



Overview

Prof. Hyuk-Yoon Kwon

<https://sites.google.com/view/seoultech-bigdata>

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

Contents

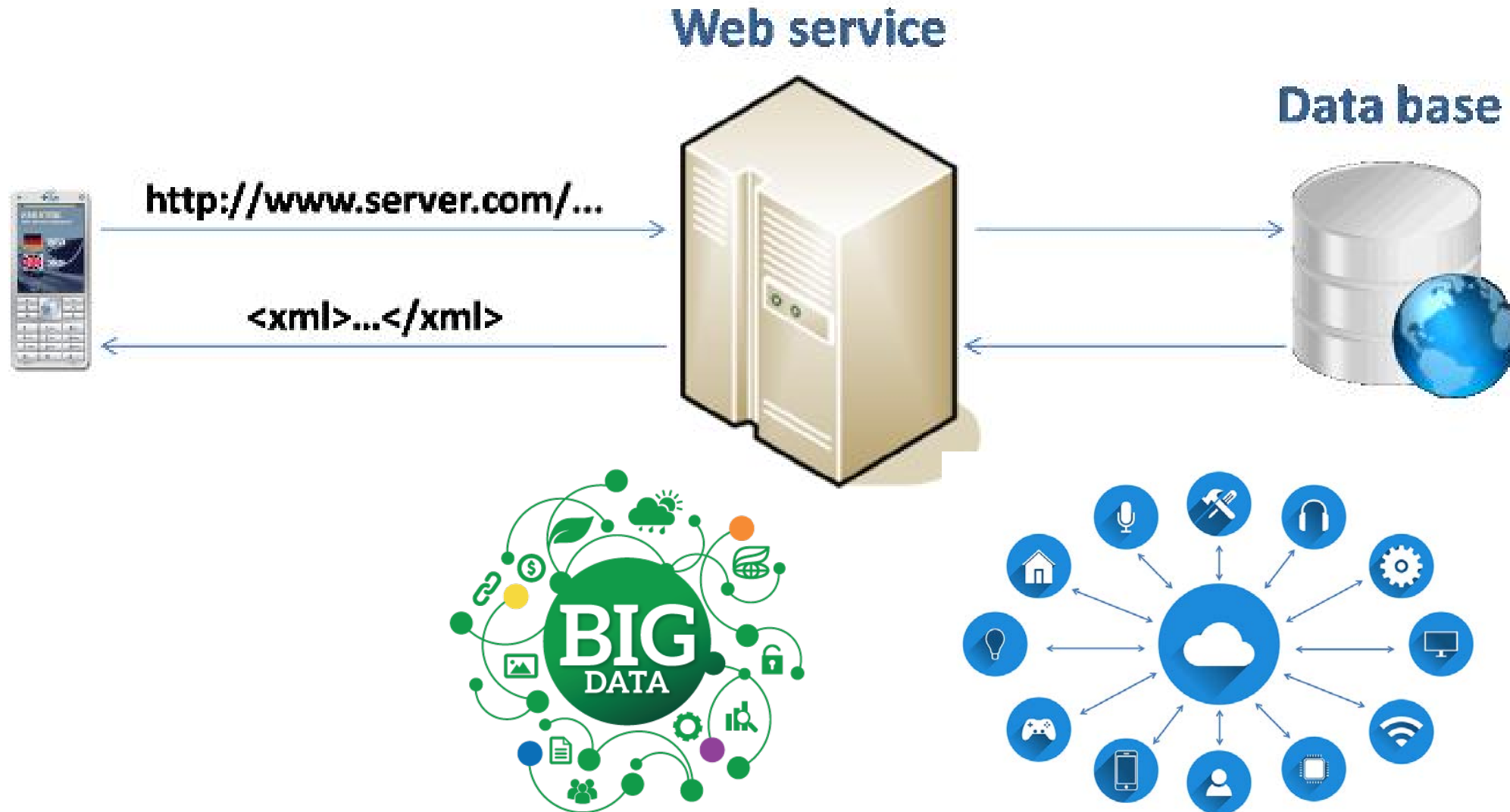
■ Part1: Course Overview

- Database Theory: Relational Model, Transaction, Indexing, and Optimization
- Database Practice: SQL, Oracle Practice, and database design

■ Part2: Introduction to Databases

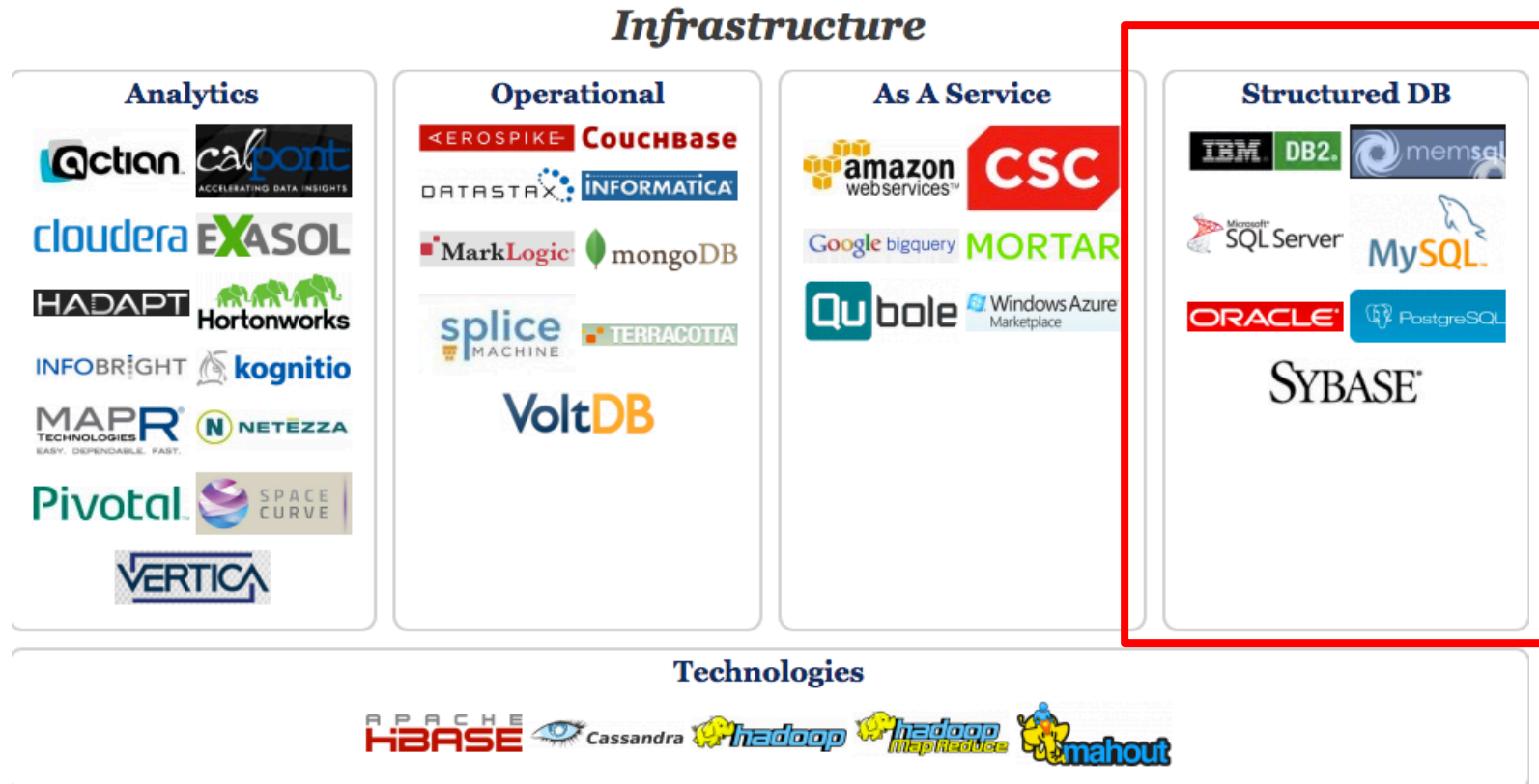
Course Theme

- To learn database theory and practice database



Why Data Is Important?

- The world is increasingly driven by data...
- Big Data Landscape...
 - Infrastructure is Changing



The Aim of Course

- 1.** To learn database concepts and basic theory
- 2.** To practice database issues such as SQL and database design
 - Using Oracle Database
- 3.** To apply the core concepts in the database into simple real applications

Course Policy: HW and Projects

■ Penalty for cheating

- The lowest grade

■ What is cheating?

- Sharing code: by copying, retyping, looking at, or supplying a file
- Searching the Web for solutions
- Helping your friend to write a lab, line by line

■ What is NOT cheating?

- Explaining how to use systems or tools
- Helping others with high-level design issues

■ Late submission is **NOT ALLOWED** for all the HW assignments and projects

- Be sure the deadline for every assignment and project

Course Policy: Grading

■ Exams (70%)

- Mid-term (30%)
- Final (40%)

■ Lab assignments (10%)

■ Individual presentation (20%)

Course Policy: Lectures

■ Flipped learning

- (Online lecture contents) The contents will be uploaded **by the previous Tuesday**
- (Offline lecture) **Every Tuesday 11:00 AM ~ (Assigned lecture time: from 10AM to 1PM)**
 - Summary (will be given in English)
 - Q&A (Korean questions are also allowed)

■ Lecture place

- Laboratory: Frontier 501

Course Policy: Attendance

■ Attendance is strongly recommended!

- Important points are announced only during the lecture
- Exam problems are inferred from the lecture

■ But, attendance itself is not included in the score

■ Attendance confirmation is done by e-class

- <http://eclass.seoultech.ac.kr>

Course Policy: Understanding

■ Encourage questions!

- Questions in Korean are allowed
- After the class, use emails and Q&A in e-class aggressively

■ For the questions that break the flow of class or need to discuss

- Please visit me during office hours (or any time if you make reservations)

Course Assumption

- **Recommended: Basic programming knowledge**

- Any kind of programming languages – Python, Java, or, C/C++

- **This course is designed for ITM students!**

Environments for Practices

■ Oracle Database

- Developer version can be found here: <http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>)

■ Live SQL

- <https://livesql.oracle.com/apex/f?p=590:1000>

Textbooks

Database Systems, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson

Seven Databases in Seven Weeks, Luc Perkins, Pragmatic Bookshelf

Getting Help

■ E-class: <http://eclass.seoultech.ac.kr>

- Complete schedule of lectures, exams, and assignments
- Lecture slides, assignments, exams, solutions
- Clarifications to assignments
- Q&A

■ Office hours

- Wednesday, 2:00-3:00pm, Frontier 614
- You can schedule 1:1 appointment in advance even not for the office hours

Course Schedule

강의 진도 계획		
주차	강의내용	강의방법, 과제, 평가
1 (영문)	Overview	Offline
2 (영문)	Relational model	Online
3 (영문)	SQL1	Online
4 (영문)	SQL2	Online
5 (영문)	SQL3	Online
6 (영문)	Database design1	Offline
7 (영문)	Individual presentation - Part1	Offline (can be changed to Online)
8 (영문)	Mid-term exam	Offline

9 (영문)	Database design2	Offline
10 (영문)	Indexing1	Online
11 (영문)	Indexing2	Online
12 (영문)	Transactions1	Online
13 (영문)	Transactions2	Offline
14 (영문)	Individual presentation - Part2	Offline (can be changed to Online)
15 (영문)	Final exam	Offline



Introduction to Databases

1. Overview of the DBMS

Contents

- 1. Definition of DBMS**
- 2. Data models & the relational data model**
- 3. Schemas & data independence**

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)

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

A Motivating, Running Example

■ Consider building a course management system (CMS):

- Students
- Courses
- Professors

} *Entities*

- Who takes what
- Who teaches what

} *Relationships*

Data Models

■ A data model is a collection of concepts for describing data

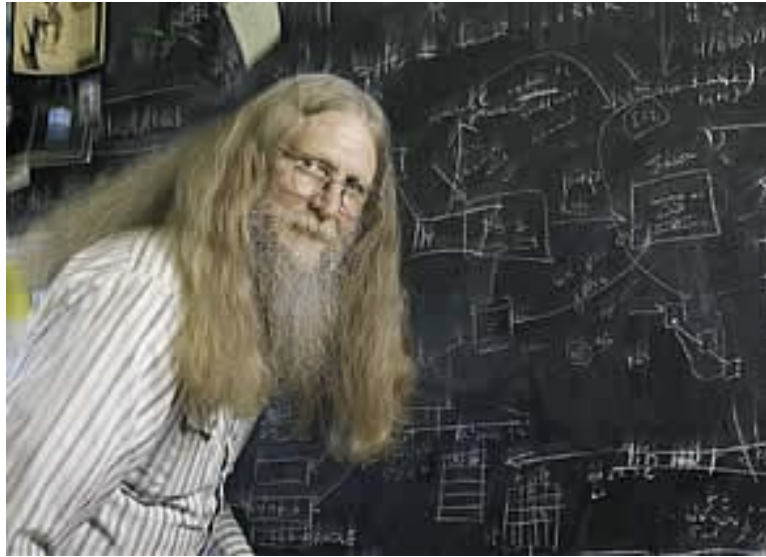
- The relational model of data is the most widely used model today
 - Main Concept: the *relation*- essentially, a table

The diagram shows a table with four columns and four rows. A blue bracket labeled 'Columns' spans the top of the table. A blue bracket labeled 'Rows' spans the left side of the table. The word 'Product' is written in red to the left of the first column. The table contains the following data:

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

■ 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.



“Relational databases form the bedrock of western civilization”

- Bruce Lindsay, IBM Research

Modeling the CMS

■ Logical Schema

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

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Relations

cid	cname	credits
564	564-2	4
308	417	2

Courses

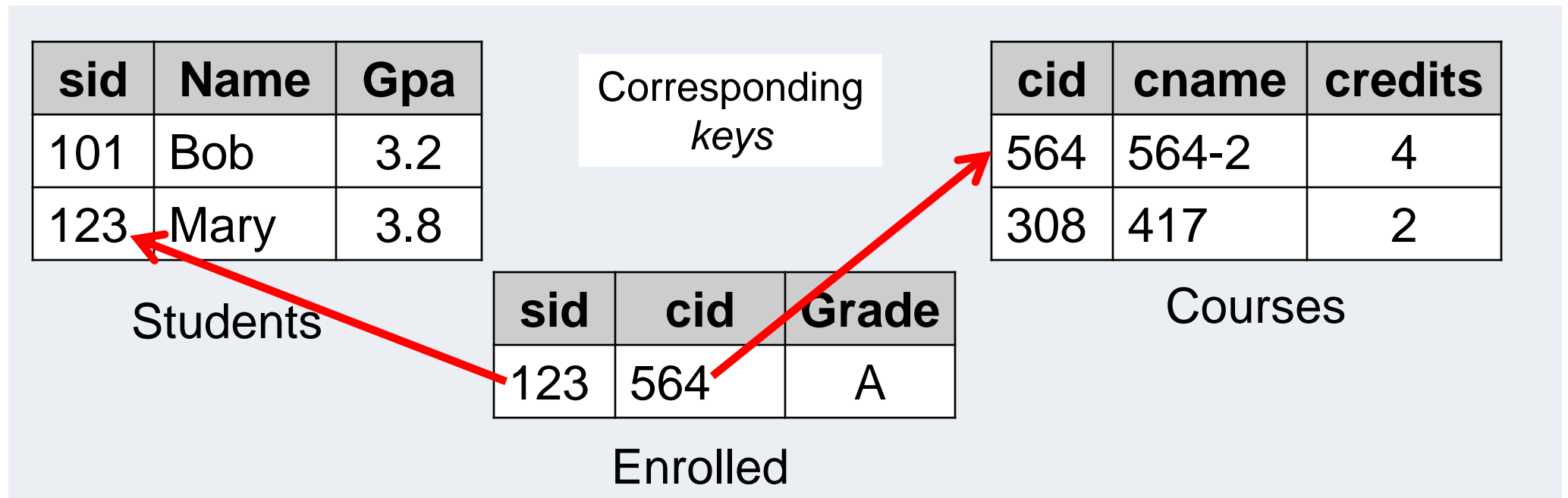
sid	cid	Grade
123	564	A

Enrolled

Modeling the CMS

■ Logical Schema

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



Other Schemas...

■ Physical Schema: describes data layout

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

■ Logical Schema: Previous slide

■ External Schema: (Views)

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



Administrators



Applications

Data Independence

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

Logical data independence: 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?

Physical data independence: 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

2. Overview of DBMS Topics: Key Concepts & Challenges

Contents

1. Transactions

2. Concurrency & locking

3. Atomicity & logging

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

- Performance: Need to provide concurrent access

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

- Consistency: Concurrency can lead to update problems

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

Transactions

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

Atomicity: An action either completes *entirely* or *not at all*

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

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

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

DB Always preserves atomicity!

Concurrency: Scheduling Concurrent Transactions

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

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

- 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...

Ensuring Atomicity

- DBMS ensures atomicity even if a transaction crashes!

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

- Key Idea: Keep a log of all the writes done.

- After a crash, the partially executed transactions are undone using the log

Write-ahead Logging

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

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

■ DB application programmers

- Can handle more users, faster, for cheaper, and with better reliability / security guarantees!

■ Database administrators (DBA)

- Easier time of designing logical/physical schema, handling security/authorization, tuning, crash recovery, and more...

Must still understand
DB internals

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

Welcome and Enjoy!