

Welcome to

CS 660: Grad Intro to Database Systems

<https://bu-disc.github.io/CS660/>

Instructor: *Manos Athanassoulis*
email: *mathan@bu.edu*

CS660: Course Overview

Design & implementation of Database Management Systems (DBMSs).

We **do not** study how to build applications on top of a DBMS.

We **do** study the internals of a DBMS.

We have a **C++ based project**; we assume you have prior C++ knowledge.

→ that is, while we will offer help we will not teach C++ in the labs

Course Summary

We will learn how to:

- **Model data**

Relational Model

- **Access and Query data**

SQL

- **Store & manage data**

Bits to Files to Disks, Storage Layouts, Indexes, Sorting

- **Reason about query performance**

Query evaluation & optimization

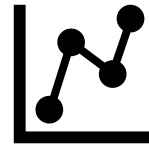
- **Update data**

Transactions, logging, ACID properties

Today: Why Study Databases?

big data

data-driven world



databases & database systems



when you see this, I want you to
speak up!
[and you can always interrupt me]

How big is “Big”?



Every day, we create 2.5 exabytes* of data — 90% of the data in the world today has been created in the last two years alone.

[Understanding Big Data, IBM]

*exabyte = 10^9 GB



CS660

we live in a *data-driven* world

CS660 is about the *basics* for
storing, using, and managing data

your lecturer (that's me!)

Manos Athanassoulis

name in greek: Μάνος Αθανασούλης

grew up in Greece

enjoys playing basketball and the sea

BSc and MSc @ University of Athens, Greece

PhD @ EPFL, Switzerland

Research Intern @ IBM Research Watson, NY

Postdoc @ Harvard University

Visiting Faculty @ Meta

some awards:

Facebook Faculty Research Award

NSF CAREER Award

Best Demo @ VLDB 2023

Best of SIGMOD 2017, VLDB 2017



photo for VISA / conferences



Myrtos, Kefalonia, Greece

<http://cs-people.bu.edu/mathan/>

Office: CDS928

Office Hours: Tue @ 10am and Thu @2pm

your awesome TFs

Head TF

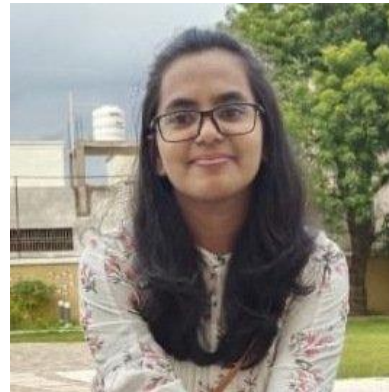


Aneesh Raman

PhD researcher in DB

aneeshr@bu.edu

OH: M & W@11am-12pm

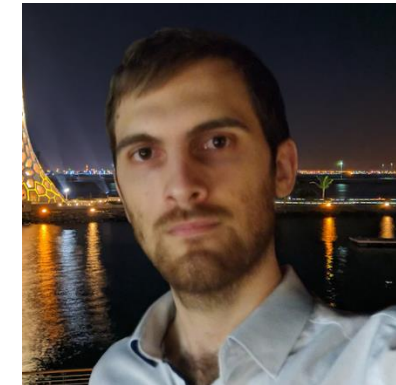


Sakshi Sharma

MS researcher in DB

phsakshi@bu.edu

OH: Tu@4-5pm



Kostas Karatsenidis

PhD researcher in DB

karatse@bu.edu

OH: W@3-4pm

Data

to make data **usable** and **manageable**

we organize them in **collections**

Databases

a large, integrated, *structured* collection of data

intended to model some real-world enterprise

Examples: a university, a company, social media

Social media: users, posts, photos

what is missing?

-- how to connect these?

-- shares, likes, friend-relationship



Database Systems

a.k.a. database management systems (DBMS)

a.k.a. data systems



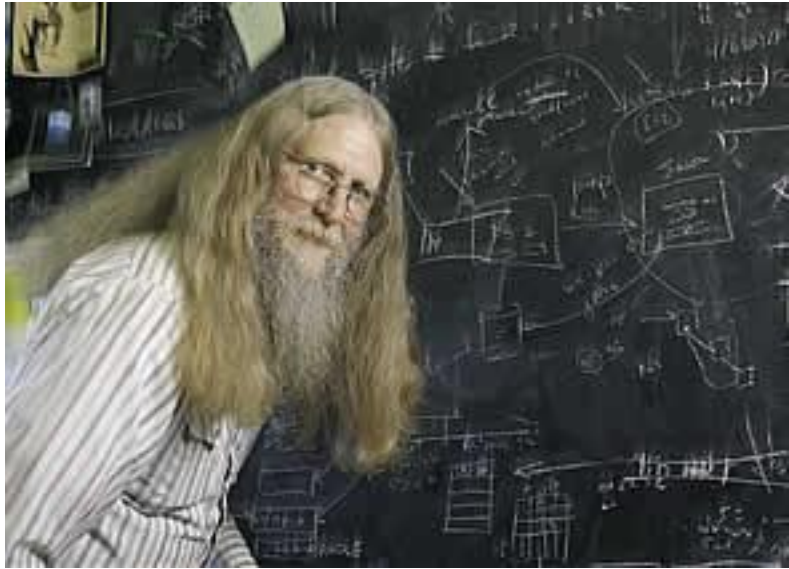
Sophisticated
pieces of software...



... which store, manage,
organize, and facilitate
access to my databases ...



... so I can do things (and ask questions) that are
otherwise hard or impossible



*“relational databases
are the foundation of
western civilization”*

Bruce Lindsay, IBM Research

ACM SIGMOD Edgar F. Codd Innovations award 2012

Ok but what really IS a database system?

Is the Internet a DBMS?



Is a File System a DBMS?



Are Social Media a DBMS?



Is the Internet a DBMS?

Not really!

Fairly sophisticated search available

web crawler *indexes* pages for fast search

.. but

data is unstructured and untyped

not well-defined “correct answer”

cannot update the data

freshness? consistency? fault tolerance?

web sites **use** a *DBMS* to provide these functions

e.g., amazon.com (Oracle), facebook.com (MySQL and others)

Is a File System a DBMS?



Thought Experiment 1:

- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?



A) Yours **B) Partner's** **C) Both** **D) Neither** **E) ???**



Is a File System a DBMS?



Thought Experiment 1:

- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?



A) Yours **B) Partner's** **C) Both** **D) Neither** **E) ???**

Thought Experiment 2:

- You're updating a file.
- The power goes out.
- Which of your changes survive?



A) All **B) None** **C) All Since last save** **D) ???**



Is a File System a DBMS?

Not really!



Thought Experiment 1:

- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?



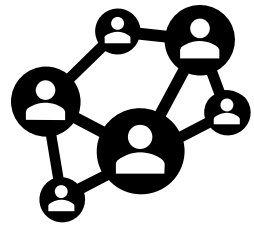
A) Yours **B) Partner's** **C) Both** **D) Neither** **E) ???**

Thought Experiment 2:

- You're updating a file.
- The power goes out.
- Which of your changes survive?



A) All **B) None** **C) All Since last save** **D) ???**



Are Social Media a DBMS?

Is the data structured & typed?

Does it offer well-defined queries?

Does it offer properties like “durability” and “consistency”?

For example, Facebook is a data-driven company that uses several database systems (>10) for different use-cases (internal or external).



Not really!

Are Large Language Models (LLMs) Databases?

What happens if I ask the same query multiple times?



Not really!

How does it get updated?

Does it offer properties like “durability” and “consistency”?



What may be a good use-case for LLMs in data management?

Why take this class?

shift from computation to information

corporate, personal (web), science (big data)

database systems everywhere

data-driven world, data companies

DBMS: much of CS as a practical discipline

languages, theory, OS, logic, architecture, HW

CS660 in a nutshell

model

data representation model

query

query languages – ad hoc queries

access (concurrently multiple reads/writes)
ensure *transactional* semantics

store (reliably)

maintain *consistency/semantics* in *failures*

A “free taste” of the class

data modeling

query languages

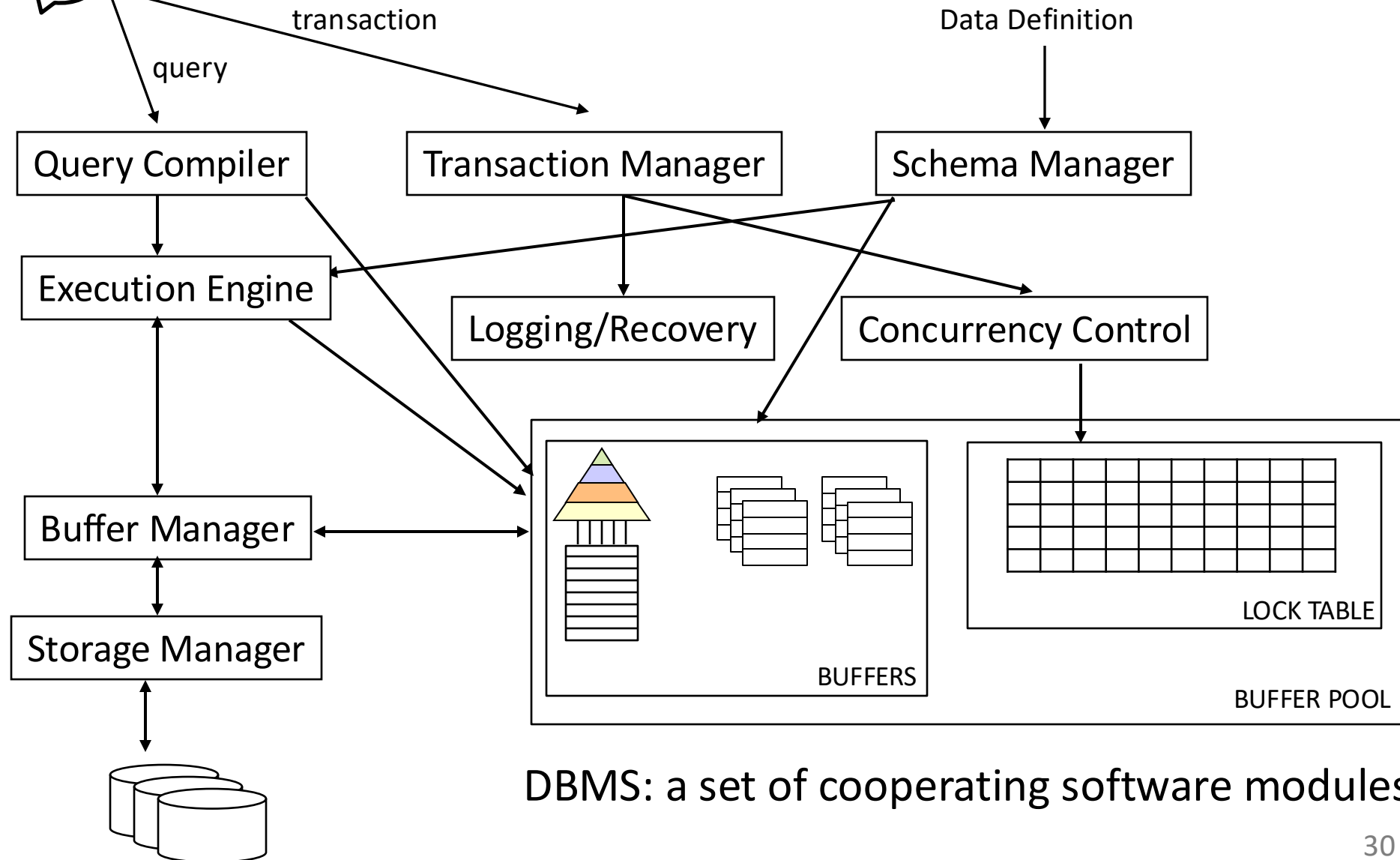
concurrent, fault-tolerant data management

DBMS architecture

Coming in next class

Discussion on *database systems designs*

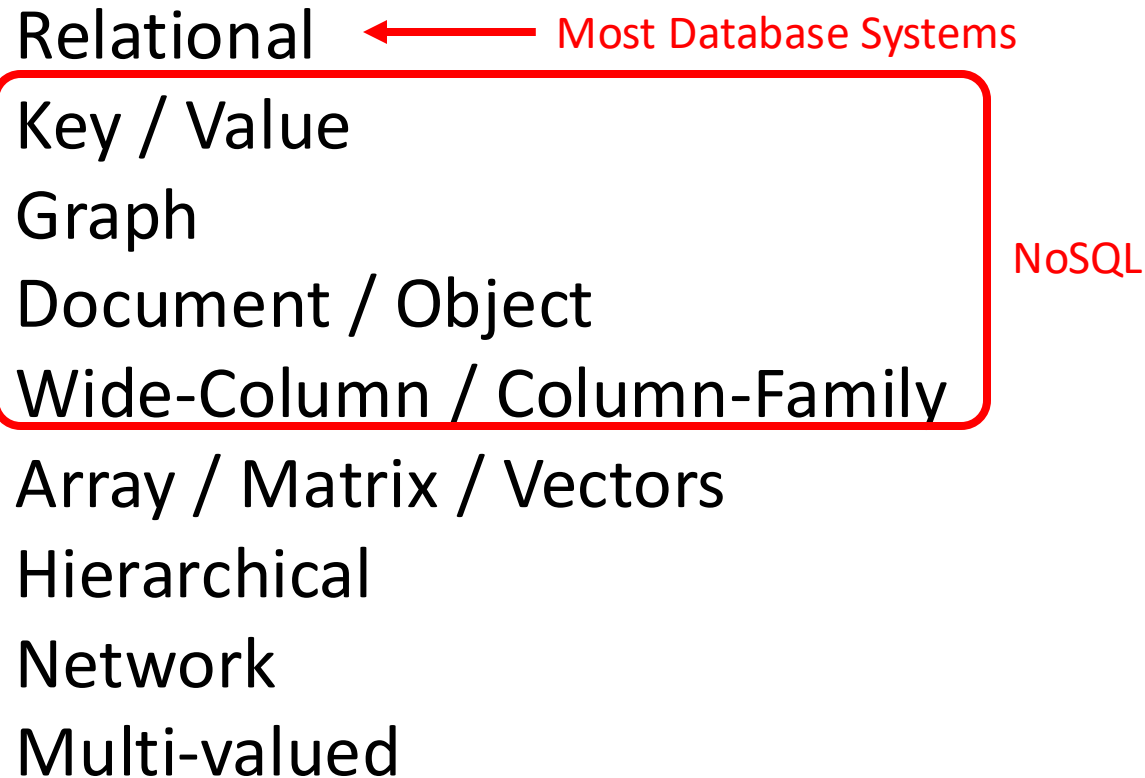
Components of a "classic" DBMS



DBMS: a set of cooperating software modules

Describing Data: Data Models

data model : a collection of concepts describing data



Relational

tables with rows and columns

well-defined schema

data model fits data rather than
functionality

deduplication

Key/Value

collections of documents

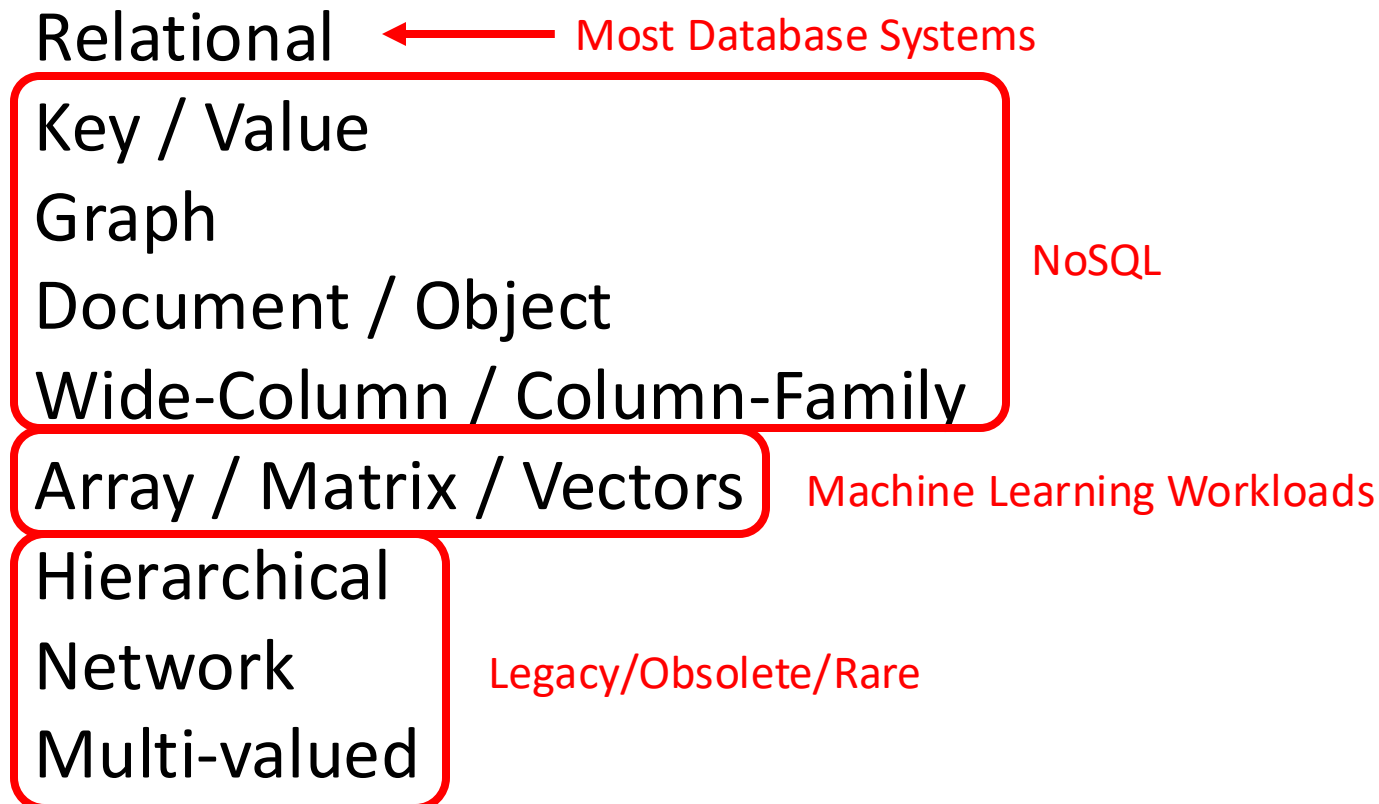
schema-less (each document can
have different schema)

data stored in an application-
friendly way

possible duplication

Describing Data: Data Models

data model : a collection of concepts describing data



Relational Model: Definitions

relational database: a collection (set) of *relations*

each relation: basically a table with rows and columns, made up of 2 parts

(1) *schema*: describes the columns (or fields) of each table

Name of relation → **Students** (*sid*: string, *name*: string, **login**: string, *age*: integer, *gpa*: real)
 ← Name and type of each column

(2) *instance* : a *table*, with rows and columns.

#rows = *cardinality*

#fields = *degree / arity*

a relation is a *set* of *tuples* (a.k.a. rows)

(1) all rows are distinct

(2) no order among rows

Schema of “University” Database

Students

sid: string, name: string, login: string, age: integer, gpa: real

Courses

cid: string, cname: string, credits: integer

Enrolled

sid: string, cid: string, grade: string



Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

cardinality = 3, arity = 5, all rows distinct



do all values in each column of a relation instance have to be distinct?

SQL - A language for Relational DBs

SQL* (a.k.a. “Sequel”), standard language

Data Definition Language (DDL)

create, modify, delete relations

specify constraints

administer users, security, etc.

Data Manipulation Language (DML)

specify *queries* to find tuples that satisfy criteria

add, modify, remove tuples

SQL Overview

```
CREATE TABLE <name> ( <field> <domain>, ... )
```

```
INSERT INTO <name> (<field names>)  
VALUES (<field values>)
```

```
DELETE FROM <name>  
WHERE <condition>
```

```
UPDATE <name>  
SET <field name> = <value>  
WHERE <condition>
```

```
SELECT <fields>  
FROM <name>  
WHERE <condition>
```

Creating Relations in SQL

type (**domain**) of each field is specified

also enforced whenever tuples are added or modified

```
CREATE TABLE Students  
  (sid CHAR(20),  
   name CHAR(20),  
   login CHAR(10),  
   age INTEGER,  
   gpa FLOAT)
```


Table Creation (continued)

Enrolled: holds information about courses students take

```
CREATE TABLE Enrolled  
  (sid CHAR(20),  
   cid CHAR(20),  
   grade CHAR(2))
```

Adding and Deleting Tuples

Can insert a single tuple using:

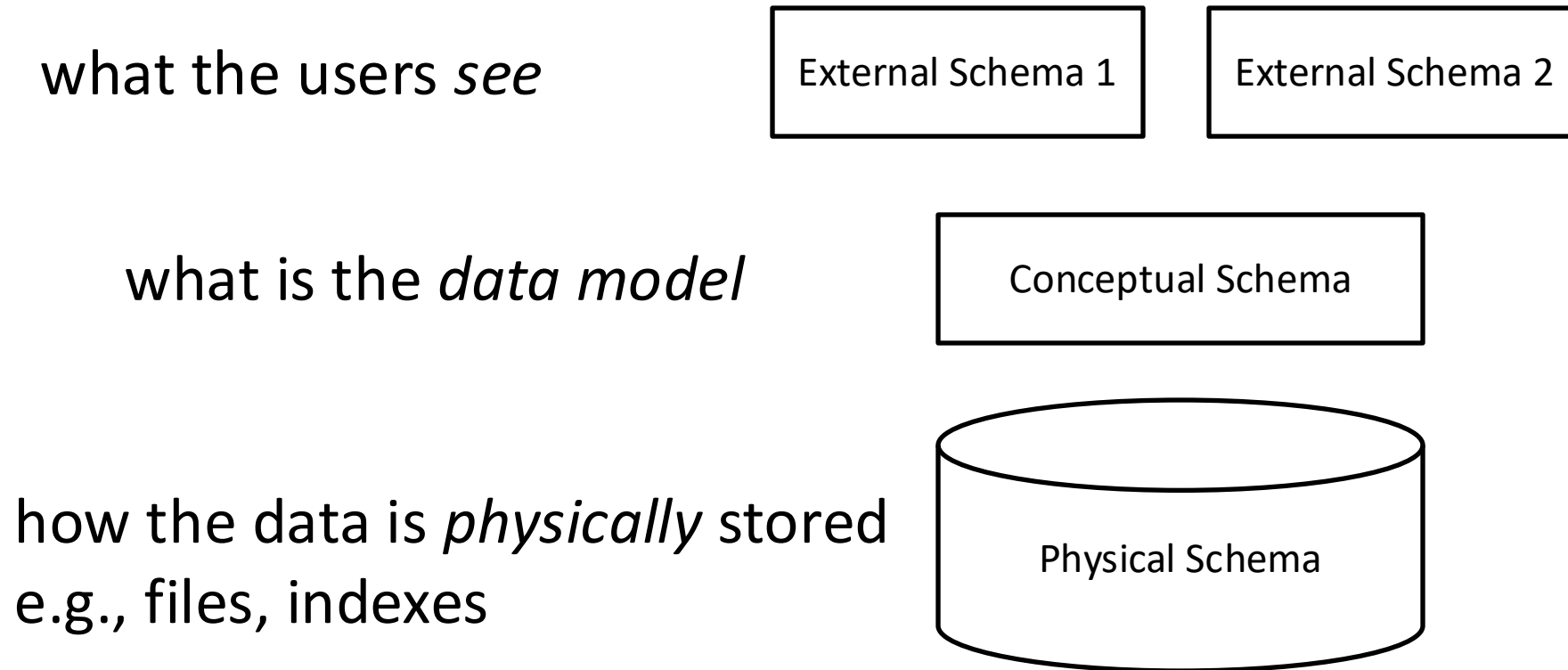
```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@cs', 18, 3.2)
```

Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

Powerful variants of these commands are available;
more later!

Levels of Abstraction



Schemas of “University” Database

Conceptual Schema

Students

sid: string, name: string, login: string, age: integer, gpa: real

Courses

cid: string, cname: string, credits: integer

Enrolled

sid: string, cid: string, grade: string

```
SELECT cid, count(sid) AS enrollment
FROM courses c, enrolled e
WHERE c.cid=e.cid
```



Physical Schema

relations stored in files on disk
indexes on sid/cid for performance

External Schema

through which query?

a “view” of data that can be derived from the existing data

conceptual + Course_Info (cid: string, enrollment:integer)

which **combines** information from Courses & Enrolled

Data Independence

Abstraction offers “application independence”

Logical data independence

Protection from changes in *logical* structure of data

Physical data independence

Protection from changes in *physical* structure of data

Q: Why is this particularly important for DBMS?



Applications can treat DBMS as black boxes!

Queries

”Bring me all students with gpa more than 3.0”

“SELECT * FROM Students WHERE gpa>3.0”

SQL – a powerful *declarative* query language

treats DBMS as a black box

What if we have multiples accesses?

Concurrency Control

multiple users/apps

Challenges



frequent accesses to slow medium

how to keep CPU busy

how to avoid *short jobs* waiting behind *long ones*

e.g., ATM withdrawal while summing all *balances*

interleaving actions of *different* programs

Concurrency Control

Problems with *interleaving* actions of diff. programs



Bill



Move 100 from
savings to checking



Alice

Bad interleaving:

Savings $-= 100$ (from Bill's transaction)

Print balances (from Alice's transaction)

Checking $+= 100$ (from Bill's transaction)

Printout is missing 100\$!

Concurrency Control

Problems with *interleaving* actions of diff. programs



Bill



Move 100 from
savings to checking



Alice

What is a correct interleaving?

Savings -= 100

Checking += 100

Print balances

How to achieve this interleaving?



Scheduling Transactions

Transactions: atomic sequences of **R**eads & **W**rites

$$T_{\text{Bill}} = \{R1_{\text{Savings}}, R1_{\text{Checking}}, W1_{\text{Savings}}, W1_{\text{Checking}}\}$$
$$T_{\text{Alice}} = \{R2_{\text{Savings}}, R2_{\text{Checking}}\}$$

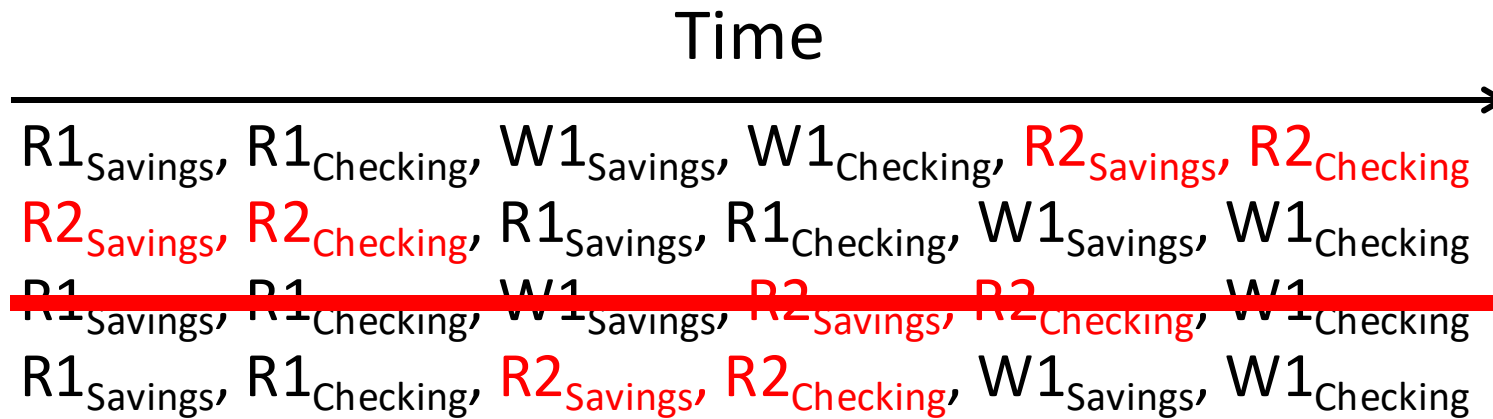
How to avoid previous problems?



Scheduling Transactions

All interleaved executions equivalent to a serial

All actions of a transaction executed as a whole



How to achieve one of these?



Locking



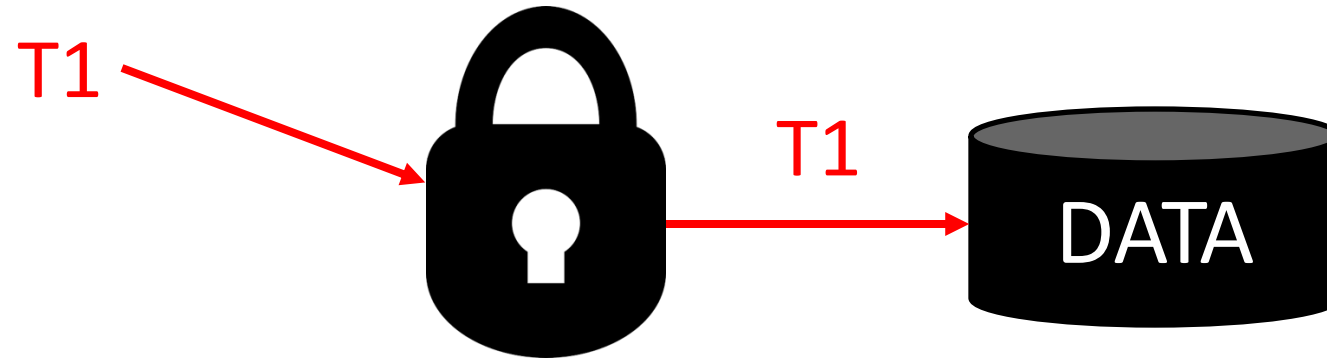
before an object is accessed a lock is requested

Locking



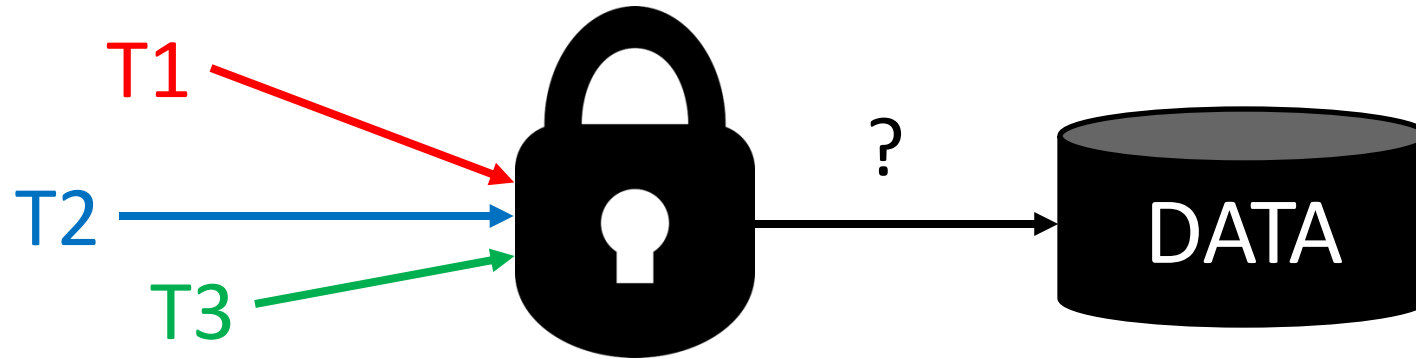
before an object is accessed a lock is requested

Locking



before an object is accessed a lock is requested

Locking



locks are held until the end of the transaction

*[this is only one way to do this, called
"strict two-phase locking"]*

Locking

$$T_1 = \{R1_{Savings}, R1_{Checking}, W1_{Savings}, W1_{Checking}\}$$
$$T_2 = \{R2_{Savings}, R2_{Checking}\}$$

Both should lock *Savings* and *Checking*

What happens:

if T1 locks Savings & Checking ?

T2 has to wait

if T1 locks Savings & T2 locks Checking ?

we have a deadlock



How to solve deadlocks?

we need a mechanism to undo

also when a transaction is incomplete
e.g., due to a crash



what can be an undo mechanism?



log every action before it is applied!

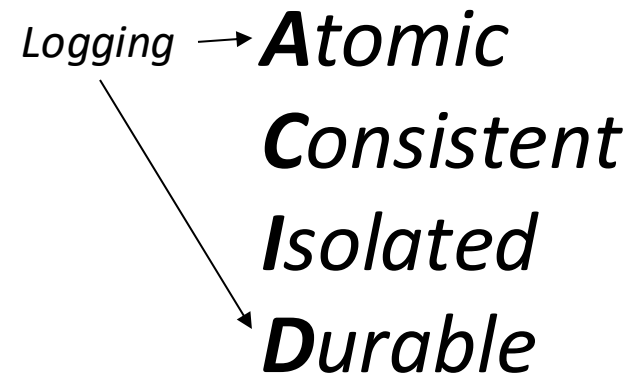
Transactional Semantics

Transaction: one execution of a user program

multiple executions → multiple transactions

Every transaction:

Logging → ***Atomic***
Consistent
Isolated
Durable

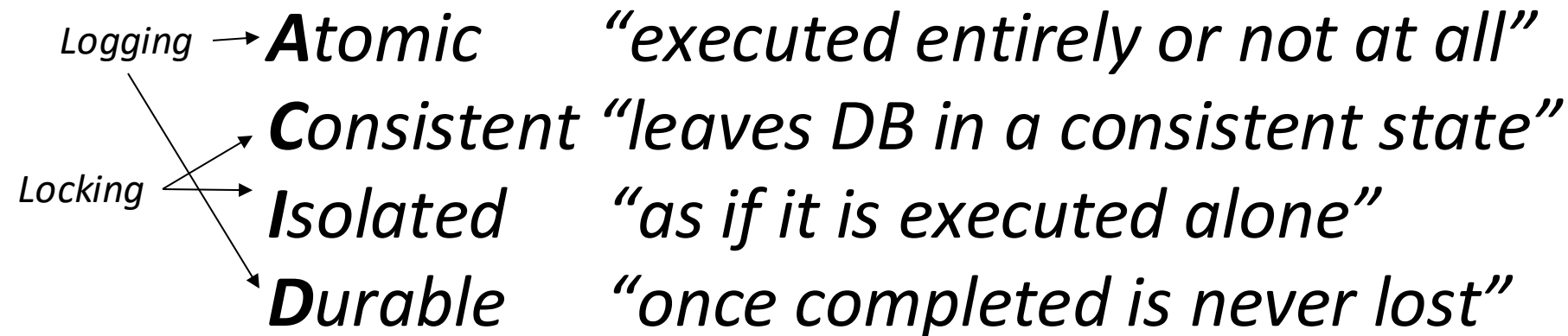
A diagram showing the word "Logging" in italics on the left. Three arrows originate from "Logging": one points horizontally to the word "Atomic" in bold italics; another points diagonally down and to the right to the word "Isolated" in bold italics; and a third points diagonally down and to the right to the word "Durable" in bold italics. The word "Consistent" in bold italics is positioned between "Atomic" and "Isolated".

Transactional Semantics

Transaction: one execution of a user program

multiple executions → multiple transactions

Every transaction:



Who else needs transactions?



lots of data

lots of users

frequent updates

background game analytics

Scaling games to epic proportions,

by W. White, A. Demers, C. Koch, J. Gehrke and R. Rajagopalan

ACM SIGMOD International Conference on Management of Data, 2007

Only “classic” DBMS?

No, there is much more!

NoSQL & Key-Value Stores: No transactions, focus on queries

Graph Stores

Querying raw data without loading/integrating costs

Database queries in large datacenters

New hardware and storage devices

Cloud data management

... many exciting open problems!

<https://bu-disc.github.io/CS660/>

Next time in ...

CS 660: Introduction to Database Systems

Database Systems Architectures

Class administrativia

Class project administrativia

Course Summary

We will learn how to:

- **Model data**

Relational Model

- **Access and Query data**

SQL

- **Store & manage data**

Bits to Files to Disks, Storage Layouts, Indexes, Sorting

- **Reason about query performance**

Query evaluation & optimization

- **Update data**

Transactions, logging, ACID properties

<https://bu-disc.github.io/CS660/>

Additional Accommodations

If you require additional accommodations please contact the Disability & Access Services office at aslods@bu.edu or 617-353-3658 to make an appointment with a DAS representative to determine which are the appropriate accommodations for your case.

Please be aware that accommodations cannot be enacted retroactively, making timeliness a critical aspect for their provision.

You can optionally choose to disclose this information to the instructor.