**1. Lock 기본**

**가. Lock이란?**

고가의 DBMS를 사용하는 이유로는 성능, 관리의 편이성 등 여러 가지 측면이 있지만, 무엇보다 트랜잭션 처리 능력이 가장 기본적이고 핵심적인 요소라고 할 수 있다. 같은 자원을 액세스하려는 다중 트랜잭션 환경에서 데이터베이스의 일관성과 무결성을 유지하려면 트랜잭션의 순차적 진행을 보장할 수 있는 직렬화(serialization) 장치가 필요하다. 영화관 좌석을 예약하는 시스템을 예로 들면, 두 명이 동시에 좌석을 요청할 때 정확히 한 명만 좌석을 배정받도록 할 수 있어야 한다. 이런 직렬화가 가능하도록 하기 위해 모든 DBMS가 공통적으로 사용하는 메커니즘이 Lock이다. 중요한 것은 DBMS마다 Lock을 구현하는 방식과 세부적인 기능이 많이 다르다는 사실이다. 따라서 자신이 사용하고 있는 DBMS만의 독특한 Lock 메커니즘을 정확히 이해하지 못한 상태에선 결코 고품질 데이터베이스를 구축할 수 없다. 본 장이 중요한 의미를 갖는 이유가 여기에 있는데, DBMS별 특징을 설명하기에 앞서 Lock에 대한 기본적인 개념부터 살펴보자.

**나. 공유 Lock과 배타적 Lock**

DBMS는 각 트랜잭션의 오퍼레이션별로 적당한 수준의 Lock을 자동으로 설정한다. 필요한 경우, 일부 Lock에 대해서는 사용자가 직접 제어하는 방법도 제공한다. 가장 기본이 되는 Lock 모드는 공유 Lock과 배타적 Lock이다. 이에 대해서 간단히 살펴보고, DBMS마다 제공되는 세부적인 Lock 모드에 대해서는 뒤에서 살펴보기로 하자.

1) 공유 Lock

공유(Shared) Lock은 데이터를 읽고자 할 때 사용된다. 다른 공유 Lock과는 호환되지만 배타적 Lock과는 호환되지 않는다. ‘호환된다’는 말은 한 리소스에 두 개 이상의 Lock을 동시에 설정할 수 있음을 뜻한다. 다시 말해, 공유 Lock을 설정한 리소스에 다른 트랜잭션이 추가로 공유 Lock을 설정할 수는 있지만 배타적 Lock은 불가능하다. 따라서 자신이 읽고 있는 리소스를 다른 사용자가 동시에 읽을 수는 있어도 변경은 불가능하다. 반대로, 다른 사용자가 읽고 있는 리소스를 동시에 읽을 수는 있어도 변경 중인 리소스를 동시에 읽을 수는 없다.

2) 배타적 Lock

배타적(Exclusive) Lock은 데이터를 변경하고자 할 때 사용되며, 트랜잭션이 완료될 때까지 유지된다. 말 그대로 배타적이기 때문에 그 Lock이 해제될 때까지 다른 트랜잭션은 해당 리소스에 접근할 수 없다. 변경이 불가능할 뿐만 아니라 읽기도 불가능하다. 반대로, 다른 트랜잭션에 의해 Lock이 설정된 리소스는, 그것이 공유 Lock이든 배타적 Lock이든, 배타적 Lock을 동시에 설정할 수 없다.

**다. 블로킹과 교착상태**

1) 블로킹

블로킹(Blocking)은, Lock 경합이 발생해 특정 세션이 작업을 진행하지 못하고 멈춰 선 상태를 말한다. 공유 Lock끼리는 호환되기 때문에 블로킹이 발생하지 않는다. 공유 Lock과 배타적 Lock은 호환되지 않아 블로킹이 발생할 수 있다. 배타적 Lock끼리는 당연히 호환되지 않는다. 블로킹 상태를 해소하는 방법은 커밋(또는 롤백)뿐이다. 즉, Lock 경합이 발생하면 먼저 Lock을 설정한 트랜잭션이 완료될 때까지 후행 트랜잭션은 기다려야 하며, 이런 현상이 자주 나타난다면 사용자가 느끼는 애플리케이션 성능이 좋을 리 만무하다. Lock에 의한 성능 저하를 최소화하는 방안을 살펴보자.

① 우선, 트랜잭션의 원자성을 훼손하지 않는 선에서 트랜잭션을 가능한 짧게 정의하려는 노력이 필요하다. Oracle은 데이터를 읽을 때 공유 Lock을 사용하지 않기 때문에 다른 DBMS에 비해 상대적으로 Lock 경합이 적게 발생한다. 그렇더라도 배타적 Lock끼리 발생하는 경합은 피하지 못하므로 ‘불필요하게’ 트랜잭션을 길게 정의해선 안 된다.

② 같은 데이터를 갱신하는 트랜잭션이 동시에 수행되지 않도록 설계하는 것도 중요하다. 특히, 트랜잭션이 활발한 주간에 대용량 갱신 작업을 수행해선 안 된다.

③ 주간에 대용량 갱신 작업이 불가피하다면, 블로킹 현상에 의해 사용자가 무한정 기다리지 않도록 적절한 프로그래밍 기법을 도입해야 한다. 예를 들어, SQL Server에서는 세션 레벨에서 LOCK\_TIMEOUT을 설정할 수 있다. 아래는 Lock에 의한 대기 시간이 최대 2초를 넘지 않도록 설정한 것이다.

set lock\_timeout 2000

Oracle이라면 update/delete 문장을 수행하기 전에 nowait이나 wait 옵션을 지정한 select … for update 문을 먼저 수행해 봄으로써 Lock이 설정됐는지 체크할 수 있고, 발생한 예외사항(exception)에 따라 적절한 조치를 취할 수 있다.

select \* from t where no = 1 for update nowait → 대기없이 Exception을 던짐 select \* from t where no = 1 for update wait 3 → 3초 대기 후 Exception을 던짐

④ 트랜잭션 격리성 수준(2절 2항 참조)을 불필요하게 상향 조정하지 않는다.

⑤ 트랜잭션을 잘 설계하고 대기 현상을 피하는 프로그래밍 기법을 적용하기에 앞서, SQL 문장이 가장 빠른 시간 내에 처리를 완료하도록 하는 것이 Lock 튜닝의 기본이고 효과도 가장 확실하다.

2) 교착상태

교착상태(Deadlock)는, 두 세션이 각각 Lock을 설정한 리소스를 서로 액세스하려고 마주보며 진행하는 상황을 말하며, 둘 중 하나가 뒤로 물러나지 않으면 영영 풀릴 수 없다. 흔히 좁은 골목길에 두 대의 차량이 마주 선 것에 비유하곤 한다. 교착상태가 발생하면, DBMS가 둘 중 한 세션에 에러를 발생시킴으로써 문제를 해결하는데, 이를 방지하려면 어떻게 해야 할까? 조금 전 설명한 Lock 튜닝 방안은 교착상태 발생 가능성을 줄이는 방안이기도 하다. 여러 테이블을 액세스하면서 발생하는 교착상태는 테이블 접근 순서를 같게 처리하면 피할 수 있다. 예를 들어, 마스터 테이블과 상세 테이블을 둘 다 갱신할 때 마스터 테이블 다음에 상세 테이블을 갱신하기로 규칙을 정하고, 모든 애플리케이션 개발자가 이 규칙을 지킨다면 교착상태는 발생하지 않을 것이다. SQL Server라면 잠시 후 설명할 갱신(Update) Lock을 사용함으로써 교착상태 발생 가능성을 줄일 수 있다.

**2. SQL Server Lock**

**가. Lock 종류**

1) 공유 Lock

SQL Server의 공유 Lock은 트랜잭션이나 쿼리 수행이 완료될 때까지 유지되는 것이 아니라 다음 레코드가 읽히면 곧바로 해제된다. 단, 기본 트랜잭션 격리성 수준(Read Committed)에서만 그렇다. 격리성 수준을 변경하지 않고도 트랜잭션 내에서 공유 Lock이 유지되도록 하려면 아래와 같이 테이블 힌트로 holdlock을 지정하면 된다. 트랜잭션 격리성 수준에 대해서는 다음 절에서 설명한다.

begin tran select 적립포인트, 방문횟수, 최근방문일시, 구매실적 from 고객 with (holdlock) where 고객번호 = :cust\_num -- 새로운 적립포인트 계산 update 고객 set 적립포인트 = :적립포인트 where 고객번호 = :cust\_num commit

나중에 변경할 목적으로 레코드를 읽을 때는 반드시 위와 같은 패턴으로 트랜잭션을 처리해야 한다. 위 사례에서 방문횟수, 최근방문일시, 구매실적에 따라 새로운 적립포인트를 계산하는데, 만약 고객 데이터를 읽고 적립포인트를 변경하기 전에 다른 트랜잭션이 해당 고객 데이터를 변경했다면 적립포인트가 비일관된 상태에 놓일 수 있기 때문이다.

2) 배타적 Lock

1항에서 설명한 내용과 같다.

3) 갱신 Lock

앞서 공유 Lock을 설명하면서 예시했던 적립포인트 변경 프로그램을 공교롭게도 두 트랜잭션이 동시에 수행했다고 가정하다. 그것도 같은 고객에 대해서 말이다. 두 트랜잭션 모두 처음에는 공유 Lock을 설정했다가 적립포인트를 변경하기 직전에 배타적 Lock을 설정하려고 할 것이다. 그러면 두 트랜잭션은 상대편 트랜잭션에 의한 공유 Lock이 해제되기만을 기다리는 교착상태에 빠지게 된다. 이런 잠재적인 교착상태를 방지하려고 SQL Server는 갱신(Update) Lock을 두게 되었고, 이 기능을 사용하려면 아래와 같이 updlock 힌트를 지정하면 된다.

begin tran select 적립포인트, 방문횟수, 최근방문일시, 구매실적 from 고객 with (updlock) where 고객번호 = :cust\_num -- 새로운 적립포인트 계산 update 고객 set 적립포인트 = :적립포인트 where 고객번호 = :cust\_num commit

한 자원에 대한 갱신 Lock은 한 트랜잭션만 설정할 수 있다. 따라서 첫 번째 트랜잭션이 고객 데이터를 읽을 때 갱신 Lock을 설정하면 두 번째 트랜잭션은 첫 번째 트랜잭션이 배타적 Lock으로 전환했다가 이를 다시 해제할 때까지 기다려야만 한다. 갱신 Lock끼리는 호환되지 않지만 공유 Lock과는 호환되므로 갱신 Lock이 설정된 데이터를 단순히 읽고자 할 때는 기다리지 않아도 된다.

4) 의도 Lock

특정 로우에 Lock을 설정하면 그와 동시에 상위 레벨 개체(페이지, 익스텐트, 테이블)에 내부적으로 의도(Intent) Lock이 설정된다. Lock을 설정하려는 개체의 하위 레벨에서 선행 트랜잭션이 어떤 작업을 수행 중인지를 알리는 용도로 사용되며, 일종의 푯말(Flag)이라고 할 수 있다. 예를 들어, 구조를 변경하기 위해 테이블을 잠그려 할 때 그 하위의 모든 페이지나 익스텐트, 심지어 로우에 어떤 Lock이 설정돼 있는지를 일일이 검사해야 한다면 좀처럼 작업이 끝나지 않을 수 있다. 의도 Lock은 그런 현상을 방지해 준다. 즉, 해당 테이블에 어떤 모드의 의도 Lock이 설정돼 있는지만 보고도 작업을 진행할지 아니면 기다릴지를 결정할 수 있다.

5) 스키마 Lock

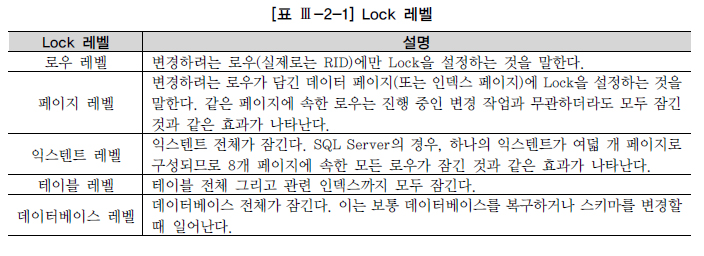
테이블 스키마에 의존적인 작업을 수행할 때 사용된다.

* Sch-S(Schema Stability) : SQL을 컴파일하면서 오브젝트 스키마를 참조할 때 발생하며, 읽는 스키마 정보가 수정되거나 삭제되지 못하도록 함
* Sch-M(Schema Modification) : 테이블 구조를 변경하는 DDL 문을 수행할 때 발생하며, 수정 중인 스키마 정보를 다른 세션이 참조하지 못하도록 함

6) Bulk Update Lock

테이블 Lock의 일종으로, 테이블에 데이터를 Bulk Copy 할 때 발생한다. 병렬 데이터 로딩(Bulk Insert나 bcp 작업을 동시 수행)을 허용하지만 일반적인 트랜잭션 작업은 허용되지 않는다.

**나. Lock 레벨과 Escalation**



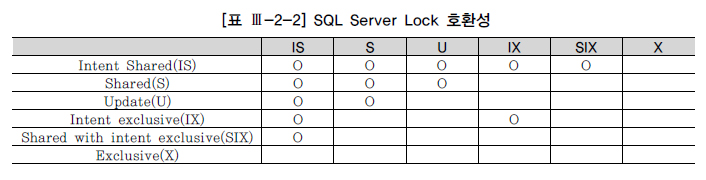
위 5가지 레벨 외에 인덱스 키(Key)에 로우 레벨 Lock을 거는 경우도 있다.

Lock Escalation

‘Lock Escalation’이란 관리할 Lock 리소스가 정해진 임계치를 넘으면서 로우 레벨 락이 페이지, 익스텐트, 테이블 레벨 락으로 점점 확장되는 것을 말한다. 이는 SQL Server, DB2 UDB처럼 한정된 메모리 상에서 Lock 매니저를 통해 Lock 정보를 관리하는 DBMS에서 공통적으로 발생할 수 있는 현상이다. Locking 레벨이 낮을수록 동시성은 좋지만 관리해야 할 Lock 개수가 증가하기 때문에 더 많은 리소스를 소비한다. 반대로, Locking 레벨이 높을수록 적은 양의 Lock 리소스를 사용하지만 하나의 Lock으로 수많은 레코드를 한꺼번에 잠그기 때문에 동시성은 나빠진다.

**다. Lock 호환성**

‘호환된다’는 말은 한 리소스에 두 개 이상의 Lock을 동시에 설정할 수 있음을 뜻한다. 앞서 설명한 Lock 종류별로 호환성을 요약하면 [표 Ⅲ-2-2]와 같다.(‘O’는 두 모드 간에 호환성이 있음을 의미함)



스키마 Lock의 호환성은 다음과 같다.

* Sch-S는 Sch-M을 제외한 모든 Lock과 호환된다.
* Sch-M은 어떤 Lock과도 호환되지 않는다.

**3. Oracle Lock**

Oracle은 공유 리소스와 사용자 데이터를 보호할 목적으로 DML Lock, DDL Lock, 래치(Latch), 버퍼 Lock, 라이브러리 캐시 Lock/Pin 등 다양한 종류의 Lock을 사용한다. 이들 중 애플리케이션 개발 측면에서 가장 중요하게 다루어야 할 Lock은 무엇보다 DML Lock이다. DML Lock은, 다중 사용자에 의해 동시에 액세스되는 사용자 데이터의 무결성을 보호해 준다. DML Lock에는 로우 Lock과 테이블 Lock이 있다.

**가. 로우 Lock**

Oracle에서 로우 Lock은 항상 배타적이다. insert, update, delete문이나 select...for update문을 수행한 트랜잭션에 의해 설정되며, 이 트랜잭션이 커밋 또는 롤백할 때까지 다른 트랜잭션은 해당 로우를 변경할 수 없다. Oracle에서 일반 select문에 의해 읽힌 레코드에는 어떤 Lock도 설정되지 않는다. 다른 DBMS처럼 읽기 작업에 대한 공유 Lock을 사용하지 않기 때문에 Oracle에서 읽기와 갱신 작업은 서로 방해하지 않는다.

* 읽으려는 데이터를 다른 트랜잭션이 갱신 중이더라도 기다리지 않는다.
* 갱신하려는 데이터를 다른 트랜잭션이 읽는 중이더라도 기다리지 않는다. (select ... for update 구문으로 읽는 경우는 제외)
* 갱신하려는 데이터를 다른 트랜잭션이 갱신 중이면 기다린다.

Oracle이 공유 Lock을 사용하지 않고도 일관성을 유지할 수 있는 것은 Undo 데이터를 이용한 다중버전 동시성 제어 메커니즘(3절에서 설명?? 없이 레코드의 속성으로서 로우 Lock을 구현했기 때문에 아무리 많은 레코드를 갱신하더라도 절대 Lock Escalation은 발생하지 않는다.

**나. 테이블 Lock**

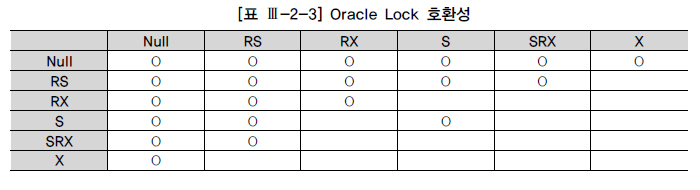
한 트랜잭션이 로우 Lock을 얻는 순간, 해당 테이블에 대한 테이블 Lock도 동시에 얻는다. 그럼으로써 현재 트랜잭션이 갱신 중인 테이블에 대한 호환되지 않는 DDL 오퍼레이션을 방지한다. 테이블 구조를 변경하지 못하도록 막는 것이다. 테이블 Lock 종류로는 아래 5가지가 있다.

* Row Share(RS)
* Row Exclusive(RX)
* Share(S)
* Share Row Exclusive(SRX)
* Exclusive(X)

대표적으로, select ... for update 문을 수행할 때 RS 모드 테이블 Lock을 얻고, insert, update, delete문을 수행할 때 RX 모드 테이블 Lock을 얻는다. DML 로우 Lock을 처음 얻는 순간 묵시적으로 테이블 Lock을 얻지만, 아래 처럼 Lock Table 명령어를 이용해 명시적으로 테이블 Lock을 얻을 수도 있다.

lock table emp in row share mode; lock table emp in row exclusive mode; lock table emp in share mode; lock table emp in share row exclusive mode; lock table emp in exclusive mode;

테이블 Lock끼리의 호환성은 [표 Ⅲ-2-3]과 같다.



‘테이블 Lock’이라고 하면, 테이블 전체에 Lock이 걸린다고 생각하기 쉽다. DML 수행 시 항상 테이블 Lock이 함께 설정된다고 했는데, 만약 이것이 SQL Server의 테이블 레벨 Lock처럼 테이블 전체를 잠그는 기능이라면 다른 트랜잭션이 더는 레코드를 추가하거나 갱신하지 못하도록 막게 될 것이다. 하지만 [표 Ⅲ-2-3]에서 보듯, RX와 RX 간에 호환성이 있으므로 그런 일은 발생하지 않는다. Oracle에서 말하는 테이블 Lock은, Lock을 획득한 선행 트랜잭션이 해당 테이블에서 현재 어떤 작업을 수행 중인지를 알리는 일종의 푯말(Flag)이다. 후행 트랜잭션은 어떤 테이블 Lock이 설정돼 있는지만 보고도 그 테이블로의 진입 여부를 결정할 수 있다.

출처 : <http://www.dbguide.net/db.db?cmd=view&boardUid=148215&boardConfigUid=9&boardIdx=138&boardStep=1>

1. LOCK의 종류

|  |  |
| --- | --- |
| 공유잠금(Shared Lock) | Select |
| 배타적잠금(Exclusive Lock) | Insert , Update , Delete |

2. LOCK 단위

|  |  |
| --- | --- |
| RID | 행 식별자. 하나의 행을 잠글 때 사용 |
| KEY | 인덱스가 있을 때의 행 잠금단위. |
| PAGE | 8KB 데이터 페이지 또는 인덱스 페이지 |
| EXTENT | 인접한 8개의 데이터 페이지 또는 인덱스 페이지 |
| TABLE | 데이터와 인덱스가 포함된 전체 테이블 |
| DB | 데이터베이스 |

3. LOCK 종류

1) SHARED(S)

읽기 동안에만 일어나며 , 다른 S 락에 대해서는 공유하난 X에 대해서는

배타적이다. 데이터를 변경하거나 업데이트하지 않는 작업(읽기전용작업)에

사용한다.

공유잠금(S)를 사용하면 , 여러 트랜잭션이 동시이에 하나의 개체를 읽을 수 있다.

즉 , 공유잠금끼리는 서로 충돌되지 않는다.

그러나 , 공유잠금이 설정되어 있는 동안에는 , 다른 트랜잭션이 데이터를

변경할 수 없다.즉, 공유 잠금은 다른 배타적 잠금을 허용하지 않으며 ,

호환되지 않는다.리소스에 대한 공유 잠금은 다음 경우를 제외하고는 ,

데이터를 읽자마자 해제된다.

예외)

      - 트랜잭션 고립수준을 REPEATABLE 레벨 이상으로 설정

      - LOCK HINT 를 써서 , 해당 공유잠금을 트랜잭션 끝까지 유지한다.

|  |
| --- |
| SELECT \* FROM EMP(HOLDLOCK) |

2) EXCLUSIVE(X)

배타적 잠금이 설정되면 동시에 여러 연결이 한 리소스에 액세스할 수 없게 된다.

오직 하나의 연결만이 해당 리소스를 점유한다. 이 잠금은 수정 시에 , 행단위에

설정 되게 된다.

배타적 잠금은 다른 트랜잭션이 읽거나 수정할 수 없다. 즉  공유잠금(S)에도 적용된

다는 뜻이다. 배타적 잠금은 공유잠금을 포함한 대부분의 락에 대해서 배타적이

고 호환되지 않는다.

3) 업데이트(U)

수정 시에 X락을 걸기전 데이터영역을 읽기 위해 거는 락이다. 해당 리소스에 대한

잠금을 미리 체크하여 데드락을 방지하기 위해서 사용된다. 공유잠금과 호환된다.

업데이트잠금(U)를 사용하면 일반적인 형태의 교착상태가 방지된다.

일반적인 업데이트패턴은 레코드를 읽고 , 리소스(페이지 혹은 행)에 대한 공유(S)

잠금을 얻은 다음 , 행을 수정하는 트랜잭션으로 구성되는데 , 행을 수정할 때는 먼

저 배타적 잠금으로 잠금을 변환해야한다.

두 트랜잭션에서 리소스에 대해 공유 모드 잠금을 얻은 다음 데이터를 동시에 업데

이트하려고 하면 한 트랜잭션이 배타적 잠금으로 잠금을 변환하려 할 것이다. 이 때,

한 트랜잭션의 배타적 잠금은 다른 트랜잭션의 공유 모드 잠금과 호환되지 않으므

로, 공유 모드를 단독 모드로 변환할 때는 잠금 대기가 발생하게 된다. 그리고 두 번

째 트랜잭션도 해당 업데이트에 대해 배타적 잠금을 얻으려고 할 것이다. 이 경우 두

트랜잭션 모두 배타적 잠금으로 변환 중이고 각각 상대 트랜잭션이 공유 모드 잠금

을 해제하기를 기다리므로 교착 상태가 발생하는 것이다.

이러한 교착 상태를 방지하기 위해서 SQL Server는 업데이트 잠금을 사용한다. 이

업데이트 잠금은 한 번에 한 트랜잭션만 리소스에 대한 업데이트 잠금을 얻을 수 있

게 하기 때문에 교착상태가 방지될 수 있다. 트랜잭션이 리소스를 수정하면 업데이

트 잠금이 배타적 잠금으로 변환되고 그렇지 않으면 잠금이 공유 모드 잠금으로 변

환된다. 그러나 업데이트 잠금을 사용하더라도 교착상태를 완전히 방지할 수는 없

다. 그것은 모든 RDBMS 에서도 마찬가지이다.

4)  INTENT(I)

공유 잠금 OR 배타적 잠금이 걸린 특정 데이터영역에 표시하는 알림간판이라고 생

각하면 된다. 의도적 잠금이라고 하며 다른 INTENT 잠금 들과 호환된다.

의도적 잠금은 SQL Server가 리소스에 대해 공유 잠금 또는 배타적 잠금을 얻으려

할 때 같이 발생한다.

예를 들어, 의도적 공유 잠금(IS)을 테이블 수준에서 설정하려고 한다면, 이것은 해

당 트랜잭션이 해당 테이블의 페이지 또는 행에 대해 공유 잠금을 설정하려고 한다

는 것을 의미한다. 이렇게 테이블 수준에서 의도적 잠금을 걸면, 이후에 다른 트랜잭

션이 해당 페이지를 포함하는 테이블에 대해 배타적 잠금을 얻을 수 없게 된다.

SQL Server는 테이블 수준에서만 의도적 잠금을 확인하여, 트랜잭션이 해당 테이블

에 대해 잠금을 얻을 수 있는지 확인하므로 의도적 잠금을 사용하면 성능이 향상된

다. 왜냐하면, 직접 테이블의 모든 행 또는 페이지 잠금을 확인할 필요가 없기 때문

이다.

- 의도적 배타적 잠금(IX)

잠금을 걸려는 트랜잭션이 각 리소스계층(테이블,페이지,행등...)에 대해 X 잠금을

설정하여 계층의 아래쪽에 있는 일부 리소스를 수정하려 하는 것을 말한다. IX는 IS

의 상위 집합이 된다.

- 공유 및 의도적 배타적 잠금(SIX)

잠금을 걸려는 트랜잭션이 각 리소스계층에 대해 IX 잠금을 설정하여 계층의 아래쪽

에 있는 모든 리소스에 대해서는 읽기 작업을 하고, 일부 리소스에 대해서는 수정작

업을 하려고 하는 것을 말한다.

최상위 수준 리소스에서는 동시 IS 잠금이 허용된다.

예를 들어, 테이블에 대한 SIX 잠금은 테이블에 대해 SIX 잠금을 설정하여 동시 IS

잠금을 허용하고, 수정 중인 페이지에 IX 잠금을 설정하고 수정된 행에 대해서는 X

잠금을 설정한다. 각 리소스 당 한 번에 하나의 SIX 잠금을 설정할 수 있으므로 다른

트랜잭션이 테이블 수준에서 IS 잠금을 얻어 계층 아래쪽에 있는 리소스를 읽을 수

있게 된다. 그라나, 그 상황에서 다른 트랜잭션이 리소스를 수정할 수는 없게 된다

5) SCHEMA((Sch)

- Sch-M (Schema Manipulation)

스키마를 변경하는 작업 즉, DDL 문 실행 시에 SQL Server 가 SCHEMA 자체에 대

해서 건다. 이 잠금은 모든 잠금에 대해서 배타적이며, 어떤 작업도 허락하지 않는

다. 그도 그럴것이 데이터베이스의 구조를 변경하는 누군가 그 데이터베이스 내의

데이터를 엑세스해서는 안될 것이기 때문이다. 이것을 스키마 변경 잠금이라고 한

다.

- Sch-S (Schema Stability)

쿼리문 컴파일 시에만 발생한다. S or X 와 호환된다. 스키마 안정성(Sch-S) 잠금은

의도적 잠금 등 다른 트랜잭션 잠금을 차단하지 않는다. 따라서 쿼리가 컴파일되는

동안에도, 테이블에 대한 의도적 잠금을 포함하여 다른 트랜잭션을 계속 실행할 수

있다. 그러나 해당 테이블에서의 DDL 작업은 수행할 수는 없다.

4. TRANSACTION ISOLATION LEVEL (잠금수준)

잠금은 그 레벨에 대해서 SQL-92 표준을 가지고 있다. 이것은 SQL Server 에만 적용되  
는 것이 아니고, 모든 RDBMS에서 자신들의 제품에 맞도록 적용하는 기준이다.

잠금수준의 표준은 네 가지가 있으며 SQL Server 2000 은 네 가지 모두를 지원한다.

오라클은 그 중 두 수준만을 지원한다.

이러한 잠금 수준은 트랜잭션 고립수준, 혹은 트랜잭션 격리수준 혹은

TRANSACTION ISOLATION LEVEL 이라고 부른다.

다음은 잠금 수준에 대한 내용을 설명한 것인데, 그 수준은 다음과 같이 레벨을 가진다.

오른쪽으로 갈수록 잠금의 정도가 심함을 나타낸다.

READ UNCOMMITTED < READ COMMITTED < REPEATABLE READ <

SERIALIZABLE

실제로 , 사용자가 제어할 일은 그렇게 많지 않다.

1) READ UNCOMMITTED

READ UNCOMMITTED 수준은 잠금의 4가지 레벨 중 가장 낮은, 즉 가장 느슨한 단

계의 잠금 수준이다. 이 수준은 특정 트랜잭션에서 배타적 잠금을 설정하고 있어도,

해당 데이터를 볼 수는 있게 한다.

물론 해당 리소스에 대한 수정작업은 모든 잠금 수준에서 불가능하다.

READ UNCOMMITTED는 배타적 잠금이 설정된 리소스(데이터)라고 하더라도, 볼

수(SELECT) 있게 하는, 잠금 중 가장 느슨한 단계이다. 배타적 잠금을 공유 잠금과

호환되게 하는 것이라고 보면 되겠다.

2) READ COMMITTED

READ COMMITTED 는 말그대로, COMMIT 된 트랜잭션만을 읽자는 것이다.

이것은 잠금을 건 트랜잭션에서의 입장이 아니다. 잠금을 건 트랜잭션에 대해서

SELECT 하려는 또 다른 트랜잭션에서의 입장이다. 현재는 해당 트랜잭션에 대해

잠금이 걸려 있으면 해당 잠금의 종류에 따라 볼 수도 있고 보지 못할 수도 있다.

공유 잠금에 대해서는 볼 수 있으며, 배타적 잠금이 걸려 있으면 볼 수 없는것이

현재 상태이다. SELECT 한다는 것은 공유 잠금이고, 공유 잠금은 공유 잠금과는

호환되지만 배타적 잠금에 대해서는 호환되지 않기 때문이다.

이것이 READ COMMITTED 의 뜻이다. 배타적 잠금에 대해서 공유 잠금을 허용하지

않겠다는 뜻이다.

3) REPEATABLE READ

읽는 데이터의 일관성에 관한 얘기이다. 잠금이 설정되어 있을 때, 트랜잭션 시작 전

과 시작 후의 데이터를 동일하게 읽게 함으로써, 읽는 데이터의 일관성을 보장하는

방법이 된다.

REPEATABLE READ 수준은 트랜잭션 진행 중인 개체에 엑세스해서 데이터를 읽으

려고 할 때, 읽고자 하는 데이터에 일어나고 있는 변경이 성공될지 취소될지 불확실

한 상태에서, 확실하게 트랜잭션이 완료되기 전까지는, 변경전의 데이터를 읽게 함

으로써 데이터의 일관성을 보장해 주는 기법이다.

4) SERIALIZABLE

데이터의 범위에 대한 입력작업에 관한 이야기이다.

한 세션에서 특정 범위의 데이터를 읽고 있는 경우에 , 다른 세션에서 읽고 있는  특

정 범위내에 포함되는 데이터를 입력하는 경우에 입력을 하지 못하도록 한다.

출처: <http://battleracoon.tistory.com/2> [개발하는 너구리]

**MYSQL LOCK 정리**

**http://casek.tistory.com/entry/Real-MySQL-12%EC%9E%A5-%EC%BF%BC%EB%A6%AC-%EC%A2%85%EB%A5%98%EB%B3%84-%EC%9E%A0%EA%B8%88?category=574308**