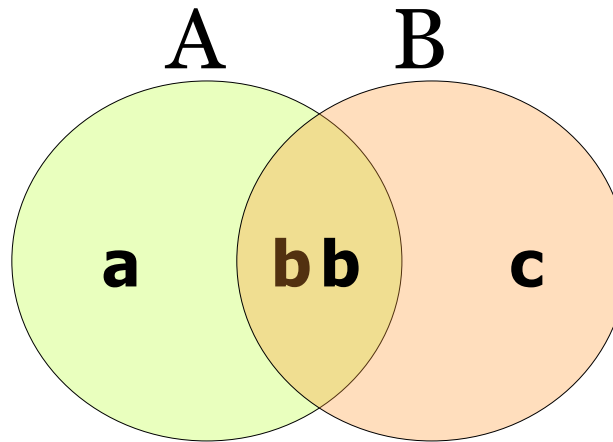


# SQL

## JOIN

# 집합(SET) Operator

- ▶ 두 집합의 결과를 가지고 집합 연산을 수행
- ▶ UNION, UNION ALL, ~~INTERSECT, EXCEPT~~ (MySQL에서는 INTERSECT, EXCEPT 연산은 지원하지 않음)
  - ▶ UNION은 중복 레코드 제거 작업을 진행해야 하기에 UNION보다 속도면에서 느리다
  - ▶ MySQL에서 INTERSECT, EXCEPT 개념을 활용하기 위해 JOIN연산을 활용한다

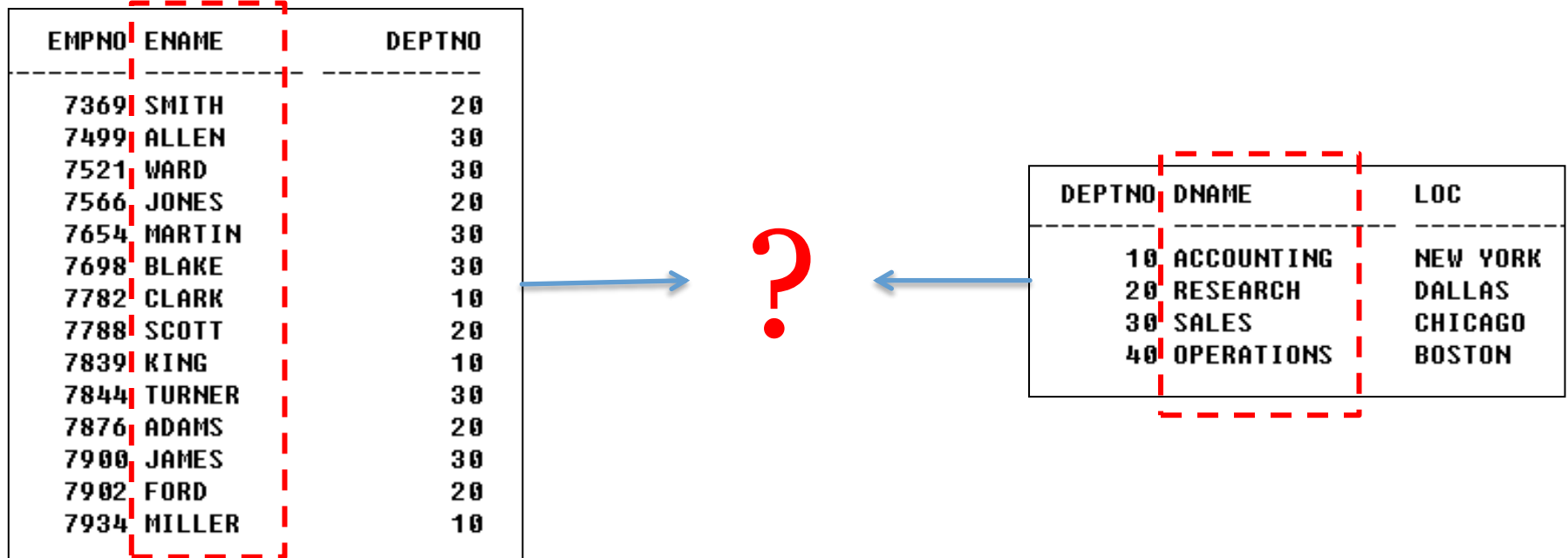


- $A \cup B = \{a, b, c\}$
- $A \cup ALL B = \{a, b, b, c\}$
- ~~$A \cap B = \{b\}$~~
- ~~$A - B = \{a\}$~~

```
SELECT first_name FROM employees
UNION
SELECT department_name FROM departments;
```

# JOIN

- ▶ 둘 이상의 테이블을 합쳐 하나의 큰 테이블로 만드는 방법
- ▶ 필요성
  - ▶ 관계형 모델에서는 데이터의 일관성이나 효율을 위하여 데이터의 중복을 최소화 (정규화)
  - ▶ Foreign Key를 이용하여 참조
  - ▶ 정규화 된 테이블로부터 결합된 형태의 정보를 추출할 필요가 있음
  - ▶ 예) 직원의 이름과 직원이 속한 부서명을 함께 보고 싶다면?



# 카티전 프로젝트

- ▶ 두 테이블에서 그냥 결과를 선택한다면
  - ▶ `SELECT first_name, department_name  
FROM employees, departments;`
  - ▶ 결과 : 두 테이블 행들의 가능한 모든 쌍이 추출
  - ▶ 일반적으로 사용자가 원하는 결과가 아님

- ▶ Cartesian Product

$$X \times Y = \{(x, y) | x \in X \text{ and } y \in Y\}$$

- ▶ Cartesian Product를 막기 위해서는  
올바른 JOIN 조건을 WHERE 절에 부여해야 함

- ▶ 양쪽 테이블로부터 조합 가능한 모든 쌍이 선택되기 때문에  
Cross Join이라 불리기도 함



first_name	department_name
Steven	Administration
Steven	Marketing
Steven	Purchasing
Steven	Human Resources
Steven	Shipping

...

William	IT Helpdesk
William	Government Sales
William	Retail Sales
William	Recruiting
William	Payroll

2889 rows in set (0.01 sec)

`employees (107) * departments (27)`  
`= 2,889`

# Simple Join

## ▶ Syntax

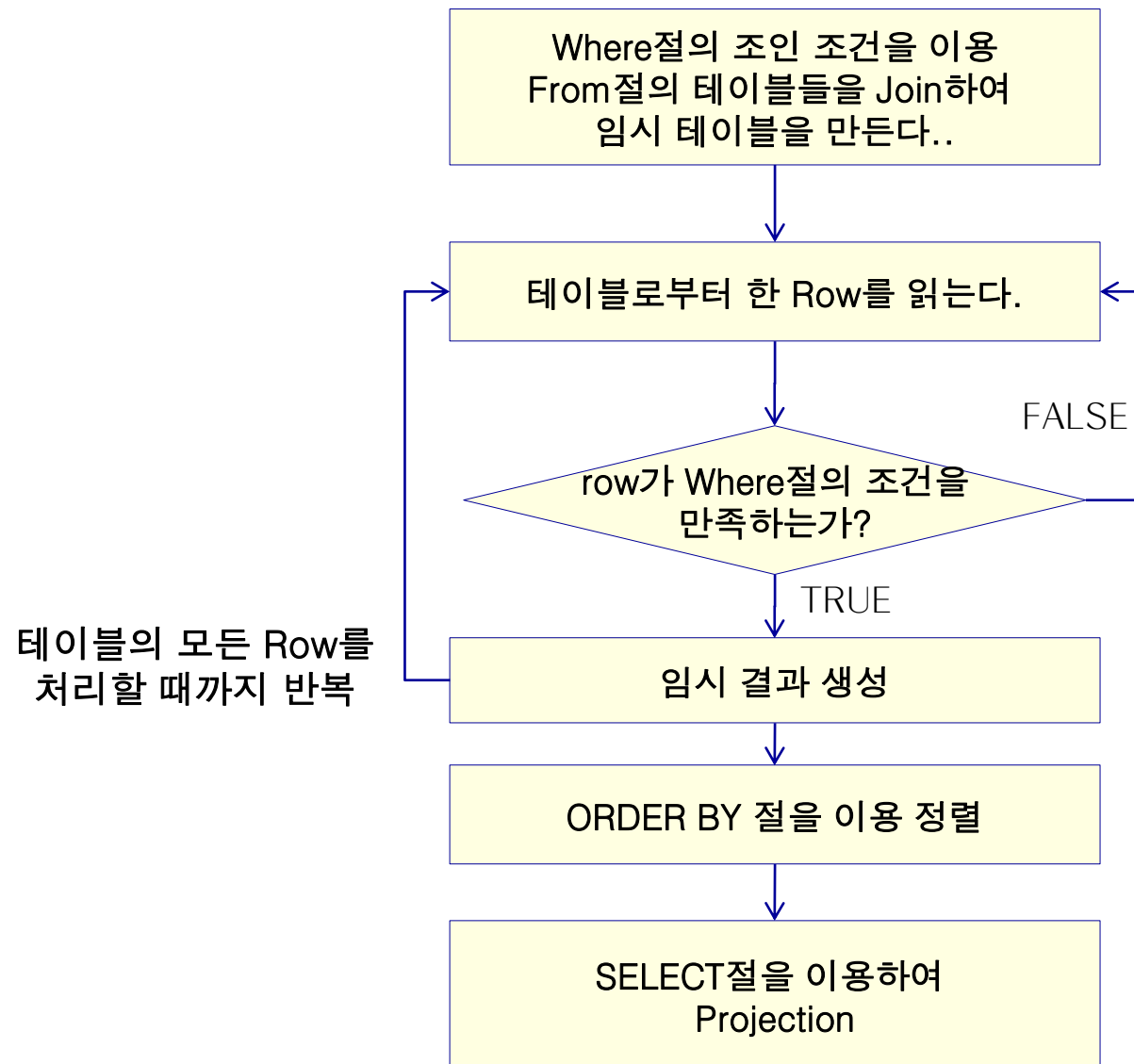
```
SELECT t1.col1, t1.col2, t2.col1 ...  
      FROM Table1 t1, Table2 t2  
      WHERE t1.col3 = t2.col3
```

## ▶ 설명

- ▶ **FROM** 절에 필요한 테이블을 모두 적는다
- ▶ 컬럼 이름의 모호성 (어느 테이블에 속하는지 불명확) 을 피하기 위해 **Table** 명에 **alias**를 사용 (테이블 이름으로 직접 지칭도 가능)
- ▶ 적절한 **Join** 조건을 **WHERE** 절에 부여 (일반적으로 테이블 개수 -1 개의 조인 조건이 필요)
- ▶ 일반적으로 **PK**와 **FK**간의 = 조건이 붙는 경우가 많음

# Join의 처리

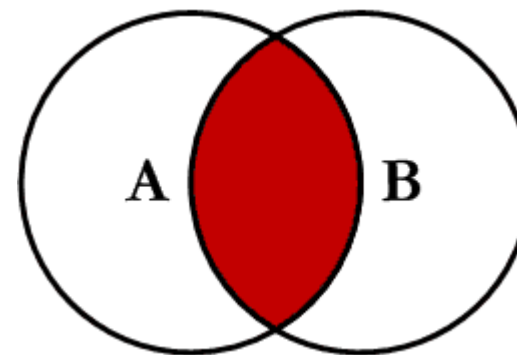
## : Flow



# Join의 종류

## ▶ 용어

- ▶ Cross Join (Cartesian Product) : 모든 가능한 쌍이 나타남
- ▶ Inner Join : Join 조건을 만족하는 튜플만 나타남
  - ▶ Theta Join : 조건(Theta)에 의한 조인
  - ▶ Equi-Join : Theta Join & 조건이 Equal (=)
  - ▶ Natural Join : Equi-Join & 동일한 컬럼명 합쳐짐
- ▶ Outer Join : 조건을 만족하지 않는 튜플(짜이 없는 튜플)도 null과 함께 나타남
- ▶ Self Join : 자기 자신과 조인



INNER JOIN

# JOIN Syntax

- ▶ FROM 절에서 바로 Join을 명시적으로 정의

```
SELECT table1.column, table2.column
FROM table1
    [CROSS JOIN table2] |
    [NATURAL JOIN table2] |
    [JOIN table2 USING (column_name)] |
    [JOIN table2
        ON (table1.column_name = table2.column_name)] |
    [LEFT|RIGHT|FULL OUTER JOIN table2
        ON (table1.column_name = table2.column_name)];
```

- ▶ 예

- ▶ `SELECT * FROM employees JOIN departments USING (department_id);`
- ▶ `SELECT * FROM employees emp JOIN departments dept  
ON emp.department_id = dept.department_id;`
- ▶ `SELECT * FROM employees emp  
RIGHT OUTER JOIN departments dept ON (emp.department_id = dept.department_id);`



# Equi-Join

employee_id	first_name	...	department_id
100	Steven	...	90
101	Neena	...	90
102	Lex	...	90
103	Alexander	...	60
104	Bruce	...	60
105	David	...	60

**EMPLOYEES**

department_id	department_name	...
10	Administration	...
20	Marketing	...
30	Purchasing	...
40	Human Resources	...
50	Shipping	...

**DEPARTMENTS**

FK

PK



employee_id	first_name	...	department_id	department_name	...
100	Steven	...	90	Executive	...
104	Bruce	...	60	IT	...
105	David	...	60	IT	...

```
SELECT * FROM employees emp, departments dept
WHERE emp.department_id = dept.department_id;
```

# Equi-Join

## ▶ [연습] hr.employees and hr.departments

- ▶ employees와 departments를 department\_id를 기준으로 Join 하여 first\_name, department\_id, department\_name을 출력해 봅시다

```
SELECT first_name, em.department_id,  
       de.department_name  
FROM   employees em, departments de  
WHERE  em.department_id = de.department_id;
```

- ▶ 총 몇 건의 ROW가 검색되는지 확인해 봅시다
  - ▶ null은 조인되지 않음을 확인합니다.
  - ▶ 부서를 배정받지 못한 사원(department\_id 가 NULL)은 누구인지 확인해 봅시다

# Theta Join

## 정의

- ▶ 임의의 조건을 Join 조건으로 사용
- ▶ Non-Equi Join이라고도 함
- ▶ Equal(=) 이외의 연산자를 사용하여 Join Condition을 작성한 경우를 일컬음

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	80/12/17	800
7499	ALLEN	SALESMAN	7698	81/02/20	1600
7521	WARD	SALESMAN	7698	81/02/22	1250
7566	JONES	MANAGER	7839	81/04/02	2975
7654	MARTIN	SALESMAN	7698	81/09/28	1250
7698	BLAKE	MANAGER	7839	81/05/01	2850
7782	CLARK	MANAGER	7839	81/06/09	2450
7788	SCOTT	ANALYST	7566	87/04/19	3000
7839	KING	PRESIDENT		81/11/17	5000
7844	TURNER	SALESMAN	7698	81/09/08	1500
7876	ADAMS	CLERK	7788	87/05/23	1100
7900	JAMES	CLERK	7698	81/12/03	950
7902	FORD	ANALYST	7566	81/12/03	3000
7934	MILLER	CLERK	7782	82/01/23	1300

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1
ADAMS	1100	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
TURNER	1500	3
ALLEN	1600	3
CLARK	2450	4
BLAKE	2850	4
JONES	2975	4
SCOTT	3000	4
FORD	3000	4
KING	5000	5

GRADE	LOSal	HISal
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

# Outer Join

## ▶ 정의

- ▶ Join 조건을 만족하지 않는(짜이 없는) 튜플의 경우 Null 을 포함하여 결과를 생성
- ▶ 모든 행이 결과 테이블에 참여

## ▶ 종류

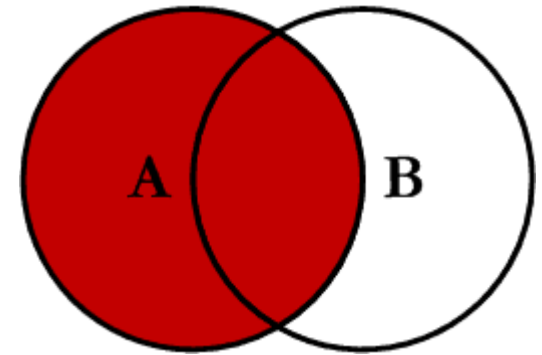
- ▶ Left Outer Join : 왼쪽의 모든 튜플은 결과 테이블에 나타남
- ▶ Right Outer Join : 오른쪽의 모든 튜플은 결과 테이블에 나타남
- ▶ Full Outer Join : 양쪽 모두 결과 테이블에 참여

# Outer Join

## : Left Outer Join

- ▶ 왼쪽 테이블의 모든 row를 결과 테이블에 나타냄

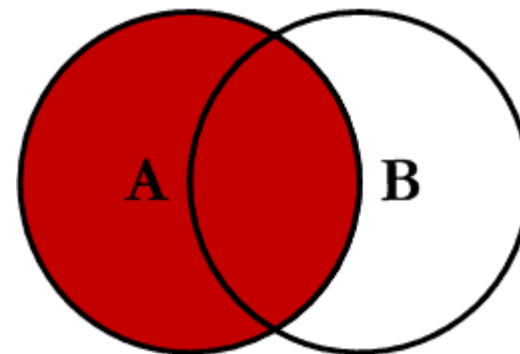
```
SELECT e.department_id, e.first_name, d.department_name  
FROM employees e LEFT OUTER JOIN departments d  
ON e.department_id = d.department_id ;
```



LEFT OUTER JOIN

# Outer Join

## : Left Outer Join



106개

1개

107개

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
100	Steven	90
101	Neena	90
102	Lex	90
108	Nancy	100
109	Daniel	100
110	John	100
111	Ismael	100
112	Jose M...	100
113	Luis	100
205	Shelley	110
206	William	110
178	Kimberely	(null)

DEPARTMENT_ID	DEPARTMENT_NAME
1	Administration
2	Marketing
3	Purchasing
4	Human Resources
5	Shipping
6	IT
7	Public Relations
8	Sales
9	Executive
10	Finance
11	Accounting
12	Treasury

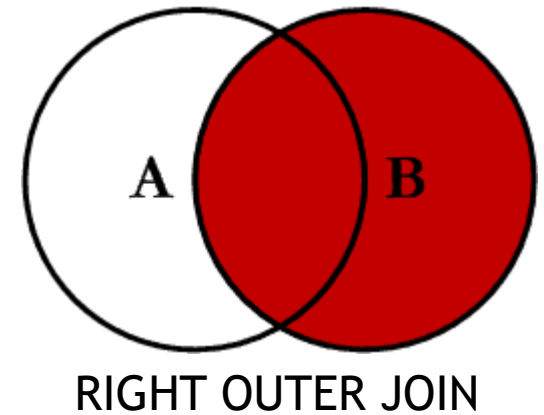
FIRST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME	DEPARTMENT_ID_1
Jose M...	100	Finance	100
Ismael	100	Finance	100
John	100	Finance	100
Daniel	100	Finance	100
Nancy	100	Finance	100
William	110	Accounting	110
Shelley	110	Accounting	110
Kimberely	(null)	(null)	(null)

# Outer Join

## : Right Outer Join

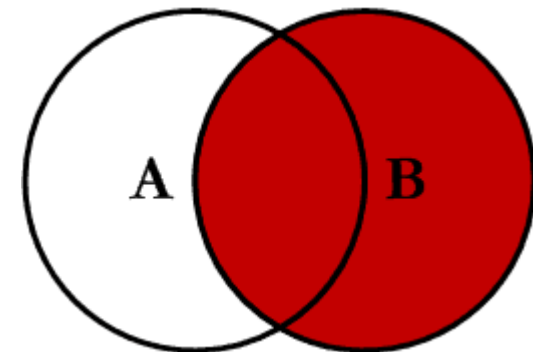
- ▶ 오른쪽 테이블의 모든 row를 결과 테이블에 나타냄

```
SELECT e.department_id, e.first_name, d.department_name  
FROM employees e RIGHT OUTER JOIN departments d  
ON e.department_id = d.department_id ;
```



# Outer Join

## : Right Outer Join



RIGHT OUTER JOIN

departments

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance
110	Accounting
120	Treasury

11개  
(사용o)

16개  
(사용x)

employees

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
100	Steven	90
101	Neena	90
102	Lex	90
108	Nancy	100
109	Daniel	100
110	John	100
111	Ismael	100
112	Jose M...	100
113	Luis	100
205	Shelley	110
206	William	110
178	Kimberely	(null)

106개

1개

103	100	Ismael	Finance
104	100	Nancy	Finance
105	110	William	Accounting
106	110	Shelley	Accounting
107	(null)	(null)	Treasury
108	(null)	(null)	Corporate Tax
109	(null)	(null)	Control And Cr...
110	(null)	(null)	Shareholder Se...
111	(null)	(null)	Benefits
112	(null)	(null)	Manufacturing
113	(null)	(null)	Construction
114	(null)	(null)	Contracting
115	(null)	(null)	Operations
116	(null)	(null)	IT Support
117	(null)	(null)	NOC
118	(null)	(null)	IT Helpdesk
119	(null)	(null)	Government Sales
120	(null)	(null)	Retail Sales
121	(null)	(null)	Recruiting
122	(null)	(null)	Payroll

106개

16개

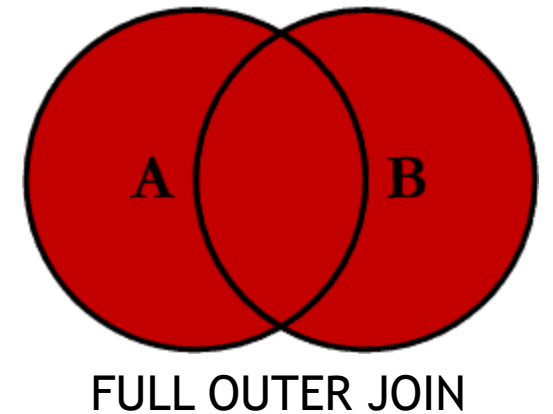
122개



# Outer Join

## : Full Outer Join

- ▶ MySQL은 Full Outer Join을 지원하지 않는다
  - ▶ 하지만 Left Join과 Right Join을 Union 하여 Full Outer Join을 구현할 수 있다.

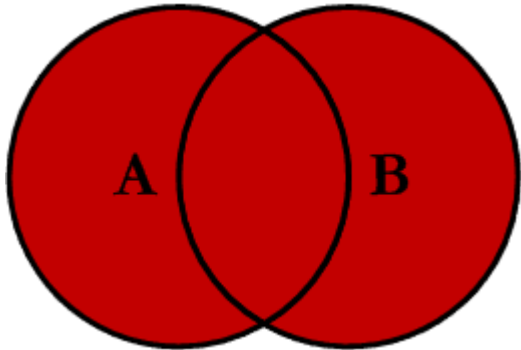


```
SELECT e.department_id, e.first_name, d.department_name
  FROM employees e LEFT OUTER JOIN departments d ON e.department_id = d.department_id
UNION
SELECT e.department_id, e.first_name, d.department_name
  FROM employees e RIGHT OUTER JOIN departments d ON e.department_id = d.department_id ;
```

76	80 Alyssa	Sales	105	70 Sherman	Public Relations
77	80 Jonathon	Sales	106	110 Shelley	Accounting
78	80 Jack	Sales	107	<del>110 William</del>	<del>Accounting</del>
79	(null) Kimberely	(null)	108	(null) (null)	NOC
80	80 Charles	Sales	109	(null) (null)	Manufacturing
81	50 Winston	Shipping	110	(null) (null)	Government Sales
82	50 Jean	Shipping	111	(null) (null)	IT Support
83	50 Martha	Shipping	112	(null) (null)	Benefits
84	50 Girard	Shipping	113	(null) (null)	Shareholder Se...
85	50 Nandita	Shipping	114	(null) (null)	Retail Sales
86	50 Alexis	Shipping	115	(null) (null)	Control And Cr...
87	50 Julia	Shipping	116	(null) (null)	Recruiting
			117	(null) (null)	Operations
			118	(null) (null)	Treasury
			119	(null) (null)	Payroll
			120	(null) (null)	Corporate Tax

# Outer Join

: Full Outer Join



FULL OUTER JOIN

76	80 Alyssa	Sales	105	70 Benjamin	Public Relations
77	80 Jonathon	Sales	106	110 Shelley	Accounting
78	80 Jack	Sales	107	<del>110 William</del>	<del>Accounting</del>
79	(null) Kimberly	(null)	108	(null) (null)	NOC
80	80 Charles	Sales	109	(null) (null)	Manufacturing
81	50 Winston	Shipping	110	(null) (null)	Government Sales
82	50 Jean	Shipping	111	(null) (null)	IT Support
83	50 Martha	Shipping	112	(null) (null)	Benefits
84	50 Girard	Shipping	113	(null) (null)	Shareholder Se...
85	50 Nandita	Shipping	114	(null) (null)	Retail Sales
86	50 Alexis	Shipping	115	(null) (null)	Control And Cr...
87	50 Julia	Shipping	116	(null) (null)	Recruiting
			117	(null) (null)	Operations
			118	(null) (null)	Treasury
			119	(null) (null)	Payroll
			120	(null) (null)	Corporate Tax
			121	(null) (null)	Construction
			122	(null) (null)	Contracting
			123	(null) (null)	IT Helpdesk

# Self Join

▶ 정의

- ▶ 자기 자신과 Join
- ▶ 동일한 테이블 명이 2번 이상 사용되므로 Alias를 사용할 수밖에 없음

PK		EMP		FK	
employee_id	first_name	...	manager_id	...	
100	Steven	...	NULL	...	
101	Neena	...	100	...	
102	Lex	...	100	...	
103	Alexander	...	102	...	
104	Bruce	...	103	...	
105	David	...	103	...	

```
SELECT * FROM employees e1, employees e2
WHERE e1.manager_id=e2.employee_id;
```

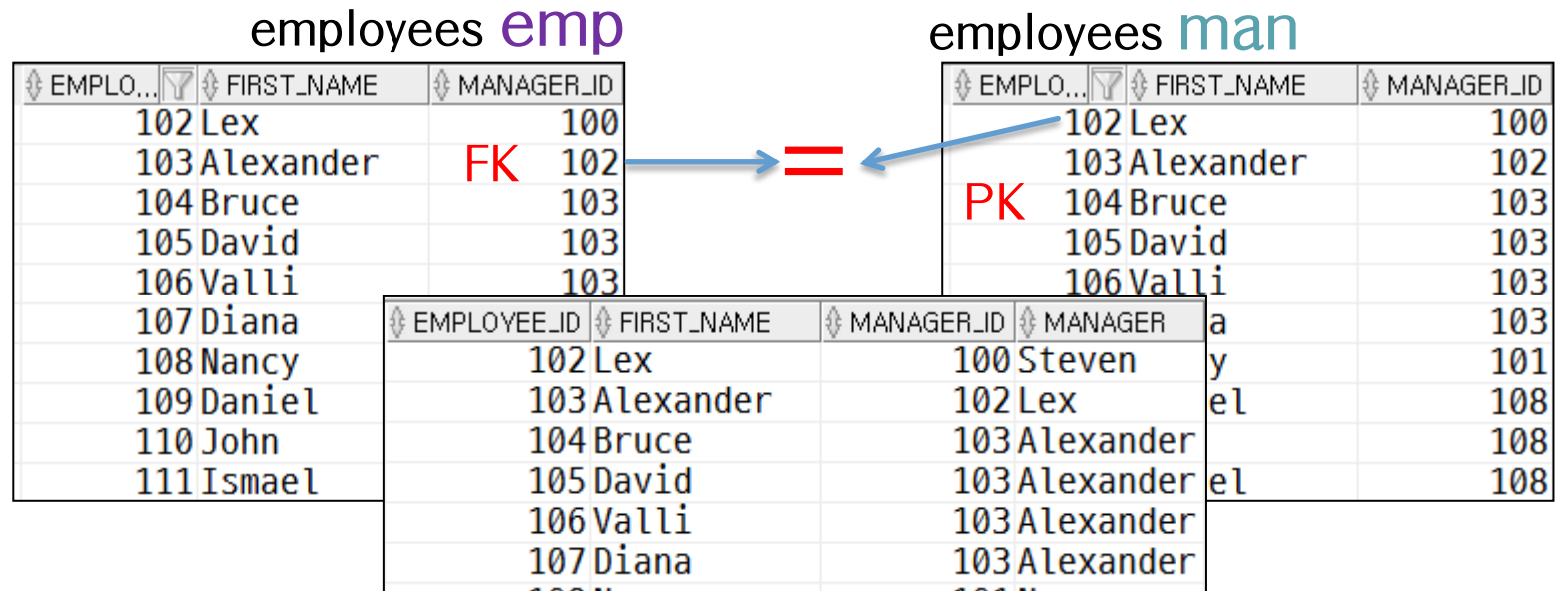
employee_id	first_name	...	manager_id	employee_id	first_name	...
101	Neena	...	100	100	Steven	...
102	Lex	...	100	100	Steven	...
103	Alexander	...	102	102	Lex	...
104	Bruce	...	103	103	Alexander	...
105	David	...	103	103	Alexander	...

# Self Join

## ▶ [연습] hr.employees

▶ SELF JOIN을 이용하여 다음의 값을 출력하시오

- ▶ EMPLOYEE\_ID
- ▶ FIRST\_NAME
- ▶ MANAGER의 EMPLOYEE ID
- ▶ MANAGER의 FIRST\_NAME



```
SELECT emp.employee_id, emp.first_name,  
       emp.manager_id, man.first_name manager  
FROM employees emp, employees man  
WHERE emp.manager_id = man.employee_id
```

# SQL

Group & Aggregation

# Aggregation Function (집계함수)

- ▶ 여러 행으로부터 하나의 결과값을 반환
- ▶ 종류
  - ▶ AVG
  - ▶ COUNT
    - ▶ COUNT(\*) : 테이블 내의 행 수 (NULL도 카운트됨)
    - ▶ COUNT(expr) : 테이블 내의 행 수 (NULL 제외)
  - ▶ MAX
  - ▶ MIN
  - ▶ SUM
  - ▶ STDDEV
  - ▶ VARIANCE

# Aggregation Function

```
SELECT salary FROM employees;
```

salary
24000.00
17000.00
17000.00
9000.00
6000.00
4800.00
4800.00
4200.00
12000.00
9000.00
8200.00
7700.00

```
SELECT AVG(salary) FROM employees;
```

AVG(salary)
6461.682243

# Aggregation Function

## ▶ count()

- ▶ 함수에 입력되는 데이터의 총 건수를 구하는 함수
- ▶ \* 를 사용하면 null을 포함한 총 Row의 개수를 구하며, 필드를 명시할 경우 null 값을 제외한다

 null 포함

```
SELECT COUNT(*), COUNT(commission_pct)
FROM employees;
```

 null 제외

```
SELECT COUNT(*)
FROM employees
WHERE salary > 16000;
```



# Aggregation Function

- ▶ `sum()`

- ▶ 입력된 데이터들의 합계 값을 구하는 함수

```
SELECT COUNT(*), SUM(salary)  
FROM employees;
```

# Aggregation Function

## ▶ avg()

- ▶ 입력된 데이터들의 평균 값을 구하는 함수
- ▶ 주의: null 값이 있는 경우 빼고 계산해야 함 = IFNULL, COALESCE 함수와의 조합

```
SELECT COUNT(*), SUM(salary), AVG(salary)
FROM employees;
```

```
SELECT COUNT(*), SUM(salary), AVG(IFNULL(salary,0))
FROM employees;
```

- ▶ NULL 값을 포함시킬 것인지, 뺄 것인지에 따라 통계 결과가 달라진다  
어떤 값을 대상으로 통계 값을 잡을 것인지는 정책으로 결정

name	point
홍길동	70
일지매	null → 0
유관순	50

•  $120 / 3 = 40$

•  $120 / 2 = 60$  ✓

# Aggregation Function

- ▶ min() / max()

- ▶ 입력된 값 중 가장 작은 값/큰 값을 구하는 함수
- ▶ 여러 건의 데이터를 순서대로 정렬 후 값을 구하기 때문에 데이터가 많을 때는 느리다 (사용에 유의)

```
SELECT COUNT(*), MAX(salary), MIN(salary)  
FROM employees;
```

# 일반적인 오류

- ▶ 부서의 평균 연봉을 구하고자 다음과 같은 Query를 실행

```
SELECT department_id, AVG(salary)
FROM employees;
```



- ▶ 주의

- ▶ 집계함수의 결과는 하나의 ROW
- ▶ department\_id는 하나의 ROW에 표현할 수 없음
- ▶ 부서별과 같은 내용이 필요할 때는 GROUP BY 절 사용

- ▶ 주의 2

- ▶ SELECT의 Col 목록에는 Group by에 참여한 필드나 aggregate 함수만 올 수 있다
- ▶ Group by 이후에는 Group by에 참여한 필드나 aggregate 함수만 남아있는 셈
  - ▶ HAVING, ORDER BY도 마찬가지

# GROUP BY

```
SELECT department_id, salary
FROM employees
ORDER BY department_id;
```

department_id	salary
NULL	7000.00
10	4400.00
20	13000.00
20	6000.00
30	11000.00
30	3100.00
30	2900.00
30	2800.00
30	2600.00
30	2500.00

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
ORDER BY department_id;
```

department_id	AVG(salary)
NULL	7000.000000
10	4400.000000
20	9500.000000
30	4150.000000

# HAVING 절

- ▶ Aggregation 결과에 대해 다시 condition을 검사할 때
- ▶ 일반적 오류
  - ▶ 평균 월급이 2000 이상인 부서는?

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 2000
GROUP BY department_id;
```



- ▶ 주의
  - ▶ WHERE 절은 Aggregation 이전, HAVING 절은 Aggregation 이후의 필터링
  - ▶ HAVING 절에는 GROUP BY에 참여한 컬럼이나 Aggregation 함수만 사용 가능

# 단일 SQL문의 실행

: Flow



# GROUP BY 절

## ▶ [예제] hr.employees

- ▶ 급여(salary) 합계가 20000 이상인 부서의 부서 번호와 인원 수, 급여 합계를 출력하기 위해 다음과 같은 쿼리를 작성했다.

```
SELECT department_id, COUNT(*), SUM(salary)
  FROM employees
 WHERE SUM(salary) > 20000
 GROUP BY department_id;
```

- ▶ 위 쿼리를 살펴보고 문제점이 무엇인지 생각해 봅시다



# GROUP BY 절

## ▶ [SOLUTION] hr.employees

- ▶ GROUP 함수는 WHERE 절 이후에 처리되므로 SUM(salary)는 WHERE 절에 사용할 수 없음

```
SELECT department_id, COUNT(*), SUM(salary)
FROM employees
WHERE SUM(salary) > 20000
GROUP BY department_id
HAVING SUM(salary) > 20000
```

- ▶ Having 절에는 그룹함수와 GROUP BY에 참여한 컬럼만 사용할 수 있음

# 단일 SQL 작성법

1. 최종 출력될 정보에 따라 원하는 컬럼을 **SELECT** 절에 추가
2. 원하는 정보를 가진 테이블들을 **FROM** 절에 추가
3. **WHERE** 절에 알맞은 **JOIN** 조건 추가
4. **WHERE** 절에 알맞은 검색 조건 추가
5. 필요에 따라 **GROUP BY, HAVING** 등을 통해 Grouping하고 Aggregate
6. 정렬 조건 **ORDER BY**에 추가

# SQL

SUBQUERY, SET Operation

# Subquery

- ▶ 하나의 SQL 질의문 속에 다른 SQL 질의문이 포함되어 있는 형태
- ▶ 예) 'Susan' 보다 급여가 많은 사람은?
  - ▶ 급여가 많은 사람의 이름?
    - ▶ SELECT first\_name FROM employees WHERE salary > ???
  - ▶ 'Susan'의 급여는?
    - ▶ SELECT salary FROM employees WHERE first\_name='Susan'

```
SELECT first_name
FROM employees
WHERE salary > ( SELECT salary
                  FROM employees
                  WHERE first_name = 'Susan');
```

# Subquery

- ▶ [연습] hr.employees
  - ▶ 'Den' 보다 급여를 많이 받는 사원의 이름과 급여는?

```
select salary
from employees
where first_name='Den'
```

11000

```
select first_name, salary
from employees
where salary > ??
```

11000



```
select employee_id, first_name, salary
from employees
where salary > (select salary
                 from employees
                 where first_name='Den');
```



# Single-Row Subquery

## ▶ [연습] hr.employees

- ▶ 급여를 가장 적게 받는 사람의 이름, 급여, 사원 번호를 출력하십시오

```
SELECT first_name, salary, employee_id  
FROM employees  
WHERE salary = (SELECT MIN(salary)  
                FROM employees);
```

- ▶ 평균 급여보다 적게 받는 사원의 이름, 급여를 출력해 보세요.

# Multi-Row Subquery

- ▶ Subquery의 결과가 둘 이상의 Row
- ▶ Multi-Row에 대한 연산을 사용해야 함: ANY, ALL, IN, EXIST ...

```
SELECT first_name, salary, department_id
FROM employees
WHERE first_name = (SELECT MIN(first_name)
                    FROM employees GROUP BY department_id);
```



```
SELECT first_name, salary, department_id
FROM employees
WHERE first_name IN (SELECT MIN(first_name)
                    FROM employees GROUP BY department_id);
```



```
SELECT first_name, salary, department_id
FROM employees
WHERE first_name =ANY (SELECT MIN(first_name)
                      FROM employees GROUP BY department_id);
```





# Multi-Row Subquery

연산자	설명
IN	리턴되는 값 중에서 조건에 해당하는 값이 있으면 참
ANY, SOME	서브쿼리에 의해 리턴되는 각각의 값과 조건을 비교하여 하나 이상을 만족하면 참
ALL	값을 서브쿼리에 의해 리턴되는 모든 값을 비교하여 모두 만족해야 참
EXISTS	메인 쿼리의 비교 조건이 서브쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 참

- ANY는 OR과 비슷
- ALL은 AND와 비슷

# Multi-Row Subquery

## ▶ [연습] hr.employees

### ▶ IN

```
select first_name, salary  
from employees
```

```
where salary IN (select salary  
                  from employees  
                  where department_id = 110);
```

	FIRST_NAME	SALARY
1	Shelley	12008
2	Nancy	12008
3	William	8300

```
where salary = 12008  
or salary = 8300
```

SALARY
12008
8300

# Multi-Row Subquery

## ▶ [연습] hr.employees

### ▶ ALL (AND)

```
select first_name, salary
from employees
where salary > ALL (select salary
                      from employees
                      where department_id = 110);
```

```
where salary > 12008
and salary > 8300
```

SALARY
12008
8300

	FIRST_NAME	SALARY
1	Michael	13000
2	Karen	13500
3	John	14000
4	Lex	17000
5	Neena	17000
6	Steven	24000

# Multi-Row Subquery

## ▶ [연습] hr.employees

### ▶ ANY (OR)

```
select first_name, salary  
from employees
```

```
where salary > ANY (select salary  
                      from employees  
                      where department_id = 110);
```

```
where salary > 12008  
or salary > 8300
```

SALARY
12008
8300

	FIRST_NAME	SALARY
1	Steven	24000
2	Neena	17000
3	Lex	17000
4	John	14000
5	Karen	13500
6	Michael	13000
7	Nancy	12008
8	Shelley	12008
9	Alberto	12000
10	Lisa	11500
11	Den	11000
12	Gerald	11000
21	Danielle	9500
22	David	9500
23	Patrick	9500
24	Peter	9000
25	Alexander	9000
26	Allan	9000
27	Daniel	9000
28	Alyssa	8800
29	Jonathon	8600
30	Jack	8400

# Subquery

: 예제

## ▶ [예제] hr.employees

- ▶ 각 부서별로 최고급여를 받는 사원을 출력하세요 - (조건절에서 비교)

```
SELECT department_id, employee_id, first_name, salary
FROM employees
WHERE (department_id, salary) in (SELECT department_id, MAX(salary)
                                   FROM employees
                                   GROUP BY department_id)
```

employees 테이블

	DEPARTME...	EMPLOYEE_ID	FIRST_NAME	SALARY
1	10	200	Jennifer	4400
2	20	201	Michael	13000
3	20	202	Pat	6000
4	30	114	Den	11000
5	30	115	Alexander	3100
9	30	119	Karen	2500
10	40	203	Susan	6500
11	50	120	Matthew	8000
12	50	121	Adam	8200
13	50	122	Deva	7000

DEPARTME...	MAX(SALARY)
10	4400
20	13000
30	11000
40	6500
50	8200
60	9000
70	10000
80	14000
90	24000

# Subquery

: 예제

## ▶ [예제] hr.employees

- ▶ 각 부서별로 최고급여를 받는 사원을 출력하세요 - (테이블에서 조인)

```
SELECT e.department_id, e.employee_id, e.first_name, e.salary
FROM employees e, (SELECT department_id, max(salary) salary
FROM employees
GROUP BY department_id) s
WHERE e.department_id = s.department_id
AND e.salary = s.salary;
```

employees e

	DEPARTME...	EMPLOYEE_ID	FIRST_NAME	SALARY
1	10	200	Jennifer	4400
2	20	201	Michael	13000
3	20	202	Pat	6000
4	30	114	Den	11000
5	30	115	Alexander	3100
6	30	116	Shelli	2900
7	30	117	Sigal	2800
8	30	118	Guy	2600
9	30	119	Karen	2500
10	40	203	Susan	6500
11	50	120	Matthew	8000
12	50	121	Adam	8200

salary s

DEPARTMENT_ID	SALARY
10	4400
20	13000
30	11000
40	6500
50	8200
60	9000
70	10000
80	14000
90	24000
100	12008
110	12008
(null)	7000

# LIMIT (MySQL)

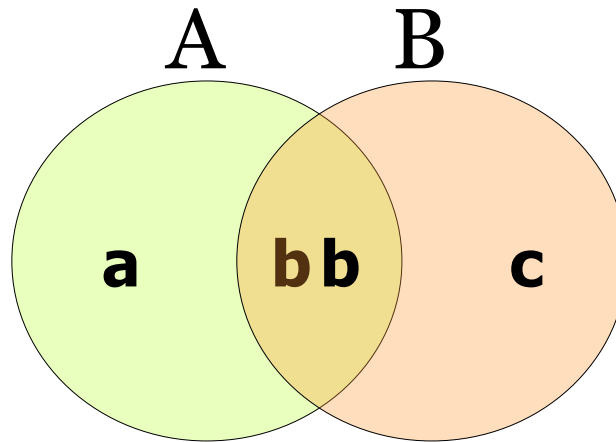
- ▶ **SELECT**로 받아온 결과(레코드)의 출력 개수를 제한하고자 할 때 **SQL** 문장의 제일 마지막에 사용
  - ▶ **LIMIT n** : 첫 번째 행부터 n개를 출력
  - ▶ **LIMIT s, n** : s 번째 행부터 n개를 출력 (s 개의 행을 **SKIP**)

```
SELECT first_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 3;
```

```
SELECT first_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 10, 3;
```

# 집합(SET) Operator

- ▶ 두 집합의 결과를 가지고 집합 연산을 수행
- ▶ UNION, UNION ALL, ~~INTERSECT, EXCEPT~~ (MySQL에서는 INTERSECT, EXCEPT 연산은 지원하지 않음)
  - ▶ UNION은 중복 레코드 제거 작업을 진행해야 하기에 UNION보다 속도면에서 느리다
  - ▶ MySQL에서 INTERSECT, EXCEPT 개념을 활용하기 위해 JOIN연산을 활용한다



- $A \cup B = \{a, b, c\}$
- $A \cup ALL B = \{a, b, b, c\}$
- ~~$A \cap B = \{b\}$~~
- ~~$A \setminus B = \{a\}$~~

```
SELECT first_name FROM employees
UNION
SELECT department_name FROM departments;
```