6.4 Lock과 트랜잭션 동시성 제어

# 6.4.1 오라클 Lock

* 래치 – SGA에 공유된 자료구조 보호를 위해 사용
* 버퍼 Lock – 버퍼 블록에 대한 액세스 직렬화를 하기 위해 사용
* 라이브러리 캐시 Lock/Pin – 라이브러리 캐시에 공유된 SQL 커서와 PL/SQL 프로그램 보호를 위해 사용
* **DML Lock** – 다중 트랜잭션이 동시에 액세스 하는 사용자 데이터 무결성을 보호

ex) – 테이블 Lock, 로우 Lock

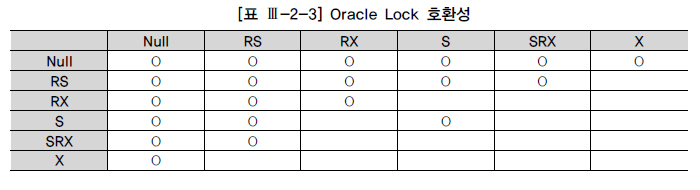
# DML 로우 Lock

* 두개의 동시 트랜잭션이 같은 로우를 변경하는 것을 막는다.
  + INSERT 로우 Lock 경합 – **Unique 인덱스가 있을 때** 발생  
    **블로킹 발생시**, 선행 트랜잭션이 **커밋**시 후행 트랜잭션 INSERT 실패   
    선행 트랜잭션 **롤백**시 후행 트랜잭션 성공  
    ( Unique 인덱스 없는 경우 Lock 경합 발생하지 않는다. )
  + MVCC 모델을 사용하는 오라클은 SELECT에 로우 Lock을 사용하지 않는다.
* 오라클에서 DML과 SELCT는 서로 진행을 방해하지 않음

( MVCC 모델을 사용하지 않는 DBMS는 SELECT에 공유 LOCK 사용  
= 두 트랜잭션이 같이 LOCK 설정 가능 )

* Lock 성능 저하 방지
  + 커밋 시점 조절
  + SQL 튜닝 = Lock 튜닝

# DML 테이블 Lock = TM Lock

* 테이블 Lock이 로우 Lock보다 먼저 설정
* 테이블 Lock 모드
  + RS : row share = SS : sub share
  + RX : row exclusive = SX : sub exclusive
  + S : share
  + SRX : share row exclusive = SSX : share/sub exclusive
  + X : exclusive[[1]](#footnote-2)
* 자신이 테이블에서 현재 어떤 작업중인지 알리는 Flag
* 호환이 가능하면 같은 로우 갱신시에만 로우 Lock 경합 발생
* Lock 모드에 따라서 후행 트랜잭션이 수행될 수 있는 범위가 달라짐
* 후행 트랜잭션의 선택지
  + Lock 해제시까지 대기
  + 일정 시간만 대기하다 포기
  + 기다리지 않고 포기

# Lock을 푸는 열쇠, 커밋

* 블로킹(Blocking) : 선행 트랜잭션의 Lock때문에 후행 트랜잭션이 작업을 하지 못하고 멈춰있는 상태
* 해결 방법 : 커밋, 롤백
* 교착상태(Deadlock) : 두 트랜잭션이 각각 특정 리소스에 Lock을 설정한 상태에서 맞은 편 트랜잭션이 또 Lock 설정을 진행하려고 하는 상황
* 해결 방법 : 둘 중 하나가 양보
* 트랜잭션이 길면, 롤백시 시간이 많이 걸린다.
* 동시성이 저하되지 않도록 적절한 시점에 커밋해야 한다.
* 커밋 명령 옵션
  + WAIT : LGWR의 파일 기록 완료 메시지를 받을 때까지 대기 (동기식)
  + NOWAIT : LGWR의 완료 메시지를 기다리지 않고 다음 트랜잭션 진행 (비동기식)
  + IMMEDIATE(Defualt) : 커밋 명령때마다 LGWR이 로그 버퍼를 파일에 기록
  + BATCH : 세션 내부에 트랜잭션 데이터를 일정량 버퍼링했다가 일괄 처리

# 6.4.2 트랜잭션 동시성 제어

* 동시에 실행되는 트랜잭션 수를 최대화 하면서, 입력, 수정, 삭제, 검색시 데이터 무결성을 유지하기 위해 노력하는 것

# 비관적 동시성 제어

* 사용자들이 같은 데이터를 동시에 수정할 것이라고 가정
* 한 사용자가 데이터 읽는 시점에서 Lock을 걸고 작업완료 전까지 유지   
  = 다른 사용자는 데이터 동시 수정 불가  
  = 잘못 사용시 동시성이 나빠짐

|  |
| --- |
| select 적립포인트, 방문횟수, 최근방문일시, 구매실적 from 고객  where 고객변호 = :cust\_num for update; |

* 고객 레코드에 Lock 설정 = 데이터 갱신 이상 문제 방지
* WAIT, NOWAIT 옵션을 사용하여 동시성 증가시키자

#낙관적 동시성 제어

* 사용자들이 같은 데이터를 동시에 수정하지 않을 것이라고 가정
* 데이터 읽을 때 Lock설정을 하지 않는다.
* 데이터 수정 전에 다른 사용자에 의해 데이터가 변경되었는지 검사해야 함

|  |
| --- |
| selct 적립포인트, 방문횟수, 최근방문일시, 구매실적, 변경일시  into :a, :b, :c, :d, :mod\_dt  from 고객  where 고객번호 = :cust\_num;  --새로운 적립포인트 계산  update 고객 set 적립포인트 = :적립포인트, 변경일시 = SYSDATE  where 고객변호 = :cust\_num  and 변경일시 = :mod\_dt; ->최종 변경일시가 앞서 읽은 값과 같은지 비교  if sql%rowcount = 0 then  alter(‘다른 사용자에 의해 변경되었습니다.’);  end if; |

* 조건절에 최종변경일시 관리 레코드를 넣어 해당 레코드의 갱신여부 판단
* 낙관적 동시성 제어에서도 nowait 옵션을 사용해서 Lock을 기다리지 않게 구현 가능
* 동시성 제어를 제대로 구현하지 않으면 고객 클레임 발생 가능성 증가

# 데이터 품질과 동시성 향상을 위한 제언

* 데이터 품질 > 성능
* 다중 트랜잭션이 존재하는 데이터 베이스 환경 -> 공유 자원 액세스 직렬화 필수   
  = 멀티 스레드 프로그래밍 -> synchronized 키워드
* 데이터 변경 목적의 읽기 = Lock 필수
* FOR UPDATE, WAIT, NOWAIT을 적절히 활용
* 동시성 향상 -> SQL 튜닝

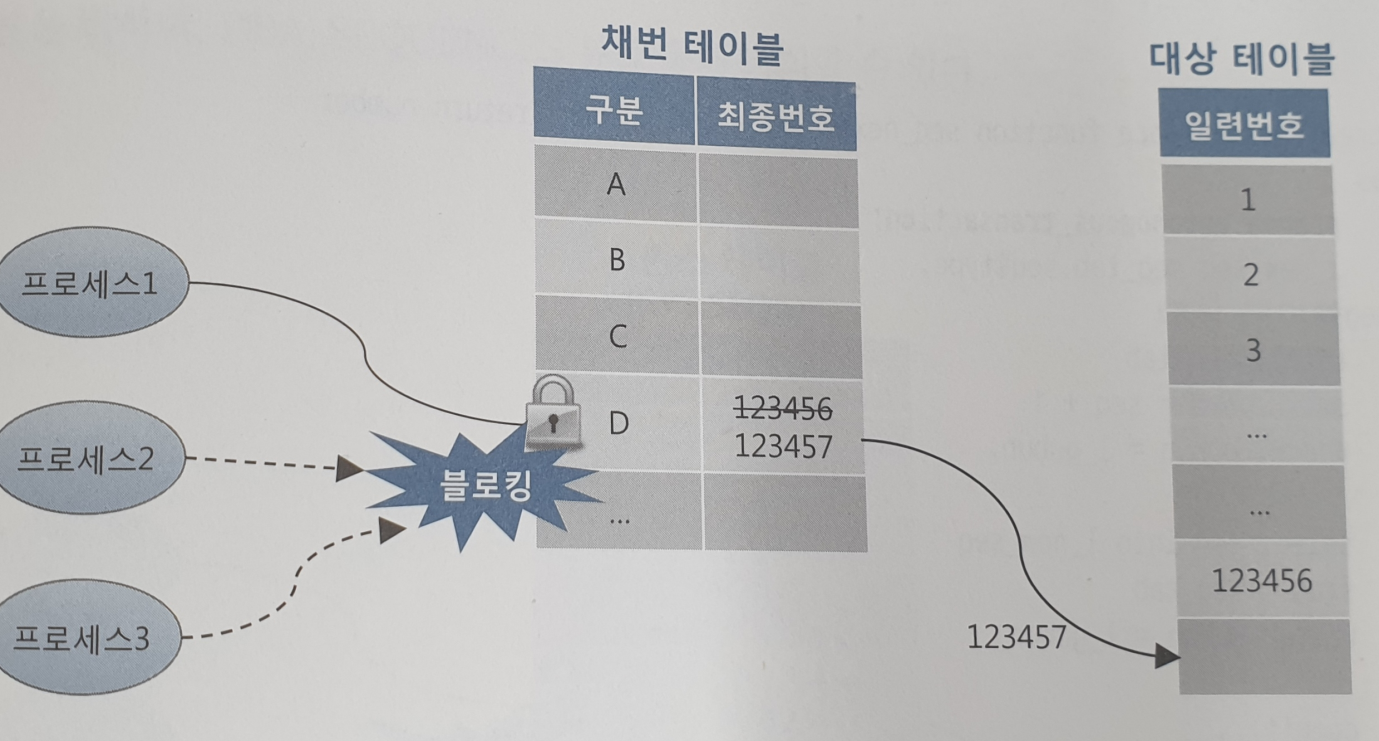
# 6.4.3 채번 방식에 따른 INSERT 성능 비교

* 채번 방식에 따라서 INSERT 성능 차이가 매우 크다

ex) 채번 테이블, 시퀀스 오브젝트, MAX +1 조회

\*용어  
복합컬럼 – 『상담원ID + 상담일자 + 상담순번』 일경우,   
순번 이외의 컬럼을 ‘구분 속성’이라고 부르자

# 채번 테이블

* **채번 레코드용 별도 테이블 관리**
* 장점
  + 범용성이 좋다.
  + INSERT 중복 레코드 발생 대비 예외 처리에 신경쓰지 않아도 된다.
  + INSERT 과정에 결번 방지 가능
  + PK가 복합컬럼일 때도 사용가능
* 단점
  + 성능이 안 좋다.
  + 채번 레코드 변경을 위한 **로우 Lock 경합**
* 동시 INSERT가 많으면 채번 테이블 블록에서도 경합이 발생   
  = 서로 다른 레코드를 변경하는 프로세스끼리도 경합  
  = Lock 경합 발생 가능성이 높다.

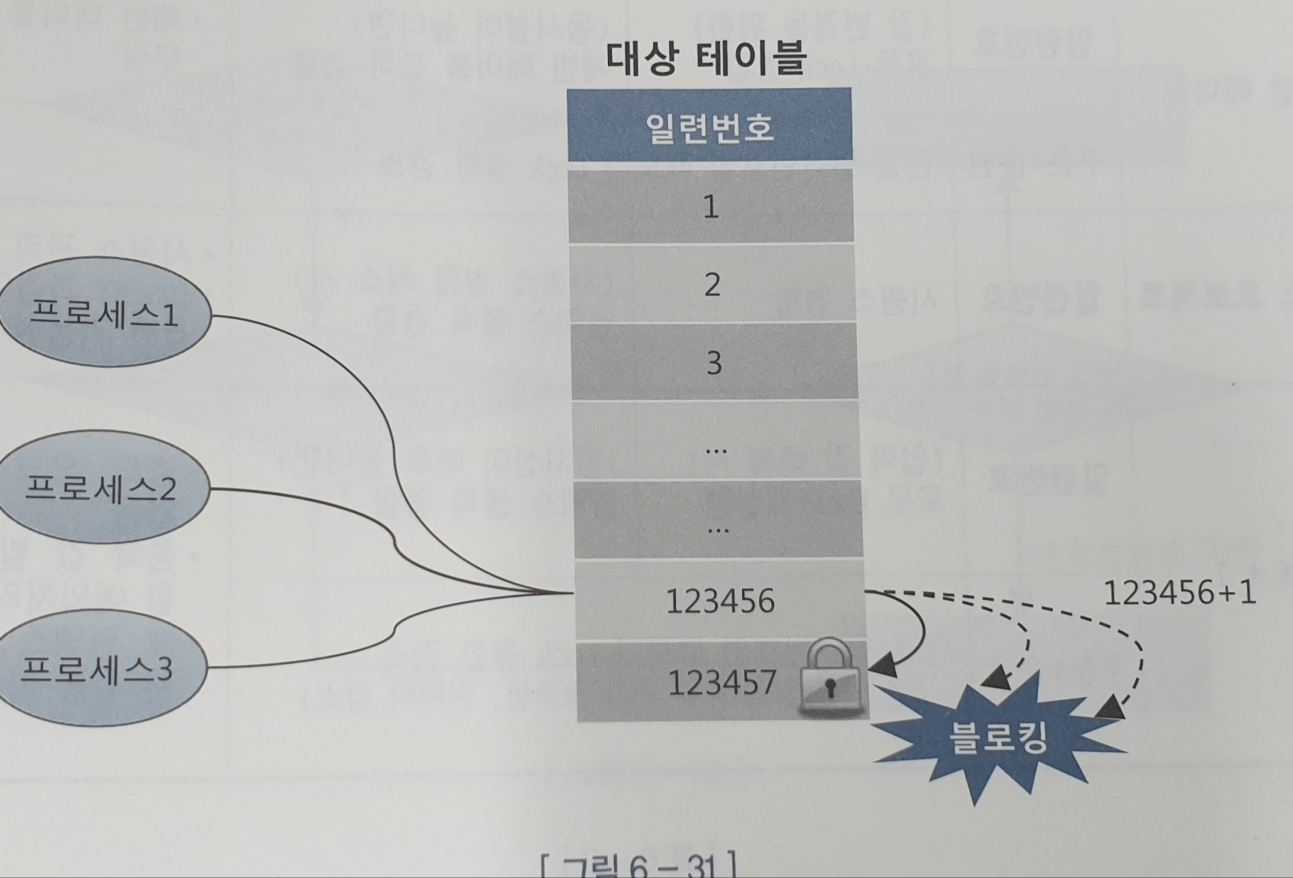
# 시퀀스 오브젝트

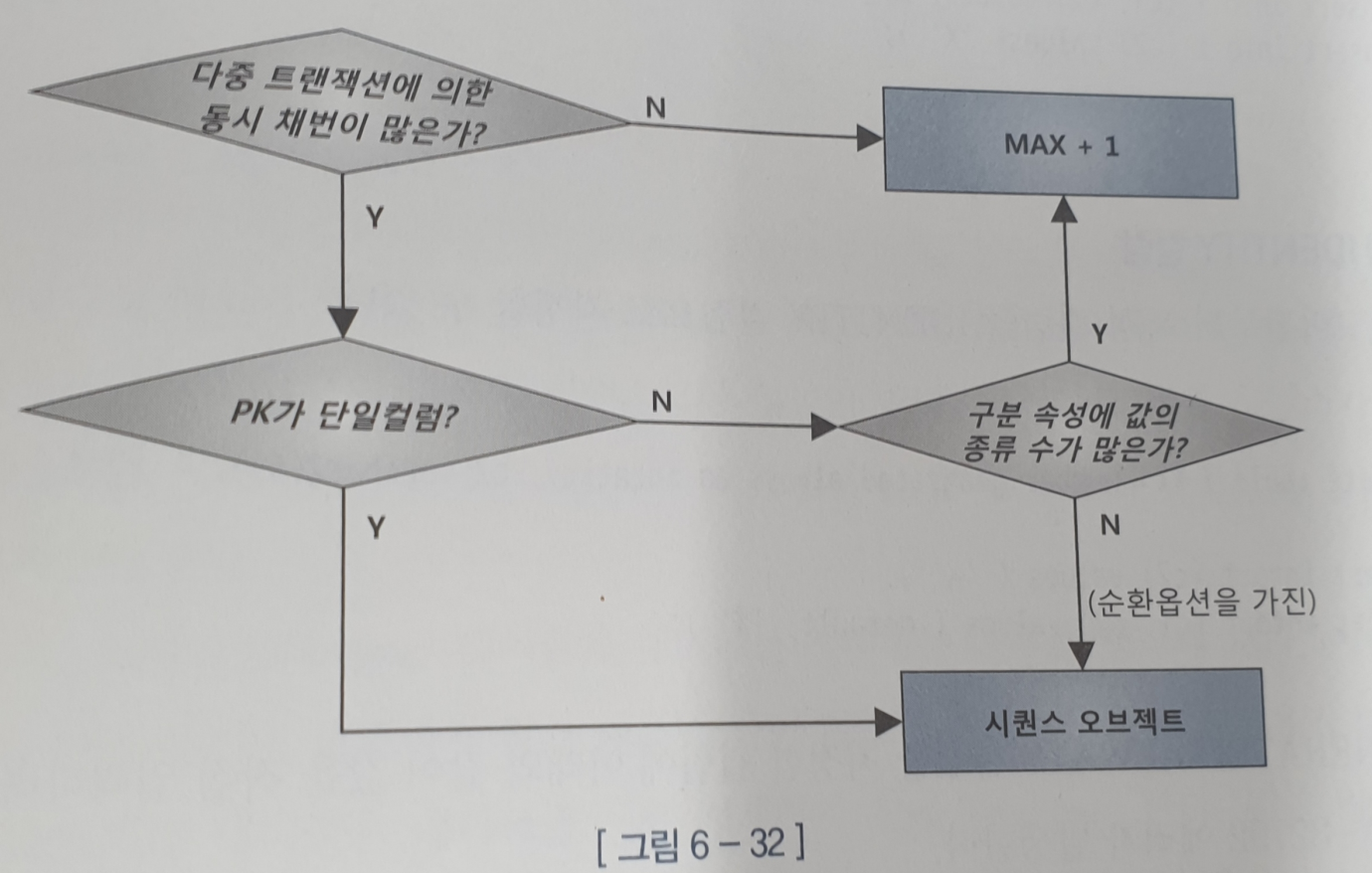
* 오라클 내부에서 관리하는 채번 테이블  
  = SYS.SEQ$테이블 = DBA\_SEQUENCES 뷰를 통해 조회 가능
* 장점
  + 성능이 빠르다.
  + INSERT 중복 레코드 발생 대비 예외 처리에 신경쓰지 않아도 된다.
* 단점
  + **시퀀스 Lock 경합**에 의한 성능 이슈
* 캐시 사이즈를 적절히 설정할 경우 가장 빠른 성능 제공
  + PK가 단일컬럼일 때만 사용 가능
  + **신규 데이터 입력 과정에서 결번 발생 가능성**
* 시퀀스 Lock
  + 로우 캐시 Lock : 로우 캐시의 정보를 읽고 쓸 때 액세스를 직렬화 하기 위해 사용 되는 Lock  
    ( 로우 캐시 – 오라클 딕셔너리 정보를 빨리 읽고 쓰기 위한 캐시 )
  + 시퀀스 캐시 Lock : 시퀀스 캐시에서 값을 얻을 때 액세스 직렬화를 하기 위해 Lock 필요
  + SV Lock : 자원 공유시, RAC 환경에서 ORDER 옵션 사용시 시퀀스 캐시에 대한 액세스를 직렬화하기 위해 사용  
    ( RAC – 여러대의 컴퓨터가 동시에 한대의 DBMS 서버에 접속하여 데이터 이용   
    네트워크를 통해 시퀀스 캐시를 주고 받으며 공유한다. )

# MAX+1 조회

* 대상 테이블의 최종 일련번호 조회 후, +1하여 INSERT하는 방식

|  |
| --- |
| insert into 상품거래(거래일련번호, 계좌번호, 거래일시, 상품코드, 거래가격, 거래수량)  values( (select max(거래일련번호) +1 from 상품거래)  , :acnt\_no, sysdate, :prod\_cd, :trd\_price, :trd\_qty ); |

* 장점
  + 시퀀스, 채번 테이블 관리 부담이 없음
  + 동시 트랜잭션 충돌이 적다
  + 성능이 빠르다
  + PK가 복합 컬럼인 경우에도 사용 가능
* 단점
  + 레코드 중복에 대비한 세밀한 예외처리가 필요함
  + **입력 값 중복시 로우 Lock 경합**

# Lock 경합 요소를 고려한 채번 방식 선택 기준 

# 시퀀스보다 좋은 솔루션

* 구분속성과 함께 뒤쪽에 순번 대신 입력일시로 PK 구조를 설계
* 채번 또는 INSERT 과정에서 생기는 Lock 이슈 해결 가능
* 채번과정을 생략하고, SYSDATE 또는 SYSTIMESTAMP 함수만 호출하면 되기 때문에 빠르고 간편하다.

#인덱스 블록 경합

* INSERT 성능이 빠르면 인덱스 경합 발생
* 채번 과정 생략 or MAX+1 방식에서 인덱스 블록 경합 발생
* Right Growing 인덱스에서 흔히 발생  
  (Right Growing 인덱스 – 항상 맨 우측 블록에만 데이터가 입력되는 것)  
  -> 입력하는 값이 달라도 같은 블록을 갱신하려는 프로세스 간 버퍼 Lock 경합이 발생
* 구분 속성을 앞에 추가
* 인덱스를 해시 파티셔닝

1. ( exclusive : lock이 걸린 자원의 공유를 허용하지 않는다.  
    share : 자원에 대해 수행되는 명령의 유형에 따라 lock된 자원 공유를 허용 ) [↑](#footnote-ref-2)