SQL: Advanced Queries

Database Systems

Department of Computer Science, CBNU
Prof. Nasridinov Aziz (aziz@chungbuk.ac.kr)

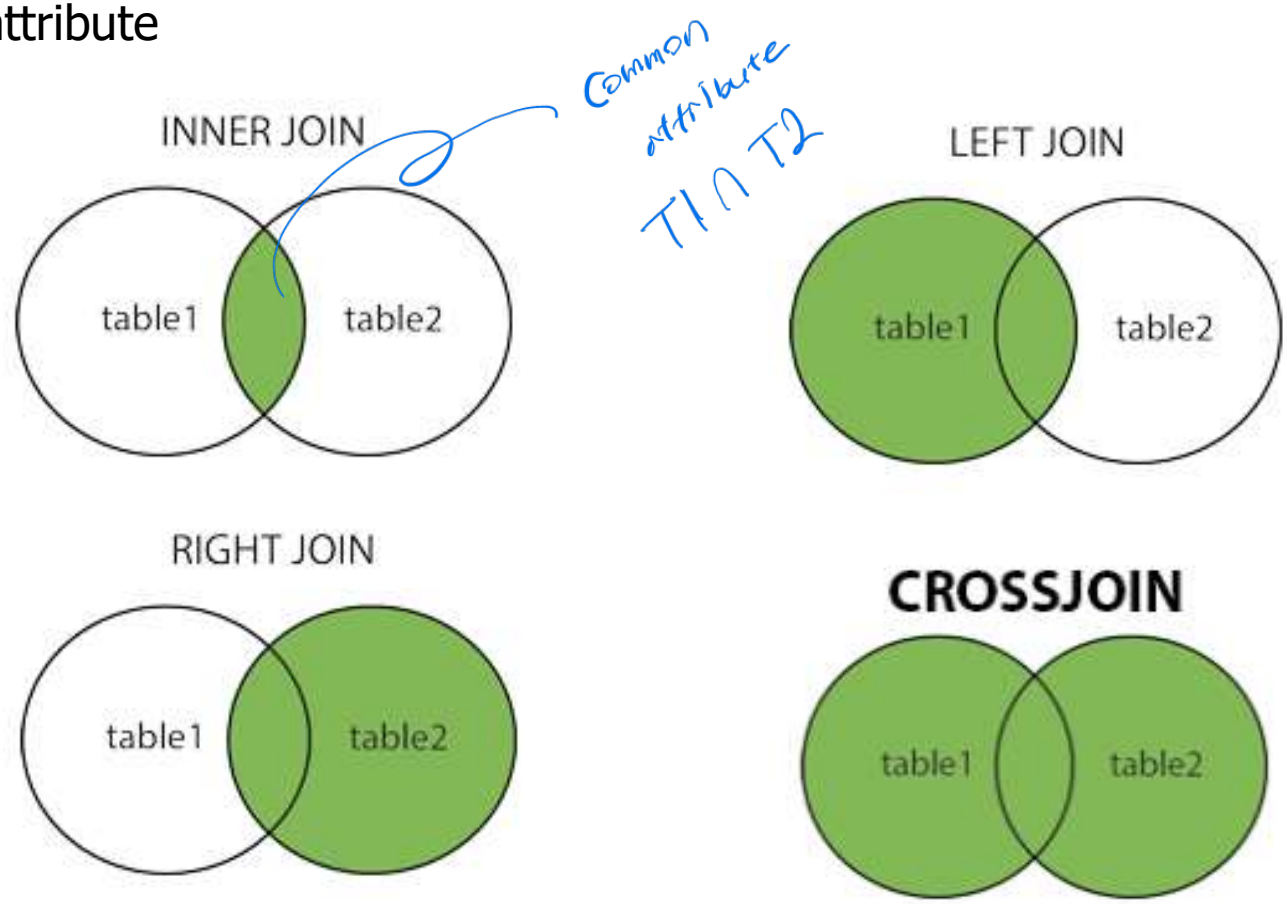
Last Lecture

- MySQL constraints
  - SQL constraints are used to specify rules for the data in a relation

  - MySQL constraints
    - **NOT NULL**
    - **PRIMARY KEY**
    - **FOREIGN KEY**
    - **UNIQUE**
    - **CHECK**
    - **DEFAULT**

Last Lecture

- Join Query
  - Combines tuples from two or more relations based on a common attribute



Last Lecture

- Join Query (Example)
  - Q1: Find the team name for each player

	ADDRESS_ID	ADDRESS_NAME	ZIP_CODE
▶	A0001	Korea	1001
	A0002	England	1002
	A0003	Spain	1003
	A0004	Italy	1004

The *ADDRESS* relation

	TEAM_ID	TEAM_NAME	PHONE	ADDRESS_ID
▶	TM011	Arsenal	12345	A0002
	TM012	Tottenham	23456	A0002
	TM013	Manchester United	23456	A0002
	TM014	FC Seoul	34567	A0001
	TM015	Barcelona	45678	A0003
	TM016	Real Madrid	NULL	A0003
	TM017	Juventus	NULL	A0004

The *TEAM* relation

	PLAYER_ID	PLAYER_NAME	PLAYER_NO	TEAM_ID
▶	P0111	Saka	7	TM011
	P0112	Leno	1	TM011
	P0113	Son	7	TM012
	P0114	Ronaldo	7	TM013
	P0115	Pogba	5	TM013
	P0116	Park	10	TM014
	P0117	Pigue	3	TM015
	P0118	Benzema	3	TM016
	P0119	Dybala	10	TM017

The *PLAYER* relation

Last Lecture

- Join Query (Example)
  - Q1: Find the team name for each player
  - Easy Solution
    - **SELECT** Player_Name, Team_ID
    - **FROM** Player;
  - Problem with this solution
    - Only retrieves the Team_ID, but we DO NOT know the team names

Player_Name	Team_ID
Saka	TM011
Leno	TM011
Son	TM012

Last Lecture

- Join Query (Example)
  - Q1: Find the team name for each player
  - Best Solution
    - Join PLAYER and TEAM tables
      - STEP 1: Find the common attributes

PLAYER_ID	PLAYER_NAME	PLAYER_NO	TEAM_ID
P0111	Saka	7	TM011
P0112	Leno	1	TM011
P0113	Son	7	TM012
P0114	Ronaldo	7	TM013
P0115	Pogba	5	TM013
P0116	Park	10	TM014
P0117	Pigue	3	TM015
P0118	Benzema	3	TM016
P0119	Dybala	10	TM017

The *PLAYER* relation

TEAM_ID	TEAM_NAME	PHONE	ADDRESS_ID
TM011	Arsenal	12345	A0002
TM012	Tottenham	23456	A0002
TM013	Manchester United	23456	A0002
TM014	FC Seoul	34567	A0001
TM015	Barcelona	45678	A0003
TM016	Real Madrid	NULL	A0003
TM017	Juventus	NULL	A0004

The *TEAM* relation

Last Lecture

❖ Join Query (Example)

- Q1: Find the team name for each player

JOIN은 즉 common attribute 끼리 묶어서 도출되는 테이블에서 끼리끼리 합쳐서 하나로 나오도록 합쳐는 것이다.

- Best Solution
  - Join PLAYER and TEAM tables
    - STEP 2: Set up the join condition

```
SELECT P. Player_Name, T.Team_Name
FROM Player P
JOIN Team T ON P.Team_ID = T.Team_ID;
```

JOIN condition

Last Lecture

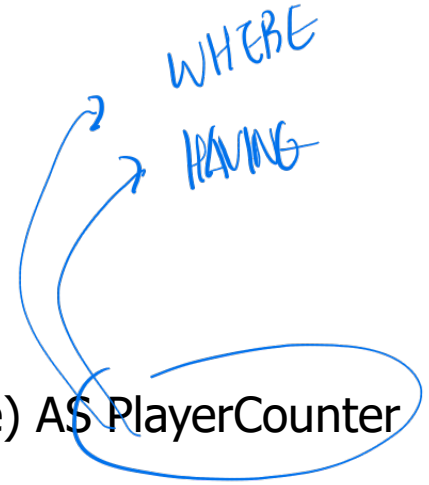
- Join Query (Example)
  - Q2: Find the number of players in each team
  - Best Solution
    - Group the players by teams
      - STEP 1: Find the attribute to make groups

```
SELECT P.Player_Name,
COUNT(T.Team_Name) AS Player
Counter
FROM Player P
JOIN TEAM T ON
P.Team_ID = T.Team_ID
Group By T.Team_Name
```

Player_Name	Team_Name
Saka	Arsenal
Leno	Arsenal
Son	Tottenham
Ronaldo	Manchester United
Pogba	Manchester United
Park	FC Seoul
Pique	Barcelona
Benzema	Real Madrid
Dybala	Juventus

Last Lecture

- Join Query (Example)
  - Q2: Find the number of players in each team
  - Best Solution
    - Group the players by teams
      - STEP 2: Add aggregation function
    - **SELECT** P. Player_Name, count(T.Team_Name) AS PlayerCounter
    - **FROM** Player P
    - **JOIN** Team T ON P.Team_ID = T.Team_ID
    - **GROUP BY** T.Team_Name;



HAVING

Last Lecture

- Join Query (Example)
  - Q2: Find the number of players in each team
  - Best Solution
    - Group the players by teams

Player_Name	PlayerCounter
Saka	2
Son	1
Ronaldo	2
Park	1
Pique	1
Benzema	1
Dybala	1

Table of Contents

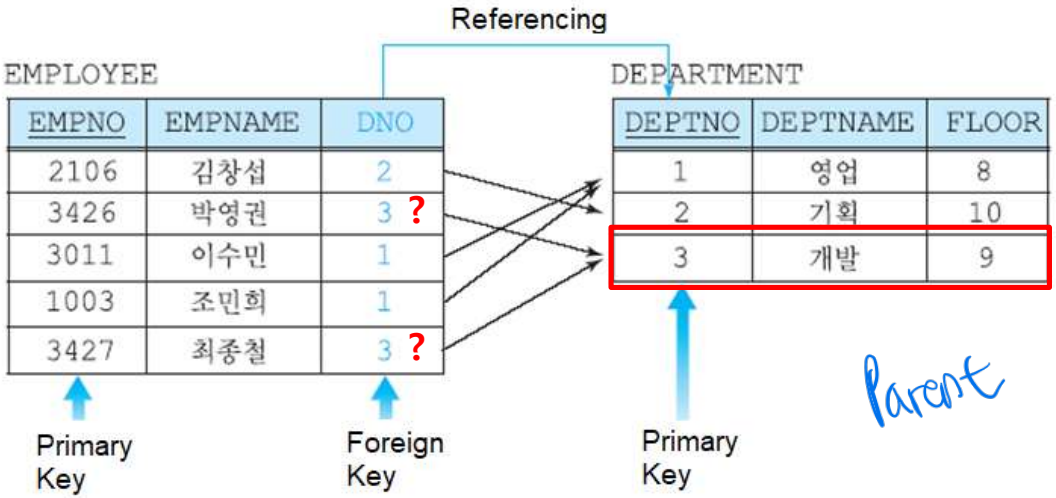
Part 1

ADVANCED DDL

1. Advanced DDL

❖ Referential Actions

- UPDATE or DELETE operation may affect a primary key value in the parent relation that has matching tuples in the child relation
  - Suppose that you want to delete department 3 from DEPARTMENT relation. What will happed to matching tuples in the EMPLOYEE relation?



1. Advanced DDL

- Referential Actions
  - The result of UPDATE and DELETE depends on the referential actions of FOREIGN KEY constraint
    - CASCADE, NO ACTION, RESTRICT, SET NULL, SET DEFAULT
  - Referential actions must be added to the FOREIGN KEY constraint
    - DELETE
      - ON DELETE CASCADE
      - ON DELETE NO ACTION
      - ON DELETE RESTRICT
      - ON DELETE SET NULL
      - ON DELETE SET DEFAULT
    - UPDATE
      - ON UPDATE CASCADE
      - ON UPDATE NO ACTION
      - ON UPDATE RESTRICT
      - ON UPDATE SET NULL
      - ON UPDATE SET DEFAULT

1. Advanced DDL

- Creating a table with referential actions
  - Example

```
CREATE TABLE employee(
        empno       int             NOT NULL,
        empname     varchar(45),
        dno         int,
        CONSTRAINT FK_Department_Employee
        FOREIGN KEY (dno)
        REFERENCES department(deptno) ON DELETE CASCADE
        );
```

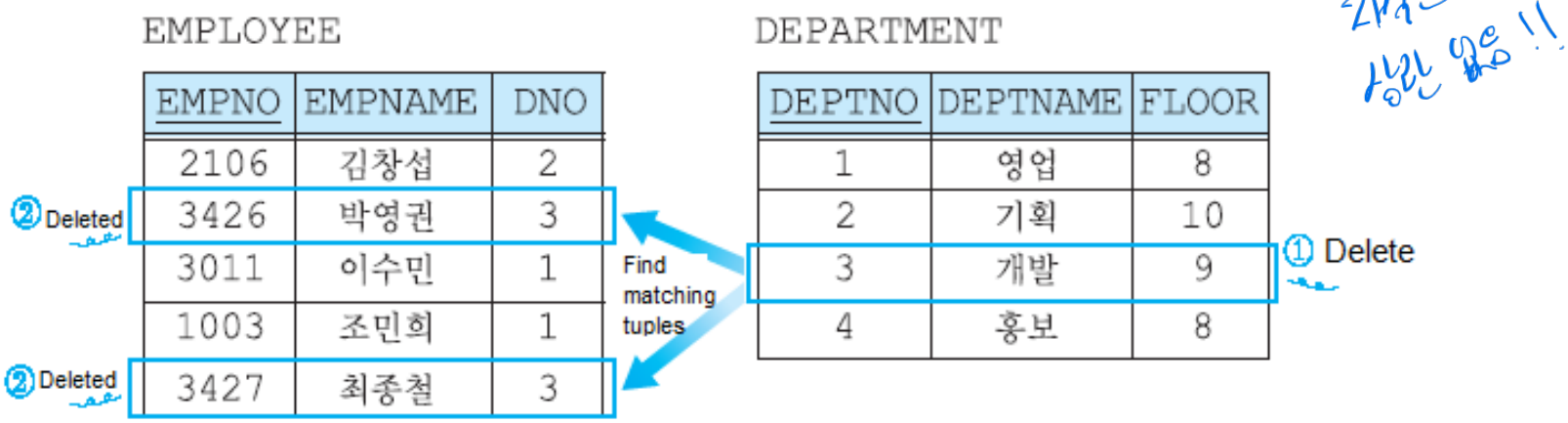
1. Advanced DDL

❖ Referential Actions – CASCADE

- Delete or update the tuples from the parent relation and automatically delete or update the matching tuples in the child relation

- In MySQL, you can specify CASCADE action for delete as follows:
  - ON DELETE CASCADE

부모 튜플 삭제 시
자식 튜플 자동 삭제



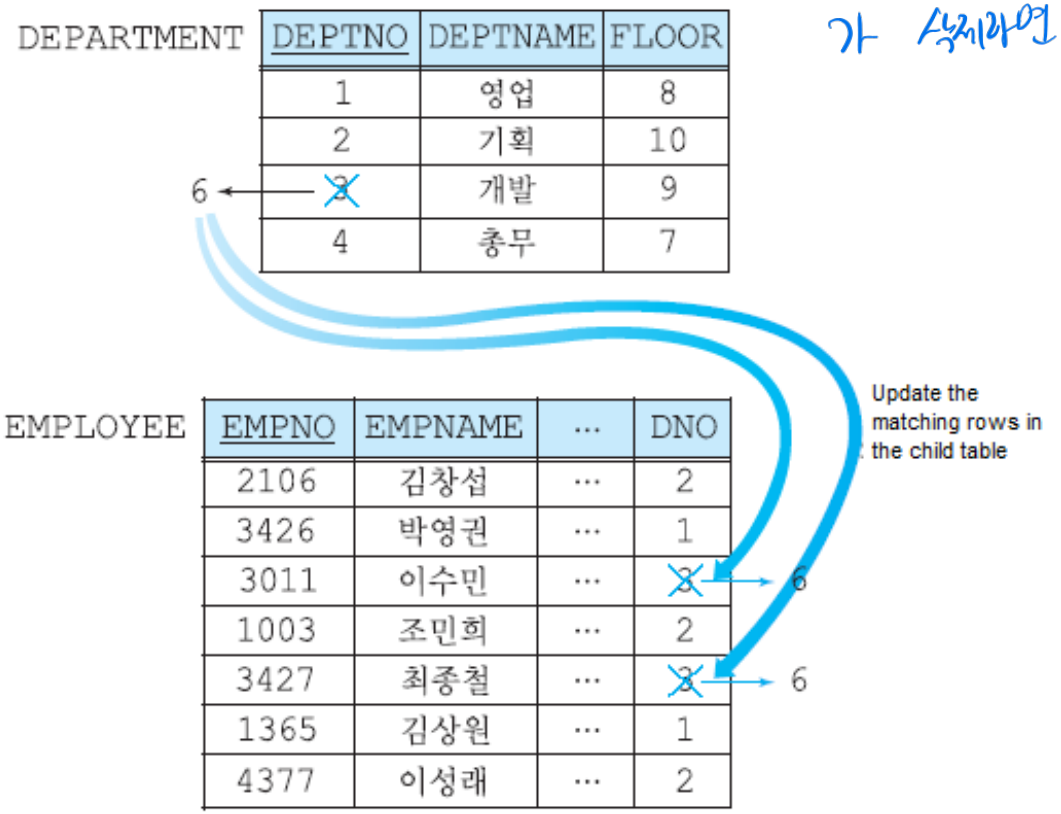
자식은 삭제 안될 수도 있음!!

1. Advanced DDL

❖ Referential Actions – CASCADE

- In MySQL, you can specify CASCADE action for update as follows:
  - ON UPDATE CASCADE

ON DELETE CASCADE
가 삭제되면 같이 업데이트.



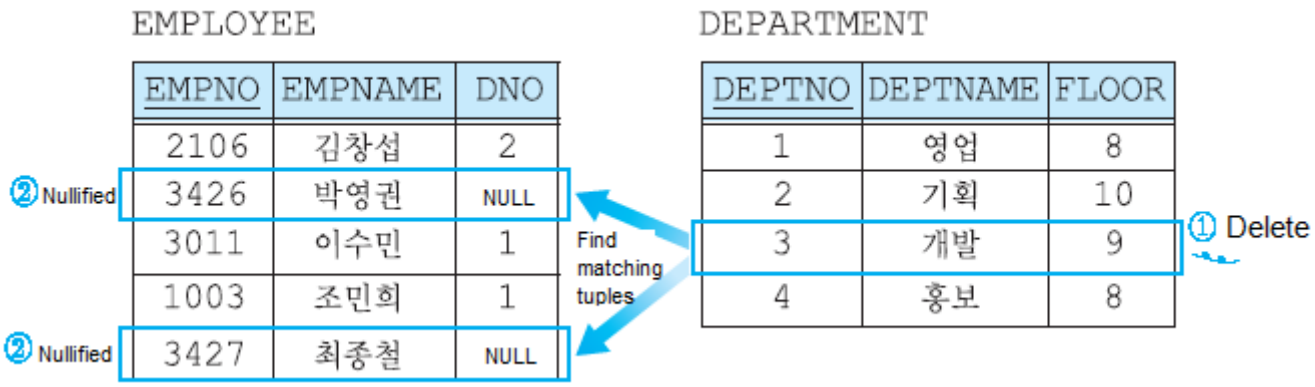
1. Advanced DDL

❖ Referential Actions – SET NULL

- Delete or update the tuples from the parent relation and set the foreign key attribute(s) in the child relation to NULL

- In MySQL, you can specify CASCADE action as follows:
  - ON DELETE SET NULL
  - ON UPDATE SET NULL

부모 튜플 삭제 or 업데이트 시
자식 튜플 값을 NULL로 설정.



1. Advanced DDL

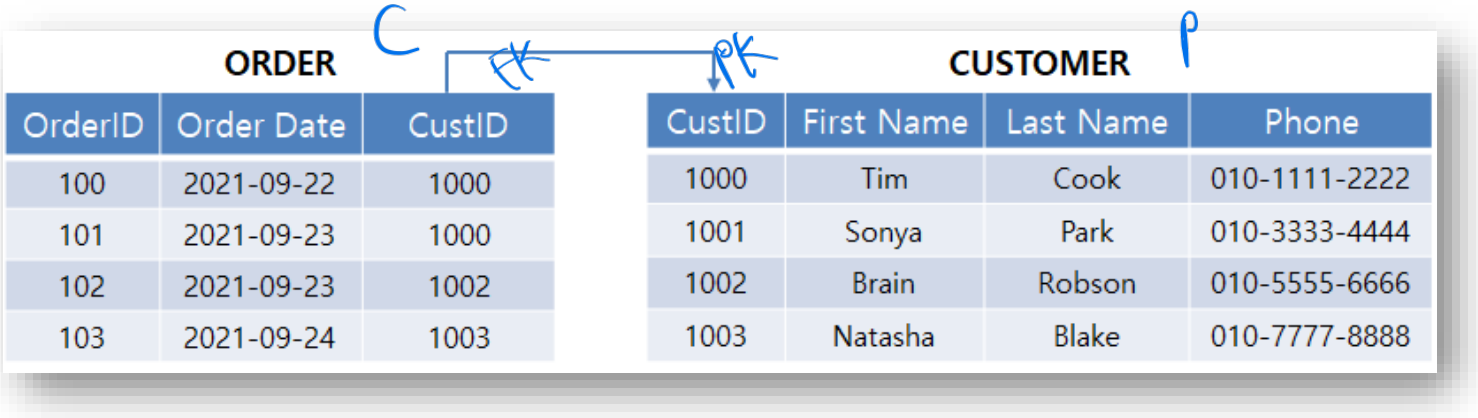
- Referential Actions – RESTRICT and NO ACTION
  - Rejects the delete or update operation for the parent relation
  - In MySQL, both RESTRICT and NO ACTION are equivalent
    - ON DELETE RESTRICT or ON DELETE NO ACTION
    - ON UPDATE RESTRICT or ON UPDATE NO ACTION

삭제나 업데이트 불가

- Referential Actions - SET DEFAULT
  - Currently, MySQL does not support SET DEFAULT action

Quiz

- Quiz
  - Given the following relational database with two relations. Describe what happens to the ORDER relation in the following cases:



ORDER

OrderID	Order Date	CustID
100	2021-09-22	1000
101	2021-09-23	1000
102	2021-09-23	1002
103	2021-09-24	1003

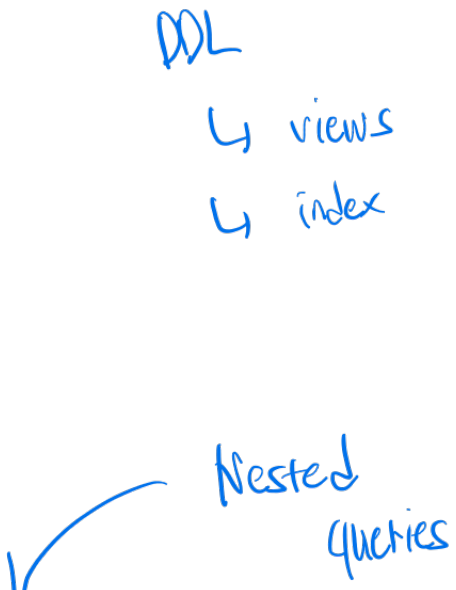
CUSTOMER

CustID	First Name	Last Name	Phone
1000	Tim	Cook	010-1111-2222
1001	Sonya	Park	010-3333-4444
1002	Brain	Robson	010-5555-6666
1003	Natasha	Blake	010-7777-8888

1. Delete customer with ID 1003 (CustID = 1003) from CUSTOMER relation when referential action is CASCADE.
2. Delete customer with ID 1002 (CustID = 1002) from CUSTOMER relation when referential action is RESTRICT.
3. Delete customer with ID 1000 (CustID = 1000) from CUSTOMER relation when referential action is SET NULL.
4. Delete customer with ID 1001 (CustID = 1001) from CUSTOMER relation when referential action is CASCADE.

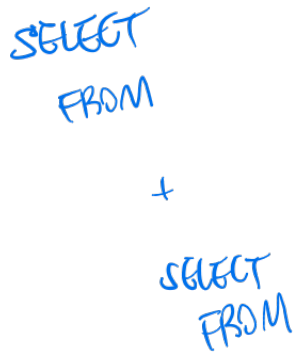
Part 2

ADVANCED DML



2. Intermediate DML

- Nested query
  - SELECT that appears inside another SQL statement
    - Also called subquery
  - Where they can appear
    - WHERE / HAVING (most common)
    - FROM (as a derived table)
    - SELECT list
    - INSERT, UPDATE, DELETE



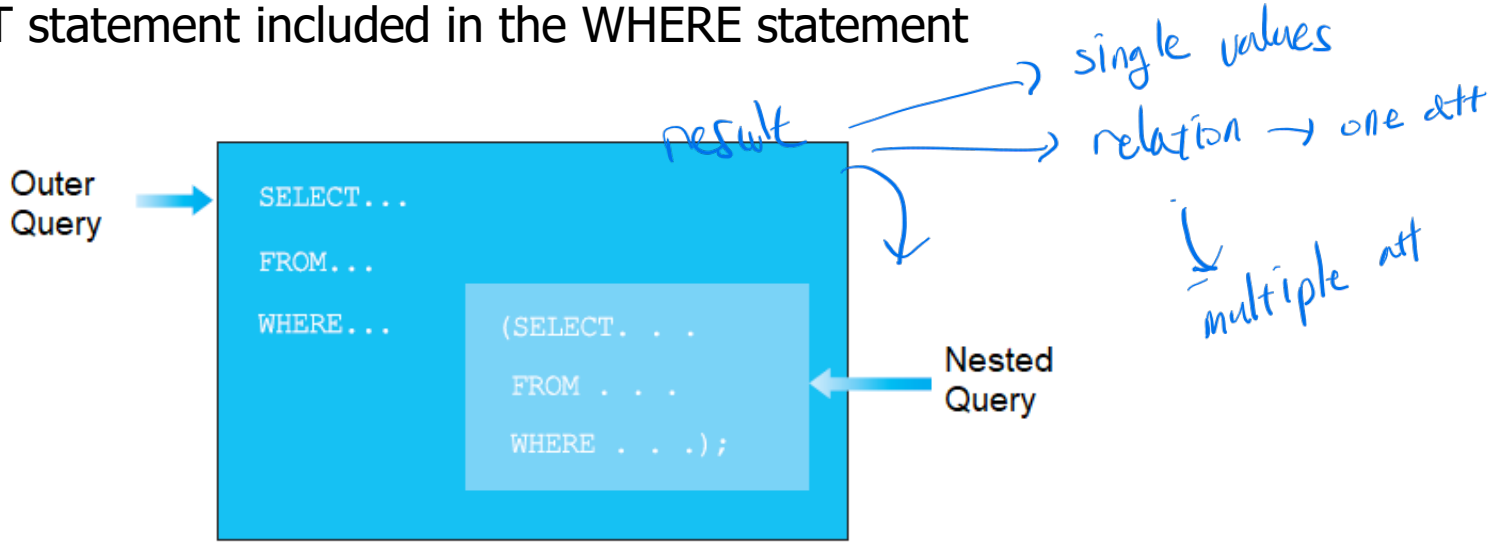
Advanced DML

- Advantages of Nested Queries
  - Improves Readability and Logical Flow
  - Enables Stepwise Filtering
  - Useful for Complex Scenarios

- Disadvantages of Nested Queries
  - Often slower
  - Harder to optimize
  - JOIN is preferred when two tables are queried

2. Intermediate DML

- Nested Query
  - SELECT statement included in the WHERE statement



  - The result of a nested query can return one of the followings
    - A single value
    - A relation with one attribute
    - A relation with multiple attributes

2. Intermediate DML

- Nested Query - When a single value is returned
  - Example: Retrieve the names and titles of all employees with the same title as 박영권.

S2

```
SELECT    EMPNAME, TITLE
FROM      EMPLOYEE
WHERE     TITLE =
          (SELECT    TITLE
           FROM      EMPLOYEE
           WHERE     EMPNAME = '박영권') ;
```

과장

S1

  - The result of example query

EMPNAME	TITLE
박영권	과장
조민희	과장

박영권과 같은 직급 사람 찾기

2. Intermediate DML

WHERE → arithmetic operator

- Nested Query - When a single value is returned
  - Q1: Countries that have the same independence year with South Korea

```
SELECT Name, IndepYear
FROM Country
WHERE IndepYear =
        (SELECT IndepYear
        FROM Country
        WHERE Name = 'South Korea');
```

South Korea의 IndepYear 값을 먼저 찾기

Nested Query를 사용한 예시는 예시와 같이 IndepYear를 모를 때 사용

  - Result

Name	IndepYear
Israel	1948
South Korea	1948
Sri Lanka	1948
Myanmar	1948
North Korea	1948

2. Intermediate DML

- Nested Query – When a relation with one attribute is returned
  - When a relation with one attribute is returned, use IN, ANY, ALL operators in WHERE statement

  - Example
    - Search for the names of employee who work in Sales or Development departments

```
SELECT    EMPNAME
FROM      EMPLOYEE
WHERE     DNO IN (1, 3)
          (SELECT   DEPTNO
           FROM     DEPARTMENT
           WHERE    DEPTNAME = '영업' OR DEPTNAME = '개발') ;
```

2. Intermediate DML

- Nested Query – When a relation with one attribute is returned
  - Note that most of the nested queries can be written using JOIN
  - Previous example can be rewritten as

```
SELECT E.EMPNAME
FROM EMPLOYEE AS E
JOIN DEPARTMENT AS D ON E.DNO = D.DEPTNO
WHERE D.DEPTNAME IN ('영업', '개발');
```

2. Intermediate DML

- Nested Query – When a relation with one attribute is returned
  - Use the subquery form to explain the concept of nested queries
  - Use the JOIN form to show how table relationships can replace subqueries for clarity and performance

  - Nested Queries vs. JOIN queries

Aspect	Subquery with `IN`	JOIN version
Readability	More intuitive for beginners	Clearer relationship between tables
Performance	Usually optimized by DB engine as a join	Often more efficient on large datasets
Teaching purpose	Good for demonstrating nested queries	Good for demonstrating table relationships

2. Intermediate DML

- Nested Query - When a relation with one attribute is returned
  - Q2: Find countries in Asia that have cities with population of more than 5 million

```
SELECT C.Name
FROM Country C
WHERE C.Code IN (SELECT T.CountryCode
                 FROM City T
                 WHERE T.Population > 5000000);
```

```
SELECT C.NAME
FROM Country C
JOIN City T ON C.Code
    = T.CountryCode
WHERE T.Population
    > 5000000;
```

  - Result

Name
Brazil
China
Congo, The Dem...
Colombia
Egypt
United Kingdom
Indonesia
India

2. Intermediate DML

- Nested Query – When a relation with multiple attribute is returned
  - When a relation with multiple attribute is returned, use EXIST operator in WHERE statement

or

NOT EXIST

  - Example
    - Search for the names of employee who work in Sales or Development departments

```
SELECT   EMPNAME
FROM     EMPLOYEE E
WHERE    EXISTS
         (SELECT   *
          FROM     DEPARTMENT D
          WHERE    E.DNO = D.DEPTNO
                   AND (DEPTNAME = '영업' OR DEPTNAME = '개발'));
```

Advanced DML

- Nested Queries (Derived Table)
  - Derived Table
    - Temporary table created from a subquery in the FROM clause
      - It is not stored in the database; it exists only while the outer query runs
  - Syntax
    - **SELECT** …
    - **FROM** (<subquery>) AS derived_table
    - **WHERE** …

Advanced DML

- MySQL Nested Queries (Derived Table)
  - Q3: Find all countries in Europe with population more than 50 million
  - Query

```
SELECT C.Name
FROM (SELECT Name, Continent
      FROM Country
      WHERE Population >50000000)
AS C
WHERE C.Continent = 'Europe' ;
```

Name
Germany
France
United Kingdom
Italy
Russian Federation
Ukraine

Advanced DML

- MySQL Nested Queries (Derived Table)
  - Note that you can create above query using more simple version
  - More simple version

    SELECT Name, Continent

    FROM Country

    WHERE Population >50000000

  - Then, why derived table is useful?
    - It makes the query more simple or clear

Advanced DML

- MySQL Nested Queries (Derived Table)
  - Q4: Count large countries by population and group them by continents

  - Query

```
SELECT Continent, COUNT(*) AS NumLargeCountries
FROM (
    SELECT Name, Continent
    FROM Country
    WHERE Population > 50000000
) AS C
GROUP BY Continent;
```

Derived table

name of derived table

C라는 파생된 테이블.

Continent	NumLargeCountries
Asia	11
South America	1
Africa	4
Europe	6
North America	2

```
SELECT Continent, COUNT(*) AS large_count
FROM ( SELECT NAME, Continent
       FROM Country
       WHERE Population > 50000000
) AS C
GROUP BY Continent;
```

Questions?

SEE YOU NEXT TIME