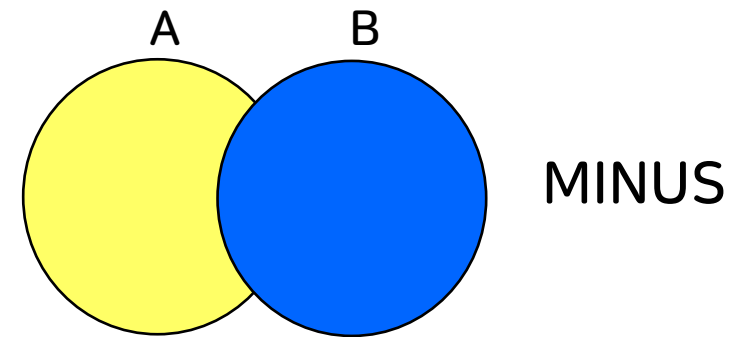
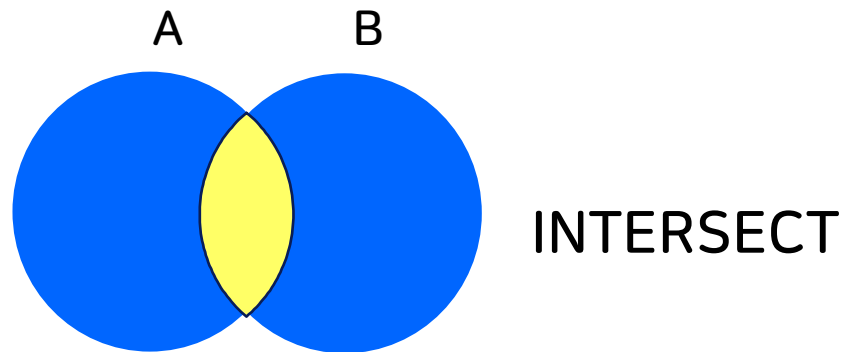
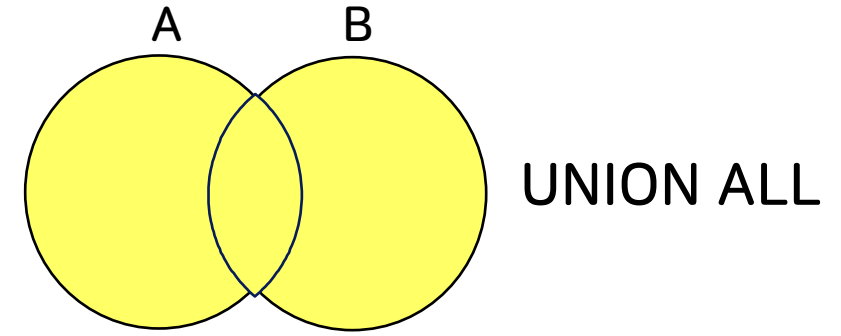
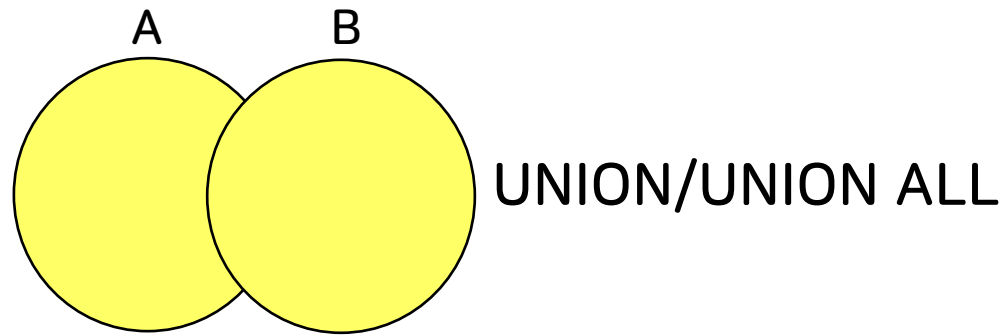


5. 기타쿼리



(1) 집합 연산자의 사용

SET 연산자



■ 구문

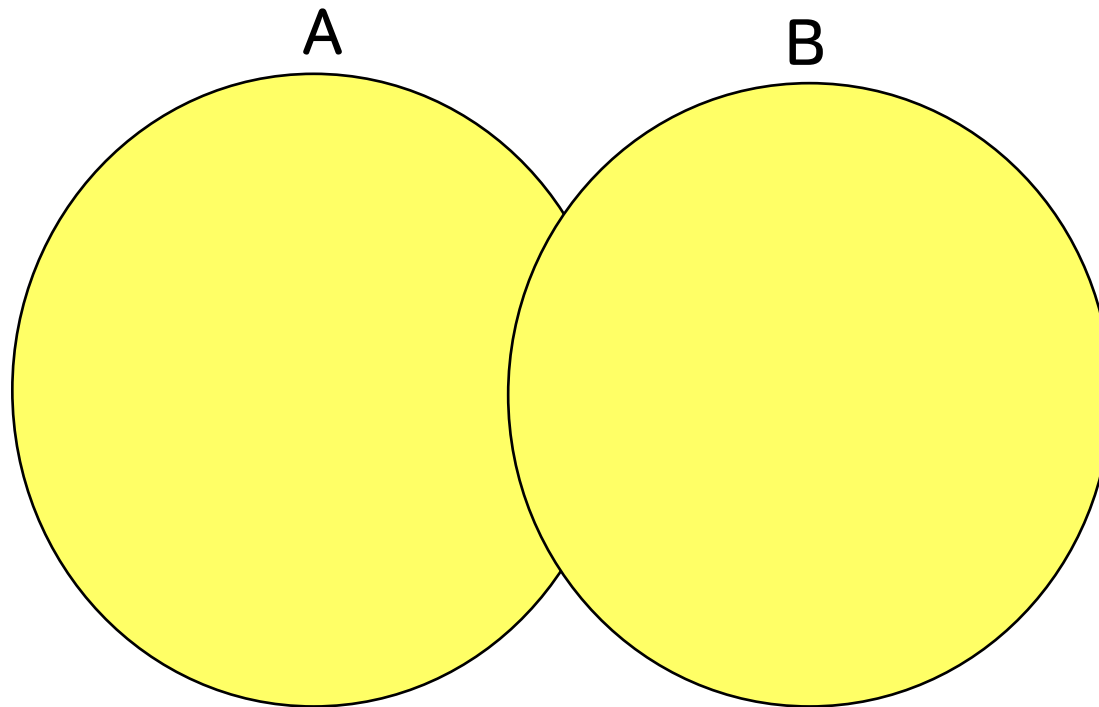
```
SELECT 문  
[UNION | UNION ALL | INTERSECT | MINUS]  
SELECT 문;
```

실습에 사용되는 테이블

- 실습에 사용되는 테이블은 다음과 같습니다.
 - EMPLOYEES: 현재의 모든 사원에 대한 세부 사항을 제공합니다.
 - JOB_HISTORY: 사원의 업무가 변경된 경우 이전 업무의 시작 및 종료 일자, 업무 식별 번호 및 부서에 관한 세부 사항을 기록합니다.

UNION 연산자

- UNION 연산자는 두 질의의 결과를 중복을 제거한 후 반환합니다.



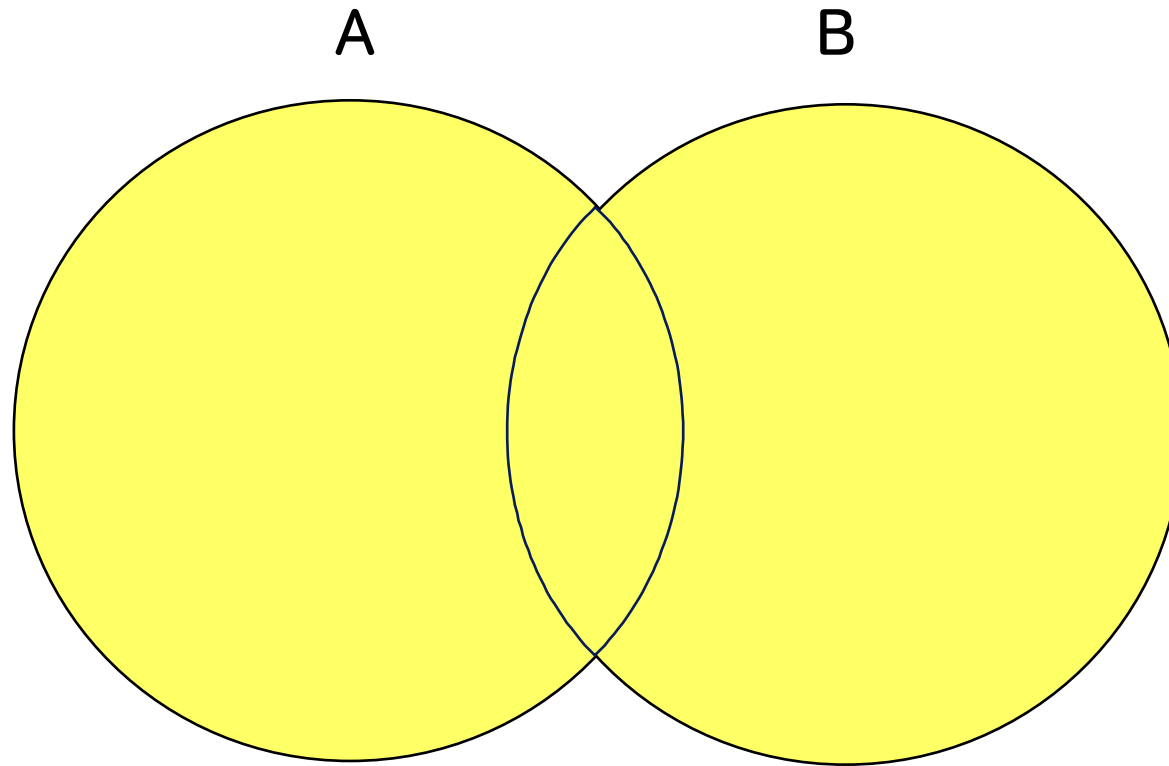
UNION 연산자의 사용

- 예제는 모든 사원의 현재 및 이전 업무에 대한 세부 사항을 표시합니다.
- 중복행을 제거한 후 각 사원에 대해 한 번씩만 표시합니다.
- 결과가 사원번호에 대해 오름차순 정렬됩니다.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

UNION ALL 연산자

- UNION ALL 연산자는 두 질의의 결과를 중복을 포함하여 반환합니다.



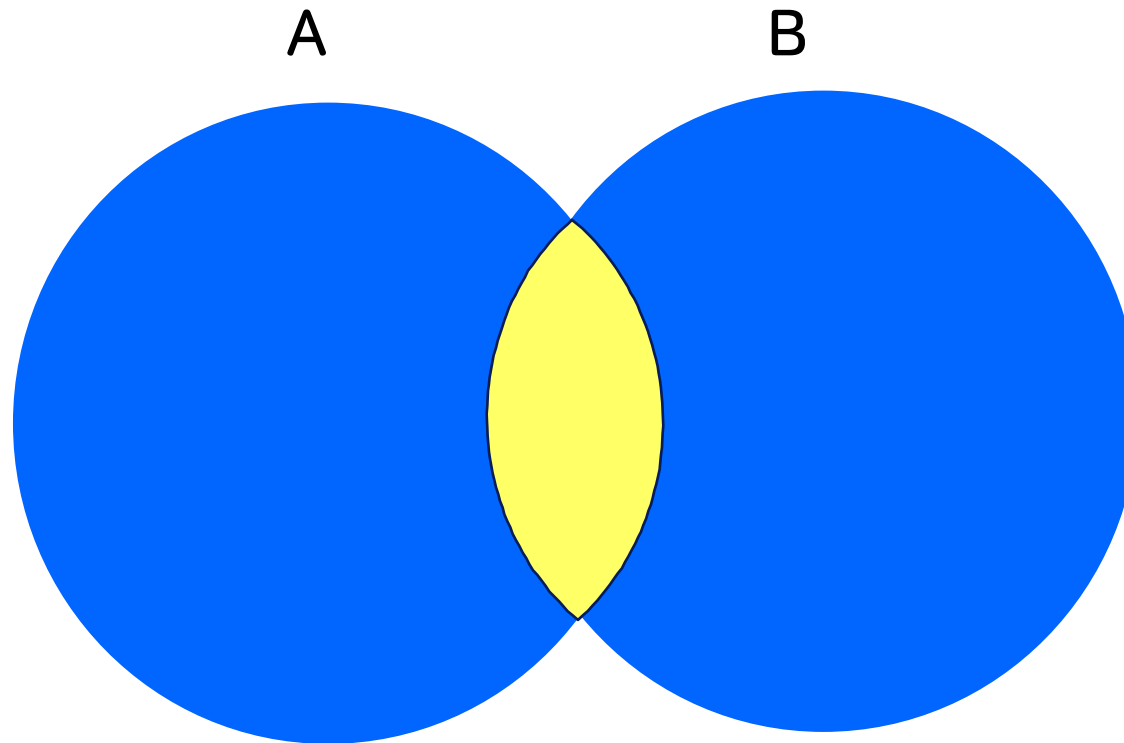
UNION ALL 연산자의 사용

- 예제는 모든 사원의 현재 및 이전 업무를 표시합니다.
- 중복행이 제거되지 않습니다.
- 결과가 사원번호에 대해 정렬되지 않습니다.

```
SELECT employee_id, job_id
FROM   employees
UNION ALL
SELECT employee_id, job_id
FROM   job_history
ORDER BY employee_id;
```


INTERSECT 연산자

- INTERSECT 연산자는 여러 질의에 공통적인 행을 모두 반환합니다.



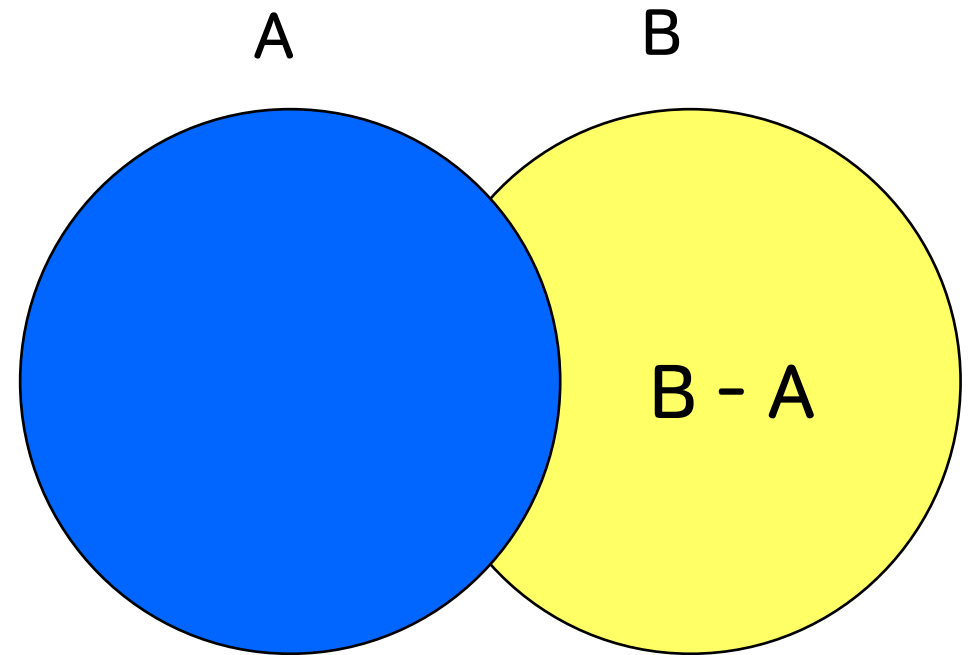
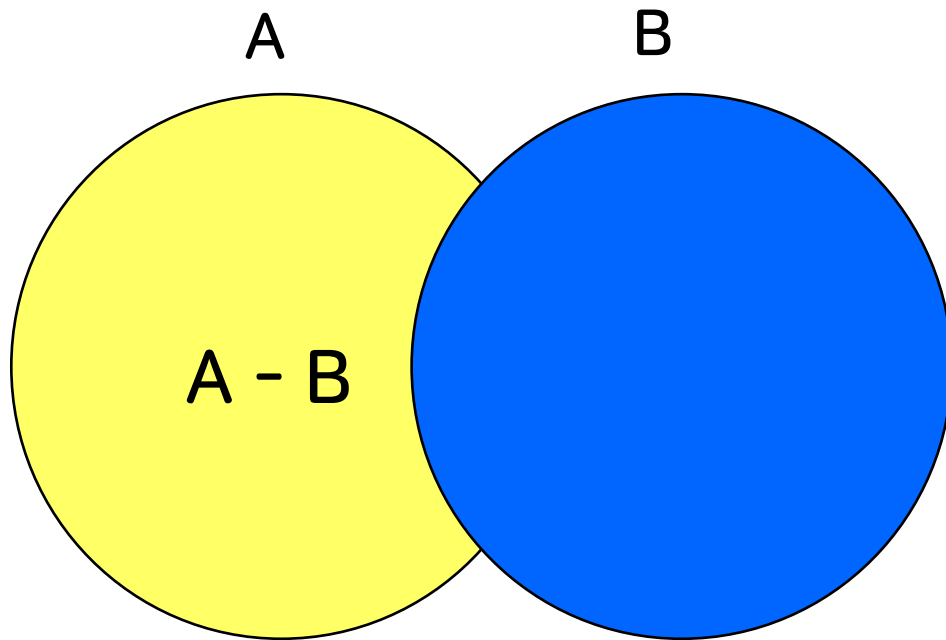
INTERSECT 연산자의 사용

- 다음 예제는 입사 이후 현재 업무와 같은 업무를 담당한 적이 있는 사원의 사원 ID와 업무 ID를 표시합니다.

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

MINUS 연산자

- MINUS 연산자를 사용하면 첫째 SELECT 문 MINUS 둘째 SELECT 문의 경우 첫째 질의가 반환한 행으로 부터 둘째 질의가 반환한 행을 제외한 행이 반환됩니다.



MINUS 연산자의 사용

- 다음 예제는 업무가 변경된 적이 없는 사원의 사원 ID를 표시합니다.

```
SELECT employee_id,job_id FROM employees  
MINUS  
SELECT employee_id,job_idFROM job_history;
```

- 위 예제를 바꾸어 다음과 같이 실행하면 JOB_HISTORY 테이블에서 퇴사한 상태의 사원정보를 찾을 수 있습니다.

```
SELECT employee_id,job_id FROM job_history  
MINUS  
SELECT employee_id,job_id FROM employees;
```

SET 연산자 사용 지침

- SELECT 목록에 있는 표현식의 개수와 데이터 유형이 서로 일치해야 합니다.
- UNION ALL을 제외한 집합연산자는 중복 행이 자동으로 제거됩니다.
- UNION ALL을 제외한 다른 연산자의 출력은 기본적으로 오름차순으로 정렬됩니다.
- 첫째 질의의 열 이름이 결과에 표시됩니다.

SELECT 문의 일치


- 두 질의의 SELECT 목록에 있는 표현식의 개수를 일치시키기 위해 숫자열의 경우에 상수를 지정할 수 있습니다.
- 다음 예제는 UNION 연산자를 사용하여 모든 사원의 사원 ID, 업무 ID 및 급여를 표시합니다.

```
SELECT employee_id, job_id, salary
FROM   employees
UNION
SELECT employee_id, job_id, -1
FROM   job_history;
```

변환함수의 응용

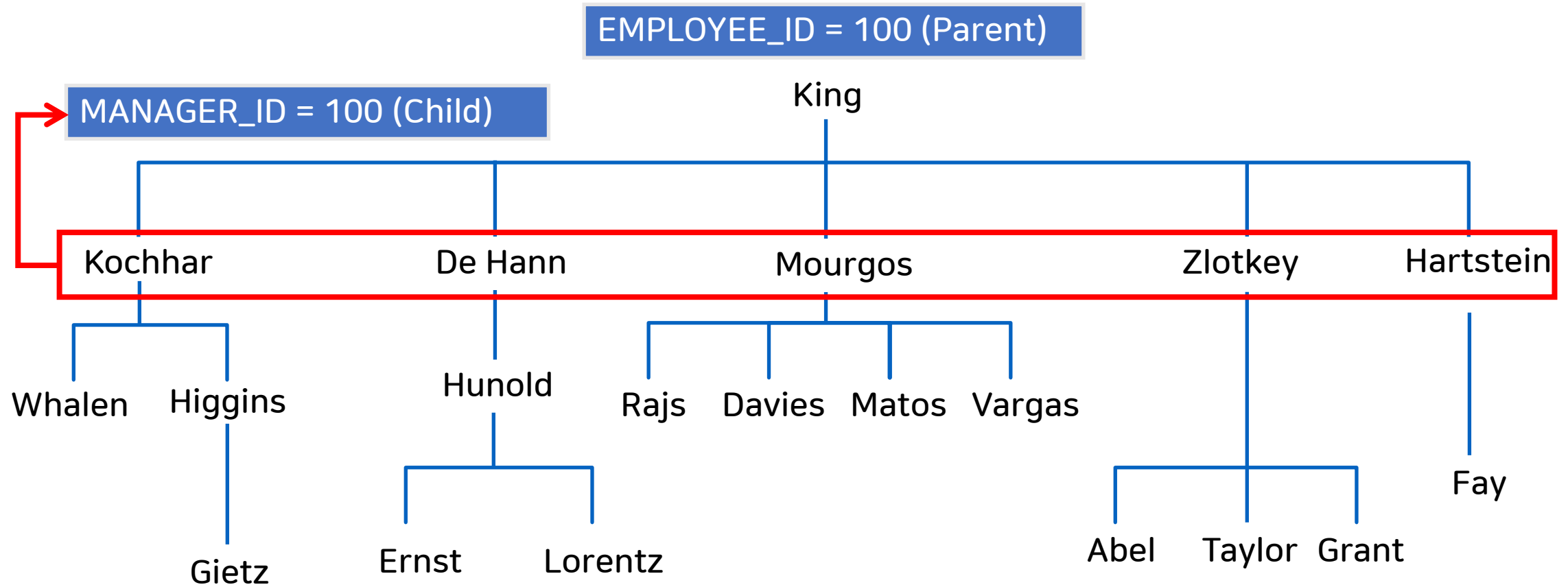
- 집합연산 시, SELECT 목록에 있는 표현식의 개수를 일치시키기 위해 데이터 유형 변환 함수를 사용할 수 있습니다.
- 다음은 UNION 연산자를 사용하여 모든 사원의 부서 ID, 위치 및 입사 일자를 표시합니다.

```
SELECT department_id, TO_NUMBER(null) AS location, hire_date
FROM   employees
UNION
SELECT department_id, location_id, TO_DATE(null)
FROM   departments;
```



〔2〕 계층퀴리

EMPLOYEES의 자연 트리 구조



계층질의

- 질의가 계층 질의인지 여부는 CONNECT BY 및 START WITH 절로 식별할 수 있습니다.
- 구문

```
SELECT [LEVEL], column, expr...  
FROM   table  
[WHERE condition(s)]  
[START WITH condition(s)]  
[CONNECT BY PRIOR condition(s)] ;
```

- START WITH 계층의 루트 행(시작점)을 지정합니다.
- CONNECT BY 부모 행과 자식 행 사이에 관계가 존재하는 열을 지정합니다.
- PRIOR 계층검색의 방향을 결정하는 필수적인 절입니다.

트리 검색

- 검색시작점
- START WITH 절에서 트리의 루트로 사용될 행이 결정됩니다.
- START WITH 절에서는 유효한 조건을 결합하여 사용하는 것이 가능합니다.

```
START WITH column1 = value
```

- START WITH 절 예시
 - 다음은 EMPLOYEES 테이블을 사용하여 이름이 Kochhar인 직원부터 시작합니다.

```
...START WITH last_name = 'Kochhar'
```

트리의 검색 방향

- 쿼리의 방향은 CONNECT BY 절의 PRIOR 뒤에 어떤 열이 오는가에 따라 결정됩니다.

- Top-Down

- 쿼리가 부모부터 시작하여 자식 방향으로 수행

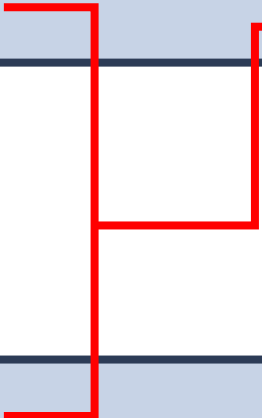
CONNECT BY PRIOR column1 = column2

- Bottom-Up

- 쿼리가 자식부터 시작하여 부모 방향으로 수행

CONNECT BY PRIOR column2 = column1

Column 1 = 부모키
Column 2 = 자식키



트리 검색 예제

- Top-Down

```
CONNECT BY PRIOR employee_id = manager_id
```

- Bottom-Up

```
CONNECT BY PRIOR manager_id = employee_id
```

트리 검색: Top-Down

- 다음 예제는 성이 king인 사원을 시작으로 위에서 아래로 검색하여 사원 전체의 트리구조를 볼 수 있습니다.

```
SELECT  employee_id, last_name, manager_id
FROM    employees
START   WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id ;
```

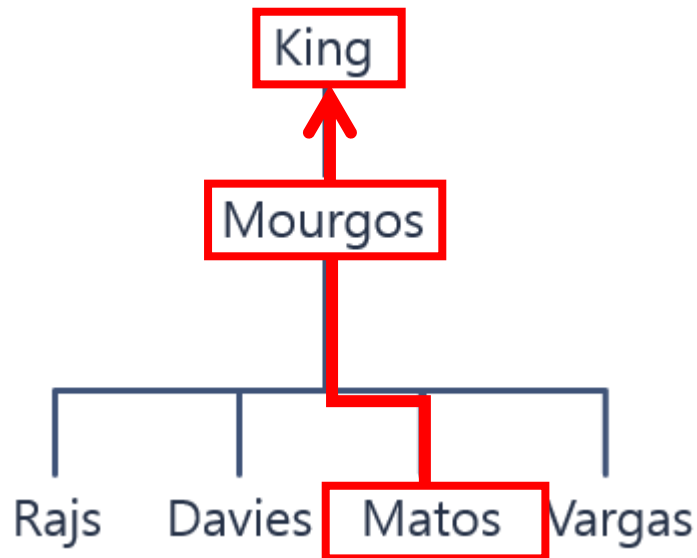
- 동일한 결과를 보기 위해 START WITH 절에 다양한 조건이 사용될 수 있습니다.

```
SELECT  employee_id, last_name, manager_id
FROM    employees
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id ;
```

트리 검색: Bottom-Up

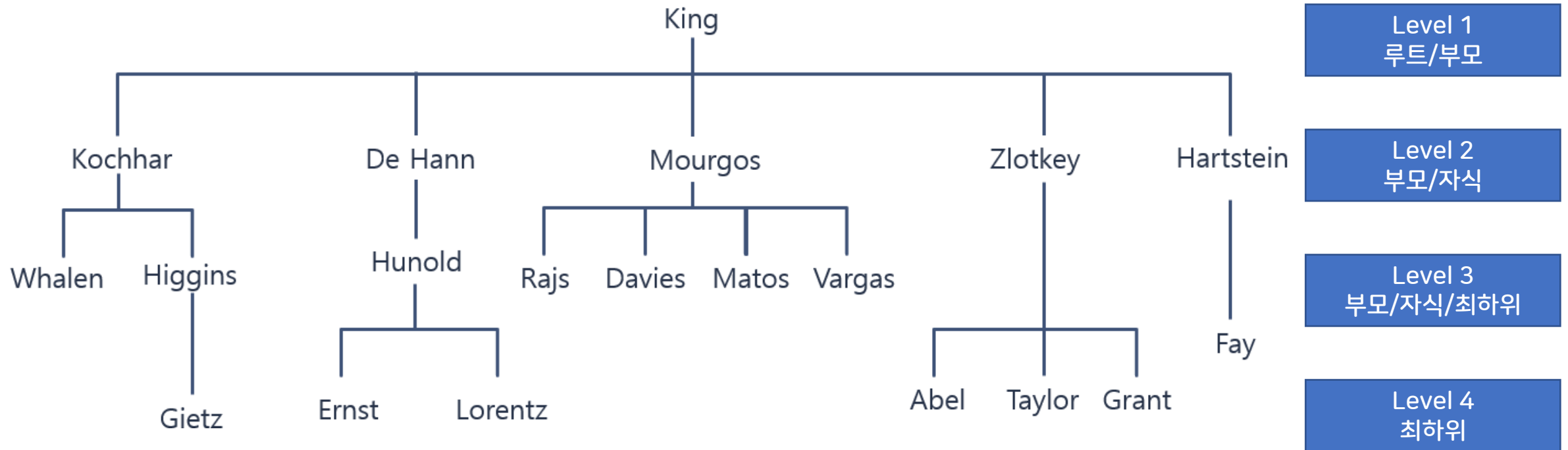
- 다음 예제는 143번 사원을 시작으로 아래에서 위로 검색하여 관리자를 검색합니다.

```
SELECT  employee_id, last_name, manager_id
FROM    employees
START   WITH employee_id = 143
CONNECT BY PRIOR manager_id = employee_id;
```



EMPLOYEE_ID	LAST_NAME	MANAGER_ID
143	Matos	124
124	Mourgos	100
100	King	(null)

Level 의사 열의 사용



Level 의사열의 사용

- 이전의 Top-Down 검색에 Level 의사열을 함께 표시합니다.

```
SELECT  LEVEL, employee_id, last_name, manager_id
FROM    employees
START   WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id ;
```

- START WITH의 시작점에 따라 LEVEL값은 달라질 수 있습니다.

```
SELECT  LEVEL, employee_id, last_name, manager_id
FROM    employees
START   WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id ;
```

LEVEL 및 LPAD를 사용하여 계층 보고서 작성

- 다음과 같이 회사 관리 레벨을 가장 높은 레벨부터 시작하여 다음 레벨을 각각 들여쓰기하여 표시하는 보고서를 작성합니다.
- LPAD 함수를 의사 열 LEVEL과 함께 사용하면 계층 보고서를 들여쓰기된 트리 형태로 표시할 수 있습니다.

```
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_') AS org_chart  
FROM   employees  
START WITH last_name='King'  
CONNECT BY PRIOR employee_id=manager_id;
```

계층 쿼리에 사용되는 함수

■ 계층 쿼리 관련 함수

함수	설명
SYS_CONNECT_BY_PATH	루트 데이터로부터 현재 전개할 데이터까지 경로를 표시
CONNECT_BY_ROOT	현재 전개할 데이터의 루트 데이터를 표시

■ 계층 쿼리 함수 사용 예제

```
SELECT  CONNECT_BY_ROOT(employee_id) AS root,  
        employee_id, last_name,  
        SYS_CONNECT_BY_PATH(employee_id, '/') AS 경로  
FROM    employees  
START   WITH employee_id = 100  
CONNECT BY PRIOR employee_id = manager_id;
```