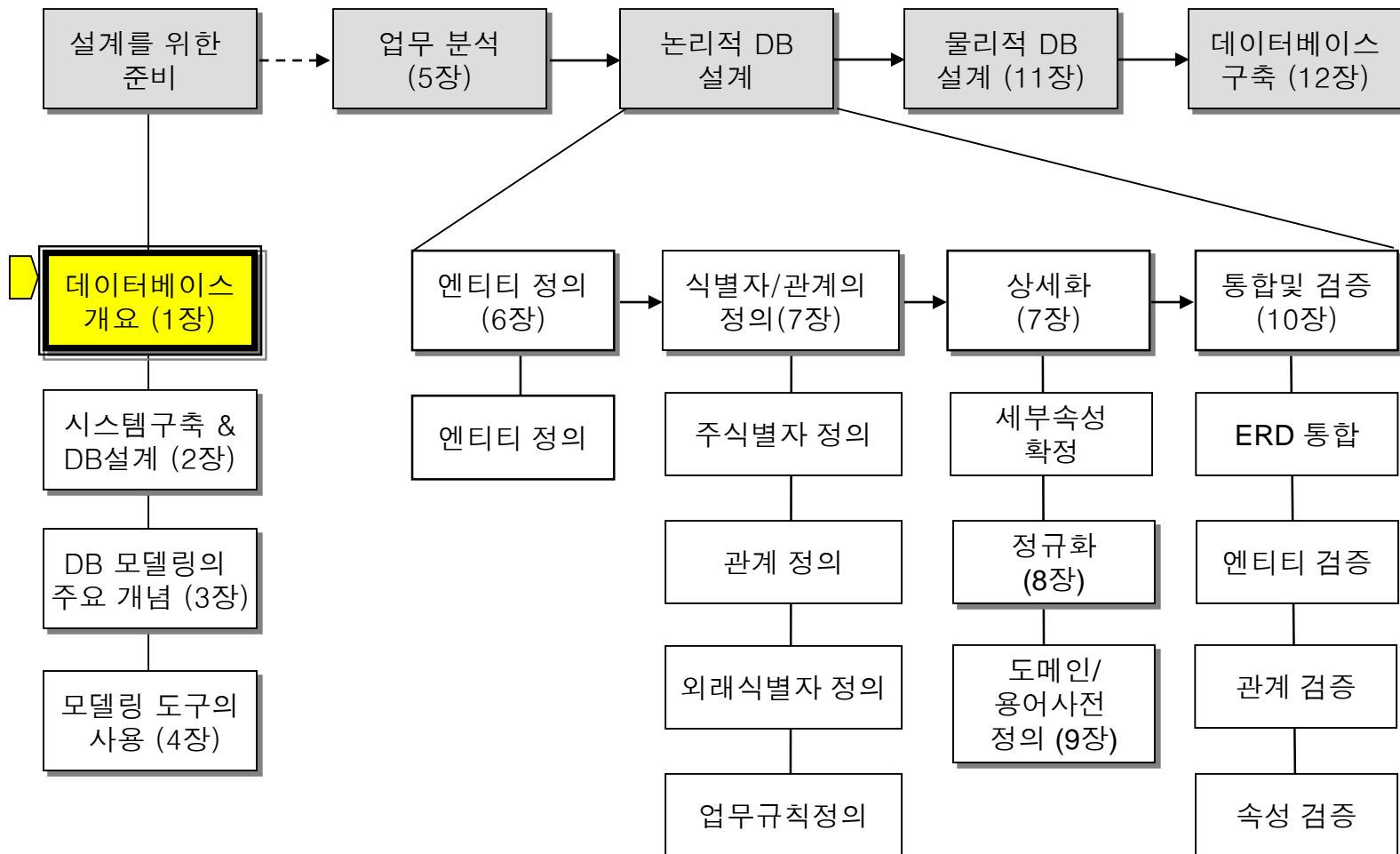




1장. 관계형 데이터베이스의 주요 개념

- 데이터베이스의 역사
- 관계형 데이터베이스 용어
- 기본키와 외래키
- 뷰
- SQL 언어



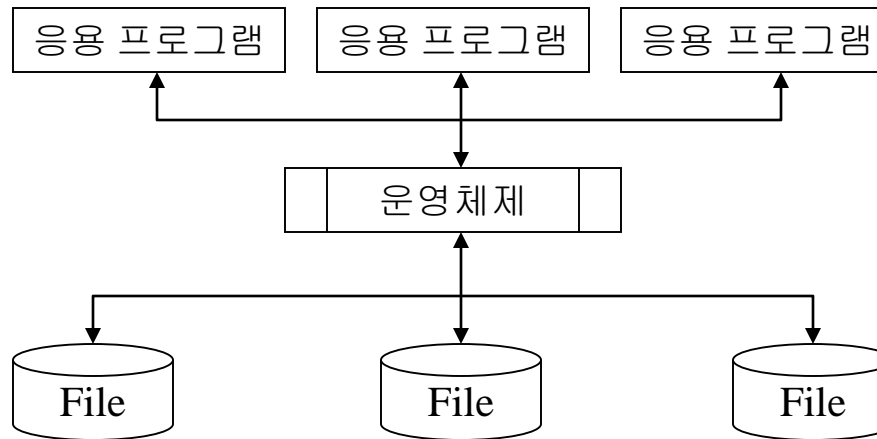
1.1 데이터베이스의 역사

□ 파일 시스템의 위기

- 컴퓨터 발전의 역사 \approx 데이터 처리의 발전사
- EDPS (Electronic Data Processing System)
 - 컴퓨터와 데이터 처리의 밀접한 관계를 보여줌
- 제 1세대 컴퓨터 시스템
 - 소프트웨어나 저장장치 등의 개발이 부족
 - 주로 기술 분야의 계산, 자료 분류 등에 사용
- 제 2세대 컴퓨터 시스템
 - 운영체제가 도입되고 FORTRAN, COBOL 등의 고급 언어 개발
 - 파일 시스템(file system)의 도입
 - 자료를 분석하고 처리하는 일에 본격적으로 사용되기 시작

1.1 데이터베이스의 역사

□ 파일 시스템의 위기



<그림 1.1> 파일 시스템에 기초한 자료처리 모델

1.1 데이터베이스의 역사

□ 파일 시스템의 위기

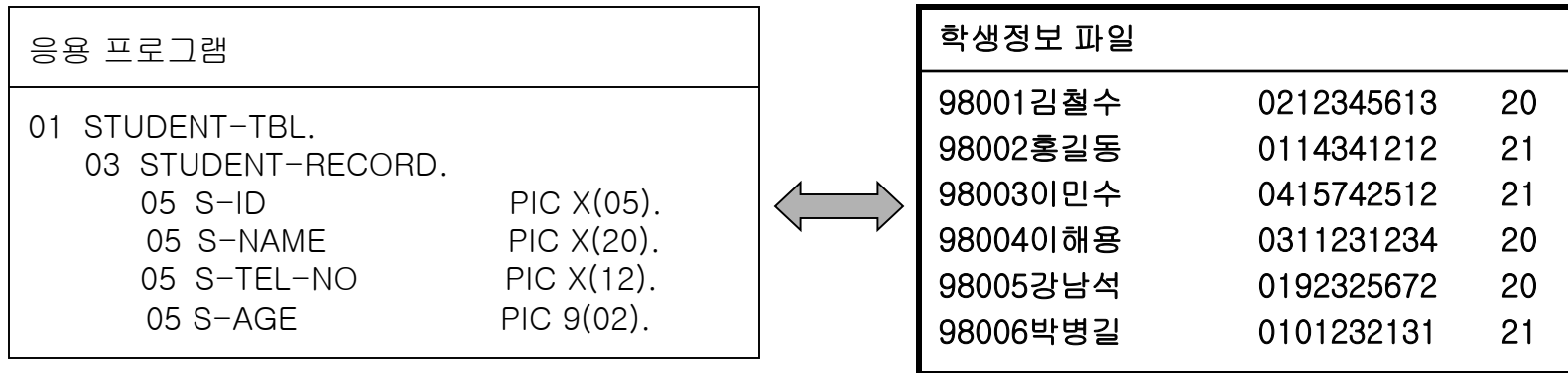
➤ 파일 시스템의 문제점

- 데이터 종속성(data dependency)

- 데이터를 사용하는 프로그램의 구조가 데이터 구조(파일 구조)의 영향을 받는다는 것을 의미
- 데이터 종속성은 프로그램의 개발과 유지 보수를 어렵게 한다

1.1 데이터베이스의 역사

□ 파일 시스템의 위기



<그림 1.2> 학생정보를 읽어 처리하는 COBOL 프로그램

- ✓ 학생의 이름을 저장하는 필드의 길이를 현재 20자리에서 30자리로 늘려야 하는 경우
- ✓ 학생정보 파일에서 필드의 순서를 바꾸어 저장하고자 하는 경우

1.1 데이터베이스의 역사

□ 파일 시스템의 위기

- 데이터 무결성(data integrity)의 침해

- 데이터 무결성이란 저장된 데이터의 내용이 본래 의도했던 데이터의 형식, 범위를 준수해야 한다는 성질
- <그림 1.2>의 학생정보 파일에서
 - » 나이(AGE) 필드는 숫자 형식이어야 하고 음수가 아닌 양수이어야 한다
 - » 나이의 범위는 20~60 사이
- 과거의 정보시스템에서는 데이터 무결성을 지켜야 할 책임이 프로그래머에게 있음
- 정보화 사회에서는 인간의 삶이 많은 부분 컴퓨터 시스템에 저장된 데이터에 의존하고 있기 때문에 데이터 무결성의 침해는 매우 심각한 문제

1.1 데이터베이스의 역사

□ 파일 시스템의 위기

- 데이터 중복성(data redundancy)

- 같은 내용의 데이터가 여러 곳에 중복하여 저장되는 것을 의미
- 과거의 정보시스템에서는 개별 부서나 응용 프로그램에서 필요로 하는 데이터 파일을 각각 만들어 사용하는 일이 많았음
- 저장 공간의 낭비 문제 발생
- 데이터의 불일치, 보안의 어려움과 같은 문제들이 발생

1.1 데이터베이스의 역사

□ 파일 시스템의 위기

- 데이터 불일치(data inconsistency)

- 중복 저장된 데이터들이 서로 일치하지 않는 것을 의미
- 예1) 우리가 이사를 하게 되면 변경된 주소를 학교나 직장, 은행, 가입된 웹사이트 등에 통보를 해야함
- 주소 정보가 여러 기관에 중복 저장되어 있다는데서 기인
- 예2) 어떤 학생이 교무과에 휴학신청을 하여 휴학을 했는데, 그 사실을 모르는 재무과에서는 등록금 고지서를 그 학생에게 발송
- 교무과의 학생 정보와 재무과의 학생 정보가 각각 관리되면서 불일치

1.1 데이터베이스의 역사

□ 파일 시스템의 위기

- 데이터 표준화(data standard)의 어려움

- 일정 규모 이상의 정보 시스템을 개발하기 위해서는 많은 수의 개발자들이 협력 작업 필요
 - » 개발자A는 응용 프로그램에서 학생이름을 'S-NAME' 으로, 길이는 20자리로 사용
 - » 개발자B는 학생이름을 'SNME' 으로, 길이는 15 자리로 사용
- 표준화가 되어 있지않으면 제 3자가 프로그램을 이해하기도 어렵고 두 응용 프로그램간의 호환성에도 문제
- 학생 이름을 지칭하고 표현하는 표준화된 규칙이 있다 하더라도 응용 프로그래머가 이를 지키지 않을 수 있음

1.1 데이터베이스의 역사

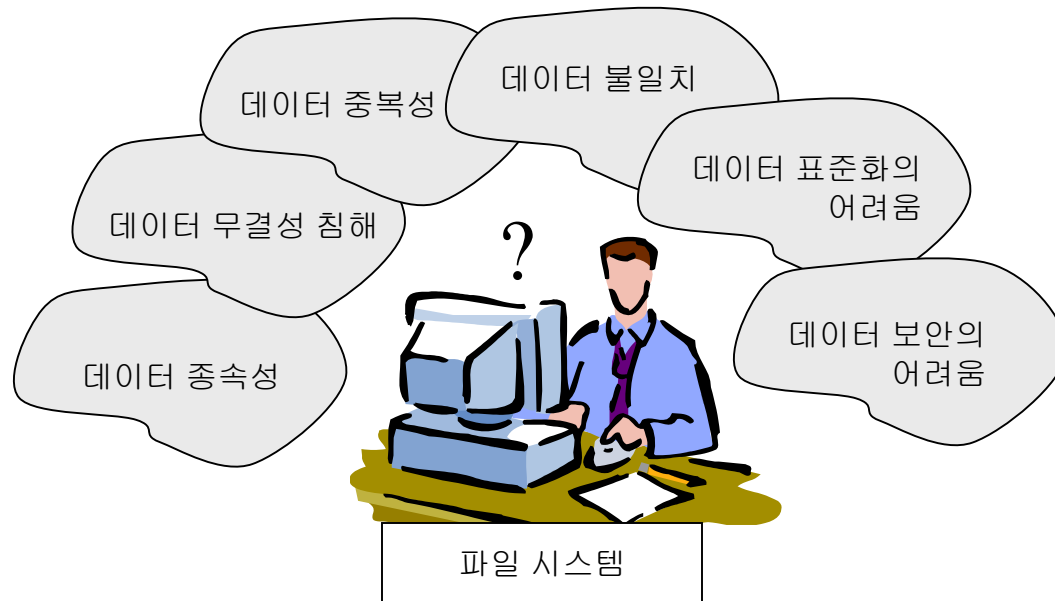
□ 파일 시스템의 위기

- 데이터 보안성(data security)의 결여

- 데이터가 저장되어 있는 파일은 그 내용이 Text 형식이나 잘 알려진 형식으로 저장되기 때문에 응용프로그램 없이도 쉽게 파일을 열어 내용을 볼 수가 있음
- 파일의 공유를 위해 접근이 쉬운 위치에 파일을 저장했기 때문에 보안을 유지하기가 어려움
- 현대의 정보시스템에는 기업의 영업 비밀이나 고객의 사생활 정보와 같은 보안을 필요로 하는 데이터가 많이 저장
- 보안성의 결여는 심각한 문제임

1.1 데이터베이스의 역사

□ 파일 시스템의 위기



<그림 1.3> 파일 시스템의 위기

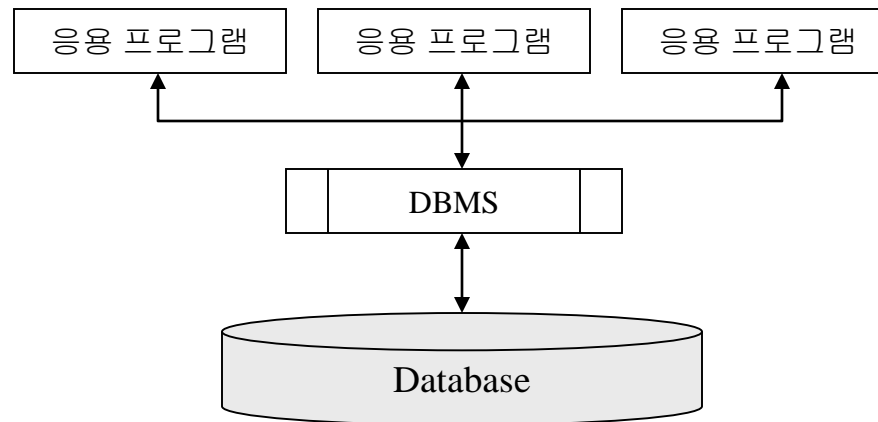
1.1 데이터베이스의 역사

□ 데이터베이스의 등장

- 파일 시스템의 단점을 극복하면서도 다수의 사용자들이 정보를 공유할 수 있어야 한다는 요구에 따라 제안됨
- 데이터베이스의 철학
 - 파일 형태로 여기저기에 흩어져 있는 데이터, 정보들을 하나로 모아 관리
 - 모아놓은 데이터들을 관리하고 사용자(응용 프로그램)와 데이터 사이에 인터페이스 역할을 할 수 있는 S/W를 제공
- 초기에는 계층형(hierarchical) 데이터베이스, 네트워크형(network) 데이터베이스가 사용되었으나 현재는 관계형(relational) 데이터베이스가 주류를 이루고 있음

1.1 데이터베이스의 역사

□ 데이터베이스의 등장



<그림 1.4> 데이터베이스 시스템의 개요

- ✓ 모아놓은 데이터의 집합 : 데이터베이스(database)
- ✓ 데이터를 관리하는 S/W : 데이터베이스 관리 시스템
(DBMS; Database Management System)

1.1 데이터베이스의 역사

□ 데이터베이스의 등장

➤ 데이터베이스의 특징

- 데이터 독립성(independency) 지원

- 사용자 혹은 응용 프로그램이 직접 데이터베이스에 접근할 수 없고 반드시 DBMS를 통해서만 접근 가능
- DBMS는 데이터베이스 내에 있는 데이터의 물리적, 논리적 변화가 응용 시스템에 영향을 미치지 않도록 함으로써 데이터 독립성을 보장

- 데이터 무결성 유지

- DBMS가 무결성을 위반하는 데이터가 들어올 경우 처리를 거절함으로써 데이터의 무결성을 지원

1.1 데이터베이스의 역사

□ 데이터베이스의 등장

- 데이터 중복성 및 불일치 최소화

- 데이터베이스 내의 데이터는 한 개인의 관점이나 특정 부서의 관점에서 관리되는 것이 아니라 데이터베이스를 공용하는 조직 전체의 관점에서 관리
- 동일 데이터가 여러 부서에서 사용하는 경우 이를 하나로 관리함으로써 중복성을 방지

- 데이터 표준화의 용이성

- 데이터베이스 관리자(DBA)가 설계과정을 주도함으로써 부서간 이해를 조정하고 관리될 데이터를 표준화시킴
- 응용 프로그램에서 데이터에 접근하기 위해서는 DBMS가 가지고 있는 구조 정보에 따라야 하기 때문에 자연스럽게 표준화됨

1.1 데이터베이스의 역사

□ 데이터베이스의 등장

- 높은 데이터 보안성

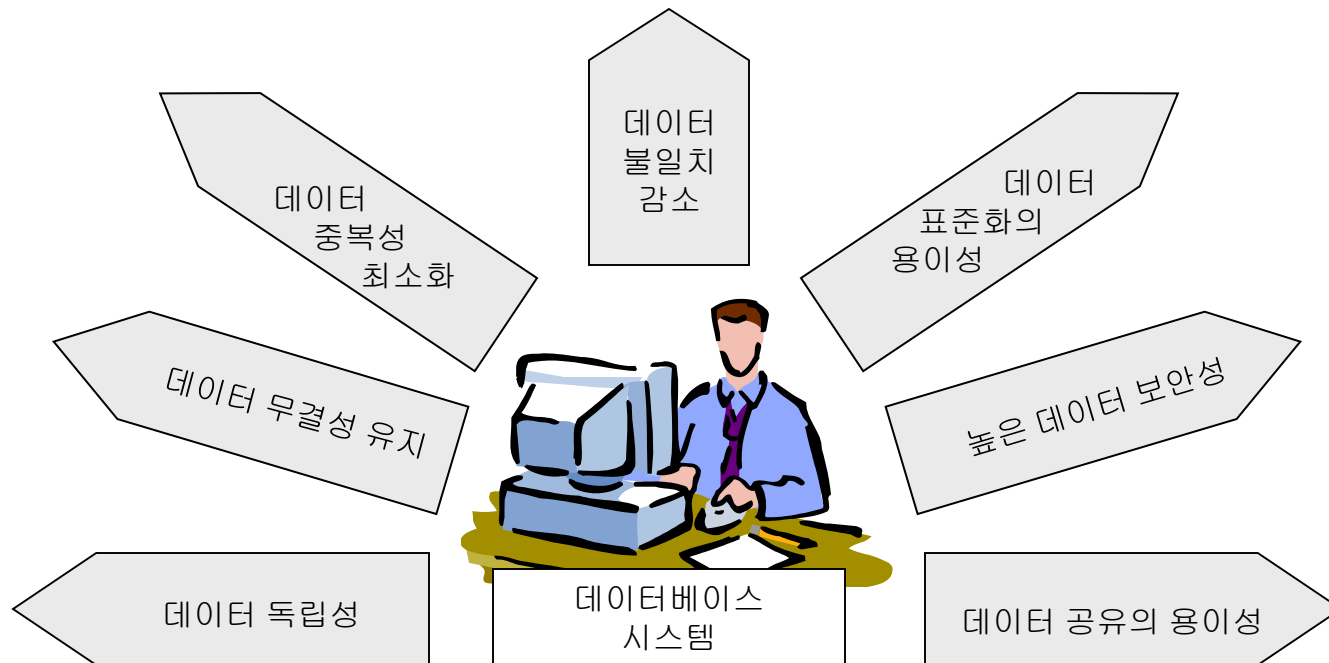
- DBMS는 사용자의 권한에 따라 데이터베이스 내에 있는 데이터에 대한 접근을 제한할 수 있음
- 저장된 데이터베이스는 일반적으로 DBMS를 통하지 않고는 외부에서 내용을 알아내기 매우 어려움

- 데이터 공유(data sharing)의 용이성

- 데이터베이스 시스템의 기본 철학이 데이터를 통합 관리하고 이를 여러 부서, 사용자들이 공유하도록 하는 것
- DBMS는 여러 사용자의 요구를 동시에 처리할 수 있는 능력을 가지고 있음
- DBMS는 데이터를 쉽게 이용할 수 있는 수단을 제공함

1.1 데이터베이스의 역사

□ 데이터베이스의 등장



<그림 1.5> 데이터베이스 시스템의 장점

1.1 데이터베이스의 역사

□ 관계형 데이터베이스 모델

- 현재 가장 많이 사용되는 데이터베이스 모델
- 데이터가 테이블 형태로 표현되며, 사용자가 데이터를 쉽게 다룰 수 있도록 해주는 질의어(SQL)를 제공
- 테이블 형태로 표현된 데이터는 단순해서 누구나 쉽게 이해할 수 있음
- SQL은 자연어에 가까운 문법을 가지고 있어서 배우기 쉽고, 데이터를 어떻게(how) 가져올 것인가 대신에 어떤(what) 데이터를 원하는지만 기술해주면 되기 때문에 사용자나 개발자의 입장에서는 데이터를 다루는 작업이 매우 단순해짐
- SQL 명령어나 문법은 표준화 되어 있기 때문에 대부분의 명령어는 모든 관계형 데이터베이스 제품에서 공통적으로 사용 가능

1.1 데이터베이스의 역사

□ 관계형 데이터베이스 모델

EMPLOYER

empno	ename	dept	tel	salary
100	김기훈	영업	1241	200
101	홍성범	기획	5621	200
102	이만수	영업	5251	250
103	강나미	생산	1231	300

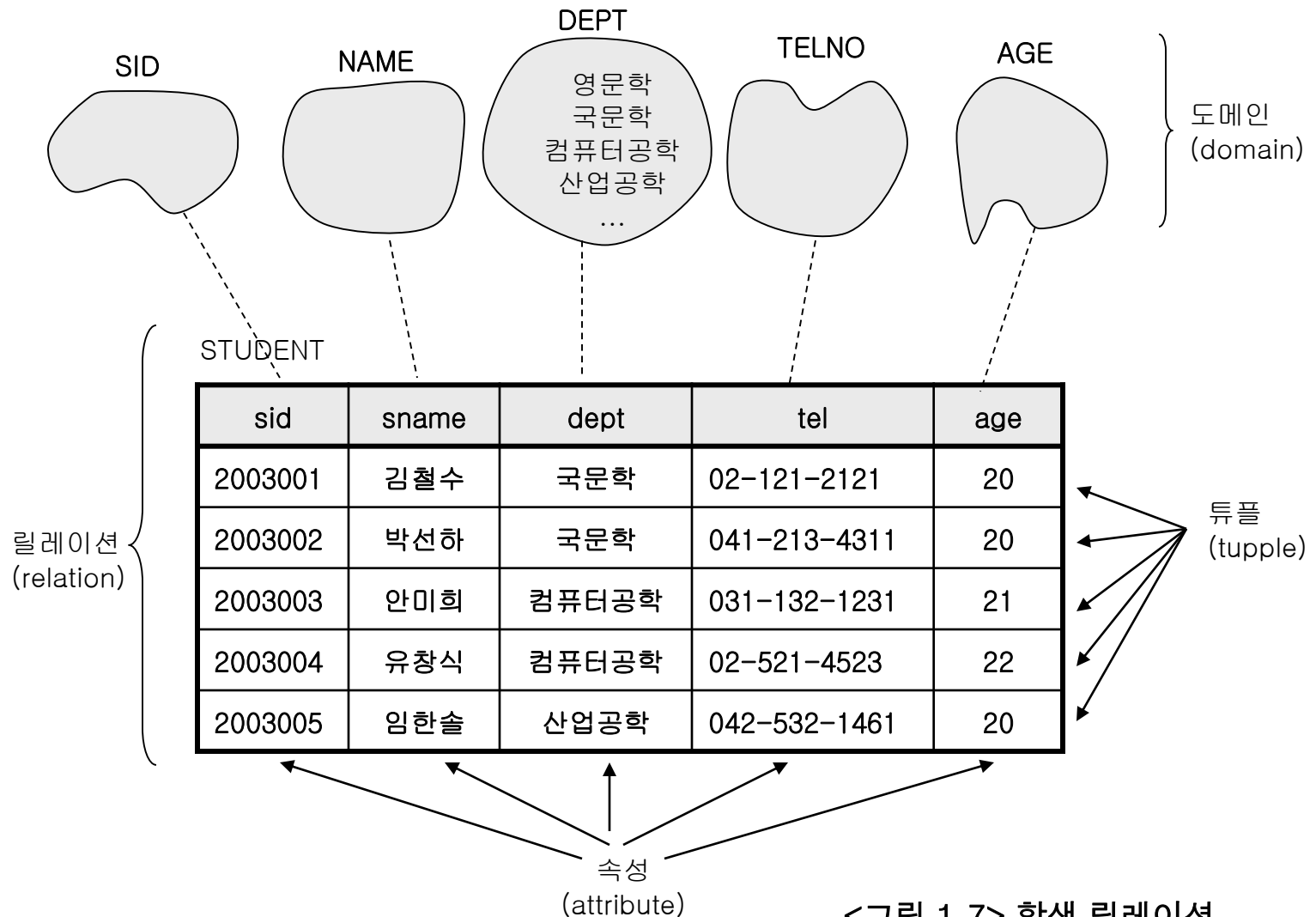
“영업부에 속한 모든 사원의 이름
과 전화 번호를 보이시오”

⇒

```
SELECT ename, tel  
FROM   employer  
WHERE  dept = '영업'
```

<그림 1.6> 사원 테이블과 질의의 예

1.2 관계형 데이터베이스 용어



<그림 1.7> 학생 릴레이션

1.2 관계형 데이터베이스 용어

※ <그림 1.7>에서 사용한 용어들은 E.F.Codd가 정의한 것으로 오늘날 일반적으로 사용하는 용어와는 차이가 있음

➤ 릴레이션(relation)

- 테이블이라는 용어로 더 많이 사용.
- 관계형 데이터베이스에서 정보를 구분하여 저장하는 기본 단위
 - STUDENT 릴레이션 : 학생에 관한 정보를 저장
 - SUBJECT라는 릴레이션 : 과목에 대한 정보를 저장
- 동일한 데이터베이스 내에서는 같은 이름을 갖는 릴레이션이 존재할 수 없다

1.2 관계형 데이터베이스 용어

➤ 속성(attribute)

- 릴레이션에서 관리하는 구체적인 정보 항목에 해당하는 것이 속성
- 현실세계의 개체(예: 학생, 교수, 과목,...)들은 많은 속성들을 갖는데 그중에서 관리해야할 필요가 있는 속성들만을 선택하여 릴레이션에 포함시킴
- 속성 역시 고유한 이름을 갖으며 동일 릴레이션 내에서는 같은 이름의 속성이 존재할 수 없음

➤ 튜플(tuple)

- 릴레이션이 현실세계의 어떤 개체를 표현한다면 튜플은 그 개체에 속한 구성원들 개개의 정보를 표현
- 예) 학생'은 개체를 나타내는 이름이고 '김철수', '박선하', '안미희',... 등은 '학생' 개체의 구성원
- 한 릴레이션에 포함된 튜플의 개수는 시간에 따라 변할 수 있으며, 한 릴레이션은 적게는 수십 개 많게는 수십만 개의 튜플을 포함할 수 있음

1.2 관계형 데이터베이스 용어

➤ 도메인(domain)

- 도메인이란 릴레이션에 포함된 각각의 속성들이 갖을 수 있는 값들의 집합
- 도메인이라는 개념이 필요한 이유는 릴레이션에 저장되는 데이터 값들이 본래 의도했던 값들만 저장되고 관리되도록 하는데 있음
- 예) '성별'이라는 속성이 있다면 이 속성이 가질 수 있는 값은 {남,여}.
 - 데이터베이스 설계자는 성별의 도메인으로 'SEX'를 정의하고 그 값으로 {남,여}를 지정한 뒤, '성별'이라는 속성은 'SEX' 도메인에 있는 값만을 갖을 수 있다고 지정해 놓으면 사용자들이 실수로 남,여 이외의 값을 입력하는 것을 DBMS가 막을 수 있음
- 현실적으로 도메인을 구현하는 것은 어렵기 때문에 대부분의 DBMS 제품에서는 **사용자 정의 데이터타입**으로 사용

1.2 관계형 데이터베이스 용어

※ 현재는 여러 용어가 혼용되고 있는 상황임

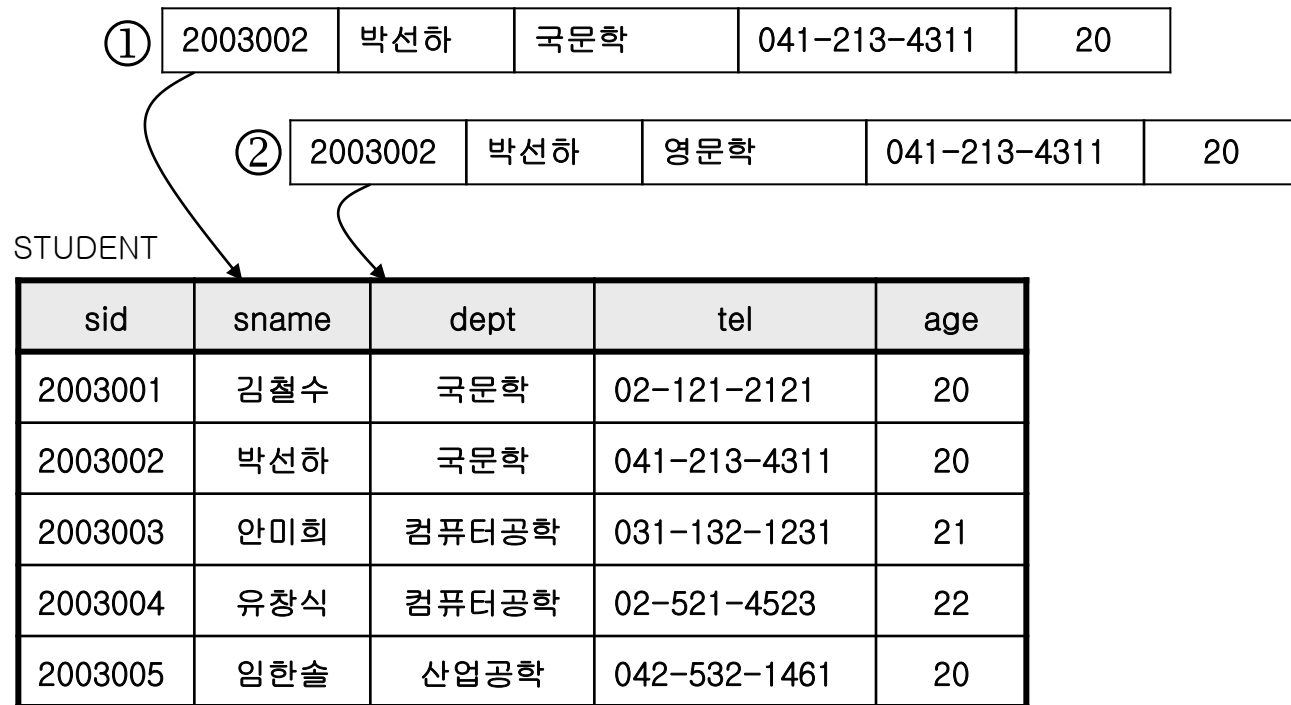
E.F.Codd 의 용어	File 시스템의 용어	자주 사용되는 용어
릴레이션(relation)	파일(file)	테이블(table)
속성(attribute)	필드(field)	열(column), 컬럼
튜플(tuple)	레코드(record)	행(row)

<표 1.2> 용어 대비표

✓ 본 강의에서는 앞으로 테이블, 컬럼, 튜플이라는 용어를 사용하기로 함

1.3 기본키와 외래키

□ 키의 필요성



<그림 1.8> 중복된 튜플의 삽입

1.3 기본키와 외래키

□ 키의 필요성

- ①번 튜플이 정상적으로 삽입된다면 동일 튜플이 이미 있으므로 중복
 - 전체 학생수는 몇명인가?
- ②번 튜플이 정상적으로 삽입된다면 물리적인 중복은 아니나 의미적으로 중복
 - ‘박선하’의 전공은 무엇인가
- 튜플의 중복 여부를 알아내기 위해서는 삽입하고자 하는 튜플과 이미 존재하는 모든 튜플을 일일이 비교해야 하는데 이는 비현실적임.
- 중복 여부를 효과적으로 알 수 있도록 하는 수단이 ‘키(key)’ 임

1.3 기본키와 외래키

□ 후보키(candidate key)

- 후보키(candidate key)란 테이블에서 각 튜플을 구별하는데 기준이 되는 하나 혹은 그 이상의 컬럼들의 집합이다. (후보키는 테이블에 있는 각 튜플을 고유하게 식별할 수 있어야 한다).
- <그림 1.8>의 STUDENT 테이블의 경우 학번(sid)이 후보키
- 튜플의 중복 여부 확인시 기존 튜플의 모든 컬럼값을 비교하는 대신 후보키 컬럼의 값만 비교

1.3 기본키와 외래키

□ 후보키(candidate key)

➤ 후보키, 기본키, 대체키

STUDENT

sid	sname	dept	pid	age
2003001	김철수	국문학	831212-1213112	20
2003002	박선하	국문학	830823-2130121	20
2003003	안미희	컴퓨터공학	820224-2013112	21
2003004	유창식	컴퓨터공학	810509-1934142	22
2003005	임한솔	산업공학	831227-1324123	20



<그림 1.9> 후보키, 기본키, 대체키

1.3 기본키와 외래키

□ 후보키(candidate key)

- 기본키(primary key)
 - 후보키 중 튜플을 식별하는데 기준으로 사용할 키
- 대체키(alternate key)
 - 후보키중 기본키로 선택되지 않은 나머지 키

기본키는 일반적으로 정보를 검색하는 기준이 된다.
기본키는 중복된 튜플이 입력 되는 것을 방지한다.
모든 테이블에는 적어도 하나의 기본키(후보키)가 존재한다

1.3 기본키와 외래키

□ 후보키(candidate key)

➤ 복합키(composite key)

- 하나의 컬럼이 후보키의 역할을 하지 못하고 두개 이상의 컬럼이 합쳐져야 후보키의 역할을 하는 경우

STUDENT_CLUB

sid	club	club_president
2003001	영어회화반	강우혁
2003001	낙시회	김민우
2003002	영어회화반	강우혁
2003003	한문강독	박문길
2003004	해커스	안홍섭

기본키

<그림 1.10> 복합키의 예

1.3 기본키와 외래키

□ 외래키(Foreign key)

- 상호 관련이 있는 테이블들 사이에서 데이터의 일관성을 보장해 주는 수단이 외래키 이다.
- 다음 슬라이드에서 사원 테이블의 부서번호(deptid)는 부서정보 테이블의 부서번호(deptid)를 참조하고 있음.
 - 부서 테이블의 첫번째 튜플이 삭제된다면?
 - 부서테이블의 100번 부서 번호가 110 으로 변경된다면?
 - 사원 테이블에 부서 번호 500인 사원이 삽입된다면?

➡ 사원테이블과 부서 테이블에 있는 데이터 사이에 불일치 발생!

1.3 기본키와 외래키

EMP

empid	ename	deptid	hire_date	job	manager	salary
1001	홍성길	100	2001.2.1	특수영업	1002	350
1002	곽희준	100	1999.1.1	영업관리	1004	400
1003	김동준	200	2000.9.1	품질관리	1005	300
1004	성재규	300	1997.2.1	급여	1009	450
1005	박성범	200	2000.2.1	수입자재	1004	320

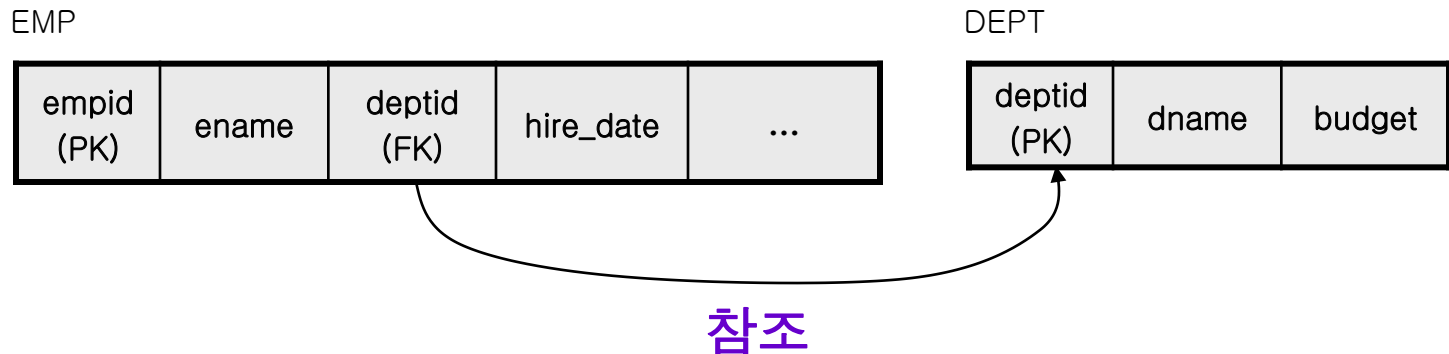
DEPT

deptid	dname	budget
100	영업부	100k
200	관리부	300k
300	구매부	220k
400	생산부	500k

<그림 1.13> 사원과 부서정보 테이블

1.3 기본키와 외래키

❑ 외래키(Foreign key)



- 두 테이블간에 외래키에 의한 참조관계에 있다면 두테이블간 데이터 불일치가 발생하는 상황이 되면 DBMS는 다음과 같은 조치를 취할 수 있다
 - 제한(restrict), 연쇄(cascade), 널값으로 대체(nullify)

1.3 기본키와 외래키

□ 외래키(Foreign key)

부서 테이블의 첫번째 튜플을 삭제하려 할 때

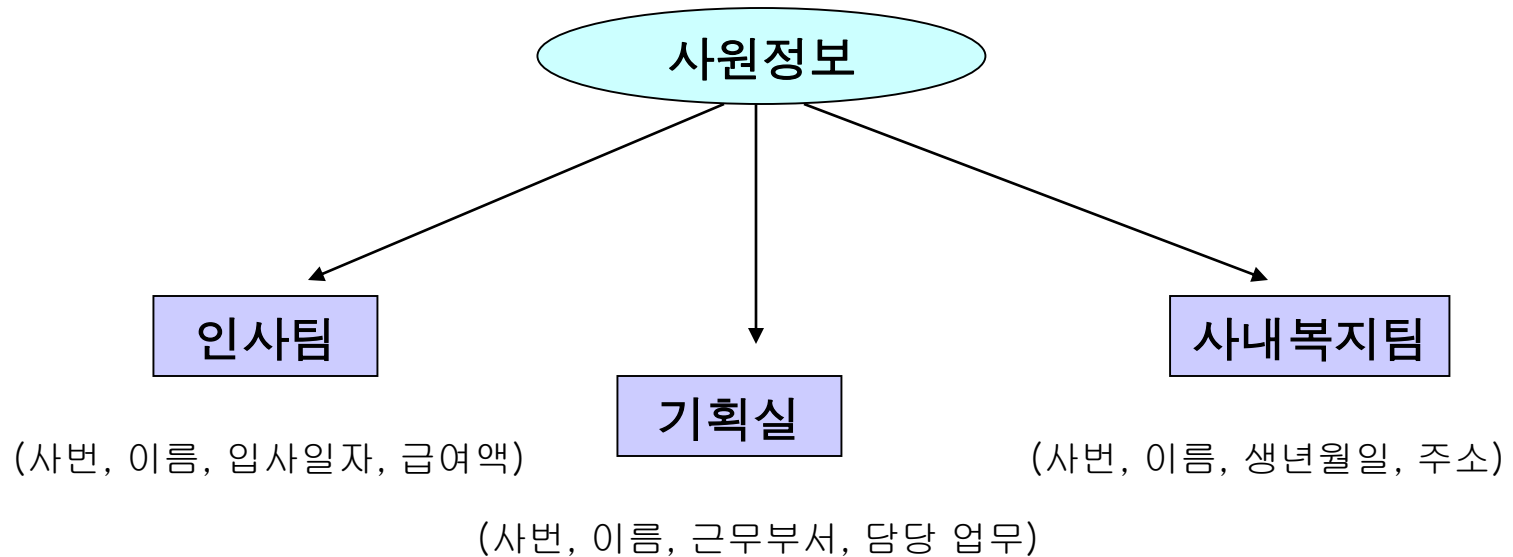
- 제한(restrict)
 - 삭제하려는 튜플의 부서번호 값을 사원 테이블에서 가지고 있는 튜플이 있으므로 삭제 연산을 거절
- 연쇄(cascade)
 - 삭제된 부서번호 값을 갖는 사원 테이블의 튜플도 함께 삭제
- 널값으로 대체(nullify)
 - 삭제연산을 수행한 뒤 삭제된 부서번호 값을 갖는 사원 테이블의 튜플에서 부서번호를 null 값으로 대체

* 외래키를 통해 두 테이블간의 데이터 무결성을 유지하는 것을 '참조 무결성 제약조건'이라고 한다.

1.4 뷰(view)

□ 뷰의 필요성

- 하나의 테이블, 혹은 여러 테이블에 대하여 특정 사용자나 조직의 관점에서 데이터를 바라볼 수 있도록 해주는 수단



1.4 뷰(view)

□ 뷰의 필요성

EMP

empid	ename	dept	hire_date	birthday	Address	job	salary
1001	홍성길	영업부	2001.2.1	1985.10.12	서울 대림동	특수영업	350
1002	곽희준	영업부	1999.1.1	1984.9.10	안양 용봉동	영업관리	400
1003	김동준	생산부	2000.9.1	1986.5.16	부산 대하동	품질관리	300
1004	성재규	인사부	1997.2.1	1982.4.10	대구 달성동	급여	450
1005	박성범	구매부	2000.2.1	1986.12.4	광주 금남동	수입자재	320

<그림 1.16> 전체 조직 관점에서의 사원 테이블

1.4 뷰(view)

VIEW_EMP1

empid	ename	hire_date	salary
1001	홍성길	2001.2.1	350
1002	곽희준	1999.1.1	400
1003	김동준	2000.9.1	300
1004	성재규	1997.2.1	450
1005	박성범	2000.2.1	320

VIEW_EMP2

empid	ename	dept	job
1001	홍성길	영업부	특수영업
1002	곽희준	영업부	영업관리
1003	김동준	생산부	품질관리
1004	성재규	인사부	급여
1005	박성범	구매부	수입자재

VIEW_EMP3

empid	ename	birthday	Address
1001	홍성길	1985.10.12	서울 대림동
1002	곽희준	1984.9.10	안양 용봉동
1003	김동준	1986.5.16	부산 대하동
1004	성재규	1982.4.10	대구 달성동
1005	박성범	1986.12.4	광주 금남동

<그림 1.17> 사원 테이블에 대한 세가지 뷰

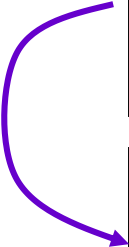
1.4 뷰(view)

□ 뷰의 정의

```
CREATE VIEW high_salary  
AS SELECT empid, ename, dept, salary  
FROM emp  
WHERE salary >= 350;
```

- 뷰에 대한 질의는 뷰가 정의된 기본 테이블에 대한 질의로 바뀌어 실행된다

```
SELECT empid, ename, salary  
FROM high_salary  
WHERE dept = '영업부';
```



```
SELECT empid, ename, salary  
FROM emp  
WHERE salary >= 350  
AND dept = '영업부';
```

1.4 뷰(view)

□ 뷰를 사용하는 경우

- <그림 1.17>의 경우와 같이 하나의 테이블에 대하여 여러 부서에서 서로 다른 관점으로 보기를 원할 때
- 테이블에 급여와 같이 일반 사용자에게는 감추어야 할 컬럼이 있을 때 그것을 제외하고 뷰를 만들어 제공함으로써 보안을 유지.
- 자주 사용하는 복잡한 질의문을 미리 뷰로 정의하여 두고 간편하게 쓰고자 할 때

- 뷰는 물리적인 데이터를 갖지 않는다
- 뷰에 대한 갱신 연산은 경우에 따라 실행될수도 있고 실행되지 않을수도 있다

1.5 SQL 언어

□ SQL 개요

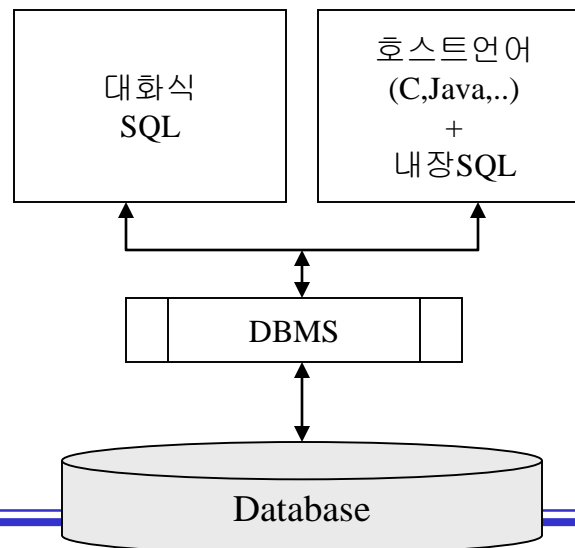
- 관계형 데이터베이스의 장점중의 하나는 사용자들이 쉽게 사용할 수 있는 SQL(Structured Query Language) 언어를 제공한다는 것
- 사용자는 간단한 SQL문을 사용하여 DBMS에게 작업을 요청
- 기본적인 SQL문들은 표준화되어 있기 때문에 거의 대부분의 DBMS 제품에 동일하게 적용
- SQL은 1974년 IBM 연구소에서 System R 프로젝트를 통해 개발되었고 1986년 미국 표준기구 ANSI에서 SQL 표준을 제정
- SQL 언어는 비절차적 언어(non-procedural language)
 - 사용자는 자신이 원하는 바만을 명시하며, 원하는 바를 DBMS가 어떻게 처리할지는 명시할 필요가 없음

1.5 SQL 언어

□ SQL 개요

➤ SQL의 두가지 사용 방식

- 대화식 SQL : DBMS 회사에서 제공하는 유틸리티 프로그램(예: ORACLE의 SQL*Plus)을 이용하여 사용자가 직접 SQL문을 입력하고 실행결과를 확인하는 방식
- 내장 SQL : SQL문이 C, Java와 같은 프로그램 안에 포함되어져서 사용되는 방식



<그림 1.19> SQL을 이용하는 두가지 방법

1.5 SQL 언어

❑ SELECT

급여가 300을 넘고 담당업무가 영업관리인 사원의 모든 정보를 보이시오

```
SELECT *  
FROM emp  
WHERE salary > 300 AND job = '영업관리' ;
```

부서 번호와 그 부서에 속한 사원들의 급여액 합계를 보이시오

```
SELECT deptid, SUM(salary)  
FROM emp  
GROUP BY deptid ;
```

1.5 SQL 언어

□ SELECT

모든 사원들의 이름과, 부서이름을 보이시오

```
SELECT emp.ename, dept.name  
FROM   emp, dept  
WHERE  emp.deptid = dept.deptid ;
```

곽희준 사원이 속한 부서의 예산은 얼마인가

```
SELECT d.name  
FROM   emp e, dept d  
WHERE  e.deptid = d.deptid  
AND    e.ename = '곽희준';
```

1.5 SQL 언어

❑ INSERT

```
INSERT INTO emp (empid, ename, deptid, hire_date, job, salary)  
VALUES (106, '강윤희', 200, '2001-01-10', '연말정산', 400);
```

```
INSERT INTO emp (empid, ename, salary)  
VALUES (107, '남진선', 500);
```

일부 컬럼을 생략하고 튜플을 삽입하는 경우, 생략된 컬럼들의 값은 null로 저장된다

1.5 SQL 언어

□ UPDATE

홍성길 사원의 부서번호를 400으로, 급여를 500으로 변경하시오

```
UPDATE emp
SET    deptid=400, salary=500
WHERE  ename='홍성길' ;
```

영업부에 속한 사원의 급여를 20% 인상하시오

```
UPDATE emp
SET    salary = salary*1.2
WHERE  deptid = (SELECT deptid
                  FROM dept
                  WHERE dname = '영업부') ;
```

1.5 SQL 언어

❑ DELETE

홍성길 사원의 정보를 사원정보에서 삭제하시오

```
DELETE FROM emp  
WHERE  ename='홍성길' ;
```

모든 사원의 정보를 사원정보에서 삭제하시오

```
DELETE FROM emp ;
```

1.5 SQL 언어

□ CREATE

```
CREATE TABLE emp  
( deptid int(10) NOT NULL,  
  dname char(20),  
  budget char(5),  
  PRIMARY KEY(deptid)  
);
```

```
CREATE TABLE emp  
( empid int(10) NOT NULL,  
  ename char(20),  
  deptid int(5),  
  hire_date date,  
  job char(20),  
  salary int(10) NOT NULL,  
  PRIMARY KEY(empid),  
  FOREIGN KEY (deptid) REFERENCES dept(deptid)) ;
```