CS 6400 A

Database Systems Concepts and Design

Lecture 1

08/19/24

Agenda

1. Course logistics and overview

2. Why study relational databases?

3. Relational data model

The essentials

Instructor: Kexin Rong

- Office: Klaus 3322

TAs:

- Rajveer Bachkaniwala
- Shreyas Kanjalkar
- Sashankh Chengavalli Kumar

How to reach us: cs6400-staff@groups.gatech.edu

- The above email reaches all of the course staff. You are strongly encouraged to use this, instead of emailing individual course staff.

The essentials

Course website: https://kexinrong.github.io/fa24-cs6400/

schedule, assignments, and course material

Canvas/Gradescope: submitting assignments

Piazza: discussing course contents and finding teammates

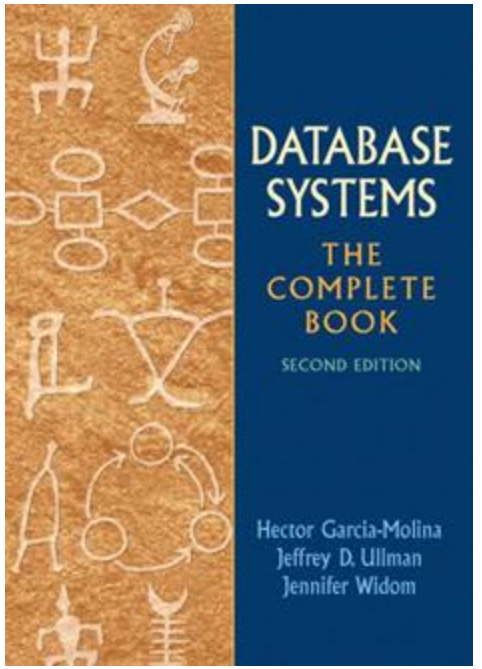
- https://piazza.com/class/lzkp9kbomoc60h

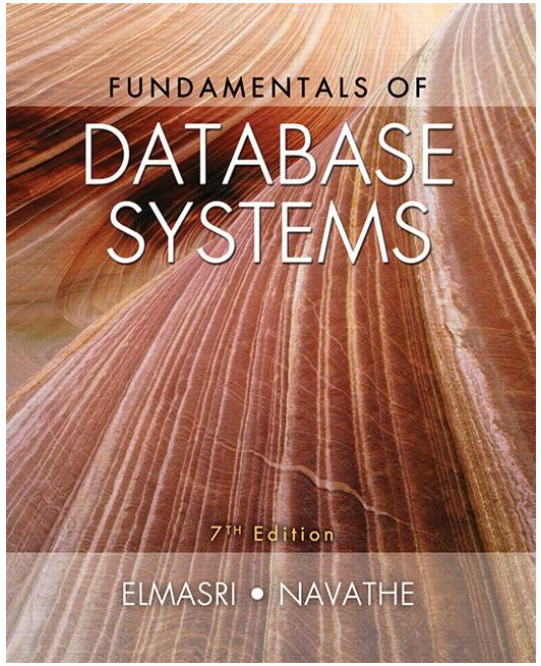
Email: special requests; mention CS6400 in the email title

OH: Starting next week. Time will be announced.

Course materials

- Textbooks:
  - Database Systems: The Complete Book (2nd edition)
  - Fundamentals of Database Systems
  - Can use interchangeably
- Both books have international versions and have PDFs searchable online





Grading

Assignments – 45%

- Mix of individual and team

Course Project – 20%

- Team-based

Exams – 35%

- Midterm 1– 20%
- Midterm 2 – 15%

Details: https://kexinrong.github.io/fa24-cs6400/grading/

Assignment 0: Questionnaire

Available on canvas

Due next Monday (Aug 26) @ 11:59PM

Tell us about your database background

* Assignment not graded

Programming Assignments

Assignment 1: JSON to SQLite (15%)

- Schema design, bulk loading and querying with SQL
- Programming language: Python and SQL

Assignment 2: Memstore (15%)

- Storage and access methods
- Programming language: Java

Done individually; A1 will be posted on Wednesday

Paper Assignments

Assignment 3: Research paper presentation (10%)

- group in-class presentation

Assignment 4: Research paper review (5%)

- Individual
- Write a critique on the paper you've presented

Course Project

One of the following:

- Novel database applications
- Implementation and improvement of some aspects of a database system
- Benchmarking of database management systems or components
- *Exception: research projects with other faculty members; need to check with the instructor

Requirements

- Need to use database systems
- Groups of 3-5, depending on size of the class

Examples of past projects can be found on Canvas

- Files -> Sample Projects

Exams

- Written tests based on material covered in lectures, assignments and readings
  - Midterm-1: in-class (Sep 25)
  - Midterm-2: take-home

- Midterm-2 will cover the entire course, but focus on the second half

Attendance

I dislike mandatory attendance.

But in the past we noticed…

- People who did not attend did worse ☹
- People who did not attend used more course resources ☹
- People who did not attend were less happy with the course ☹

This year's policy: voluntary attendance

- Except during classes that you need to present

Course Policy - IMPORTANT

Follow the Georgia Tech Honor Code!

Late policy: One automatic late day without penalty. Otherwise 10% deduction per 24 hours.

Generative AI policy: Clearly attribute AI-generated contents (e.g., direct quotes, different color text). Do not use generative AI tools to write code for you.

Details: https://kexinrong.github.io/fa24-cs6400/policy/

Why study relational databases?

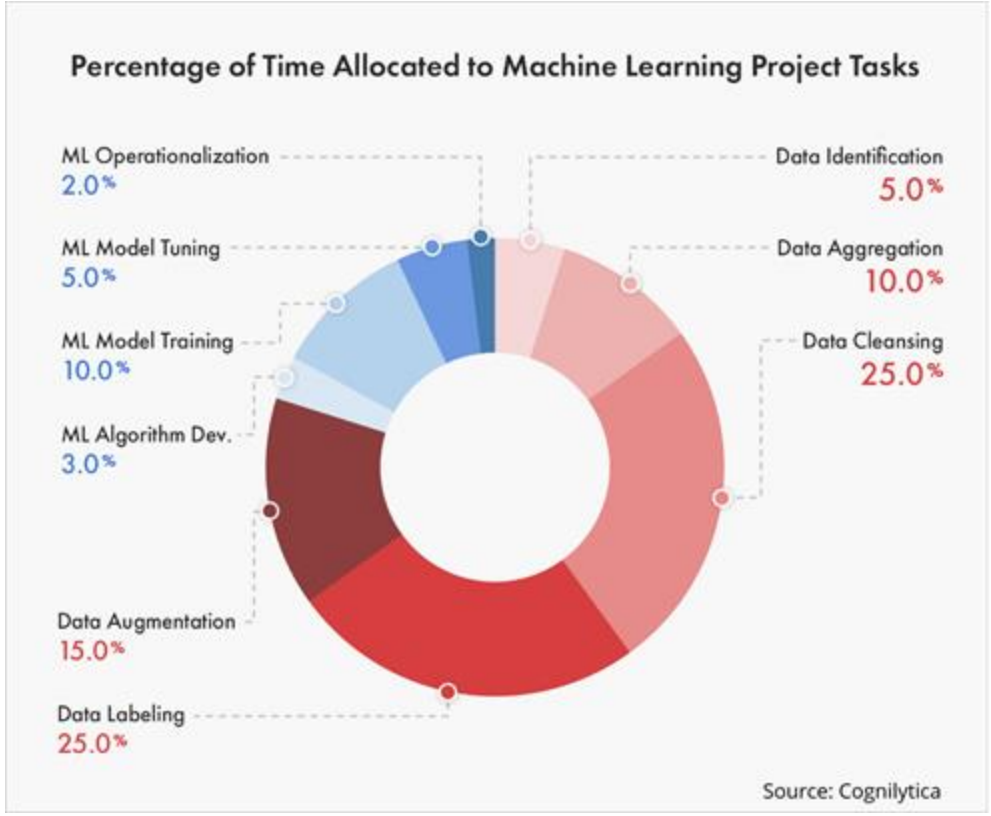
Why study relational databases?

Most important computer applications must manage, update and query datasets

- Bank, store, search app…

Data quality, quantity & timeliness becoming even more important with AI

- Machine learning = algorithms that generalize from data



Relational databases => data-intensive systems

Relational databases are the most popular type of data-intensive system (MySQL, Oracle, etc)

Many other systems facing similar concerns: key-value stores, streaming systems, ML frameworks, your custom app?

Goal:

- Learn how to use and design relational databases
- Get a taste of the main issues and principles that span all data-intensive systems

Typical System Challenges

Reliability in the face of hardware crashes, bugs, bad user input, etc

Concurrency: access by multiple users

Performance: throughput, latency, etc

Access interface from many, changing apps

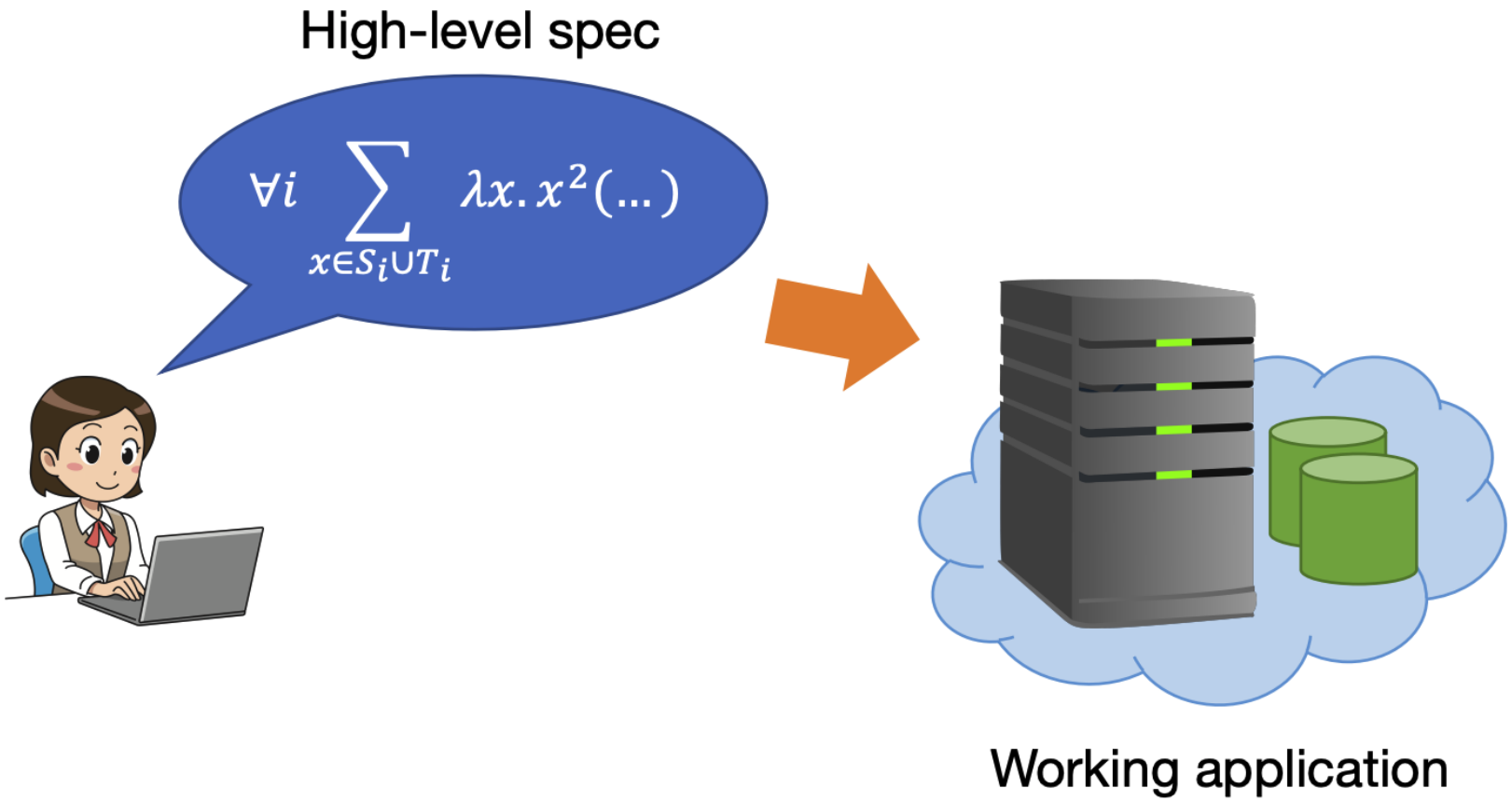
Security and data privacy (not covered in this course)

Scientific Interest

Interesting algorithmic and design ideas

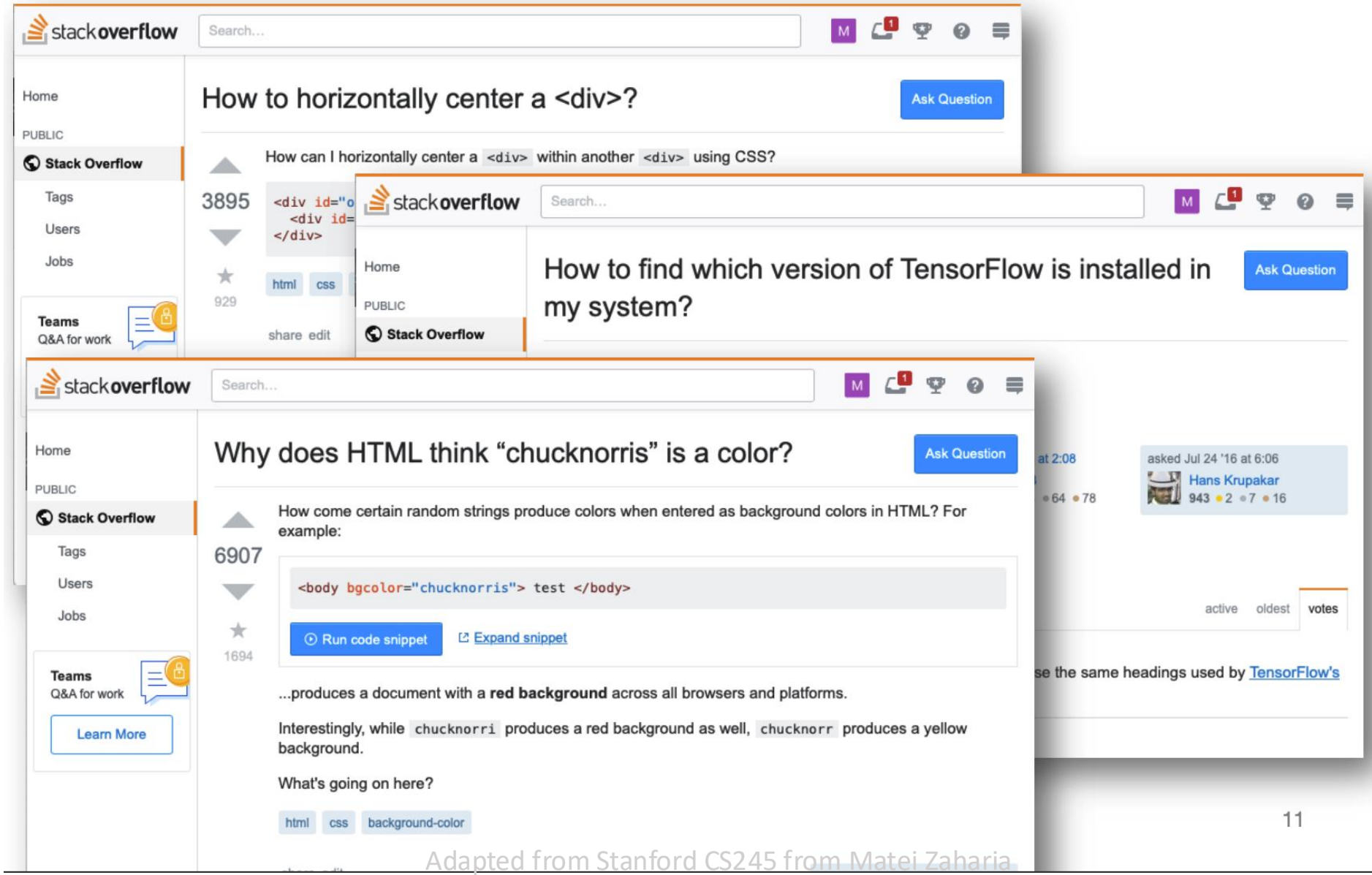
In many ways, data systems are the highest-level successful programming abstractions

Programming: The Dream



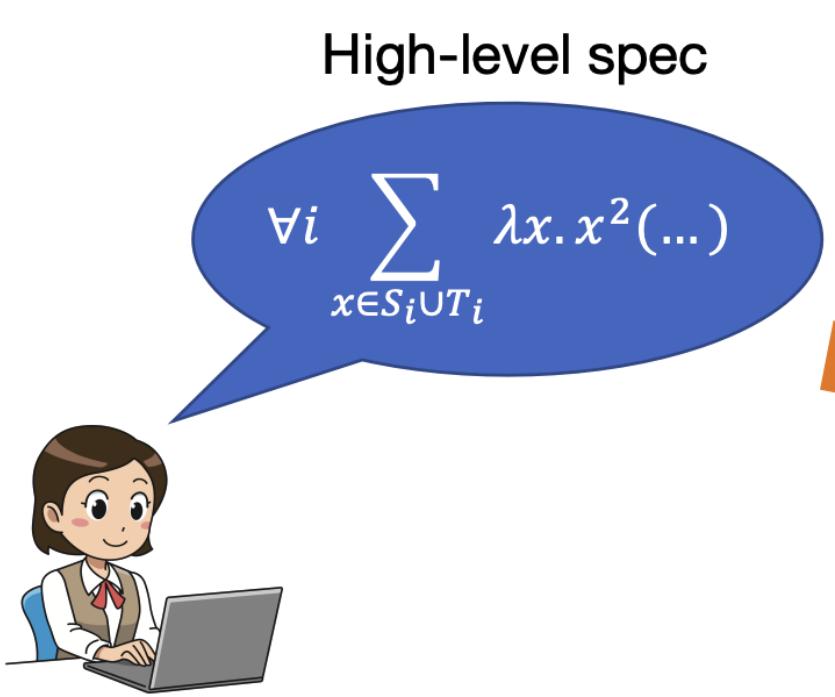
Adapted from Stanford CS245 from Matei Zaharia

Programming: The Reality



Adapted from Stanford CS245 from Matei Zaharia

Programming with databases



PostgreSQL

Working application

Actually manages:

- Durability
- Concurrency
- Query optimization
- Security
- …

Case study: building a book-selling website

- E.g., your own version of mini-Amazon
- Large data! (think about all books in the world or even in English)

Where do we get started?

Q1: Who are the key people?

- At least two types:
  - Database admin (assuming they own all copies of all the books)
  - Customers who purchase books
  - Let's proceed with these two only

- Other people:
  - Sellers
  - Who deal with the warehouse of the books
  - …

Q2: What should the user be able to do?

- i.e., what does the interface look like? (think about Amazon)

1. Search for books
   - According to author, title, price range, …
2. Purchase books
3. Add to wishlist
4. …

Q3: What should the platform do?

1. Returns books as searched by the authors
2. Check that the payment method is valid
3. Update no. of copies as books are sold
4. Manage total money it has
5. Add new books as they are published
6. …

Q4: What are the desired and necessary properties of the platform?

- Should be able to handle a large amount of data
- Should be efficient and easy to use (e.g., search with authors as well as title)
- If there is a crash or loss of power, information should not be lost or inconsistent
  - Imagine a user was in the middle of a transaction when a crash happened, paid the money, but the book has not been purchased
- No surprises with multiple users logged in at the same time
  - Imagine one last copy of a book that two users are trying to purchase at the same time
- Easy to update and program
  - For the admin

That was the design phase (a basic one though)

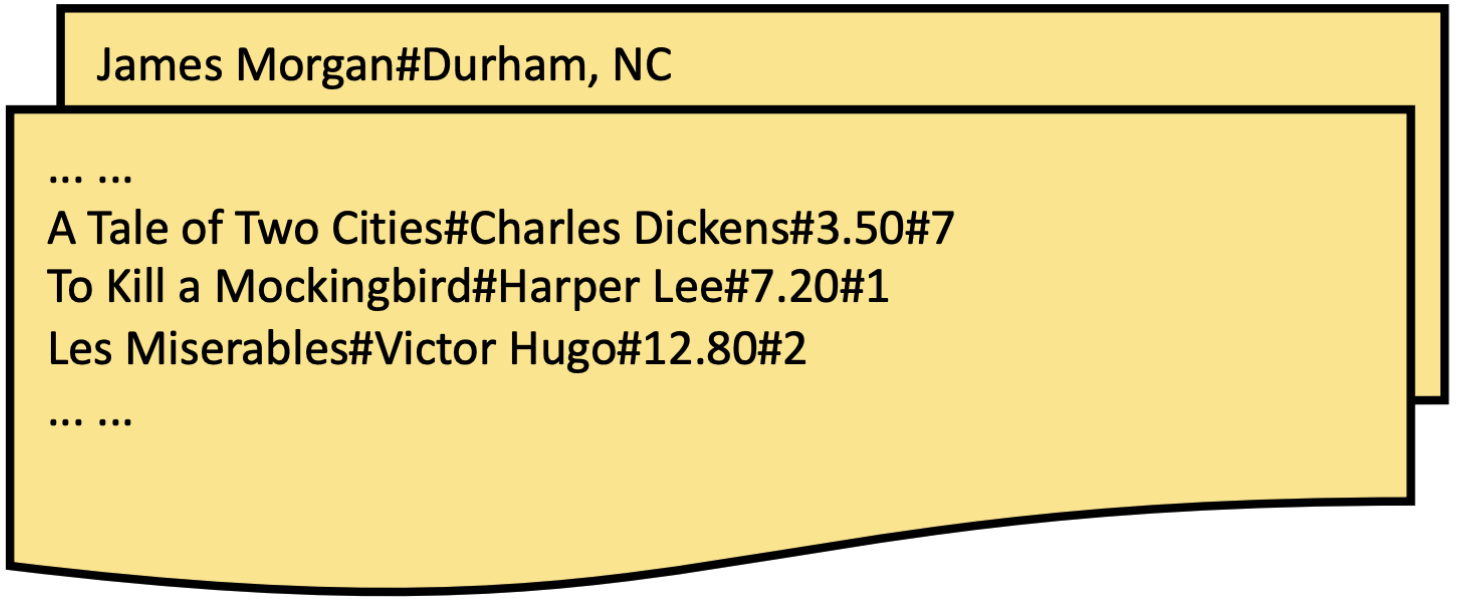
Let's implement this!

How about:

- Your favorite programming language
- On data stored in large files

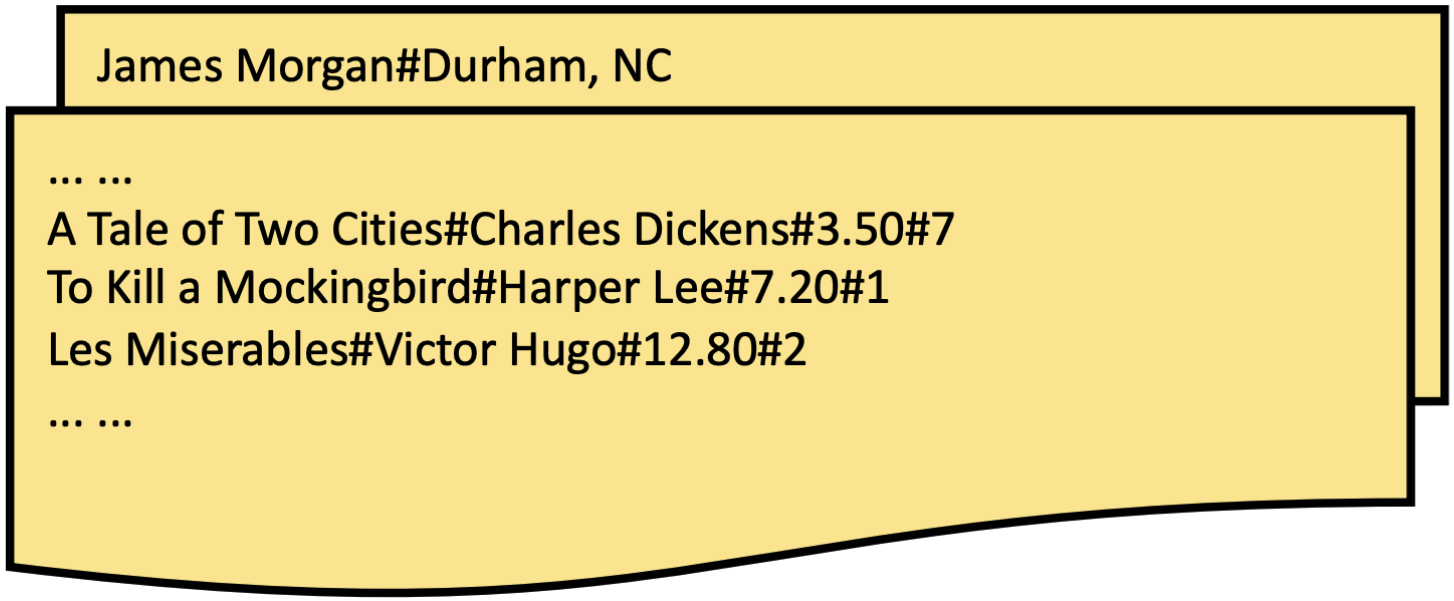
Image source: https://www.flaticon.com/free-icon/girl_5986069

Sounds simple!



- Text files – for books, customers, ..
- Books listed with title, author, price and no. of copies
- Fields separated by #’s

Query by programming



- James Morgan wants to buy "To Kill a Mockingbird"
- A simple script
  - Scan through the book files
  - Look for the line containing "To Kill a Mockingbird"
  - Check if there are more than 1 copy left
  - Charge James $7.20 and reduce the number of copies by 1

Better idea than scanning?

Binary search (with file sorted on titles)

What if he changes the "query" and wants to buy a book by Victor Hugo?

Revisit: What are the desired and necessary properties of the platform?

- Should be able to handle a large amount of data

Try to open a 10-100 GB file

- Should be efficient and easy to use (e.g., search with authors as well as title)

Try to search both on a large flat file

- If there is a crash or loss of power, information should not be lost or inconsistent
  - Imagine a user was in the middle of a transaction when a crash happened, paid the money, but the book has not been purchased

Imagine programmer's task

- No surprises with multiple users logged in at the same time
  - Imagine one last copy of a book that two users are trying to purchase at the same time
- Easy to update and program
  - For the admin

Imagine adding a new book or updating copies (+ allow search) on a 10-100 GB text file

Solution?

DBMS = Database Management System

What is a DBMS?

A large, integrated collection of data

Models a real-world *enterprise*

- *Entities* (e.g., Customers, Books)
- *Relationships* (e.g., James purchases a tale of two cities)

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

A DBMS takes care of all of the following (and more): In an easy-to-use, efficient, and robust way

- Should be able to handle a large amount of data
- Should be efficient and easy to use (e.g., search with authors as well as title)
- If there is a crash or loss of power, information should not be lost or inconsistent
- No surprises with multiple users logged in at the same time
- Easy to update and program

Optimization

Index

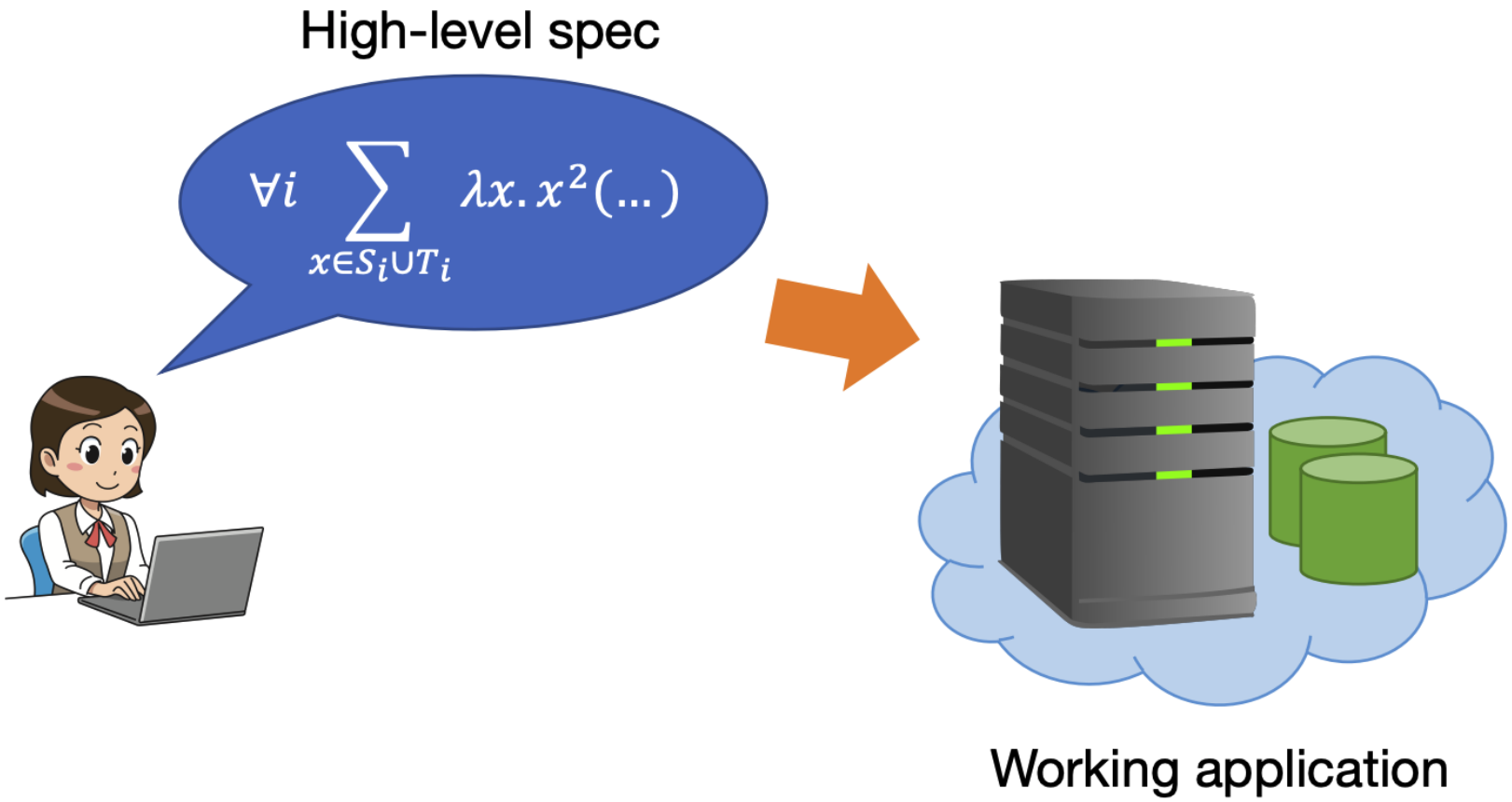
Recovery

Concurrency Control

Declarative

** We will learn these in this course!*

Programming: The Dream



Adapted from Stanford CS245 from Matei Zaharia

This course gives an (advanced) intro to DBMS

1. How can one use a DBMS (programmer's/designer's perspective)

- Run queries, update data (SQL, Relational Algebra)
- Design a good database (ER diagram, design theory)

2. How does a DBMS work (system's perspective, also for programmers for writing better queries)

- Storage and index
- Query processing and optimizations
- Transactions: recovery and concurrency control

3. Glimpse of advanced topics and other DBMS

- Map Reduce, Spark, NewSQL
- Selected research papers

Should I take this class?

You are expected to be comfortable programming in languages such as Python and Java, but not in SQL.

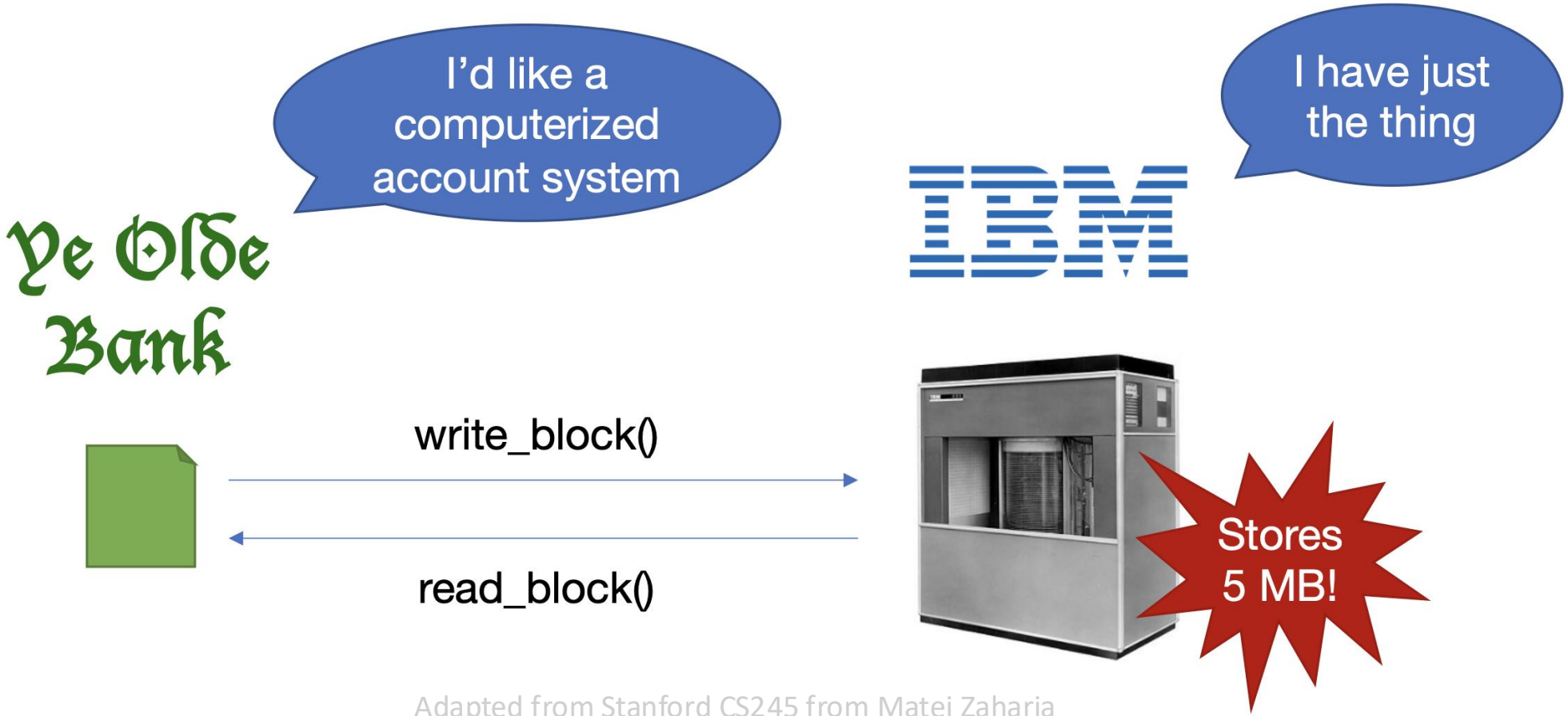
The class does NOT assume prior background in databases.

Check out our syllabus - if you have sufficient undergraduate database coursework, consider taking *CS6422: Database System Implemnt* instead

Relational Model

Early Data Management

At first, each application did its own data management directly against storage (e.g., our book-selling website example)



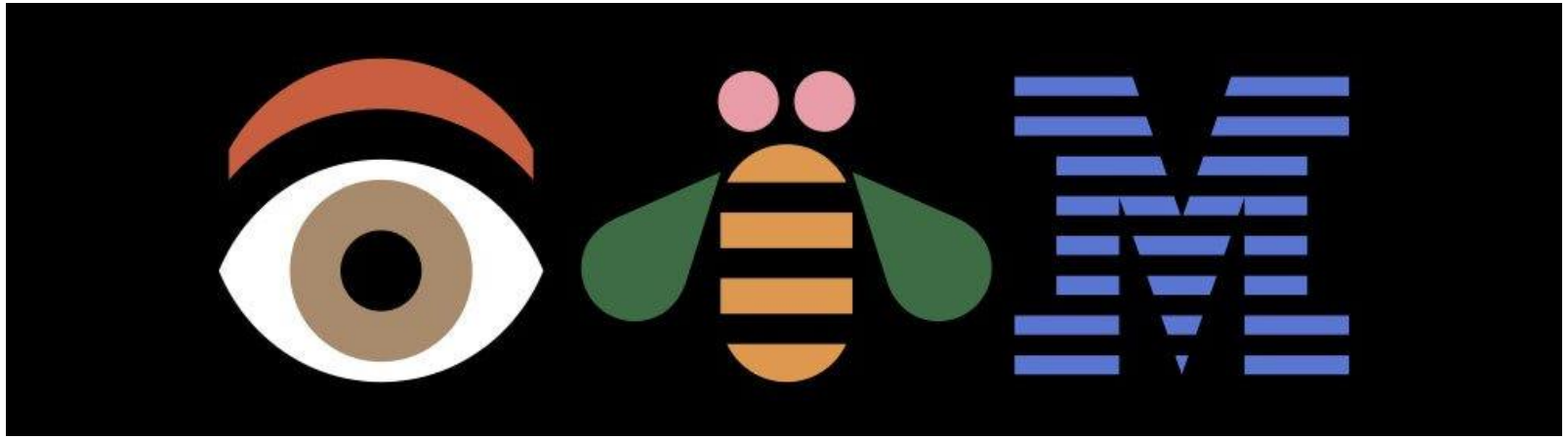
Problems with App Storage Management

- How should we lay out and navigate data?
- How do we keep the application reliable?
- What if we want to share data across apps?

Every app is solving the *same* problems!

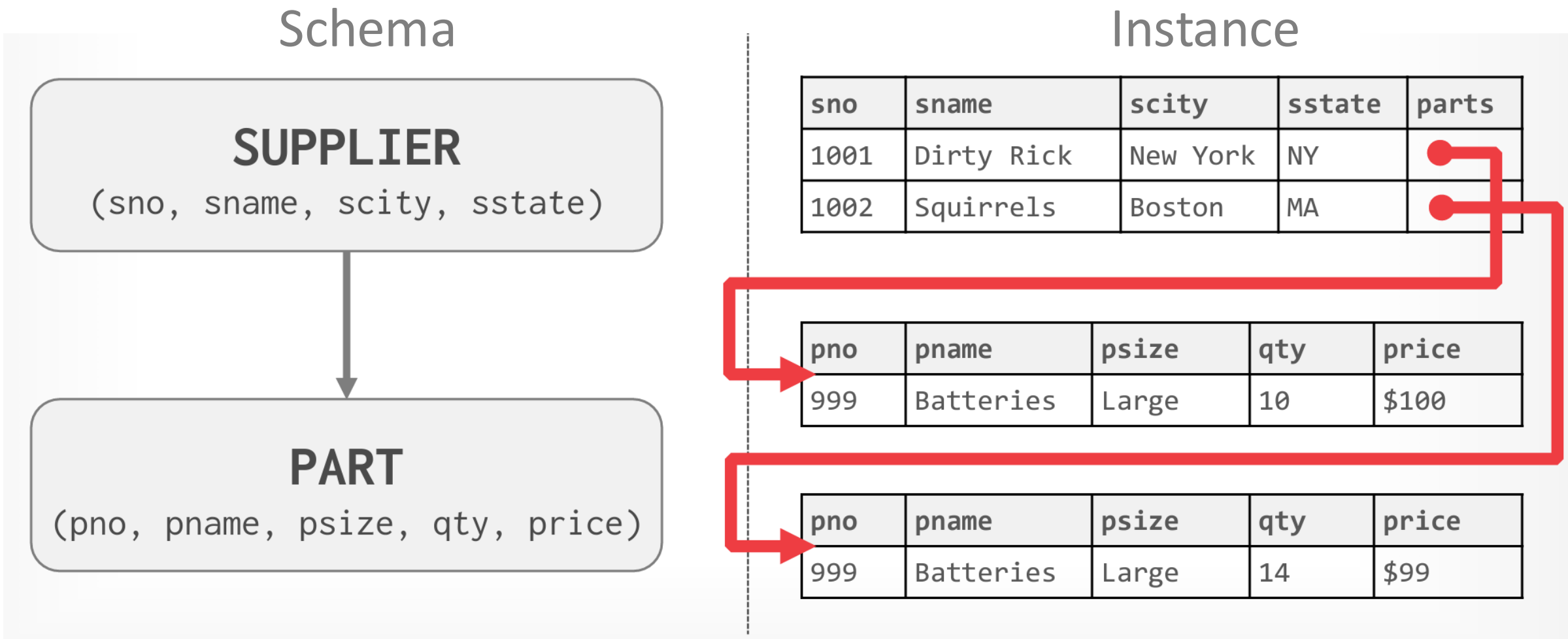
1960s – IBM IMS

- Information Management System
- Early database system developed to keep track of purchase orders for Apollo moon mission.
  - Hierarchical data model.
  - Programmer-defined physical storage format.
  - Tuple-at-a-time queries.



Acknowledgement: Prof. Andy Pavlo, CMU

Hierarchical Data Model



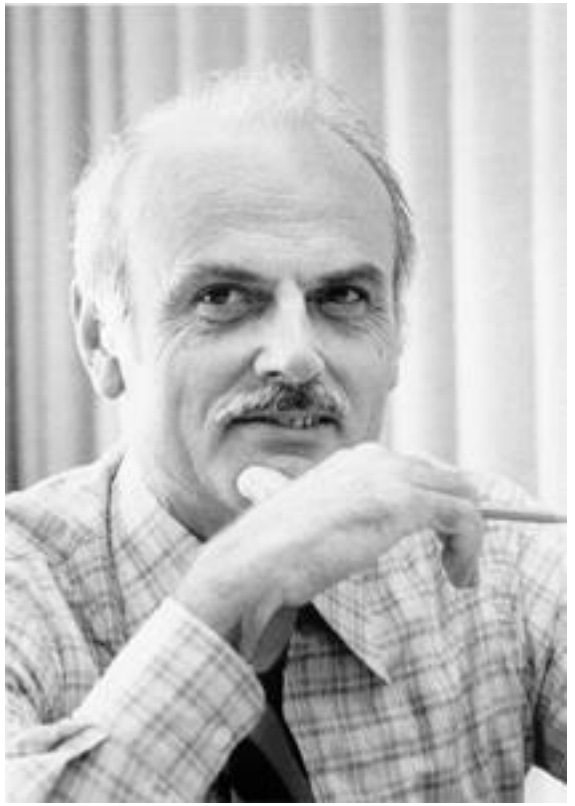
Acknowledgement: Prof. Andy Pavlo, CMU

1970s - Relational data model

Turing Award 1981

- Ted Codd was a mathematician working at IBM Research. He saw developers spending their time rewriting IMS programs every time the database's schema or layout changed.

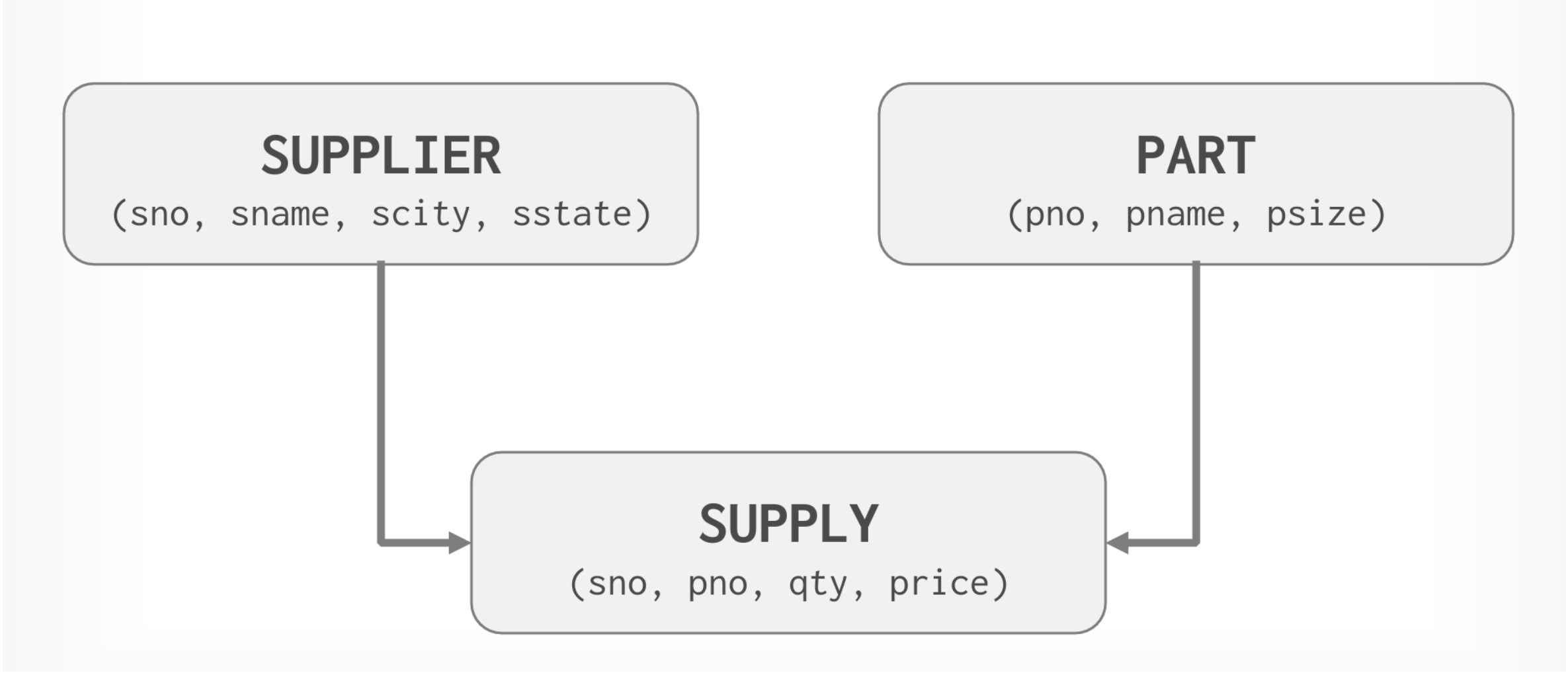
- Database abstraction to avoid this maintenance:
  - Store database in simple data structures.
  - Access data through set-at-a-time high-level language.
  - Physical storage left up to implementation.



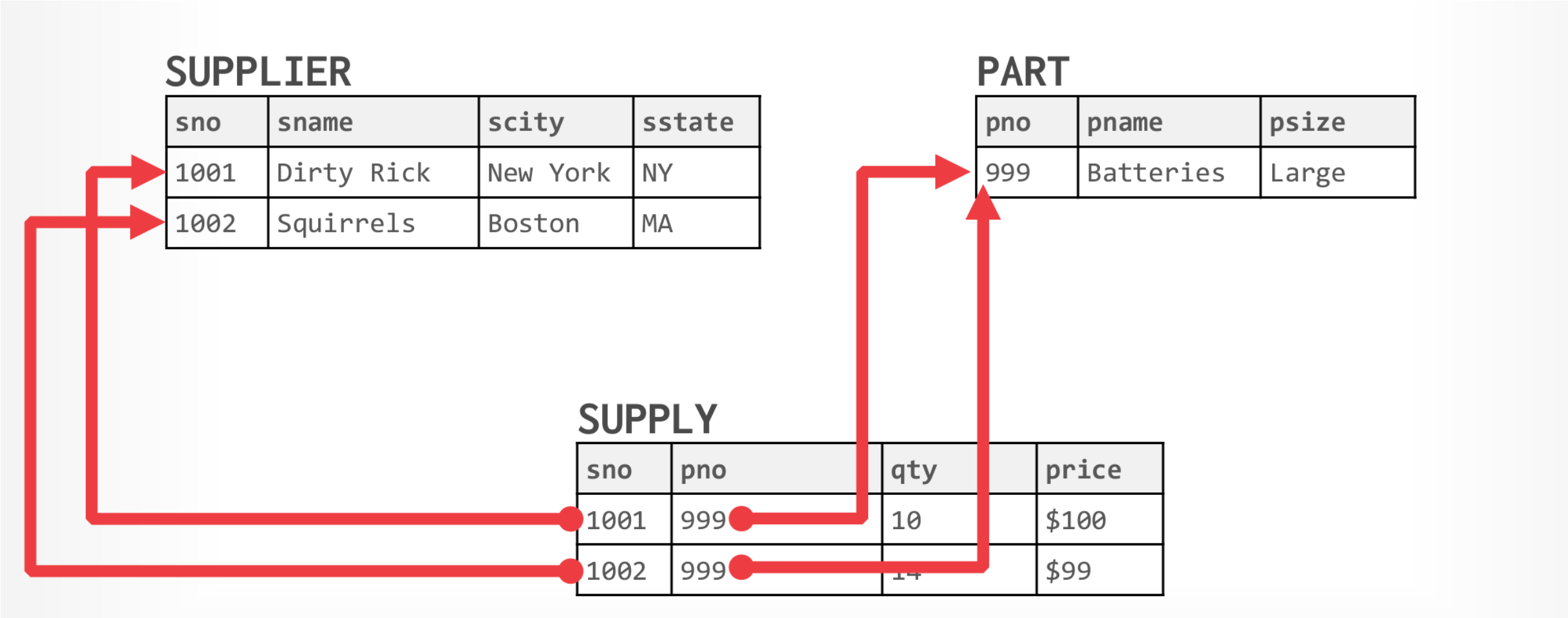
Codd

Acknowledgement: Prof. Andy Pavlo, CMU

Relational Data Model - *schema*



Relational Data Model - *instance*



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 table, or remove a field in a table 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

Data model

A notation for describing data or information

Consists of:

- Structure of the data
- Operations on the data
- Constraints on the data

Structure of the data

- Referred to as a “conceptual model” of the data
- Higher level than “physical models” or data structures like arrays and lists
- Example: a relation consists of a schema, attributes, and tuples

title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime

Operations on the data

Usually a limited set of operations that can be performed

- Queries (operations that retrieve information)
- Modifications (operations that change the database)
- Relation algebra

This is a strength, not a weakness

- Programmers can describe operations at a very high level
- The DBMS implements them efficiently
- Not easy to do when coding in C

```
SELECT *
FROM Movies
WHERE studioName = ‘Disney’
AND year = 2013;
```

Constraints on the data

Usually have limitations on the data; helpful for data quality

Examples

- Day of a week is an integer between 1 and 7
- Age is larger than 0
- Student IDs are unique

Data models

- **Relational** → Most DBMS's
- Key/Value
- Graph
- Document (Semi-structured)
- Column-family

(Key/Value, Graph, Document, Column-family) } No SQL

- Array/Matrix → Machine Learning
- Hierarchical
- Network

(Hierarchical, Network) } Obsolete

The relational model

- Structure
  - Based on tables (relations)
  - Looks like an array of structs in C, but this is just one possible implementation
  - In database systems, tables are not stored as main-memory structures and must take into account the need to access relations on disk

title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime

The relational model

- Operations
  - Relational algebra
  - E.g., all the rows where genre is “anime”
- Constraints
  - E.g., Genre must be one of a fixed list of values, no two movies can have the same title

title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime

The semi-structured model

Structure
- Resembles trees or graphs, rather than tables or arrays
- Represent data by hierarchically nested tagged elements

Operations
- Involve following path from element to subelements

Constraints
- Involve types of values associated with tags
- E.g., <Length> tag values are integers, each <Movie> element must have a <Length>

```
<Movies>
  <Movie title=”Oldboy”>
    <Year>2003</Year>
    <Length>120</Length>
    <Genre>mystery</Genre>
  </Movie>
  <Movie title=”Ponyo”>
    <Year>2008</Year>
    …
</Movies>
```

The key-value model

- Structure
  - (key, value) pairs
  - Key is a string or integer
  - Value can be any blob of data
- Operations
  - get (key), put(key, value)
  - Operations on values not supported
- Constraints
  - E.g., key is unique, value is not NULL

key	*value*
1000	(oldboy, 2003)
1001	(ponyo, 2008)
1002	(frozen, 2013)

Comparison of modeling approaches

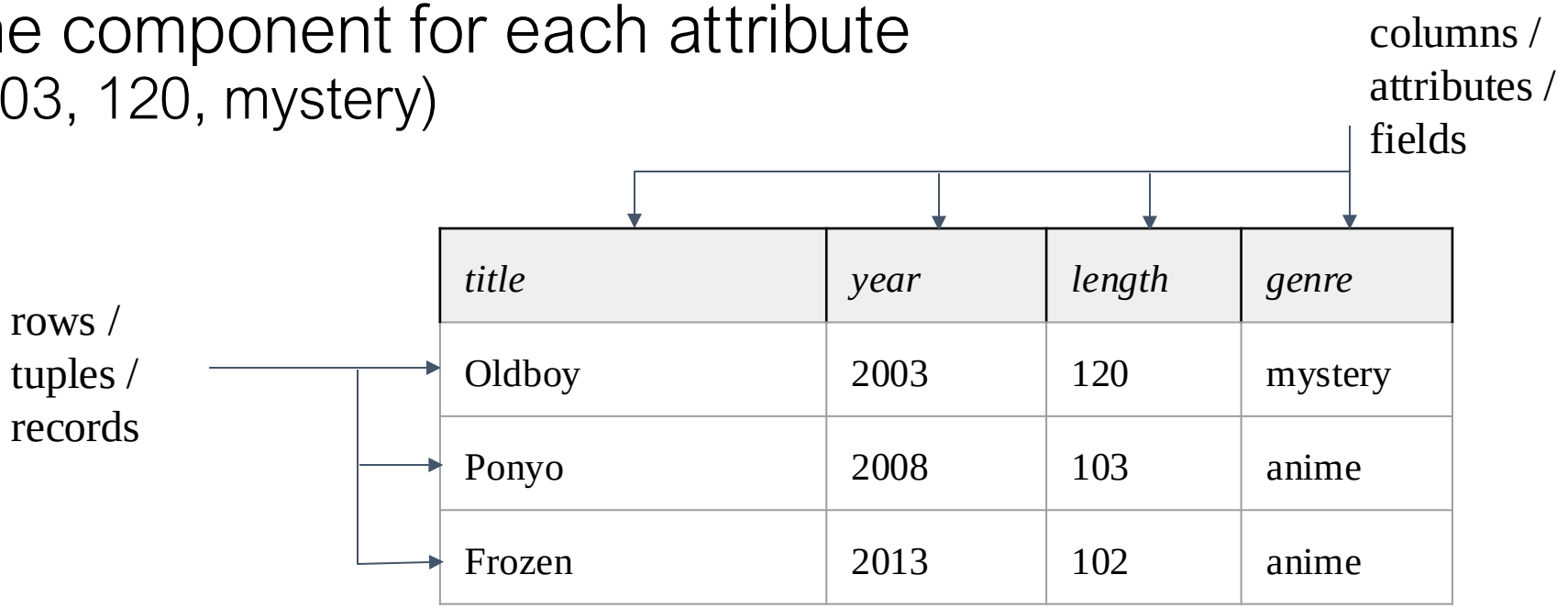
- Relational model
  - Simple and limited, but reasonably versatile
  - Limited, but useful operations
  - Efficient access to large data
  - A few lines of SQL can do the work of 1000's of lines of C code
  - Preferred in DBMS's
- Semi-structured model
  - More flexible, but slower to query
- Key-value model
  - Even more flexible, but cannot query

Basics of the relational model

- A database is a collection of relations (or tables)
- Each relation has a set of attributes (or columns)
- Each attribute has a name and a domain (or type)
  - Set-valued attributes are not allowed (e.g., you cannot store a list/set of bars in a cell, all cells have to contain atomic values)
- Each relation contains a "set" of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Ordering of rows doesn't matter (even though output is always in some order)

Basics of the relational model

- Relation: two-dimensional table containing data
- Schema: relation name and set of attributes
  - Movies(title, year, length, genre)
- Database schema: set of schemas for the relations of a database
- A tuple has one component for each attribute
  - (Oldboy, 2003, 120, mystery)



title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime

Equivalent representations of a relation

- A relation is a set of tuples (not a list)
- A schema is a set of attributes (not a list)
- Hence, the order of tuples or attributes of a relation is immaterial

title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime

=

year	*genre*	*title*	*length*
2013	anime	Frozen	102
2003	mystery	Oldboy	120
2008	anime	Ponyo	103

In-class exercise

How many ways are there to represent this relation?

title	*year*	*length*	*genre*
Oldboy	2003	120	mystery
Ponyo	2008	103	anime
Frozen	2013	102	anime