Database

(데이터베이스)

김민수

TCL
(Transaction Control Language)

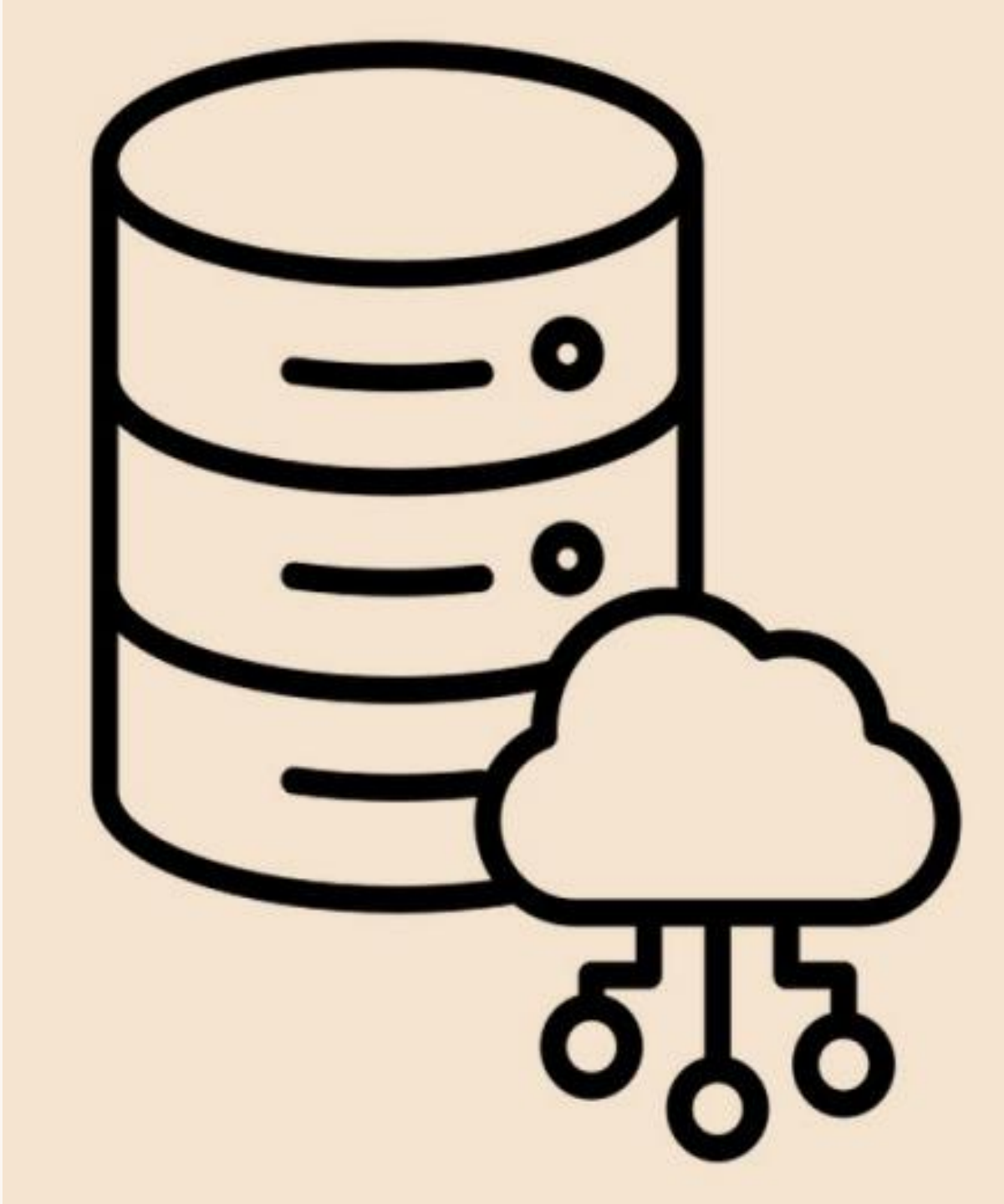
1 TCL 이란?

2 트랜잭션 이해하기

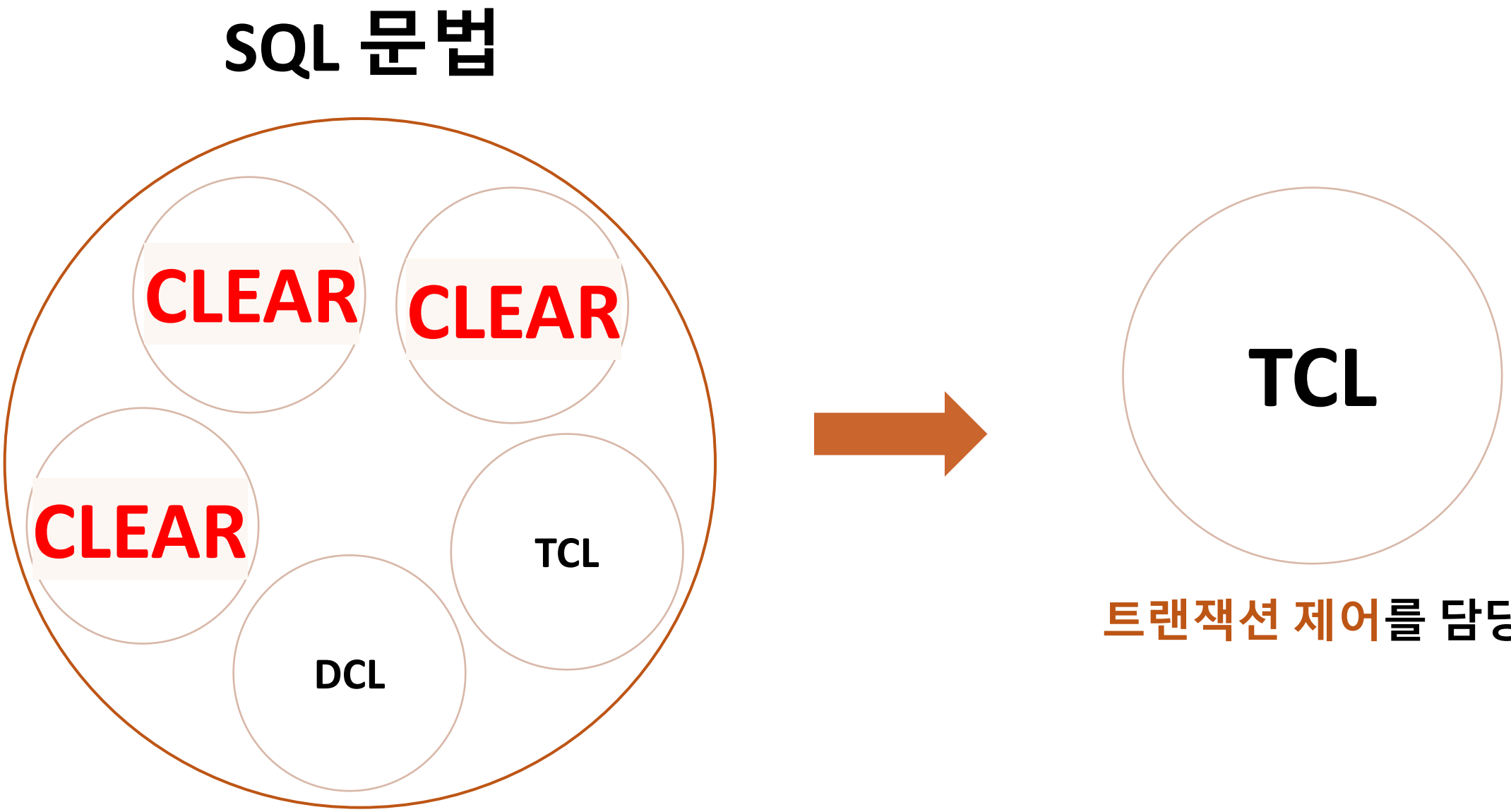
3 COMMIT / ROLLBACK

4 SAVEPOINT

5 LOCK의 개념



TCL 이란?



TCL 이란?

TCL (Transaction Control Language)
: 트랜잭션 제어를 담당

→ 그럼 여기서 트랜잭션(Transaction)이란??

"쪼갤 수 없는 업무 처리의 최소 단위"

데이터베이스에서는 상태를 변화시키기 위해 수행하는 작업의 단위

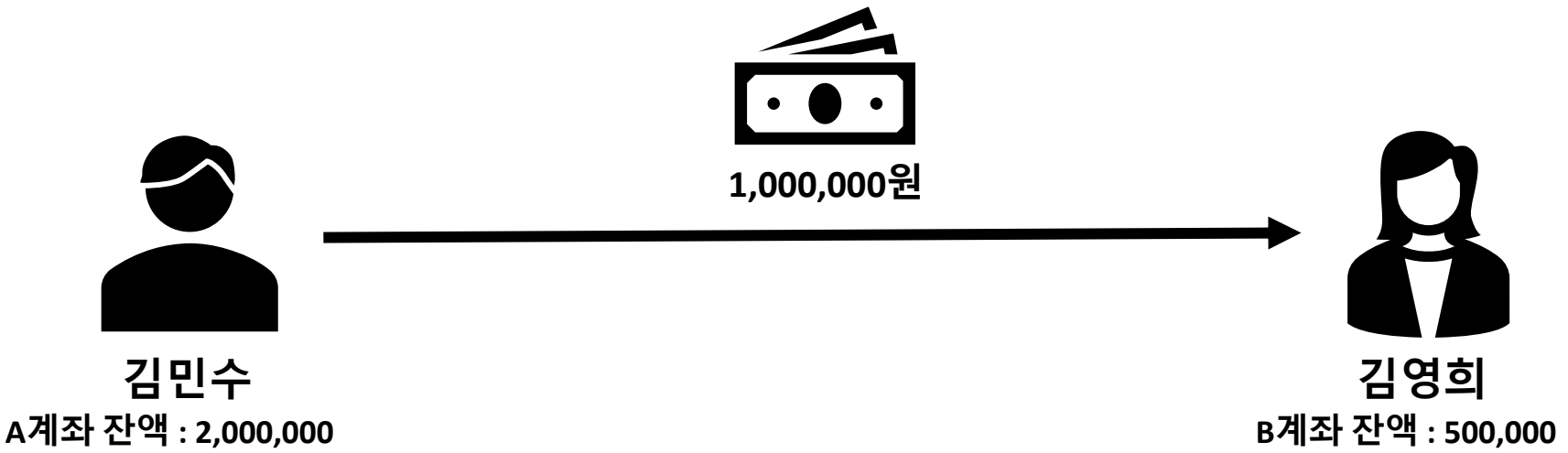
실제로 은행에서 트랜잭션 수행하는 과정이 어떻게 이루어 지는지

예시로 확인해보자!!!

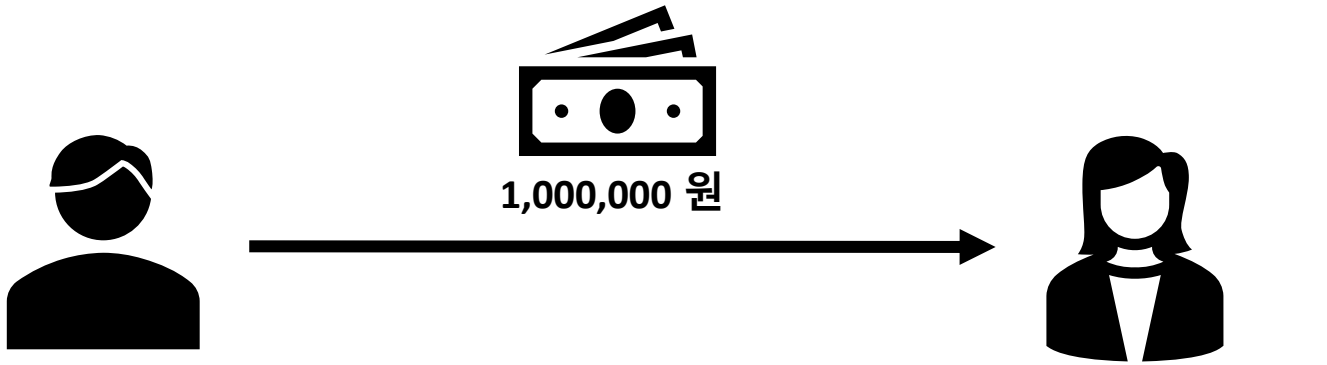
트랜잭션 이해하기

김민수(A계좌)가 김영희(B계좌)에게 1,000,000원을 송금하려고 한다.

어떤 일련의 과정(트랜잭션)을 거쳐 송금 업무가 이루어질까?



트랜잭션 이해하기



계좌정보 테이블

계좌번호	이름	잔액
A계좌	김민수	2000000
B계좌	김영희	500000
…	…	…

1. 김민수라는 사람의 A계좌가 존재하는지 확인

```
SELECT 이름 FROM 계좌정보 WHERE 이름 = '김민수' AND 계좌번호 = 'A계좌' ;
```

2. 김영희라는 사람의 B계좌가 존재하는지 확인

```
SELECT 이름 FROM 계좌정보 WHERE 이름 = '김영희' AND 계좌번호 = 'B계좌' ;
```

3. 김민수의 A계좌에 잔액이 일백만원 이상인지 확인

```
SELECT 잔액 FROM 계좌정보 WHERE 계좌번호 = 'A계좌' AND 잔액 >= 1,000,000 ;
```

4. 김민수의 A계좌 잔액에서 일백만원을 차감

```
UPDATE 계좌정보 SET 잔액 = 잔액 -1,000,000 WHERE 계좌번호 = 'A계좌' ;
```

5. 김영희의 B계좌 잔액에서 일백만원을 추가

```
UPDATE 계좌정보 SET 잔액 = 잔액+1,000,000 WHERE 계좌번호 = 'B계좌' ;
```

6. 송금 업무가 완료

```
COMMIT;
```

COMMIT / ROLLBACK

계좌정보 테이블

계좌번호	이름	잔액
A계좌	김민수	1000000
B계좌	김영희	1500000
…	…	…

송금을 위한 트랜잭션이 정상 처리되면

데이터를 영구 반영하기 위해 COMMIT

T1 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김민수' AND 계좌번호 = 'A계좌' ;

T2 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김영희' AND 계좌번호 = 'B계좌' ;

T3 -> SELECT 잔액 FROM 계좌정보 WHERE 계좌번호 = 'A계좌' AND 잔액 >= 1,000,000 ;

T4 -> UPDATE 계좌정보 SET 잔액 = 잔액 -1,000,000 WHERE 계좌번호 = 'A계좌' ;

T5 -> UPDATE 계좌정보 SET 잔액 = 잔액+1,000,000 WHERE 계좌번호 = 'B계좌' ;

COMMIT; (송금 업무를 위한 일련의 절차가 정상 처리되었으니 데이터 영구반영)

COMMIT / ROLLBACK

계좌정보 테이블

계좌번호	이름	잔액
A계좌	김민수	2000
B계좌	김영희	500000
…	…	…

BUT!!

만약 트랜잭션 도중에 오류가 발생한다면?

T1 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김민수' AND 계좌번호 = 'A계좌' ;

T2 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김영희' AND 계좌번호 = 'B계좌' ;

~~T3 -> SELECT 잔액 FROM 계좌정보 WHERE 계좌번호 = 'A계좌' AND 잔액 >= 1,000,000 ;~~

ERROR!!
잔액 부족!!!

T4 -> UPDATE 계좌정보 SET 잔액 = 잔액 -1,000,000 WHERE 계좌번호 = 'A계좌' ;

T5 -> UPDATE 계좌정보 SET 잔액 = 잔액+1,000,000 WHERE 계좌번호 = 'B계좌' ;

트랜잭션 오류!! 현재까지 실행된 트랜잭션 취소 요구!!!

COMMIT / ROLLBACK

계좌정보 테이블

계좌번호	이름	잔액
A계좌	김민수	2000
B계좌	김영희	500000
…	…	…

트랜잭션 도중에 하나라도 오류가 발생하면

이전에 했던 모든 활동 ROLLBACK

T1 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김민수' AND 계좌번호 = 'A계좌' ;

T2 -> SELECT 이름 FROM 계좌정보 WHERE 이름 = '김영희' AND 계좌번호 = 'B계좌' ;

~~T3 -> SELECT 잔액 FROM 계좌정보 WHERE 계좌번호 = 'A계좌' AND 잔액 >= 1,000,000 ;~~ ERROR!!

T4 -> UPDATE 계좌정보 SET 잔액 = 잔액 -1,000,000 WHERE 계좌번호 = 'A계좌' ;

T5 -> UPDATE 계좌정보 SET 잔액 = 잔액+1,000,000 WHERE 계좌번호 = 'B계좌' ;

김영희씨와 계좌 송금 1건에 관련하여 이루어진 트랜잭션은 모두 ROLLBACK

COMMIT / ROLLBACK

COMMIT ;

⇒ 마지막 COMMIT 시점 이후 실행한 트랜잭션 결과를 데이터베이스에 영구 저장

ROLLBACK ;

⇒ 실행한 트랜잭션 결과에 대해서 실행하기 전 상태로 원상복구

⇒ 마지막으로 COMMIT 한 시점까지만 ROLLBACK 가능

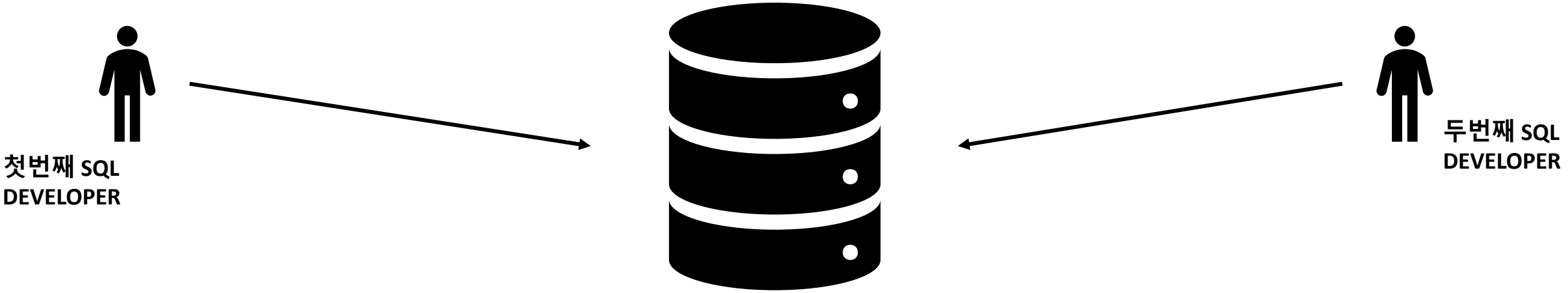
보통 DML (UPDATE , INSERT ,DELETE) 한 대상은 바로 테이블에
영구 반영되는 것이 아니므로 COMMIT 을 명시해야 반영

COMMIT / ROLLBACK

실제 COMMIT과 ROLLBACK의 구조를 이해하기 위해

SQL DEVELOPER를 하나 더 실행하여

두 명이 한 서버에 접근하여 작업 중인 것처럼 설정하고 실습

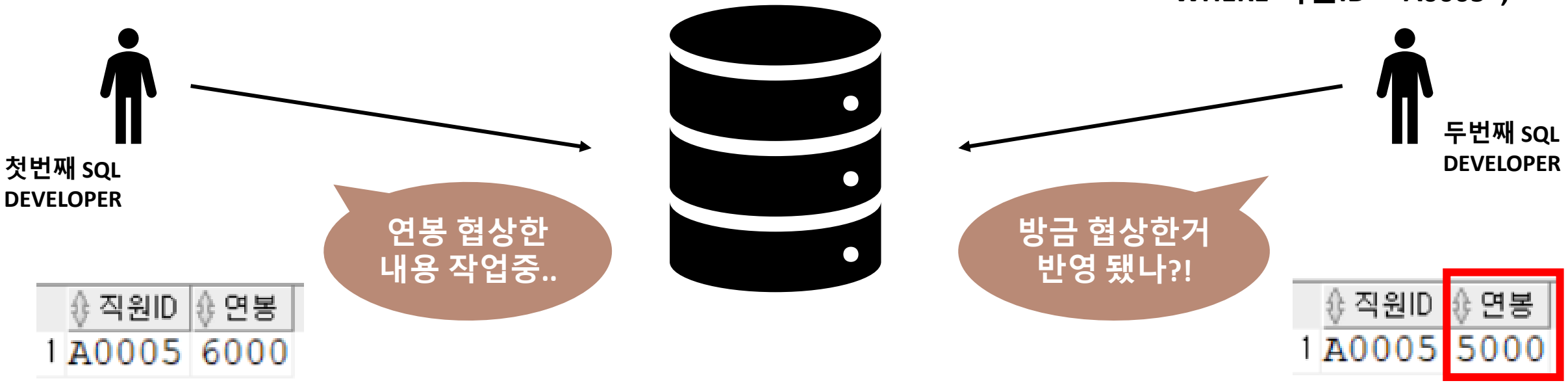


COMMIT / ROLLBACK

UPDATE 직원 SET 연봉 = 연봉 + 1000 WHERE 직원ID = 'A0005';

… (작업 중) …

SELECT 직원ID, 연봉
FROM 직원
WHERE 직원ID = 'A0005' ;



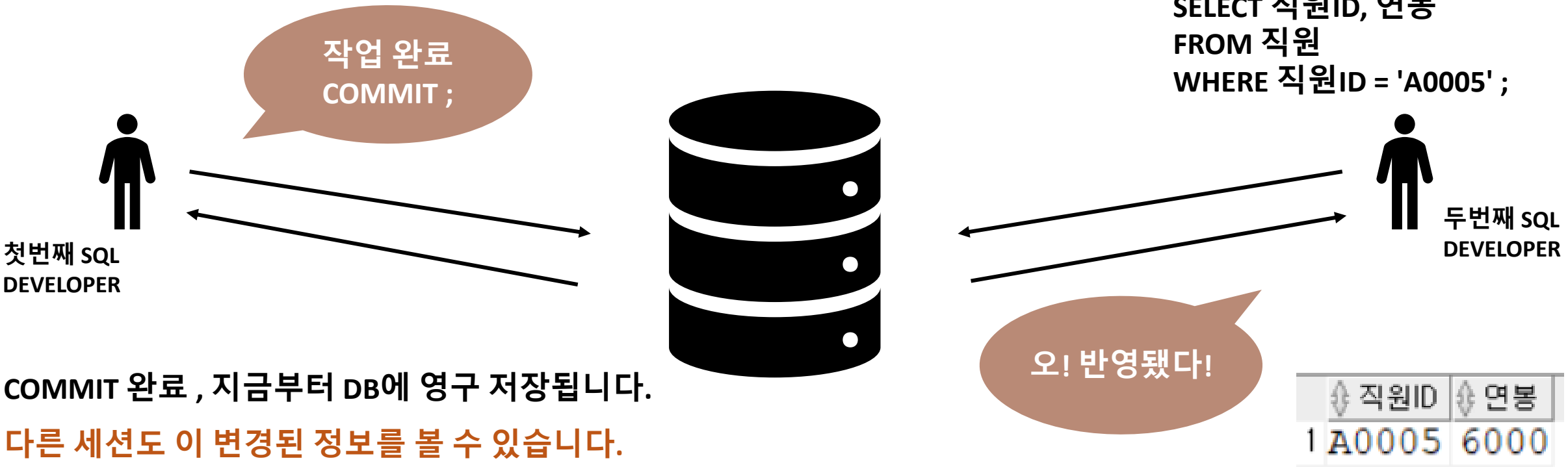
	직원ID	연봉
1	A0005	6000

	직원ID	연봉
1	A0005	5000

변경을 가한 세션에서는 데이터 내용이 변경된 것처럼 보이지만

아직 데이터베이스에 영구반영 된 것이 아니기 때문에(COMMIT 이전) 다른 세션에서는 이전 값 출력

COMMIT / ROLLBACK



COMMIT 완료 , 지금부터 DB에 영구 저장됩니다.

다른 세션도 이 변경된 정보를 볼 수 있습니다.

ROLLBACK은 내가 작업한 내용을 되돌리는 것이므로 다른 세션에서 확인 X

만약, COMMIT을 해버리면 되돌릴 수 없으므로 트랜잭션 순서를 거슬러 올라가며 데이터 복구

SAVEPOINT

SAVEPOINT란?

ROLLBACK 명령어에 대해 특정 지점까지만 복구하도록 조절

SAVEPOINT

```
UPDATE 직원 SET 연봉 = 연봉 + 1000 WHERE 직원ID = 'A0001' ;
SAVEPOINT SV1 ;
UPDATE 직원 SET 연봉 = 연봉 + 1000 WHERE 직원ID = 'A0002' ;
SAVEPOINT SV2 ;
UPDATE 직원 SET 연봉 = 연봉 + 1000 WHERE 직원ID = 'A0003' ;
SAVEPOINT SV3;
```

```
ROLLBACK TO SV2 ;
```

```
ROLLBACK;
```

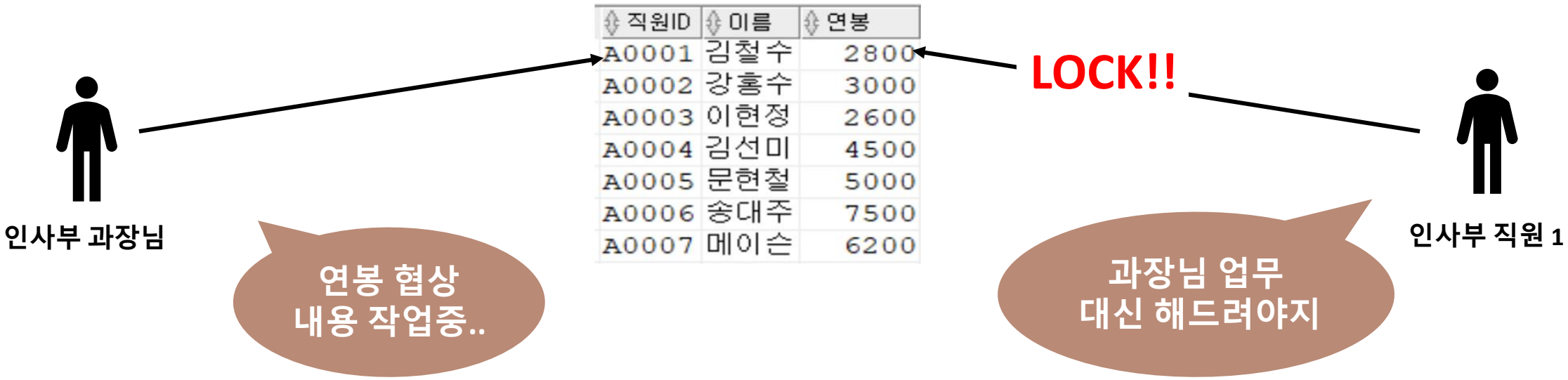
SV2 지점까지만 ROLLBACK !!

마지막 COMMIT 시점까지 ROLLBACK !!

LOCK의 개념

LOCK은 둘 이상의 세션이

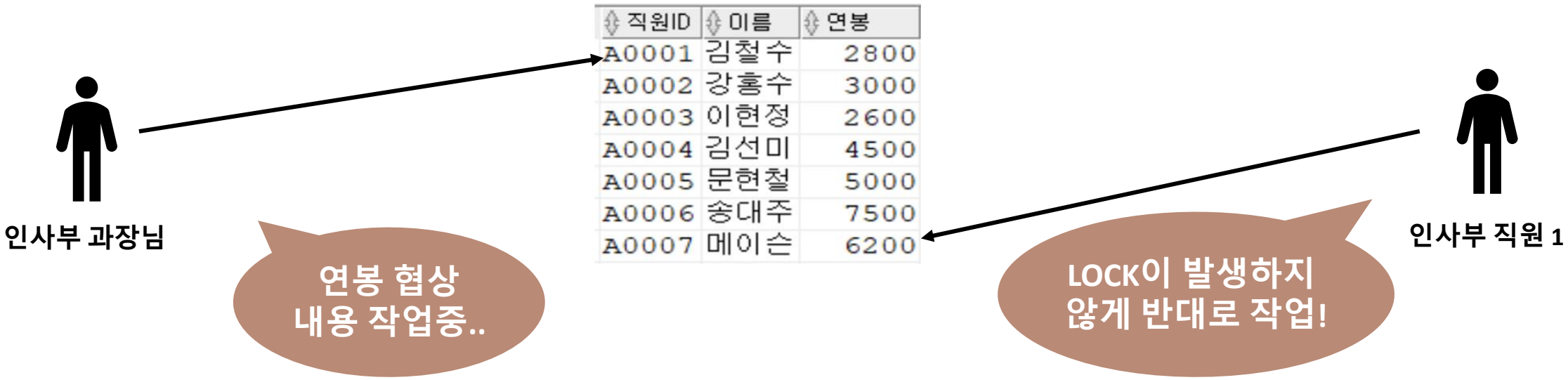
똑같은 행을 조작하려고 할 때 충돌하는 현상 (베타 LOCK)



LOCK의 개념

서로 다른 행을 조작한다면 LOCK이 발생하지 않으니 괜찮지만,

DML 작업 시 미리 협의하는 것이 중요



LOCK의 개념

그럼에도 불구하고 LOCK에 걸렸다면?

→ 트랜잭션이 작업을 진행하지 못하고 멈추는 현상 발생 (BLOCKING)

LOCK 해제방법

1. 원인이 되는 세션에서 COMMIT 혹은 ROLLBACK 을 실행 (LOCK = 자물쇠 , COMMIT/ROLLBACK = 열쇠)
2. DBA 에게 가서 LOCK을 풀어달라고 요청
3. LOCK 및 세션의 KILL 방법을 조회해 직접 세션 kill 실행

   → 이런 경합은 성능에 좋지 않으니 최소화!

LOCK의 개념

그럼에도 불구하고 LOCK에 걸렸다면?

→ 트랜잭션이 작업을 진행하지 못하고 멈추는 현상 발생 (BLOCKING)

LOCK 해제방법

1. 원인이 되는 세션에서 COMMIT 혹은 ROLLBACK 을 실행 (LOCK = 자물쇠 , COMMIT/ROLLBACK = 열쇠)
2. DBA 에게 가서 LOCK을 풀어달라고 요청
3. LOCK 및 세션의 KILL 방법을 조회해 직접 세션 KILL 실행

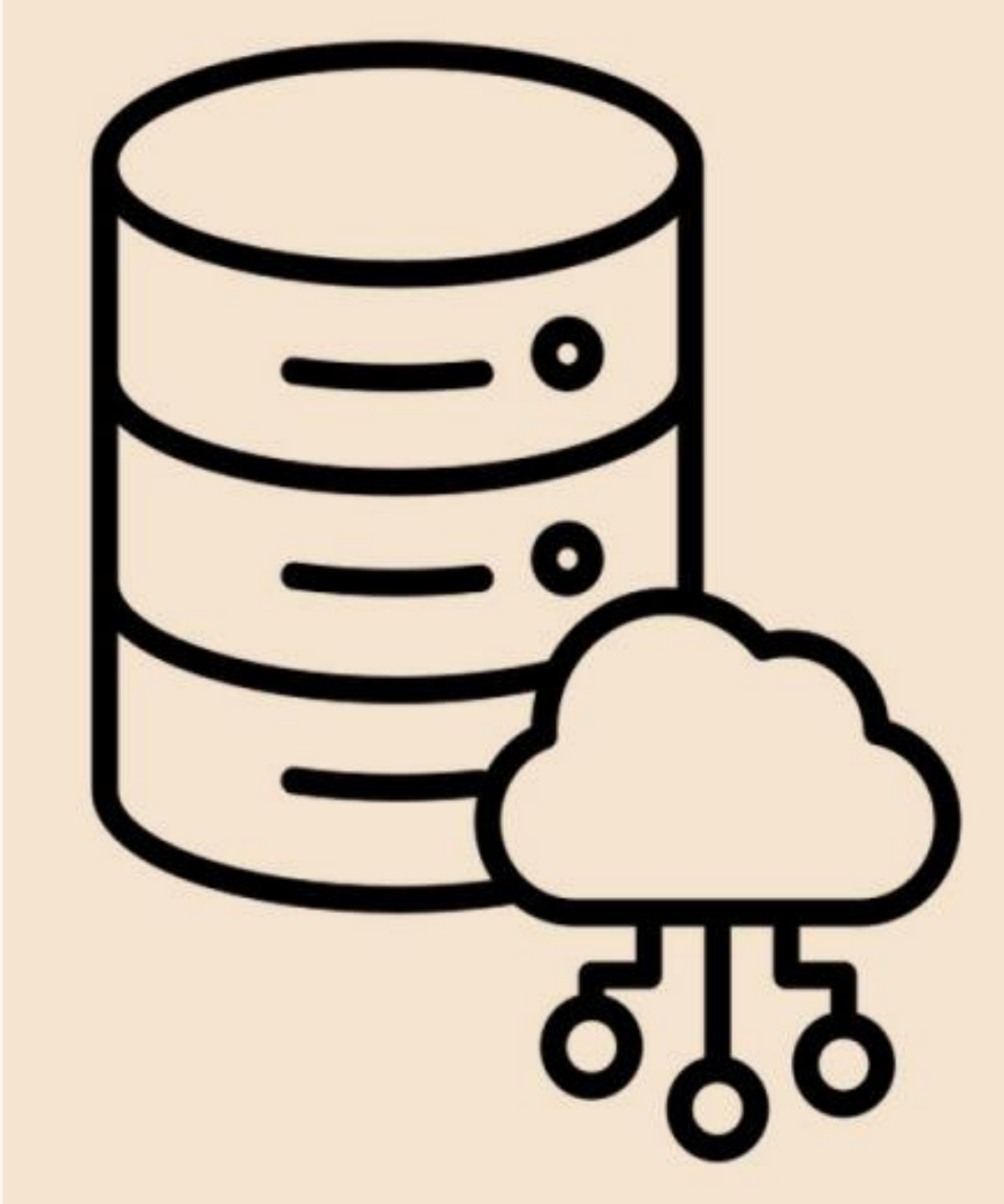
   → 이런 경합은 성능에 좋지 않으니 최소화!

DCL
(Data Control Language)

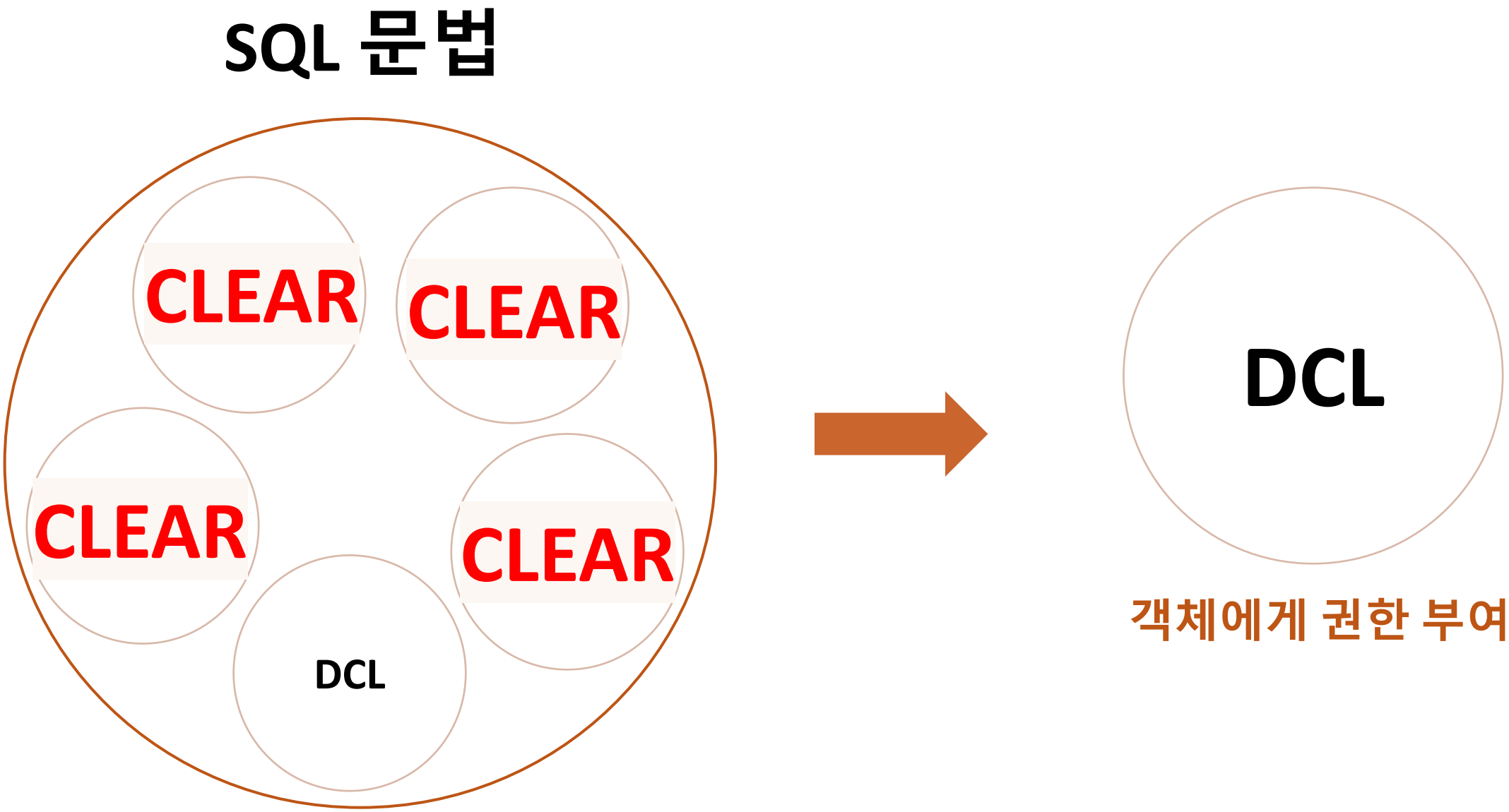
1 DCL 이란?

2 GRANT, REVOKE, ROLE

3 권한 부여 실습



DCL 이란?



DCL 이란?

DCL(Data Control Language)

- 데이터 제어어
- **데이터에 접근하거나 객체에 권한을 부여하는 역할**
- **GRANT** : 특정 작업에 대한 수행 권한 부여
- **ROLE** : 여러 권한을 그룹으로 묶어서 사용할 수 있게 함
- **REVOKE** : 특정 작업에 대한 권한 박탈, 회수

GRANT, REVOKE, ROLE

GRANT

특정 작업에 대한 수행 권한 부여

GRANT 부여할권한 (ON 대상객체) TO 부여받을계정

예) GRANT CREATE TABLE TO SERVICE

→ SERVICE 계정에 테이블을 만들 수 있는 권한 부여

GRANT, REVOKE, ROLE

GRANT SELECT ON 조회할 계정 . 객체이름 TO 권한을 받을 계정

```
SQL> GRANT SELECT ON SERVICE.직원 TO FUNDB;

Grant succeeded.
```

이제 FUNDB 서버를 사용하는 유저는 SERVICE 서버의 직원테이블을 조회할 수 있다!

마찬가지로 INSERT, DELETE, UPDATE 등 다양한 권한 부여 가능

Q. 만약 사용자가 늘어난다면 권한을 하나하나 부여해야 할까?

GRANT, REVOKE, ROLE

ROLE

여러 권한을 그룹으로 묶어서 사용할 수 있게 함

1.
```
SQL> CREATE ROLE ROLE2 ;
```

2.
```
SQL> GRANT CREATE TABLE, CREATE SESSION TO ROLE2 ;
```

3.
```
SQL> GRANT ROLE2 TO SERVICE ;
```

GRANT, REVOKE, ROLE

REVOKE

특정 작업에 대한 권한 박탈, 회수

REVOKE 회수할권한 (ON 회수할객체) FROM 회수당할계정

```
SQL> REVOKE CREATE SESSION FROM ROLE2 ;
```

권한을 회수한 대상이 나누어 줬던 권한도 함께 회수!

```
REVOKE CREATE SESSION FROM SERVICE
*
ERROR at line 1:
ORA-01952: system privileges not granted to 'SERVICE'
```

권한 부여 실습

실습 문제

1. SQL COMMAND LINE을 열어 SYSTEM 관리자 계정으로 접속 (ID : SYSTEM , PWD : 12345)

2. SERVICE TABLE이 가지고 있는 VIEW 생성 권한, SEQUENCE 생성 권한을 회수

3. ROLE3이라는 이름의 ROLE을 생성

3. VIEW 생성 권한, SEQUENCE 생성 권한을 하나의 ROLE로 만들고 이름을 ROLE3으로 저장

4. SERVICE 계정에게 ROLE3이 가진 권한을 부여

권한 부여 실습

실습 풀이

```
SQL> CONN SYSTEM / 12345 ;
```

1. SQL COMMAND LINE을 열어 SYSTEM 관리자 계정으로 접속 (ID : SYSTEM , PWD : 12345)

2. SERVICE TABLE이 가지고 있는 VIEW 생성 권한, SEQUENCE 생성 권한을 회수

```
SQL> REVOKE CREATE VIEW, CREATE SEQUENCE FROM SERVICE ;
```

3. ROLE3이라는 이름의 ROLE을 생성

```
SQL> CREATE ROLE ROLE3 ;
```

4. ROLE3에게 VIEW 생성 권한, SEQUENCE 생성 권한을 부여

```
SQL> GRANT CREATE VIEW, CREATE SEQUENCE TO ROLE3 ;
```

5. SERVICE 계정에게 ROLE3이 가진 권한을 부여

```
SQL> GRANT ROLE3 TO SERVICE ;
```