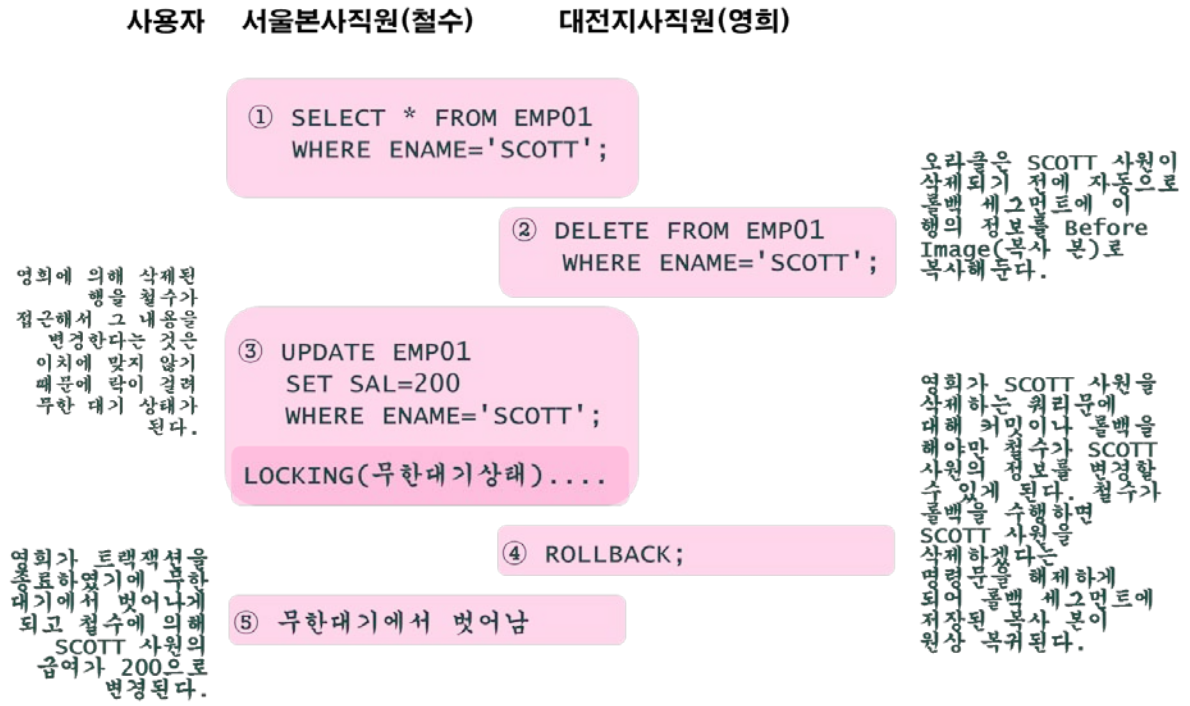
12 데이터 읽기 일관성과 락

12.1. 데이터 읽기의 일관성과 락

- 오라클이 데이터 읽기의 일관성을 제공해 준다는 것을 증명을 하기 위해서 우선 다음과 같은 가정을 하겠다.
- 오라클 서버가 서울 본사에 설치되어 있고 이 데이터베이스를 서울 본사 직원과 대전 지사 직원이 공용하고 있다고 하자.
- 어느 날 서울 본사 직원과 대전 지사 직원이 동일한 테이블을 같은 시간에 접근해서 사용할 경우 어떻게 일이 일어날 수 있는지 살펴보도록 하자.



[실습] 데이터 읽기의 일관성과 락

```
-- 1. 도스창(A)을 띄운 후 SCOTT 계정으로 접속한 뒤, EMP 테이블과 구조와 내용이 동일한 EMP01 테이블을 만든다.
SET LINESIZE 100
SET PAGESIZE 30
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

-- 2. SCOTT 사원이 존재하는지 확인한다.
SELECT * FROM EMP01
WHERE ENAME='SCOTT';

-- 3. 또 다른 도스창(B)을 띄운 후 SCOTT 계정으로 접속한다. 도스창이 2개 뜬 상태가 된다. 나중에 띄운 도스창에서
SCOTT 사원을 삭제한다.
DELETE FROM EMP01
WHERE ENAME='SCOTT';
```

```
-- 4. 먼저 띄운 도스창(A)에서 SCOTT 사원의 정보를 변경한다. 도스창(A)에 SQL 프롬프트가 나타나지 못하여 무한 대기 상태로 빠지게 된다.
UPDATE EMP01
SET SAL=200
WHERE ENAME='SCOTT';

-- 5. 도스창(B)는 대기 상태가 아니기에 롤백을 수행한다.
ROLLBACK;

-- 6. 도스창(A)의 UPDATE 작업이 성공적으로 수행된다. 명령어도 완벽히 끝내려면 커밋이나 롤백을 수행해야 한다.
ROLLBACK;
```

12.2. 데드 락

■ 데드락이란 한마디로 무한 교착상태이다. 하나의 상자에 들어 있는 물건을 서로 가지려고 두 사람이 자기만의 자물쇠로 상자를 잠궈 버린 경우라고 할 수 있다. 이렇게 되면 서로 상자의 물건을 어느 누구도 가질 수 없게 된다.

철수(사용자: 서울본사직원)	영희(사용자: 대전지사직원)
1. UPDATE EMP01 SET SAL=100 WHERE ENAME='SCOTT';	
	2. UPDATE EMP01 SET SAL=20 WHERE ENAME='SMITH';
3. UPDATE EMP01 SET SAL=300 WHERE ENAME='SMITH';	
	4. UPDATE EMP01 SET SAL=400 WHERE ENAME='SCOTT';
LOCKING(무한 대기 상태) ...	
	LOCKING(무한 대기 상태) ...

[실습] 데드 락

```
-- 1. 왼쪽 사용자가 SCOTT 사원의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=100
WHERE ENAME='SCOTT';

-- 2. 오른쪽 사용자가 SMITH 사원의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=20
WHERE ENAME='SMITH';

-- 3. 왼쪽 사용자가 SMITH 사원의 정보를 갱신하려고 시도한다.
UPDATE EMP01 SET SAL=300
WHERE ENAME='SMITH';

-- 4. 왼쪽 사용자가 대기 상태인데 오른쪽 사용자가 커밋이나 롤백을 하지 않고 왼쪽 사용자가 사용 중인 SCOTT의 정보를 변경하려고 시도한다.
UPDATE EMP01 SET SAL=400
WHERE ENAME='SCOTT';

-- 5. 오른쪽 사용자가 ROLLBACK을 입력하여 트랜잭션을 마무리 짓고,
```

12.3. SET UNUSED

- DDL 작업을 하는 동안에도 락이 발생하게 되는데, 이럴경우에 SET UNUSED라는 키워드를 사용하여 해결한다.
- SET UNUSED는 해당 컬럼을 실제로 삭제하는 것이 아니라 삭제된다는 표시만 하는 논리적 삭제만 일어나게 하므로 실제 수행 시간은 극히 짧다. 따라서 락이 걸리는 시간도 짧으므로 다른 사용자의 사용 제한 시간이 상대적으로 짧다.

```
DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP;

ALTER TABLE EMP02
SET UNUSED (JOB);
SELECT * FROM EMP02;  -- 실제로 JOB 컬럼이 존재하지만 논리적으로 사용을 제한하기 위해서 UNUSED 속성을 지정함.
```

12.4. DDL 명령의 롤백

- DDL은 자동 커밋이 일어나므로 이전 상태로 되돌리기 위해서 롤백할 수 없다.
- DDL 작업을 수행하기 전에 원본 테이블을 복사해 놓고 롤백이 필요할 경우에 복사본을 원본 테이블로 대체한다.

```
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;  -- 원본 테이블(EMP01)

DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP01;  -- DDL 작업을 수행하기 전에 원본 테이블의 복사본(EMP02)을 생성한다.

ALTER TABLE EMP01
DROP COLUMN JOB;  -- 원본 테이블에서 컬럼을 삭제한다.

SELECT * FROM EMP01;
SELECT * FROM EMP02;

DROP TABLE EMP01;
RENAME EMP02 TO EMP01;  -- 원본 테이블을 삭제하고 복사본을 원본 테이블명으로 변경한다. (롤백과 같은 효과)

SELECT * FROM EMP01;
```

12.5. TRUNCATE와 DELETE의 차이

- TRUNCATE
  - 모든 행을 삭제한다.
  - DDL 명령어로 자동으로 커밋이 발생한다.
- DELETE
  - WHERE 절을 추가하여 조건에 만족하는 행만 삭제할 수 있다.
  - DML 명령어로 롤백이 가능하다는 장점이 있지만 롤백을 위해서 무수히 많은 BEFORE IMAGE가 생성되어야 하므로 삭제하는 속도도 늦고 이를 위한 자원이 마련되어야 한다.

```
DROP TABLE EMP01;
CREATE TABLE EMP01
AS
SELECT * FROM EMP;
SELECT * FROM EMP01;

TRUNCATE TABLE EMP01;  -- DDL 명령어로서 자동 커밋이 발생하기 때문에 이전으로 되돌릴 수 없다.
SELECT * FROM EMP01;
ROLLBACK;
SELECT * FROM EMP01;

DROP TABLE EMP02;
CREATE TABLE EMP02
AS
SELECT * FROM EMP;
SELECT * FROM EMP02;

DELETE FROM EMP02;  -- DML 명령어이기 때문에 롤백을 수행하면 삭제 이전으로 되돌릴 수 있다.
SELECT * FROM EMP02;
ROLLBACK;
SELECT * FROM EMP02;

DROP TABLE EMP03;
CREATE TABLE EMP03
AS
SELECT * FROM EMP;
SELECT * FROM EMP03;

DELETE FROM EMP03
WHERE DEPTNO>=10 OR DEPTNO=20;
SELECT * FROM EMP03;
COMMIT;   -- 커밋 후에는 트랜잭션이 종료된 것이므로 롤백 이미지가 제거된다.
ROLLBACK; -- 따라서 삭제 이전 상태로 돌아갈 수 없다.
SELECT * FROM EMP03;
```