CMSC 424 – Database design
Lecture 2:
Design, Modeling, Entity-Relationship

Book: Chap. 1 and 6

Mihai Pop

Administrative issues

- TA: Sharath Srinivas
- TA office hours: Mon 10-11:30, Wed 3-4:30, AVW 1112

- Glue and Oracle accounts.

- When you email me or TA: List CMSC424 in Subject line!!!

- Policy amendment:
  - Laptops OK but JUST FOR NOTE TAKING!

Today

- Data management challenges in a very simple application
  - Why we can’t use a file system to do database management
- Data Modeling
  - Going from conceptual requirements of a application to a concrete data model

Example

- Simple Banking Application
  - Need to store information about:
    - Accounts
    - Customers
  - Need to support:
    - ATM transactions
    - Queries about the data
- Instructive to see how a naïve solution will work

A *file-system* based solution

- Data stored in files in ASCII format
  - #-separate files in /usr/db directory
  - /usr/db/accounts

    Account Number # Balance

    101 # 900

    102 # 700

    …
  - /usr/db/customers

    Customer Name # Customer Address # Account Number

    Johnson # 101 University Blvd # 101

    Smith # 1300 K St # 102

    Johnson # 101 University Blvd # 103

    …

A *file-system* based solution

- Write application programs to support the operations
  - In your favorite programming language
  - To support withdrawals by a customer for amount $X from account Y
    - Scan /usr/db/accounts, and look for Y in the 1st field
    - Subtract $X from the 2nd field, and rewrite the file
  - To support finding names of all customers on street Z
    - Scan /usr/db/customers, and look for (partial) matches for Z in the addess field
  - …

What's wrong with this solution ?

1. Data redundancy and inconsistency
   - No control of *redundancy*

Customer Name # Customer Address # Account Number
Johnson # 101 University Blvd # 101
Smith # 1300 K St # 102
Johnson # 101 University Blvd # 103
…
Especially true when programs/data organization evolve over time

   - Inconsistencies
     - Data in different files may not agree
     - Very critical issue

What's wrong with this solution ?

2. Evolution of the database is hard
   - Delete an account
     - Will have to rewrite the entire file
   - Add a new field to the *accounts* file, *or*

     split the *customers* file in two parts:
     - Rewriting the entire file least of the worries
     - Will probably have to rewrite all the application programs

What's wrong with this solution ?

3. Difficulties in Data Retrieval
   - No sophisticated tools for selective data access
     - Access only the data for customer X
     - Inefficient to scan the entire file
   - Limited reuse
     - Find customers who live in area code 301
     - Unfortunately, no application program already written
     - Write a new program every time ?

What's wrong with this solution ?

4. Semantic constraints
   - Semantic integrity constraints become part of program code
     - *Balance* should not fall below 0
     - Every program that modifies the *balance* will have to enforce this constraint
   - Hard to add new constraints or change existing ones
     - *Balance* should not fall below 0 unless overdraft-protection enabled
     - Now what?
       - Rewrite every program that modifies the *balance* ?

What's wrong with this solution ?

5. Atomicity problems because of failures

Jim transfers $100 from Acct #55 to Acct #376

1. Get balance for acct #55
2. If balance55 > $100 then
   a. balance55 := balance55 - 100
   b. update balance55 on disk
   c. get balance from database for acct #376
   d. balance376 := balance376 + 100
   e. update balance376 on disk

CRASH

Must be *atomic*

Do all the operations or none of the operations

What's wrong with this solution ?

6. Durability problems because of failures

Jim transfers $100 from Acct #55 to Acct #376

1. Get balance for acct #55
2. If balance55 > $100 then
   a. balance55 := balance55 - 100
   b. update balance55 on disk
   c. get balance from database for acct #376
   d. balance376 := balance376 + 100
   e. update balance376 on disk
   f. print receipt

CRASH

After reporting success to the user, the changes better be there when he checks tomorrow

What's wrong with this solution ?

7. Concurrent access anomalies

Joe@ATM1: Withdraws $100 from Acct #55

1. Get balance for acct #55
2. If balance55 > $100 then
    a. balance55 := balance55 – 100
    b. dispense cash
    c. update balance55

Jane@ATM2: Withdraws $50 from Acct #55

1. Get balance for acct #55
2. If balance55 > $50 then
    a. balance55 := balance55 – 50
    b. dispense cash
    c. update balance55

What's wrong with this solution ?

7. Concurrent access anomalies

Joe@ATM1: Withdraws $100 from Acct #55

1. Get balance for acct #55
2. If balance55 > $100 then
    a. balance55 := balance55 – 100
    b. dispense cash
    c. update balance55

Jane@ATM2: Withdraws $50 from Acct #55

1. Get balance for acct #55
2. If balance55 > $50 then
    a. balance55 := balance55 – 50
    b. dispense cash
    c. update balance55

Balance would only reflect one of the two operations

Bank loses money

What's wrong with this solution ?

8. Security Issues
   - Need fine grained control on who sees what
     - Only the manager should have access to accounts with balance more than $100,000
     - How do you enforce that if there is only one accounts file ?

Database management provide an end-to-end solution to all of these problems

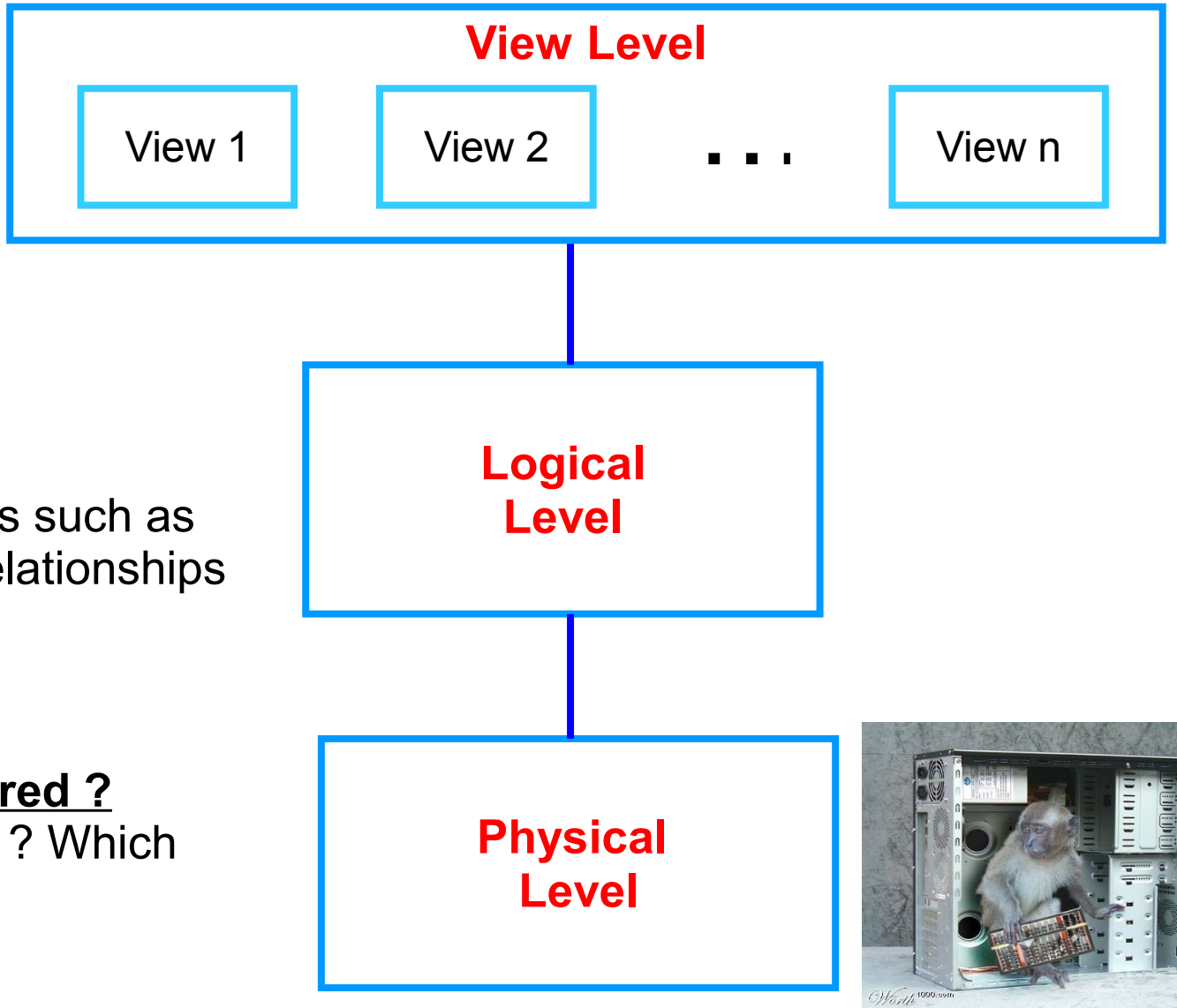
Data Abstraction

- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system

- Through use of *logical abstractions*

Data Abstraction

What data users and application programs see ?



What data is stored ?

describe data properties such as data semantics, data relationships

How data is actually stored ?

e.g. are we using disks ? Which file system ?

Data Abstraction: Banking Example

- Logical level:
  - Provide an abstraction of *tables*
  - Two tables can be accessed:
    - *accounts*
      - Columns: account number, balance
    - *customers*
      - Columns: name, address, account number
- View level:
  - A teller (non-manager) can only see a part of the *accounts* table
    - Not containing high balance accounts

Data Abstraction: Banking Example

- Physical Level:
  - Each table is stored in a separate ASCII file
  - # separated fields

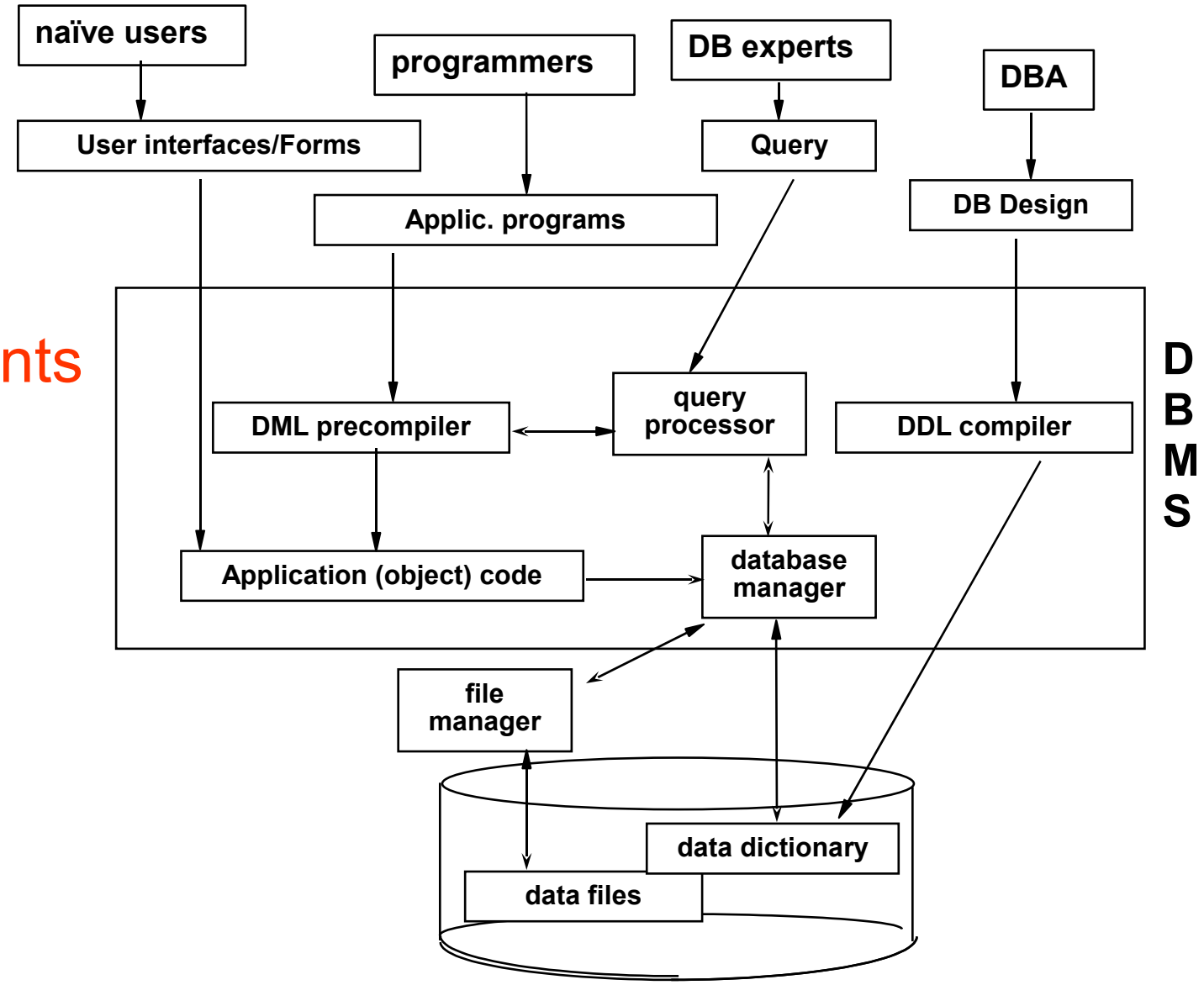
- Identical to what we had before ?
  - BUT the users are not aware of this
    - They only see the tables
    - The application programs are written over the tables abstraction
  - Can change the physical level without affecting users
  - In fact, can even change the logical level without affecting the *teller*

DBMS at a Glance

1. Data Modeling
2. Data Retrieval
3. Data Storage
4. Data Integrity

Architecture of a DBMS

System Components & Interfaces



DML: Data Manipulation Language
DDL: Data Definition Language
DBA: Data Base Administrator

Data Modeling

- A data model is a collection of concepts for describing data properties and domain knowledge:
  - Data relationships
  - Data semantics
  - Data constraints
- We will discuss two models extensively in this class
  - Entity-relationship Model
  - Relational Model
- Probably discuss XML as well

Data Retrieval

- Query = Declarative data retrieval program
  - describes *what* data to acquire, not *how* to acquire it
  - Non-declarative:
    - scan the *accounts* file
    - look for number 55 in the 2nd field
    - subtract $50 from the 3rd field
  - Declarative (posed against the *tables* abstraction):
    - Subtract $50 from the column named *balance* for the row corresponding to *account number* 55 in the *accounts* table
    - How to do it is not specified.
- Why ?
  - Easier to write
  - More efficient to execute (why ?)

Data Storage

- Where and how to store data ?
  - Main memory ?
    - What if the database larger than memory size ?
  - Disks ?
    - How to move data between memory and disk ?
  - Etc etc…

Data Integrity

- Manage concurrency and crashes
  - *Transaction*: A sequence of database actions enclosed within special tags
  - Properties:
    - ***Atomicity***: Entire transaction or nothing
    - ***Consistency***: Transaction, executed completely, take database from one consistent state to another
    - ***Isolation***: Concurrent transactions *appear* to run in isolation
    - ***Durability***: Effects of committed transactions are not lost
  - Consistency: Transaction programmer needs to guarantee that
    - DBMS can do a few things, e.g., enforce constraints on the data
  - Rest: DBMS guarantees

Data Integrity

- Semantic constraints
  - Typically specified at the logical level
  - E.g. *balance* > 0

DBMS at a glance

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

- Not fully disjoint categorization !!

Motivation

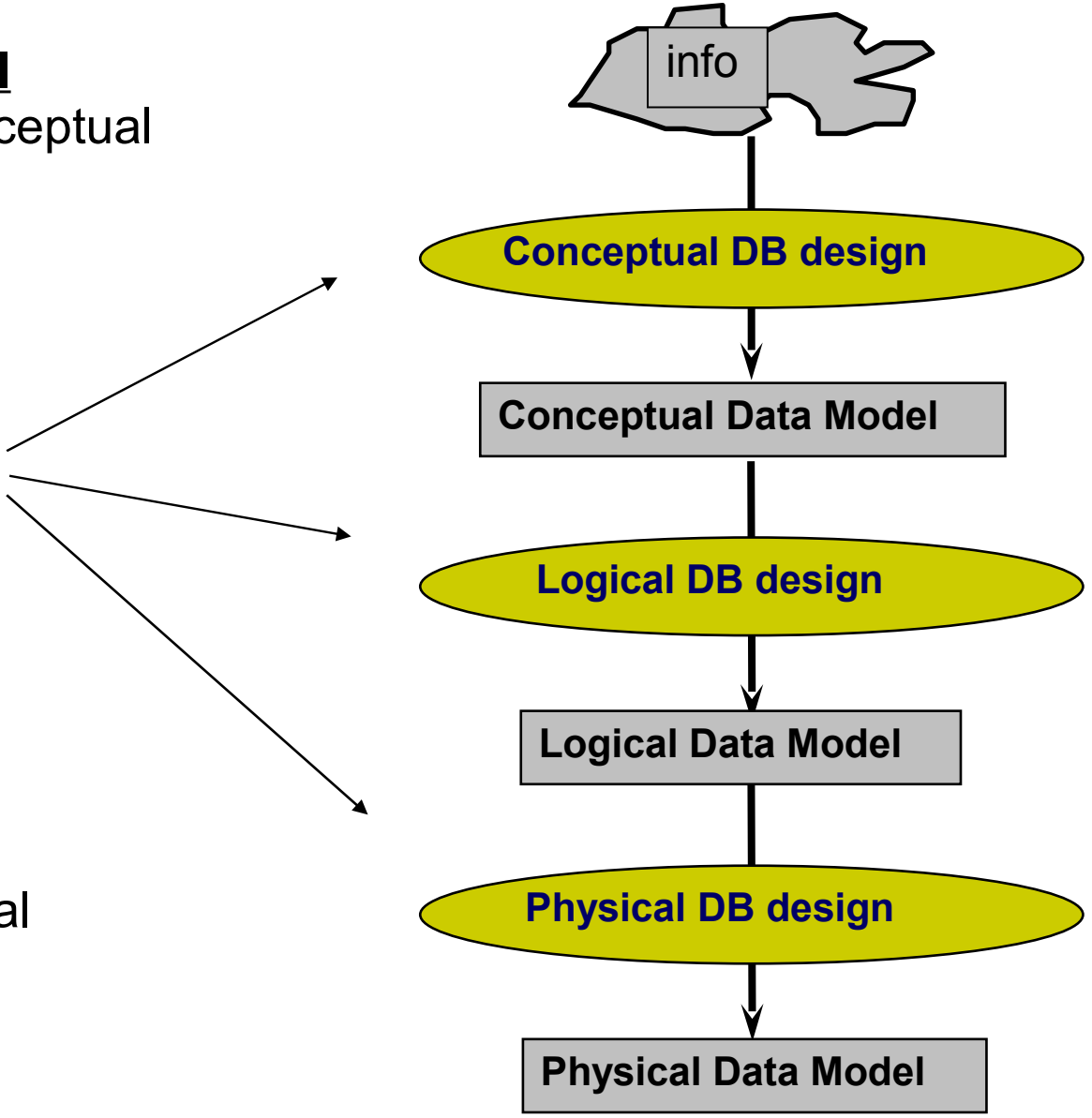
- You've just been hired by Bank of America as their DBA for their online banking web site.
- You are asked to create a database that monitors:
  - customers
  - accounts
  - loans
  - branches
  - transactions, …
- Now what??!!!

Database Design Steps

Entity-relationship Model

Typically used for conceptual database design

Three Levels of Modeling



Relational Model

Typically used for logical database design

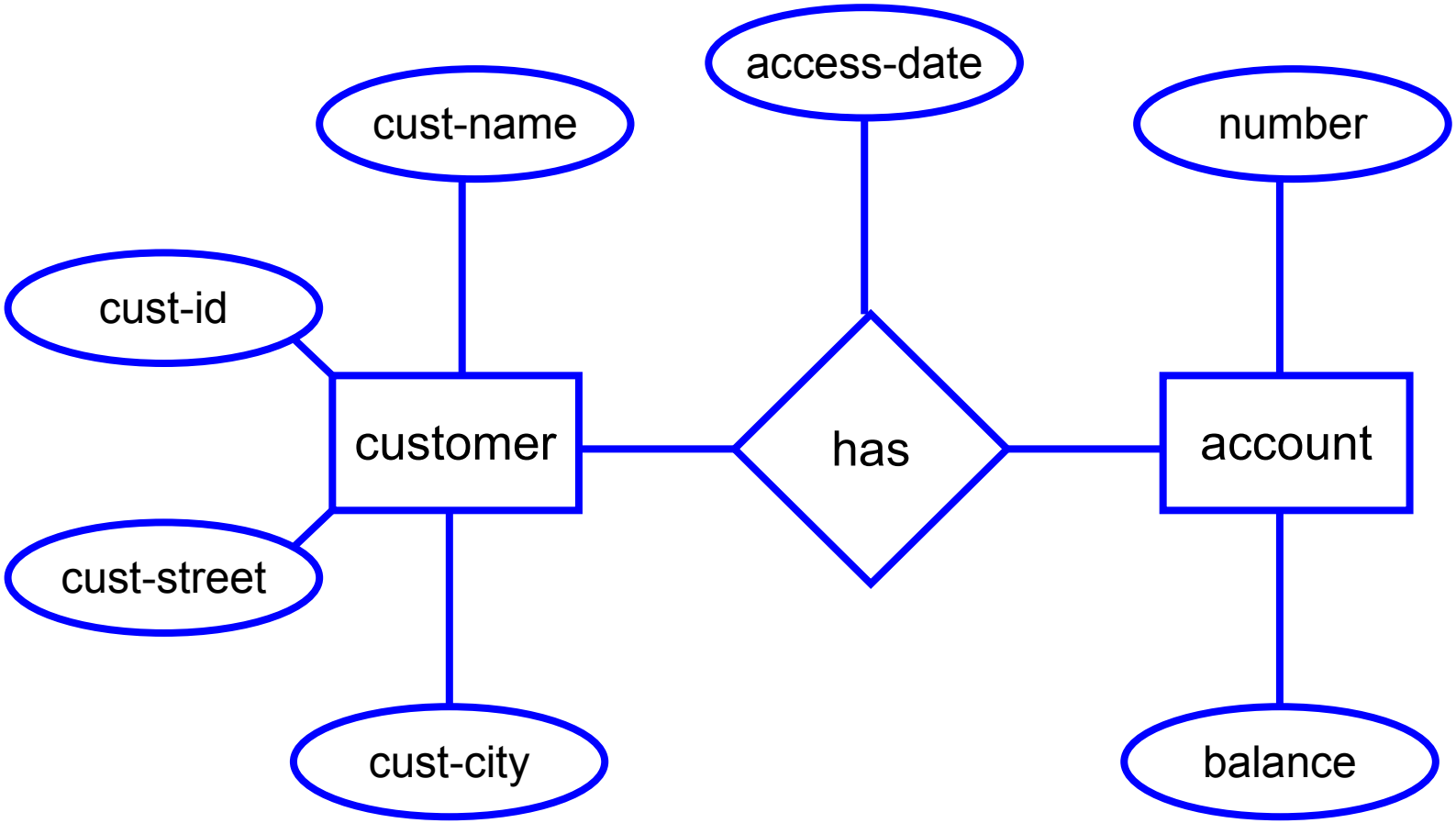
Entity-Relationship Model

- Two key concepts
  - *Entities*:
    - An object that *exists* and is *distinguishable* from other objects
      - Examples: Bob Smith, BofA, CMSC424
    - Have *attributes* (people have names and addresses)
    - Form *entity sets* with other entities of the same type that share the same properties
      - Set of all people, set of all classes
    - Entity sets may overlap
      - Customers and Employees

Entity-Relationship Model

- Two key concepts
  - *Relationships*:
    - Relate 2 or more entities
      - E.g. Bob Smith *has account at* College Park Branch
    - Form *relationship sets* with other relationships of the same type that share the same properties
      - Customers *have accounts at* Branches
    - Can have attributes:
      - *has account at* may have an attribute *start-date*
    - Can involve more than 2 entities
      - Employee *works at* Branch *at* Job

ER Diagram: Starting Example



- Rectangles: entity sets
- Diamonds: relationship sets
- Ellipses: attributes

Next: Relationship Cardinalities

- We may know:

  One customer can only open one account

  OR

  One customer can open multiple accounts

- Representing this is important
- Why ?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - Application logic will have to do it; NOT GOOD
  - Remember: If not represented in conceptual model, the domain knowledge may be lost

Relationships

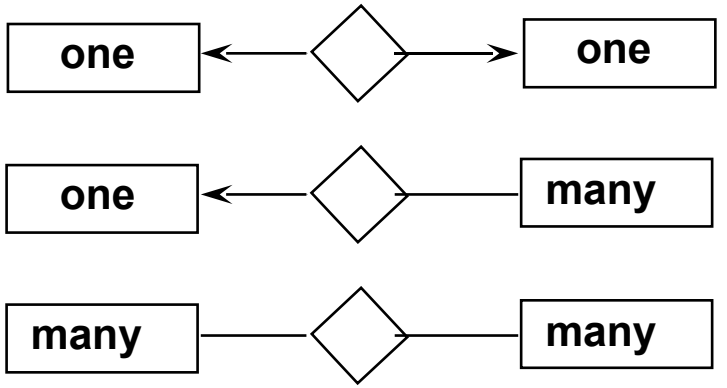
- **relationship:** an association among entities
  - Joe Doe lives in the White House

- **relationship set:** a collection of relationships of the same type
  - PEOPLE LIVE in HOUSEs
  - formally is a relation on n>=2 (possibly non distinct) entity sets
    - {(e1,e2,…,en) | e1ε E1,…en ε En}
    - where (e1,e2,…,en) is a relationship

- relationships can also have attributes (properties that have a single value),
  - e.g. LIVE has an attribute DATE-MOVED-IN (e.g. to store the value the PERSON moved in the HOUSE (January 20th, 2001) and DATE-MOVED-OUT (e.g. January 19th, 2005)

Mappings amongst relationships

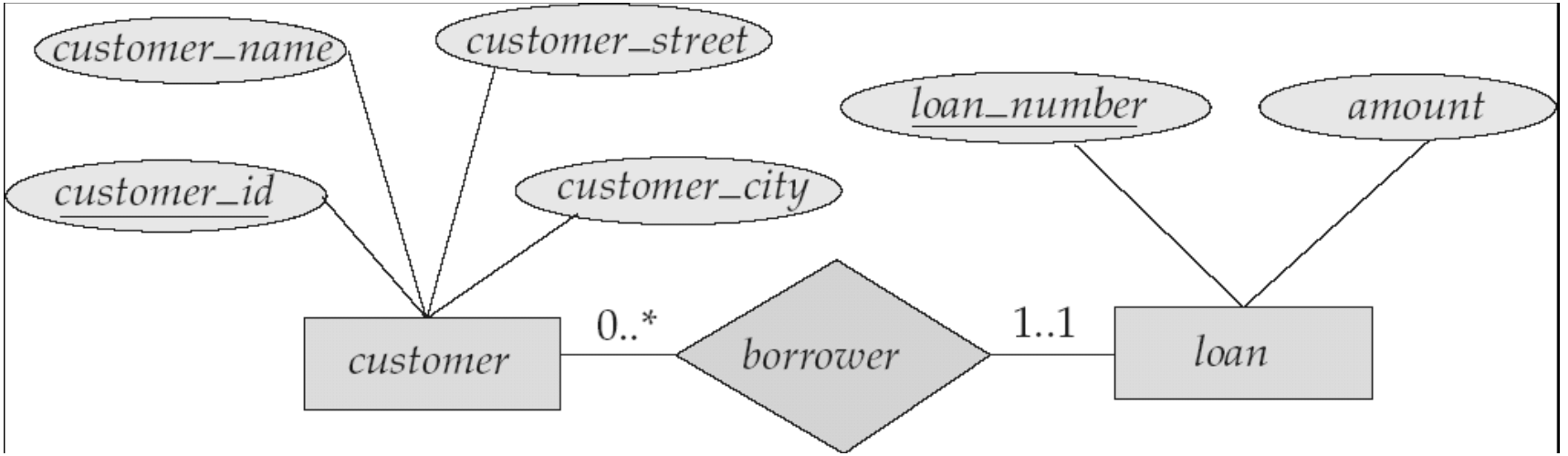
- **1-1 (PERSONs and IRS-RECORD)**
- **1-many (PERSON and ACCOUNTs)**
- **many-many (STUDENTs and COURSEs)**



Note arrow points to the one

Alternative Notation for Cardinality Limits

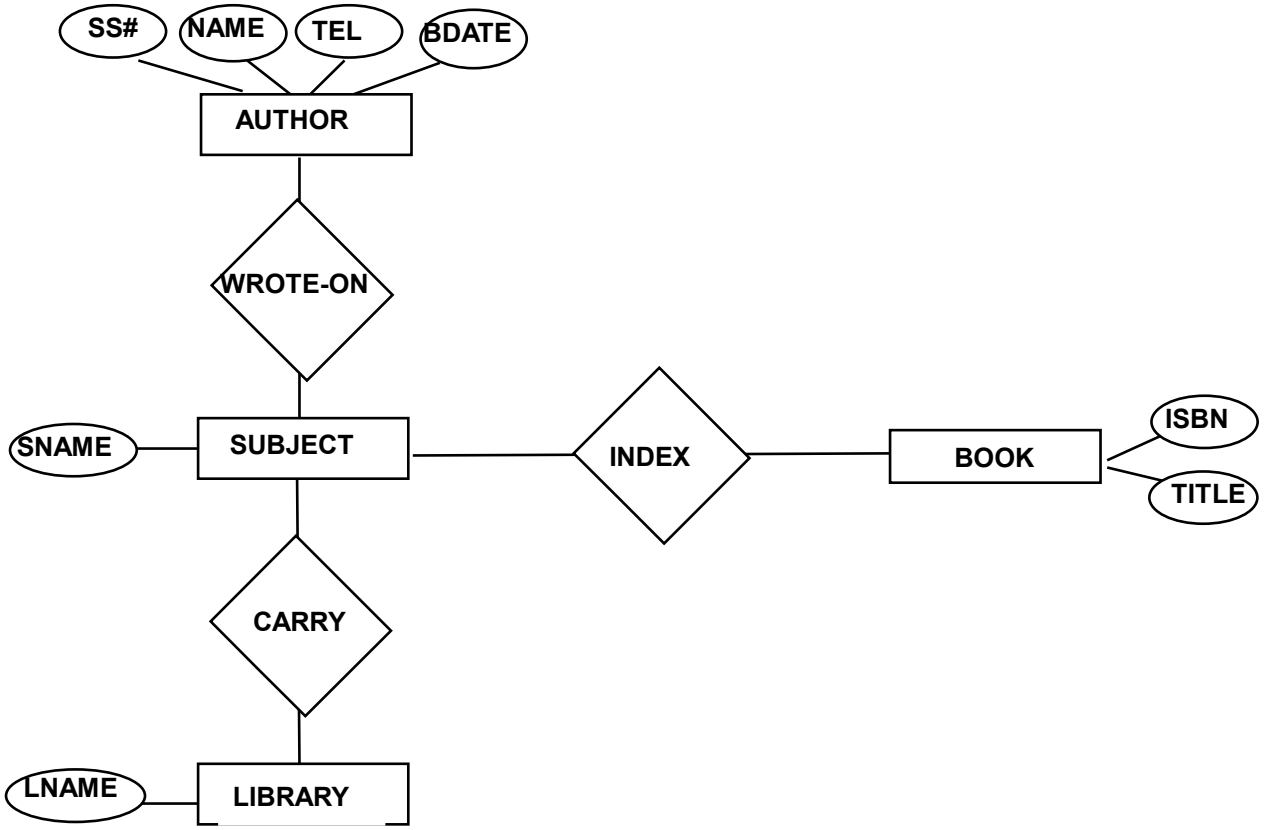
Cardinality limits can also express participation constraints



Our First Database Design

Application:

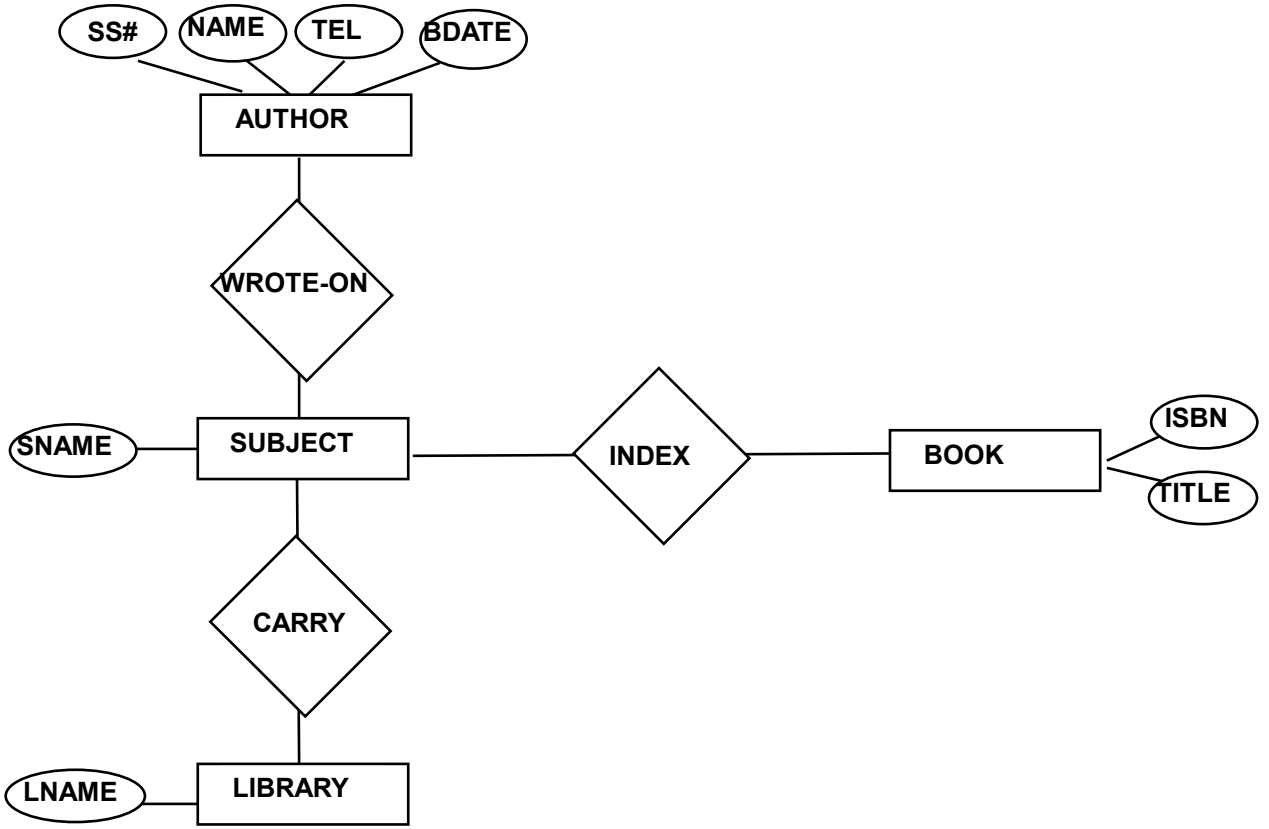
A library database that stores authors who have written books about various subjects. The database will also store info about libraries that carry books on these subjects.



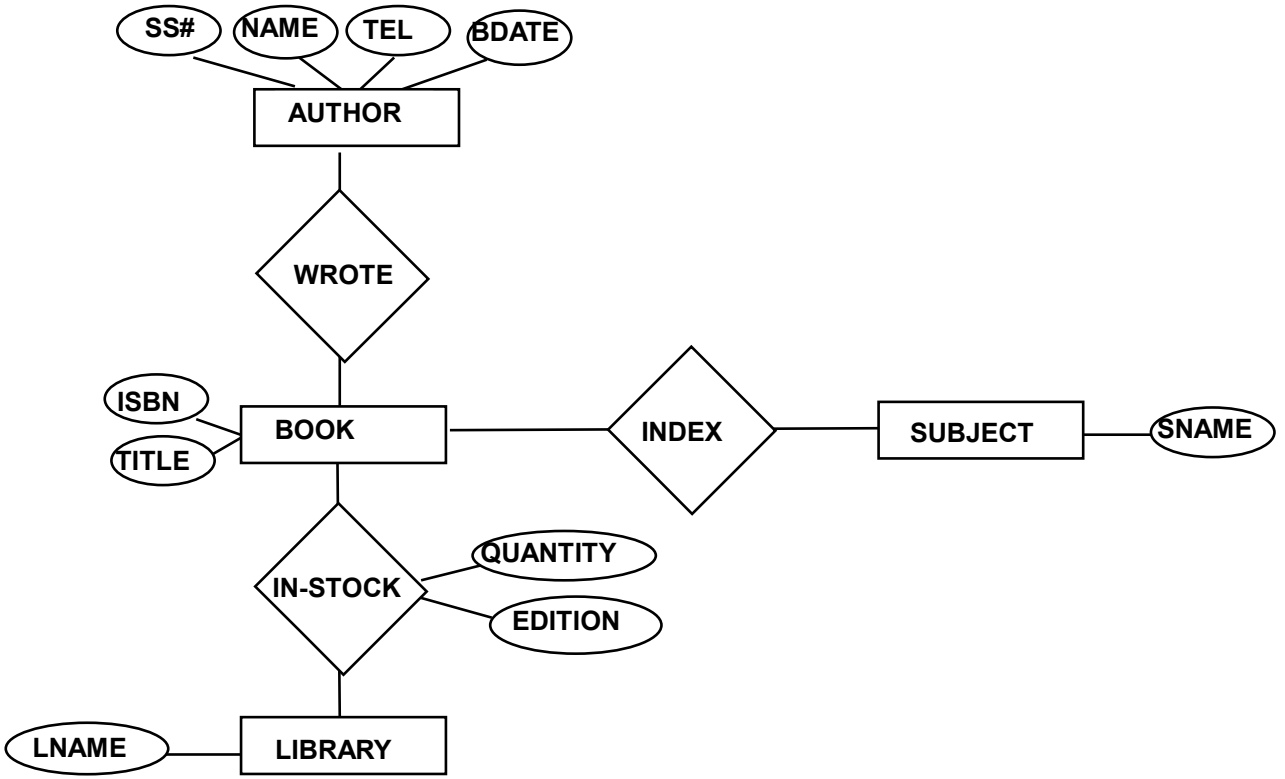
What's wrong?

Problems in our First Design

- does not capture the fact that a library carries books of a specific author
- does not capture the fact that a library carries a specific book
- does not capture the fact that an author has written a specific book
- does not store which edition of the book the library has, how many copies, etc.



2nd Attempt to the Library Design



- Much better