MYONGJI UNIVERSITY

Transaction

Dongseop Kwon

Goal

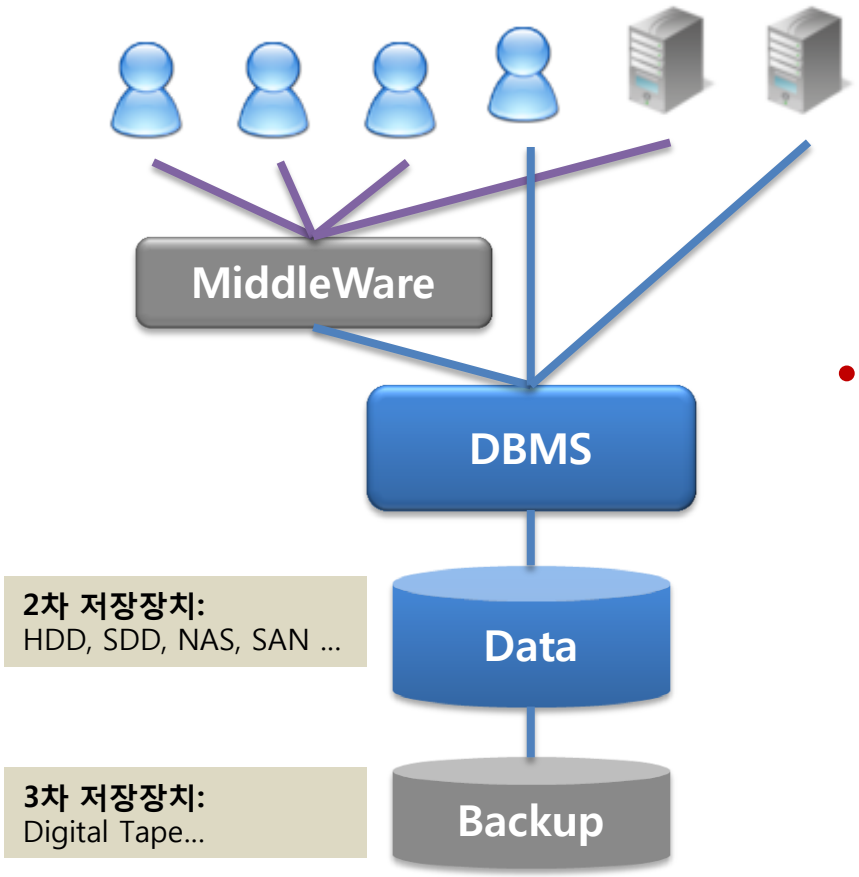
- 트랜잭션의 정의
- ACID Properties
- Transaction Isolation Level

Transaction

- A **sequence** of **operations** treated as a **unit**.
  - Operation == SQL
  - Unit == All or Nothing

- SQL 표준에서의 Transaction
  - SQL문을 이용하면 자동적으로 Transaction시작
  - **Commit**을 호출하면 트랜잭션 종료 (혹은 세션이 종료될 때)
  - **Rollback**을 호출하면 트랜잭션 취소
  - **Autocommit** 모드일 경우 SQL 문장 하나 단위로 Transaction

트랜잭션의 목적



- **Concurrency**
  - 동시에 여러 요청을 처리
  - Concurrency Control
- **Recovery**
  - 회복/복구
  - 시스템 오류로부터 데이터 보호

Concurrency Control의 목적

- **정확성**
  - 정확한 결과를 보장
  - **Consistency**: 결과의 일관성, 무결성을 보장
    - Constraint등의 Integrity가 보장되어야 함
  - **Isolation**: 각 트랜잭션이 개별적(독립적)으로 수행된 결과와 동일한 결과를 원함

- **성능**
  - 가능한 동시에 여러 트랜잭션을 수행하여 성능을 높임
  - Concurrent execution:병행 처리

Attribute-level Inconsistency

T1: UPDATE products **SET** qty = qty – 10 **WHERE** id = 1233

T2: UPDATE products **SET** qty = qty – 30 **WHERE** id = 1233

PRODUCTS

ID	NAME	QTY	PRICE
…	…	…	…
1233	iPad2	100	800,000
…	…	…	…

T1

1233	iPad2	100 - 10	800,000

T2

1233	iPad2	100 - 30	800,000

Tuple-level Inconsistency

T1: UPDATE products **SET** qty = qty – 10 **WHERE** id = 1233

T2: UPDATE products **SET** price = 500000 **WHERE** id = 1233

PRODUCTS

ID	NAME	QTY	PRICE
…	…	…	…
1233	iPad2	100	800,000
…	…	…	…

T1

1233	iPad2	100 - 10	800,000

T2

1233	iPad2	100	500,000

Relation-level Inconsistency

T1: UPDATE products **SET** price = price * 0.5 **WHERE** qty >= 200

T2: UPDATE products **SET** SALE = Y **WHERE** price <= 400000

PRODUCTS

ID	NAME	QTY	PRICE	SALE
…	…	…	…	…
1233	iPad2	300	800000	N
1477	NEX3	300	500000	N
…	…	…	…	…

T1

PRODUCTS

ID	NAME	QTY	PRICE	SALE
…	…	…	…	…
1233	iPad2	300	400000	N
1477	NEX3	300	500000	N
…	…	…	…	…

T2

System Crash로 부터의 안전성

```
UPDATE products SET qty = qty – 1 WHERE id = 1311;
INSERT INTO sales VALUES ('dongseop', 1311, 1, 500000);
```

- **Atomicity**
  - 문제점: 재고는 줄어들었는데 판매 내역은 없다???
  - All-or-Nothing
    - 판매 기록도 되고, 재고도 남던지,
      아니면 판매기록도 없고, 재고도 그대로던지...

- **Durability**
  - 문제점: 분명 정상적으로 구매완료가 되었는데...
    오늘 정전으로 구매가 안된 것으로 나옴???
  - 지속성:
    - 한번 완료된 트랜잭션은 영원히 완료된 것으로 있어야 함

ACID Properties: 트랜잭션이 지녀야 할 기본 속성

- **A**tomicity
  - 원자성: 트랜잭션의 작업들이 모두 수행되거나 전혀 수행되지 않아야 함. 일부만 수행된 상태가 되어서는 안됨. (all-or-nothing)
- **C**onsistency
  - 일관성: 트랜잭션의 수행 이후에도 데이터는 항상 일관되고 무결성이 유지된 상태에 있어야 함 (integrity 보장)
- **I**solation
  - 고립성: 각 트랜잭션은 다른 트랜잭션의 수행에 영향을 끼치지 않아야 함
  - **Serializability: 각 트랜잭션을 따로 수행한 결과와 동일하여야 함**
    - **트랜잭션 간의 순서와는 무관함**
- **D**urability
  - 지속성: 한번 commit된 트랜잭션의 결과는 계속적으로 유지되어야 한다.

Serializable (직렬가능)

- 다음 트랜잭션의 올바른 수행 결과는?

T1: UPDATE t **SET** a = a * 2**;**

T2: UPDATE t **SET** a = a + 100**;**

..	a	...
...	100	...
...	200	...

T1 → T2

..	a	...
...	300	...
...	500	...

T2 → T1

..	a	...
...	400	...
...	600	...

- 트랜잭션을 하나씩 차례로 수행(serial schedule)했을 때 나올 수 있는 결과는 모두 올바른 것임

Isolation Level

- Read Uncommitted
- Read Committed
- Repeatable Read
- **Serializable : Serial한 수행과 동일한 결과를 보장**

높은 수준의 isolation level일 수록
concurrency는 떨어지지만 (성능)
더 높은 수준의 **consistency**를 보장한다. (정확성)

Dirty Read

- commit되지 않은 데이터 변경을 읽을 수 있음
  - 문제점
    - Serializable 하지 않은 결과가 나올 수 있음
    - Rollback이 일어나면???

```
T1: UPDATE products SET qty = 500 WHERE id = 1233
      ....
      ....
```

```
T2: SELECT qty, supplier_id FROM products WHERE id = 1233
    if qty > 100 then
       UPDATE suppliers SET decision = 'Y' WHERE id = supplier_id
    end
```

```
T3: SELECT COUNT(*) FROM suppliers WHERE decision = 'Y'
```

Isolation Level: Read Uncommitted

T1: UPDATE products **SET** price = price * 1.2

T2: Set Transaction Isolation Level Read Uncommitted;
SELECT AVG(price**) FROM** products

- Dirty read를 허용함
  - T1 수행 중에 T2가 동시에 수행 가능함
- Serializable 하지 않은 Schedule 발생 가능
  - T1→T2 or T2→T1으로 나올 수 없는 결과가 나올 수 도 있음

Isolation Level: Read Committed

T1: UPDATE products **SET** price = price * 1.2

T2: Set Transaction Isolation Level Read Committed;
SELECT AVG(price**) FROM** products;
SELECT AVG(price**) FROM** products;

- **Dirty Read를 허용하지 않음**
  - T1의 수행 중간에 T2가 수행되지 않음
- **Nonrepeatable Read:**
  - Repeatable하지 않은 read가 발생할 수 있음
  - T2:S →T1:U →T2:S 순서로 수행가능
  - 동일한 SELECT문인데 서로 다른 결과가 발생
- Serializable 하지 않은 Schedule 발생 가능

Isolation Level: Repeatable Read

```
T1: UPDATE products SET price = price * 1.2
```

```
T2: Set Transaction Isolation Level Repeatable Read;
    SELECT AVG(price) FROM products;
    SELECT AVG(price) FROM products;
```

- **Dirty Read를 허용하지 않음**
- **Repeatable Read 보장**
  - 이미 읽은 데이터는 다시 읽어도 동일한 값이 되도록 보장
- Serializable 하지 않은 Schedule 발생 가능

```
T1: UPDATE products SET price = price * 1.2
    UPDATE suppliers SET amount = amount + 1000
```

```
T2: Set Transaction Isolation Level Repeatable Read;
    SELECT AVG(price) FROM products;
    SELECT AVG(amount) FROM suppliers ;
```

Isolation Level: Repeatable Read

- **Phantom Read problem** 발생 가능

T1: INSERT INTO products **VALUES (.....);**

T2: Set Transaction Isolation Level Repeatable Read;
SELECT AVG(price**) FROM** products;
SELECT AVG(price**) FROM** products;

  - 한 번 읽었던 Tuple의 값은 변경 안되지만, 새로운 Tuple이 추가 될 수는 있음 (이미 읽었던 tuple은 아니므로)
  - T1에 의하여 추가된 tuple은 이전에 존재하지 않았으므로 T2:S1 →T1→T2:S2 순서로 수행 가능
  - 동일한 결과가 나오지 않음

Isolation Level: Serializable

- 항상 Serializable한 결과 보장
- No Dirty Read
- Repeatable Read
- No Phantom Read

Isolation Level 요약

- ANSI/ISO 표준 Transaction Isolation Level

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read Uncommitted	가능	가능	가능
Read Committed	-	가능	가능
Repeatable Read	-	-	가능
Serializable	-	-	-

- **주의!**
  - 실제 DBMS별로 제공하는 Transaction Level과 Default가 다름
  - **Oracle: Read Committed(default), Serializable** 만 제공
  - MS-SQL Server: Read Uncommitted, Read Committed(default), Repeatable Read, Snapshot, Serializable 제공
  - MySQL: InnoDB에서만 사용 가능. Repeatable Read가 default
  - Postgresql: Read Committed(default), Serializable 제공
    (다른 모드는 제공은 되지만, 실제로는 두가지로 동작)

Concurrency Control 구현 방법

- 대부분 **Lock**을 이용하여 구현됨
  - Two-phased Locking Protocol
  - Lock overhead나 waiting이 주요한 성능 저하의 원인
  - **Deadlock 문제 발생 가능**
    - 서로 순환적으로 Lock을 대기하여 트랜잭션 수행이 정지됨

- Transaction의 Isolation Level이나 속성을 잘 지정해야 함
  - 높은 수준의 isolation level이 일반적으로 더 많은 lock을 요구
  - **SET TRANSACTION READ ONLY**
    - Read연산만 가능한 Transaction 정의
    - 일반적으로 성능 향상에 도움을 줌
    - Oracle의 경우 Transaction Level의 Read Consistency를 보장
      - No-Dirty Read, Repeatble Read, No-Phantom Read

Recovery 구현 방법

- 일반적으로 LOG를 별도로 기록함
  - REDO
  - UNDO

- Checkpoint 기법을 주기적으로 사용
  - Recovery 시간을 절약