Overview

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Contents

Most parts are based on slides used in Stanford (http://web.stanford.edu/class/cs145)

Introduction to Databases

1. Overview of the DBMS

Contents

What is a DBMS?

- **A large, integrated collection of data**
- **Models a real-world *enterprise***
  - *Entities* (e.g., Students, Courses)
  - *Relationships* (e.g., Alice is enrolled in database course)

- **Consider building a course management system (CMS):**
  - Students
  - Courses
  - Professors

  Entities

  - Who takes what
  - Who teaches what

  Relationships

A **Database Management System (DBMS)** is a piece of software designed to store and manage databases

Data Models

- **A data model is a collection of concepts for describing data**
  - The relational model of data is the most widely used model today
    - Main Concept: the *relation*- essentially, a table

Columns

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Rows

- **A schema is a description of a particular collection of data, using the given data model**
  - E.g. every *relation* in a relational data model has a *schema* describing types, etc.

Modeling the CMS

Physical Schema: describes data layout

- Relations as unordered files
- Some data in sorted order (index)

Physical Schema : DB 물리적 구조 정의
-> Table / Index ... 의 물리적 저장소, Data file 위치 & 크기, Data 저장 방법 ... 포함
-> Logical Modeling 마친 후 물리적인 저장 방법으로 변환하는 과정을 거쳐야 함
=> 이때 물리적 구조 정의 역할

Logical Schema

- Students(sid: *string*, name: *string*, gpa: *float*)
- Courses(cid: *string*, cname: *string*, credits: *int*)
- Enrolled(sid: *string*, cid: *string*, grade: *string*)

Administrators

Applications

Relations

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

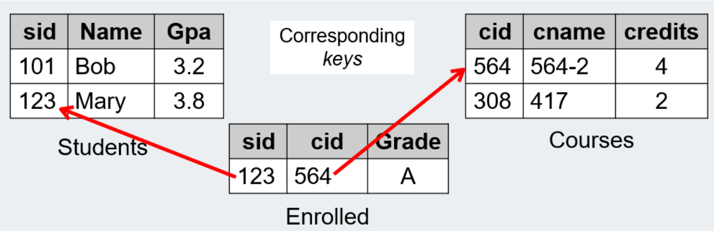
Students

cid	cname	credits
564	564-2	4
308	417	2

Courses

sid	cid	Grade
123	564	A

Enrolled



Logical Schema : DB의 논리적 구조 정의
= DB에서 쓰이는 data 형식, 구조 , 관계, .. 를 정의
-> 각 Entity & Attribute 간의 관계, 제약 조건, .. 정의

* 논리적 구조 : DB 사용하는 모든 사용자가 공유
=> data 일관성 & 무결성 보장 가능
-> Index, View, ... 의 구성 요소 정의 시에도 사용됨

External Schema: (Views)

- Course_info(cid: *string*, enrollment: *integer*)
- Derived from other tables

External Schema
: 특정 사용자 / 응용 프로그램이 DB에서 필요로 하는 data의 논리적 구조 정의
-> 필요한 data만 선택적으로 볼 수 있음

Data Independence

<u>Concept:</u> **Applications do not need to worry about *how the data is structured and stored***

<u>Logical data independence:</u>
protection from changes in the *logical structure of the data*

I.e. should not need to ask:
can we add a new entity or attribute without rewriting the application?

<u>Physical data independence:</u>
protection from physical layout changes

I.e. should not need to ask:
which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS

Data Independence
: 논리/물리적 구조에서의 변경이 다른 구조에 영향을 주지 않도록 하는 것
= 논리적 구조 변경해도 External Schema / App Code에 영향 X
= 물리적 구조 변경해도 논리적 구조 / 응용 프로그램에 영향 X
* 논리적 구조 : DB Schema
  물리적 구조 : DB 저장 방식

2. Overview of DBMS Topics: Key Concepts & Challenges

Contents

Challenges with Many Users

- **Suppose that our CMS application serves 1000's of users or more- what are some challenges?**
  - Security: Different users, different roles

    We won't look at too much in this course, but is extremely important

    사용자 인증, 접근 제어, 암호화 및 기타 보안 메커니즘을 제공

    -> 데이터의 기밀성, 무결성 및 가용성을 보호

  - Performance: Need to provide concurrent access

    Disk/SSD access is slow,
    DBMS hide the latency by doing more CPU work concurrently

    많은 양의 데이터를 빠른 처리 & 많은 수의 동시 접속 처리 필요

    -> DBMS는 쿼리 최적화, 인덱싱, 캐시 등의 기능을 제공

  - Consistency: Concurrency can lead to update problems

    DBMS allows user to write programs as if they were the **only** user

    여러 사용자가 동시에 DB 접근하면 data 일관성에 문제 발생 가능

    ⇒ Transaction & Lock Mechanism 제공

    -> But, Lock mechanism은 Concurrency(동시성) 저해할 수 있음

Transactions

- **A key concept is the transaction: an atomic sequence of DB actions (reads/writes)**

Acct	Balance
a10	20,000
a20	15,000

Transfer $3k from a10 to a20:
1. Debit $3k from a10
2. Credit $3k to a20

Acct	Balance
a10	17,000
a20	18,000

<u>**Atomicity**</u>: An action either completes *entirely* or *not at all*

Written naively, in which states is **atomicity** preserved?

- Crash before 1,
- After 1 but before 2,
- After 2.

DB Always preserves atomicity!

1) Transaction 실행 X -> DB에 영향 X
2) 1에서의 변경 사항은 commit X
   -> Transaction Rollback = DB 복원
3) Transaction 모든 작업 완료 -> commit

Transaction

1) Atomicity, 원자성
-> Transaction 은 전체 성공 / 실패인 원자적 작업 단위
⇒ 모든 작업 성공적 완료 시 DB 상태 변화
⇒ 작업 중 하나라도 실패 시 롤백 (일관성 유지에 중요한 역할)

2) Consistency, 일관성
-> Transaction 실행 전후의 DB 상태는 일관성 있어야 함
= Transaction 실행되는 동안 DB는 일관된 상태 유지해야 함

3) Isolation, 격리성
-> 다른 Transaction에 영향 X = 독립적 실행 필요 (동시 작업이어도)

4) Durability, 지속성
-> 성공적으로 작업 완료 시 영구적으로 반영
= system 장애 발생 시에도 해당 작업은 보존되어야 함

Concurrency: Scheduling Concurrent Transactions

- **The DBMS ensures that the execution of $\{T_1,\ldots,T_n\}$ is equivalent to some serial execution**

- **One way to accomplish this: Locking**
  - Before reading or writing, transaction requires a lock from DBMS, holds until the end

- **Key Idea*:* If T_i wants to write to an item x and T_j wants to read x, then T_i, T_j conflict*.***

Solution via locking:

- only one winner gets the lock
- loser is blocked (waits) until winner finishes

All concurrency issues handled by the DBMS…

A set of transactions is **isolated**
if their effect is as if all were executed serially

Locking
: Transaction이 Data 읽기/쓰기 전에 DBMS로부터 lock 요청 & 해당 lock 끝날 때까지 유지
-> 오직 하나의 transaction이 lock 얻을 수 있음
-> 다른 transaction은 잠금 해제 대기
⇒ 동시에 실행되는 여러 transaction 충돌 방지 & transaction 독립성 보장
⇒ DB 무결성 & 일관성 유지 가능

Isolated Transaction
: 모든 transaction이 일련의 serial 실행
& 동등한 결과를 만들도록 보장하는 것

Ensuring Atomicity

WAL
: 작업 수행 전 해당 작업에 대한 Log 기록을 disk에 저장
(Transaction 포함 & 작업 성공 여부 관계 X)
-> log 통해 system 장애 발생 / 새로운 요청 시 DB 복구 가능
⇒ System, 안정성 & 신뢰성 보장

- **DBMS ensures atomicity even if a transaction crashes!**

- **One way to accomplish this: Write-ahead logging (WAL)**

Write-ahead Logging (WAL):
Before any action is finalized, a corresponding log entry is forced to disk

- **Key Idea: Keep a log of all the writes done.**
  - After a crash, the partially executed transactions are undone using the log

All atomicity issues also handled by the DBMS…

A Well-Designed DBMS makes many people happy!

- **End users and DBMS vendors**
  - Reduces cost and makes money

Must still understand DB internals

- **DB application programmers**
  - Can handle more users, faster, for cheaper, and with better reliability / sec urity guarantees!

- **Database administrators (DBA)**
  - Easier time of designing logical/physical schema, handling security/authoriz ation, tuning, crash recovery, and more…

Summary

- **Key abstractions give data independence**

- **DBMS are used to maintain, query, and manage large datasets.**
  - Provide concurrency, recovery from crashes, quick application development, integrity, and security