데이터베이스

- 순서 : Ch4(관계대수와 SQL)
- 학기 : 2018학년도 2학기
- 학과 : 가천대학교 컴퓨터공학과 2학년
- 교수 : 박양재

데이터베이스

- 목차

4장. 관계 대수와 SQL

- 관계 데이터베이스에 정보를 저장하고 검색하는 언어 필요
- 관계 데이터 모델에서 지원되는 두 가지 정형적인 언어
  - ✓ 관계 해석(relational calculus)
    - 원하는 데이터만 명시하고 질의를 어떻게 수행할 것인가는 명시하지 않는 선언적인 언어
  - ✓ 관계 대수(relational algebra)-E.F. Codd 제안
    - 어떻게 질의를 수행할 것인가를 명시하는 절차적 인어
    - 관계 대수는 상용 관계 DBMS들에서 널리 사용되는 SQL의 이론적인 기초
    - 관계 대수는 SQL을 구현하고 최적화하기 위해 DBMS의 내부 언어로서도 사용됨
- SQL(Structured Query Language)
  - ✓ 상용 관계 DBMS들의 사실상의 표준 질의어인 SQL을 이해하고 사용할 수 있는 능력은 매우 중요함
  - ✓ 사용자는 SQL을 사용하여 관계 데이터베이스에 릴레이션(테이블)을 정의하고, 관계 데이터베이스에서 정보를 검색하고, 관계 데이터베이스를 갱신하며, 여러 가지 무결성 제약조건들을 명시할 수 있고, 고급 프로그래밍 언어(C, C++, 자바.)에 내포시켜 응용프로그램을 작성할 수 있다.

4.1 관계 대수

❑ 관계 대수

create table NEWTABLE as
select * from EMP;

- ✓ 기존의 릴레이션들로부터 새로운 릴레이션을 생성함
- ✓ 릴레이션이나 관계 대수식(이것의 결과도 릴레이션임)에 연산자들을 적용하여 보다 복잡한 관계 대수식을 점차적으로 만들 수 있음
- ✓ 기본적인 연산자들의 집합으로 이루어짐
- ✓ 산술 연산자와 유사하게 단일 릴레이션이나 두 개의 릴레이션을 입력으로 받아 하나의 결과 릴레이션을 생성함

  예: 관계연산과 산술연산은 유사하다(1 + 5 = 6, 서로 다른 릴레이션의 연산으로 새로운 릴레이션을 생성)

  산술연산자 (+, −, *, /)−두개의 입력으로 하나의 결과를 얻음

  −이항연산자(binary operator).

  (−7)은 양수를 음수로 변환 −단항 연산자(unary operator)
- ✓ 결과 릴레이션은 또 다른 관계 연산자의 입력으로 사용될 수 있음

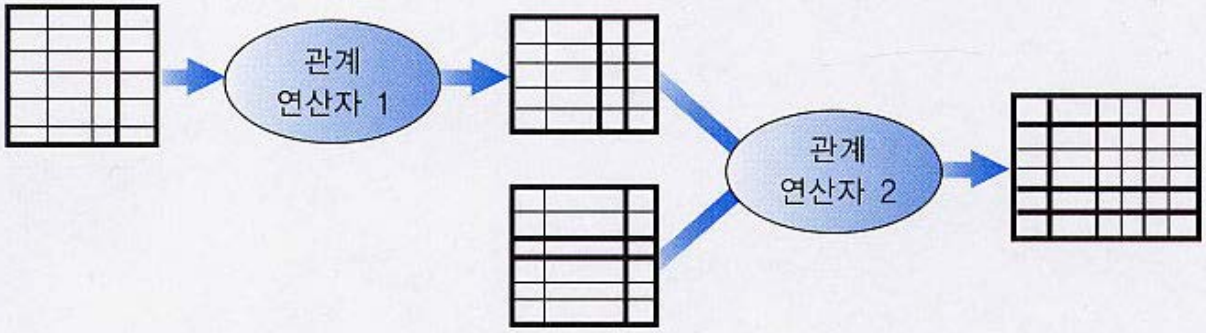
4.1 관계 대수(계속)

❑관계 대수

✓관계 대수 연산자들은 중복된 투플을 갖고 있지 않은 릴레이션들에 적용되며 결과로 생기는 릴레이션에도 중복된 투플이 존재하지 않는다.

✓이유 : 릴레이션이 투플들의 집합이기 때문이다.

✓단항 연산자 (관계연산자1), 이항 연산자(관계연산자 2)



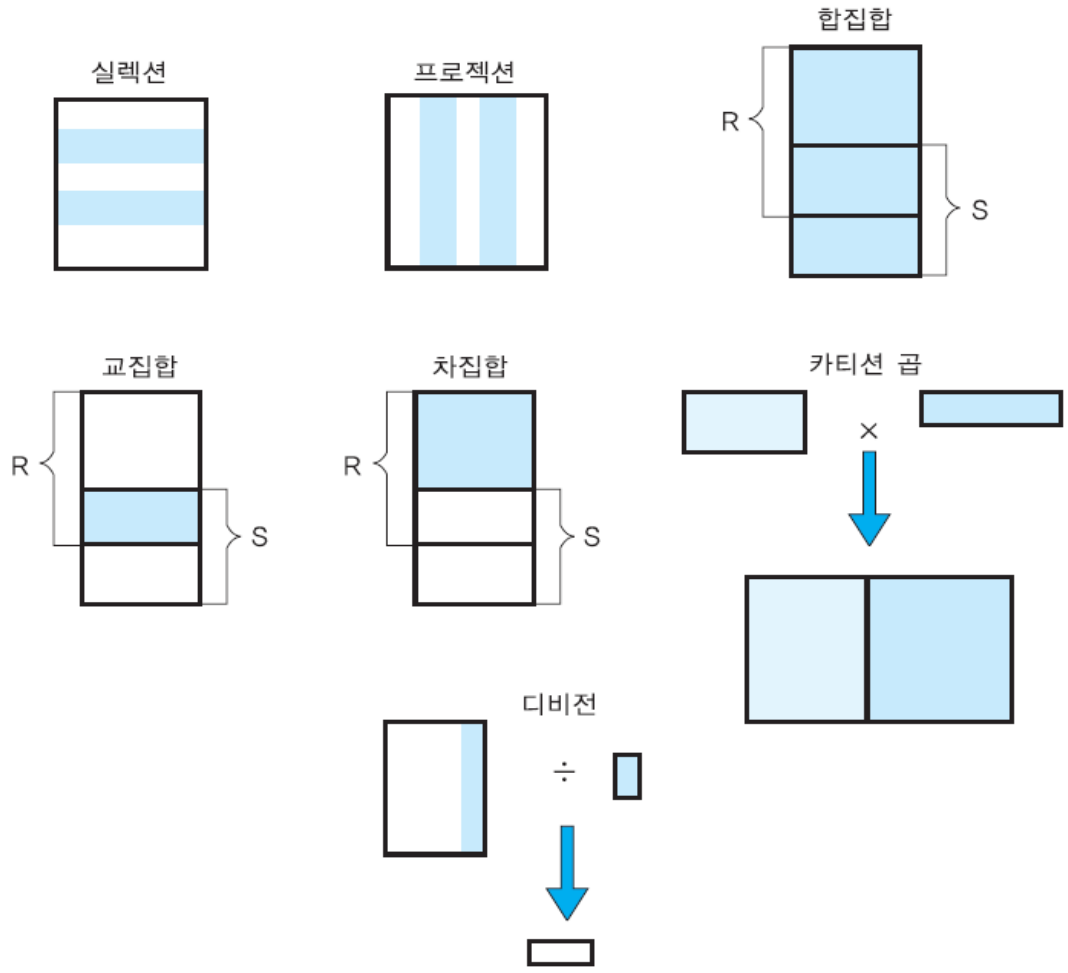
[그림 4.1] 관계 연산자

4.1 관계 대수(계속)

〈표 4.1〉 관계 연산자들의 종류와 표기법

분류	연산자	표기법	단항 또는 이항
필수적인 연산자	실렉션(selection) 튜플을 꺼내오는 것	σ	단항
	프로젝션(projection) 어트리뷰트를 꺼내오는 것 (뒤 그림 참조)	π	단항
	합집합(union)	∪	이항
	차집합(difference)	−	이항
	카티션 곱(Cartesian product) 매트릭스(행렬)에서 곱하는거랑 같음 - 모든 원소들을 다 곱해줘야함	×	이항
편의를 위해 유도된 연산자	교집합(intersection)	∩	이항
	세타 조인(theta join)	⋈	이항
	동등 조인(equijoin)	⋈	이항
	자연 조인(natural join)	*	이항
	세미 조인(semijoin)	⋉	이항
	디비전(division)	÷	이항

4.1 관계 대수(계속)



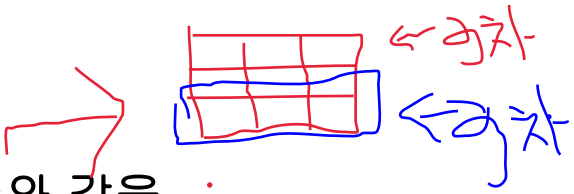
[그림 4.2] 관계 연산자들의 기능

4.1 관계 대수(계속)

❑ 실렉션 연산자

- ✓ 한 릴레이션에서 실렉션 조건(selection condition)을 만족하는 투플들의 부분 집합을 생성함
- ✓ 단항 연산자
- ✓ 결과 릴레이션의 차수는 입력 릴레이션의 차수와 같음

degree(attribute의 개수)



- ✓ 결과 릴레이션의 카디날리티는 항상 원래 릴레이션의 카디날리티보다 작거나 같음

- > 추출했기 때문에 같거나 적어짐

- ✓ 실렉션 조건은 애트리뷰트들에 대하여 명시된 부울식(Boolean expression), 프레디키트(predicate)라고도 함

예측하다라는 뜻(조건식)

- ✓ 실렉션 조건은 일반적으로 릴레이션의 임의의 애트리뷰트와 상수, = , <>, <=, <, >=, > 등의 비교 연산자, AND, OR, NOT 등의 부울 연산자를 포함할 수 있음

4.1 관계 대수(계속)

❑ 실렉션 연산자

✓ 실렉션 조건은 일반적으로 릴레이션의 임의의 애트리뷰트와 상수, = , <>, <=, <, >=, > 등의 비교 연산자, AND, OR, NOT 등의 부울 연산자를 포함할 수 있음

✓ 예: <애트리뷰트> 비교연산자 <상수>

<애트리뷰트> 비교 연산자 (애트리뷰트>

<조건> AND <조건>

<조건> OR <조건>

NOT <조건>

✓ 형식: σ [람다]라고 읽음 <실렉션조건> <릴레이션>

4.1 관계 대수(계속)

❑ 실렉션 연산자(계속)

예 : 실렉션

질의: EMPLOYEE 릴레이션에서 3번 부서에 소속된 사원들을 검색하라.

EMPLOYEE

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

원하는 투플

$\sigma_{DNO=3}(EMPLOYEE)$

- > 해석하면 DNO가 3인 튜플을 선택 (EMPLOYEE)에서

차수는 같고
카디날리티는 감소했다

RESULT

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3

SQL로 표현하면
select * from EMPLOYEE where DNO=3;

4.1 관계 대수(계속)

❑ 프로젝션 연산자

- ✓ 한 릴레이션의 애트리뷰트들의 부분 집합을 구함
- ✓ 결과로 생성되는 릴레이션은 <애트리뷰트 리스트>에 명시된 애트리뷰트들만 가지며, 애트리뷰트 리스트 순서대로 릴레이션에 나타난다.
- ✓ 새로운 릴레이션의 차수는 애트리뷰트 리스트내의 애트리뷰트 개수와 같다.
- ✓ 실렉션의 결과 릴레이션에는 중복 투플이 존재할 수 없지만, 프로젝션 연산의 결과 릴레이션에는 중복된 투플들이 존재할 수 있음
  <애트리뷰트 리스트>에 키가 포함되어 있으면 결과 릴레이션에 중복된 투플이 존재하지 않는다.

- ✓ 형식 $\pi_{\langle 애트리뷰트\ 리스트\rangle}\langle 릴레이션\rangle$

4.1 관계 대수(계속)

예 : 프로젝션

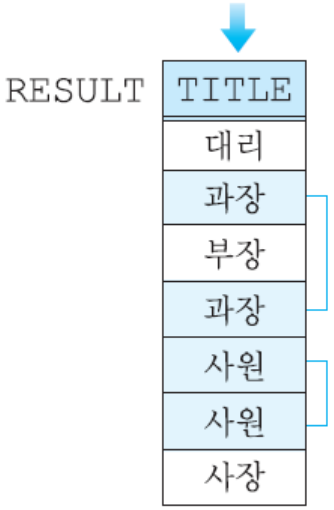
질의: 모든 사원들의 직급을 검색하라.

EMPLOYEE

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

$\pi_{TITLE}(EMPLOYEE)$

4.1 관계 대수(계속)



중복이 존재하는 릴레이션

distinct 키워드 사용

중복이 제거된 릴레이션

4.1 관계 대수(계속)

❑ 집합 연산자

- ✓ 릴레이션이 투플들의 집합이기 때문에 기존의 집합 연산이 릴레이션에 적용됨
- ✓ 세 가지 집합 연산자: 합집합, 교집합, 차집합 연산자
- ✓ 집합 연산자의 입력으로 사용되는 두 개의 릴레이션은 합집합 호환(union compatible)이어야 함
- ✓ 이항 연산자

❑ 합집합 호환

- ✓ 두 릴레이션 R1(A1, A2, ..., An)과 R2(B1, B2, ..., Bm)이 합집합 호환일 필요 충분 조건은 n=m이고, 모든 $1 \le i \le n$에 대해 domain(Ai)=domain(Bi)
  (차수가 동일해야 한다)
- ✓ 결론 : 두 릴레이션의 애트리뷰트 수가 같고, 대응되는 애트리뷰트의 도메인이 같다는 것

4.1 관계 대수(계속)

예 : 합집합 호환

아래의 EMPLOYEE 릴레이션 스키마와 DEPARTMENT 릴레이션 스키마는 애트리뷰트 수가 다르므로 합집합 호환이 되지 않는다.

EMPLOYEE (EMPNO, EMPNAME, TITLE, MANAGER, SALARY, DNO)

DEPARTMENT (DEPTNO, DEPTNAME, FLOOR)

그러나 EMPLOYEE 릴레이션에서 DNO를 프로젝션한 결과 릴레이션 (π_{DNO} (EMPLOYEE)) 과 DEPARTMENT 릴레이션에서 DEPTNO를 프로젝션한 결과 릴레이션 (π_{DEPTNO} (DEPARTMENT)) 은 애트리뷰트 수가 같으며 DNO와 DEPTNO의 도메인이 같으므로 합집합 호환이다.

합치면 차수가 똑같이 1이고, 도메인
이 같으므로 합집합 조건을 만족한다.
R U S

4.1 관계 대수(계속)

❑ 합집합 연산자

- ✓ 두 릴레이션 R과 S의 합집합 R ∪ S는 R 또는 S에 있거나 R과 S 모두에 속한 투플들로 이루어진 릴레이션
- ✓ 결과 릴레이션에서 중복된 투플들은 제외됨
- ✓ 결과 릴레이션의 차수는 R 또는 S의 차수와 같으며, 결과 릴레이션의 애트리뷰트 이름들은 R의 애트리뷰트들의 이름과 같거나 S의 애트리뷰트들의 이름과 같음

4.1 관계 대수(계속)

예 : 합집합

질의: 김창섭이 속한 부서이거나 개발 부서의 부서번호를 검색하라.

EMPLOYEE

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

$$RESULT1 \leftarrow \pi_{DNO}(\sigma_{EMPNAME='김창섭'}(EMPLOYEE))$$

RESULT1

DNO
2

4.1 관계 대수(계속)

DEPARTMENT

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

$RESULT2 \leftarrow \pi_{DEPTNO}(\sigma_{DEPTNAME='개발'}(DEPARTMENT))$

RESULT2

DEPTNO
3

$RESULT3 \leftarrow RESULT1 \cup RESULT2$

RESULT3

DEPTNO
2
3

4.1 관계 대수(계속)

❑ 교집합 연산자

- ✓ 두 릴레이션 R과 S의 교집합 R ∩ S는 R과 S 모두에 속한 투플들로 이루어진 릴레이션
- ✓ 결과 릴레이션의 차수는 R 또는 S의 차수와 같으며, 결과 릴레이션의 애트리뷰트 이름들은 R의 애트리뷰트들의 이름과 같거나 S의 애트리뷰트들의 이름과 같음

4.1 관계 대수(계속)

예 : 교집합

질의: 김창섭 또는 최종철이 속한 부서이면서 기획 부서의 부서번호를 검색하라.

EMPLOYEE

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

$$RESULT1 \leftarrow \pi_{DNO}(\sigma_{EMPNAME='김창섭' \text{ OR } EMPNAME='최종철'}(EMPLOYEE))$$

RESULT1

DNO
2
3

4.1 관계 대수(계속)

DEPARTMENT

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

$RESULT2 \leftarrow \pi_{DEPTNO}(\sigma_{DEPTNAME='기획'}(DEPARTMENT))$

RESULT2

DEPTNO
2

$RESULT3 \leftarrow RESULT1 \cap RESULT2$

RESULT3

DEPTNO
2

4.1 관계 대수(계속)

❑ 차집합 연산자

- ✓ 두 릴레이션 R과 S의 차집합 R – S는 R에는 속하지만 S에는 속하지 않은 투플들로 이루어진 릴레이션
- ✓ 결과 릴레이션의 차수는 R 또는 S의 차수와 같으며, 결과 릴레이션의 애트리뷰트 이름들은 R의 애트리뷰트들의 이름과 같거나 S의 애트리뷰트들의 이름과 같음

4.1 관계 대수(계속)

예 : 차집합

질의: 소속된 직원이 한 명도 없는 부서의 부서번호를 검색하라.

DEPARTMENT

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

$$RESULT1 \leftarrow \pi_{DEPTNO}(DEPARTMENT)$$

RESULT1

DEPTNO
1
2
3
4

4.1 관계 대수(계속)

EMPLOYEE

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

$RESULT2 \leftarrow \pi_{DNO}(EMPLOYEE)$

RESULT2

DNO
2
1
3

RESULT3 ← RESULT1 - RESULT2

RESULT3

DNO
4

4.1 관계 대수(계속)

❑ 카티션 곱 연산자

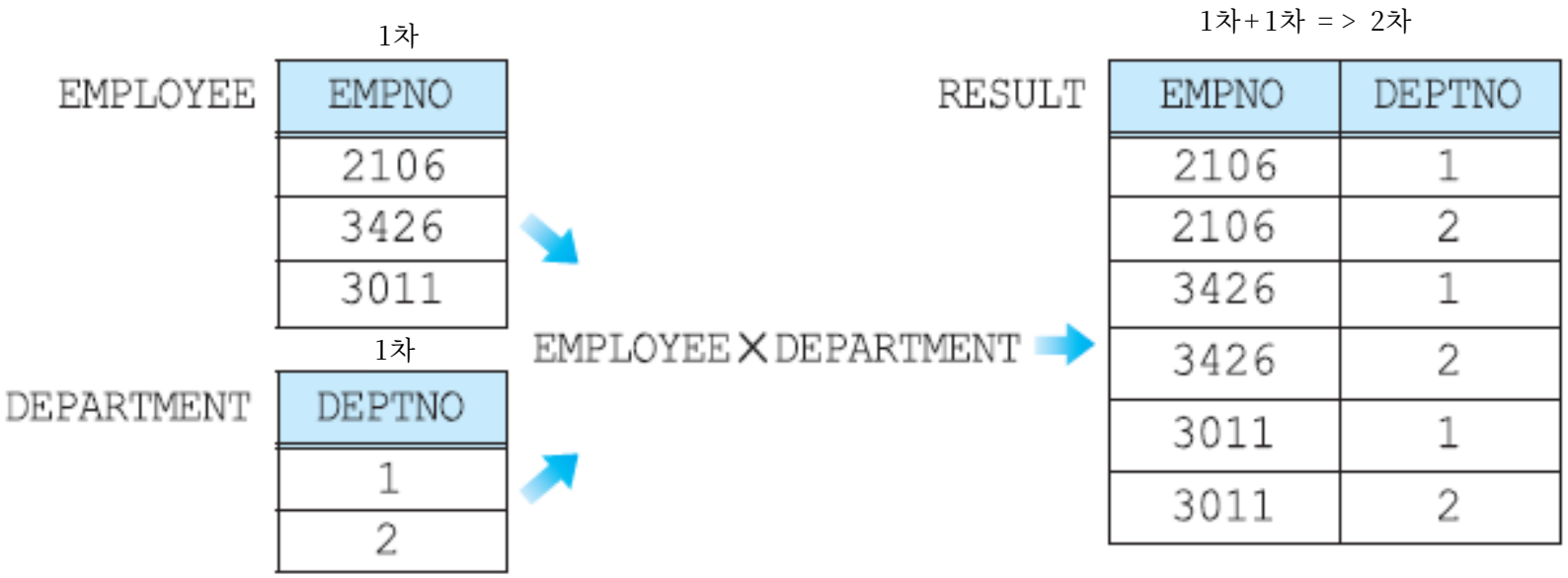
✓ 카디날리티가 i인 릴레이션 R(A1, A2, ..., An)과 카디날리티가 j인 릴레이션 S(B1, B2, ..., Bm)의 카티션 곱 R × S는 차수가 n+m이고, 카디날리티가 i*j이고, 애트리뷰트가 (A1, A2, ..., An, B1, B2, ..., Bm)이며, R과 S의 투플들의 모든 가능한 조합으로 이루어진 릴레이션

✓ 카티션 곱의 결과 릴레이션의 크기가 매우 클 수 있으며, 사용자가 실제로 원하는 것은 카티션 곱의 결과 릴레이션의 일부인 경우가 대부분이므로 카티션 곱 자체는 유용한 연산자가 아님

4.1 관계 대수(계속)

예 : 카티션 곱

질의: EMPLOYEE 릴레이션과 DEPARTMENT 릴레이션의 카티션 곱을 구하라.



4.1 관계 대수(계속)

❑ 관계 대수의 완전성

✓ 실렉션, 프로젝션, 합집합, 차집합, 카티션 곱은 관계 대수의 필수적인 연산자

✓ 다른 관계 연산자들은 필수적인 관계 연산자를 두 개 이상 조합하여 표현할 수 있음

예: 조인 연산자: 카티션곱과 실렉션, 디비전 연산자: 프로젝션, 카티션곱, 차집합

✓ 임의의 질의어가 적어도 필수적인 관계 대수 연산자들만큼의 표현력을 갖고 있으면 관계적으로 완전(relationally complete)하다고 말함

4.1 관계 대수(계속)

❑ 조인 연산자

✓ 두 개의 릴레이션으로부터 연관된 투플들을 결합하는 연산자

✓ 관계 데이터베이스에서 두 개 이상의 릴레이션들의 관계를 다루는데 매우 중요한 연산자

-결과만 보면 조인 연산자는 카티션곱을 수행 후 실렉션 적용과 같다.

-하지만 관계 데이터베이스에서는 조인 연산자의 결과를 구하기 위하여

-먼저 카티션곱을 수행하는 경우는 없다. WHY? 시간이 오래 소요 됨.

✓ 조인을 효율적으로 수행하기 위한 여러가지 알고리즘이 개발되었다.

✓ 세타 조인(theta join), 동등 조인(equal join), 자연 조인(natural join), 외부 조인(outer join), 세미 조인(semi join) 등

4.1 관계 대수(계속)

❑ 세타 조인과 동등 조인

- ✓ 두 릴레이션 R(A1, A2, ..., An)과 S(B1, B2, ..., Bm)의 세타 조인의 결과는 차수가 n+m이고, 애트리뷰트가 (A1, A2, ..., An, B1, B2, ..., Bm)이며, 조인 조건을 만족하는 투플들로 이루어진 릴레이션

- ✓ 세타는 {=, <>, <=, <, >=, >} 중의 하나

- ✓ 동등 조인은 세타 조인 중에서 비교 연산자가 =인 조인

- ✓ 동등조인 형식 : $R \bowtie_{R.attribute=S.attribute} S$

4.1 관계 대수(계속)

❑ 조인 연산 결과

✓카티션 곱 수행 후 실렉션과 같다.

EMPLOYEE

EMPNO	EMPNAME	DNO
2106	김창섭	2
3426	박영권	1
3011	이수민	3
1003	조민희	2
3427	최종철	3

DEPARTMENT

DEPTNO	DEPTNAME
1	영업
2	기획
3	개발
4	총무

EMPLOYEE x DEPARTMENT

EMPNO	EMPNAME	DNO	DEPTNO	DEPTNAME
2106	김창섭	2	1	영업
2106	김창섭	2	2	기획
2106	김창섭	2	3	개발
2106	김창섭	2	4	총무
3427	최종철	1	1	영업
3427	최종철	1	2	기획
3427	최종철	1	3	개발
3427	최종철	1	4	총무
3011	이수민	3	1	영업
3011	이수민	3	2	기획
3011	이수민	3	3	개발
3011	이수민	3	4	총무
1003	조민희	2	1	영업
1003	조민희	2	2	기획
1003	조민희	2	3	개발
1003	조민희	2	4	총무
3427	최종철	3	1	영업
3427	최종철	3	2	기획
3427	최종철	3	3	개발
3427	최종철	3	4	총무

(EMPLOYEE) $\sigma_{DNO=DEPTNO}$ (DEPARTMENT)

EMPNO	EMPNAME	DNO	DEPTNO	DEPTNAME
2106	김창섭	2	2	기획
3427	최종철	1	1	영업
3011	이수민	3	3	개발
1003	조민희	2	2	기획
3427	최종철	3	3	개발

4.1 관계 대수(계속)

예 : 동등 조인

질의: EMPLOYEE 릴레이션과 DEPARTMENT 릴레이션을 동등 조인하라.

EMPLOYEE

EMPNO	EMPNAME	DNO
2106	김창섭	2
3426	박영권	1
3011	이수민	3
1003	조민희	2
3427	최종철	3

DEPARTMENT

DEPTNO	DEPTNAME
1	영업
2	기획
3	개발
4	총무

EMPLOYEE $\bowtie_{DNO=DEPTNO}$ DEPARTMENT

RESULT

EMPNO	EMPNAME	DNO	DEPTNO	DEPTNAME
2106	김창섭	2	2	기획
3426	박영권	1	1	영업
3011	이수민	3	3	개발
1003	조민희	2	2	기획
3427	최종철	3	3	개발

4.1 관계 대수(계속)

❑ 자연 조인

- ✓ 동등 조인에서는 두 릴레이션에서 조인조건에 사용된 두 애트리뷰트가 결과 릴레이션에 포함 된다. 그러나 이 두 애트리뷰트는 이름이 다를수 있어도 결과 릴레이션의 각 투플에서 두 애트리뷰트의 값이 같으므로 둘 중 한 애트리뷰트만 결과 릴레이션에 포함시켜도 무방하다.

- ✓ 동등 조인의 결과 릴레이션에서 조인 애트리뷰트를 한 개 제외한 조인
- ✓ 여러 가지 조인 연산자들 중에서 가장 자주 사용됨
- ✓ 실제로 관계 데이터베이스에서 대부분의 질의는 실렉션, 프로젝션, 자연 조인으로 표현 가능
- ✓ 자연 조인 형식 : R * $_{R.attribute,\ S.attribute}$ S (, 또는)

4.1 관계 대수(계속)

예 : 자연 조인

질의: EMPLOYEE 릴레이션과 DEPARTMENT 릴레이션을 자연 조인하라.

EMPLOYEE

EMPNO	EMPNAME	DNO
2106	김창섭	2
3426	박영권	1
3011	이수민	3
1003	조민희	2
3427	최종철	3

DEPARTMENT

DEPTNO	DEPTNAME
1	영업
2	기획
3	개발
4	총무

EMPLOYEE $*_{DNO,DEPTNO}$ DEPARTMENT

RESULT

EMPNO	EMPNAME	DNO	DEPTNAME
2106	김창섭	2	기획
3426	박영권	1	영업
3011	이수민	3	개발
1003	조민희	2	기획
3427	최종철	3	개발

4.1 관계 대수(계속)

❑ 디비전 연산자

✓ 차수가 n+m인 릴레이션 R(A1, A2, ..., An, B1, B2, ..., Bm)과 차수가 m인 릴레이션 S(B1, B2, ..., Bm)의 디비전 R ÷ S는 차수가 n이고, S에 속하는 모든 투플 u에 대하여 투플 tu(투플 t와 투플 u을 결합한 것)가 R에 존재하는 투플 t들의 집합

4.1 관계 대수(계속)

예 : 디비전

AB

A#	B#
a1	b1
a1	b2
a1	b4
a1	b5
a2	b2
a2	b4
a2	b6
a3	b3
a4	b1
a4	b2
a4	b3

÷ C

B#
b1

→ RESULT1

A#
a1
a4

÷ C

B#
b2
b4

→ RESULT2

A#
a1
a2

÷ C

B#
b1
b2
b3

→ RESULT3

A#
a4

4.1 관계 대수(계속)

❑ 관계 대수 질의의 예

예 : 실렉션, 프로젝션

질의: 2번 부서나 3번 부서에 근무하는 모든 사원들의 이름과 급여를 검색하라.

EMPLOYEE (EMPNO, EMPNAME, TITLE, MANAGER, SALARY, DNO)

DEPARTMENT (DEPTNO, DEPTNAME, FLOOR)

$$\pi_{EMPNAME,\ SALARY}(\sigma_{DNO=2\ OR\ DNO=3}(EMPLOYEE))$$

예 : 실렉션, 프로젝션, 조인

질의: 개발 부서에서 근무하는 모든 사원들의 이름을 검색하라.

$$\pi_{EMPNAME}(EMPLOYEE \bowtie_{DNO=DEPTNO} (\sigma_{DEPTNAME='개발'}(DEPARTMENT)))$$

4.1 관계 대수(계속)

❑ 관계 대수의 한계

- ✓ 관계 대수 연산자는 한 개 이상의 릴레이션을 입력받아 새로운 결과 릴레이션을 한 개 생성하지만 다음과 같은 몇가지 제한사항이 있다.
- ✓ 관계 대수 연산자로 데이터베이스 질의들을 표현하는데 충분치 않다.
- ✓ 이 문제를 관계 DBMS의 표준 질의어인 SQL이 요구사항을 지원한다.

- ✓ 관계 대수는 산술 연산을 할 수 없음(SALARY 값을 10% 인상 계산X)
- ✓ 집단 함수(aggregate function)를 지원하지 않음(SUM, AVG, COUNT,..)
- ✓ 정렬을 나타낼 수 없음(모든 투플을 사원의 이름 순으로 정렬) 어센딩, 디센딩 등
- ✓ 데이터베이스를 수정할 수 없음(SALARY값을 10% 인상 수정)
- ✓ 프로젝션 연산의 결과에 중복된 투플을 나타내는 것이 필요할 때가 있는데 이를 명시하지 못함

4.1 관계 대수(계속)

❑ 추가된 관계 대수 연산자

✓ 집단 함수

예 : 집단 함수

질의: 모든 사원들의 급여의 평균이 얼마인가?

EMPLOYEE

EMPNO	···	SALARY	···
2106	···	2500000	···
3426	···	3000000	···
3011	···	4000000	···
1003	···	3000000	···
3427	···	1500000	···
1365	···	1500000	···
4377	···	5000000	···

$AVG_{SALARY}(EMPLOYEE)$ → 2,928,571

4.1 관계 대수(계속)

❑ 추가된 관계 대수 연산자(계속)

✓ 그룹화

예 : 그룹화

질의: 각 부서별 사원들의 급여의 평균이 얼마인가?

EMPLOYEE

EMPNO	···	SALARY	DNO
3426	···	3000000	1
1365	···	1500000	1
2106	···	2500000	2
1003	···	3000000	2
4377	···	5000000	2
3011	···	4000000	3
3427	···	1500000	3

RESULT

DNO	AVG(SALARY)
1	2250000
2	3500000
3	2750000

$_{DNO}\mathcal{G}_{AVG(SALARY)}(EMPLOYEE)$

(Group)

4.1 관계 대수(계속)

❑ 추가된 관계 대수 연산자(계속)

✓ 외부 조인

- 상대 릴레이션에서 대응되는 투플을 갖지 못하는 투플이나 조인 애트리뷰트에 널값이 들어 있는 투플들을 다루기 위해서 조인 연산을 확장한 조인
- 두 릴레이션에서 대응되는 투플들을 결합하면서, 대응되는 투플을 갖지 않는 투플과 조인 애트리뷰트에 널값을 갖는 투플도 결과에 포함시킴
- 왼쪽 외부 조인(left outer join), 오른쪽 외부 조인(right outer join), 완전 외부 조인(full outer join)

4.1 관계 대수(계속)

❑ 왼쪽 외부 조인

✓ 릴레이션 R과 S의 왼쪽 외부 조인 연산은 왼쪽 R의 모든 투플들을 결과에 포함시키고, 만일 릴레이션 S에 관련된 투플이 없으면 결과 릴레이션에서 릴레이션 S의 애트리뷰트들은 널값으로 채움

4.1 관계 대수(계속)

❑ 오른쪽 외부 조인

✓ 릴레이션 R와 S의 오른쪽 외부 조인 연산은 S의 모든 투플들을 결과에 포함시키고, 만일 릴레이션 R에 관련된 투플이 없으면 결과 릴레이션에서 릴레이션 R의 애트리뷰트들은 널값으로 채움

예 : R과 S의 오른쪽 외부 조인

R

A	B	C
a1	b1	c1
a2	b2	c2

R ⟖ S

S

C	D	E
c1	d1	e1
c3	d2	e2

→ RESULT

A	B	C	D	E
a1	b1	c1	d1	e1
∧	∧	c3	d2	e2

4.1 관계 대수(계속)

예 : 자연 조인과 왼쪽 외부 조인

R과 S의 자연 조인

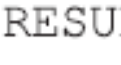
R

A	B	C
a1	b1	c1
a2	b2	c2

R*S

S

C	D	E
c1	d1	e1
c3	d2	e2



RESULT

A	B	C	D	E
a1	b1	c1	d1	e1

EMPLOYEE

R과 S의 왼쪽 외부 조인

R

A	B	C
a1	b1	c1
a2	b2	c2

R ⟕ S

S

C	D	E
c1	d1	e1
c3	d2	e2

RESULT

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2	∧	∧

4.1 관계 대수(계속)

❑ 완전 외부 조인

✓ 릴레이션 R와 S의 완전 외부 조인 연산은 R과 S의 모든 투플들을 결과에 포함시키고, 만일 상대 릴레이션에 관련된 투플이 없으면 결과 릴레이션에서 상대 릴레이션의 애트리뷰트들은 널값으로 채움

예 : 완전 외부 조인

R

A	B	C
a1	b1	c1
a2	b2	c2

R ⟗ S

S

C	D	E
c1	d1	e1
c3	d2	e2

→ RESULT

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2	∧	∧
∧	∧	c3	d2	e2

4.2 SQL 개요

❑ SQL 개요

+ DCL, TCL(Trasaction Control Language)

- ✓ 자연어의 모호성과 정확한 질의를 위해 구조적 질의어 필요
- ✓ 데이터베이스와 릴레이션의 구조 정의(DDL), 릴레이션에 투플 삽입, 삭제, 수정(DML), 질의
- ✓ SQL은 현재 DBMS 시장에서 관계 DBMS가 압도적인 우위를 차지하는데 중요한 요인의 하나
- ✓ SQL은 IBM 연구소에서 1974년에 System R이라는 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 SQL이 널리 사용되는데 기여
- ✓ 다양한 상용 관계 DBMS마다 지원하는 SQL 기능에 다소 차이가 있음
- ✓ 본 책에서는 SQL2를 따름

4.2 SQL 개요(계속)

❑SQL 개요

✓관계 데이터베이스 표준 질의어를 사용 이점 : 직원에 대한 교육비용이 절감되고, 생산성이 높아지며, 응용 프로그램의 이식성이 향상되고, 특정 DBMS에 대한 의존도가 감소하며, DBMS간의 통신이 원활해 진다.

〈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, 멀티미디어 기능 등이 추가됨. 약 2000페이지 분량

현재 사용 중 >

4.2 SQL 개요(계속)

❑ SQL 개요(계속)

✓ SQL의 장점 : 자연어에 가까운 구문을 사용하여 질의 표현 할 수 있다.

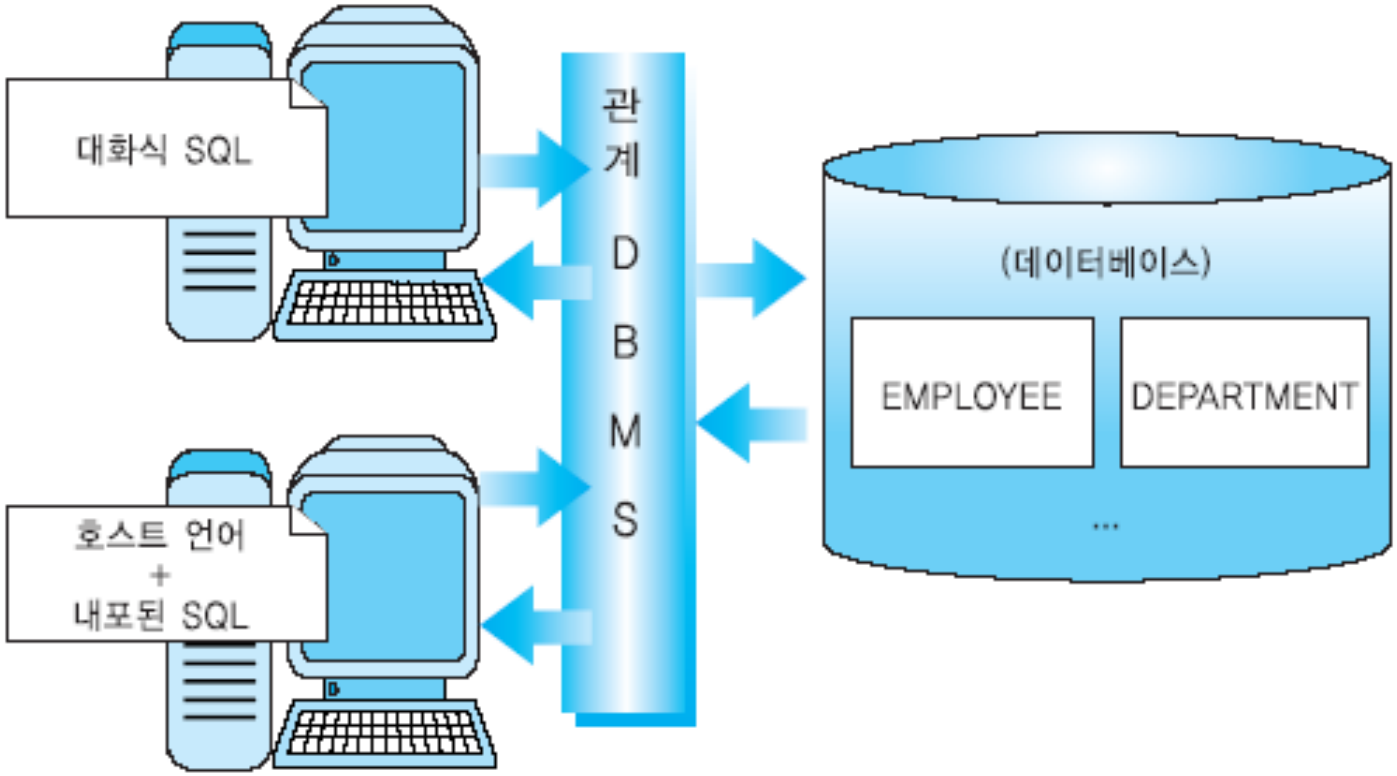
✓ SQL은 비절차적 언어(선언적 언어)이므로 사용자는 자신이 원하는 바(what)만 명시하며, 원하는 것을 처리하는 방법(how)은 명시할 수 없음

✓ 관계 DBMS는 사용자가 입력한 SQL문을 번역하여 사용자가 요구한 데이터를 찾는데 필요한 모든 과정을 담당(프로그래밍 노력이 적게 소요)

✓ 두 가지 인터페이스

- 대화식 SQL(interactive SQL)-데이터베이스에 접근하는 최종 사용자
- 내포된 SQL(embedded SQL)-고급언어 내에 SQL을 포함하는 방식.응용 프로그래머가 전문적인 데이터베이스 응용 프로그램 개발 시 사용.

4.2 SQL 개요(계속)



[그림 4.3] 관계 데이터베이스에 대한 두 가지 인터페이스

4.2 SQL 개요(계속)

❑ 오라클 SQL의 구성요소

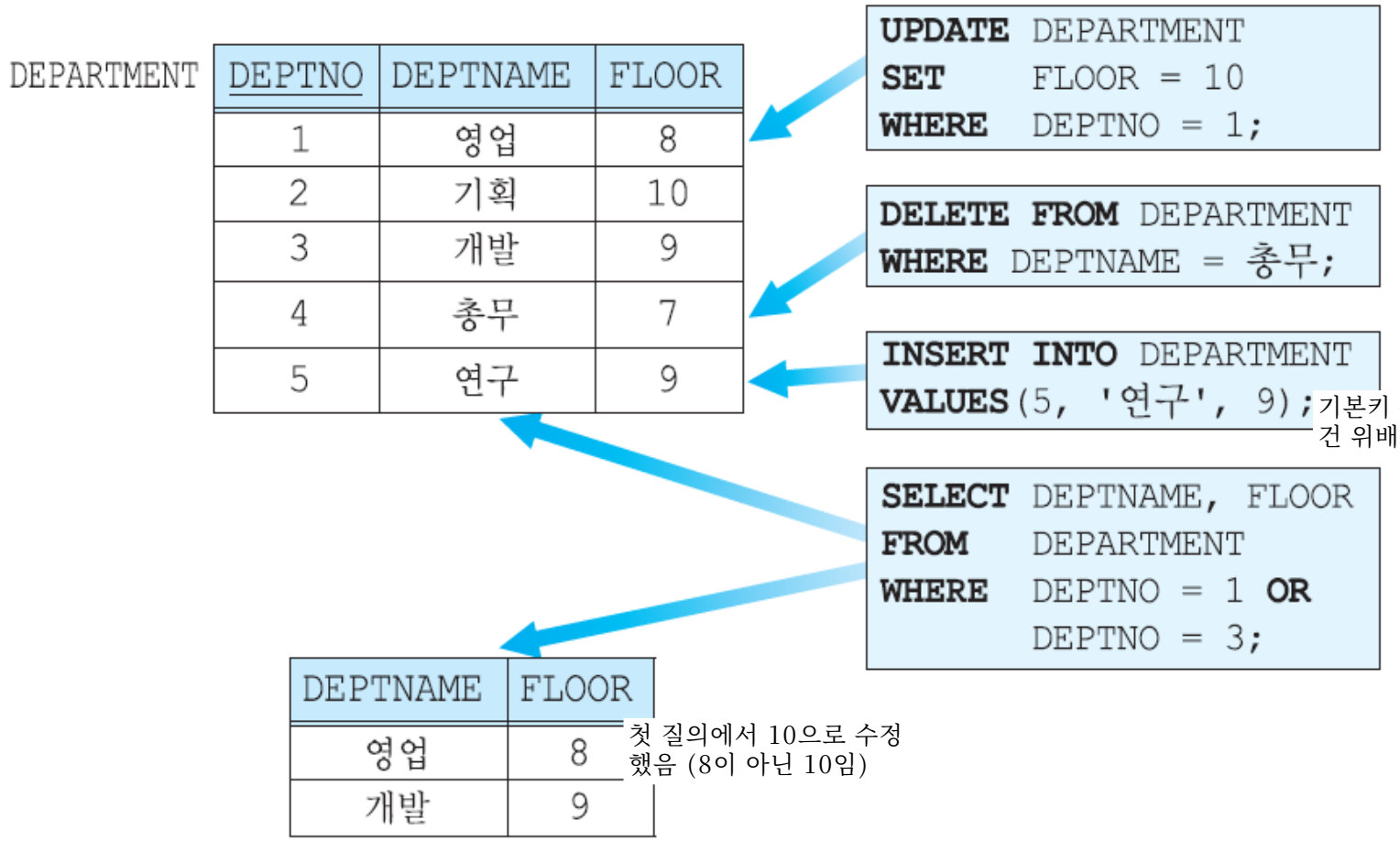
- ✓ 데이터 검색
- ✓ 데이터 조작어(DML : Data Manipulation Language)
- ✓ 데이터 정의어(DDL : Data Definition Language)
- ✓ 트랜잭션 제어(TCL : Transaction Control Language)
- ✓ 데이터 제어어(DCL : Data Control Language)

4.2 SQL 개요(계속)

❑ 오라클 SQL 문장의 종류

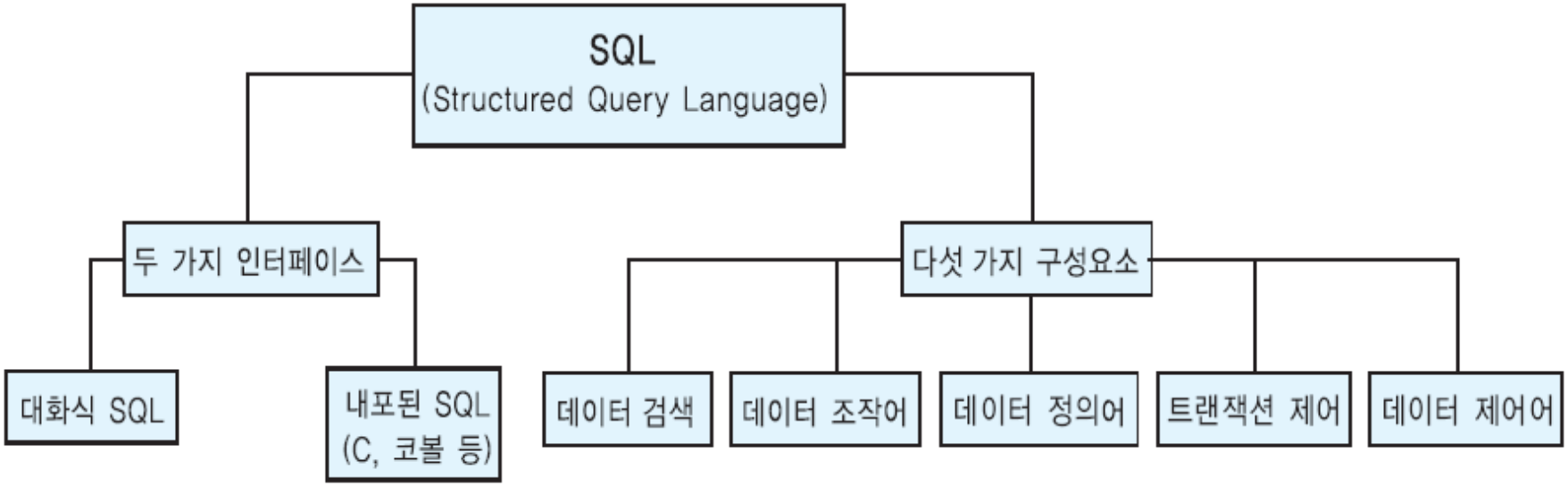
명령어 종류	명령어	설명
데이터 조작어 (DML : Data Manipulation Language	SELECT	데이터베이스에 들어 있는 데이터를 조회하거나 검색하기 위한 명령어를 말하는 것으로 RETRIEVE 라고도 한다.
	INSERT UPDATE DELETE	데이터베이스의 테이블에 들어 있는 데이터에 변형을 가하는 종류의 명령어들을 말한다. 데이터 삽입, 수정, 삭제
데이터 정의어 (DDL : Data Definition Language)	CREATE ALTER DROP RENAME TRUNCATE	테이블과 같은 데이터 구조를 정의하는데 사용되는 명령어들로 그러한 구조를 생성하거나 변경하거나 삭제하거나 이름을 바꾸는 데이터 구조와 관련된 명령어들을 DDL이라고 부른다.
데이터 제어어 (DCL : Data Control Language)	GRANT REVOKE	데이터베이스에 접근하고 객체들을 사용하도록 권한을 주고 회수하는 명령어를 DCL이라고 한다.
트랜잭션 제어어 (TCL : Transaction Control Language)	COMMIT ROLLBACK SAVEPOINT	논리적인 작업의 단위를 묶어서 DML에 의해 조작된 결과를 작업단위(트랜잭션) 별로 제어하는 명령어를 말한다.

4.2 SQL 개요(계속)



[그림 4.4] 데이터 검색과 데이터 조작어의 기능

4.2 SQL 개요(계속)



[그림 4.5] SQL의 인터페이스와 구성요소

4.3 데이터 정의어와 무결성 제약조건

〈표 4.4〉 데이터 정의어의 종류

CREATE	DOMAIN	도메인을 생성
	TABLE	테이블을 생성
	VIEW	뷰를 생성
	INDEX	인덱스를 생성. SQL2의 표준이 아님
ALTER	TABLE	테이블의 구조를 변경
DROP	DOMAIN	도메인을 제거
	TABLE	테이블을 제거
	VIEW	뷰를 제거
	INDEX	인덱스를 제거. SQL2의 표준이 아님

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 데이터 정의어

✓ 스키마의 생성과 제거

- SQL2에서는 동일한 데이터베이스 응용에 속하는 릴레이션, 도메인, 제약조건, 뷰, 권한 등을 그룹화하기 위해서 스키마 개념을 지원
- 각 사용자는 권한을 허가받지 않은 한 다른 사용자의 스키마에 속한 데이터를 볼 수 없다.
- Kim이라는 계정을 가진 사용자가 MY_DB라는 스키마를 생성 및 삭제
- 스키마 삭제 시 비어 있지 않으면, DBMS가 거절한다.

```
CREATE SCHEMA MY_DB AUTHORIZATION kim;

DROP SCHEMA MY_DB;
```

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 릴레이션 정의

```
CREATE TABLE EMPLOYEE
       (EMPNO      NUMBER    NOT NULL,
        EMPNAME    CHAR(10),
        TITLE      CHAR(10),
        MANAGER    NUMBER,
        SALARY     NUMBER,
        DNO        NUMBER,
        PRIMARY KEY(EMPNO),
        FOREIGN KEY(MANAGER) REFERENCES EMPLOYEE(EMPNO),
        FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DEPTNO));
```

[그림 4.6] DEPARTMENT 릴레이션과 EMPLOYEE 릴레이션의 생성

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 릴레이션 정의

```
CREATE TABLE DEPARTMENT
        (DEPTNO        NUMBER        NOT  NULL,
         DEPTNAME      CHAR(10),
         FLOOR         NUMBER,
         PRIMARY KEY(DEPTNO));
```

❑ 릴레이션 정의 할 때 EMPLOYEE 릴레이션에서 DEPARTMENT 릴레이션의 기본 키를 외래 키로 참조하기 때문에 EMPLOYEE 릴레이션을 먼저 정의하면 데이터베이스에 없는 DEPARTMENT 릴레이션의 기본 키를 참조하는 현상이 발생되어 DEPARTMENT 릴레이션 정의 후 EMPLOYEE 릴레이션을 정의 한다. (참조무결성 제약조건 위배)

4.3 데이터 정의어와 무결성 제약조건(계속)

〈표 4.5〉 릴레이션의 정의에 사용되는 오라클의 데이터 타입

데이터 타입	의미
INTEGER 또는 INT	정수형
NUMBER(n, s)	소수점을 포함한 n개의 숫자에서 소수 아래 숫자가 s개인 십진수
CHAR(n) 또는 CHARACTER(n)	n바이트 문자열. n을 생략하면 1
VARCHAR(n), VARCHAR2(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. 날짜와 시간을 저장
BINARY_FLOAT	오라클 10g부터 도입되었는데, 32비트에 실수를 저장
BINARY_DOUBLE	오라클 10g부터 도입되었는데, 64비트에 실수를 저장
BLOB	Binary Large OBject. 멀티미디어 데이터 등을 저장

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 릴레이션 제거

```
DROP TABLE DEPARTMENT;
```

예 : 릴레이션의 정의와 투플들이 모두 제거, 다른 릴레이션이나 뷰에서 참조되지 않는 릴레이션들만 제거된다.

DEPARTMENT 릴레이션을 제거하려면 **EMPLOYEE** 릴레이션을 정의 할 때 **DEPARTMENT** 릴레이션의 기본 키인 **DEPTNO**를 외래 키로 참조하도록 정의해서 **DROP TABLE DEPARTMENT** 문은 수행되지 않는다.

해결책 : **EMPLOYEE** 릴레이션에서 **DEPARTMENT** 릴레이션을 참조하는 외래 키정의를 **ALTER TABLE**문으로 먼저 제거 후 삭제해야 한다.

4.3 데이터 정의어와 무결성 제약조건(계속)

- ALTER TABLE
  - ✓ 릴레이션을 만든 후 투플들을 삽입하고 계속 사용하다가 응용의 요구사항이 변하여 기존 릴레이션에 애트리뷰트를 추가하거나 삭제 할 때
  - ✓ 추가된 애트리뷰트는 릴레이션의 마지막 애트리뷰트
  - ✓ 추가된 애트리뷰트에는 디폴트 값을 명시하지 않는 경우에는 NOT NULL을 지정할 수 없다(추가된 애트리뷰트에 대해 기존의 투플들은 널값을 갖기때문)
  - ✓ 새로 추가된 애트리뷰트에 대해 사용자가 일일이 update문으로 값을 입력

  ✓ `ALTER TABLE EMPLOYEE ADD PHONE CHAR(13);`

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 인덱스 생성 및 삭제

- ✓ CREATE INDEX문은 하나 이상의 애트리뷰트에 대해 인덱스를 생성
- ✓ 뷰(VIEW)는 인덱스를 생성할 수 없다.뷰는 뷰 정의에 사용된 기본 릴레이션에 생성된 인덱스를 사용
- ✓ 인덱스는 검색 성능을 향상 시키기 위해 사용
- ✓ 인덱스는 저장공간을 추가로 필요하고 갱신연산 속도를 저하 시킨다.
- ✓ 인덱스를 생성하면 디폴트는 오름차순
- ✓ 인덱스가 정의된 릴레이션의 투플의 변경이 생기면 DBMA가 자동적으로 인덱스에 반영

- ✓ **CREATE INDEX** EMPDNO_IDX **ON** EMPLOYEE(DNO);
- ✓ **DROP INDEX EMPLOYEE.EMPNO_IDX;**

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 애트리뷰트의 제약조건

```
CREATE TABLE EMPLOYEE
   (EMPNO    NUMBER    NOT NULL,                                    (1)
    EMPNAME  CHAR(10)  UNIQUE,                                      (2)
    TITLE    CHAR(10)  DEFAULT '사원',                              (3)
    MANAGER  NUMBER,
    SALARY   NUMBER    CHECK (SALARY < 6000000),                    (4)
    DNO      NUMBER    CHECK (DNO IN (1,2,3,4,5,6)) DEFAULT 1,      (5)
    PRIMARY KEY(EMPNO),                                             (6)
    FOREIGN KEY(MANAGER) REFERENCES EMPLOYEE(EMPNO),                (7)
    FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DEPTNO)                  (8)
            ON DELETE CASCADE);                                     (9)
```

[그림 4.7] 릴레이션 정의에서 다양한 제약조건을 명시

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 애트리뷰트의 제약조건

(1) NOT NULL – 애트리뷰트는 디폴트로 널값을 가질 수 있기 때문에 만일 어떤 애트리뷰트에 널 값을 허용하지 않으려면 'NOT NULL'명시해야 한다.

- 이런 애트리뷰트에 대해 디폴트 값을 지정하지 않으면 INSERT문에서 이 애트리뷰트에 반드시 값을 입력해야 하며, 그렇치 않으면 제약조건을 위배하는 연산이므로 거절된다.

- 기본 키는 이 제약조건을 명시해야 한다.

- 투플을 삽입하는 시점에 어떤 애트리뷰트의 값을 알지 못하는 경우에는 그 애트리뷰트를 'NOT NULL'로 명시해서는 안된다.

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 애트리뷰트의 제약조건

(2) UNIQUE – 동일한 애트리뷰트 값을 갖는 투플이 두개 이상 존재하지 않도록 보장

- ‘NOT NULL’을 명시 하지 않았다면 한 개의 투플에서는 이 애트리뷰트에 널값을 가질 수 있다.
- 한 릴에이션에 UNIQUE절을 여러 개 명시 할 수 있다.
- 오라클은 UNIQUE절이 명시된 애트리뷰트에 자동적으로 인덱스를 생성

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 애트리뷰트의 제약조건

(3) DEFAULT – 애트리뷰트에 널값 대신에 특정 값을 지정 할 수 있다.

- 애트리뷰트의 이름과 애트리뷰트의 데이터 타입을 명시한 곳 어디서든지 DEFALULT절 사용하여 디폴트 값을 지정 할 수 있다.

✓ **TITLE CHAR(10) DEFAULT '사원',**

- EMPLOYEE 릴레이션에 투플을 삽입할 때 TITLE 애트리뷰트의 값을 입력하지 않으면 디폴트 값인 '사원'이 입력

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 애트리뷰트의 제약조건

(4)(5) CHECK – 한 애트리뷰트가 가질 수 있는 값들의 범위를 지정.

```
CREATE TABLE EMPLOYEE (
     ID NUMBER,
     NAME CHAR(10),
     SALARY NUMBER CHECK (SALARY < 6000000),,
     DNO  NUMBER CHECK (DNO IN (1,2,3,4,5,6)) DEFAULT 1,
```

- 모든 사원의 급여 최고 액은 6,000,000미만이다.
- 모든 사원이 속한 부서번호 DNO는 1,2,3,4,5,6중 하나의 값만 가지며, 입력하지 않으면 디폴트 부서번호는 1이다.

4.3 데이터 정의어와 무결성 제약조건(계속)

❑애트리뷰트의 제약조건

(4)(5) CHECK – 한 애트리뷰트가 가질 수 있는 값들의 범위를 지정

```
CREATE TABLE EMPLOYEE (
     ID NUMBER,
     NAME CHAR(10),
     SALARY NUMBER,
     MANAGER_SALARY NUMBER,CHECK (MANAGER_SALARY > SALARY));
```

- 서로 다른 애트리뷰트의 값에 대해 CHECK 옵션 사용
- 모든 사원의 급여는 자신의 상사의 급여보다 작아야 한다.

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 기본 키 제약 조건

- 기본 키는 엔티티 무결성 제약조건에 의해 널 값을 갖지 않아야 한다. 릴레이션 마다 최대 한 개의 기본 키 지정, 기본 키에는 자동적으로 인덱스가 생성된다.

```
CREATE TABLE EMPLOYEE
   (EMPNO    NUMBER    NOT NULL,                                    (1)
    EMPNAME  CHAR(10)  UNIQUE,                                      (2)
    TITLE    CHAR(10)  DEFAULT '사원',                              (3)
    MANAGER  NUMBER,
    SALARY   NUMBER    CHECK (SALARY < 6000000),                    (4)
    DNO      NUMBER    CHECK (DNO IN (1,2,3,4,5,6)) DEFAULT 1,      (5)
    PRIMARY KEY(EMPNO),                                             (6)
    FOREIGN KEY(MANAGER) REFERENCES EMPLOYEE(EMPNO),                (7)
    FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DEPTNO)                  (8)
            ON DELETE CASCADE);                                     (9)
```

[그림 4.7] 릴레이션 정의에서 다양한 제약조건을 명시

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 참조 무결성 제약조건 유지

- 참조 무결성 제약조건은 한 릴레이션에 들어 있는 외래 키의 개수 만큼 참조 무결성 제약 조건을 명시할 수 있다. 참조 무결성 제약조건은 외래 키의 무결성을 보장한다.
- EMPLOYEE 릴레이션의 임의의 투플의 MANAGER도 이 회사의 사원이므로, 반드시 EMPLOYEE 릴레이션의 EMPNO에 나타나야 하고, EMPLOYEE 릴레이션에 나타나는 부서번호 DNO는 반드시 DEPARTMENT 릴레이션의 부서번호 DEPTNO에 존재해야 한다.

```
CREATE TABLE EMPLOYEE
   (EMPNO    NUMBER    NOT NULL,                                    (1)
    EMPNAME  CHAR(10)  UNIQUE,                                      (2)
    TITLE    CHAR(10)  DEFAULT '사원',                              (3)
    MANAGER  NUMBER,
    SALARY   NUMBER    CHECK (SALARY < 6000000),                    (4)
    DNO      NUMBER    CHECK (DNO IN (1,2,3,4,5,6)) DEFAULT 1,      (5)
    PRIMARY KEY(EMPNO),                                             (6)
    FOREIGN KEY(MANAGER) REFERENCES EMPLOYEE(EMPNO),                (7)
    FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DEPTNO)                  (8)
            ON DELETE CASCADE);                                     (9)
```

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 참조 무결성 제약조건 유지

● 참조 무결성 제약조건에서 참조되는 애트리뷰트는 참조되는 릴레이션에서 동일한 데이터 타입을 가지면서 UNIQUE 또는 기본 키로 정의 되어 있어야 한다.

●참조 무결성 제약 조건을 보존하기 위해서 데이터베이스가 갱신된 후에 검사를 수행해야 한다.

기사 시험 언급★

●참조되는 릴레이션을 변경하는 삽입, 삭제, 수정 중에서 참조 무결성 제약 조건을 위배할 수 있눈 연산은 삭제와 수정 연산이다.

4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 참조 무결성 제약조건 유지

- **ON DELETE NO ACTION =RESTRICT(제한)**

  참조되는 **DEPARTMENT** 릴레이션에서 어떤 투플을 삭제하려는데, 이 투플의 기본 키의 값을 참조하고 있는 투플이 **EMPLOYEE** 릴레이션에 존재하면 삭제 연산을 거절한다.

- **ON DELETE CASCADE**

  DEPARTMENT 릴레이션에서 기본 키의 값이 삭제 되면, **EMPLOYEE**릴레이션에서도 이 값을 참조하는 모든 투플들이 삭제

- **ON DELETE SET NULL**

  DEPARTMENT 릴레이션에서 기본 키의 값이 삭제 되면, **EMPLOYEE** 릴레이션에서도 이 값을 참조하는 모든 투플들의 부서번호는 널값을 갖는다.

- **ON DELETE SET DEFAULT**

  DEPARTMENT 릴레이션에서 기본 키의 값이 삭제 되면, **EMPLOYEE** 릴레이션에서도 이 값을 참조하는 모든 투플들의 부서번호는 디폴트 값(부서번호 1)을 갖는다.

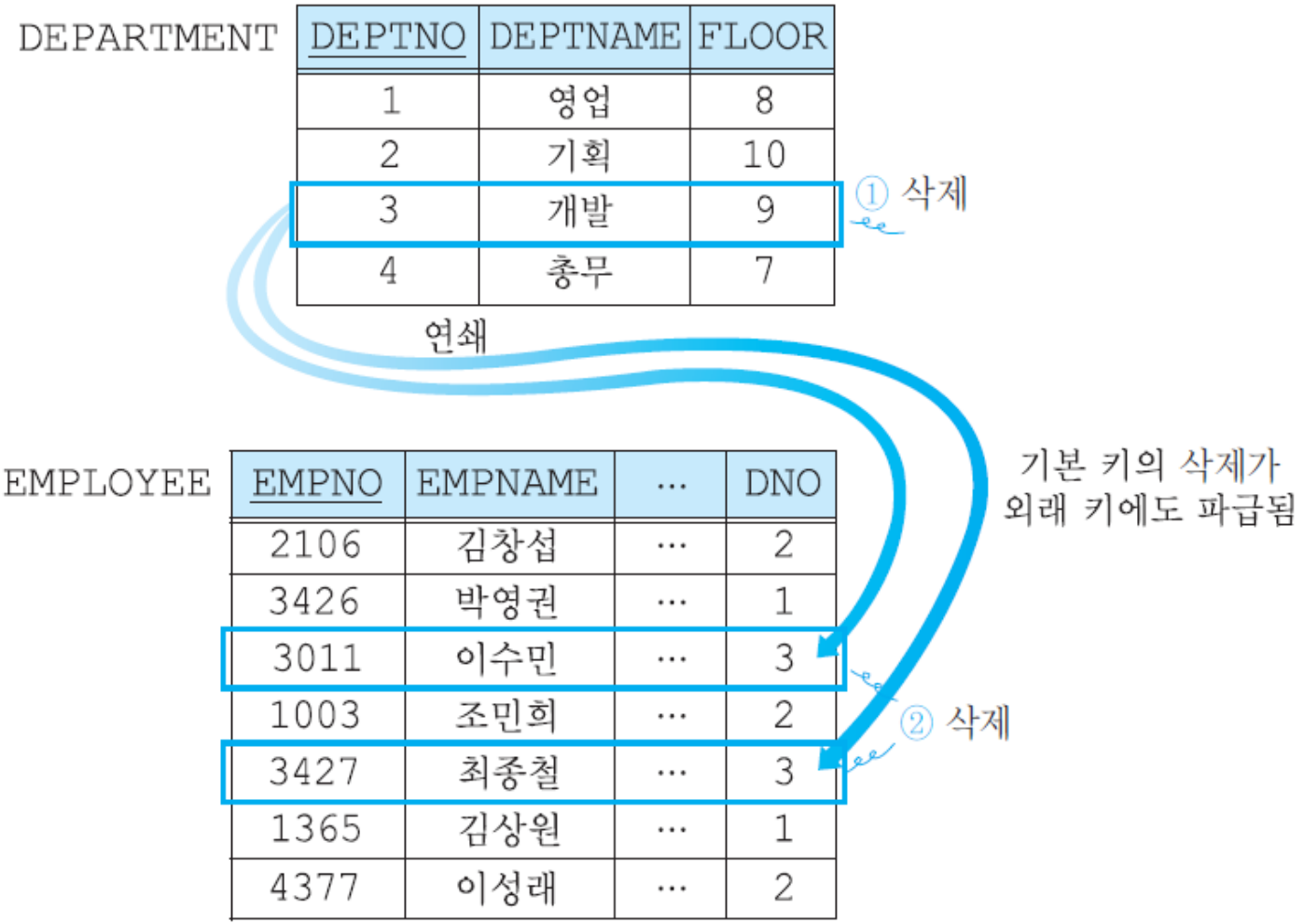
4.3 데이터 정의어와 무결성 제약조건(계속)

예 : ON DELETE CASCADE

4.5절에서 설명할 DELETE문을 사용하여 다음과 같이 DEPARTMENT 릴레이션에서 3번 부서의 투플을 삭제하면, EMPLOYEE 릴레이션에서 3번 부서에 근무하는 모든 사원들의 투플도 자동적으로 삭제된다.

```
DELETE DEPARTMENT
WHERE  DEPTNO = 3;
```

4.3 데이터 정의어와 무결성 제약조건(계속)



4.3 데이터 정의어와 무결성 제약조건(계속)

❑ 무결성 제약조건의 추가 및 삭제

- 릴레이션이 생성된 후에도 기본 키 제약조건을 포함하여 여러가지 제약조건들을 추가 할 수 있다.

- 기본 키 제약조건 이름 : **STUDENT_PK**, 기본 키 : **STNO**

```
ALTER TABLE STUDENT ADD CONSTRAINT STUDENT_PK
    PRIMARY KEY (STNO);
```

- 제약조건이 더 이상 필요하지 않으면 삭제

```
ALTER TABLE STUDENT DROP CONSTRAINT STUDENT_PK;
```

- 기본 키 제약조건이 삭제 되면 연관된 인덱스도 함께 삭제된다. 그러나 외래 키 제약조건이 삭제 되면 연관된 인덱스는 삭제 되지 않는다.

4.4 SELECT문

❑ SELECT문

- ✓ 관계 데이터베이스에서 정보를 검색하는 SQL문

셀렉션은 튜플의 모든 어트리뷰트를 다 보는 것

- ✓ 관계 대수의 실렉션과 의미가 완전히 다름

- ✓ 관계 대수의 실렉션, 프로젝션, 조인, 카티션 곱 등을 결합한 것

- ✓ 관계 데이터베이스에서 가장 자주 사용됨

- ✓ 질의를 작성 시 고려 사항들
  - –어떤 애트리뷰트를 보고자 하는가?
  - –이 애트리뷰트들이 어떤 릴레이션에 속해 있는가?
  - –다수의 릴레이션이 사용될 때는 어떻게 릴레이션들이 조인 되는가?

4.4 SELECT문(계속)

EMPLOYEE

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

DEPARTMENT

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

[그림 4.8] 관계 데이터베이스 상태

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

✓ SELECT절과 FROM절만 필수적인 절이고, 나머지는 선택 사항

```
SELECT      [DISTINCT] 애트리뷰트(들)          (1)  } 필수
FROM        릴레이션(들)                       (2)  }
[WHERE      조건                               (3)  } 선택
                  [중첩 질의]]                 (4)  }
[GROUP BY   애트리뷰트(들)]                    (5)  }
[HAVING     조건]                              (6)  }
[ORDER BY   애트리뷰트(들) [ASC|DESC]];        (7)  }
```

[그림 4.9] SELECT문의 형식

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(1) SELECT절

- 질의 결과에 포함하려는 애트리뷰트들의 리스트를 열거
- 관계대수의 프로젝션 연산에 해당
- 질의결과에서 중복을 제거하려면 먼저 정렬해야하는데 정렬연산은 시간이 많이 소요되므로 DBMS가 자동적으로 질의결과에서 중복을 제거하지 않는다.

또한 사용자가 중복된 투플을 원하거나 집단함수를 정확하게 적용하기 위해서 중복된 투플이 필요한 경우도 있다.

- 사용자가 DISTINCT절을 사용하면 명시적으로 요청이 있을 때만 중복을 제거

(2) FROM절

- 질의에서 필요로하는 릴레이션 리스트 열거

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(3) WHERE절

- 관계대수의 실렉션 연산의 프리디키트(실렉션 조건)에 해당
- FROM절에서 열거한 릴레이션에 속하는 애트리뷰트들만 사용해서 실렉션 조건을 표현해야 한다.
- WHERE절의 조건은 결과 릴레이션의 투플들이 만족시켜야 하는 조건
  예: WHERE SALARY > 1500000 ; 투플들이 급여가 15000000보다 크다.

- WHERE절의 실렉션 조건(프리디키트=predict)는 SELECT문, DELETE문, UPDATE문에 사용된다.

Select ~ From ~ Where (이 부분에서 Select를 또 쓸 수있음)

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(3) WHERE절(프리디키트들)

- 비교연산자

연산자	연산자의 의미
=	같다.
〉	보다 크다.
〉=	보다 크거나 같다.
〈	보다 작다.
〈=	보다 작거나 같다.

- SQL 연산자

연산자	연산자의 의미
BETWEEN a AND b	a와 b의 값 사이에 있으면 된다.(a와 b의 값이 포함됨)
IN (list)	리스트에 있는 값 중에서 어느 하나라도 일치하면 된다.
LIKE '비교문자열'	비교 문자열과 형태가 일치하면 된다.
IS NULL	NULL 값인 경우

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(3) WHERE절

- 와일드 카드

와일드 카드	설명
%	0개 이상의 어떤 문자를 의미한다.
_	1개인 단일 문자를 의미한다.

- 논리 연산자

연산자	연산자의 의미
AND	앞에 있는 조건과 뒤에 오는 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞의 조건과 뒤의 조건을 동시에 만족해야 하는 것이다.
OR	앞의 조건이 참(TRUE)이거나 뒤의 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞뒤의 조건 중 하나만 참(TRUE)이면 된다.
NOT	뒤에 오는 조건에 반대되는 결과를 되돌려 준다.

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(3) WHERE절

- 부정 연산자

종류	연산자	연산자의 의미
부정 논리 연산자	!=	같지 않다.
	^=	같지 않다.
	◇	같지 않다. (ANIS/ISO 표준, 모든 운영체제에서 사용가능)
	NOT 칼럼명 =	~와 같지 않다.
	NOT 칼럼명 〉	~보다 크지 않다.
부정 SQL 연산자	NOT BETWEEN a AND b	a와 b의 값 사이에 있지 않다. (a, b값을 포함하지 않는다)
	NOT IN (list)	list 값과 일치하지 않는다.
	IS NOT NULL	NULL 값을 갖지 않는다.

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

(4) 중첩질의

- WHERE 절에 포함된 SELECT문

(5) GROUP BY절

- 질의 결과를 GROUP BY 다음에 명시된 애트리뷰트에 동일한 값을 갖는 투플들을 한 그룹으로 묶음

(6) HAVING절

- 투플들의 그룹이 만족해야 하는 조건을 나타낸다.

(7) ORDER BY절

- 결과 투플들의 정렬순서를 지정한다.

4.4 SELECT문(계속)

❑ 기본적인 SQL 질의

- SQL문이 수행되는 개념적인 순서

  단계1: 투플들을 구하고(카티션 곱)

  단계2: 조건을 만족하는 !튜플 들을 식별하고

  단계3: 그룹들을 구하고

  단계4: HAVING을 적용하여 일부 그룹을 제거하고

  단계5:집단함수의 값을 구하고

  단계6:결과 투플들을 정렬하여 사용자에게 제시한다.

4.4 SELECT문(계속)

❑ 별칭(alias)

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

```
EMPLOYEE.DNO
```

✓ 투플변수를 사용하는 방법 (투플변수=별칭(alias))

AS는 생략해도 된다

```
FROM EMPLOYEE AS E, DEPARTMENT AS D
```

; EMPLOYEE릴레이션은 E, DEPARTMENT 릴레이션은 D로 사용 가능

; AS 생략가능하며, 투플변수의 편리성과 질의의 이해도를 높이기 위해 사용

4.4 SELECT문(계속)

- SELECT 절에서 * 사용 주의해야 한다.(처리 시간 많이 소요, 응용 프로그램내에 SELECT문에 *가 포함되면 릴레이션에 애트리뷰트가 추가되면, SELECT 문에서 검색되는 애트리뷰트들의 개수가 변하게 되므로 응용프로그램을 다시 컴파일해야 한다.

- 릴레이션의 모든 애트리뷰트나 일부 애트리뷰트들을 검색(실습과제)

예 : *를 사용하여 모든 애트리뷰트들을 검색

질의: 전체 부서의 모든 애트리뷰트들을 검색하라.

```
SELECT      *
FROM        DEPARTMENT;
```

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

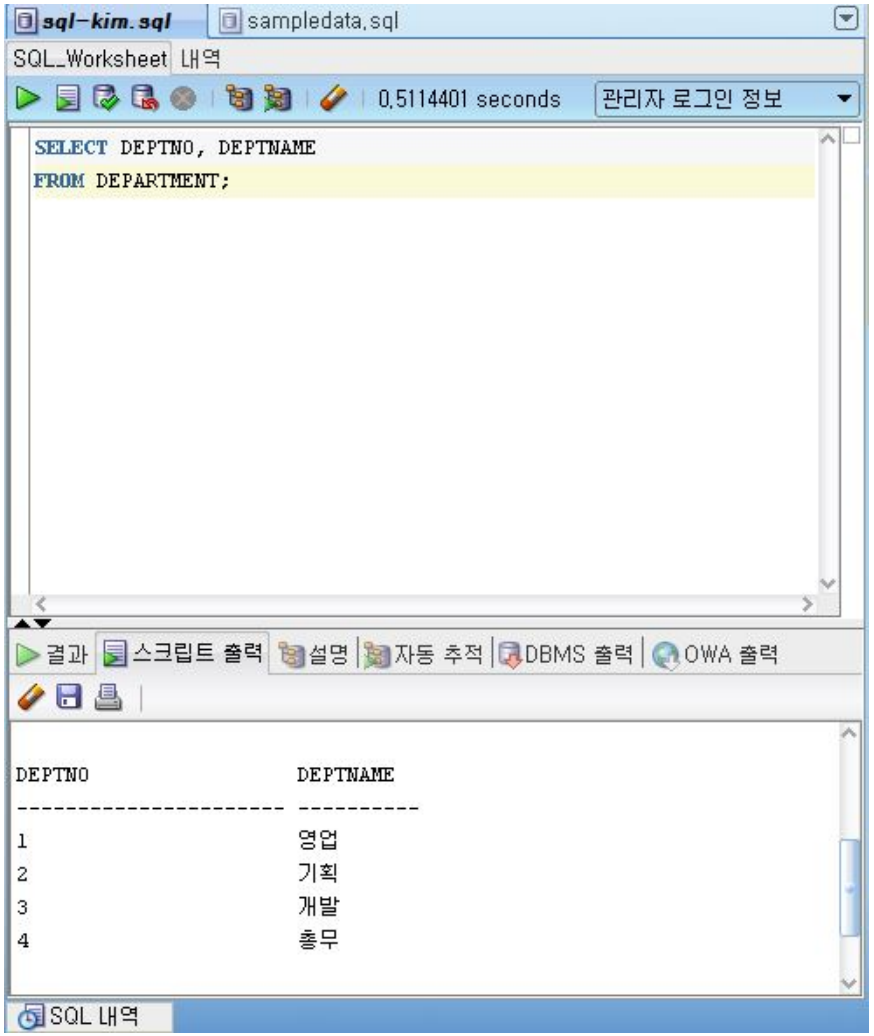
4.4 SELECT문(계속)

예 : 원하는 애트리뷰트들의 이름을 열거

질의: 모든 부서의 부서번호와 부서이름을 검색하라.

```
SELECT      DEPTNO, DEPTNAME
FROM        DEPARTMENT;
```

DEPTNO	DEPTNAME
1	영업
2	기획
3	개발
4	총무



4.4 SELECT문(계속)

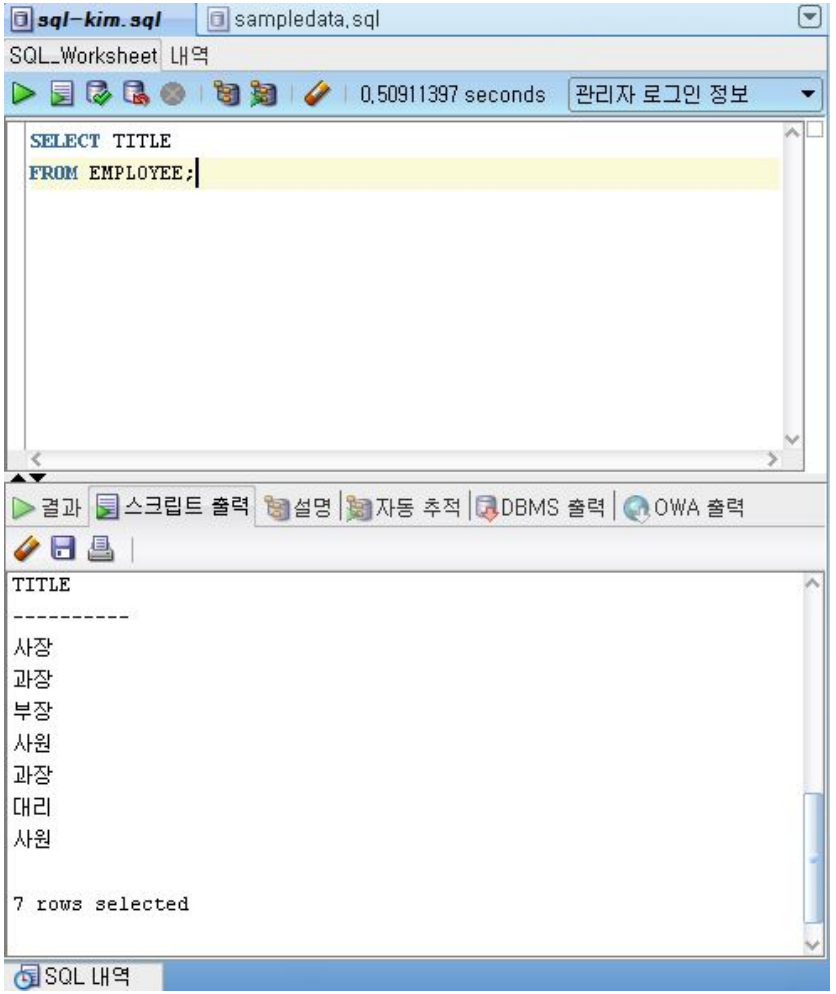
❑ 상이한 값들을 검색

예 : DISTINCT절을 사용하지 않을 때

질의: 모든 사원들의 직급을 검색하라.

```
SELECT      TITLE
FROM        EMPLOYEE;
```

TITLE
대리
과장
부장
과장
사원
사원
사장



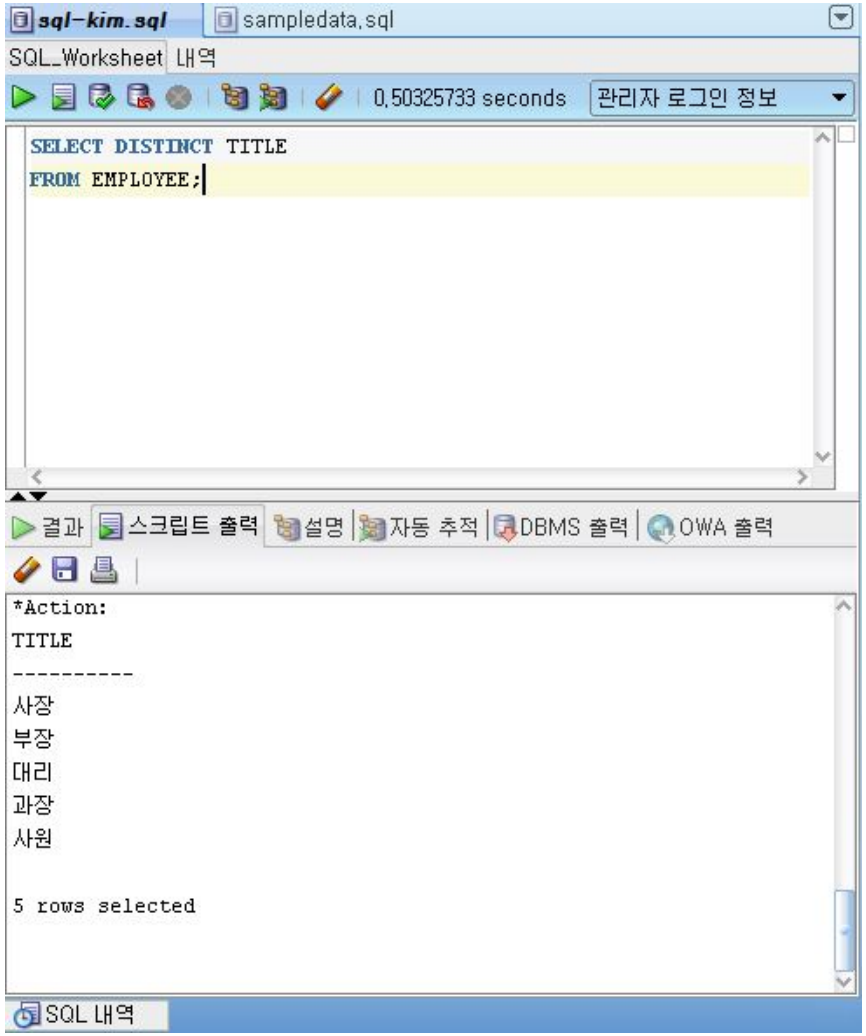
4.4 SELECT문(계속)

예 : DISTINCT절을 사용할 때

질의: 모든 사원들의 상이한 직급을 검색하라.

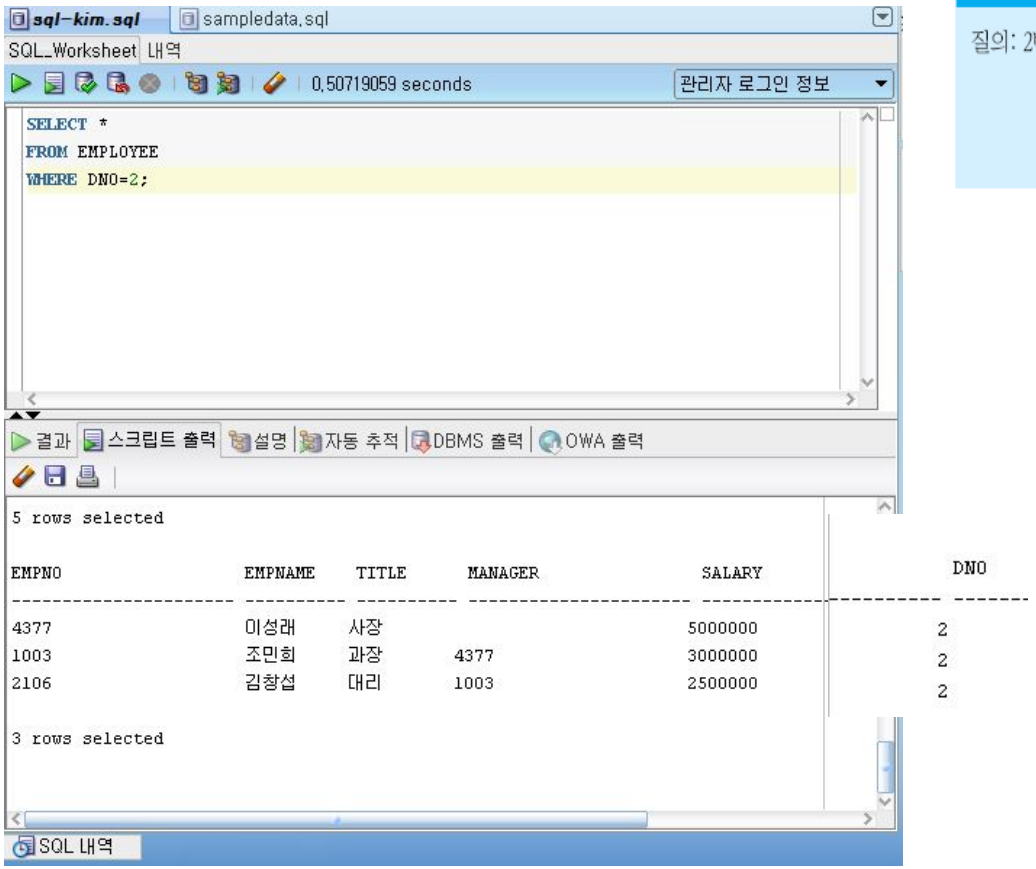
```
SELECT      DISTINCT TITLE
FROM        EMPLOYEE;
```

TITLE
대리
과장
부장
사원
사장



4.4 SELECT문(계속)

❑ 특정한 투플들의 검색



예 : WHERE절을 사용하여 검색 조건을 명시

질의: 2번 부서에 근무하는 사원들에 관한 모든 정보를 검색하라.

```
SELECT    *
FROM      EMPLOYEE
WHERE     DNO = 2;
```

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
1003	조민희	과장	4377	3000000	2
2016	김창섭	대리	1003	2500000	2
4377	이성래	사장	∧	5000000	2

4.4 SELECT문(계속)

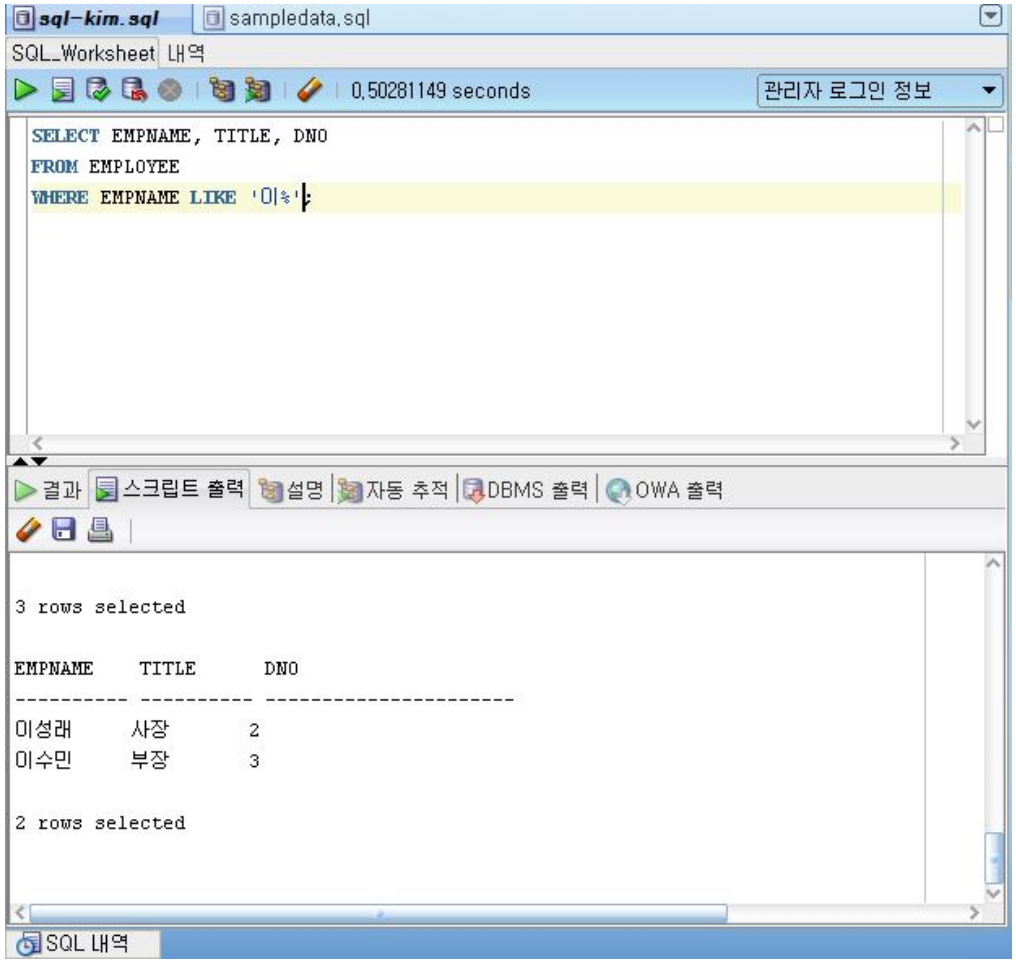
❑ 문자열 비교

예 : %를 사용하여 문자열 비교

질의: 이씨 성을 가진 사원들의 이름, 직급, 소속 부서번호를 검색하라.

```
SELECT    EMPNAME, TITLE, DNO
FROM      EMPLOYEE
WHERE     EMPNAME LIKE '이%';
```

EMPNAME	TITLE	DNO
이수민	부장	3
이성래	사장	2



4.4 SELECT문(계속)

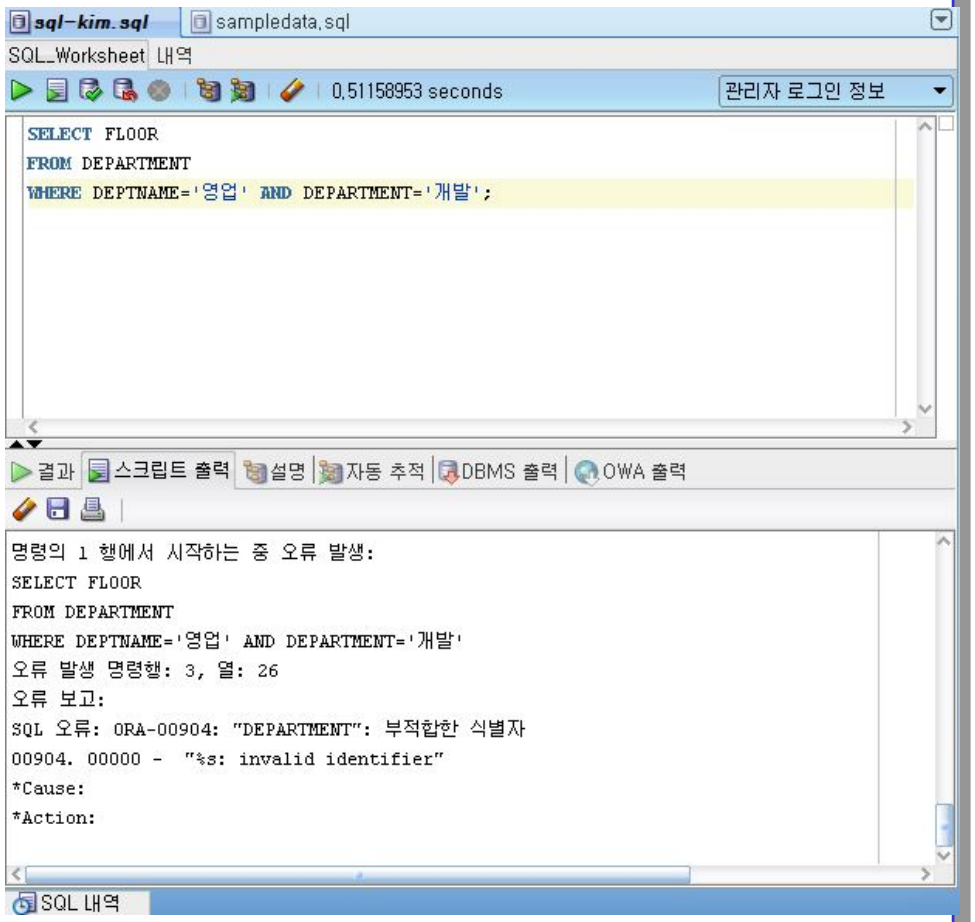
❑ 다수의 검색 조건

✓ 아래와 같은 질의는 잘못되었음

```
SELECT      FLOOR
FROM        DEPARTMENT
WHERE       DEPTNAME='영업' AND DEPTNAME='개발';
```

〈표 4.6〉 연산자들의 우선 순위

연산자	우선순위
비교 연산자	1
NOT	2
AND	3
OR	4



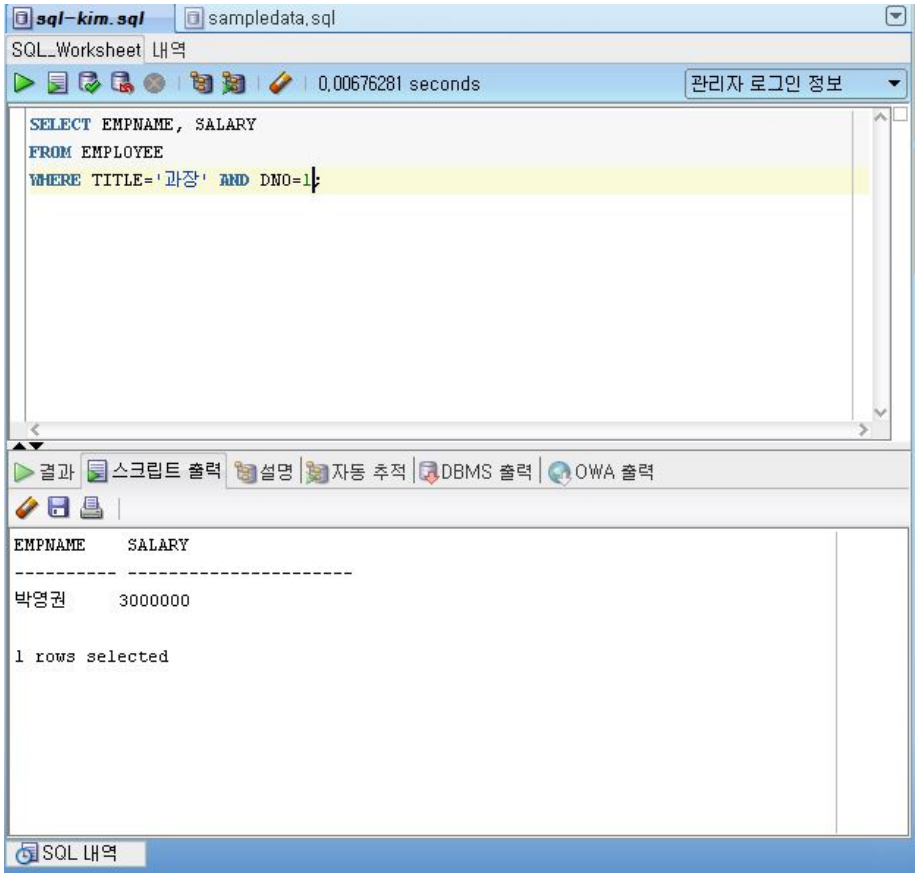
4.4 SELECT문(계속)

예 : 부울 연산자를 사용한 프레디키트

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

```
SELECT      EMPNAME, SALARY
FROM        EMPLOYEE
WHERE       TITLE = '과장' AND DNO = 1;
```

EMPNAME	SALARY
박영권	3000000



4.4 SELECT문(계속)

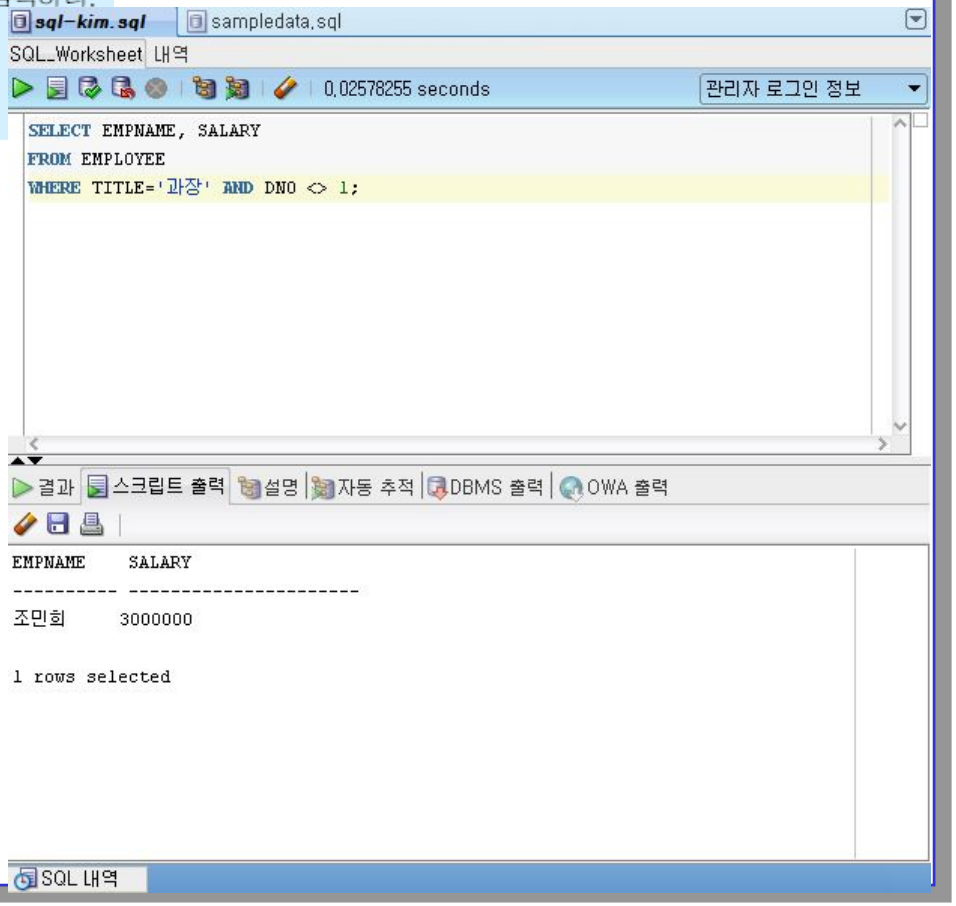
❑ 부정 검색 조건

예 : 부정 연산자

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

```
SELECT    EMPNAME, SALARY
FROM      EMPLOYEE
WHERE     TITLE = '과장' AND DNO <> 1;
```

EMPNAME	SALARY
조민희	3000000



4.4 SELECT문(계속)

❑ 범위를 사용한 검색

예 : 범위 연산자

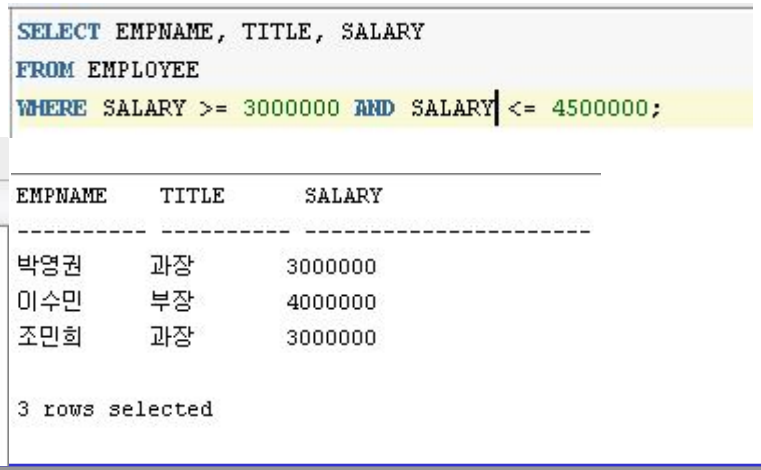
질의: 급여가 3000000원 이상이고, 4500000원 이하인 사원들의 이름, 직급, 급여를 검색하라.

```
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



4.4 SELECT문(계속)

❑ 리스트를 사용한 검색(IN은 리스트 내의 값과 비교)

예 : IN

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

```
SELECT      *
FROM        EMPLOYEE
WHERE       DNO IN (1, 3);
```

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
1365	김상원	사원	3426	1500000	1
3011	이수민	부장	4377	4000000	3
3426	박영권	과장	4377	3000000	1
3427	최종철	사원	3011	1500000	3

```
SELECT *
FROM EMPLOYEE
WHERE DNO IN(1,3);
```

```
EMPNO                EMPNAME    TITLE      MANAGER              SALARY               DNO
-------------------- ---------- ---------- -------------------- -------------------- -------
3426                 박영권     과장       4377                 3000000              1
3011                 이수민     부장       4377                 4000000              3
3427                 최종철     사원       3011                 1500000              3
1365                 김상원     사원       3426                 1500000              1

4 rows selected
```

4.4 SELECT문(계속)

❑ SELECT절에서 산술 연산자(+, -, *, /) 사용

예 : 산술 연산자

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

```
SELECT      EMPNAME, SALARY, SALARY * 1.1 AS NEWSALARY
FROM        EMPLOYEE
WHERE       TITLE = '과장';
```

EMPNAME	SALARY	NEWSALARY
박영권	3000000	3300000
조민희	3000000	3300000

```
SELECT EMPNAME, SALARY, SALARY*1.1 AS NEWSALARY
FROM EMPLOYEE
WHERE TITLE='과장';
```

```
EMPNAME    SALARY                NEWSALARY
---------- --------------------- ---------------------
박영권     3000000               3300000
조민희     3000000               3300000

2 rows selected
```

4.4 SELECT문(계속)

❑ 널값

✓ 널값을 포함한 다른 값과 널값을 +, – 등을 사용하여 연산하면 결과는 널

✓ COUNT(*)를 제외한 집단 함수들은 널값을 무시함

✓ 어떤 애트리뷰트에 들어 있는 값이 널인가 비교하기 위해서 'DNO=NULL'처럼 나타내면 안됨 (DNO IS NULL)

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

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

EMPNO                EMPNAME
-------------------- ----------

0 rows selected
```

4.4 SELECT문(계속)

❑ 널값(계속)

✓ 다음과 같은 비교 결과는 모두 거짓

NULL > 300

NULL = 300

NULL <> 300

NULL = NULL

NULL <> NULL

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

```
EMPNO                EMPNAME
-------------------- ----------

0 rows selected
```

✓ 올바른 표현(부정시 비교 연산자 :**IS NOT NULL**)

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

4.4 SELECT문(계속)

❑ 널값(계속)

✓ 올바른 표현(부정시 비교 연산자 :IS NOT NULL)

```
SELECT *
FROM EMPLOYEE
WHERE SALARY < 2000000 OR SALARY >=2000000;

SELECT *
FROM EMPLOYEE
WHERE SALARY < 2000000 OR SALARY >=2000000
OR SALARY IS NULL;
```

✓ 차이점: SALARY가 NULL인 투플도 검색

4.4 SELECT문(계속)

❑ ORDER BY절

- ✓ 사용자가 SELECT문에서 질의 결과의 순서를 명시하지 않으면 릴레이션에 투플들이 삽입된 순서대로 사용자에게 제시됨
- ✓ ORDER BY절에서 하나 이상의 애트리뷰트를 사용하여 검색 결과를 정렬할 수 있음
- ✓ SELECT문에서 가장 마지막에 사용되는 절
- ✓ 디폴트 정렬 순서는 오름차순(ASC)
- ✓ DESC를 지정하여 정렬 순서를 내림차순으로 지정할 수 있음
- ✓ 널값은 오름차순에서는 가장 마지막에 나타나고, 내림차순에서는 가장 앞에 나타남
- ✓ SELECT절에 명시한 애트리뷰트들을 사용해서 정렬해야 함

4.4 SELECT문(계속)

예 : ORDER BY

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

```
SELECT      SALARY, TITLE, EMPNAME
FROM        EMPLOYEE
WHERE       DNO = 2
ORDER BY    SALARY;
```

SALARY	TITLE	EMPNAME
2500000	대리	김창섭
3000000	과장	조민희
5000000	사장	이성래

```
SELECT SALARY, TITLE, EMPNAME
FROM EMPLOYEE
WHERE DNO = 2
ORDER BY SALARY;
```

```
SALARY               TITLE      EMPNAME
-------------------- ---------- ----------
2500000              대리       김창섭
3000000              과장       조민희
5000000              사장       이성래

3 rows selected
```

4.4 SELECT문(계속)

❑ORDER BY절

✓여러 개의 애트리뷰트를 사용하여 정렬

```
SELECT SALARY,TITLE, EMPNAME
 FROM EMPLOYEE
WHERE DNO = 2
ORDER BY TITLE, SALARY  DESC;
```

✓ TITLE 애트리뷰트 오름차순(ASC), SALARY 애트리뷰트는 내림차순(DESC) 정렬

```
SELECT SALARY, TITLE, EMPNAME
FROM EMPLOYEE
WHERE DNO = 2
ORDER BY TITLE, SALARY DESC;
```

```
SALARY               TITLE      EMPNAME
-------------------- ---------- ----------
3000000              과장       조민희
2500000              대리       김창섭
5000000              사장       이성래

3 rows selected
```

4.4 SELECT문(계속)

❑ 집단 함수

- ✓ 데이터베이스에서 검색된 여러 투플들의 집단에 적용되는 함수
- ✓ 한 릴레이션의 한 개의 애트리뷰트에 적용되어 단일 값을 반환함
- ✓ SELECT절과 HAVING절에만 나타날 수 있음
- ✓ COUNT(*)를 제외하고는 널값을 제거한 후 남아 있는 값들에 대해서 집단 함수의 값을 구함
- ✓ COUNT(*)는 결과 릴레이션의 모든 행들의 총 개수를 구하는 반면에 COUNT(애트리뷰트)는 해당 애트리뷰트에서 널값이 아닌 값들의 개수를 구함
- ✓ 키워드 DISTINCT가 집단 함수 앞에 사용되면 집단 함수가 적용되기 전에 먼저 중복을 제거함

4.4 SELECT문(계속)

예 : 집단 함수

질의: 모든 사원들의 평균 급여와 최대 급여를 검색하라.

```
SELECT    AVG(SALARY) AS AVGSAL, MAX(SALARY) AS MAXSAL
FROM      EMPLOYEE;
```

AVGSAL	MAXSAL
2928571	5000000

〈표 4.10〉 집단 함수의 기능

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

```
SELECT AVG(SALARY)AS AVGSAL, MAX(SALARY)AS MAXSAL
FROM EMPLOYEE;

AVGSAL                    MAXSAL
------------------------- -------------------------
2928571.428571428571428571428571428571428571 5000000

1 rows selected
```

```
SELECT ROUND(AVG(SALARY))AS AVGSAL,MAX(SALARY)AS MAXSAL
FROM EMPLOYEE;

AVGSAL                    MAXSAL
------------------------- -------------------------
2928571                   5000000

1 rows selected
```

❖ 집단함수 결과의 소수점이하 절사방법

4.4 SELECT문(계속)

❑ 그룹화

- ✓ GROUP BY절에 사용된 애트리뷰트에 동일한 값을 갖는 투플들이 각각 하나의 그룹으로 묶임
- ✓ 이 애트리뷰트를 그룹화 애트리뷰트(grouping attribute)라고 함
- ✓ 각 그룹에 대하여 결과 릴레이션에 하나의 투플이 생성됨
- ✓ SELECT절에는 각 그룹마다 하나의 값을 갖는 애트리뷰트, 집단 함수, 그룹화에 사용된 애트리뷰트들만 나타날 수 있음
- ✓ 다음 질의는 그룹화를 하지 않은 채 EMPLOYEE 릴레이션의 모든 투플에 대해서 사원번호와 모든 사원들의 평균 급여를 검색하므로 잘못됨

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

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

```
명령의 1 행에서 시작하는 중 오류 발생:
SELECT EMPNO, AVG(SALARY)
FROM EMPLOYEE
오류 발생 명령행: 1, 열: 7
오류 보고:
SQL 오류: ORA-00937: 단일 그룹의 그룹 함수가 아닙니다
00937. 00000 -  "not a single-group group function"
*Cause:
*Action:
```

4.4 SELECT문(계속)

예 : 그룹화

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

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

group by DNO; 생략하면 오류 발생!

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

```
DNO                  AVGSALARY            MAXSALARY
-------------------- -------------------- ----------
1                    2250000              3000000
2                    3500000              5000000
3                    2750000              4000000

3 rows selected
```

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	사장	∧	5000000	2
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3

그룹

DNO	AVGSAL	MAXSAL
1	2250000	3000000
2	3500000	5000000
3	2750000	4000000

4.4 SELECT문(계속)

❑ HAVING절

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

4.4 SELECT문(계속)

예 : 그룹화

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

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

★자던 사람도 알아야 하는 부분??

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	사장	∧	5000000	2
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3

그룹

GROUP BY →

DNO	AVGSAL	MAXSAL
1	2250000	3000000
2	3500000	5000000
3	2750000	4000000

HAVING →

DNO	AVGSAL	MAXSAL
2	3500000	5000000
3	2750000	4000000

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

```
DNO                  AVGSALARY            MAXSALARY
-------------------- -------------------- ---------------
2                    3500000              5000000
3                    2750000              4000000

2 rows selected
```

4.4 SELECT문(계속)

합집합 호환성(필요충분조건)
- 차수가 같아야한다
- 도메인이 같아야한다

❑ 집합 연산

- ✓ 집합 연산을 적용하려면 두 릴레이션이 합집합 호환성을 가져야 함
- ✓ 첫번째 질의의 애트리뷰트 이름들이 결과에 나타난다.
- ✓ UNION ALL(합집합)을 제외하고 결과가 오름차순으로 정렬된다.
- ✓ UNION ALL(합집합)을 제외하고 모든 집합연산의 결과 릴레이션에서 중복된 투플들이 자동적으로 삭제된다.

- ✓ UNION(합집합), EXCEPT(차집합), INTERSECT(교집합), UNION ALL(합집합), EXCEPT ALL(차집합), INTERSECT ALL(교집합)

4.4 SELECT문(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');
```

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

DNO
2
3

```
DNO
--------------------
2
3

2 rows selected
```

EMPLOYEE

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

DEPARTMENT

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

4.4 SELECT문(계속)

❑ 조인

- ✓ 두 개 이상의 릴레이션으로부터 연관된 투플들을 결합
- ✓ 일반적인 형식은 아래의 SELECT문과 같이 FROM절에 두 개 이상의 릴레이션들이 열거되고, 두 릴레이션에 속하는 애트리뷰트들을 비교하는 조인 조건이 WHERE절에 포함됨
- ✓ 조인 조건은 두 릴레이션 사이에 속하는 애트리뷰트 값들을 비교 연산자로 연결한 것
- ✓ 가장 흔히 사용되는 비교 연산자는 =

```
SELECT   …
FROM     R, S
WHERE    R.A 〈비교 연산자〉 S.B ;
```

조인 조건

4.4 SELECT문(계속)

❑ 조인(계속)

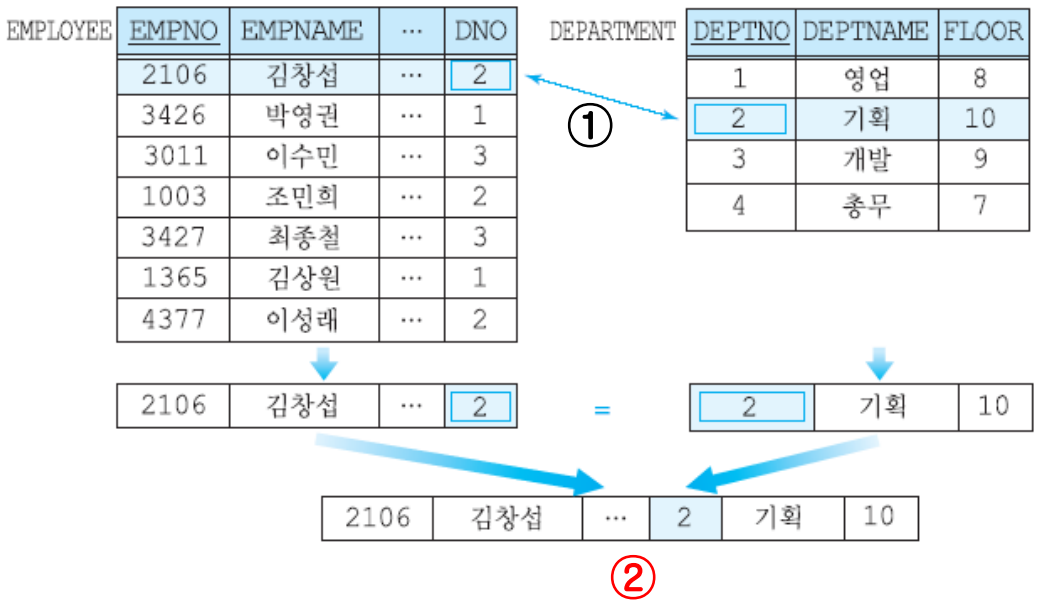
- ✓ 조인 조건을 생략했을 때와 조인 조건을 틀리게 표현했을 때는 카티션 곱(catesian product)이 생성됨.
- ✓ 조인 질의가 수행되는 과정을 개념적으로 살펴보면 ①먼저 조인 조건을 만족하는 투플들을 찾고, ②이 투플들로부터 SELECT절에 명시된 애트리뷰트들만 프로젝트하고, ③필요하다면 중복을 배제하는 순서로 진행됨
- ✓ 조인 조건이 명확해지도록 애트리뷰트 이름 앞에 릴레이션 이름이나 투플 변수를 사용하는 것이 바람직
- ✓ 두 릴레이션의 조인 애트리뷰트 이름이 동일하다면 반드시 애트리뷰트 이름 앞에 릴레이션 이름이나 투플 변수를 사용해야 함

4.4 SELECT문(계속)

예 : 조인 질의

질의: 모든 사원의 이름과 이 사원이 속한 부서 이름을 검색하라.

```
SELECT    EMPNAME, DEPTNAME
FROM      EMPLOYEE AS E, DEPARTMENT AS D
WHERE     E.DNO = D.DEPTNO;
```




```
SELECT EMPNAME, DEPTNAME
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DNO=DEPARTMENT.DEPTNO;
```

```
이성래      기획
박영권      영업
이수민      개발
최종철      개발
조민희      기획
김창섭      기획
김상원      영업

7 rows selected
```

4.4 SELECT문(계속)

최종 결과 릴레이션은 아래의 릴레이션에서 EMPNAME과 DEPTNAME을 프로젝션한 것이다.

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

4.4 SELECT문(계속)

- 릴레이션 엘리어스 사용법(오라클에서는 주로 ③ 사용)

①

```
SELECT EMPNAME, DEPTNAME
FROM EMPLOYEE AS E, DEPARTMENT AS D
WHERE E.DNO=D.DEPTNO;
```

오류!인데 왜 오류인지 모르겠음

```
명령의 1 행에서 시작하는 중 오류 발생:
SELECT EMPNAME, DEPTNAME
FROM EMPLOYEE AS E, DEPARTMENT AS D
WHERE E.DNO=D.DEPTNO
오류 발생 명령행: 2, 열: 14
오류 보고:
SQL 오류: ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다
00933. 00000 -  "SQL command not properly ended"
*Cause:
*Action:
```

②

```
SELECT EMPNAME, DEPTNAME
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DNO=DEPARTMENT.DEPTNO;
```

이성래	기획
박영권	영업
이수민	개발
최종철	개발
조민희	기획
김창섭	기획
김상원	영업

7 rows selected

③

```
SELECT EMPNAME, DEPTNAME
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DNO=D.DEPTNO;
```

이성래	기획
박영권	영업
이수민	개발
최종철	개발
조민희	기획
김창섭	기획
김상원	영업

7 rows selected

4.4 SELECT문(계속)

❑ 자체 조인(self join)

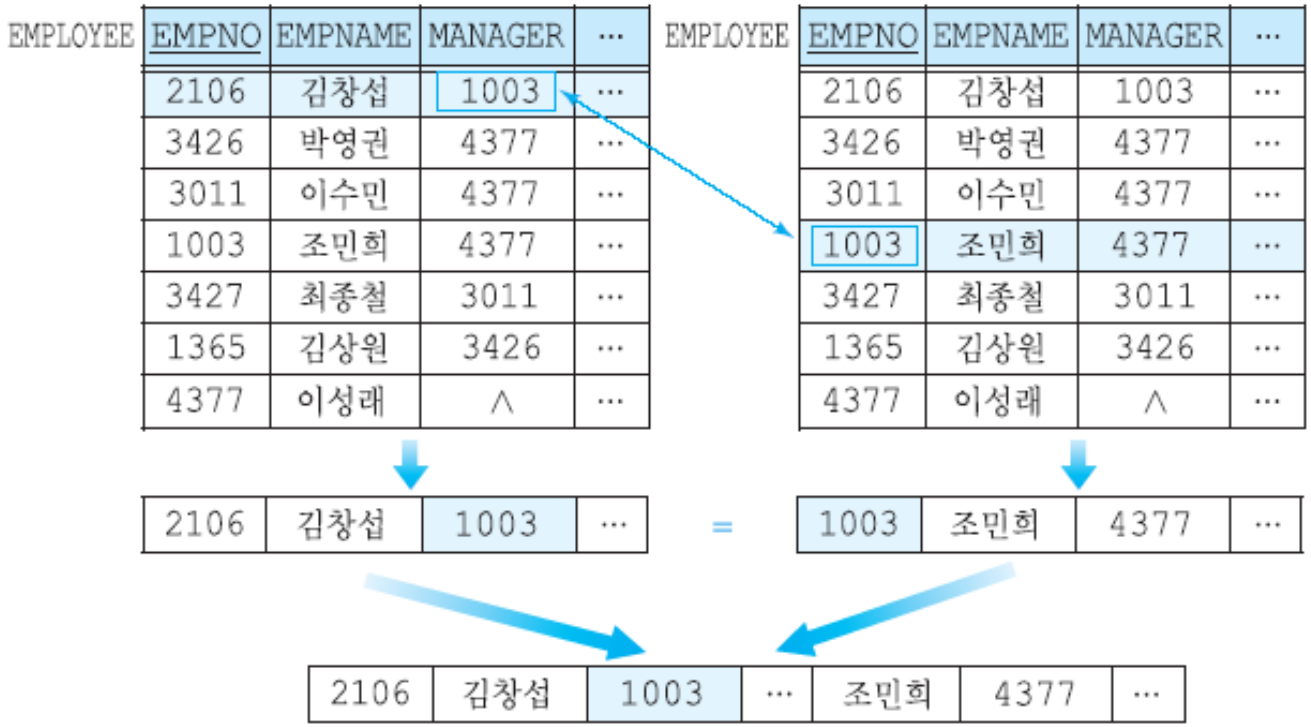
- ✓ 한 릴레이션에 속하는 투플을 동일한 릴레이션에 속하는 투플들과 조인하는 것
- ✓ 실제로는 한 릴레이션이 접근되지만 FROM절에 두 릴레이션이 참조되는 것처럼 나타내기 위해서 그 릴레이션에 대한 별칭을 두 개 지정해야 함

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;
```

4.4 SELECT문(계속)



4.4 SELECT문(계속)

최종 결과 릴레이션은 아래와 같다.

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

EMPLOYEE

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

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

```
EMPNAME    EMPNAME
---------- ----------
조민희     이성래
이수민     이성래
박영권     이성래
김상원     박영권
최종철     이수민
김창섭     조민희

6 rows selected
```

4.4 SELECT문(계속)

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.

```
SELECT      DEPTNAME, EMPNAME, TITLE, SALARY
FROM        EMPLOYEE E, DEPARTMENT D
WHERE       E.DNO = D.DEPTNO
ORDER BY    DEPTNAME, SALARY DESC;
```

오름차순

DEPTNAME	EMPNAME	TITLE	SALARY
개발	이수민	부장	4000000
개발	최종철	사원	1500000
기획	이성래	사장	5000000
기획	조민희	과장	3000000
기획	김창섭	대리	2500000
영업	박영권	과장	3000000
영업	김상원	사장	1500000

내림차순 (SALARY)

4.4 SELECT문(계속)

```
SELECT DEPTNAME,EMPNAME, TITLE, SALARY
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DNO=D.DEPTNO
ORDER BY DEPTNAME, SALARY DESC;
```

```
DEPTNAME   EMPNAME    TITLE      SALARY
---------- ---------- ---------- --------------------
개발       이수민     부장       4000000
개발       최종철     사원       1500000
기획       이성래     사장       5000000
기획       조민희     과장       3000000
기획       김창섭     대리       2500000
영업       박영권     과장       3000000
영업       김상원     사원       1500000

7 rows selected
```

4.4 SELECT문(계속)

- 중첩 질의(nested query)
  - ✓ 외부 질의의 WHERE절에 다시 SELECT ... FROM ... WHERE 형태로 포함된 SELECT문
  - ✓ 부질의(subquery)라고 함
  - ✓ INSERT, DELETE, UPDATE문에도 사용될 수 있음
  - ✓ 중첩 질의의 결과로 한 개의 스칼라값(단일 값), 한 개의 애트리뷰트로 이루어진 릴레이션, 여러 애트리뷰트로 이루어진 릴레이션이 반환될 수 있음

이 반환된 값은 첫 SELECT의 조건으로 사용함

4.4 SELECT문(계속)

[그림 4.10] 중첩 질의의 구조

4.4 SELECT문(계속)

❑ 한 개의 스칼라값이 반환되는 경우

예 : 한개의 스칼라 값이 반환되는 경우

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

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

과장

중첩 질의

EMPNAME	TITLE
박영권	과장
조민희	과장

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

```
EMPNAME    TITLE
---------- ----------
박영권     과장
조민희     과장

2 rows selected
```

4.4 SELECT문(계속)

❑ 한 개의 애트리뷰트로 이루어진 릴레이션이 반환되는 경우

- ✓ 중첩 질의의 결과로 한 개의 애트리뷰트로 이루어진 다수의 투플들이 반환될 수 있음
- ✓ 외부 질의의 WHERE절에서 IN, ANY(SOME), ALL, EXISTS와 같은 연산자를 사용해야 함
- ✓ 키워드 IN은 한 애트리뷰트가 값들의 집합에 속하는가를 테스트할 때 사용됨
- ✓ 한 애트리뷰트가 값들의 집합에 속하는 하나 이상의 값들과 어떤 관계를 갖는가를 테스트하는 경우에는 ANY를 사용
- ✓ 한 애트리뷰트가 값들의 집합에 속하는 모든 값들과 어떤 관계를 갖는가를 테스트하는 경우에는 ALL을 사용

4.4 SELECT문(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.

예 : ANY

(3000000 < ANY {2500000, 3000000, 4000000})은 참이다.

(4000000 < ANY {2500000, 3000000, 4000000})은 거짓이다.

*. IN : 한 애트리뷰트가 값들의 집합에 속하는가?

*. ANY : 한 애트리뷰트가 값들의 집합에 속하는 하나 이상의 값들과 어떤 관계를 갖는가?

*. ALL : 한 애트리뷰트가 값들의 집합에 속하는 모든 값들과 어떤 관계를 갖는가를

4.4 SELECT문(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.

*. ALL : 한 애트리뷰트가 값들의 집합에 속하는 모든 값들과 어떤 관계를 갖는가?

4.4 SELECT문(계속)

예 : IN을 사용한 질의

질의: 영업부나 개발부에 근무하는 사원들의 이름을 검색하라.

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

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

```
EMPNAME
----------
박영권
이수민
최종철
김상원

4 rows selected
```

4.4 SELECT문(계속)

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

```
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

4 rows selected
```

4.4 SELECT문(계속)

❑ 여러 애트리뷰트들로 이루어진 릴레이션이 반환되는 경우

- ✓ 중첩 질의의 결과로 여러 애트리뷰트들로 이루어진 릴레이션이 반환되는 경우에는 EXISTS 연산자를 사용하여 중첩 질의의 결과가 빈 릴레이션인지 여부만를 검사함
- ✓ 중첩 질의의 결과가 적어도 하나의 투플이 들어 있으면 참이 되고, 그렇지 않으면 거짓
- ✓ NOT EXISTS 반대 개념.

4.4 SELECT문(계속)

❑여러 애트리뷰트들로 이루어진 릴레이션이 반환되는 경우

✓중첩 질의의 결과가 적어도 하나의 투플이 들어 있으면 참이 되고, 그렇지 않으면 거짓(상관중첩 질의)

예 : EXISTS를 사용한 질의

질의: 영업부나 개발부에 근무하는 사원들의 이름을 검색하라.

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

```
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

```
EMPNAME
----------
박영권
이수민
최종철
김상원

4 rows selected
```

4.4 SELECT문(계속)

- 상관 중첩 질의(correlated nested query)
  - ✓ 중첩 질의의 WHERE절에 있는 프레디키트(실렉션 조건)에서 외부 질의에 선언된 릴레이션의 일부 애트리뷰트를 참조하는 질의
  - ✓ 중첩 질의의 수행 결과가 단일 값이든, 하나 이상의 애트리뷰트로 이루어진 릴레이션이든 외부 질의로 한 번만 결과를 반환하면 상관 중첩 질의가 아님
  - ✓ 상관 중첩 질의에서는 외부 질의를 만족하는 각 투플이 구해진 후에 중첩 질의가 수행되므로 상관 중첩 질의는 외부 질의를 만족하는 투플 수만큼 여러 번 수행될 수 있음

where절

4.4 SELECT문(계속)

❑상관 중첩 질의(correlated nested query)

✓릴레이션 이름을 함께 표현하지 않은 DNO 애트리뷰트는 중첩이름을 중첩 질의의 EMPLOYEE 릴레이션을 참조한다. 외부질의에 명시된 EMPLOYEE 릴레이션의 DNO 애트리뷰트를 참조하려면 E.DNO와 같이 별칭을 사용해야

예 : 상관 중첩 질의

질의: 자신이 속한 부서의 사원들의 평균 급여보다 많은 급여를 받는 사원들에 대해서 이름, 부서번호, 급여를 검색하라.

```
SELECT    EMPNAME, DNO, SALARY
FROM      EMPLOYEE E
WHERE     SALARY >
          (SELECT   AVG(SALARY)
           FROM     EMPLOYEE
           WHERE    DNO = E.DNO);
```

상관

```
SELECT EMPNAME, DNO, SALARY
FROM EMPLOYEE E
WHERE SALARY >
      (SELECT AVG(SALARY)
      FROM EMPLOYEE
      WHERE DNO = E.DNO);
```

EMPNAME	DNO	SALARY
박영권	1	3000000
이수민	3	4000000
이성래	2	5000000

```
EMPNAME    DNO                  SALARY
---------- -------------------- --------------------
이성래     2                    5000000
박영권     1                    3000000
이수민     3                    4000000

3 rows selected
```

DML

4.5 INSERT, DELETE, UPDATE문

❑ INSERT문

- ✓ 기존의 릴레이션에 투플을 삽입

employee가 참조하는 릴레이션
department가 참조되는 릴레이션

- ✓ 참조되는 릴레이션에 투플이 삽입되는 경우에는 참조 무결성 제약조건의 위배가 발생하지 않으나 참조하는 릴레이션에 투플이 삽입되는 경우에는 참조 무결성 제약조건을 위배할 수 있음(FK선언부 있음=EMPLOYEE)
- ✓ 릴레이션에 한 번에 한 투플씩 삽입하는 것과 한 번에 여러 개의 투플들을 삽입할 수 있는 것으로 구분
- ✓ 릴레이션에 한 번에 한 투플씩 삽입하는 INSERT문

```
INSERT
INTO    릴레이션(애트리뷰트1, ..., 애트리뷰트n)
VALUES (값1, ..., 값n);
```

4.5 INSERT, DELETE, UPDATE문(계속)

예 : 한 개의 투플을 삽입

질의: DEPARTMENT 릴레이션에 (5, 연구, ∧) 투플을 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	0

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

```
1 행 삽입됨
```

```
SELECT *
FROM DEPARTMENT;
```

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

5 rows selected
```

4.5 INSERT, DELETE, UPDATE문(계속)

❑ INSERT문(계속)

✓ 릴레이션에 한 번에 여러 개의 투플들을 삽입하는 INSERT문

```
INSERT
INTO    릴레이션(애트리뷰트1, ..., 애트리뷰트n)
SELECT   ...  FROM   ...  WHERE   ...;
```

예 : 여러 개의 투플을 삽입

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

```
INSERT     INTO HIGH_SALARY(ENAME, TITLE, SAL)
SELECT     EMPNAME, TITLE, SALARY
FROM       EMPLOYEE
WHERE      SALARY >= 3000000;
```

```
CREATE TABLE HIGH_SALARY (
  ENAME    CHAR(10) UNIQUE,
  TITLE    CHAR(10)  DEFAULT '사원',
  SAL      NUMBER
  );
```

```
CREATE TABLE을(를) 성공했습니다.
```

4.5 INSERT, DELETE, UPDATE문(계속)

❑ INSERT문(계속)

✓ 데이터베이스 외부에서 릴레이션에 다수의 투플을 쉽게 삽입할 수 있도록 오라클에서는 SQL Loader 프로그램을 제공한다.

적재하는 기계? 라는 뜻

```
INSERT INTO HIGH_SALARY(ENAME, TITLE, SAL)
SELECT  EMPNAME, TITLE, SALARY
FROM employee
WHERE SALARY >=3000000;
```

```
4 행 삽입됨
```

```
SELECT *
FROM HIGH_SALARY;
```

```
ENAME      TITLE      SAL
---------- ---------- ----------
이성래     사장       5000000
박영권     과장       3000000
이수민     부장       4000000
조민희     과장       3000000

4 rows selected
```

4.5 INSERT, DELETE, UPDATE문(계속)

❑ DELETE문

- ✓ 한 릴레이션으로부터 한 개 이상의 투플들을 삭제함
- ✓ 참조되는(DEPARTMENT) 릴레이션의 삭제 연산의 결과로 참조 무결성 제약조건이 위배될 수 있으나, 참조하는(EMPLOYEE) 릴레이션에서 투플을 삭제하면 참조 무결성 제약조건을 위배하지 않음
- ✓ DELETE문의 구문

```
DELETE
FROM      릴레이션
WHERE     조건;
```

예 : DELETE문

질의: DEPARTMENT 릴레이션에서 5! 부서를 삭제하라.

```
DELETE FROM DEPARTMENT
WHERE DEPTNO = 5;
```

```
SELECT *
FROM DEPARTMENT ;
```

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

4 rows selected

```
DELETE FROM department
WHERE DEPTNO = 5;
```

1 행 삭제됨

4.5 INSERT, DELETE, UPDATE문(계속)

❑ DELETE문

예 : DELETE문

질의: DEPARTMENT 릴레이션에서 5번 부서를 삭제하라.

```
DELETE FROM DEPARTMENT
WHERE DEPTNO = 5;
```

```
DELETE FROM department
WHERE DEPTNO = 5;

1 행 삭제됨
```

```
SELECT *
FROM DEPARTMENT ;
```

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

4 rows selected

4.5 INSERT, DELETE, UPDATE문(계속)

❑ UPDATE문

✓ 한 릴레이션에 들어 있는 투플들의 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;
```

4.5 INSERT, DELETE, UPDATE문(계속)

❑ UPDATE문

예 : UPDATE문

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

교수님이 지우라고 하신 부분

```
SELECT *
FROM EMPLOYEE;
```

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

4.5 INSERT, DELETE, UPDATE문(계속)

❑ UPDATE문

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;
```

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

1 행 갱신됨

```
SELECT *
FROM EMPLOYEE;
```

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
4377	이성래	사장		5000000	2
3426	박영권	과장	4377	3000000	1
3011	이수민	부장	4377	4000000	3
3427	최종철	사원	3011	1500000	3
1003	조민희	과장	4377	3000000	2
2106	김창섭	대리	1003	2625000	3
1365	김상원	사원	3426	1500000	1

방아쇠

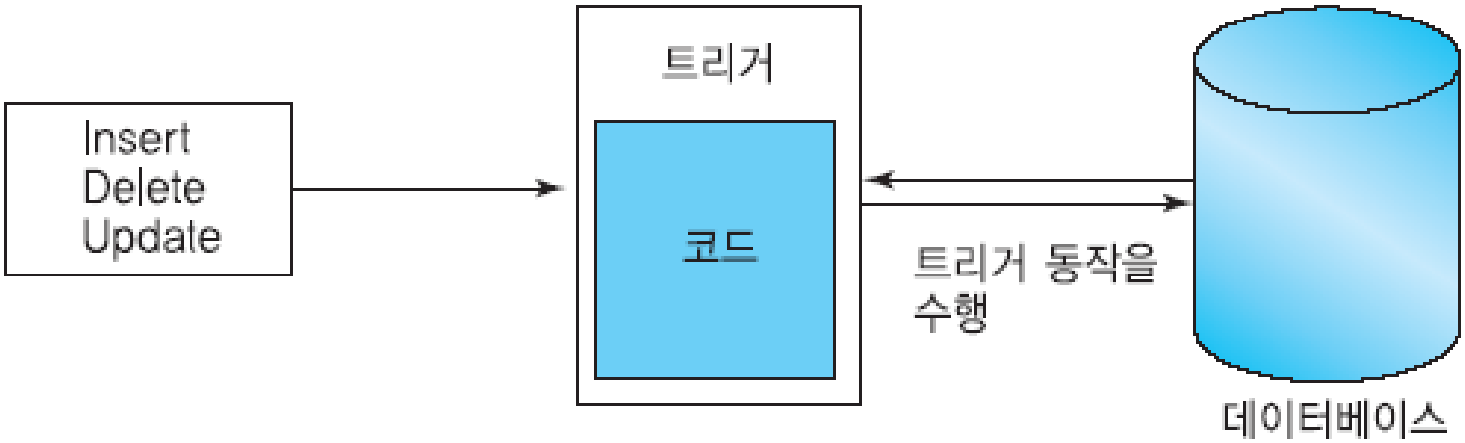
4.6 트리거(trigger)와 주장(assertion)

❑ 트리거

- ✓ 명시된 이벤트(데이터베이스의 갱신)가 발생할 때마다 DBMS가 자동적으로 수행하는, 사용자가 정의하는 문(프로시저)
- ✓ 데이터베이스의 무결성을 유지하기 위한 일반적이고 강력한 도구
- ✓ 테이블 정의시 표현할 수 없는 기업의 비즈니스 규칙들을 시행하는 역할
- ✓ 트리거를 명시하려면 트리거를 활성화시키는 사건인 이벤트, 트리거가 활성화되었을 때 수행되는 테스트인 조건, 트리거가 활성화되고 조건이 참일 때 수행되는 문(프로시저)인 동작을 표현해야 함
- ✓ 트리거를 이벤트-조건-동작(ECA) 규칙이라고도 부름

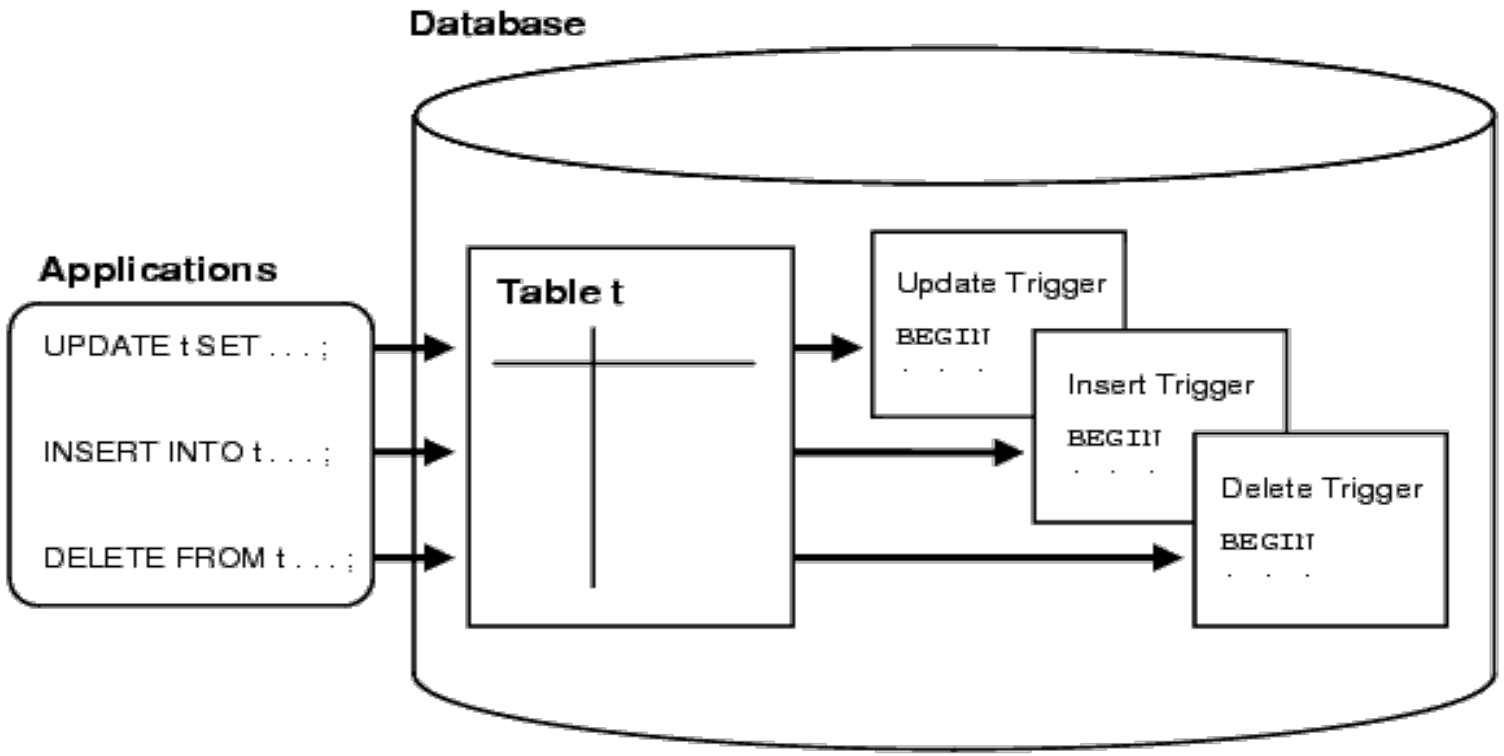
  E는 Event, C는 Condition, A는 Action을 의미
- ✓ SQL3 표준에 포함되었으며 대부분의 상용 관계 DBMS에서 제공됨

4.6 트리거와 주장(계속)



[그림 4.11] 트리거의 개념

4.6 트리거와 주장(계속)


Database
Applications
UPDATE t SET . . . ;
INSERT INTO t . . . ;
DELETE FROM t . . . ;
Table t
Update Trigger
BEGIN
. . .
Insert Trigger
BEGIN
. . .
Delete Trigger
BEGIN
. . .

4.6 트리거와 주장(계속)

❑ 트리거(계속)

- ✓ SQL3에서 트리거의 형식

```
CREATE TRIGGER  〈트리거 이름〉
AFTER   〈트리거를 유발하는 이벤트들이 OR로 연결된 리스트〉 ON 〈릴레이션〉   ← 이벤트
[WHEN   〈조건〉]                                                ← 조건
BEGIN   〈SQL문(들)〉 END                                        ← 동작
```

- ✓ 이벤트의 가능한 예로는 테이블에 투플 삽입, 테이블로부터 투플 삭제, 테이블의 투플 수정 등이 있음
- ✓ 조건은 임의의 형태의 프레디키트
- ✓ 동작은 데이터베이스에 대한 임의의 갱신
- ✓ 어떤 이벤트가 발생했을 때 조건이 참이 되면 트리거와 연관된 동작이 수행되고, 그렇지 않으면 아무 동작도 수행되지 않음
- ✓ 삽입, 삭제, 수정 등이 일어나기 전(before)에 동작하는 트리거와 일어난 후(after)에 동작하는 트리거로 구분

4.6 트리거(trigger)와 주장(assertion)

❑ 트리거

- 예제 : 단순한 메시지를 출력하는 트리거

사원 테이블에 새로운 데이터가 입력되면 '신입사원이 입사했습니다.'라는 메시지를 출력 하도록 문장 레벨 트리거를 작성해 보자.

① 트리거 실습 예제 테이블 작성

```
SQL> CREATE TABLE EMP_TRIG(
  2  EMPNO NUMBER(4) PRIMARY KEY,
  3  ENAME VARCHAR2(20),
  4  JOB VARCHAR2(20)
  5  );

테이블이 생성되었습니다.
```

②

```
SQL> ED TRIG01
```

```
CREATE OR REPLACE TRIGGER TRIG01
AFTER INSERT
ON EMP_TRIG
BEGIN
  DBMS_OUTPUT.PUT_LINE('신입사원이 입사했습니다.');
END;
/
```

4.6 트리거(trigger)와 주장(assertion)

❑ 트리거

- 예제 : 단순한 메시지를 출력하는 트리거

사원 테이블에 새로운 데이터가 입력되면 '신입사원이 입사했습니다.'라는 메시지를 출력 하도록 문장 레벨 트리거를 작성해 보자.

③트리거 실행

```
SQL> @TRIG01

트리거가 생성되었습니다.
```

④ EMP_TRIG 테이블에 새로운 데이터 입력

```
SQL> SET SERVEROUTPUT   ON
SQL> INSERT INTO EMP_TRIG VALUES(1, '박가천', '학생') ;
신입사원이 입사했습니다.

1 개의 행이 만들어졌습니다.
```

4.6 트리거와 주장(계속)

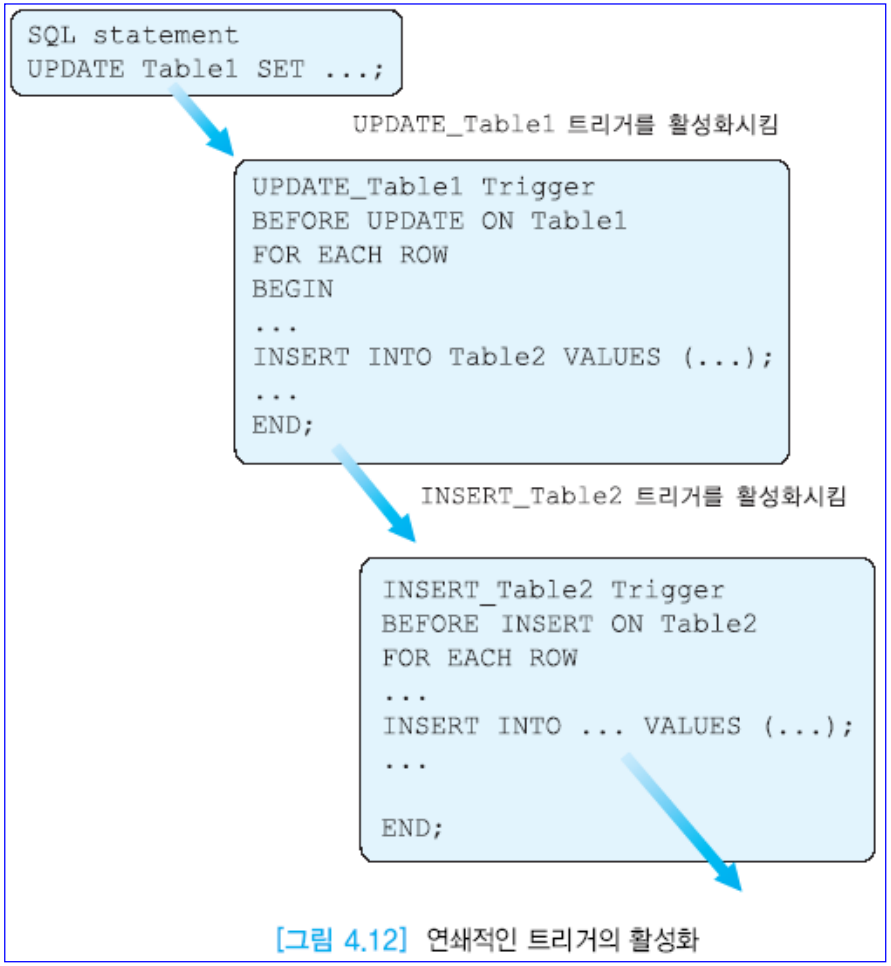
예 : 트리거

새로운 사원이 입사할 때마다, 사원의 급여가 1500000 미만인 경우에는 급여를 10% 인상하는 트리거를 작성하라. 여기서 이벤트는 새로운 사원 투플이 삽입될 때, 조건은 급여 < 1500000, 동작은 급여를 10% 인상하는 것이다. 오라클에서 트리거를 정의하는 문장은 SQL3의 트리거 정의문과 동일하지는 않다.

```
CREATE TRIGGER RAISE_SALARY
AFTER INSERT ON EMPLOYEE
REFERENCING NEW AS newEmployee
FOR EACH ROW
WHEN    (newEmployee.SALARY < 1500000)
UPDATE  EMPLOYEE
SET     newEmployee.SALARY = SALARY * 1.1
WHERE   EMPNO = newEmployee.EMPNO;
```

4.6 트리거와 주장(계속)

- 연쇄적으로 활성화되는 트리거
  - ✓ 하나의 트리거가 활성화되어 이 트리거 내의 한 **SQL**문이 수행되고, 그 결과로 다른 트리거를 활성화하여 그 트리거 내의 **SQL**문이 수행될 수 있음



[그림 4.12] 연쇄적인 트리거의 활성화

4.6 트리거와 주장(계속)

❑ 주장

- ✓ SQL3에 포함되어 있으나 대부분의 상용 관계 DBMS가 아직 지원하고 있지 않음
- ✓ 주장:데이터베이스가 항상 만족하기를 바라는 조건을 직접적으로 표현
- ✓ 트리거는 제약조건을 위반했을 때 수행할 동작을 명시하는 것이고, 주장은 제약조건을 위반하는 연산이 수행되지 않도록 함
- ✓ 주장의 구문

```
CREATE ASSERTION 이름
CHECK   조건;
```

- ✓ 트리거보다 좀더 일반적인 무결성 제약조건
- ✓ DBMS는 주장의 프레디키트를 검사하여 만일 참이면 주장을 위배하지 않는 경우이므로 데이터베이스 수정이 허용됨
- ✓ 일반적으로 두 개 이상의 테이블에 영향을 미치는 제약조건을 명시하기 위해 사용됨

4.6 트리거와 주장(계속)

예 : 주장

STUDENT(학생) 릴레이션과 ENROLL(수강) 릴레이션의 스키마가 아래와 같다. STUDENT 릴레이션의 기본 키는 STNO이다. ENROLL 릴레이션의 STNO는 STUDENT 릴레이션의 기본 키를 참조한다.

STUDENT(STNO, STNAME, EMAIL, ADDRESS, PHONE)
ENROLL(STNO, COURSENO, GRADE)

ENROLL 릴레이션에 들어 있는 STNO는 반드시 STUDENT 릴레이션에 들어 있는 어떤 학생의 STNO를 참조하도록 하는 주장을 정의하려 한다. 다시 말해서 STUDENT 릴레이션에 없는 어떤 학생의 학번이 ENROLL 릴레이션에 나타나는 것을 허용하지 않으려고 한다. 대부분의 주장은 아래의 예처럼 NOT EXISTS를 포함한다.

```
CREATE ASSERTION EnrollStudentIntegrity
CHECK   (NOT EXISTS
         (SELECT   *
          FROM     ENROLL
          WHERE    STNO NOT IN
                   (SELECT STNO FROM STUDENT)));
```

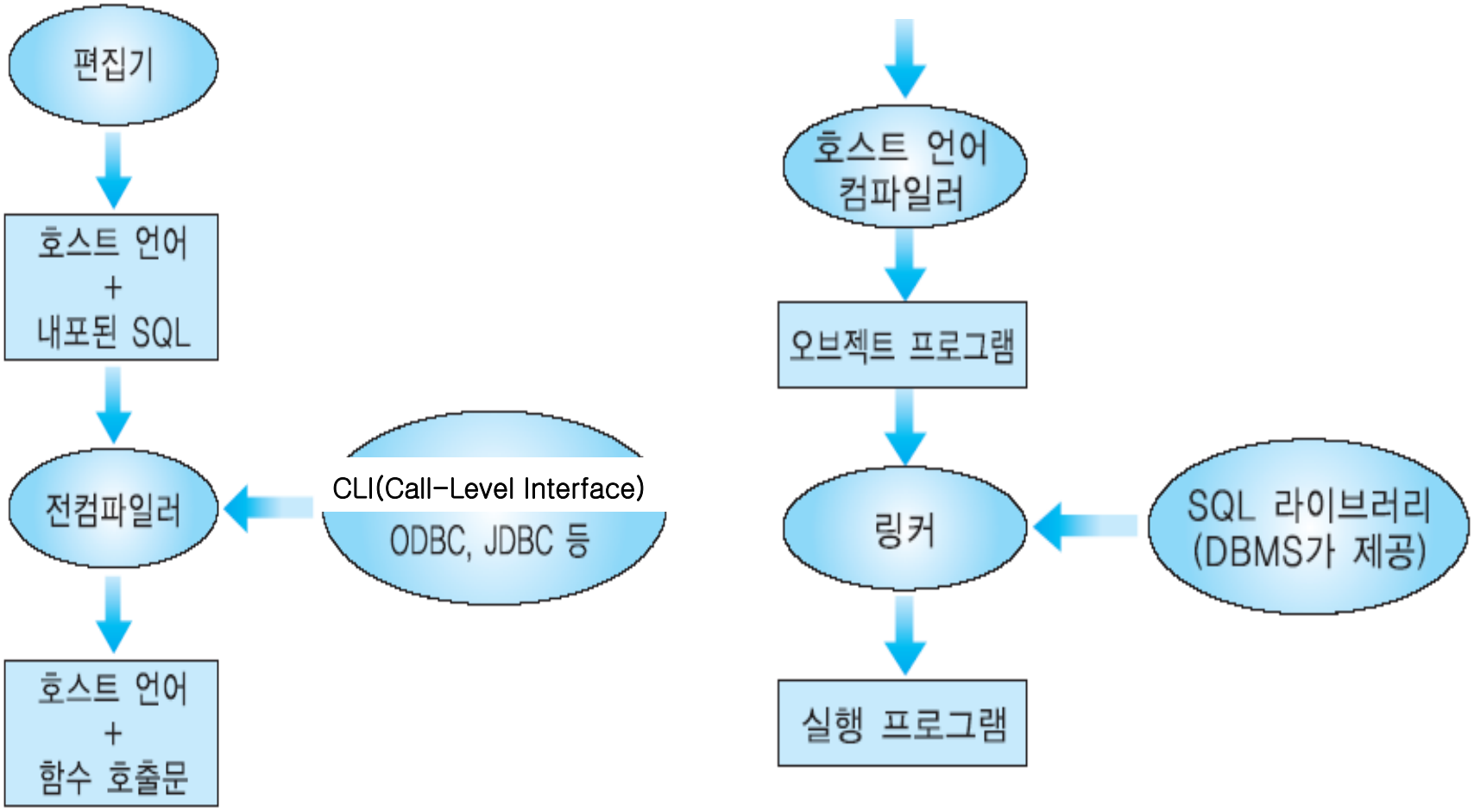
4.7 내포된 SQL

- 내포된 SQL(embedded SQL)
  - ✓ SQL이 호스트 언어의 완전한 표현력을 갖고 있지 않기 때문에 모든 질의를 SQL로 표현할 수는 없음
  - ✓ SQL은 호스트 언어가 갖고 있는 조건문(IF문), 반복문(WHILE문), 입출력 등과 같은 동작, 사용자와의 상호 작용, 질의 결과를 GUI로 보내는 등의 기능을 갖고 있지 않음
  - ✓ C, C++, 코볼, 자바 등의 언어로 작성하는 프로그램에 SQL문을 삽입하여, 데이터베이스를 접근하는 부분을 SQL이 맡고 SQL에 없는 기능은 호스트 언어로 작성하는 것이 필요
  - ✓ 호스트 언어에 포함되는 SQL문을 내포된 SQL이라 부름
  - ✓ 데이터 구조가 불일치하는 문제(impedance mismatch 문제)

4.7 내포된 SQL(계속)

- 내포된 SQL(계속)
  - ✓ 오라클에서 C 언어에 SQL문을 내포시키는 환경을 Pro*C라 부름
  - ✓ 일반적으로 내포된 SQL문이 포함된 소스 파일의 확장자는 .pc
  - ✓ 이 파일을 Pro*C를 통하여 전컴파일(precompiler)하면 확장자가 .c인 C 소스 프로그램이 생성됨
  - ✓ 호스트 언어로 작성 중인 프로그램에 SQL문을 내포시킬 때 해당 호스트 언어의 컴파일러가 어떻게 호스트 언어의 문과 SQL문을 구별할 것인가?
  - ✓ 호스트 언어로 작성 중인 프로그램에 포함된 SELECT, INSERT, DELETE, UPDATE 등 모든 SQL문에는 반드시 문장의 앞부분에 EXEC SQL을 붙임
  - ✓ Pro*C 전컴파일러는 내포된 SQL문을 C 컴파일러에서 허용되는 함수 호출로 변환함

4.7 내포된 SQL(계속)



[그림 4.13] 내포된 SQL문의 컴파일 과정

4.7 내포된 SQL(계속)

- Pro*C
  - ✓ 윈도우7 환경에서 Pro*C를 실습하려면 비주얼 스튜디오 6.0 등의 통합 개발 환경이 필요
  - ✓ 정적인 SQL문은 C 프로그램에 내포된 완전한 Transact-SQL문
  - ✓ 정적인 SQL문은 입력값과 출력 데이터를 위해서 C 프로그램의 변수들을 포함할 수 있음
  - ✓ 동적인 SQL문은 응용을 개발할 때 완전한 SQL문의 구조를 미리 알고 있지 않아도 됨
  - ✓ 동적인 SQL문은 불완전한 Transact-SQL문으로서 일부 또는 전부를 질의가 수행될 때 입력 가능
  - ✓ SQL문에 포함된 C 프로그램의 변수를 호스트 변수(host variable)라고 부름

4.7 내포된 SQL(계속)

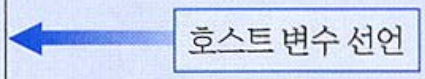
예 : 호스트 변수

아래의 부분 프로그램은 호스트 변수를 사용한 C 프로그램의 예를 보여준다. 이 프로그램은 사용자에게 사원의 번호를 입력하도록 하고, 사용자가 입력한 값을 호스트 변수 no에 저장한다. 그 다음에 프로그램은 DBSERVER 데이터베이스의 EMPLOYEE 릴레이션에서 그 사원의 직급을 검색하여 호스트 변수 title에 저장한다.

```
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
  int     no;
  varchar title[10];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA.H;     /* SQL 통신 영역 */
```



❑SQL 통신 영역(SQLCA: SQL Communications Area)

✓C 프로그램에 내포된 SQL문에 발생하는 에러들을 사용자에게 알려줌

✓사용자는 SQLCA 데이터 구조(SQLCH.H)의 에러 필드와 상태 표시자를 검사하여 내포된 SQL문이 성공적으로 수행되었는가 또는 비정상적으로 수행되었는가를 파악할 수 있음