CS 6400 A

Database Systems Concepts and Design

Lecture 1

08/24/26

Agenda

1. Course logistics and overview

2. Why study relational databases?

3. Relational data model

The essentials

Instructor: Kexin Rong

TAs:

- Jixian Su
- Yixuan Zheng
- Shrey Bansal

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/fa26-cs6400

schedule, course material, and general course information

Canvas/Gradescope: submitting assignments

Piazza: discussing course contents and finding teammates

- https://piazza.com/class/msx8rg9jep94q0

Email: special requests

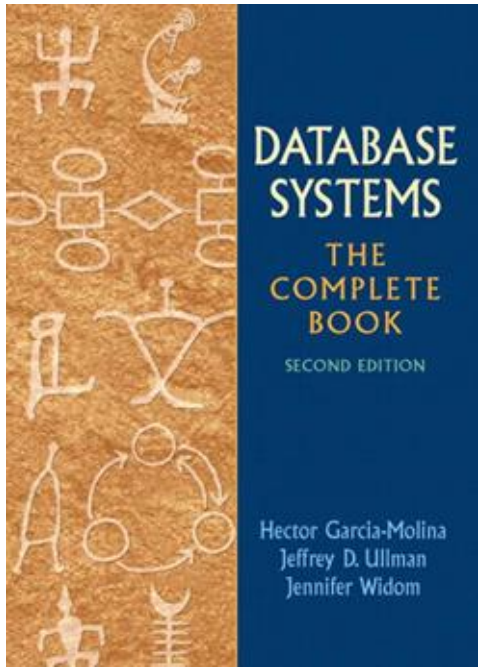
OH: Starting next week. Time will be announced.

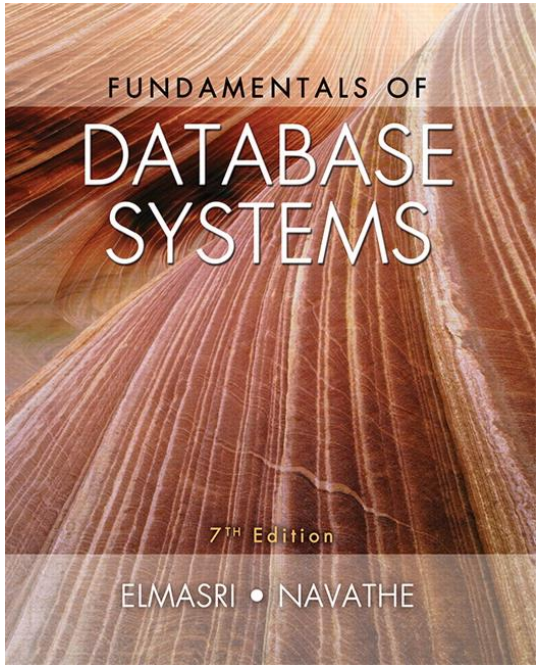
Course materials

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 – 40%

- Individual, mostly programming

Course Project – 25%

- Team-based

Exams – 35%

- Midterm (in-class) – 20%
- Quizzes (in-class) – 15%
  - Best 2 out of 3

Grading Scale:

- A: >=90%
- B: [80%, 90%)
- C: [70%, 80%)
- D: [60%, 70%)

Details: https://kexinrong.github.io/fa26-cs6400/grading/

Assignments

Assignment 1: JSON to SQLite (10%)

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

Assignment 2: Text to SQL (15%)

- Prompt engineering – how to get LLMs to generate the right SQL?
- Programming language: Python and SQL

Assignment 3: In-memory Data Layout (15%)

- Storage and access methods
- Programming language: Java

Exams

Written tests based on material covered in lectures, assignments and readings

Midterm (20%):

- in-class (Sep 23)
- NO make up exams!

Quiz (15%):

- Three in-class quizzes (Oct 19, Nov 11, Nov 30)
- We will take the best 2 out of 3.

Course Project

Teams of 4 people

Timeline:

- W6: Proposal (5%)
- W11: Milestone (5%)
- W16: Final report and code (15%)

Project options:

- Option 1: RAG-based question answering
- Option 2: Replicate research

Attendance

NO mandatory attendance.

Lectures will be recorded and posted on Canvas

That said, this class is designed for in-person learning.
So come join us if you can!

Course Policy - IMPORTANT

Follow the Georgia Tech Honor Code!

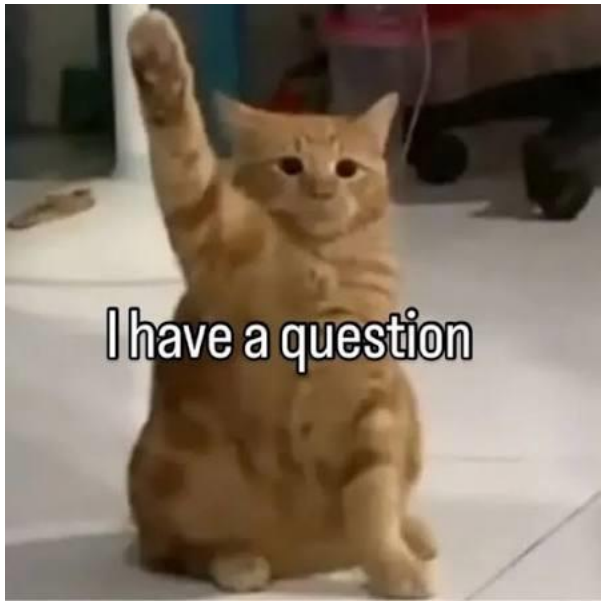
Late policy: One automatic late day without penalty. Otherwise 10% deduction per 24 hours. Does not apply to projects and exams.

Makeup exam policy: No makeup exam for midterm.

Generative AI policy: Check assignment instructions on whether AI use is allowed; for writing, you can use AI to polish but not generate new content that you submit as your own.

Details: https://kexinrong.github.io/fa26-cs6400/policy/

Questions?



Why study relational databases (in 2026)?

Why study databases in 2026?

Almost every important application is data-intensive

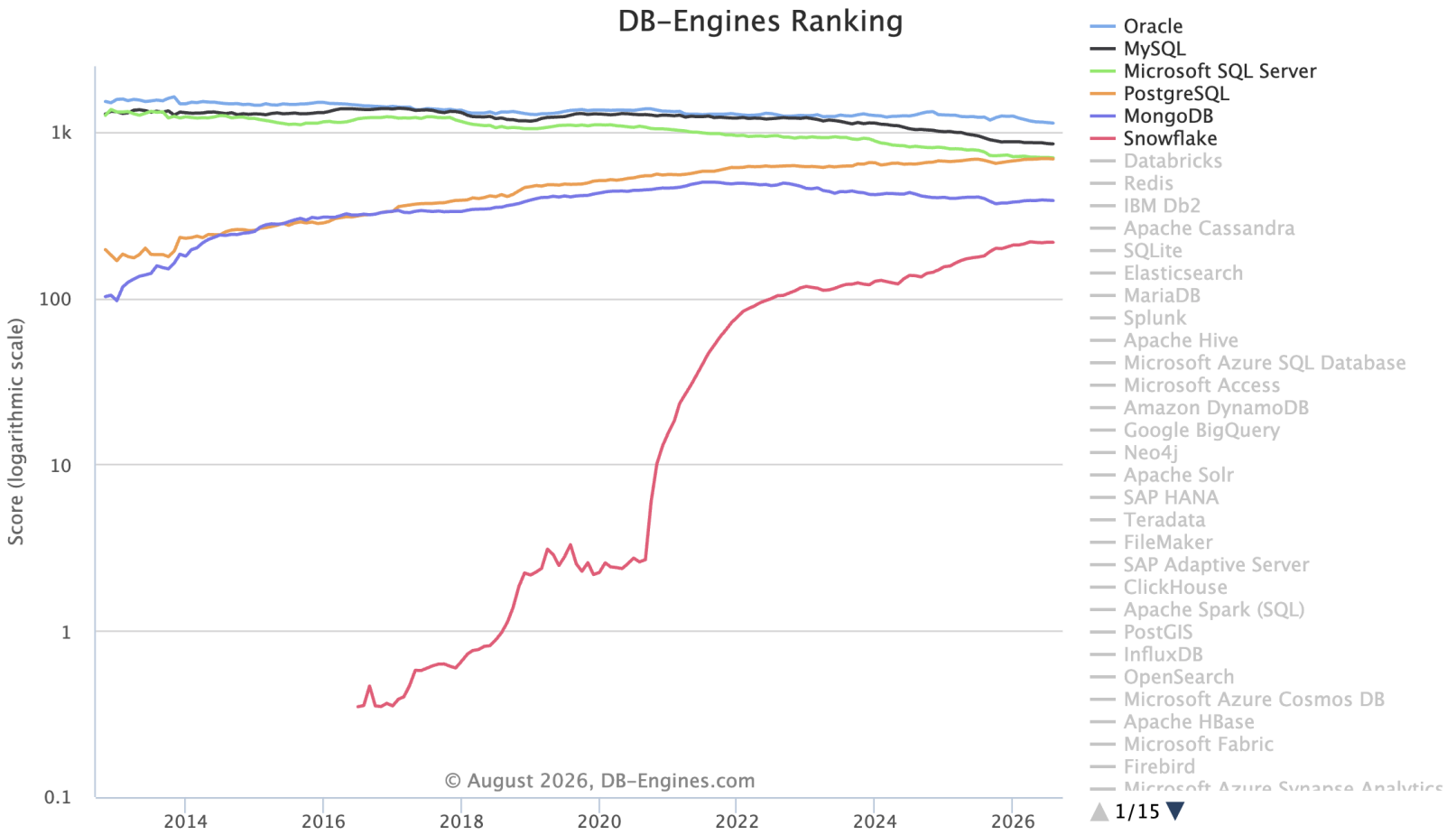
- Banking, e-commerce, social media, search, healthcare, …
- And increasingly: AI applications and agents
- They all need to **store, query, update, and share data**

AI makes data management more important, not less.

Why study *relational* databases?

Relational databases are still the most popular type of data-intensive system

- 5 out of the top 6 are relational databases



Why study *relational* databases?

The same challenges tackled by relational databases appear in many other systems:

- key-value stores
- streaming systems
- compound-AI systems
- your custom applications?

Goal:

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

What makes a relational database interesting?

The promise:

- Tell the database what you want.
- Let the database figure out how to do it.

The hard part (handled by the DB):

- **Scale:** manage data much larger than memory
- **Performance**: answer query efficiently
- **Reliability:** crashes shouldn't lose your data
- **Concurrency**: many users can safely access the same data
- **Abstraction:** applications don't care how data is physically stored
- **Security** and data privacy (not covered in this course)

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?

Many types:

- Customers who purchase books
- Book Sellers
- Database admin (assuming they own all copies of all the books)
- Logistics/warehouse staff
- …

Let’s focus on database admin and customers for now.

Q2: What should customers 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. View previous orders
5. …

Q3: What should the platform do?

i.e., what happens behind the scene

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

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)

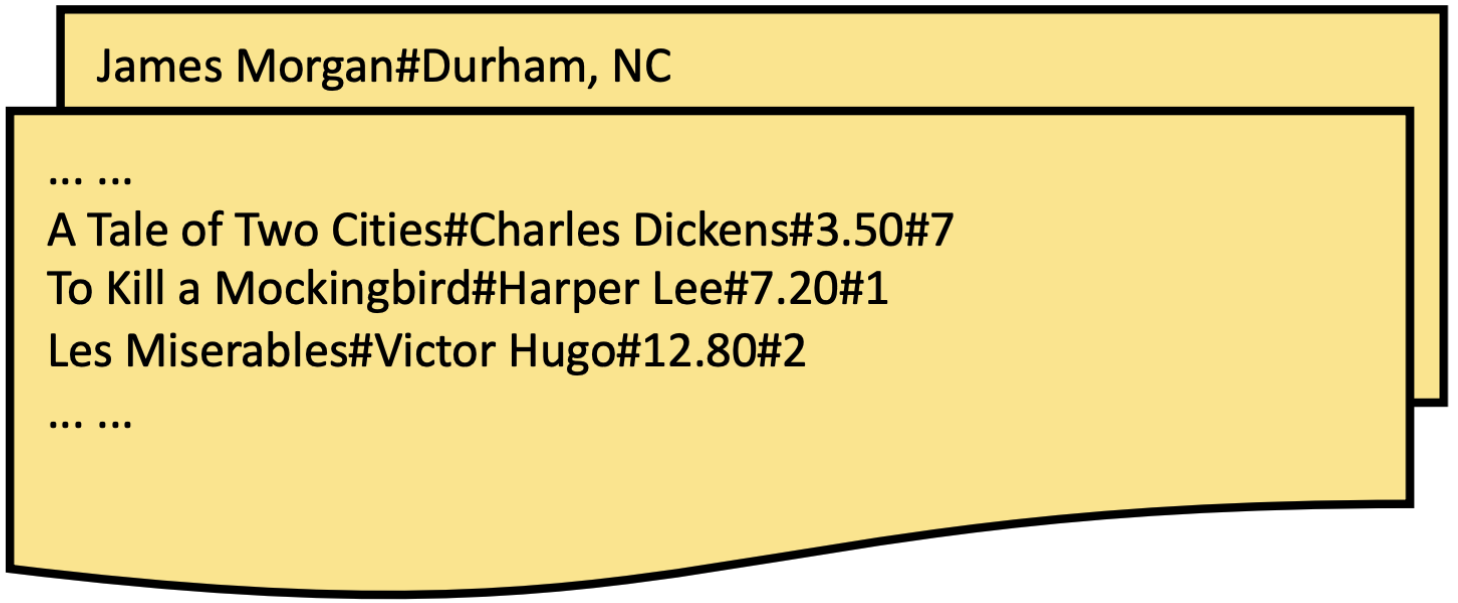


Let’s implement this!

How about:

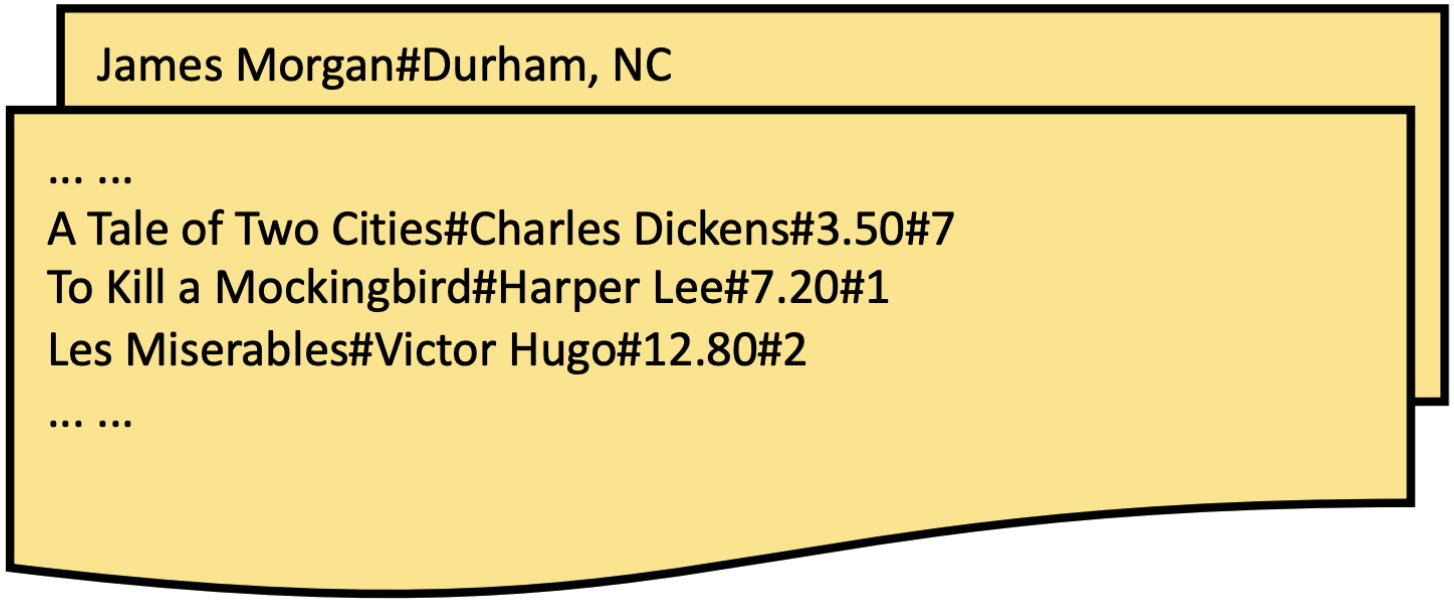
- Your favorite programming language
- On data stored in large files

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 programmer's task
  - 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 — Imagine adding a new book or updating copies (+ allow search) on a 10-100 GB text file
  - For the admin

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!*

This course gives an (advanced) intro to DBMS

1. How to 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 optimization
- Transactions: recovery and concurrency control

3. Glimpse of advanced topics and other data-intensive systems

- MapReduce, Spark, NoSQL
- Parallel and distributed DBMS

Should I take this class?

The class does NOT assume prior background in databases.

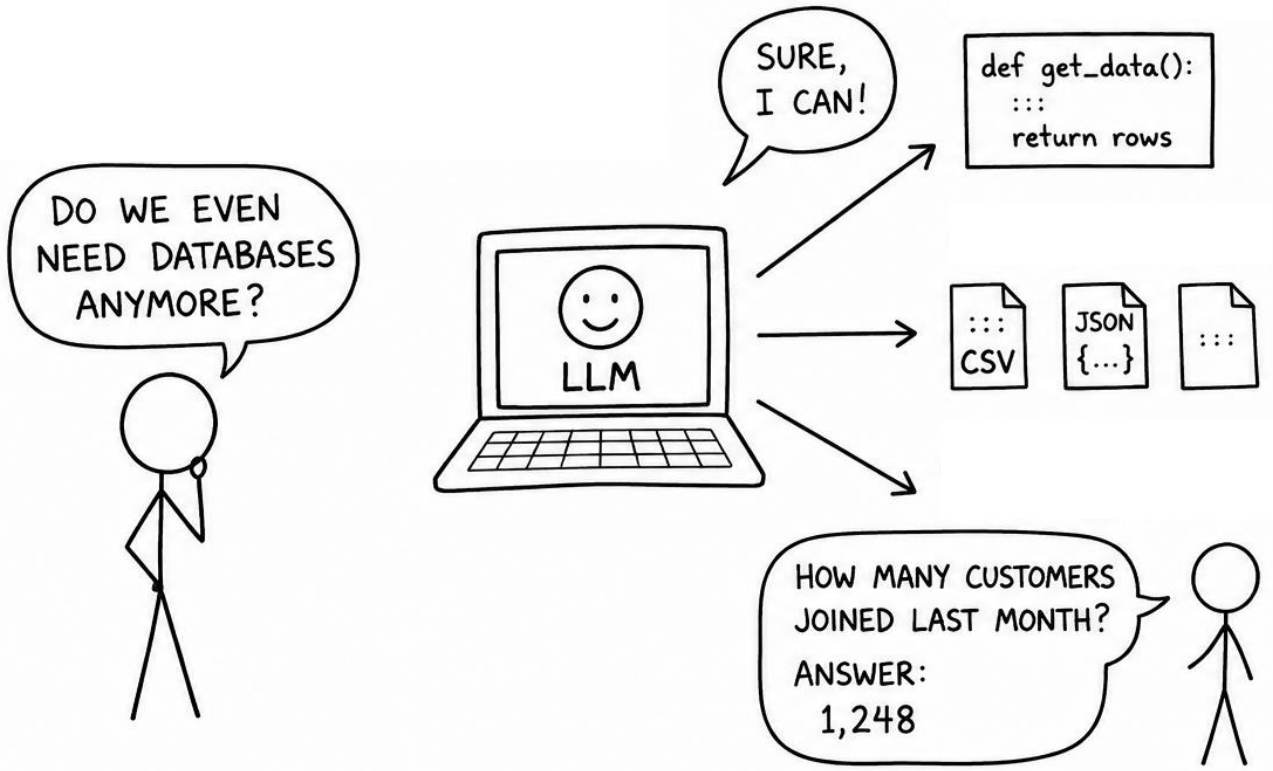
You are expected to be comfortable programming in languages such as Python and Java, but not in SQL.

Check out our syllabus - if you have sufficient undergraduate database coursework, consider taking *CS6422/6423: (Advanced) Database System Implementation* instead

Are LLMs Going to Make DBMSs Obsolete?

A fair question in 2026:

an LLM can write your code, read your files, and answer questions in English



Why an LLM is not a DBMS

Scale: context windows hold megabytes; your data is terabytes.

Exactness: “how many orders shipped yesterday?” must be correct, and give the same answer twice.

Durability & concurrency: an agent that charges a card and decrements inventory still needs transactions.

Governance: access control, auditing, and constraints live in the DBMS, not in the model.

What LLMs do change

How we interact with data

- Relational model (Codd, 1970): say **what** you want, not **how** to get it.
- Natural language raises this one level further.
- Users can ask complex questions without SQL or programming.

What counts as data:

- Text, documents, images, logs, and other unstructured data become easier to query and combine.

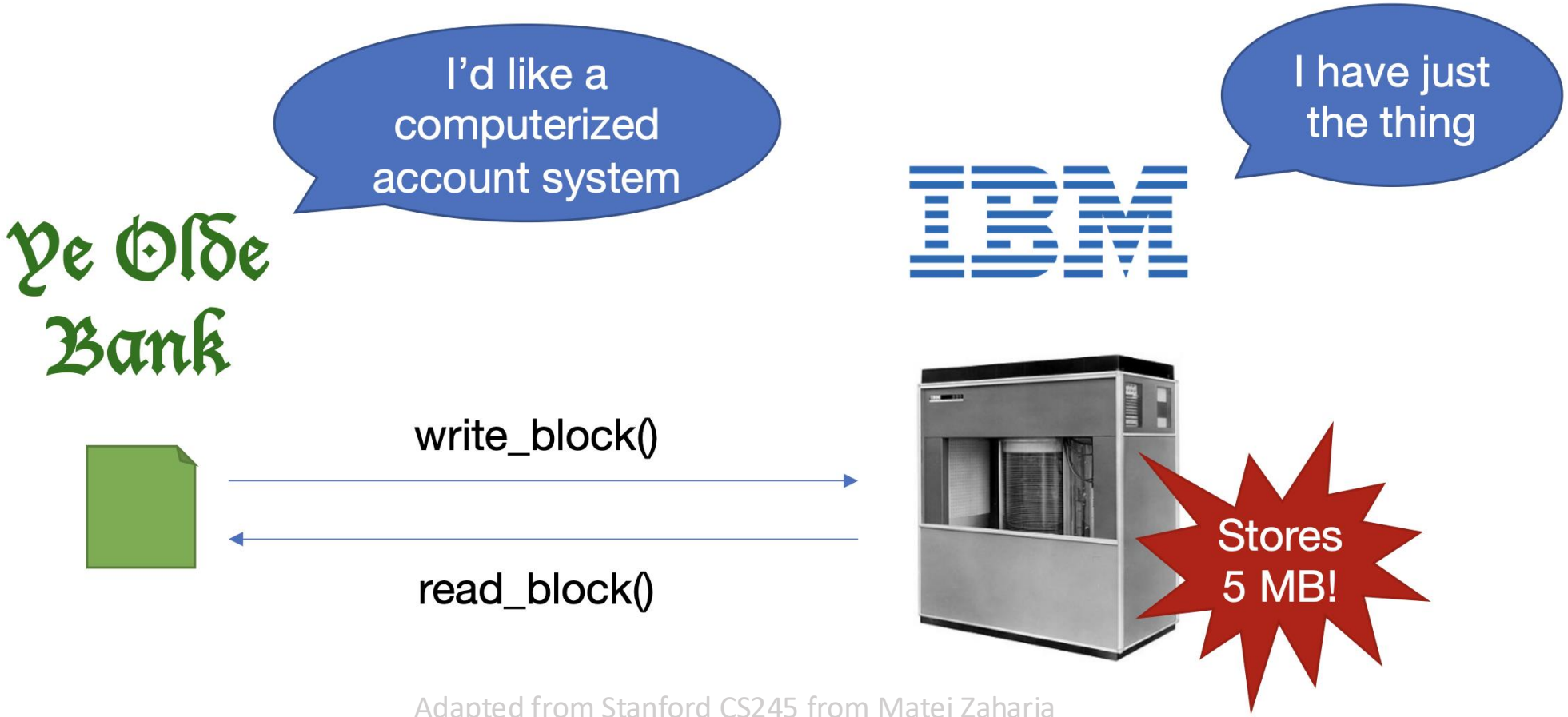
What database systems need to support:

- New data types: embeddings and vector similarity search
- Agents as users: many autonomous clients reading and writing at once.
- LLM calls inside queries: a very expensive new operator to optimize.

Relational Data Model

Early Data Management

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



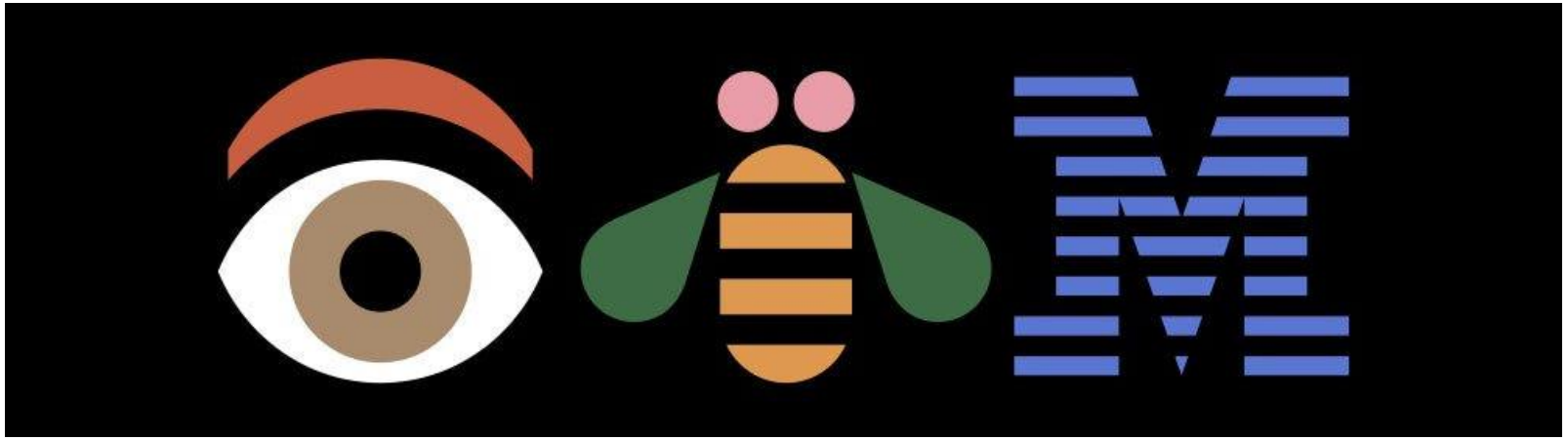
Problems with App Storage Management

- 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.

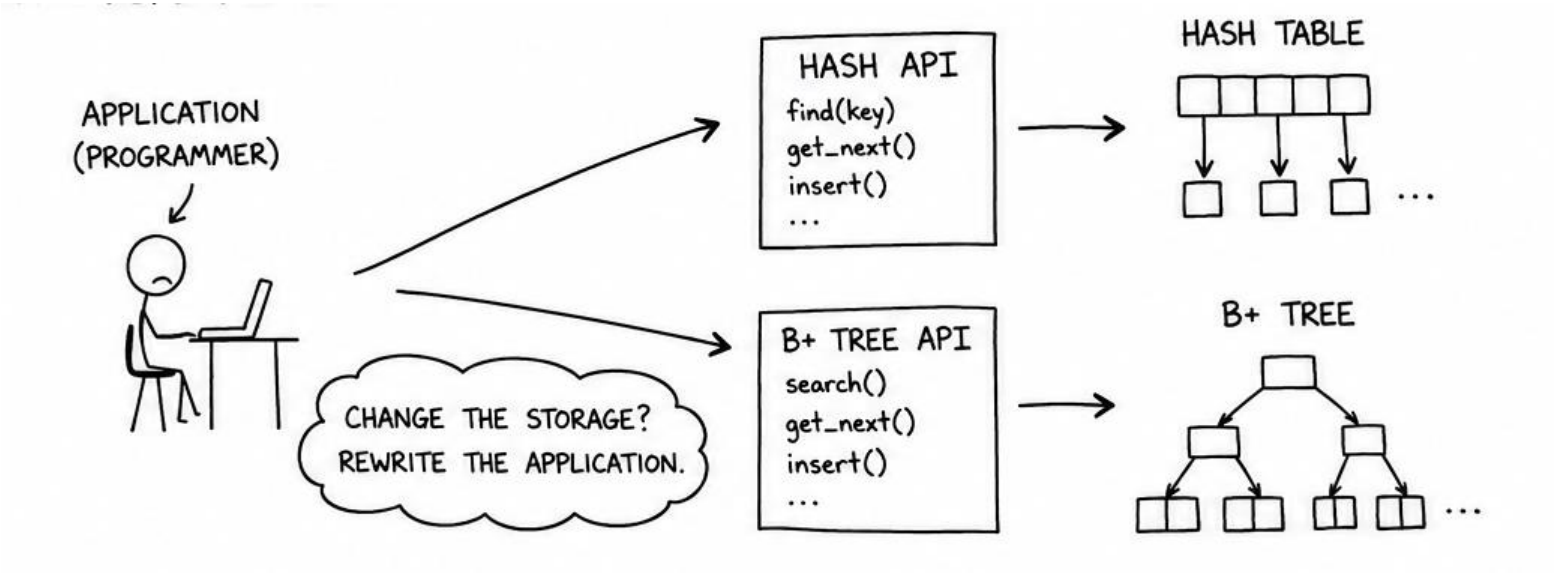


Acknowledgement: Prof. Andy Pavlo, CMU

Programmer-defined physical storage format

In IMS, applications depend on how data is physically organized

- Programmer chooses the storage/access structure
  - e.g., sequential file, hash table, B+ tree
- Each structure exposes a **different way to navigate the data**
- Change the physical layout → **change the application code**



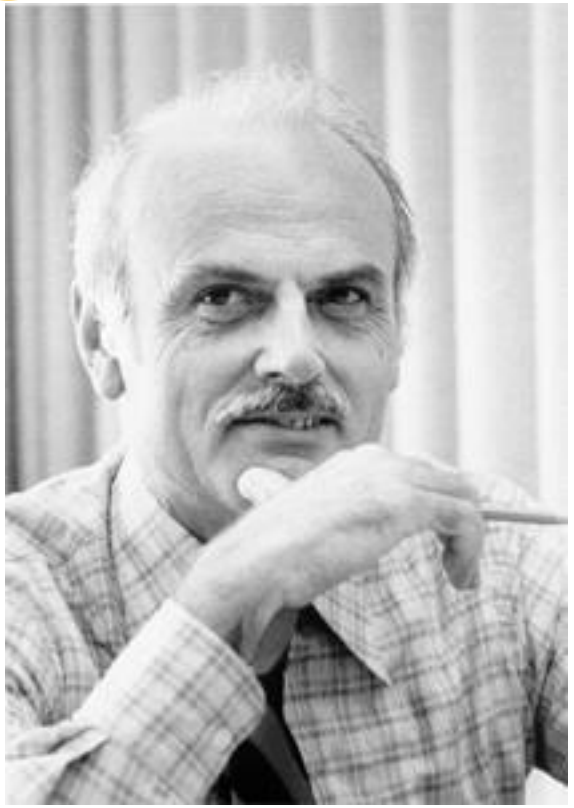
1970s - Relational data model



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.

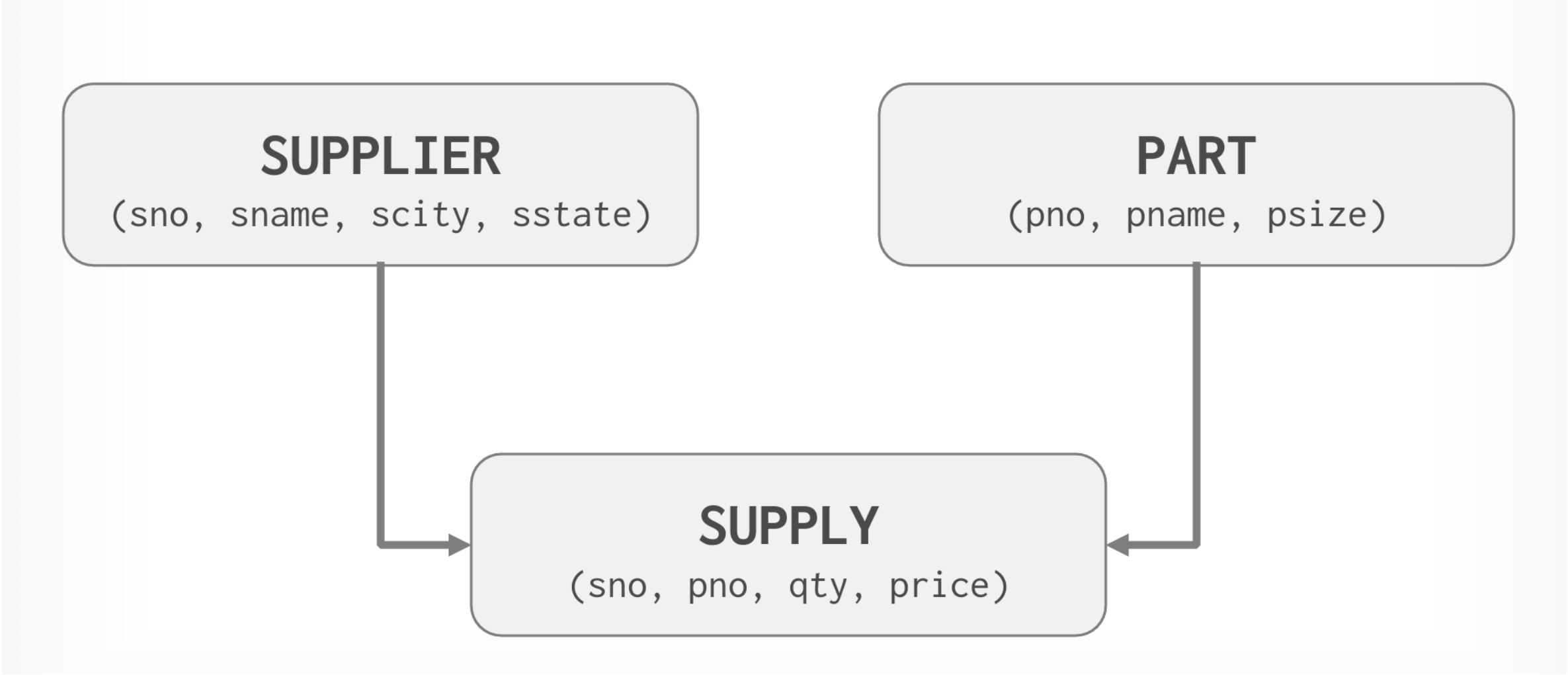
- Database abstraction to avoid this maintenance:
  - Store database in simple data structures.
  - Access data through set-at-a-time high-level language.
  - Physical storage left up to implementation.



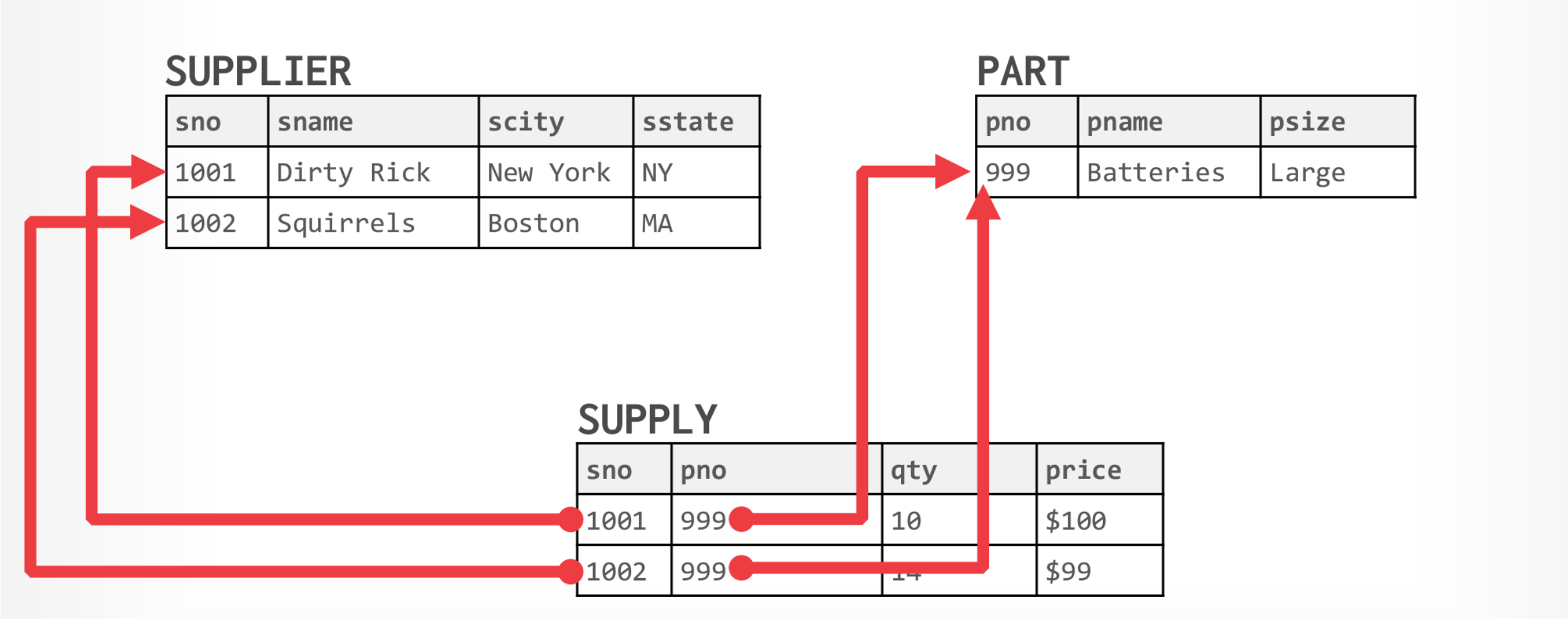
Codd

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?

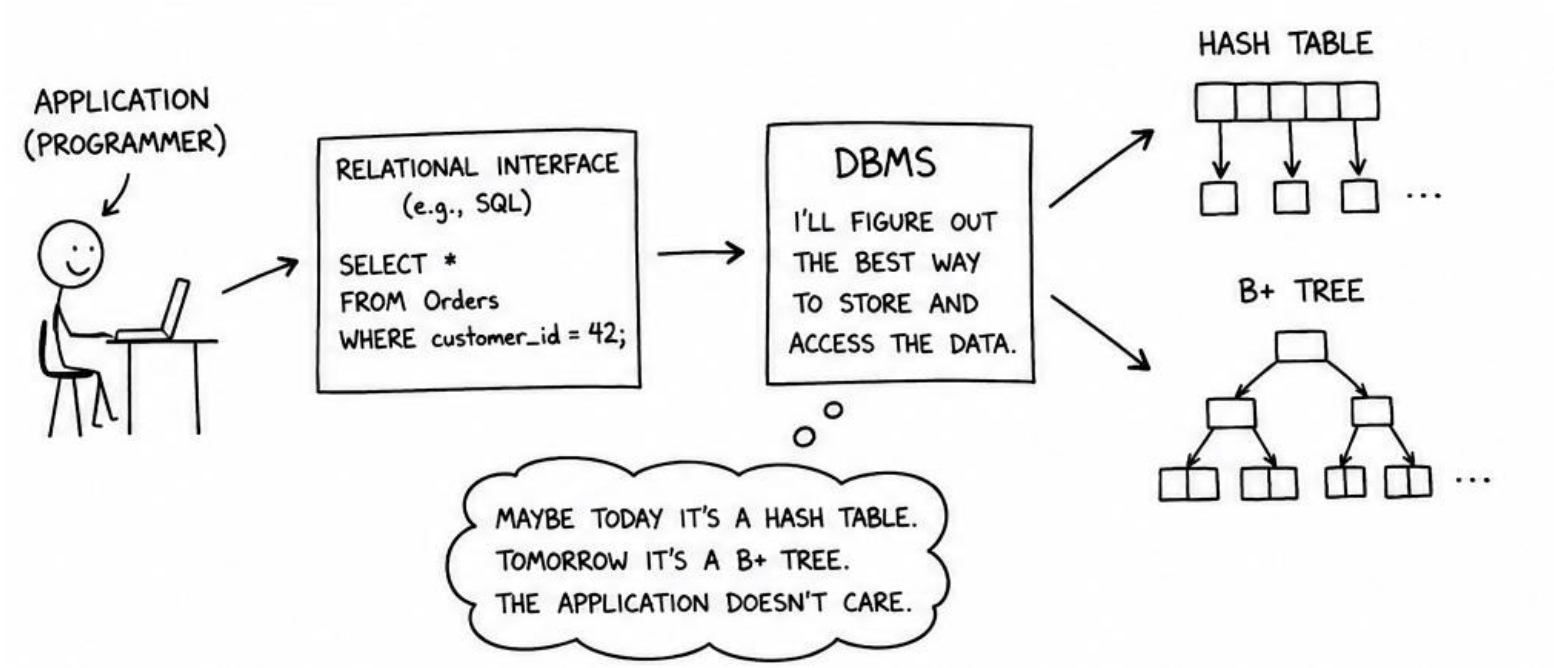
Physical data independence:
protection from *physical layout changes*

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 independence

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



One of the most important reasons to use a DBMS

Data model

A notation for describing data or information

Consists of:

- **Structure**: how is data represented?
- **Operations**: what can we do with it?
- **Constraints**: what rules must the data satisfy?

Data models

- **Relational** → Most DBMS's

- Key/Value
- Graph
- Document (Semi-structured)
- Column-family

} NoSQL

- Array/Matrix/Vector → AI / ML

- Hierarchical
- Network

} Obsolete

The relational model

- Structure
  - ”Conceptual model” of data: 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

- Queries and modification
- Defined by relational algebra
- This is a strength, not a weakness

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

Constraints

- Usually have limitations on the data; helpful for data quality
- E.g., Genre must be one of a fixed list of values,
  no two movies can have the same title

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 data models

Let's compare the relational model and the key-value model in the following aspects. Which one is better?

- Flexibility? Key-value
- Queryability? Relational
- Performance? Relational