1. Overview of the relational data model

Spring 2017

Schedule*

0. Seminar Overview
1. Overview of the Relational Data Model
2. Why DB Design?
3. Development Process
4. Requirements
5. Conceptual Data Model
6. Generalization & Specialization
7. Relational Database Design
8. Normalization
9. Keys and Constraints

*subject to change without notification

What will be dealt with in this section

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

What is a DBMS?

- Database - A large, integrated collection of data

- Models a real-world *enterprise*
  - *Entities* (e.g., Students, Courses)
  - *Relationships* (e.g., Jinsoo is enrolled in SEP545)

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*

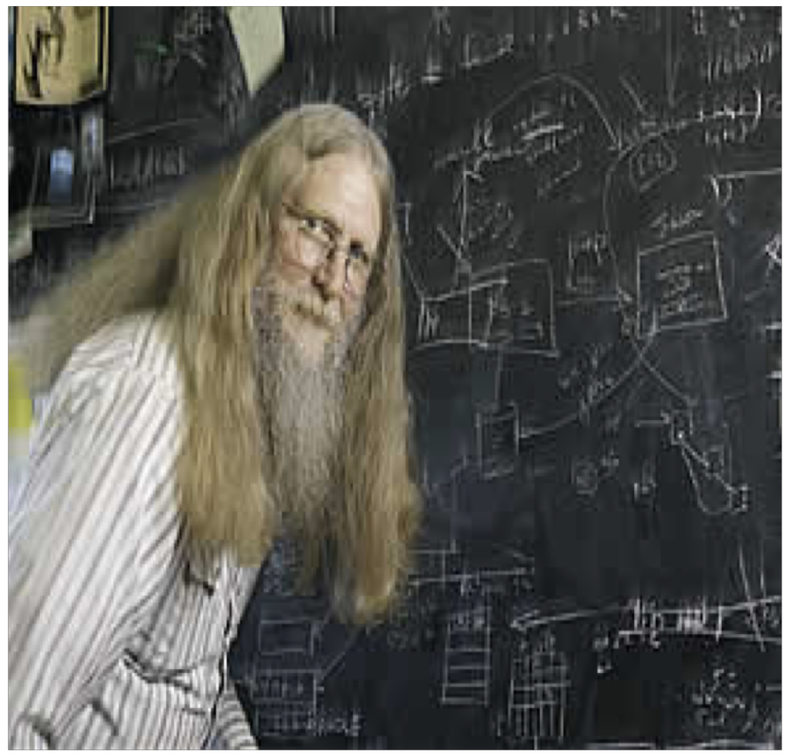
Data models

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

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

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

Corresponding *keys*

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

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*)

Relations

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Other Schemata…

Physical Schema: describes data layout

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

Administrators

Logical Schema: Previous slide

External Schema: (Views)

Applications

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

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

Overview of DBMS topics (첨부)

Key concepts & challenges

What will be dealt with in this section

1. Transactions
2. Concurrency & locking
3. Atomicity & logging
4. Summary

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 (TXN)**: 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!

Transactions

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

- If a user cancels a TXN, it should be as if nothing happened!

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

Transactions leave the DB in a **consistent** state

- Users may write integrity constraints, e.g., 'each course is assigned to exactly one room'

Consistency: An action results in a state which conforms to all integrity constraints

However, note that the DBMS does not understand the *real* meaning of the constraints- consistency burden is still on the user!

Challenge: Scheduling Concurrent Transactions

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

A set of TXNs 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

What if T_i and T_j need X and Y, and T_i asks for X before T_j, and T_j asks for Y before T_i?
-> *Deadlock!* One is aborted…

All concurrency issues handled by the DBMS…

Ensuring Atomicity & Durability

DBMS ensures **atomicity** even if a TXN 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 TXNs are undone using the log

We assume that the log is on "stable" storage

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…

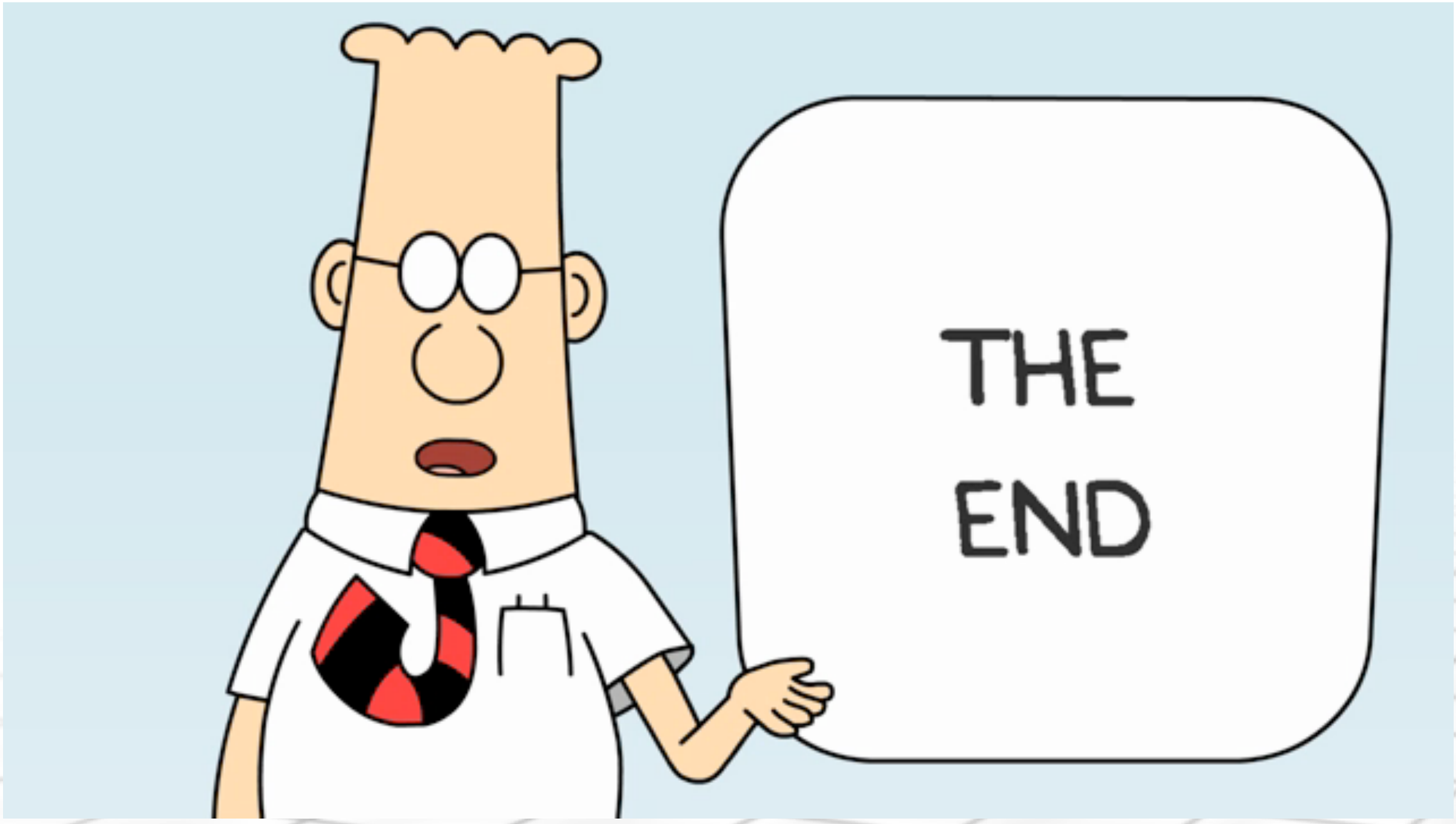
Must still understand DB internals

Summary of DBMS

DBMS are used to maintain, query, and manage large datasets.

- Provide concurrency, recovery from crashes, quick application development, integrity, and security

Key abstractions give **data independence**



출처: metachannels.com

Thank you!