# CompSci 516
# Database Systems

## Lecture 2
## Data Models
## and
## More SQL

Instructor: Sudeepa Roy

# Announcements - 01/11 (Tues)

- HW1-Part 1 posted on sakai: Resources -> Homeworks -> HW1
  - Publication data, convert XML -> Relational database and store into Postgres, using Python/Java/etc.
  - Remember: You have to learning these frameworks yourself with online material, and some help from TAs
  - Start working on it!
  - Part 2 will have SQL queries and data analysis
  - Both parts due on 01/27/2022 (Thursday)
- Office hour times + Zoom link posted on Ed
- Threads for project teams posted on Ed
  - If you are looking for teammates or a team, please post
- If you have any question about the class before the add/drop date, let me know after class
  - Recording will stop at 11:30 am

# Review of Database Systems
# “What” and “Why”

In this course

## “How to use”
## “How it works”

# What is a Database?

- A database is a collection of data
  – typically related and describing activities of an organization
- A database may contain information about
  – Entities
    - students, faculty, courses, classroom
  – Relationships between entities
    - students' enrollment, faculty teaching courses, rooms for courses

# Revisit: Why use a DBMS?

- Recall the book-selling-platform exercise!

- Some nice properties of a DBMS?

# Revisit: Why use a DBMS?

- A DBMS is a piece of software (i.e., a big program written by someone else) that makes these tasks easier
  – Quick access
  – Robust access
  – Safe access
  – Simpler access

# Why use a DBMS?

1. Data Independence
   - Application programs should not be exposed to the data representation and storage
   - DBMS provides an abstract view of the data

2. Efficient Data Access
   - A DBMS utilizes a variety of sophisticated techniques to store and retrieve data (from disk) efficiently

# Why use a DBMS?

3. Data Integrity and Security
   - DBMS enforces “integrity constraints” – e.g.  check whether total salary is less than the budget
   - DBMS enforces “access controls” – whether salary information can be accesses by a particular user

4. Data Administration
   - Centralized professional data administration by experienced users can manage data access, organize data representation to minimize redundancy, and fine tune the storage

# Why use a DBMS?

5. Concurrent Access and Crash Recovery
   - DBMS schedules concurrent accesses to the data such that the users think that the data is being accessed by only one user at a time
   - DBMS protects data from system failures

6. Reduced Application Development Time
   - Supports many functions that are common to a number of applications accessing data
   - Provides high-level interface
   - Facilitates quick and robust application development

# When NOT to use a DBMS?

- DBMS is optimized for certain kind of workloads and manipulations
- There may be applications with tight real-time constraints or a few well-defined critical operations
- Abstract view of the data provided by DBMS may not suffice
- To run complex, statistical/ML analytics on large datasets

# Data Model

- Applications need to model some real-world units
- Entities:
  - Students, Departments, Courses, Faculty, Organization, Employee, …
- Relationships:
  - Course enrollments by students, Product sales by an organization

- A data model is a collection of high-level data description constructs that hide many low-level storage details

# Data Model

Can Specify:

1. Structure of the data
   – like arrays or structs in a programming language
   – but at a higher level (conceptual model)

2. Operations on the data
   – unlike a programming language, not any operation can be performed
   – allow limited sets of queries and modifications
   – a strength, not a weakness!

3. Constraints on the data
   – what the data can be
   – e.g., a movie has exactly one title

# Important Data Models

- Structured Data
- Semi-structured Data
- Unstructured Data

What are these?

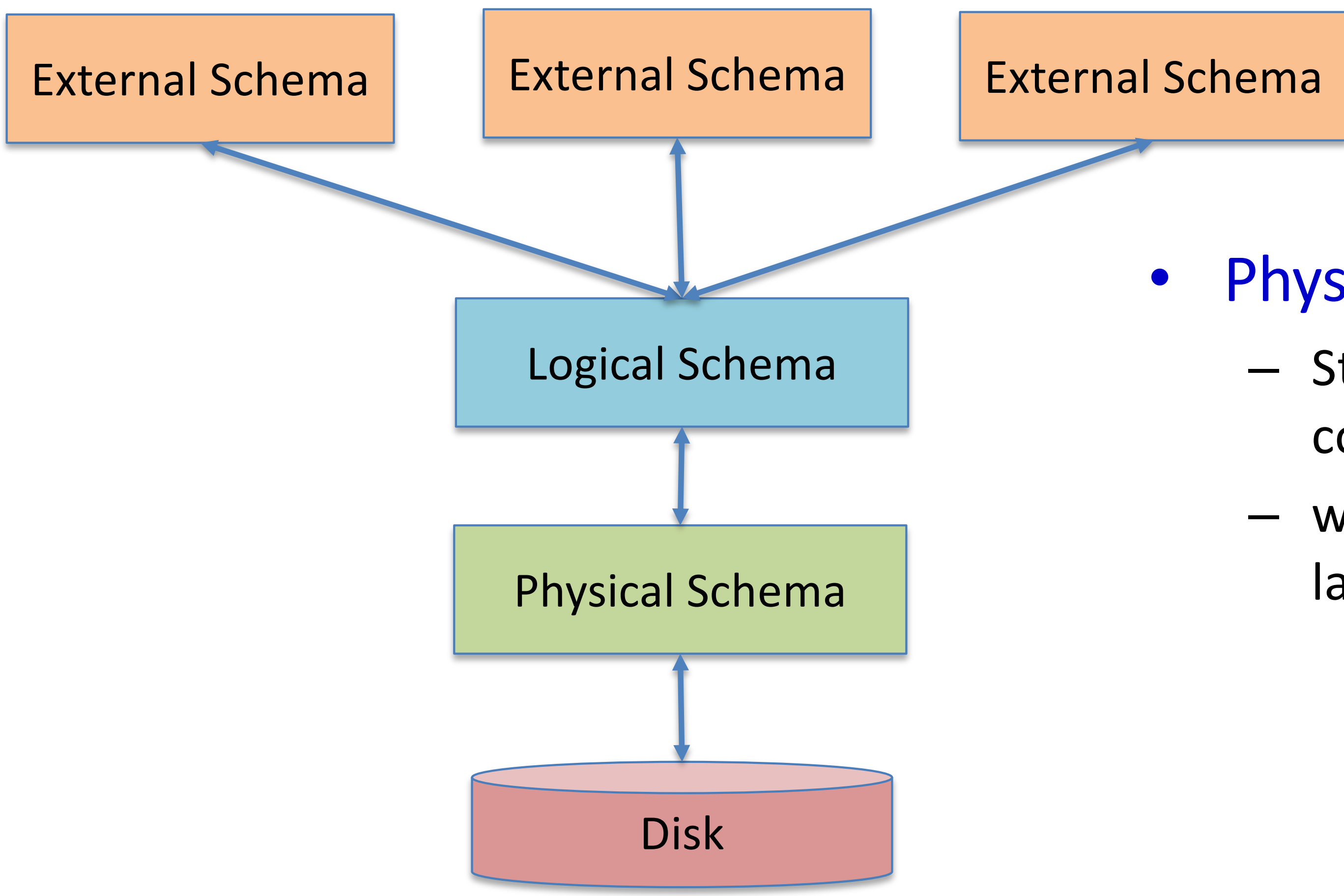
# Important Data Models

- Structured Data
  - All elements have a fixed format
  - Relational Model (table, Lecture-1)

- Semi-structured Data
  - Some structure but not fixed
  - Hierarchically nested tagged-elements in tree structure
  - XML (HW1)

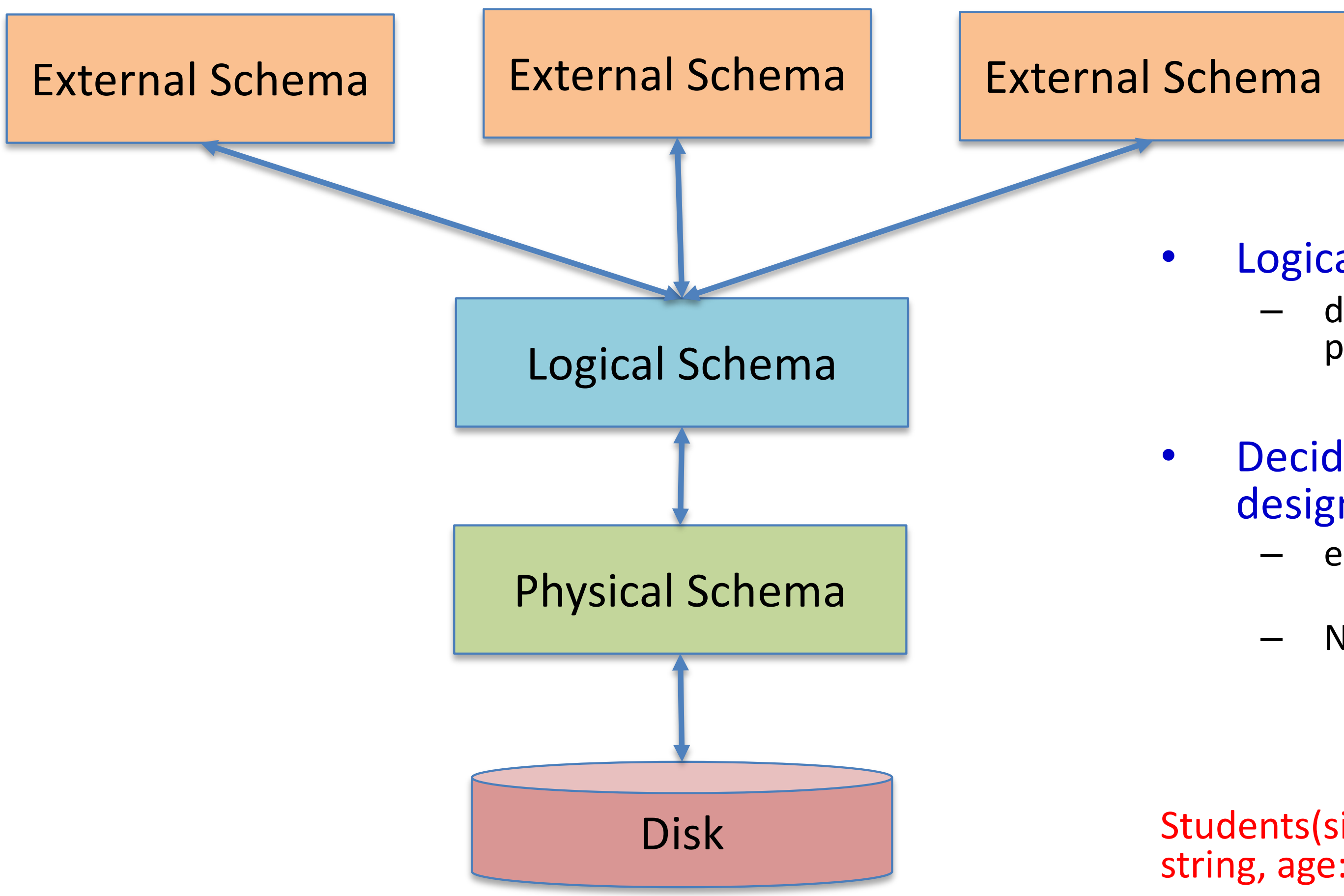
- Unstructured Data
  - No structure, not covered in this course
  - Text, image, audio, video (still some structure)

# Levels of Abstractions in a DBMS




- Physical schema
  - Storage as files, row vs. column store, indexes
  - will discuss these in later lectures

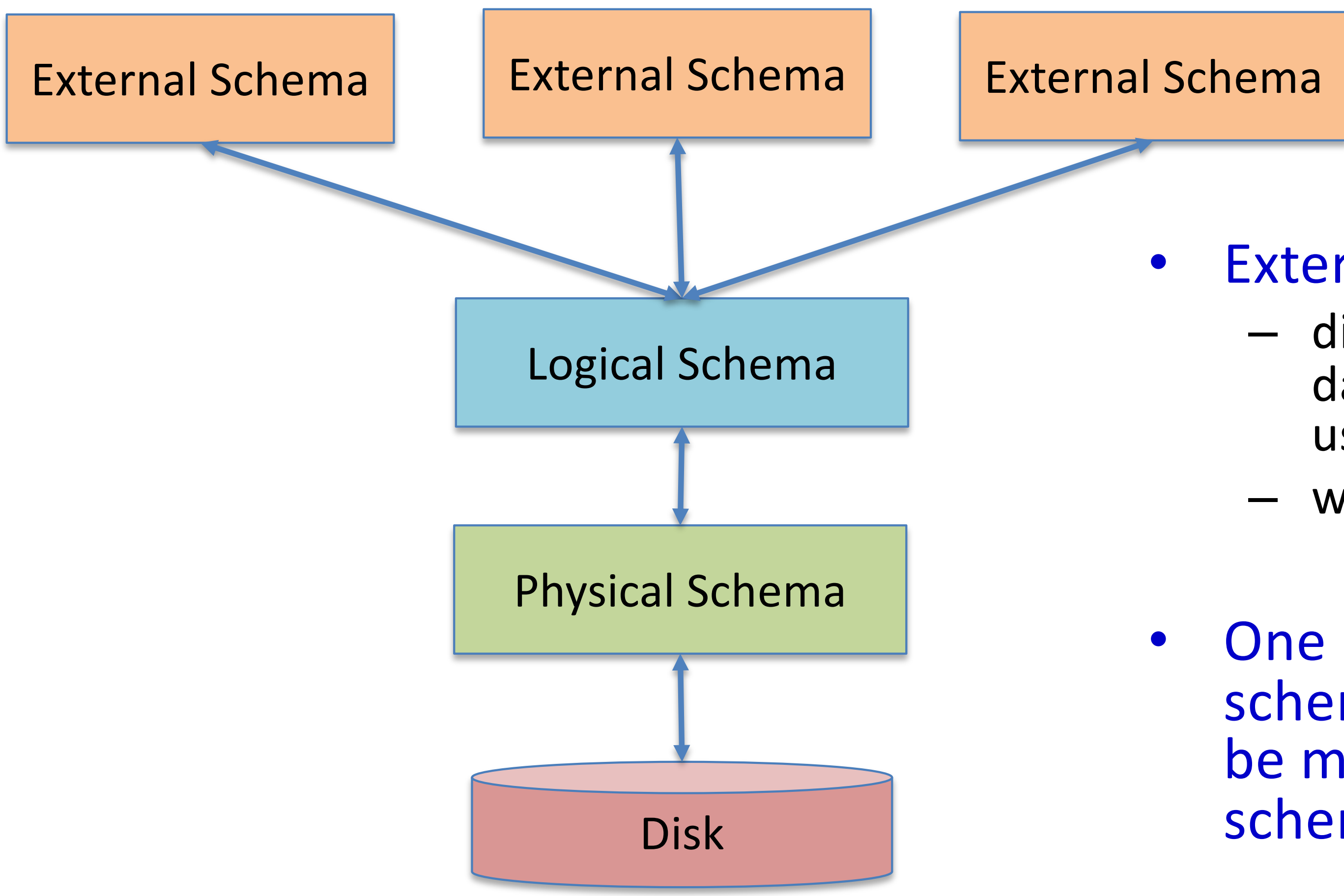
# Levels of Abstractions in a DBMS




- Logical/Conceptual schema
  - describes the stored data in the physical schema
- Decided by conceptual schema design
  - e.g. ER Diagram
    - not covered in this course
  - Normalization
    - will be covered

Students(sid: string, name: string, login: string, age: integer, gpa: real)

# Levels of Abstractions in a DBMS




- External schema
  - different "views" of the database to different users
  - will discuss views later

- One physical and logical schema but there can be multiple external schemas

# Data Independence

- Application programs are insulated from changes in the way the data is structured and stored

- A very important property of a DBMS

- Logical and Physical

# Logical Data Independence

- Users can be shielded from changes in the logical structure of data
- e.g. Students:

  Students(sid: string, name: string, login: string, age: integer, gpa: real)
- Divide into two relations

  Students_public(sid: string, name: string, login: string)

  Students_private(sid: string, age: integer, gpa: real)
- Still a “view” Students can be obtained using the above new relations
  - by “joining” them with sid
- A user who queries this view Students will get the same answer as before

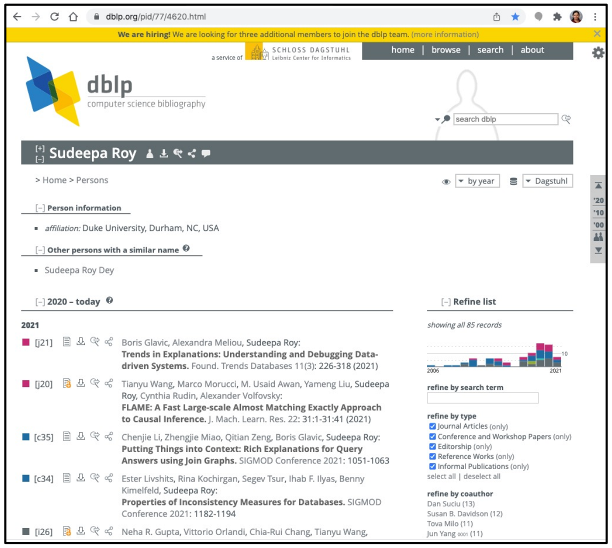
# Physical Data Independence

- The logical/conceptual schema insulates users from changes in physical storage details
  - how the data is stored on disk
  - the file structure
  - the choice of indexes
- The application remains unaltered
  - But the performance may be affected by such changes

# XML: A brief overview

# Semi-structured Data and XML

- XML: Extensible Markup Language
- Will not be covered in detail in class, but many datasets available to download (DBLP, Yelp) are in this form
  - You will download the DBLP publication dataset ([https://dblp.org/](https://dblp.org/), CS Bibliography) in XML format and transform into relational form (in HW1)




- Data does not have a fixed schema
  - "Attributes" are part of the data
  - The data is "self-describing"
  - Tree-structured

# XML: Example

Attributes

```
<article mdate="2011-01-11” key="journals/acta/Saxena96">
    <author>Sanjeev Saxena</author>
    <title>Parallel Integer Sorting and Simulation Amongst CRCW
        Models.</title>
    <pages>607-619</pages>
    <year>1996</year>
    <volume>33</volume>
    <journal>Acta Inf.</journal>
    <number>7</number>
    <url>db/journals/acta/acta33.html#Saxena96</url>
    <ee>http://dx.doi.org/10.1007/BF03036466</ee>
</article>
```

Elements

# Attribute vs. Elements

- Elements can be repeated and nested
- Attributes are unique and atomic

# Why XML?

+ Serves as a model suitable for integration of databases containing similar data with different schemas
  - e.g. try to integrate two student databases: S1(sid, name, gpa) and S2(sid, dept, year)
  - Many "NULL"s (for unknown data) if done in relational model, very easy in XML

+ Flexible – easy to change the schema and data

- Makes query processing more difficult

Which one is easier?

- XML (semi-structured) to relational (structured)

or

- relational (structured) to XML (semi-structured)?

There are query languages for XML – XPATH, XQUERY etc., not covered in this course

# XML to Relational Model

- Problem 1: Repeated attributes

<book>

<author>Ramakrishnan</author>

<author>Gehrke</author>

<title>Database Management Systems</title>

<pubisher> McGraw Hill

</book>

What is a good relational schema?

# XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <title>Database Management Systems</title>
    <pubisher> McGraw Hill</publisher>
</book>
```

| Title | Publisher | Author1 | Author2 |
|---|---|---|---|
| | | | |

# XML to Relational Model

- Problem 1: Repeated attributes

```
<book>
    <author>Garcia-Molina</author>
    <author>Ullman</author>
    <author>Widom</author>
    <title>Database Systems – The Complete Book</title>
    <pubisher>Prentice Hall</publisher>
</book>
```

Does not work

| Title | Publisher | Author1 | Author2 |
|---|---|---|---|
| | | | |

Better design?

# XML to Relational Model

Book

| BookId | Title | Publisher |
|---|---|---|
| b1 | Database Management Systems | McGraw Hill |
| b2 | Database Systems – The Complete Book | Prentice Hall |

BookAuthoredBy

| BookId | Author |
|---|---|
| b1 | Ramakrishnan |
| b1 | Gehrke |
| b2 | Garcia-Molina |
| b2 | Ullman |
| b2 | Widom |

# XML to Relational Model

- Problem 2: Missing attributes

<book>
<author>Ramakrishnan</author>
<author>Gehrke</author>
<title>Database Management Systems</title>
<pubisher> McGraw Hill
<edition>Third</edition>
</book>
<book>
<author>Garcia-Molina</author>
<author>Ullman</author>
<author>Widom</author>
<title>Database Systems – The Complete Book</title>
<pubisher>Prentice Hall</publisher>
</book>

| BookId | Title | Publisher | Edition |
|---|---|---|---|
| b1 | Database Management Systems | McGraw Hill | Third |
| b2 | Database Systems – The Complete Book | Prentice Hall | null |

# Summary: Data Model

- Relational data model is the most standard for database managements
  - semi-structured model/XML is also used in practice – you will use them in hw assignments
  - unstructured data (text/photo/video) is unavoidable, but won't be covered in this class

- A DBMS provides data independence and insulates the application programmer from many low level details

- We will learn about those low level details as well as high level data management in this course

# SQL Programming

# SQL Programming: Working with SQL through an API

- E.g.: Python psycopg2, JDBC, ODBC (C/C$^{++}$/VB)
  - All based on the SQL/CLI (Call-Level Interface) standard
  - You can use any of these in HW1

- The application program sends SQL commands to the DBMS at runtime
- Responses/results are converted to objects in the application program

# Example API: Python psycopg2

```
import psycopg2
conn = psycopg2.connect(dbname='beers')
cur = conn.cursor()
# list all drinkers:
cur.execute('SELECT * FROM Drinker')
for drinker, address in cur:
    print(drinker + ' lives at ' + address)
# print menu for bars whose name contains “a”:
cur.execute('SELECT * FROM Serves WHERE bar LIKE %s', ('%a%',))
for bar, beer, price in cur:
    print('{} serves {} at ${:,.2f}'.format(bar, beer, price))
cur.close()
conn.close()
```

beers database
Drinker(drinker, address)
Serves(bar, beer, price)

You can iterate over cur one tuple at a time

Placeholder for query parameter

Tuple of parameter values, one for each %s
(note that the trailing “,” is needed when the tuple contains only one value)

# More psycopg2 examples

```
# “commit” each change immediately (later in transactions)—need to set this option just
once at the start of the session
conn.set_session(autocommit=True)
# ...
bar = input('Enter the bar to update: ').strip()
beer = input('Enter the beer to update: ').strip()
price = float(input('Enter the new price: '))
try:
    cur.execute('''
UPDATE Serves
SET price    = %s
WHERE bar = %s AND beer = %s''', (price, bar, beer))
    if cur.rowcount != 1:
        print('{} row(s) updated: correct bar/beer?'\
            .format(cur.rowcount))
except Exception as e:
    print(e)
```

# of tuples modified

Exceptions can be thrown
(e.g., if positive-price constraint is violated)

# End of Lecture-2 (01/11)

- TODOs:
  1. Look at HW1-Part I:
     - Sakai -> Resources -> Homeworks -> HW1
  2. Start working on HW1 early – quite a bit of self-learning needed
  3. Read course policy (link) carefully before you start
  4. Go to office hours if you have questions
     - Links on Ed
  5. Check out the Project thread on Ed and keep looking for teams / teammates