

제 7 장

SQL-99: 스키마 정의, 기본 제약조건, 질의어

Fundamentals of Database Systems

R. A. Elmasri and S. B. Navathe

SQL 개요

□ SQL 개요

- ✓ **SQL**은 현재 **DBMS** 시장에서 관계 **DBMS**가 압도적인 우위를 차지하는데 중요한 요인의 하나
- ✓ **SQL**은 **IBM** 연구소에서 1974년에 **System R**이라는 관계 **DBMS** 시제품을 연구할 때 관계 대수와 관계 해석을 기반으로, 집단 함수, 그룹화, 갱신 연산 등을 추가하여 개발된 언어
- ✓ 1986년에 **ANSI**(미국 표준 기구)에서 **SQL** 표준을 채택함으로써 **SQL**이 널리 사용되는데 기여
- ✓ 다양한 상용 관계 **DBMS**마다 지원하는 **SQL** 기능에 다소 차이가 있음

SQL 개요(계속)

〈표 4.2〉 SQL의 발전 역사

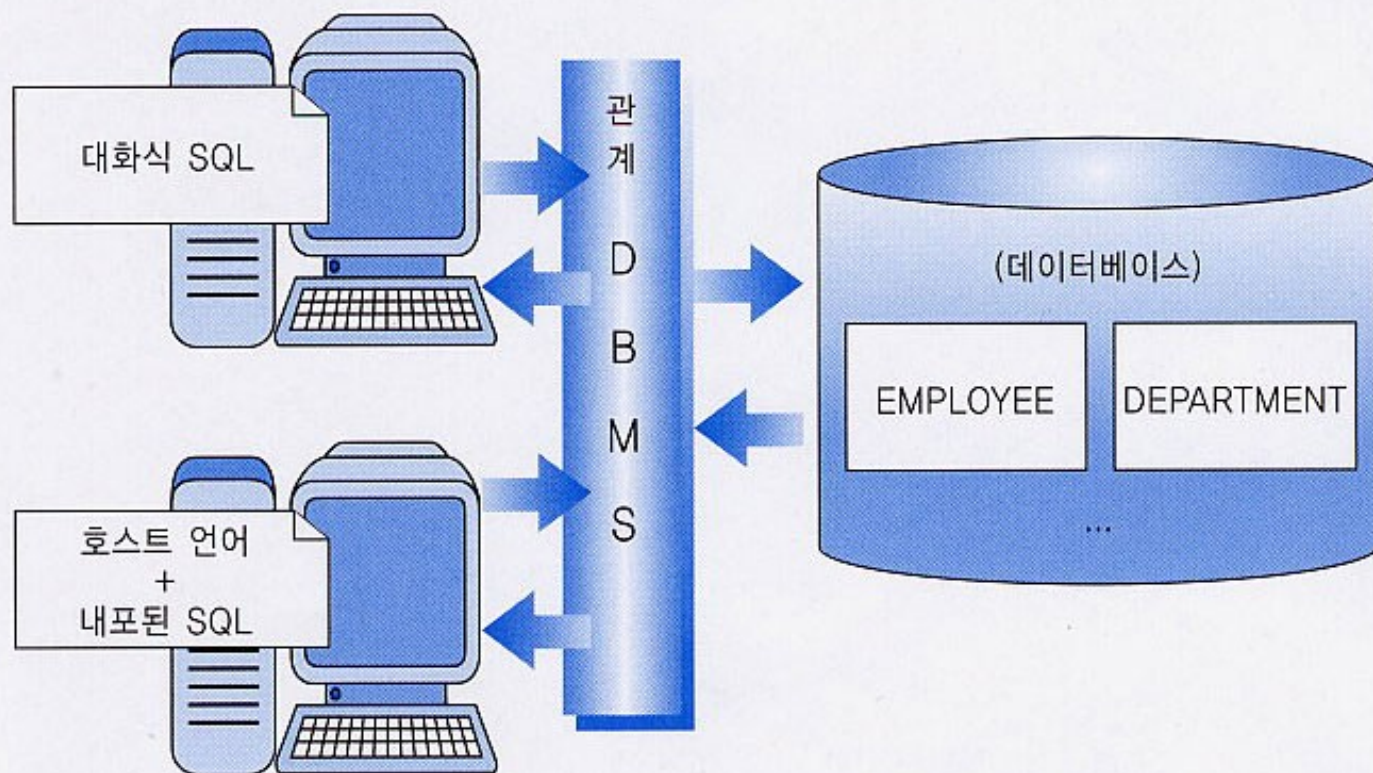
버전	특징
SEQUEL	Structured English Query Language의 약어. Sysetm R 프로젝트에서 처음으로 제안됨
SQL	Structured Query Language의 약어. 1983년에 IBM의 DB2, 1991년에 IBM SQL/DS에 사용됨
SQL-86	1986년에 미국 ANSI에서 표준으로 채택됨. 1987년에 ISO에서 표준으로 채택됨
SQL-89	무결성 제약조건 기능이 강화됨
SQL2(SQL-92)	새로운 데이터 정의어와 데이터 조작어 기능이 추가됨. 약 500페이지 분량
SQL3(SQL-99)	객체 지향과 순환 기능 등이 추가됨. 약 2000페이지 분량

SQL 개요(계속)

□ SQL 개요(계속)

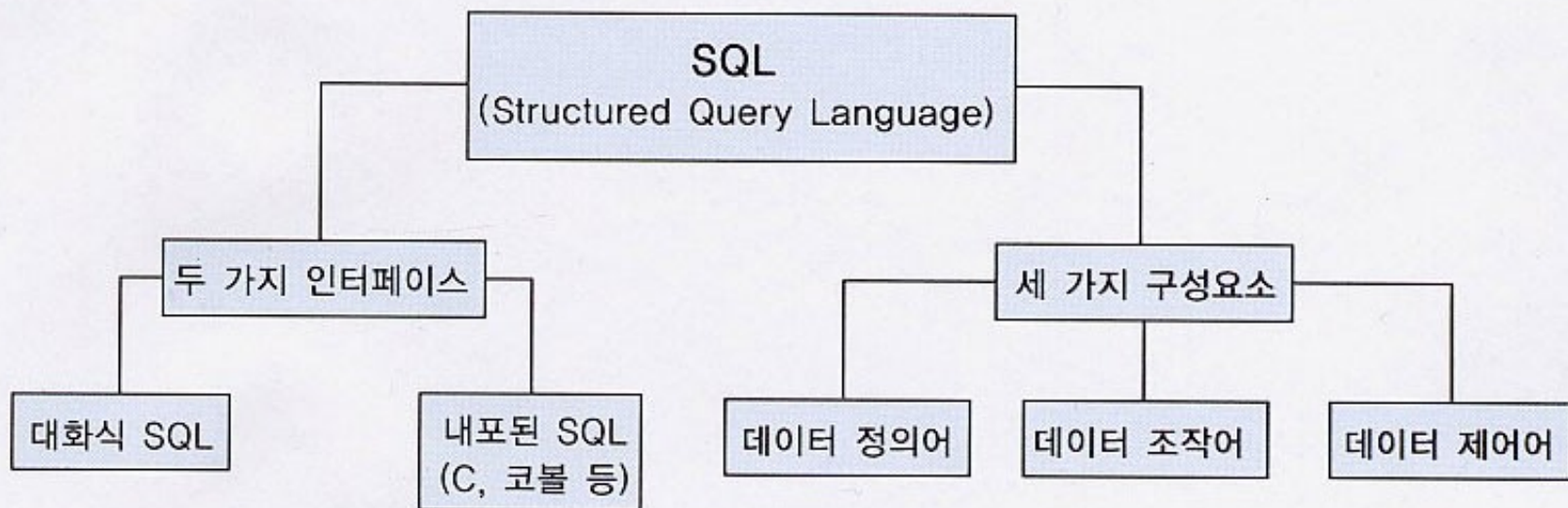
- ✓ **SQL**은 비절차적 언어(선언적 언어)이므로 사용자는 자신이 원하는 바 (what)만 명시하며, 원하는 것을 처리하는 방법(how)은 명시할 수 없음
- ✓ 관계 **DBMS**는 사용자가 입력한 **SQL**문을 번역하여 사용자가 요구한 데이터를 찾는데 필요한 모든 과정을 담당
- ✓ 장점
 - 자연어에 가까운 구문을 사용하여 질의를 표현할 수 있다는 것
- ✓ 두 가지 인터페이스
 - 대화식 **SQL(interactive SQL)**
 - 내포된 **SQL(embedded SQL)**

SQL 개요(계속)



[그림 4.3] 관계 데이터베이스에 대한 두 가지 인터페이스

SQL 개요(계속)



[그림 4.5] SQL의 인터페이스와 구성요소

7.1~3 데이터 정의, 제약조건 및 스키마 변경

- 데이터베이스의 테이블들(릴레이션들)의 생성, 제거, 갱신 위해 사용
 - CREATE SCHEMA
 - CREATE TABLE
 - DROP TABLE
 - ALTER TABLE

데이터 정의어와 무결성 제약조건

〈표 4.4〉 릴레이션의 정의에 사용되는 데이터 타입

데이터 타입	의미
INTEGER 또는 INT	정수형
SMALLINT	작은 정수형
NUMBER(n, s) 또는 DECIMAL(n, s)	n개의 숫자에서 소수 아래 숫자가 s개인 십진수
REAL	실수형
FLOAT (n)	적어도 n개의 숫자가 표현되는 실수형
CHAR (n) 또는 CHARACTER (n)	n바이트 문자열. n을 생략하면 1
VARCHAR (n) 또는 CHARACTER VARYING (n)	최대 n바이트까지의 가변 길이 문자열
BIT (n) 또는 BIT VARYING (n)	n개의 비트열 또는 최대 n개까지의 가변 비트열
DATE	날짜형
BLOB	Binary Large Object, 멀티미디어 데이터 등을 저장

CREATE TABLE

- 새로운 기본 릴레이션을 생성하는데 사용하며, 릴레이션의 이름과 함께 각 애트리뷰트와 데이터 유형을 기술함
- **NOT NULL** 제약조건을 각 애트리뷰트에 명시할 수 있음

– Example :

```
CREATE TABLE DEPARTMENT
(
    DNAME VARCHAR(10) NOT NULL,
    DNUMBER      INTEGER      NOT NULL,
    MGRSSN       CHAR(9),
    MGRSTARTDATE CHAR(9) );
```

CREATE TABLE

- **CREATE TABLE** 명령은 **Primary Key**와 **Secondary Keys**, 그리고 참조 무결성 제약(**Foreign Keys**)을 명시할 수 있음
- **Key** 애트리뷰트들은 **Primary Key**와 **UNIQUE** 절을 통해 명시할 수 있음

- Example

```
CREATE TABLE  DEPT
(
    DNAME          VARCHAR(10)  NOT NULL,
    DNUMBER        INTEGER      NOT NULL,
    MGRSSN         CHAR(9),
    MGRSTARTDATE   CHAR(9),
    PRIMARY KEY (DNUMBER),
    UNIQUE (DNAME),
    FOREIGN KEY (MGRSSN) REFERENCES
EMP );
```

DROP TABLE

- 릴레이션(기본 테이블)과 그 정의를 제거함
 - 제거된 릴레이션은 질의(queries), 갱신(updates), 또는 다른 명령어들을 더 이상 사용하지 못함
- **Example:**
DROP TABLE DEPENDENT;

ALTER TABLE

- 기본 릴레이션에 하나의 애트리뷰트를 추가하기 위해 사용
- 이 명령이 실행할 경우, 릴레이션에 포함된 모든 튜플들은 새로 추가된 애트리뷰트에 대해 **NULL** 값으로 지정됨
 - 추가되는 열에 대해 **NOT NULL** 제약조건을 사용할 수 있을까?

- **Example:**

```
ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);
```

- 각 **EMPLOYEE** 튜플에 대해 새로운 애트리뷰트 **JOB**의 값을 별도로 입력해야 함
 - 이는 **UPDATE** 명령을 사용하여 수행

기타 데이터 정의어

- 스키마 생성(**Create Schema**)

- 새로운 데이터베이스 스키마는 스키마의 이름과 함께 기술함

```
CREATE SCHEMA Company AUTHORIZATION Jsmith;
```

- 인덱스 생성(**Create Index**)

```
CREATE UNIQUE INDEX EMPINDEX ON EMPLOYEE (EMPNO) ;
```

무결성 제약조건

□ 제약조건

```
CREATE TABLE EMPLOYEE  
  (EMPNO    INTEGER NOT NULL,                (1)  
   EMPNAME  CHAR(10) UNIQUE,                  (2)  
   TITLE    CHAR(10) DEFAULT '사원',         (3)  
   MANAGER  INTEGER,  
   SALARY   INTEGER CHECK (SALARY < 6000000), (4)  
   DNO      INTEGER CHECK (DNO IN (1,2,3,4)) DEFAULT 1, (5)  
  PRIMARY KEY (EMPNO),                        (6)  
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DEPTNO) (7)  
    ON DELETE SET DEFAULT ON UPDATE CASCADE); (8)
```

[그림 4.7] 릴레이션 정의에서 다양한 제약조건을 명시

데이터 정의어와 무결성 제약조건

□ 참조 무결성 제약조건 유지

ON DELETE NO ACTION

ON DELETE CASCADE

ON DELETE SET NULL

ON DELETE SET DEFAULT

ON UPDATE NO ACTION

ON UPDATE CASCADE

ON UPDATE SET NULL

ON UPDATE SET DEFAULT

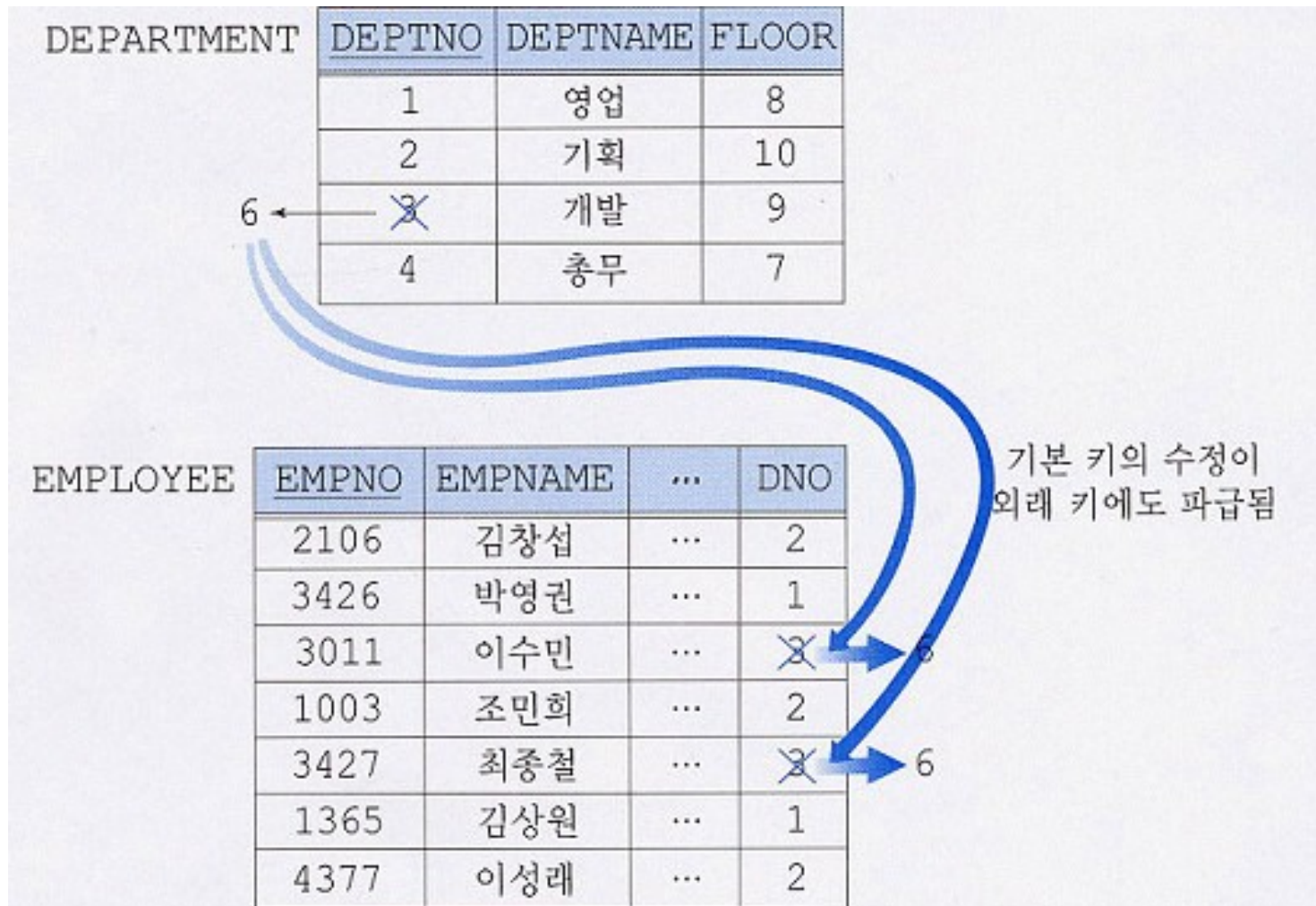
데이터 정의어와 무결성 제약조건

예 : ON UPDATE CASCADE

4.5절에서 설명할 UPDATE문을 사용하여 다음과 같이 DEPARTMENT 릴레이션의 3번 부서의 부서번호를 6번으로 수정하면 EMPLOYEE 릴레이션에서 3번 부서에 근무하는 모든 직원들의 소속 부서번호가 자동적으로 6으로 수정된다.

```
UPDATE  DEPARTMENT
SET     DEPTNO = 6
WHERE   DEPTNO = 3;
```

데이터 정의어와 무결성 제약조건



SQL에서 기본제약조건 명시

- **애트리뷰트 제약조건과 디폴트값 명시**
 - 애트리뷰트에 널 값의 허용여부 명시 : NOT NULL
 - 애트리뷰트에 디폴트 값 지정 : DEFAULT <값>
 - 애트리뷰트나 도메인 값을 특정한 값(들)으로 한정 : CHECK (값의 범위)
- **키와 참조 무결성 제약조건의 명시**
 - CREATE TABLE 명령
 - PRIMARY KEY : 릴레이션의 기본 키를 구성하는 애트리뷰트들을 명시
 - UNIQUE : 대체키(또는 보조키)를 명시
 - FOREIGN KEY 절 : 참조 무결성을 지정
 - 외래 키(foreign key)를 정의할 때 참조 무결성의 위반시 취할 동작을 명시할 수 있음;
 - 종류 : SET NULL, CASCADE, SET DEFAULT
 - 위반의 종류를 나타내는 ON DELETE나 ON UPDATE와 함께 사용해야 함
 - 대부분의 SQL 시스템들은 참조 무결성과 기본 키를 지정함

SQL에서 검색 질의

- **SQL**은 데이터베이스로부터 정보를 검색하는 문장을 가짐
 - **SELECT** 문
- 관계 대수의 **SELECT** 연산과는 무관함
- 관계 대수의 실렉션, 프로젝션, 조인, 카티션 곱 등을 결합한 것
- 관계 데이터베이스에서 가장 자주 사용됨
- 결과에 모든 애트리뷰트의 값이 동일한 두개 이상의 튜플이 존재할 수 있음

SQL에서 검색 질의 (계속)

- **SQL SELECT 문의 기본 형식은 사상(mapping) 또는 *SELECT-FROM-WHERE* 블록이라고 불림**

SELECT <attribute list>
FROM <table list>
WHERE <condition>

$\Pi_{\mathbf{L}} \sigma_{\mathbf{c}} (\mathbf{R} \mathbf{X} \mathbf{S})$

SELECT **L**
FROM **R,S**
WHERE **C**

- <attribute list> : 질의 결과에 나타나는 애트리뷰트 이름 목록
- <table list> : 질의의 대상이 되는 릴레이션 목록
- <condition> : 질의 결과에 포함될 튜플들을 표시하는 조건(부울) 식

SELECT문(계속)

□ 별칭(alias)

- ✓ 서로 다른 릴레이션에 동일한 이름을 가진 애트리뷰트가 속해 있을 때 애트리뷰트의 이름을 구분하는 방법

EMPLOYEE.DNO

FROM EMPLOYEE **AS** E, DEPARTMENT **AS** D

SELECT문(계속)

- 릴레이션의 모든 애트리뷰트나 일부들을 검색

예 : 원하는 애트리뷰트들의 이름을 열거

질의: 모든 부서의 부서번호와 부서이름을 검색하라.

$\Pi_{\text{DEPTNO, DEPTNAME}}(\text{DEPARTMENT})$

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

예 : * 를 사용하여 모든 애트리뷰트들을 검색

질의: 전체 부서의 모든 애트리뷰트들을 검색하라.

$\Pi_{\text{DEPTNO, DEPTNAME, FLOOR}}$ (DEPARTMENT)

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

□ 상이한 값들을 검색

예 : DISTINCT절을 사용하지 않을 때

질의: 모든 사원들의 직급을 검색하라.

$\Pi_{\text{TITLE}}(\text{EMPLOYEE})$

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

예 : DISTINCT절을 사용할 때

질의: 모든 사원들의 상이한 직급을 검색하라.

Π_{TITLE} (EMPLOYEE)

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

□ 특정한 튜플들의 검색

예 : WHERE절을 사용하여 검색 조건을 명시

질의: 2번 부서에 근무하는 사원들에 관한 모든 정보를 검색하라.

$\sigma_{DNO=2}(EMPLOYEE)$

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

□ 문자열 비교

$\Pi_{NAME, TITLE, DNO} (\sigma_{NAME='이순신'} (EMPLOYEE))$

“이순신” 사원의 이름, 직급, 소속 부서번호를 검색하라.

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

예 : 부울 연산자를 사용한 프레디키트

$\Pi_{NAME, SALARY} (\sigma_{TITLE='과장' \text{ AND } DNO=1} (EMPLOYEE))$

질의: 직급이 과장이면서 1번 부서에서 근무하는 직원들의 이름과 급여를 검색하라.

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

□ 부정 검색 조건

예 : 부정 연산자

$\Pi_{NAME, SALARY} (\sigma_{TITLE='과장' \text{ AND } DNO \neq 1} (EMPLOYEE))$

질의: 직급이 과장이면서 1번 부서에 속하지 않은 사원들의 이름과 급여를 검색하라.

EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)

SELECT문(계속)

□ 범위를 사용한 검색

예 : 범위 연산자

질의: 급여가 3000000원 이상이고, 4500000원 이하인 직원들의 이름, 직급, 급여를 검색하라.

```
SELECT    EMPNAME, TITLE, SALARY
FROM      EMPLOYEE
WHERE      SALARY BETWEEN 3000000 AND 4500000;
```

BETWEEN은 양쪽의 경계값을 포함하므로 이 질의는 아래의 질의와 동등하다.

```
SELECT    EMPNAME, TITLE, SALARY
FROM      EMPLOYEE
WHERE      SALARY >= 3000000 AND SALARY <= 4500000;
```

EMPNAME	TITLE	SALARY
박영권	과장	3000000
이수민	부장	4000000
조민희	과장	3000000
이성래	이사	5000000

SELECT문(계속)

□ 리스트를 사용한 검색

예 : IN

질의: 1번 부서나 3번 부서에 소속된 직원들에 관한 모든 정보를 검색하라.

```
EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)
```

SELECT문(계속)

❑ SELECT절에서 산술 연산자(+, -, *, /) 사용

예 : 산술 연산자

질의: 직급이 과장인 직원들에 대하여 이름과, 현재의 급여, 급여가 10% 인상됐을 때의 값을 검색하라.

```
EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)  
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)
```


SELECT문(계속)

□ 널값

- ✓ 널값을 포함한 다른 값과 널값을 +, - 등을 사용하여 연산하면 결과는 널이 됨
- ✓ **COUNT(*)**를 제외한 집단 함수들은 널값을 무시함
- ✓ 어떤 애트리뷰트에 들어 있는 값이 널인가 비교하기 위해서 'DNO=NULL'처럼 나타내면 안됨

```
SELECT    EMPNO, EMPNAME
FROM      EMPLOYEE
WHERE     DNO = NULL;
```

SELECT문(계속)

□ 널값(계속)

✓ 다음과 같은 비교 결과는 모두 거짓

NULL > 300

NULL = 300

NULL <> 300

NULL = NULL

NULL <> NULL

✓ 올바른 표현

```
SELECT  EMPNO, EMPNAME
FROM    EMPLOYEE
WHERE   DNO IS NULL;
```

SELECT문(계속)

〈표 4.6〉 unknown에 대한 OR 연산

	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

〈표 4.7〉 unknown에 대한 AND 연산

	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

SELECT문(계속)

〈표 4.8〉 unknown에 대한 NOT 연산

true	false
false	true
unknown	unknown

SELECT문(계속)

□ ORDER BY절

- ✓ 사용자가 **SELECT**문에서 질의 결과의 순서를 명시하지 않으면 **DBMS**가 튜플들을 검색한 임의의 순서대로 사용자에게 제시됨
- ✓ **ORDER BY**절에서 하나 이상의 애트리뷰트를 사용하여 검색 결과를 정렬할 수 있음
- ✓ **ORDER BY**절은 **SELECT**문에서 가장 마지막에 사용되는 절
- ✓ 디폴트 정렬 순서는 오름차순(**ASC**)
- ✓ **DESC**를 지정하여 정렬 순서를 내림차순으로 지정할 수 있음
- ✓ 넓은 오름차순에서는 가장 마지막에 나타나고, 내림차순에서는 가장 앞에 나타남
- ✓ **SELECT**절에 명시한 애트리뷰트들을 사용해서 정렬해야 함

SELECT문(계속)

예 : ORDER BY

질의: 2번 부서에 근무하는 사원들의 급여, 직급, 이름을 검색하여 급여의 오름차순으로 정렬하라.

```
EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)
```

집단함수

- ❑ 데이터베이스에서 검색된 여러 튜플들의 집단에 적용되는 함수
 - 각 집단 함수는 한 릴레이션의 한 개의 애트리뷰트에 적용되어 단일 값을 반환함
 - **SELECT**절과 **HAVING**절에만 나타날 수 있음
 - **COUNT(*)**를 제외하고는 모든 집단 함수들이 널값을 제거한 후 남아 있는 값들에 대해서 집단 함수의 값을 구함
 - **COUNT(*)** : 결과 릴레이션의 모든 행들의 총 개수
 - **COUNT(애트리뷰트)** : 해당 애트리뷰트에서 널값이 아닌 값들의 개수
 - 키워드 **DISTINCT**가 집단 함수 앞에 사용되면 집단 함수가 적용되기 전에 먼저 중복을 제거함

집단함수(계속)

〈표 4.9〉 집단 함수의 기능

집단 함수	기능
COUNT	튜플이나 값들의 개수
SUM	값들의 합
AVG	값들의 평균값
MAX	값들의 최대값
MIN	값들의 최소값

집단함수(계속)

예 : MAX

질의: 모든 사원들의 평균 급여와 최대 급여를 검색하라.

```
SELECT      AVG (SALARY) , MAX (SALARY)
FROM        EMPLOYEE;
```

AVG (SALARY)	MAX (SALARY)
2928571	5000000

그룹화

- ❑ GROUP BY절에 사용된 애트리뷰트 (그룹화 애트리뷰트) 에 동일한 값을 갖는 튜플들이 각각 하나의 그룹으로 묶임
 - ✓ 각 그룹에 대하여 결과 릴레이션에 하나의 튜플이 생성됨
 - ✓ SELECT절에는 각 그룹마다 하나의 값을 갖는 애트리뷰트, 집단 함수, 그룹화에 사용된 애트리뷰트들만 나타날 수 있음

그룹화

예 : 그룹화

질의: 모든 사원들에 대해서 사원들이 속한 부서번호별로 그룹화하고, 각 부서마다 부서번호, 평균 급여, 최대 급여를 검색하라.

```
SELECT      DNO, AVG (SALARY) , MAX (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO;
```

그룹화(계속)

EMPLOYEE

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
3426	박영권	과장	4377	3000000	1
1365	김상원	사원	3426	1500000	1
2106	김창섭	대리	1003	2500000	2
1003	조민희	과장	4377	3000000	2
4377	이성래	이사	^	5000000	2
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3



DNO	AVG (SALARY)	MAX (SALARY)
1	2250000	3000000
2	3500000	5000000
3	2750000	4000000

HAVING

- 어떤 조건을 만족하는 그룹들에 대해서만 집단 함수를 적용할 수 있음
 - ✓ 각 그룹마다 하나의 값을 갖는 애트리뷰트를 사용하여 각 그룹이 만족해야 하는 조건을 명시함
 - ✓ **HAVING**절은 그룹화 애트리뷰트에 같은 값을 갖는 튜플들의 그룹에 대한 조건을 나타내고, 이 조건을 만족하는 그룹들만 질의 결과에 나타남
 - ✓ **HAVING**절에 나타나는 애트리뷰트는 반드시 **GROUP BY**절에 나타나거나 집단 함수에 포함되어야 함

HAVING(계속)

예 : 그룹화

질의: 모든 직원들에 대해서 직원들이 속한 부서번호별로 그룹화하고, 평균 급여가 2500000원 이상인 부서에 대해서 부서번호, 평균 급여, 최대 급여를 검색하라.

```
SELECT      DNO, AVG (SALARY) , MAX (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO
HAVING      AVG (SALARY) >= 2500000;
```

SELECT문(계속)

EMPLOYEE

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
3426	박영권	과장	4377	3000000	1
1365	김상원	사원	3426	1500000	1
2106	김창섭	대리	1003	2500000	2
1003	조민희	과장	4377	3000000	2
4377	이성래	이사	^	5000000	2
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3

그룹

GROUP BY

DNO	AVG (SALARY)	MAX (SALARY)
1	2250000	3000000
2	3500000	5000000
3	2750000	4000000

HAVING

DNO	AVG (SALARY)	MAX (SALARY)
2	3500000	5000000
3	2750000	4000000

집합연산

□ 집합 연산

- ✓ 집합 연산을 적용하려면 두 릴레이션이 합집합 호환성을 가져야 함
- ✓ UNION(합집합), EXCEPT(차집합), INTERSECT(교집합), UNION ALL(합집합), EXCEPT ALL(차집합), INTERSECT ALL(교집합)

합집합(계속)

예 : 합집합

질의: 김창섭이 속한 부서이거나 개발 부서의 부서번호를 검색하라.

```
(SELECT      DNO
FROM      EMPLOYEE
WHERE      EMPNAME = '김창섭' )
UNION
(SELECT      DEPTNO
FROM      DEPARTMENT
WHERE      DEPTNAME = '개발' );
```

DNO
2
3

조인

- 조인은 두 개의 릴레이션으로부터 연관된 튜플들을 결합
 - ✓ 조인의 일반적인 형식은 아래의 **SELECT**문과 같이 **FROM**절에 두 개 이상의 릴레이션들이 열거되고, 두 릴레이션에 속하는 애트리뷰트들을 비교하는 조인 조건이 **WHERE**절에 포함됨
 - ✓ 조인 조건은 두 릴레이션 사이에 속하는 애트리뷰트 값들을 비교 연산자로 연결한 것

```
SELECT    ...  
FROM      R, S  
WHERE     R.A <비교 연산자> S.B ;
```

↑
조인 조건

조인(계속)

□ 조인(계속)

- ✓ 조인 조건을 생략했을 때와 조인 조건을 틀리게 표현했을 때는 어떻게 될까? 카테시안 프로덕트
- ✓ 조인 질의가 수행되는 과정
 1. 조인 조건을 만족하는 튜플들을 찾는다.
 2. 이 튜플들로부터 **SELECT**절에 명시된 애트리뷰트들만 프로젝션한다.
 3. 필요하다면 중복을 배제한다.
- ✓ 애트리뷰트 이름이 동일한 경우
 - 릴레이션 이름이나 튜플 변수를 사용
(예) $R.A < S.A$

조인(계속)

예 : 조인 질의

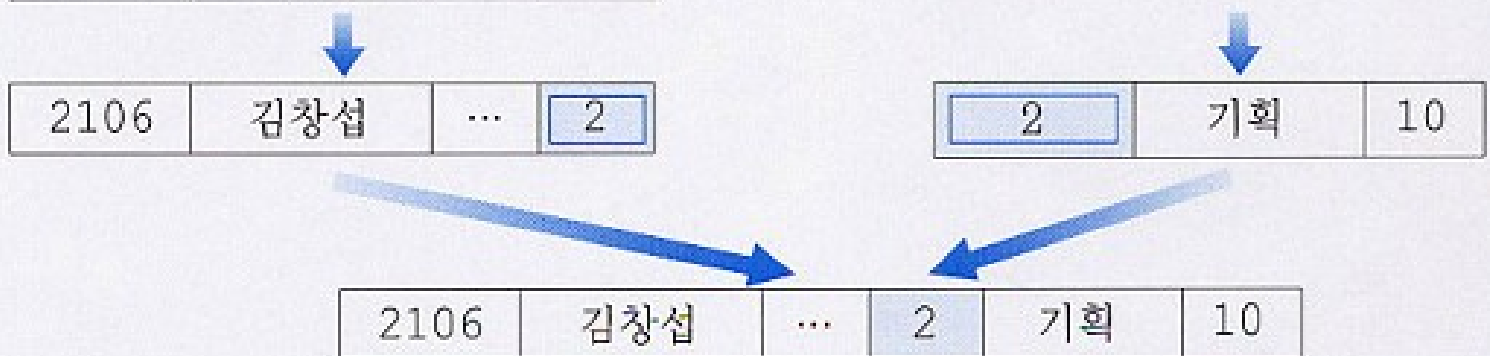
질의: 모든 사원의 이름과 이 사원이 속한 부서 이름을 검색하라.

```
EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)  
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)
```

조인(계속)

EMPLOYEE	EMPNO	EMPNAME	...	DNO
	2106	김창섭	...	2
	3426	박영권	...	1
	3011	이수민	...	3
	1003	조민희	...	2
	3427	최종철	...	3
	1365	김상원	...	1
	4377	이성래	...	2

DEPARTMENT	DEPTNO	DEPTNAME	FLOOR
	1	영업	8
	2	기획	10
	3	개발	9
	4	총무	7



조인(계속)

최종 결과 릴레이션

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO	DEPTNAME	FLOOR
2106	김창섭	대리	1003	2500000	2	기획	10
3426	박영권	과장	4377	3000000	1	영업	8
3011	이수민	부장	4377	4000000	3	개발	9
1003	조민희	과장	4377	3000000	2	기획	10
3427	최종철	사원	3011	1500000	3	개발	9
1365	김상원	사원	3426	1500000	1	영업	8
4377	이성래	이사	^	5000000	2	기획	10

자체조인

□ 자체 조인(self join)

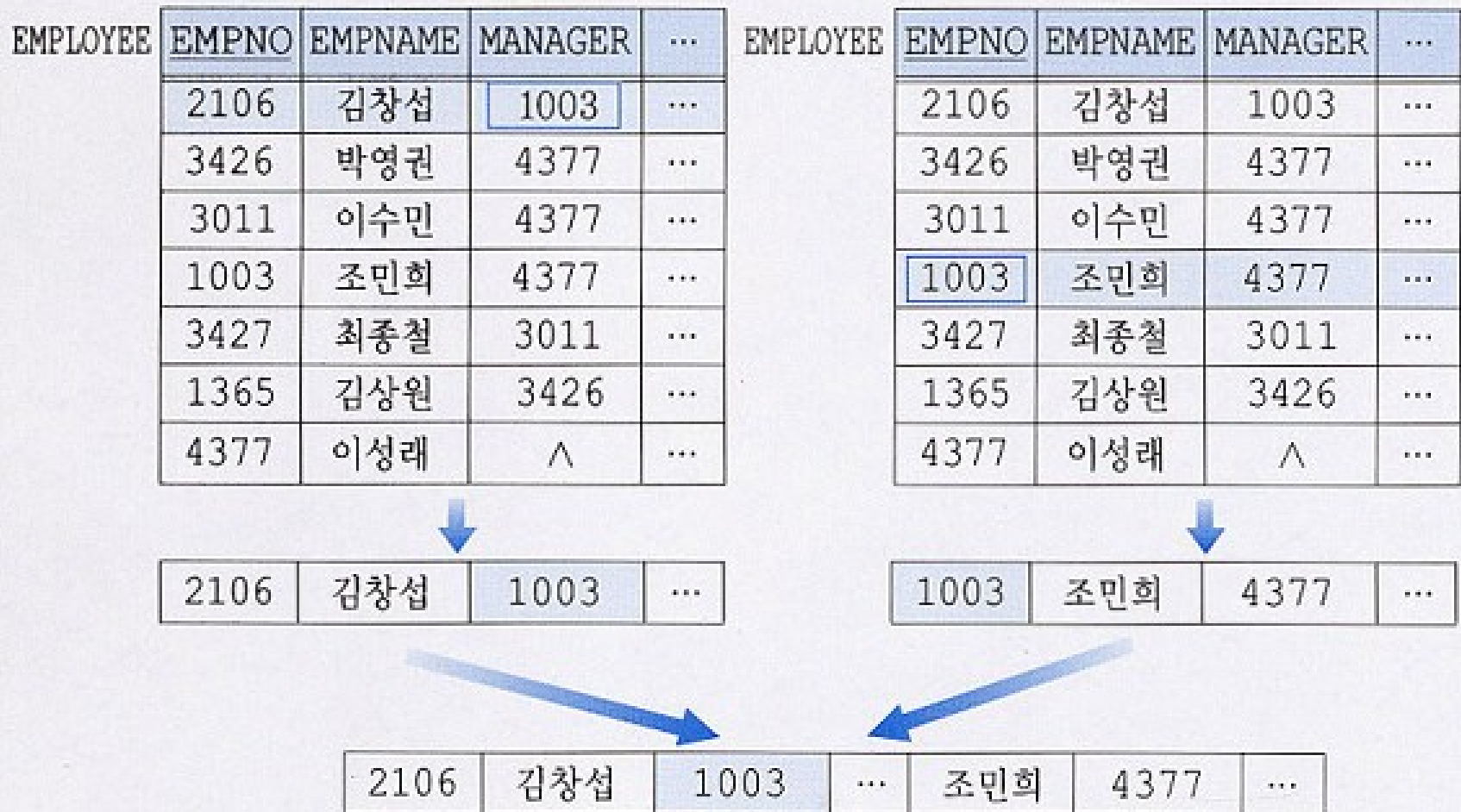
- ✓ 한 릴레이션에 속하는 튜플을 동일한 릴레이션에 속하는 튜플들과 조인하는 것
- ✓ 실제로는 한 릴레이션이 접근되지만 **FROM**절에 두 릴레이션이 참조되는 것처럼 나타내기 위해서 그 릴레이션에 대한 별칭을 두 개 지정해야 함

예 : 자체 조인

질의: 모든 사원에 대해서 사원의 이름과 직속 상사의 이름을 검색하라.

```
SELECT      E.EMPNAME, M.EMPNAME
FROM        EMPLOYEE E, EMPLOYEE M
WHERE       E.MANAGER = M.EMPNO;
```

자체조인



자체조인

최종 결과 릴레이션

E. EMPNAME	M. EMPNAME
김창섭	조민희
박영권	이성래
이수민	이성래
조민희	이성래
최종철	이수민
김상원	박영권

연습 문제

예 : 조인과 ORDER BY의 결합

질의: 모든 사원에 대해서 소속 부서이름, 사원의 이름, 직급, 급여를 검색하라. 부서 이름에 대해서 오름차순, 부서이름이 같은 경우에는 SALARY에 대해서 내림차순으로 정렬하라.

```
EMP(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPT(DEPTNO, DEPTNAME, FLOOR)
```

중첩질의

□ 중첩 질의(nested query)

- ✓ 외부 질의의 **WHERE**절에 다시 **SELECT ... FROM ... WHERE** 형태로 포함된 **SELECT**문
- ✓ **부질의(subquery)**라고도 함
- ✓ **INSERT, DELETE, UPDATE**문에도 사용될 수 있음
- ✓ 중첩 질의의 결과로 한 개의 스칼라값(단일 값), 한 개의 애트리뷰트로 이루어진 릴레이션, 여러 애트리뷰트로 이루어진 릴레이션이 반환될 수 있음

중첩질의

외부 질의



SELECT...

FROM...

WHERE...

(SELECT. . .

FROM . . .

WHERE . . .);

중첩 질의

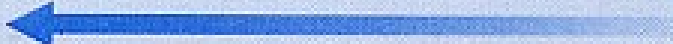
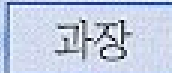
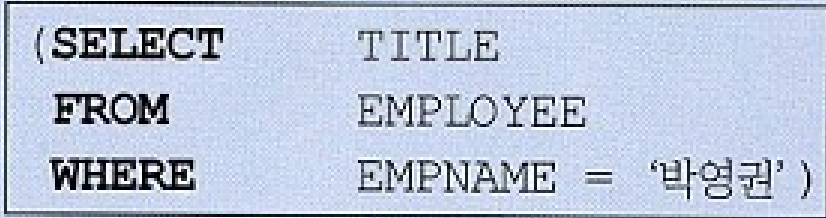


[그림 4.10] 중첩 질의의 구조

중첩질의

- 한 개의 스칼라값이 반환되는 경우

질의: 박영권과 같은 직급을 갖는 모든 사원들의 이름과 직급을 검색하라.

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

EMPNAME	TITLE
박영권	과장
조민희	과장

중첩질의

- 한 개의 애트리뷰트로 이루어진 릴레이션이 반환되는 경우
 - ✓ 중첩 질의의 결과로 한 개의 애트리뷰트로 이루어진 다수의 튜플들이 반환될 수 있음
 - ✓ 외부 질의의 WHERE 절에서 IN, ANY(SOME), ALL, EXISTS와 같은 연산자를 사용해야 함
 - **IN** : 한 애트리뷰트가 값들의 집합에 속하는가를 테스트할 때 사용됨
 - **ANY** : 한 애트리뷰트가 값들의 집합에 속하는 하나 이상의 값들과 어떤 관계를 갖는가를 테스트하는 경우에 사용
 - **ALL** : 한 애트리뷰트가 값들의 집합에 속하는 모든 값들과 어떤 관계를 갖는가를 테스트하는 경우에 사용
 - **EXISTS** : 여러 애트리뷰트로 이루어진 릴레이션이 반환되는 경우에 사용

중첩질의(계속)

예 : IN

(3426 IN

2106
3426
3011

)은 참이다.

(1365 IN

2106
3426
3011

)은 거짓이다.

(1365 NOT IN

2106
3426
3011

)은 참이다.

중첩질의(계속)

예 : ANY

(3000000 < ANY

2500000
3000000
4000000

)은 참이다.

(4000000 < ANY

2500000
3000000
4000000

)은 거짓이다.

중첩질의(계속)

예 : ALL

(3000000 < ALL

2500000
3000000
4000000

)은 거짓이다.

(1500000 < ALL

2500000
3000000
4000000

)은 참이다.

(3000000 = ALL

2500000
3000000
4000000

)은 거짓이다.

(1500000 <> ALL

2500000
3000000
4000000

)은 참이다.

중첩질의(계속)

예 : IN을 사용한 질의

질의: 영업부나 개발부에 근무하는 직원들의 이름을 검색하라.

```
SELECT  EMPNAME  
FROM    EMPLOYEE  
WHERE   DNO IN
```

(1, 3)

```
( SELECT  DEPTNO  
  FROM    DEPARTMENT  
  WHERE   DEPTNAME = '영업' OR DEPTNAME = '개발' ) ;
```

중첩질의(계속)

이 질의를 중첩 질의를 사용하지 않은 다음과 같은 질의로 나타낼 수 있다. 실제로, 중첩 질의를 사용하여 표현된 대부분의 질의를 중첩 질의가 없는 질의로 표현할 수 있다.

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

EMPNAME
박영권
이수민
최종철
김상원

중첩질의(계속)

- ❑ 여러 애트리뷰트들로 이루어진 릴레이션이 반환되는 경우
 - ✓ EXISTS 연산자를 사용하여 중첩 질의의 결과가 빈 릴레이션인지 여부를 검사함
 - ✓ 중첩 질의의 결과가 빈 릴레이션이 아니면 참이 되고, 그렇지 않으면 거짓

중첩질의(계속)

예 : EXISTS를 사용한 질의

질의: 영업부나 개발부에 근무하는 직원들의 이름을 검색하라.

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

EMPNAME
박영권
이수민
최종철
김상원

SQL 질의에 대한 요약

- **SQL에서 하나의 질의는 6개의 절로 구성**
 - 필수적으로 질의에 나타내야 하는 두개의 절은 **SELECT**와 **FROM** 절임
- **6개의 절은 다음 순서로 명시함**

SELECT	<attribute list>
FROM	<table list>
[WHERE	<condition>]
[GROUP BY	<grouping attribute(s)>]
[HAVING	<group condition>]
[ORDER BY	<attribute list>]

SQL 질의에 대한 요약 (계속)

- **SELECT** 절은 결과에 포함될 애트리뷰트들이나 함수를 나열함
- **FROM** 절은 질의에서 필요한 모든 릴레이션(별명)들을 명시함
 - 중첩 질의들에 사용되는 릴레이션들은 명시하지 않음
- **WHERE** 절은 조인조건을 포함하여 **FROM** 절에 명시된 릴레이션들로부터 튜플들을 선택하기 위한 조건들을 명시
- **GROUP BY**절은 그룹화 애트리뷰트를 명시
- **HAVING** 절은 선택된 튜플들의 그룹들에 대한 조건을 명시
- **ORDER BY**절은 질의 결과를 출력하는 순서를 명시
- 질의는 **WHERE**절, **GROUP BY**절과 **HAVING**절의 순서로 적용함으로써 평가됨

INSERT, DELETE, UPDATE문

□ INSERT문

- ✓ 기존의 릴레이션에 튜플을 삽입
- ✓ 참조하는 릴레이션에 튜플이 삽입되는 경우에는 참조 무결성 제약조건을 위배할 수 있음
- ✓ 릴레이션에 한 번에 한 튜플씩 삽입하는 것과 한 번에 여러 개의 튜플들을 삽입할 수 있는 것으로 구분
- ✓ 릴레이션에 한 번에 한 튜플씩 삽입하는 **INSERT문**

INSERT

INTO 릴레이션 (애트리뷰트1, ..., 애트리뷰트_n)

VALUES (값1, ..., 값_n);

INSERT, DELETE, UPDATE문(계속)

예 : 한 개의 튜플을 삽입

질의: DEPARTMENT 릴레이션에 (5, 연구, ^) 튜플을 삽입하는 INSERT문은 아래와 같다.

```
INSERT INTO DEPARTMENT  
VALUES (5, '연구', );
```

DEPARTMENT

DEPTNO	DEPTNAME	FLOOR
1	영업	8
2	기획	10
3	개발	9
4	총무	7
5	연구	^

INSERT, DELETE, UPDATE문(계속)

□ INSERT문(계속)

- ✓ 릴레이션에 한 번에 여러 개의 튜플들을 삽입하는 **INSERT**문

INSERT

INTO 릴레이션 (애트리뷰트1, ..., 애트리뷰트n)

SELECT ... **FROM** ... **WHERE** ...;

예 : 여러 개의 튜플을 삽입

질의: EMPLOYEE 릴레이션에서 급여가 3000000 이상인 직원들의 이름, 직급, 급여를 검색하여 HIGH_SALARY라는 릴레이션에 삽입하라. HIGH_SALARY 릴레이션은 이미 생성되어 있다.

EMP(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPT(DEPTNO, DEPTNAME, FLOOR)

INSERT, DELETE, UPDATE문(계속)

❑ DELETE문

- ✓ 삭제 연산은 한 릴레이션으로부터 한 개 이상의 튜플들을 삭제함
- ✓ 참조되는 릴레이션의 삭제 연산의 결과로 참조 무결성 제약조건이 위배될 수 있음
- ✓ **DELETE문의 구문**

DELETE

FROM 릴레이션

WHERE 조건;

INSERT, DELETE, UPDATE문(계속)

예 : DELETE문

질의: DEPARTMENT 릴레이션에서 4번 부서를 삭제하라.

```
EMPLOYEE(EMPNO, NAME, TITLE, MANAGER, SALARY, DNO)
DEPARTMENT(DEPTNO, DEPTNAME, FLOOR)
```

INSERT, DELETE, UPDATE문(계속)

□ UPDATE문

- ✓ 한 릴레이션에 들어 있는 튜플들의 애트리뷰트 값들을 수정
- ✓ 기본 키나 외래 키에 속하는 애트리뷰트의 값이 수정되면 참조 무결성 제약조건을 위배할 수 있음
- ✓ UPDATE문의 구문

UPDATE 릴레이션
SET 애트리뷰트 = 값 또는 식 [, ...]
WHERE 조건;

예 : UPDATE문

질의: 사원번호가 2106인 사원의 소속 부서를 3번 부서로 옮기고, 급여를 5% 올려라.

```
UPDATE EMPLOYEE
SET     DNO = 3, SALARY = SALARY * 1.05
WHERE  EMPNO = 2106;
```

SQL에서 뷰: 예제

- 다른 WORKS_ON 테이블의 명시

```
CREATE TABLE WORKS_ON_NEW AS  
SELECT FNAME, LNAME, PNAME, HOURS  
FROM EMPLOYEE, PROJECT, WORKS_ON  
WHERE SSN=ESSN AND PNO=PNUMBER  
GROUP BY PNAME;
```

가상 테이블의 사용

- 새로이 생성된 테이블(뷰)에서 SQL 질의를 명시함

```
SELECT FNAME, LNAME  
FROM WORKS_ON_NEW  
WHERE PNAME='Seena' ;
```

- 어떤 뷰가 더 이상 필요하지 않으면 뷰를 제거함

```
DROP WORKS_ON_NEW;
```

효율적인 뷰 구현

- **질의수정(query modification) 방식**
 - 뷰에 대한 질의를 기본 테이블들에 대한 질의로 변환하여 처리
 - 단점: 복잡한 질의로 정의된 뷰들은 비효율적
 - 특히 짧은 시간 내에 뷰에 많은 질의가 적용될 때
- **뷰의 실체화(view materialization)**
 - 임의 뷰 테이블을 물리적으로 생성하고 유지하는 방식
 - 가정: 뷰에 다른 질의들이 사용됨
 - 문제점: 기본 테이블이 갱신되면 뷰 테이블도 변경해야 함

뷰의 갱신

- 집단함수를 사용하지 않는 단일 뷰의 갱신
 - 뷰의 갱신은 단일 기본 테이블에 대한 갱신으로 사상될 수 있음
- 조인을 포함하는 뷰의 갱신
 - 기본 릴레이션들에 대한 갱신 동작으로 사상될 수 있음 (항상 가능한 것은 아님)
- 그룹화와 집단함수를 사용하여 정의된 뷰는 갱신할 수 없음
- 일반적으로 다수의 테이블을 조인하여 정의한 뷰는 갱신할 수 없음

7.9 SQL의 기타 기능

- **데이터베이스 연결 기능**
 - 내포된 (혹은 동적) SQL, SQL/CLI, ODBC, SQL/PSM
- **권한 기능**
 - SQL은 데이터베이스 사용자에게 권한을 부여하고 취소하는 기능을 제공함
- **호스트 언어와 결합되어 사용**
 - SQL은 C, C++, COBOL, JAVA, PASCAL 등과 같은 범용 프로그래밍 언어 내에서 사용될 수 있음
 - Embedded SQL/C, C++, COBOL, JAVA, PASCAL
- **트랜잭션 기능**
 - SQL은 트랜잭션 제어 명령문을 가짐
- **기타 유용한 명령어**
 - 상용 DBMS는 SQL 명령 이외에도 물리적 데이터베이스 설계 매개변수와 릴레이션들을 위한 파일 구조, 그리고 인덱스와 같은 접근경로를 명시하기 위한 명령어의 집합을 가지고 있음

요약

- **SQL2에서 데이터 정의어, 제약조건 및 스키마 변경**
- **SQL에서의 기본질의**
 - Select/From/Where 절의 사용
- **더 복잡한 SQL 질의들**
 - 중첩질의, 집단함수, 집합과 널, 그룹핑 연산, Select/From/Where 절의 사용
- **SQL에서 삽입, 삭제, 갱신 구문**
- **SQL 뷰**
- **주장으로 추가적인 제약조건 명시**
- **SQL의 부가적인 기능들**

최종 텀프로젝트

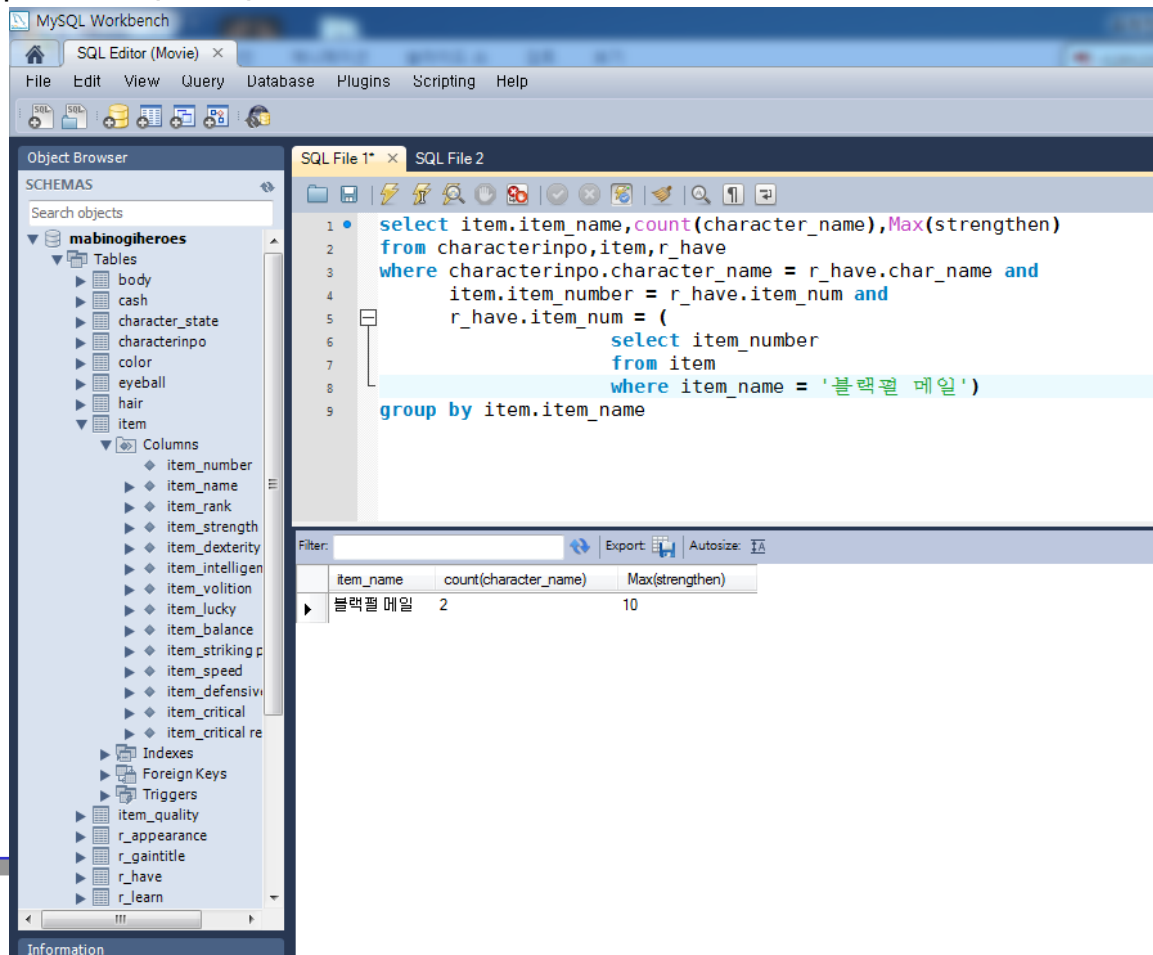
1. 설계한 관계형 데이터 모델에 대하여 다음 작업을 수행하라.
 - 가상의 데이터를 릴레이션 당 최소 **10**개씩 입력하라.
 - 질의어 **10**개를 선정하여 **SQL**로 작성하라.
 - **DBMS**에서 질의를 하여 나온 캡처한다.
2. 전체 텀프로젝트 과정에 대한 설명을 **PPT**로 작성하여 제출하라.
 - 한학기를 마치면서 또한 텀프로젝트를 하면서 느낀 소감을 첨부하라.

과제 제출일 : **12월 22일** 자정

과제 제출방법 : **e-class** 제출

질의어 예 - 특정 아이템의 수와 가장 높은 강화수치

- 질의어 : 블랙펄메일을 가지고 있는 유저의 수와 가장 높은 강화 수치를 보여준다.
 - 집계함수를 이용하여 해당 아이템이 어느 정도 물량이 풀렸는가와 가장 높은 강화수치를 조회한다.



The screenshot shows the MySQL Workbench interface. On the left, the Object Browser displays the 'mabinogiheroes' database schema, including tables like 'body', 'cash', 'character_state', 'characterinfo', 'color', 'eyeball', 'hair', 'item', and columns like 'item_number', 'item_name', 'item_rank', etc. The main SQL Editor window shows a query in 'SQL File 2' that uses a subquery to find the highest 'strengthen' value for '블랙펄 메일' (Black Pearl Mail) and then counts the number of users who have this item.

```
1 select item.item_name, count(character_name), Max(strengthen)
2 from characterinfo, item, r_have
3 where characterinfo.character_name = r_have.char_name and
4 item.item_number = r_have.item_num and
5 r_have.item_num = (
6     select item_number
7     from item
8     where item_name = '블랙펄 메일')
9 group by item.item_name
```

Below the query, the results are displayed in a table:

item_name	count(character_name)	Max(strengthen)
블랙펄 메일	2	10