

# 4. INTERMEDIATE SQL

Join Expressions, View

# JOIN

- **Join** 연산 : 두 개의 릴레이션을 결합하여 하나의 릴레이션을 결과로 넘기는 것
- **Catesian product** 라고 할 수 있음
  - = (어떤 조건 하에서) 두 릴레이션의 튜플을 짝지어줌
  - = 결과에 들어갈 **attribute**를 지정할 수 있음
- **From** 절에서 **Subquery**로 사용됨



# JOIN - 예시

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

*course* 릴레이션

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*prereq* 릴레이션

**course** 릴레이션 = 각 과목의 과목명 / 과목 개설 학과 / 학점 수

**prereq** 릴레이션 = 각 과목의 선수과목

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*course* 릴레이션은 **CS-347**의 정보가 없고, *prereq* 릴레이션은 **CS-315**의 정보가 없음

→ 두 릴레이션을 NATURAL JOIN하면 CS-315, CS-347은 결과에 포함 X

# OUTER JOIN

- Outer Join (외부 조인)

: natural join을 수행할 때, 결과에서 제외된 튜플들을 결과 릴레이션에 포함시킨

Join 연산 (짝을 이루지 못한 튜플의 상대 속성값은 null)

- Inner Join (내부 조인)

: 짝이 없는 튜플을 결과에서 제외시키는 Join (Outer Join과 구별하기 위해)

- Outer Join이 왜 필요한가?

예) 모든 과목의 ID, 이름, 개설학과, 학점 수, 선수과목 등을 알고 싶을 때, natural join을 하면

선수과목이 없는 과목은 결과에서 제외되기 때문에 모든 과목의 정보를 알 수 없기 때문



# LEFT/RIGHT OUTER JOIN

- Natural Left Outer Join

: Left Outer Join 연산의 왼쪽에 나타나는 릴레이션의 튜플을 유지

- Natural Right Outer Join

: Right Outer Join 연산의 오른쪽에 나타나는 릴레이션의 튜플을 유지

- 과정

1. Inner Join

2. 결과에 포함되지 않은 튜플 중 연산자의 왼쪽에 위치한 릴레이션의 튜플들을  
결과 릴레이션에 넣기 (속성값은 **null**로 채우기)

# LEFT/RIGHT OUTER JOIN - 과정

## 1. Inner Join

2. 결과에 포함되지 않은 튜플 중 연산자 왼쪽 릴레이션의 튜플들을 결과 릴레이션에 넣기  
(속성값은 null로 채우기)

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

*course* 릴레이션

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*prereq* 릴레이션

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>

*course* NATURAL LEFT OUTER JOIN *prereq*



# FULL OUTER JOIN

- Natural Full Outer Join

: 두 릴레이션의 모든 튜플 유지 (Left + Right Outer Join)

- 과정

1. Inner Join

2. 결과에 포함되지 않은 튜플을 속성값을 null로 하여 결과에 추가

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

*course* NATURAL FULL OUTER JOIN *prereq*

# ON CONDITION

- ON : JOIN 될 릴레이션에 대한 조건 정하기
- WHERE 절처럼 사용 (WHERE 대신 ON 키워드 사용)
- 조건식의 마지막에 나타남



# WHERE VS ON - 예시

## 1. ON 사용

```
SELECT *  
FROM course JOIN prereq ON course.course_id = prereq.course_id ;
```

## 2. WHERE 절 사용

```
SELECT *  
FROM course, prereq  
WHERE course.course_id = prereq.prereq_id ;
```

1, 2 모두 동일한 결과임

```
mysql> Select * from course1 join prereq on course1.course_id = prereq.course_id;  
+-----+-----+-----+-----+-----+-----+  
| course_id | title       | dept_name | credits | course_id | prereq_id |  
+-----+-----+-----+-----+-----+-----+  
| BIO-301   | Genetics    | Biology   | 4       | BIO-301   | BIO-101   |  
| CS-190    | Game Design | Comp. Sci. | 4       | CS-190    | CS-101    |  
+-----+-----+-----+-----+-----+-----+  
2 rows in set (0.00 sec)  
  
mysql> Select * from course1, prereq where course1.course_id = prereq.course_id;  
+-----+-----+-----+-----+-----+-----+  
| course_id | title       | dept_name | credits | course_id | prereq_id |  
+-----+-----+-----+-----+-----+-----+  
| BIO-301   | Genetics    | Biology   | 4       | BIO-301   | BIO-101   |  
| CS-190    | Game Design | Comp. Sci. | 4       | CS-190    | CS-101    |  
+-----+-----+-----+-----+-----+-----+  
2 rows in set (0.00 sec)
```

# JOIN (1)

- **JOIN** : 2개 릴레이션을 가지고 1개의 릴레이션을 결과로 넘겨주는 연산
- **FROM** 절에서 **Subquery** 형태로 주로 이용됨
- **JOIN condition** : 어떤 튜플끼리 매치할지, 어떤 속성들을 결과에 포함할지 결정하는 조건
  - **NATURAL JOIN** : 이름과 데이터타입이 같은 속성 중 같은 튜플만 매칭 → 새 튜플 생성,  
여러 개가 같은 경우 그것들 모두 동일해야 함
  - **ON** : 조인될 릴레이션에 대한 조건 정하기
  - **USING** : **USING** 절에 쓰인 특정 속성( $A_1, \dots, A_n$ )에 대해서만 **NATURAL JOIN**



## JOIN (2)

- **JOIN type** : 짝이 없는 튜플들을 어떻게 처리할지 정하는 것
  - **INNER JOIN** : 매치되지 않은 튜플을 유지하지 않는 것
  - **LEFT / RIGHT OUTER JOIN**
    - : INNER JOIN의 결과에 포함되지 않은 튜플 중 연산자의 왼쪽 또는 오른쪽에 있는 릴레이션의 튜플을 결과에 유지
  - **FULL OUTER JOIN = LEFT + RIGHT JOIN**
    - : 연산자 양쪽에 있는 릴레이션의 모든 튜플을 결과에 유지

# JOIN – 예시 (1)

## 1. ON 사용 – INNER JOIN

```
course INNER JOIN prereq ON  
course.course_id = prereq.course_id
```

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

Q. NATURAL JOIN과 JOIN ON의 차이?

A. 결과 릴레이션에 공통 속성이 몇 번 나타나는지의 차이

JOIN ON : 2번 ( *course\_id* : 맨 앞, 맨 뒤 속성 )

NATURAL JOIN : 1번



## JOIN – 예시 (2)

### 2. ON 사용 – LEFT OUTER JOIN

```
course LEFT OUTER JOIN prereq ON  
course.course_id = prereq.course_id
```

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	<i>null</i>	<i>null</i>

Q. LEFT OUTER JOIN ON 한 것과 NATURAL LEFT OUTER JOIN의 차이

A. 왼쪽 뿐만 아니라, 오른쪽 릴레이션의 공통 속성도 한 번 더 결과에 나온다는 점.

LEFT OUTER JOIN ON : 공통속성 2번 ( *course\_id* : 맨 앞, 맨 뒤 속성 )

+ 매칭되지 않은 왼쪽 속성에는 해당 오른쪽 속성에 *null* 값

NATURAL LEFT OUTER JOIN : 공통속성 1번 나타남

## JOIN – 예시 (3)

### 3. NATURAL RIGHT OUTER JOIN

*course* **NATURAL RIGHT OUTER JOIN** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

- 결과에 공통속성인 *course\_id* 중복 X
- 오른쪽 릴레이션에 있는 튜플들의 값을 보존하는 것은 동일함



## JOIN – 예시 (4)

### 4. USING 조건 사용 – FULL OUTER JOIN

*course* **FULL OUTER JOIN** *prereq* **USING** (*course\_id*)

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

- 같은 속성이 *course\_id* 밖에 없기 때문에 NATURAL JOIN과 동일한 결과

# VIEW의 필요성

- 모든 사용자가 DB에 저장된 모든 릴레이션(논리적인 모델)을 보는 것은 바람직 **x**
  1. 보안유지 : 특정 데이터를 사용자로부터 숨기기
  2. 효율성 : 사용자 목적에 더 부합하는 릴레이션 제공

예) 교수님들의 이름, 학과만 알고, 연봉은 알 필요가 없는 경우

```
SELECT ID, name, dept_name  
FROM instructor
```

위 쿼리의 결과 릴레이션만 보면 충분함 (연봉은 포함 **x**)



# VIEW

- **View** (뷰) : 실제 릴레이션을 기반으로 만들어진 가상의 릴레이션 (virtual relation)
- 실제 릴레이션과 달리 데이터를 실제로 저장 x, 논리적으로만 존재함
- 실제 테이블의 형태  $x \rightarrow$  쿼리 형태로 저장되어 있음
- 뷰를 사용할 때마다 쿼리를 수행해서 결과(가상 릴레이션)를 만들어냄

# VIEW 정의

- **CREATE VIEW** : 뷰를 정의할 때 사용하는 명령어
- 형태 : **CREATE VIEW** 뷰이름 **AS** <뷰를 만들어낼 쿼리식>
- 뷰는 실제 릴레이션과 동일하게 사용됨 (이름도 동일)
- DB에는 릴레이션 형태의 뷰(쿼리결과)가 아닌, 뷰를 정의하는 쿼리식이 저장  
(릴레이션 형태 ✕ → 쿼리 형태 ○)  
왜? 뷰의 업데이트를 위해 (뷰를 만드는 데 사용된 릴레이션이 수정될 경우)



## VIEW – 예시 (1)

예) salary를 가지고 있지 않은 *instructor*의 뷰 만들기

```
CREATE VIEW faculty AS  
    SELECT ID, name, dept_name  
    FROM instructor
```

예) 생물학과의 모든 교수님 찾기 – 새롭게 만들어진 뷰를 참조하는 예시

```
SELECT name  
FROM faculty    // 앞의 예시에서 만든 뷰 faculty가 생성하는 가상의 릴레이션 사용  
WHERE dept_name = 'Biology'
```

## VIEW – 예시 (2)

예) 학과별로 모든 교수님의 급여 합을 출력하는 예제

```
CREATE VIEW departments_total_salary (dept_name, total_salary) AS
    SELECT dept_name, SUM (salary) // SUM (salary) → total_salary로 이름 지음
    FROM instructor
    GROUP BY dept_name ;
```



## VIEW – 예시 (3)

예) 다른 뷰를 사용하여 뷰를 정의하는 예제

```
CREATE VIEW physics_fall_2009 AS  
    SELECT course.course_id, sec_id, building, room_number  
    FROM course, section  
    WHERE course.course_id = section.course_id  
        AND course.dept_name = 'Physics'  
        AND section.semester = 'Fall'  
        AND section.year = '2009' ;
```

← “다른 뷰 ”

```
CREATE VIEW physics_fall_2009_watson AS  
    SELECT course_id, room_number  
    FROM phsycis_fall_2009  
    WHERE buliding = 'Watson' ;
```

← 위의 뷰를 이용해서 뷰 정의

# MATERIALIZED VIEW

- MATERIALIZED VIEW (실체화 뷰)

: 뷰를 정의하는 쿼리의 결과로 생성된 튜플을 가진 물리적인 테이블을 저장

- MATERIALIZED VIEW MAINTENANCE : 뷰를 최신 상태로 유지하는 것

- 방법 3가지

1. 뷰 정의에 사용된 릴레이션이 수정될 때마다 Materialized view 업데이트
2. 수정 → 업데이트 x / 사용자가 뷰에 접근할 때 뷰 업데이트
3. 릴레이션의 수정이나 사용자의 접근과 관계없이 정기적으로 뷰 수정