[격파르타] SQLD 자격증 챌린지 - 챕터 9

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- SQLD 시험에서 중요한 SQL을 사용하는 기본적인 방법에 대해 학습합니다.

[목차]

01. GROUP BY와 HAVING
02. ORDER BY
03. TCL
04. DCL

모든 토글을 열고 닫는 단축키
Windows :
`Ctrl` + `alt` + `t`
Mac :
`⌘` + `⌥` + `t`

---

01. GROUP BY와 HAVING

데이터를 그룹으로 묶고 집계하는 법에 대해 학습합니다.

▼ 1) 집계 함수 (Aggregate Function)

집계 함수는 여러 데이터들의 정보를 집계하여 연산을 해주는 함수입니다. GROUP BY 절에 작성한 칼럼을 기준으로, 그룹으로 모인 상태에서 각 그룹의 집계를 계산하는 데 사용됩니다. 집계 함수는 SELECT, HAVING, 그리고 ORDER BY 절에 사용할 수 있습니다.

☑️ 집계 함수의 기본 구조

- ALL
  - 모든 값을 기준으로 집계 할 때 사용하는 옵션입니다.
  - 기본값이므로 생략 가능합니다.
- DISTINCT
  - 같은 값을 하나의 데이터로 간주할 때 사용하는 옵션입니다.
  - SELECT 문의 결과에서 유일한 하나의 행만 출력합니다.

```
집계 함수명( [DISTINCT | ALL] 칼럼이나 표현식 )
```

☑️ 집계 함수의 종류

항목	결과
COUNT(*)	**NULL 값을 포함한** 행의 수를 출력합니다.
COUNT(표현식)	표현식의 값이 NULL 값인 것을 제외한 행의 수를 출력합니다.
SUM([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 합계를 출력합니다.
AVG([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 평균을 출력합니다.
MAX([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 최댓값을 출력합니다. (문자, 날짜 타입도 사용 가능)
MIN([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 최솟값을 출력합니다. (문자, 날짜 타입도 사용 가능)
STDDEV([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 표준 편차를 출력합니다.
VARIAN([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 표현식의 분산을 출력합니다.
기타 통계 함수	DBMS를 제공하는 회사 별로 다양한 통계식을 제공합니다. 표현식의 NULL 값을 제외합니다.

☑️ 집계 함수의 특징

- 집계 함수는 COUNT(*)를 제외하고는 NULL 값은 제외합니다.
- 집계 함수는 WHERE 절에 올 수 없습니다.
    - WHERE 절이 GROUP BY 절 보다 먼저 수행이 되기 때문에 행들이 소그룹으로 묶이기 전에 집계 함수가 실행되어 제대로 된 집계를 할 수 없어지기 때문입니다.

▼ 2) GROUP BY 절

☑️ 기본개념

GROUP BY 절은 데이터들을 작은 그룹으로 분류하여 해당 그룹에 대한 항목별로 통계 정보를 얻고자 할 때 사용하는 절입니다. 주로 SQL 쿼리문에서 FROM과 WHERE 뒤에 위치하게 되며 그룹별 기준을 정한 후 SELECT에 집계함수를 이용하여 원하는 데이터를 확인할 수 있습니다. 이때 집계 함수의 통계 정보에서 NULL 값은 제외됩니다.

☑️ GROUP BY 절 특징

GROUP BY 절에 적은 칼럼을 기준으로 결과 집합을 그룹화 합니다. 일반적으로 FROM TABLE 이후에 작성합니다. GROUP BY절을 사용할 때 숙지해야 할 부분에 대해 학습해 보겠습니다.

- GROUP BY 절을 사용할 때는 SELECT 절과는 다르게 ALIAS 명을 사용할 수가 없습니다.
- WHERE 절은 전체 데이터를 GROUP으로 나누기 전에 행들을 미리 주어진 조건에 맞춰 가져옵니다.
- 그럼 만약, GROUP BY로 그룹을 지은 상태에서 원하는 조건으로 필터링 하고싶다면?
  이때 HAVING 절을 사용하면 됩니다. HAVING 절은 GROUP BY절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건을 표시할 수 있습니다.
- GROUP BY 절에 의한 소그룹 별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력할 수 있습니다.
- 원칙적으로 관계형 데이터베이스 환경에서는 ORDER BY 절을 명시해야 데이터가 정렬이 됩니다.
- GROUP BY 절보다 WHERE 절이 먼저 수행되기 때문에 집계 함수는 WHERE 절에 올 수 없습니다.

- 기본 구조

```
SELECT [DISTINCT] 칼럼명 [ALIAS명]
FROM 테이블명
[WHERE 조건식]
```

```
[GROUP BY 칼럼이나 표현식]
[HAVING 그룹조건식] ;
```

- 예시

```
SELECT position '포지션', AVG(height) '평균 키' FROM player;
--> Error 발생, 단일 그룹의 집계 함수가 아님

SELECT position '포지션', AVG(height) '평균 키'
FROM player
GROUP BY position '포지션';
--> Error 발생, GROUP BY에서 Alias 사용 불가

SELECT
    position 포지션,
    COUNT(*) '인원수',
    COUNT(hegiht) '키 대상',
    MAX(hegiht) '최대 키',
    MIN(hegiht) '최소 키',
    ROUND(AVG(hegiht),2) '평균 키'
FROM player GROUP BY position;
--> 정상 조회 가능
```

- 실습

> 다음 포켓몬 데이터에서 attr별로 평균 weight를 조회해보세요.

▼ 결과

ATTR	AVG(WEIGHT)
Fire	140
grass	76.6666666666666666666666666666666666666667
normal	-
Electric	15

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(20) NOT NULL,
  attr VARCHAR2(20) DEFAULT 'normal',
    weight NUMBER
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 30);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 50);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 150);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 200);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 15);
INSERT INTO pokemon (pm_id, name) VALUES (86, 'Seel');
```

```
-- 전체 데이터 먼저 확인하고 진행
SELECT * FROM pokemon;

-- 그룹 별로 평균 몸무게 조회
SELECT
    attr,
```

```
    AVG(weight)
FROM pokemon
GROUP BY attr;
```

▼ 3) HAVING 절

☑️ 기본

HAVING절은 WHERE절과 같이 특정 조건을 만족하는 결과 데이터만 표시를 할 수 있으나, 집계된 결과에 대해 사용한다는 것이 특징입니다.

☑️ 특징

- 집계된 결과를 기준으로 조건이 필요한 경우 HAVING 절을 활용합니다.
- WHERE 절과 비슷하지만 그룹을 나타내는 결과의 행에 조건이 적용된다는 점에서 차이가 있습니다.
- GROUP BY 절과 함께 사용하여 특정한 제한 조건의 그룹화된 내용을 필터링 합니다.

☑️ HAVING 절과 WHERE절의 차이

구분	HAVING	WHERE
조건에서 집계 함수 사용 가능 여부	가능	불가능
단독 사용 가능 여부	- 불가능 - GROUP BY와 함께 사용 가능	- 가능 - GROUP BY 이전에 WHERE 절이 먼저 실행
기타 특징	SELECT 절에 사용되지 않은 칼럼이나 집계 함수가 아니더라도 GROUP BY 절의 기준 항목이나 집계 함수를 이용하여 조건을 표시할 수 있습니다.	WHERE 절이 GROUP BY 보다 먼저 실행되기 때문에 그룹으로 묶인 데이터의 개수가 WHERE의 조건에 따라 다를 수 있습니다.

- 기본 구조

  HAVING 절의 위치는 GROUP BY 뒤에 오는 것이 적절합니다. HAVING 절이 GROUP BY 앞에 와도 결괏값은 동일하지만 쿼리문의 논리적 문맥의 흐름상 GROUP BY 뒤에 HAVING 절이 있는 것이 자연스럽기 때문입니다.

```
SELECT [DISTINCT] 칼럼명 [ALIAS명]
FROM 테이블명
[WHERE 조건식]
[GROUP BY 칼럼이나 표현식]
[HAVING 그룹조건식] ;
```

- 예시

```
SELECT country, COUNT(id) AS COUNT
    FROM supplier
GROUP BY country
HAVING COUNT(id) > 2;
```

- 실습

> 다음 포켓몬 데이터에서 attr 별로 평균 weight를 조회해보세요.
> (단, attr 이 NULL인 경우는 조회 결과에서 제외하세요.)

▼ 결과

1번

ATTR	AVG(WEIGHT)
Grass	30
Fire	80
Rock	-
Electric	15

2번

ATTR	AVG(WEIGHT)
Grass	30
Fire	80
Rock	-
-	85
Electric	15

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(20) NOT NULL,
  attr VARCHAR2(20),
    weight VARCHAR2(20)
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'Grass', '30');
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', '80');
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', '15');
INSERT INTO pokemon (pm_id, name) VALUES (54, 'Psyduck');
INSERT INTO pokemon (pm_id, name, attr) VALUES (76, 'Golem', 'Rock');
INSERT INTO pokemon (pm_id, name, weight) VALUES (86, 'Seel', '85');
```

```
-- Oracle

-- 1번과 2번 비교하기
-- attr 칼럼의 NULL 값 여부에 따른 차이

--1번
SELECT
    attr,
    AVG(weight)
FROM pokemon
GROUP BY attr
HAVING attr IS NOT NULL;


-- 2번
SELECT
    attr,
    AVG(weight)
FROM pokemon
GROUP BY attr
-- HAVING attr IS NOT NULL;
```

▼ 4) 집계 함수와 NULL

조회 결과 중에서 NULL 이 있는 경우 집계 연산을 하기 위해 NVL 혹은 ISNULL 함수를 사용하여 0으로 변경할 수 있습니다. 다만, 이 함수를 집계 함수 내부에 넣어 사용하는 경우에는 불필요한 부하가 발생합니다. 다행히 집계 함수는 NVL, ISNULL 과 같은 함수를 내부에서

사용하지 않아도 NULL 데이터는 집계 함수의 대상에서 제외하고 함수 연산을 처리합니다. (단, COUNT(*) 는 NULL 을 포함하여 모든 칼럼을 계산해서 제외)

만약 100명의 성적 데이터 중에서 10명의 성적이 NULL이라면 AVG(성적)에서는 NULL을 제외한 90명의 데이터를 기준으로 평균값을 구하게 됩니다.

- 예시

```
CREATE TABLE student (
  id NUMBER,
  class NUMBER,
  score NUMBER
);

INSERT INTO student VALUES (1, 1, 10);
INSERT INTO student VALUES (2, 1, 10);
INSERT INTO student VALUES (3, 1, 10);
INSERT INTO student VALUES (4, 1, 10);
INSERT INTO student VALUES (5, 1, 10);
INSERT INTO student VALUES (6, 1, 10);
INSERT INTO student VALUES (7, 1, 10);
INSERT INTO student VALUES (8, 1, 10);
INSERT INTO student VALUES (9, 1, null);
INSERT INTO student VALUES (10, 1, null);
```

```
SELECT
    class,
    AVG(score),
    COUNT(score),
    COUNT(*)
FROM student
GROUP BY class;
```

CLASS	AVG(SCORE)	COUNT(SCORE)	COUNT(*)
1	10	8	10

▼ 연습문제

☑️ 문제 1

Q. 문제	**집계 함수의 종류에 대한 설명으로 옳지 않은 것은?**
A. (1)	STDDEV : NULL 값을 제외한 표준 편차를 출력한다
A. (2)	COUNT(*) : NULL 값을 제외한 행의 수를 출력한다
A. (3)	VARIAN : NULL 값을 제외한 표현식의 분산을 출력한다
A. (4)	AVG : NULL 값을 제외한 평균을 출력한다

▼ 정답

(2) NULL 값을 포함한 행의 수를 출력한다

☑️ 문제 2

Q. 문제	**HAVING 절과 WHERE절의 차이에 대한 설명으로 옳지 않은 것은?**
A. (1)	WHERE절은 조건에서 집계 함수 사용이 가능하다
A. (2)	HAVING절은 단독 사용이 불가하다
A. (3)	GROUP BY 이전에 WHERE 절이 먼저 실행되어야 한다
A. (4)	HAVING절은 GROUP BY와 함께 사용 가능하다

▼ 정답

(1) WHERE절은 조건에서 집계 함수 사용이 불가하다

☑️ 문제 3

Q. 문제	**SELECT문 실행 순서로 옳은 것은?**
A. (1)	SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY → LIMIT
A. (2)	SELECT → FROM → WHERE → GROUP BY → ORDER BY → HAVING → LIMIT
A. (3)	FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT
A. (4)	FROM → SELECT → WHERE → GROUP BY → HAVING → ORDER BY → LIMIT

▼ 정답

(3)

02. ORDER BY

✔️ **조회되는 데이터들을 특정 칼럼을 기준으로 정렬하여 출력하는 ORDER BY에 대해 학습합니다.**

▼ 1) ORDER BY 기본 개념

☑️ 기본

ORDER BY 절은 SQL 문장으로 조회되는 데이터들을 특정 칼럼을 기준으로 정렬하여 출력하기 위해 사용합니다. 정렬하기 위해서는 칼럼명을 사용해도 되지만 SELECT 절에서 사용하는 ALIAS 명을 사용하거나 1 부터 칼럼의 순서를 정수로 명시해도 됩니다. 기본적으로 정렬은 오름차순으로 정렬이 되고 내림차순으로 정렬을 하기 위해서는 `DESC` 를 옵션으로 추가해 주면 됩니다.

☑️ 특징

- ORDER BY 절에 기재한 칼럼 뒤에 정렬 방식을 설정할 수 있으며 아무것도 적지 않으면 기본 값은 오름차순(ASC)으로 설정됩니다. 만약 내림차순으로 지정하려면 DESC로 명시하면 됩니다.
- ORDER BY 절에 칼럼명 대신에 ALIAS로 설정된 값을 사용할 수 있습니다.
- 데이터 별 오름차순 정렬 기준
  - 숫자는 작은 값부터 정렬
  - 날짜는 가장 빠른 날짜부터 정렬
- NULL에 대하여 ORDER BY 했을 때 처리하는 부분이 DBMS를 제공하는 회사 별로 처리 방식이 다릅니다.
  - Oracle에서는 NULL을 가장 큰 값으로 간주
  - SQL Server는 NULL을 가장 작은 값으로 간주

- 기본 구조

```
ORDER BY 칼럼(Column)이나 표현식
```

- 예시

```
SELECT dname, loc, deptno FROM dept ORDER BY dname, loc, deptno DESC;
--> dname 을 먼저 정렬하고
-- 같은 dname인 경우에는 loc를 기준으로 정렬하고
-- 같은 dname과 loc인 경우에는 deptno를 이용하여 내림차순 정렬
```

- 실습

> 다음 포켓몬 데이터에서 name과 height 를 조회하는데 이 때 height를 내림차순 순서로 정렬해서 출력해주세요. 만약 키가 같다면 name 칼럼을 기준으로 알파벳 순서로 정렬해주세요.

▼ 결과

NAME	HEIGHT
Venusaur	250
Charmeleon	120
Ivysaur	90
Charmander	80
Bulbasaur	50
Pikachu	50

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(20) NOT NULL,
  attr VARCHAR2(20) DEFAULT 'normal',
    height NUMBER
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 50);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 90);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 250);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 120);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 50);
```

```
SELECT
    name,
    height
FROM pokemon
ORDER BY height DESC, name ASC;
```

▼ 2) SELECT 문장 실행 순서

GROUP BY와 ORDER BY를 동시에 사용하는 경우 SELECT 문장은 6개의 절로 구성되며 실행 순서는 다음과 같습니다.

- 실행 순서

```
5. SELECT 칼럼명 [ALIAS명]          -- 5. 데이터의 값을 출력/계산
1. FROM 테이블명                    -- 1. 발췌대상 테이블 참조
2. WHERE 조건식                     -- 2. 발췌 대상 데이터가 아닌 것은 제거
3. GROUP BY 칼럼(Column)이나 표현식 -- 3. 행동들을 소그룹화
4. HAVING 그룹조건식                -- 4. 그룹핑된 값의 조건에 맞는 것만을 출력
6. ORDER BY 칼럼(Column)이나 표현식;-- 6. 데이터를 정렬
```

GROUP BY 절에서 그룹화될 칼럼을 정의하게 되면 데이터베이스는 일반적인 SELECT 문장처럼 FROM 절에 정의된 테이블 구조를 그대로 가져가지 않습니다. GROUP BY 절의 그룹 될 칼럼과 집계 함수에 사용될 수 있는 숫자형 데이터 칼럼들의 집합을 새로 만들게 됩니다. 그렇게 되면 GROUP BY 이후에 실행되는 SELECT 절이나 ORDER BY 절에서는 새롭게 만들어진 데이터에 접근을 하기 때문에 기존 테이블의 일반 칼럼을 사용할 경우에는 에러가 발생하게 됩니다. 다만 ORDER BY 절은 집계 함수를 이용한 정렬은 정상적으로 이루어집니다.

- 예시

```
SELECT job, sal FROM emp GROUP BY job HAVING COUNT(*) > 0 ORDER BY sal;
--> Error 발생, sal 은 GROUP BY로 job 이 그룹화 될 것이기 때문에
-- 일반 칼럼 sal 사용으로 에러가 발생

SELECT job FROM emp GROUP BY job HAVING COUNT(*) > 0 ORDER BY sal;
--> Error 발생, ORDER BY 에서 일반 칼럼 sal 을 사용함

SELECT job FROM emp GROUP BY job HAVING COUNT(*) > 0
ORDER BY MAX(empno), MAX(mgr), SUM(sal), COUNT(deptno), MAX(hiredate);
--> 정상 조회 가능, ORDER BY 에서 집계 함수를 이용한 정렬은 가능함
```

▼ 3) TOP N 쿼리

TOP N 쿼리는 상위 N개의 데이터를 보여주기 위한 쿼리를 지칭합니다. 이때는 데이터를 어떤 순서로 정렬해서 보여줄 건지 결정해야 합니다. Oracle은 ROWNUM, SQL Server는 TOP (N) 함수를 이용해서 해당 부분을 처리할 수 있습니다.

(1) ROWNUM (Oracle)

ROWNUM 은 조회된 데이터에 번호를 매겨 원하는 개수의 데이터를 가져올 수 있습니다. 이때 주의해야 할 사항이 하나 있습니다. 정렬이 완료되고 데이터의 일부가 ROWNUM으로 추출되는 것이 아니라 WHERE 절에 작성된 ROWNUM으로 데이터의 일부가 먼저 추출이 되

고 나중에 데이터의 정렬 작업이 일어나게 됩니다.

- 예시

```
SELECT ename, sal FROM emp WHERE ROWNUM < 4 ORDER BY sal DESC;
--> sal 이 제일 높은 3개의 데이터를 조회하는 것이 아닌
-- 3개를 랜덤하게 가져온 후 정렬을 하게 됨
```

- 실습

> 다음 포켓몬 데이터에서 키가 가장 작은 포켓몬의 이름, 속성, 그리고 키를 3개만 조회해주세요. 이때 동일한 데이터 중복으로 같은 데이터가 있는 경우에도 같이 출력해주세요.

▼ 결과

NAME	ATTR
Bulbasaur	grass
Ivysaur	grass
Venusaur	grass

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(20) NOT NULL,
  attr VARCHAR2(20) DEFAULT 'normal',
    height NUMBER
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 50);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 90);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 250);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 120);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 50);
INSERT INTO pokemon VALUES (86, 'Seel', 'Ice', 80);
```

```
-- Oracle

SELECT
    name,
    attr,
    height
FROM pokemon WHERE ROWNUM < 4 ORDER BY height, name;
```

(2) TOP (SQL Server)

SQL Server는 TOP 함수를 이용하여 일부 데이터를 출력할 수 있습니다. SQL Server는 TOP 조건을 사용하게 되면 ORDER BY로 데이터 정렬 후 원하는 데이터의 일부를 추출할 수 있습니다.

- 기본 구조
  - 실습

> 다음 포켓몬 데이터에서 키가 가장 작은 포켓몬의 이름, 속성, 그리고 키를 3개만 조회해주세요. 이때 동일한 데이터 중복으로 같은 데이터가 있는 경우에도 같이 출력해주세요.

▼ 결과

NAME	ATTR	HEIGHT
Bulbasaur	grass	50
Ivysaur	grass	90
Venusaur	grass	250

```
CREATE TABLE pokemon (
  pm_id INT PRIMARY KEY NOT NULL,
  name VARCHAR(20) NOT NULL,
  attr VARCHAR(20) DEFAULT 'normal',
    height INT
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 50);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 90);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 250);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 120);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 50);
INSERT INTO pokemon VALUES (86, 'Seel', 'Ice', 80);
```

```
-- SQL Server

SELECT
    TOP(3) WITH TIES
    name, attr,
    height
FROM pokemon ORDER BY height, name;
```

- **PERCENT**

  Expression의 몇 퍼센트를 출력할지 정하는 옵션입니다.

- **WITH TIES**

  ORDER BY 절의 조건을 기준으로 데이터의 값이 동일하여 Expression 의 개수를 초과하게 되는 경우에도 동일한 값의 데이터를 추가로 표시를 하고자 할 때 사용하는 옵션입니다.

```
TOP (Expression) [PERCENT] [WITH TIES]
```

- 예시

```
SELECT TOP(2) WITH TIES ename, sal FROM emp ORDER BY sal DESC;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**ORDER BY 절에 사용되는 키워드로 적절한 것은?**
A. (1)	NOT NULL
A. (2)	LIKE
A. (3)	BETWEEN
A. (4)	DESC

▼ 정답

(4) 데이터 정렬 시 오름차순, 내림차순을 사용한다

☑️ 문제 2

Q. 문제	**ORDER BY절에 대한 특징으로 옳지 않은 것은?**
A. (1)	칼럼명 대신에 ALIAS로 설정된 값을 사용할 수 있다
A. (2)	오름차순의 정렬 기준 중, 숫자는 작은 값부터 정렬한다
A. (3)	내림차순의 정렬 기준 중, 날짜는 가장 오래된 날부터 정렬한다
A. (4)	Oracle에서는 NULL을 가장 큰 값으로 간주한다

▼ 정답

(3) 내림차순은 최근 날짜에서 과거 날짜 순서로 정렬하는 방식이다

☑️ 문제 3

Q. 문제	**아래에 있는 SELECT문의 구문 중 가장 늦게 실행되는 것은?**
A. (1)	SELECT
A. (2)	ORDER BY
A. (3)	WHERE
A. (4)	FROM

▼ 정답

(2)

03. TCL

✔️ **트랜잭션의 개념과 트랜잭션 제어어 TCL에 대해 학습합니다.**

▼ 1) 개요

☑️ TCL과 트랜잭션

TCL(Transaction Control Language)은 데이터를 삽입, 삭제, 수정하는 DML의 수행 이후 COMMIT이나 ROLLBACK을 하기 위해 사용하는 SQL입니다. TCL을 이해하기 위해선 TCL이 사용되는 지점, 즉 트랜잭션에 대해 알아야 하는데요. 지금까지 DML을 통해서 데이터를 삽입하고 수정과 삭제를 하는 방법에 대해 학습했습니다. 이러한 DML구문을 통해 데이터베이스의 상태는 변경됩니다. 그런데 이때 우리가 생각하는 것처럼 해당 구문 실행 이후에 바로 명령어가 실행되어 적용되는 것은 아닙니다. 마치 결재 서류에 사인을 해야 작성한 내용이 유효한 것처럼 커밋(COMMIT) 명령어를 수행 해야만 최종적으로 데이터베이스에 반영합니다. 또한, 사인한 결재 내용은 특정한 작업 진행 전이라면 취소 가능한 것과 마찬가지로 데이터베이스에 반영하기 전이라면 롤백(ROLLBACK) 명령어를 통해 데이터의 변경 사항을 취소할 수도 있습니다. TCL 구문은 이러한 데이터의 생성, 수정 및 삭제 과정을 커밋하거나 롤백 하는 일련의 논리적인 과정으로 구성됩니다. 조금 더 자세하게 살펴보겠습니다.

☑️ 트랜잭션

트랜잭션(Transaction)은 논리적인 연산의 단위로 생각해 볼 수 있습니다. 여기서 '논리적인 연산'이라고 하는 것은 분할할 수 없는 최소한의 단위로, 어떠한 연산을 전부 적용하거나 혹은 적용하지 않는 양자택일의 관계로 구성됩니다. 밀접히 관련되어 분리될 수 없는 한 개 이상의 데이터베이스 조작을 의미하는데, 이에 대해서 조금 더 구체적으로 살펴보겠습니다.

은행계좌에서 돈을 다른 은행의 통장으로 이체할 때의 과정을 생각해 보겠습니다. 통장에서 10,000원을 꺼낸 후 다른 통장으로 이체를 하면 어떻게 될까요? 이체를 받은 통장에는 내가 보낸 10,000원이 보내지겠죠? 이렇게만 생각하면 전혀 문제 될 게 없습니다. 그러면 만약 송금을 하는 과정에서 '송금' 버튼을 눌렀는데 갑자기 전력이 끊겨 중간에 연결이 끊어지면 어떻게 될까요? 금액의 절반만 보내져야 할까요? 아닙니다. 모든 과정이 취소되어야 합니다. 내 통장에서는 10,000원이 나가고 상대방의 통장에는 10,000원이 들어가는 과정이 완료되지 않았다면 모든 과정은 취소되고 처음부터 시작되어야 합니다. 이러한 '송금'이라는 하나의 논리적인 과정을 트랜잭션이라고 합니다.

☑️ 트랜잭션의 기본 특성 (ACID)

정리해보면 트랜잭션은 '데이터베이스에서 처리되는 논리적인 연산단위'라고 볼 수 있습니다. 트랜잭션이 연산 과정을 처리하며 지켜야 하는 대표적인 4가지 원칙이 있습니다. 원자성, 일관성, 고립성, 그리고 지속성 이렇게 4가지가 존재합니다. 각 특성의 영문 앞글자를 따서 'ACID'라고 부릅니다. 이에 대해서 자세하게 알아보겠습니다.

- **원자성(Atomicity)**
  - 모든 연산은 '최소한의 조작 묶음'이기 때문에 하나의 트랜잭션을 완료했다는 말은 모든 연산이 정확하게 수행 되었다는 것을 의미합니다.
  - 조작 과정에서 하나의 논리적인 연산이라도 실패하게 된다면 단순하게 실패 상태로 남는 것이 아닌 아무것도 수행하지 않은 상태로 남아 있어야 합니다.
  - 즉, 물건을 사기 위해 돈을 줬으면 물건이 있는 경우라면 물건을 받아야 하고 물건이 없다면 돈을 돌려받아야 합니다. 이러한 특성을 '원자성'이라고 부릅니다. (All or Nothing)

- **일관성(Consistency)**
  - 트랜잭션이 수행하기 전의 상태 그리고 수행하고 나서의 상태도 잘못된 내용이 없어야 합니다.
  - 돈이 없다면 송금이 되어서는 안되고, 송금된 돈을 받았는데 통장에는 송금된 돈이 저장이 되어야 하지 전달받은 돈의 이미지 형태가 저장이 되지 않아야 합니다.

- **고립성/격리성(Isolation)**
  - 트랜잭션이 다른 트랜잭션에 영향을 받아 잘못된 결과를 만들어서는 안됩니다.
  - 각각의 트랜잭션은 개별적으로 동작이 보장되어야 합니다.

- **지속성(Durability)**
  - 트랜잭션이 성공적으로 수행된다면 그 결과를 갱신한 데이터베이스는 내용을 영구적으로 유지하고 있어야 합니다.
  - 특정 시간 동안만 유지되고 다시 돌아가는 불안정한 부분이 있어서는 안됩니다.

▼ 2) COMMIT / ROLLBACK

☑️ COMMIT

COMMIT은 입력한 자료나 수정한 자료, 혹은 삭제한 자료에 대해서 문제가 없다고 판단이 되었을 때, 최종 트랜잭션을 확정하는 TCL 명령어입니다. COMMIT을 하지 않아도 변경된 데이터의 내용을 SELECT 문을 통해 조회가 가능합니다. 다만, 다른 사용자는 현재 변경된 내용에 접근하여 데이터의 변화를 확인할 수 없으며 변경 중인 데이터 베이스도 **Locking**이 되어 다른 사용자가 변경할 수도 없습니다. 데이터베이스를 변경하고 COMMIT을 하지 않으면 최종 데이터베이스에 변경 내용 저장이 확정되지 않았기 때문에 데이터 변경 이전 상태로 복구가 가능합니다. COMMIT을 하고 나면 변경된 내용이 데이터베이스에 반영이 되고 변경되기 전의 데이터는 사라지게 됩니다. Locking이 풀리면서 다른 사용자들이 값을 조회하고 수정할 수 있게 됩니다.

SQL Server는 기본적으로 AUTO COMMIT 모드이므로 사용자가 직접 COMMIT을 호출할 필요는 없습니다. 만약 에러가 발생을 한다면 자동적으로 ROLLBACK 처리가 되어 이전 데이터로 값이 복구가 됩니다. 물론 명시적으로 SQL Server 도 직접 트랜잭션의 시작과 끝을 지정하여 COMMIT 동작을 설정할 수 있습니다.

> 💡 **LOCKING(잠금)이란?**
>
> 잠금은 트랜잭션이 수행하는 동안 특정 데이터에 대해서 다른 트랜잭션이 동시에 접근하지 못하도록 제한 하는 기법입니다. 잠금이 걸린 데이터는 잠금 주체자만 접근할 수 있습니다.

- 기본 형태

```
-- Oracle

SQL 쿼리문;
COMMIT;
```

```
-- SQL Server

BEGIN {TRANSACTION|TRAN}
SQL 쿼리문
COMMIT [TRANSACTION];
```

- 예시

```
-- Oracle

INSERT INTO PLAYER VALUES ('1997035', 'K02', '이운재', 'GK', 182, 82, 1);
COMMIT;

UPDATE PLAYER SET HEIGHT = 100;
COMMIT;

DELETE FROM PLAYER;
COMMIT;
```

- 실습

  Rollback 실습과 통합하여 진행

☑️ ROLLBACK

트랙잭션 진행중 에러가 발생하거나 잘못된 연산이 이뤄진 경우 변경 전 가장 최신 상태로 되돌릴 수 있으며 이를 'ROLLBACK'이라고 부릅니다. ROLLBACK이 실행되면 COMMIT되지 않은 모든 데이터 변경 사항들이 취소되어 데이터의 이전 상태로 복구되고 관련된 Locking이 풀리면서 다른 사용자들이 데이터를 변경할 수 있게 됩니다.

✅ COMMIT과 ROLLBACK을 사용할 때 장점

💡 1. 데이터의 무결성 보장
2. 영구적인 변경을 하기 전에 데이터의 변경 사항 확인 가능
3. 논리적으로 연관된 작업을 묶어(Grouping) 처리 가능

✅ COMMIT과 ROLLBACK을 사용하지 않아도 트랜잭션이 종료되는 경우

💡 1. DDL 문장 실행 시 자동으로 COMMIT
2. DML 문장 이후에 COMMIT 없이 DDL 문장이 실행될 때
3. 정상적으로 DB 접속을 종료하면 COMMIT
4. 장애로 인한 비정상 종료로 DB 접속이 단절되었을 경우에는 AUTO ROLLBACK

- 기본 형태

```
-- Oracle
```

```
SQL 쿼리문
ROLLBACK;
```

```
-- SQL Server

BEGIN {TRANSACTION|TRAN}
SQL 쿼리문
ROLLBACK [TRANSACTION];
```

- 예시

```
-- Oracle

INSERT INTO PLAYER VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1);
ROLLBACK;

UPDATE PLAYER SET HEIGHT = 100;
```

```
-- SQL Server

BEGIN TRAN
INSERT INTO PLAYER VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1);

ROLLBACK;
```

- 실습

> 1. pokemon 테이블에 (25, 'Pickachu', 'Electric') 데이터를 추가하고 ROLLBACK 실행 후 전체 데이터를 조회해보세요.

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(10) NOT NULL,
  attr VARCHAR2(10) DEFAULT 'normal'
);
```

```
-- Oracle

INSERT INTO pokemon VALUES (25, 'Pickachu', 'Electric');
-- ROLLBACK이 있을 때와 없을 때의 차이
-- ROLLBACK;

SELECT * FROM pokemon;
```

> 2. pokemon 테이블에 (26, 'Raichu', 'Electric') 데이터를 추가하고 COMMIT을 하고 나서 ROLLBACK을 실행하고 전체 데이터를 조회해보세요.

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(10) NOT NULL,
  attr VARCHAR2(10) DEFAULT 'normal'
);
```

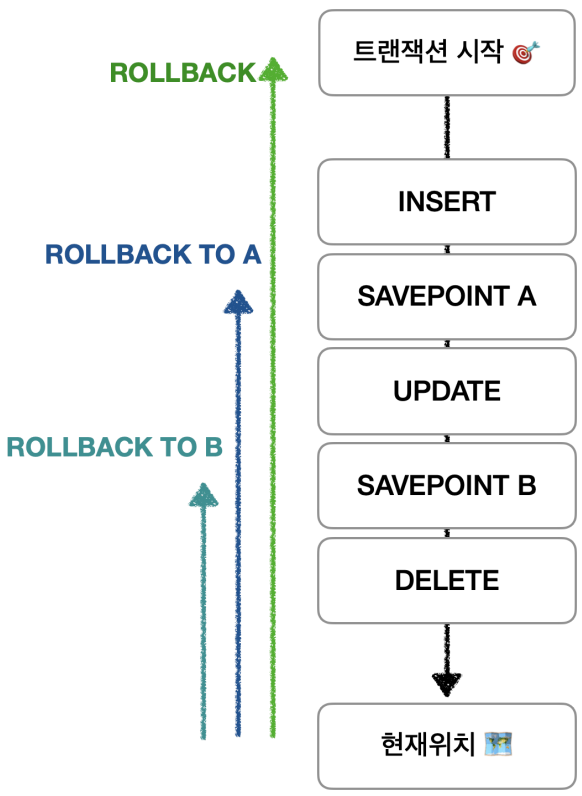
```
-- Oracle

INSERT INTO pokemon VALUES (26, 'Raichu', 'Electric');
-- 1번 실습과 어떤 차이점이 있는지 살펴보기
COMMIT;
ROLLBACK;
SELECT * FROM pokemon;
```

✅ **COMMIT과 ROLLBACK을 사용할 때의 차이**

1. **DML의 데이터 삽입, 수정, 삭제 이후 COMMIT or ROLLBACK 전 데이터 상태**
   - 데이터베이스에 최종적으로 적용된 상태가 아니라 메모리 영역에만 적용된 상태이기 때문에 변경 전 상태로 복구 가능합니다.
   - SELECT 구문을 통해 삽입, 수정, 삭제의 결과를 조회 할 수 있습니다.
     - 작업자는 가능
     - 작업자외 다른 사람은 불가능
     - 변경된 행은 Locking 설정으로 인해 다른 사용자가 변경할 수 없음
2. **DML의 데이터 삽입, 수정, 삭제 이후 COMMIT한 상황**
   - 데이터의 삽입, 수정, 삭제 내역이 데이터베이스에 완전하게 반영됩니다.
   - COMMIT 실행 시점 이후에는 해당 구문의 실행으로 인한 변화로 이전 데이터는 완전히 삭제합니다.
   - 작업자 뿐만 아니라 모든 사람은 SELECT 구문을 통해 동일한 결과를 얻을 수 있습니다.
   - 변경된 행의 Locking이 모두 풀리고 모든 사용자는 특정한 행에 대한 조작이 가능합니다.

▼ 3) SAVEPOINT

트랜잭션 중간에 마치 게임을 저장하듯 'SAVEPOINT'라는 것을 설정할 수 있습니다. 간단한 트랜잭션의 경우 그냥 ROLLBACK을 하여도 문제는 없지만 트랜잭션의 크기가 크고 시스템의 부하를 많이 주는 트랜잭션인 경우 그냥 ROLLBACK을 하는 경우 처음으로 되돌아 가기 때문에 비효율적일 수 있습니다. 이때 트랜잭션 중간에 SAVEPOINT를 설정하여 처음으로 되돌리는 것이 아닌 SAVEPOINT 설정한 지점 까지만 되돌려서 시스템의 부하를 줄일 수 있습니다.



- 기본 형태

```
-- Oracle

SAVEPOINT 저장점_이름;
```

```
ROLLBACK TO 저장점_이름;
```

```
-- SQL Server

SAVE TRANSACTION 저장점_이름;

ROLLBACK TRANSACTION 저장점_이름;
```

- 예시

```
-- Oracle

SAVEPOINT SVPT1;
INSERT INTO PLAYER VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1);
ROLLBACK TO SVPT1;
```

```
-- SQL Server

SAVE TRAN SVTR1;
INSERT INTO PLAYER  VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1);
ROLLBACK TRAN SVTR1;
```

- 실습

> 다음 쿼리문을 보고 어떤 결과가 나타나는지 먼저 고민해보고 실제 실행 결과와 비교해보세요.

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(10) NOT NULL,
  attr VARCHAR2(10) DEFAULT 'normal'
);
```

```
-- Oracle

INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric');

-- 첫 번째 SAVEPOINT의 이름 설정
SAVEPOINT SP1;

UPDATE pokemon SET name = 'Raichu';

-- 두 번째 SAVEPOINT의 이름 설정
SAVEPOINT SP2;

DELETE FROM pokemon;

-- 세 번째 SAVEPOINT의 이름 설정
SAVEPOINT SP3;

-- 두 번째 SAVEPOINT로 롤백
ROLLBACK TO SP2;
```

▼ LiveSQL

exec is a requirement for LiveSQL. You do not need this prefix in other environments.

LiveSQL 환경에서는 `exec` 를 붙여줘야한다.

다만 현재 LiveSQL 환경에서 `rollback to savepoint point_name` 동작에 이슈가있다.

▼ 4) 트랜잭션의 병렬처리시 문제점

지금까지 트랜잭션에 대한 개념과 특성, 트랜잭션 실행(COMMIT)과 취소(ROLLBACK), 저장(SAVEPOINT)을 위한 TCL 명령어를 배웠습니다. 데이터베이스는 정확한 데이터 유지 및 오류 발생 시 빠르게 복구하고, 일관된 상태를 유지할 수 있도록 다양한 기능을 제공하는데, 그 중심에서 가장 중요한 역할을 하는 것이 바로 트랜잭션입니다.

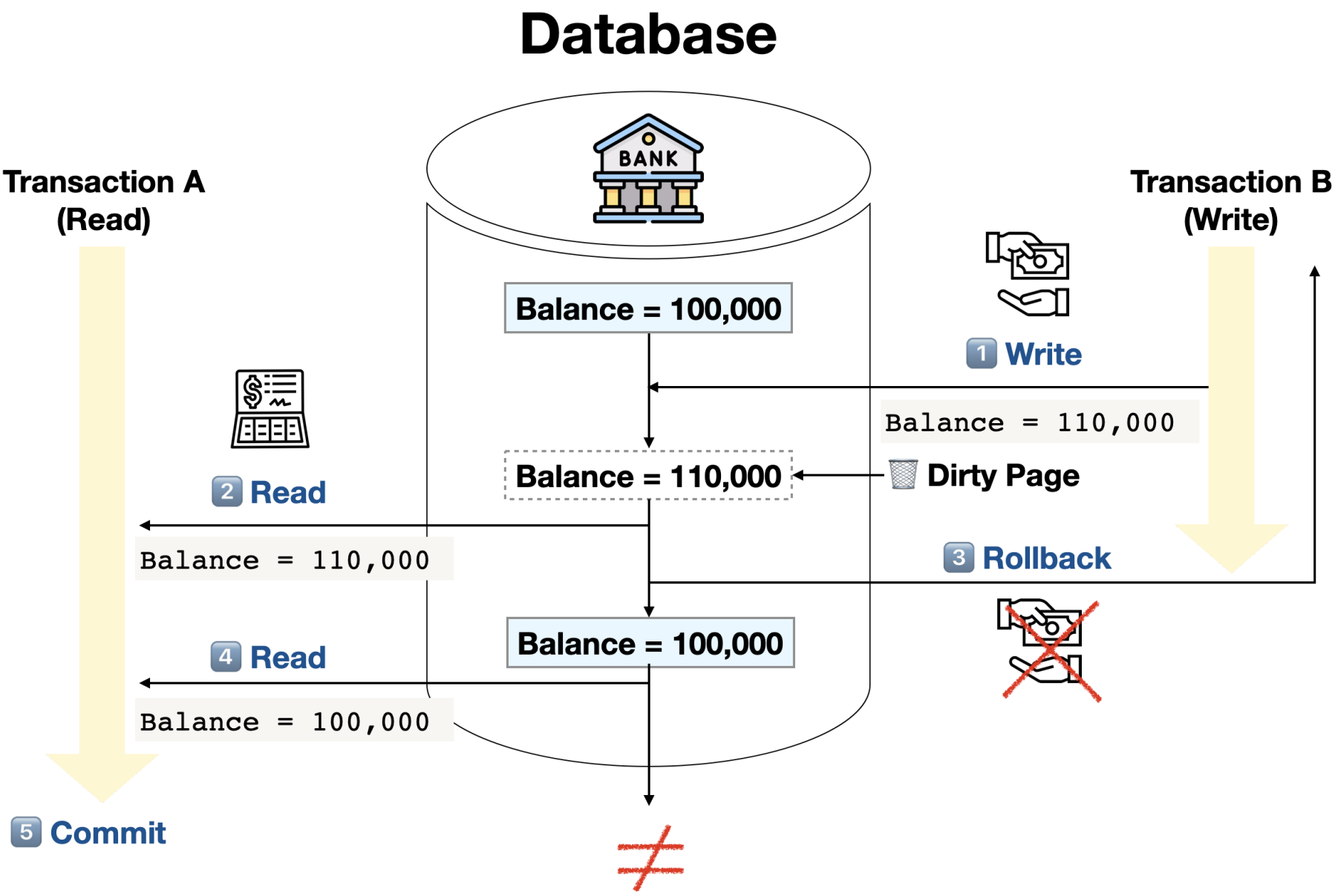
트랜잭션은 기본적으로 수행되는 동안 특정 데이터에 대해서 다른 트랜잭션이 동시에 접근을 하지 못하게 제한을 하는데, 이를 락킹(Locking)이라고 합니다. Locking 덕분에 현재 진행하고 있는 트랜잭션만 독점적으로 데이터에 접근을 할 수 있으며 앞서 설명한 트랜잭션의 4가지 특성 중 원자성, 일관성, 고립성을 보장할 수 있습니다.

살펴본 트랜잭션의 진행 과정 중에는 당연하게도 실행 중인 트랜잭션의 중간 결과에 다른 트랜잭션이 접근을 해서는 안 됩니다. 의도한 연산의 결과가 나오지 않을 수 있기 때문에 일관성과 고립성을 침해할 수 있습니다. 하지만 일부 부여된 권한에 맞게 접근이 되는 경우가 있으며 이때는 반드시 고려해야 하는 상황들이 있습니다. 지금부터 어떤 문제가 발생할 수 있는지 알아 보겠습니다.

< 현재 **수행중인 트랜잭션의 데이터에 다른 트랜잭션이 접근 가능할 때 발생할 수 있는 문제점** >

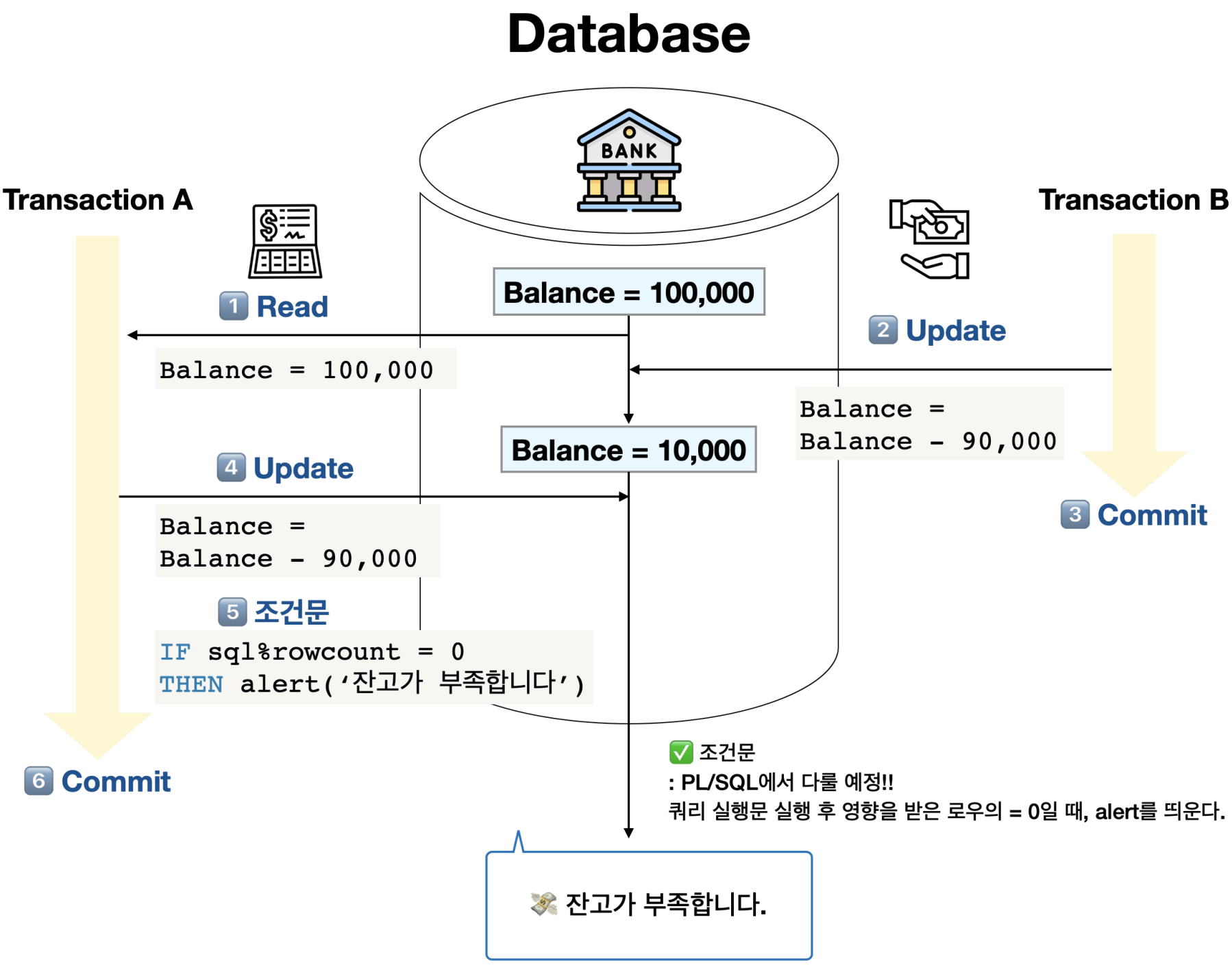
☑️ Dirty Read

- 다른 c트랜잭션에 의해 수정이 되었지만 아직 커밋되지 않은 데이터를 읽는 경우가 발생할 수 있습니다. 이때 중간에 트랜잭션이 취소되는 경우에는 잘못된 정보를 읽은 상태가 되기 때문에 문제가 발생합니다.
- 예를 들어 살펴보겠습니다. 기존 통장잔고는 100,000원이었습니다. 시점 t1에서 쿼리를 통해 자신의 계좌에 10,000원을 송금버튼을 눌렀으나, 곧바로 t3 시점에서 잘못된 송금임을 인지하고 급하게 중간에 취소를 눌렀습니다. t1과 t3 사이 t2 시점에서의 잘못된 잔고조회가 바로 dirty read입니다. 아직 송금이 완료되지 않은 상태에서 dirty page의 잘못된 잔고를 읽은 상태인 것입니다.



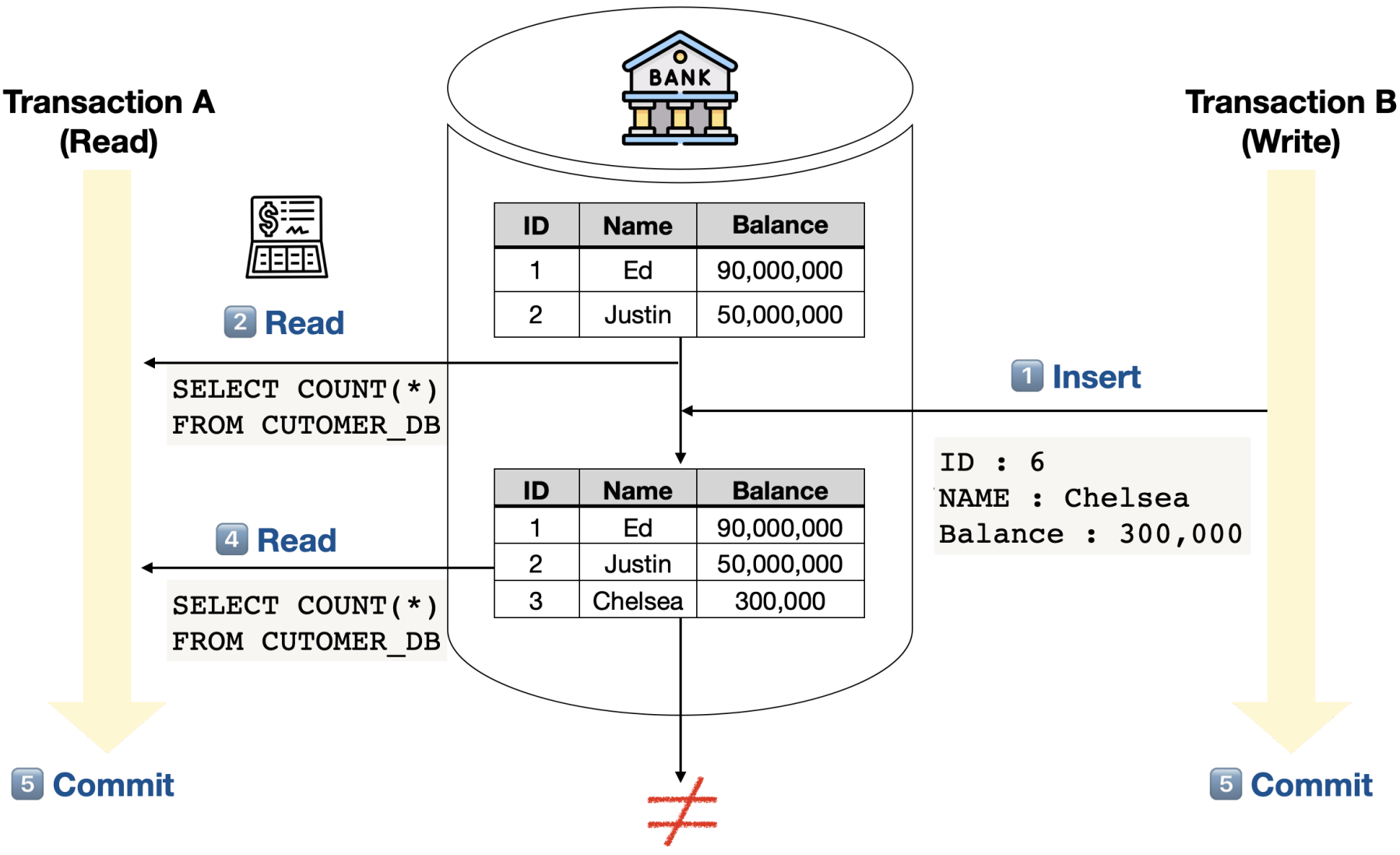
☑️ Non-Repeatable Read

- 한 트랜잭션 내에서 같은 쿼리를 두 번 수행했는데, 그 사이에 다른 트랜잭션이 값을 수정 또는 삭제하는 바람에 두 쿼리 결과가 다르게 나타나는 현상을 의미하는 것을 의미합니다.
- 은행 예시를 다시 들어보겠습니다. t1 시점에 123번 계좌번호의 잔고는 마찬가지로 100,000원이었다고 가정하겠습니다. 조회 쿼리를 통해 자신의 계좌에 100,000원이 남아 있음을 확인하고 t4 시점에 50,000원을 인출하려는데, 중간에 트랜잭션B에 의해 이 계좌의 잔고가 10,000원으로 변경되었습니다. 그러면 트랜잭션 A 사용자는 잔고가 충분한 것을 확인하고 인출을 시도했음에도 불구하고 t2 조건 잔고가 부족하다는 메시지를 받게 됩니다. 트랜잭션 내에서 한 번 읽은 데이터가 트랜잭션이 끝나기 전에 변경이 되었다면 다시 읽었을 때 동일한 값이 아닌 새로운 값이 읽히는 문제가 발생한 것입니다.



☑️ Phantom Read

- 같은 쿼리를 두 번 수행했는데, 중간에 다른 트랜잭션으로 인해 값이 추가가 되어 적용이 된다면 첫 번째 쿼리에서 없던 유령(Phantom) 데이터가 두 번째 쿼리 결과로 나타나는 문제점이 발생합니다.
- 고객 테이블에서 트랜잭션A가 고객 수를 연속해서 집계하는 도중에 새로운 고객이 트랜잭션B에 의해 등록된 상황입니다. 그 결과, 같은 쿼리를 수행했음에도 서로 결괏값이 다른 상태에 놓이게 됩니다.



▼ 연습문제

☑️ 문제 1

Q. 문제	**트랜잭션의 특성에 대한 설명으로 옳지 않은 것은?**
A. (1)	원자성은 정의된 연산들이 실패할 경우 아무것도 수행하지 않은 상태로 남아 있어야 한다
A. (2)	일관성은 트랜잭션 수행 전과 후의 상태에 잘못된 내용이 없어야 한다
A. (3)	고립성은 트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받으면 즉시 작업을 중지하여 잘못된 결과를 만들지 않는다
A. (4)	지속성은 특정 시간 동안만 유지되고 다시 돌아가는 불안정한 부분이 있어서는 안된다

▼ 정답

(3) 고립성은 트랜잭션이 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안 된다

☑️ 문제 2

Q. 문제	**COMMIT과 ROLLBACK의 장점으로 옳지 않은 것은?**
A. (1)	영구적인 변경을 미연에 방지할 수 있다
A. (2)	논리적으로 연관된 작업을 그룹핑 하여 처리할 수 있다
A. (3)	영구적인변경을하기전에데이터의변경사항을확인할수있다
A. (4)	데이터의 무결성을 보장한다

▼ 정답

(1) COMMIT으로 영구적인 변경을 할 수 있도록 한다

☑️ 문제 3

Q. 문제	**트랜잭션은 연산에 대하여 전부 혹은 전무 실행만이 존재하며, 일부 실행으로는 트랜잭션의 기능을 가질 수 없다는 트랜잭션의 특성은?**
A. (1)	Durability

A. (2)	Isolation
A. (3)	Atomicity
A. (4)	Consistency

▼ 정답

(3) 해당 문제의 내용은 트랜잭션의 특성 중 원자성을 설명한 글이다

04. DCL

✔️ **특정 유저의 권한을 제어할 수 있는 명령어 DCL에 대해 학습합니다.**

▼ 1) DCL 개요

☑️ DCL 개요

우리는 지금까지 테이블 생성과 조작에 관련된 DDL, 데이터를 조작하기 위한 DML, 그리고 트랜잭션을 제어하기 위한 TCL 문장을 살펴봤습니다. 이번 레슨에서는 DCL(Data Control Language)에 대해 알아보겠습니다.

DCL은 특정 유저의 권한을 제어할 수 있는 명령어로, 권한을 부여할 수 있는 GRANT문과 권한을 회수할 수 있는 REVOKE문이 있습니다. DDL과 마찬가지로 COMMINT, ROLLBACK이 필요없이 실행하는 순간 적용됩니다.

💡 **사용자 및 스키마**

- 사용자(User)는 사용자 계정과 패스워드를 통해 DBMS에 접속할 수 있습니다
- 스키마(Schema)는 사용자가 소유하고 있는 오브젝트의 모음입니다.

☑️ DCL 구문 정리

종류	SQL문
사용자 계정 생성	CREATE USER user IDENTIFIED BY “password”;
사용자 비밀번호 변경	ALTER USER user IDENTIFIED BY “password”;
사용자 계정 잠금/해제	ALTER USER user ACCOUNT LOCK/UNLOCK;
사용자 계정 삭제	DROP USER user [CASCADE];
사용자에 권한 부여	시스템 권한 GRANT privilege TO user [WITH ADMIN OPTION]; 오브젝트 권한 GRANT privilege ON object TO [WITH GRANT OPTION]; *privilege : 부여할 권한* *object : 대상 오브젝트* *WITH ADMIN/GRANT OPTION : 부여 받은 권한을 다른 사용자에게 부여하거나 회수할 수 있음*
사용자의 권한 회수	시스템 권한 REVOKE privilege FROM user; 오브젝트 권한 REVOKE privilege ON object FROM user;

* ***CASCADE란?***

CASCADE는 DB의 값을 수정/삭제할 때, 해당 값을 참조하고 있는 레코드 역시 종속적으로 수정/삭제를 가능하게 합니다. 따라서 DROP USER문과 함께 사용 시 사용자가 소유한 모든 오브젝트를 함께 삭제합니다.

▼ 2) 유저와 권한

데이터를 특정한 집단 내부에서만 공유하고 사용하는 경우는 문제 될 만한 일이 크게 없습니다. 하지만 데이터를 공유하기 위해 기존 운영자가 아닌 새로운 대상에게 데이터베이스를 오픈해야 하는 경우가 발생하게 됩니다. 이때, DCL을 이용하여 유저에 권한을 부여하여 데이터를 보호할 수 있습니다. 데이터에 대한 보호와 보안 차원에서 기존 유저를 오픈하는 것이 아닌, 새로운 유저를 생성하고 새로 필요한 테이블 혹은 객체에 대한 접근 권한만을 부여하는 것입니다. 권한(Privileges)이란 특정 타입의 SQL문을 실행하거나 데이터베이스나 객체에 접근할 수 있는 권리를 의미합니다.

오라클 DBMS는 기본적으로 아래와 같은 기본 유저를 제공합니다.

- 오라클 기본 유저

유저	설명	default PW
SYS	DBA ROLE을 부여받은 최상위 유저 → DB생성과 제거 가능	CHANGE_ON_INSTALL
SYSTEM	DBA ROLE을 부여받은 유저 → DB생성과 제거 불가능	MANAGER

(이외에 테스트용 샘플 유저가 더 추가되어있다)

Oracle과 SQL Server의 사용자에 대한 아키텍처는 다른 면이 많습니다. 먼저, Oracle 접근 방식에 대해 알아보겠습니다. Oracle은 유저를 통해 데이터베이스에 접속을 하는 형태입니다. 즉, 아이디와 비밀번호 방식을 통해 인스턴스에 접속을 하고 그에 해당하는 스키마에 오브젝트 생성 등의 권한을 부여받는 구조입니다.

이에 반해 SQL Server는 유저를 생성하기 전 먼저 로그인을 생성해야 합니다. 유저를 생성한 후 로그인과 유저를 매핑해 주어야 합니다. 로그인을 생성할 수 있는 권한을 가진 로그인은 기본적으로 sa입니다. 특정 유저는 특정 데이터베이스 내의 특정 스키마에 대해 권한을 부여받을 수 있습니다.

💡 ❕ **참고 - SQL Server 로그인 방식**

- Windows 인증 방식으로 Windows에 로그인한 정보를 가지고 SQL Server에 접속하는 방식 (트러스트 된 연결)
- 혼합 모드(Windows 인증 또는 SQL 인증) 방식으로 기본적으로 Windows 인증으로도 SQL Server에 접속 가능

이제 유저 생성에 대한 두 방식의 차이를 살펴보겠습니다.

- Oracle 유저 생성 및 권한 부여 과정

```
--1. Oracle SQL Plus 사용
CONN SCOTT/TIGER;

--2. ERROR 발생
CREATE USER PJS IDENTIFIED BY KOREA7;

--3. 유저 생성 권한 부여
CONN SYSTEM/비밀번호;
GRANT CREATE USER TO SCOTT;

--4. 정상 동작
CONN SCOTT/TIGER;
CREATE USER PJS IDENTIFIED BY KOREA7;

--5. ERROR 발생
CONN PJS/KOREA7;

--6. 로그인 권한 부여
CONN SYSTEM/비밀번호;
```

```
GRANT CREATE SESSION TO PJS;

--7. 정상 동작
CONN PJS/KOREA7;

--8. ERROR 발생
CREATE TABLE MENU (MENU_SEQ NUMBER NOT NULL, TITLE VARCHAR2(10));

--9. 테이블 생성 권한 부여
CONN SYSTEM/비밀번호;
GRANT CREATE TABLE TO PJS;

--10. 정상 동작
CONN PJS/KOREA7;
CREATE TABLE MENU (MENU_SEQ NUMBER NOT NULL, TITLE VARCHAR2(10));
```

- SQL Server 유저 생성 및 권한 부여 과정

```
--1. 유저 매핑
CREATE LOGIN PJS WITH PASSWORD='KOREA7', DEFAULT_DATABASE=AdventureWorks

--2. 데이터베이스로 이동하여 유저 생성
USE ADVENTUREWORKS;
GO CREATE USER PJS FOR LOGIN PJS WITH DEFAULT_SCHEMA = dbo;

--3. ERROR 발생
CREATE TABLE MENU (MENU_SEQ INT NOT NULL, TITLE VARCHAR(10));

--4. 권한 부여
GRANT CREATE TABLE TO PJS;

--5. 스키마에 권한 부여
GRANT Control ON SCHEMA::dbo TO PJS

--6. 정상 동작
CREATE TABLE MENU (MENU_SEQ INT NOT NULL, TITLE VARCHAR(10));
```

▼ 3) 권한

사용자가 실행하는 모든 DDL 문장은 그에 해당하는 적절한 권한이 있어야만 문장을 실행할 수 있습니다. 권한은 시스템 권한, 객체 권한, ROLE을 이용한 권한 이렇게 크게 3가지로 분류됩니다. 그럼 시스템 권한부터 자세히 살펴보도록 하겠습니다.

☑️ 시스템 권한

시스템권한은 사용자가 데이터베이스에서 특정 작업을 수행 할 수 있도록 합니다. 먼저 시스템 권한을 부여하는 기본 문법을 살펴본 후, 예시를 통해 자세히 알아보도록 하겠습니다.

- 기본 문법

```
-- 기본 구조

GRANT [system_privilege|role] TO [user|role|PUBLIC]
[WITH ADMIN OPTION];
```

  - system_privilege
    - 부여할 시스템 권한의 이름

    - role
        - 부여할 데이터베이스 역할의 이름
    - user, role
        - 부여할 사용자 이름과 다른 데이터 베이스 역할 이름
    - PUBLIC
        - 시스템권한, 또는 데이터베이스 역할을 모든 사용자에게 부여 가능
    - WITH ADMIN OPTION
        - 권한을 부여 받은 사용자도 부여 받은 권한을 다른 사용자 또는 역할로 부여 가능

- Oracle 예시
    - 시스템 권한 부여

```
-- SYS 권한으로 접속합니다.
CONN sys/비밀번호

-- scott 사용자에게 사용자를 생성, 수정, 삭제 할 수 있는 권한을 부여하고
-- scott 사용자도 다른 사용자에게 그 권한을 부여 할 수 있도록 권한 부여합니다.
GRANT CREATE USER, ALTER USER, DROP USER TO scott
WITH ADMIN OPTION;
```

    - 시스템 권한 회수

```
-- scott 사용자에게 부여한 생성, 수정, 삭제 권한을 회수합니다.
REVOKE CREATE USER, ALTER USER, DROP USER
FROM scott;
```

☑️ 객체권한

이번에는 특정 유저가 소유한 객체 권한에 대해 알아보겠습니다. 객체권한은 USER가 소유하고 있는 특정 객체를 다른 사용자들이 엑세스하거나 조작할 수 있게 하기 위해 생성합니다. 객체 권한을 통해 특정 객체인 테이블 등에 대한 SELECT, INSERT, DELETE, UPDATE 작업 명령어를 수행할 수 있게 됩니다.

모든 유저는 각각 자신이 생성한 테이블 외에 다른 유저의 테이블에 접근하려면 해당 테이블에 대한 오브젝트 권한을 소유자로부터 부여받아야 합니다. 객체 소유자는 다른 사용자에게 특정 객체권한을 부여할 수 있으며 기본적으로 소유한 객체에 대해서는 모든 권한이 자동적으로 획득됩니다.

	Both			Oracle	SQL Server
객체권한	테이블	뷰	프로시저	시퀀스	Function
ALTER	○			○	○
DELETE	○	○			○
EXECUTE			○		
INDEX	○				
INSERT	○	○			
SELECT	○	○		○	○
UPDATE	○	○			

객체 권한과 오브젝트와의 관계

▼ SQL Server

SQL Server도 Oracle과 같은 방식으로 동작합니다. 한 가지 다른 점은 SQL Server의 경우 유저는 단지 스키마에 대한 권한만을 가진다는 점입니다. 다시 말하면 테이블과 같은 오브젝트는 유저가 소유하는 것이 아니고 스키마가 소유를 하게 되며 유저는 스키마에 대해 특정한 권한을 가지는 것입니다. 다른 유저가 소유한 객체에 접근하기 위해서는 객체 앞에 객체를 소유한 유저의 이름을 붙여서 접근해야 합니다.

```
-- 예시 [유저.객체]
GRANT SELECT, UPDATE ON A_User.TB_A TO B_User;
```

1. 객체 권한 부여

객체에 대한 권한 부여를 어떻게 하는지 문법과 예시를 통해 확인하겠습니다. 위의 표에서 맨 왼쪽에 있는 ALTER, DELETE, EXECUTE.. 등등은 object_privilege란에 오면 되고, 맨 윗줄에 있는 테이블, 뷰, 시퀀스, 프로시저 등은 ON 다음에 있는 object에 입력하면 됩니다.

- 기본 문법

```
GRANT object_privilege [column]
ON object
TO {user[.user]|role|PUBLIC}
[WITH GRANT OPTION];
```

  - object_privilege
    - 부여할 객체권한의 이름
  - object
    - 객체명
  - user, role
    - 부여할 사용자 이름과 다른 데이터 베이스 역할 이름
  - PUBLIC
    - 객체권한, 또는 데이터베이스 역할을 모든 사용자에게 부여 가능
    - PUBLIC으로 권한을 부여하면 회수할 때도 PUBLIC으로 설정
  - WITH GRANT OPTION
    - 권한을 부여 받은 사용자도 부여 받은 권한을 다른 사용자 또는 역할로 부여 가능

예시를 통해 확인해 봅시다.

- Oracle 예시

```
-- scott USER에게 emp테이블을 SELECT, INSERT할 수 있는 권한을 부여합니다.
-- scott USER도 다른 USER에게 그 권한을 부여 할 수 있습니다.
GRANT SELECT, INSERT
ON emp
TO scott
WITH  GRANT  OPTION;
```

2. 객체 권한의 회수

객체 권한을 철회하는 것은 해당 권한을 부여한 유저만이 수행할 수 있습니다.

- 기본 문법

```
REVOKE {privilege[.privilege] | ALL}
ON object
```

```
FROM {user[.user] | role | PUBLIC}
[CASCADE CONSTRAINTS];
```

- CASCADE CONSTRAINTS
  - 참조 객체 권한에서 사용 된 참조 무결성 제한을 같이 삭제 할 수 있습니다.
- WITH GRANT OPTION
  - 객체 권한을 부여한 사용자의 객체 권한을 철회하면, 권한을 부여받은 사용자가 부여한 객체 권한 또한 같이 철회되는 종속철회가 발생합니다.

- Oracle 예시

```
-- scott USER에게 부여한 emp 테이블에 대한 SELECT, INSERT 권한 회수합니다.
-- 만약 scott USER가 다른 사용자에게 SELECT, INSERT권한을 부여했으면 그 권한들도 같이 회수됩니다.
REVOKE SELECT, INSERT ON emp FROM scott;
```

💡 ❕ `WITH ADMIN OPTION` **VS** `WITH GRANT OPTION`

`WITH ADMIN OPTION`

- 시스템 권한을 가진 계정인 DB만 부여가 가능합니다.
- 자신이 부여받은 권한에 대해서 다른 계정의 사용자에게 권한을 부여할 수 있습니다.
- 권한 회수 시 회수한 유저의 권한만 회수됩니다.

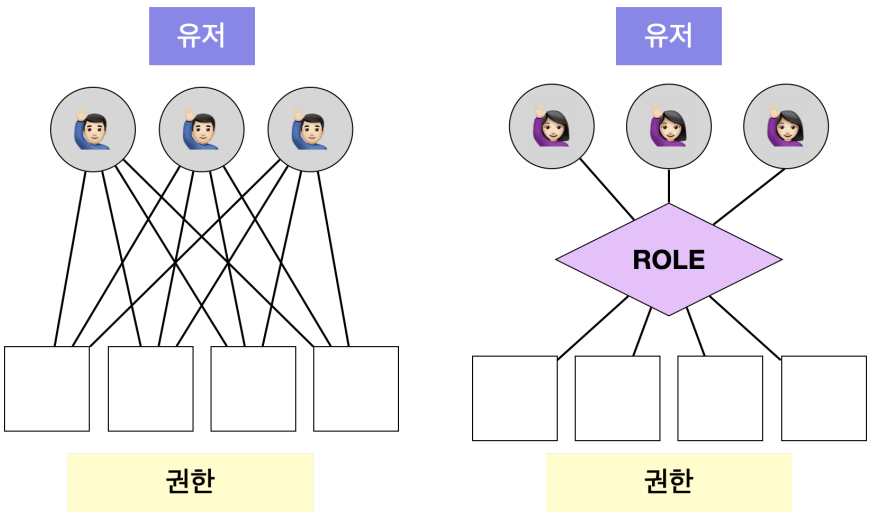
`WITH GRANT OPTION`

- 자신이 부여받은 권한에 대해서 다른 계정의 사용자에게 권한을 부여할 수 있습니다.
- 권한을 회수할 경우 연쇄적으로 권한이 회수됩니다.

☑️ ROLE

유저를 생성하면 기본적으로 CREATE SESSION, CREATE TABLE 등 많은 권한을 부여해야 합니다. ROLE을 이용하면 권한 부여와 회수를 쉽게 할 수 있습니다.

롤(ROLE)이란 사용자에게 허가할 수 있는 권한들의 집합으로 데이터베이스 관리자는 ROLE을 생성하고 각종 권한들을 부여한 후 이를 다른 ROLE이나 유저에게 부여할 수 있습니다. 회사에서 팀장이라는 ROLE을 만들어 인원 평가 관리하는 권한을 위임하는 것과 같은 맥락입니다. ROLE에 포함되어 있는 권한들이 필요한 유저에게는 해당 ROLE만을 부여함으로써 빠르고 정확하게 필요한 권한을 부여할 수 있게 됩니다.



ROLE을 부여하기 위해선 1️⃣ ROLE을 생성하고 2️⃣ ROLE에 권한을 부여하고 3️⃣ ROLE을 사용자 혹은 ROLE하는 순서를 거치게됩니다.

- 예시

```
CONN SYSTEM/비밀번호;

-- ① ROLE의 생성합니다.
CREATE ROLE LOGIN_TABLE;
-- ② ROLE에 권한 부여합니다.
GRANT CREATE SESSION, CREATE TABLE TO LOGIN_TABLE;
-- ③ ROLE을 사용자 또는 ROLE에게 부여합니다.
GRANT LOGIN_TABLE TO CHELSEA;
```

이렇게 만들어 놓은 ROLE을 삭제를 하기 위해선 USER를 DROP하는 쿼리를 작성합니다.

- 유저 삭제

```
CONN SYSTEM/비밀번호;

-- CHELSEA가 생성한 오브젝트를 삭제한 이후에 유저를 삭제합니다.
DROP USER CHELSEA CASCADE;
```

지금까지 ROLE을 만들어 사용하는 것이 권한을 직접 부여하는 것보다 빠르고 안전하게 유저를 관리할 수 있는 방법임을 확인해봤습니다. Oracle에서는 기본적으로 몇 가지 ROLE 을 제공하고 있습니다.

먼저 **CONNECT ROLE**입니다. CONNECT ROLE은 사용자가 데이터베이스에 접속 가능하도록 하기 위해서 사전 정의한 가장 기본적인 시스템 권한으로 이 권한이 없으면 해당 유저로 접속이 되지 않습니다.

```
-- CONNECT ROLE (Release 11.2 버전)
CREATE SESSION
```

다음으로 **RESOURCE ROLE**이 있습니다. 사용자가 객체(테이블, 뷰, 인덱스)를 생성할 수 있도록 하기 위해서 시스템 권한을 묶어 놓은 것으로 총 8가지의 부여 권한이 있습니다. 부여권한은 아래와 같습니다.

```
-- RESOURCE ROLE
CREATE TRIGGER, CREATE SEQUENCE, CREATE TYPE , CREATE PROCEDURE,
CREATE CLUSTER, CREATE OPERATOR, CREATE INDEXTYPE, CREATE TABLE
```

✅ 오라클 DBMS에서 제공하는 ROLE 정리

ROLE	부여 권한
CONNECT (접속)	- CREATE SESSION
RESOURCE (객체 생성)	- CREATE CLUSTER - CREATE PROCEDURE - CREATE TYPE - CREATE SEQUENCE - CREATE TRIGGER - CREATE OPERATOR - CREATE TABLE - CREATE INDEXTYPE

▼ 연습문제

☑️ 문제 1

Q. 문제	**테이블의 행을 삭제할 때는 DELETE, 테이블의 모든 데이터를 비울 때는 TRUNCATE를 사용한다. 그렇다면 특정 사용자에게 준 권한을 취소할 때는 어떤 명령을 사용할 것인가?**
A. (1)	DROP
A. (2)	CANCEL
A. (3)	REVOKE
A. (4)	REVERT

▼ 정답

(3)

☑️ 문제 2

Q. 문제	**아래의 <SQL문>은 'DCL' 유저가 가지고 있는 모든 오브젝트 및 유저를 제거하는 SQL문이다. ㉠에 들어갈 키워드로 옳은 것은?** <SQL문> --SYSTEM 유저로 접속 **DROP USER DCL ㉠;**
A. (1)	SESSION
A. (2)	CASCADE
A. (3)	IDENTIFIED
A. (4)	GRANT

▼ 정답

(2) CASCADE를 사용하게 되면 사용자 이름과 관련된 모든 데이터베이스 스키마가 데이터 사전으로부터 삭제되며 모든 스키마 객체들 또한 물리적으로 삭제된다.

☑️ 문제 3

Q. 문제	아래 보기에서 설명하는 <이것>으로 옳은 것은? a. 데이터베이스 관리자는 유저가 생성될 때마다 각각의 권한들을 유저에게 부여하는 작업을 수행해야 한다. b. 간혹 권한을 빠트릴 수도 있으므로 각 유저별로 어떤 권한이 부여되었는지 관리해야 한다. c. <이것>은 이와 같은 문제를 줄이기 위하여 많은 데이터베이스에서 유저들과 권한들 사이에서 중개역할을 하며 <이것>을 생성 후 각종 권한을 유저에게 부여한다.
A. (1)	CREATE
A. (2)	WITH ADMIN OPTION
A. (3)	RESOURCE
A. (4)	ROLE

▼ 정답

(4)