제 1 장. 데이타베이스 시스템 개요

- 데이타베이스란?
- 파일 시스템과 비교
- DBMS의 장점
- DBMS의 구조

데이타베이스란?

- 하나 이상의 서로 관련된 데이타들의 모임
- 실세계의 조직체를 모델링
  - 개체 (예, 학생, 교수, 과목, 강의실)
  - 개체들간의 관계
    (예, 학생 A가 CS233 과목을 등록,
    교수 B가 CS233 과목을 강의)
- 데이타베이스의 크기와 복잡도는 가변
  - 이름과 주소의 list : 수백 record의 간단한 구조
  - 큰 도서관의 도서 자료 : 거의 백만 card 구조
  - 미국의 재무성 자료 : 4×10^{11} byte의 데이타베이스

데이터베이스 개념

- 데이터 베이스 정의
  - 서로 관련된 데이터들의 집합체
  - 하나의 조직에 있는 여러 응용 시스템들이 서로 공용할 수 있도록 통합, 저장된 실제의 운영 데이터들의 집합
- 정의의 의미
  - 통합(integrated) 데이터 : 같은 데이터가 원칙적으로 중복되어 있지 않다
  - 저장(stored) 데이터 : 저장 매체에 저장
  - 운영(operational) 데이터 : 목적이나 기능을 수행하는데 필요한 데이터 집합
  - 공용(shared) 데이터 : 여러 사용자들이 다른 목적으로 공동으로 이용

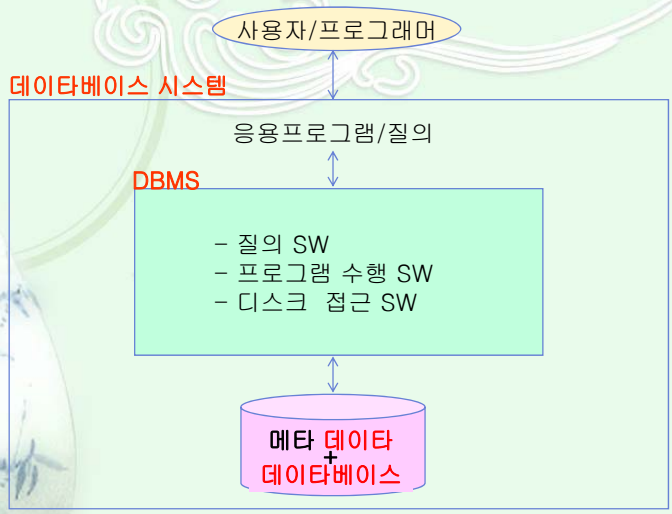
DBMS 란?

- DataBase Management System : 데이타베이스 관리 시스템
- 데이타베이스를 유지 관리하고 이용하는데 도움되도록 설계된 소프트웨어 시스템
  - database + 응용 프로그램
- 사용자에게 편리하고 효율적인 환경 제공
- 대부분의 전산학 부분을 포함함
  - 언어, 프로그래밍 개념, 컴파일러, OS, 자료구조, 알고리즘, 인공지능, 사용자 인터페이스 등

데이터베이스 시스템

- 데이타 (data)
  - 의미를 가지면서 기록될 수 있는 알려진 사실
- 데이타베이스 (database)
  - 관련있는 데이타의 모임
- 데이타베이스 관리 시스템 (DBMS)
  - 데이터베이스의 생성과 관리를 담당하는 소프트웨어 패키지
- 데이타베이스 시스템 (Database System)
  - Database와 그를 관리하는 소프트웨어 (DBMS, 응용 프로그램) 모두를 칭하는 용어

데이터베이스 시스템



데이터베이스 응용

- 멀티미디어 데이터베이스
- 대화형 비디오(interactive video)
- 전자 도서관(digital library)
- 인간 유전자 지도작성(human genome mapping)
- NASA 의 지구관측 시스템
- 데이터 마이닝(data mining)

DBMS 필요성 이해

- 어떤 대학이 교무, 학생, 장학, 수업 등에 관한 500GB의 대용량 자료 관리
  - 여러 직원 및 학생에 의해 동시에 접근
  - 데이터에 관한 질문은 신속히 답변
  - 여러 사용자에 의한 데이터 변경은 일관성있게 적용
  - 데이터의 어떤 부분(예, 성적, 급여)은 접근 제한

파일시스템의 문제점

- 500GB의 자료를 주기억장치에 저장 불가
  - 디스크 같은 저장장치에 저장
  - 필요시 관련된 부분 주기억장치로 가져 옴
- 32bit 컴퓨터에서는 주기억장치의 용량이 500GB 라도 4GB 이상 참조할 수 없음

파일시스템의 문제점

- 데이터 종속성(data dependency)
  - 데이터를 사용하는 프로그램의 구조가 파일 구조에 영향을 받음
  - 예: 학생 정보 레코드 검색
    - 학생의 이름 필드의 크기를 20자리에서 30자리로 늘릴 경우
    - 모든 응용 프로그램에서 학생정보 처리하는 곳 변경해야 함
    - 만약 한 프로그램에서 변경하지 않았다면, 문제 발생
  - 응용 프로그램이 데이터에 종속되어 있음

파일시스템의 문제점

- 데이터 무결성(data integrity)의 침해
  - 저장된 데이터의 내용이 본래 의도했던 데이터의 형식, 범위를 준수해야함
  - 예: 학생 정보 파일에서 나이(age) 필드
    - 숫자 형식, 양수이면서 범위는 20~65 사이가 일반적
    - 조건에 위배되는 데이터가 저장될 때, 무결성 침해됨
    - 즉, 응용 프로그램에서 데이터 무결성 검증해야 함
  - 사용자의 실수로 무결성이 침해되는 경우 발생

파일시스템의 문제점

- 데이터 중복성(data redundancy)
  - 사용자 부서별 다른 유형의 특별한 프로그램 필요
  - 예:
    - 교무과 : 수업 및 성적 관리 프로그램
    - 학생과 : 장학 관리 프로그램
  - 각 부서별 별도의 데이터를 저장
  - 동일 데이터의 중복 저장이 발생함, 저장 공간 낭비
  - 데이터 불일치 및 보안의 어려움 같은 문제 발생

교무과	학번	이름	학과	주소	전화번호	과목1	과목2	…
학생과	학번	이름	학과	주소	전화번호	상벌	장학금	…

파일시스템의 문제점

- 데이터 불일치(data inconsistency)
  - 여러 사용자들이 중복 저장된 데이터를 사용할 시, 불일치 발생 가능성 존재
  - 예, 한 학생이 교무과에 휴학신청으로 인한 학적 변동
    - 학생과에 통보하지 않음으로 인해 등록금 고지서 발송
    - 교무과와 학생과의 학생정보를 따로 관리함으로 인해 불일치 발생

교무과	22013	홍길동	전산학과	부산시남구	987-1234	과목1	과목2	…

학생과	22013	홍길동	전산학과	부산시중구	987-1234	상벌	장학금	…

  - 한 부서에서 변경 후, 중복된 횟수만큼 반복해서 변경
  - 응용 프로그램에서 이런 세부적인 프로그램을 처리하려면, 프로그램의 복잡성 증가

파일시스템의 문제점

- 데이터 보안성(data security)의 결여
  - 운영체제는 보안기법으로 password 식별기능만 제공
  - 파일은 text 형식으로 저장되어 쉽게 내용을 볼 수 있음
  - 사용자들 마다 데이터의 각기 다른 부분 (예, 특정한 레코드의 필드)에 접근할 수 있도록 하는 보안정책을 집행하기에는 융통성 부족

DBMS의 장점

- 데이터 독립성(independency) 지원
  - 응용프로그램은 데이터 표현 및 저장에 대해 독립
    - 사용자 혹은 응용 프로그램이 직접 데이터베이스에 접근할 수 없고 DBMS를 통해서만 접근 가능
  - DBMS는 이러한 세부사항을 은닉하는 데이터의 추상적 관점 제공함으로써 데이터 독립성 보장
- 효율적인 데이터 접근
  - DBMS는 데이터를 효율적으로 저장하고 검색하기 위해 여러 정교한 기술을 이용

DBMS의 장점

- 데이터 무결성 유지
  - 무결성 제약조건을 집행
  - DBMS는 데이터베이스 내에 저장될 데이터에 대한 정보를 가지고 있으며, 이를 위반하는 데이터가 들어올 경우 처리를 거절함으로써 데이터 무결성을 지원
  - 예)
    - 키 제약조건, 외래키 제약조건 등

DBMS의 장점

- 데이터 중복성 및 불일치 최소화
  - 여러 사용자가 데이터를 공유할 때, 데이터 관리를 중앙집중화 하는 것이 상당한 개선을 야기
  - 중복성 최소, 효율적 검색
  - 중복된 데이터간 간의 불일치 문제 해결
  - 일관성있는 데이터 관리 가능
- 높은 데이터 보안성
  - 사용자의 권한에 따라 데이터에 대한 접근을 제한

DBMS의 장점

- 데이터 공유(data sharing)의 용이성
  - 한 데이터에 대한 동시접근 시, 한번에 한 사용자만 사용하는 것처럼 여러 사용자 동시 처리
  - 시스템의 붕괴로 인한 영향으로부터 사용자들을 보호
- 응용 개발시간 감축
  - 많은 응용에 공통인 중요한 기능 지원
  - 필요한 tool 제공

데이터 모델

- data model : data, data 관계, data 의미 및 제약 조건들을 기술하기위한 개념들의 집합
- 관계 데이터 모델(relational data model)
  - 대부분의 DBMS에서 사용
- 의미적 데이터 모델(semantic data model)
  - 더 추상적이고 고 수준의 데이터 모델
  - 데이터의 초기 구성 시 좋음
  - 개체-관계(entity-relationship: ER) 모델

관계 모델

- 릴레이션(relation: 관계) : record들의 집합
  - 행과 열로 구성된 일종의 table
  - 각 relation 마다 소속 열(field)들을 기술하는 schema 존재
- 스키마(schema) : 데이터베이스의 논리적 구조
  - relation 이름, 각 필드(attribute 또는 column), 그리고 각 필드의 type
- 인스턴스(instance)
  - 특별한 시점에서의 database의 실제 내용

schema와 instance

- 예) 학생 정보에 대한 schema

  Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)

- 예) Students relation에 대한 instance
  - 각 행은 학생 한명을 표시

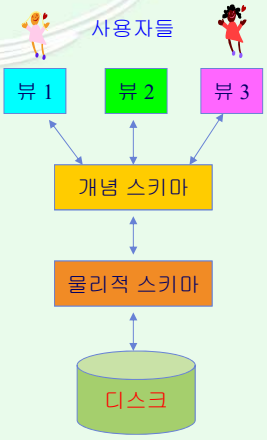
sid	*name*	*login*	*age*	*gpa*
53666	Jones	joens@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

기타 데이터 모델

- 계층(hierarchical) 모델
  - IBM의 IMS
- 네트워크(network) 모델
  - IDS, IDMS
- 객체지향(objected-oriented) 모델
  - ObjectStore, Versant
- 객체-관계(object-relational) 모델
  - IBM, Informix, Oracle 등

Data 추상화 3단계

- 내부단계
  - 데이터베이스가 실제로 저장되는 방법의 표현
  - 하나의 **물리적 스키마(physical schema)**
  - 사용파일과 인덱스 기술 등 포함
- 개념단계
  - 전체 데이터베이스의 정의를 의미
  - 하나의 **개념(conceptual) 스키마**
  - 데이터와 관계, 제약사항, 무결성에 대한 내용 포함
- 외부단계
  - 일반 사용자나 프로그래머가 접근하는 계층
  - 여러 개의 **외부(external) 스키마**
  - 뷰의 개념
  - 사용자마다 다른 뷰



예: 대학 데이터베이스

- 개념 schema(논리적 schema) : relation에 대한 명세
  - 모든 relation 들 기술
  - 개체에 관한 정보와 관계에 관한 정보 포함

Students(*sid*: string, *name*: string, *login*: string,
age: integer, *gpa*: real)
Faculty(*fid*: string, *fname*: string, *sal*: real)
Cources(*cid*: string, *cname*: string, *credits*: integer)
Rooms(*rno*: integer, *address*: string, *capacity*: integer)
Enrolled(*sid*: string, *cid*: string, *grade*: string)
Teaches(*fid*: string, *cid*: string)
Meets_In(*cid*: string, *rno*: integer, *time*: string)

  - 개념적 데이터베이스 설계

예: 대학 데이터베이스

- 물리적 schema : 상세한 저장내역 기술
  - relation들은 비정렬 레코드 파일구조로 저장
  - Students, Faculty, Cources의 첫 필드에 인덱스 구성
  - Faculty의 *sal* 필드, Rooms의 *capacity* 필드에 인덱스
  - 물리적 데이터베이스 설계
- 외부 schema (view; 뷰) : 특정 사용자에 대한 맞춤 형태로 여러 개 존재
  - 일반 사용자의 요구대로

Courceinfo(*cid*: string, *fname*: string,
enrollment: integer)

데이터 독립성(independence)

- 응용 프로그램은 데이터의 구성 및 저장 방식의 변화로부터 독립됨
- 어떤 부분의 변화가 다른 것에 심각한 영향을 미치지 않게 정의 되어야 함(module화)
- 논리적 데이터 독립성
- 물리적 데이터 독립성

데이터 독립성(independence)

- 논리적 데이터 독립성
  - 외부 단계와 개념 단계 사이의 독립성
  - 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록 지원
  - 논리적 구조가 변경되어도 응용 프로그램에 영향 없음
  - 논리적 level 수정은 데이터베이스의 **논리적 구조 변경 시** 필요

데이터 독립성(independence)

- 물리적 데이터 독립성
  - 개념 단계와 내부 단계 사이의 독립성
  - 내부 스키마가 변경되어도 개념 스키마에는 영향을 미치지 않도록 지원
  - 물리적 저장장치를 재구성할 경우 개념 스키마나 응용 프로그램에 영향 없음
  - 물리적 level 수정은 **성능 개량위해** 필요

DBMS의 질의

- 질의의 예
  - 학번이 123456인 학생의 이름은?
  - 얼마나 많은 학생들이 CS564 과목을 수강하는가?
  - CS564 과목을 수강하는 학생 중 3.0이하의 평균평점을 받은 사람은?
- 질의(query)
  - DBMS에 저장되어있는 데이터에 관한 질문
  - 질의어(query language)라는 특수한 언어 사용
  - 데이터 조작어(DML)를 통하여 데이터를 생성, 수정, 질의함

데이터베이스 언어

- 데이터 정의어(DDL:Data Definition Language)
  - 데이터베이스 schema를 정의하기 위한 도구
  - 저장구조와 access 방법 등을 정의
  - 예) CREATE, ALTER, DROP 등
- 데이터조작어(DML:Data Manipulation Language)
  - 데이터를 조작하고 처리하기위한 언어
  - 예) SELECT, INSERT, UPDATE 등
- 데이터 제어어(DCL:Data Control Language)
  - 데이터에 대한 접근 권한 부여 등을 관리
  - 예) GRANT, REVOKE 등

트랜잭션(transaction) 관리

- transaction : database 응용에서 하나의 논리적 함수를 수행하는 동작들의 집합

- 은행 예 : 계좌를 access and update (두 연산)
  - read(X) : database로부터 buffer로 data X 전송
  - write(X) : buffer로부터 database로 data X 전송
  - 예) 계정 A로부터 B로 $50 송금

T_1
Read(A);
A := A – 50;
Write(A);
Read(B);
B := B + 50;
Write(B);

	A	B
수행 전	1000	2000
수행 후	950	2050

트랜잭션 개념

- atomicity(원자성)
  - T_1의 수행전에 A=1000, B=2000 가정
  - T_1의 수행동안 failure 발생
    - A=950, B=2000; A+B가 보존되지 않음
    - 비정상적인 수행
  - 부분적으로 수행되는 transaction은 없음
  - transaction은 완전히 수행되거나(all), 전혀 수행되지 않거나(nothing) 하여야 함
    - all : A = 950, B = 2050
    - nothing : A = 1000, B = 2000

T_1
Read(A);
A := A – 50;
Write(A);
Read(B);
B := B + 50;
Write(B);

	A	B
all	950	2050
nothing	1000	2000

- consistency(일관성)
  - 트랜잭션의 수행에 의해 A+B의 값이 변화되지 않음

동시 수행

- 다중 트랜잭션 허용
- 동시수행(concurrency) 허용 이유
  - 한 트랜잭션은 여러 단계 포함(I/O 연산, CPU 연산)
    - I/O와 CPU는 병렬로 동작 가능
    - 따라서, 동시에 여러 트랜잭션을 병렬로 수행 가능
    - 효율 증가
  - 긴 트랜잭션과 짧은 트랜잭션이 동시에 수행 가능
    - 순서대로 수행경우, 짧은 트랜잭션은 긴 트랜잭션 완료동안 쓸데없이 많이 기다림
    - 동시에 수행하는 것이 좋음
    - 평균 응답시간(response time) 감소
- 일관성 유지하면서 동시수행 가능(concurrency control)

Transaction 동시 수행

- 원칙적으로 순서대로 처리되어야 함
- 은행 예 : 두 transaction
  - T_1 : 계정 A로부터 B로 $50 전송
  - T_2 : 계정 A의 잔고중 10%를 B로 전송
  - A, B의 현재 잔고 : 1000, 2000

	A	B
수행 전	1000	2000
수행 후	950	2050

T_1

```
Read(A);
A := A – 50;
Write(A);
Read(B);
B := B + 50;
Write(B);
```

T_2

```
Read(A);
tmp := A * 0.1;
A := A – tmp;
Write(A);
Read(B);
B := B + tmp;
Write(B);
```

	A	B
수행 전	1000	2000
수행 후	900	2100

직렬 스케줄

- 직렬(sereial) 스케줄
  - 트랜잭션들이 시작부터 끝까지 하나씩 순서적으로 실행되는 스케줄
- 스케줄 S1 : < T_1 ; T_2 > T_1 수행 후에, T_2 수행

T_1	T_2
Read(A); *A := A – 50;* *Write(A);* *Read(B);* *B := B + 50;* *Write(B);*	*Read(A);* *tmp := A * 0.1;* *A := A – tmp;* *Write(A);* *Read(B);* *B := B + tmp;* *Write(B);*

T_1	A	B
수행 전	1000	2000
수행 후	950	2050

T_2	A	B
수행 전	950	2050
수행 후	855	2145

- 수행 후
  - A=855, B=2145
  - A+B 가 보존됨

직렬 스케줄

- 스케줄 S2 : < T_2 ; T_1 > T_2 수행 후에, T_1 수행

T_1	T_2
Read(A); *A := A – 50;* *Write(A);* *Read(B);* *B := B + 50;* *Write(B);*	*Read(A);* *tmp := A * 0.1;* *A := A – tmp;* *Write(A);* *Read(B);* *B := B + tmp;* *Write(B);*

- 수행 후
  - A=850, B=2150 ; A+B 가 보존됨
- 동시수행시 직렬일 필요 없음

동시 수행 – 예

- 스케줄 S3 : T_1 , T_2 가 번갈아 가면서 수행

T_1	T_2
Read(A);	
A := A – 50;	
Write(A);	
	Read(A);
	*tmp := A * 0.1;*
	A := A – tmp;
	Write(A);
Read(B);	
B := B + 50;	
Write(B);	
	Read(B);
	B := B + tmp;
	Write(B);

T_1	A	B
수행 전	1000	2000
수행 후	950	

T_2	A	B
수행 전	950	2000
수행 후	855	

T_1	A	B
수행 전	855	2000
수행 후		2050

T_2	A	B
수행 전	855	2050
수행 후		2145

- 수행 후
  - 스케줄 S1과 같은 결과
  - A(855)+B(2145) 가 보존됨

동시 수행 – 예

- 스케줄 S5 : 모든 동시 수행이 올바른 것은 아님

T_1	T_2
Read(A);	
A := A – 50;	
	Read(A);
	*tmp := A * 0.1;*
	A := A – tmp;
	Write(A);
	Read(B);
Write(A);	
Read(B);	
B := B + 50;	
Write(B);	
	B := B + tmp;
	Write(B);

T_1	A	B
수행 전	1000	2000
수행 후		

T_2	A	B
수행 전	1000	2000
수행 후	900	

T_1	A	B
수행 전	1000	2000
수행 후	950	2050

T_2	A	B
수행 전	950	2000
수행 후		2100

- 수행 후
  - A=950, B=2100
  - A+B 가 보존되지 않음

Locking(잠금)

- 병행처리 시에는 직렬수행과 같은 결과를 같도록 규칙을 정함 : locking protocol 이용
  - shared(공용) lock(S-lock) : 한 객체에 여러 트랜잭션이 동시에 걸 수 있다
  - exclusive(전용) lock(X-lock) : 한 객체에 어떤 트랜잭션이 걸어 놓으면 다른 트랜잭션이 어떠한 lock도 걸 수 없다

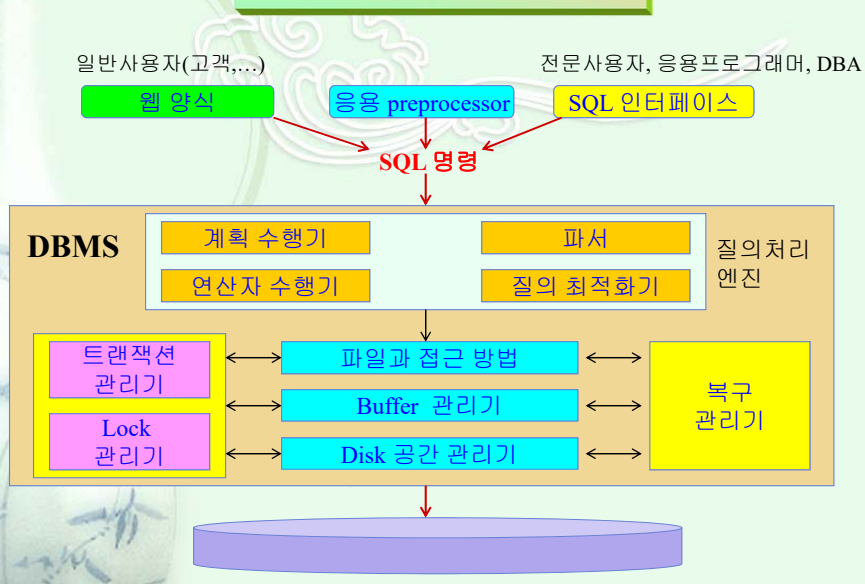
```
T3 : S(B);
     R(B);
     unlock(B);
     S(A);
     R(A);
     unlock(A);
     display(A+B).
```

```
T1 : X(A);
     R(A);
     A := A – 50;
     W(A);
     unlock(A);
     X(B)
     R(B);
     B := B + 50;
     W(B)
     unlock(B).
```


원자성(atomicity)

- 전부(all) 혹은 전무(nothing) 요구조건
- DBMS는 수행 도중 시스템 장애가 발생하더라도 원자성 보장
- 수행 도중 DBMS가 한 모든 단위 작업들의 로그(일지)를 보관 함으로서 해결
  - DB를 실제로 고치기 전에, 해당하는 로그 엔트리를 안전한 저장소에 먼저 강제 출력
  - 장애 발생 이후에는, 부분적으로 수행된 트랜잭션의 효과들을, 로그를 이용하여 무효화(undo)

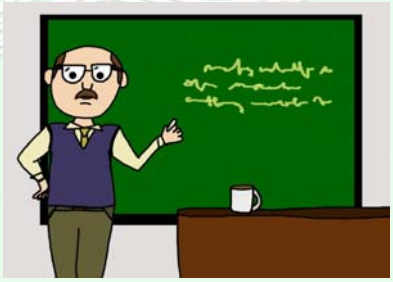
DBMS의 구조



Database 사용자

- 데이터베이스 개발자
  - DBMS S/W 구축, 공급업체에서 근무
- 최종 사용자(end user)
  - 작성된 응용만 사용, 기술적 지식 필요 없음
- 응용 프로그래머
  - 단순 사용자를 위하여 데이터 접근용 패키지 개발
  - 고급언어, SQL, DBMS 공급업체의 도구 이용
- 데이터베이스 관리자(DBA)
  - 논리적/물리적 스키마 설계
  - 보안과 사용권한부여 사항 처리
  - 데이터 가용성, 손상 복구
  - 데이터베이스 튜닝과 확장

Q & A



43