지를 트랜잭션 커밋 시점에 디스크에 반영하지 않는 정책

여기서 주의 깊게 봐야 할 부분은 ¬FORCE 정책이 수정했던 페이지(데이터)를 디스크에 반영하지 않는다는 점이지 커밋 시점에 어떠한 것도 쓰지 않는다는 것은 아니다. 어떤 일들을 했었다고 하는 로그는 기록하게 되는데 이 부분은 아래에서 자세히 설명한다.

FORCE 정책을 따르면 트랜잭션이 커밋되면 수정되었던 페이지들이 이미 디스크 상의 데이터베이스에 반영되었으므로 REDO 복구가 필요 없게 된다. 반면에 ¬FORCE 정책을 따른다면 커밋한 트랜잭션의 내용이 디스크 상의 데이터베이스 상에 반영되어 있지 않을 수 있기 때문에 반드시 REDO 복구가 필요하게 된다. 사실 FORCE 정책을 따르더라도 데이터베이스 백업으로부터의 복구, 즉 미디어(media) 복구 시에는 REDO 복구가 요구된다. 거의 모든 DBMS가 채택하는 정책은 ¬FORCE 정책이다.

정리해보면 DBMS는 버퍼 관리 정책으로 STEAL과 ¬FORCE 정책을 채택하고 있어, 이로 인해서 UNDO 복구와 REDO 복구가 모두 필요하게 된다.

트랜잭션 관리

지금까지 설명한 UNDO 복구와 REDO 복구를 위해서 가장 널리 쓰이는 구조는 로그(log)이다. Shadow paging(nilavalagan, 2009)이라고 불리는 복구 기법도 존재하지만, 여기서는 보편적으로 사용되는 로그 기법에 대해서만 설명하기로 한다.

로그

로그는 로그 레코드의 연속이며 데이터베이스의 모든 갱신 작업을 기록한다. 로그는 이론적으로는 안정적 저장 매체(stable storage)에 기록된다고 하는데, 안정적 저장 매체는 어떤 경우에도 절대로 손실이 발생하지 않는 이른바 이상적인 매체이다. 바꿔 말하면 현실 상에서는 존재하지 않는다고 봐야 하는데, RAID 등 인프라 시스템의 도움 외에도 DBMS 자체적으로 여러 벌의 로그를 유지하는 등 안정적 저장 매체처럼 동작하게 하는 기법을 사용하기도 한다. 하지만 대부분 DBMS는 성능 상의 이유로 하나의 로그를 유지한다.

Recovery process

A Transaction Processing System may fail for many reasons such as system failure, human errors, hardware failure, incorrect or invalid data, computer viruses, software application errors or natural or man-made disasters. As it's not possible to prevent all failures, a TPS must be able to detect and correct errors when they occur and cope with failures. A TPS will go through a recovery of the database which may involve the backup, journal, checkpoint, and recovery manager:

Checkpoint

The purpose of checkpointing is to provide a snapshot of the data within the database. A checkpoint, in general, is any identifier or other reference that identifies the state of the database at a point in time. Modifications to database pages are performed in memory and are not necessarily written to disk after every update. Therefore, periodically, the database system must perform a checkpoint to write these updates which are held in-memory to the storage disk. Writing these updates to storage disk creates a point in time in which the database system can apply changes contained in a transaction log during recovery after an unexpected shut down or crash of the database system. If a checkpoint is interrupted and a recovery is required, then the database system must start recovery from a previous successful checkpoint.

Two-phase locking

In databases and transaction processing, two-phase locking (2PL) is a concurrency control method that guarantees serializability. The protocol utilizes locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life.

By the 2PL protocol, locks are applied and removed in two phases:

1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

Write-lock (exclusive lock) is associated with a database object by a transaction (Terminology: "the transaction locks the object," or "acquires lock for it") before *writing* (inserting/modifying/deleting) this object.

Read-lock (shared lock) is associated with a database object by a transaction before *reading* (retrieving the state of) this object.

Lock compatibility table

Lock type	read-lock	write-lock
read-lock		X
write-lock	X	X