RUTGERS UNIVERSITY
PRINCIPLES OF INFORMATION AND DATA MANAGEMENT

Evolution:

- Information is stored in DNA. Species adapt by modifying their DNA.
- This process is extremely slow.
- Humans have produced and consumed information since the stone age. The techniques required to make tools were passed from person to person.
- Writing made things easier since information could be preserved from generation to generation.
- Huge leaps after that: invention of the Printing Press (1440), Invention of the Computer (1950), Databases (1970), Internet (1990).

DATA AND INFORMATION (1)

Humans are Information processing beings.

What do we need information for? What is information? Is data the same thing?

DATA: Facts. Product of observation

Beware of count and mass nouns

Data is plural of Datum (that is rarely used in Computer Science)

Example of data:

John Smith took the following courses in Spring 2016:
Discrete Structures I, Computer Architecture, Linear Algebra,
Software Methodology
John Smith: SAT score: 570 math, 600 verbal
High school GPA: 3.5

Computer Science question:

How do we organize and access data?

DATA AND INFORMATION (2)

Why do we want to keep data?

To take some action based on the data.

But the data itself, just a bunch of facts is not really helpful for decision making. The data must be somehow processed to become useful.

Data that a University might find useful for decision making:

- Retention rate
- Dropout rate
- Percentage of students failing at least one class

More analysis: The retention rate of _____ is lower than average (with respect to other universities)

The data has now become INFORMATION

DATA AND INFORMATION (3)

The data becomes information as answer to an inquiry:
Who? What? When? Where? How many?
etc.

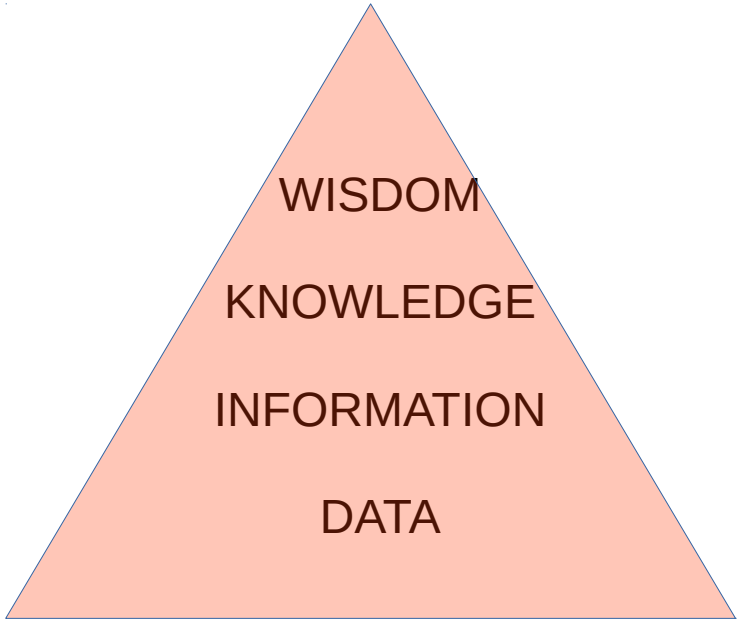
Structural or Functional

Is the difference based on the structure of data vs the structure of information? (structural)

Is the difference based on the purpose of data vs the purpose of information? (functional) ------- THIS ONE SEEMS TO BE RIGHT

Information is inferred from data ------ USE OF INFORMATION SYSTEMS

DIKW PYRAMID



Knowledge:

Know-how, decision making, transform information into instructions.

Wisdom:

Why?

Is not discussed and not really used in computer Science.

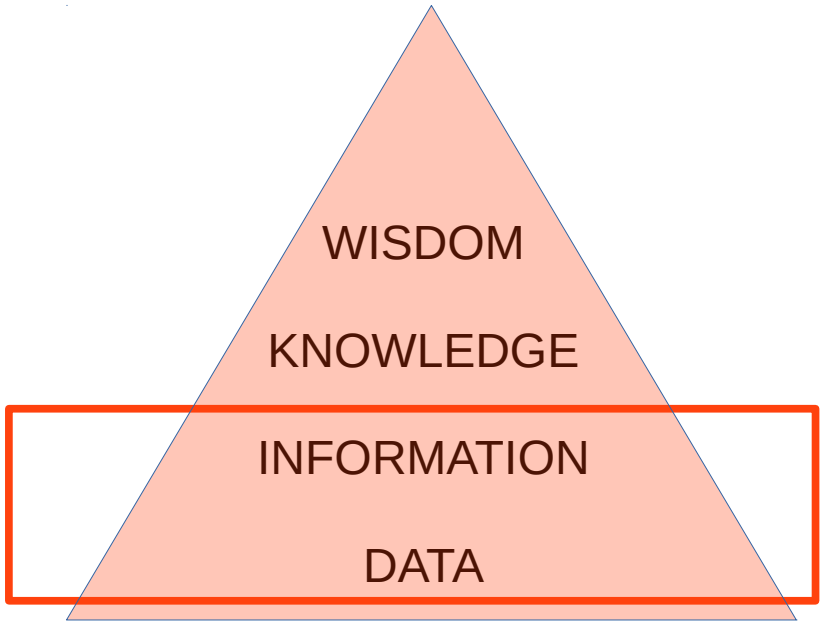
Probably not even Scientific

Pragmatic view. Positivism. If the decision making process only involves deductions made from data, and not from other "Not observable" sources or properties. It is thinking only "inside the box" (pyramid).

Is DATA considered to be "True" data?
Is "THE TRUTH" known through data?
In other words: is there no truth outside of the data collected?

BEWARE: This model encourages the mindless collection and use of data.

DIKW PYRAMID (2)



Where does this course fit in this pyramid?

We are going to talk about:

- Data collection
- Data organization
- Data storage
- Data retrieval
- Reporting and generation of information.

We are not going to cover Knowledge based systems, inference engines, Artificial Intelligence, Semantic Networks, Expert Systems, etc.

SOURCE OF DATA

Mini World:

- Data cannot be known about the entire world or about everything or every topic.
- We need to restrict the domain from which we are going to gather data.
- This restriction leads us to the Mini-World concept

The data that we store about our mini-world can be:

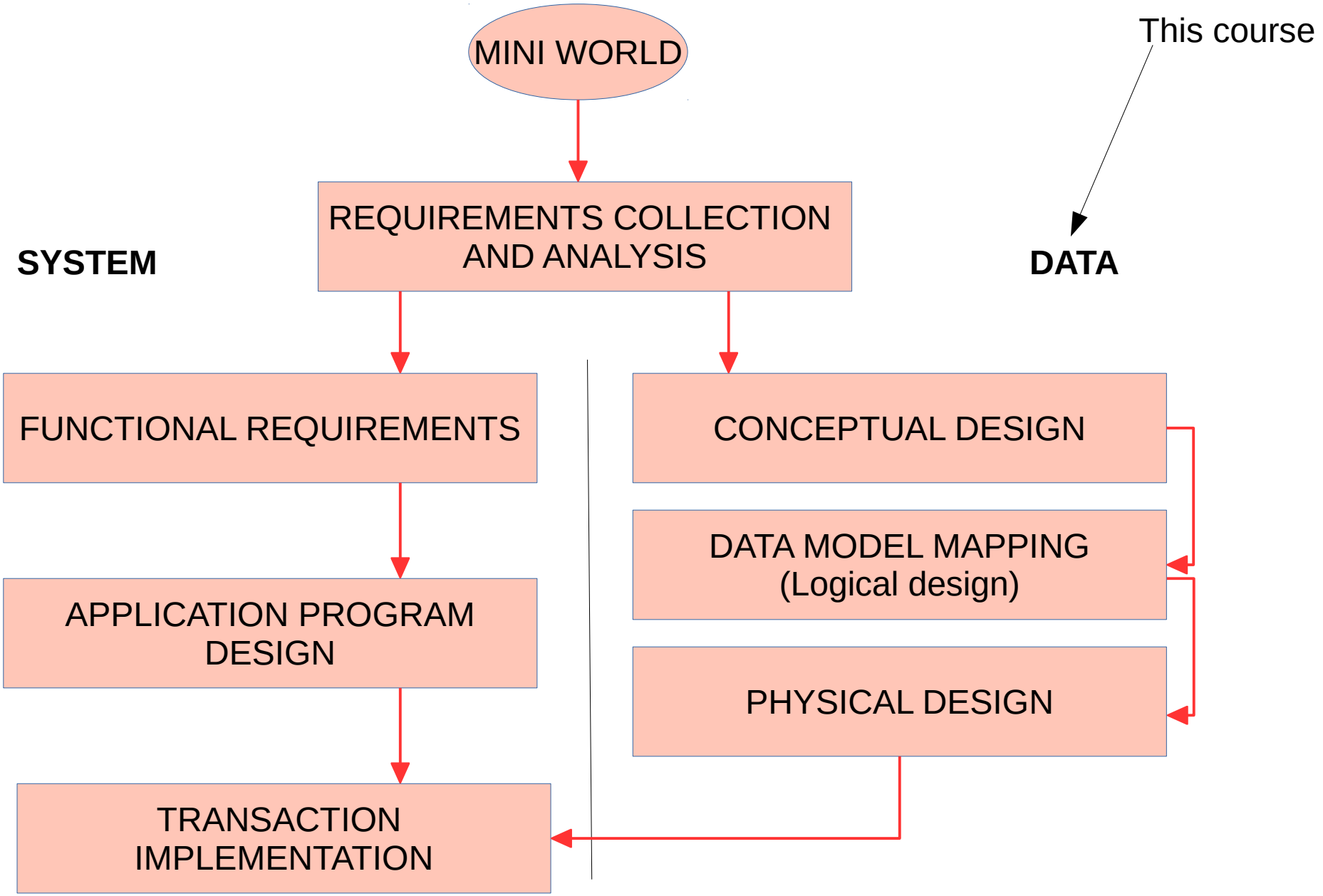
- Used directly
- Through a system (Information System)

In real life it is usually done both ways: the design is made for a system, but there are users who can access and use the information directly (through an interface).

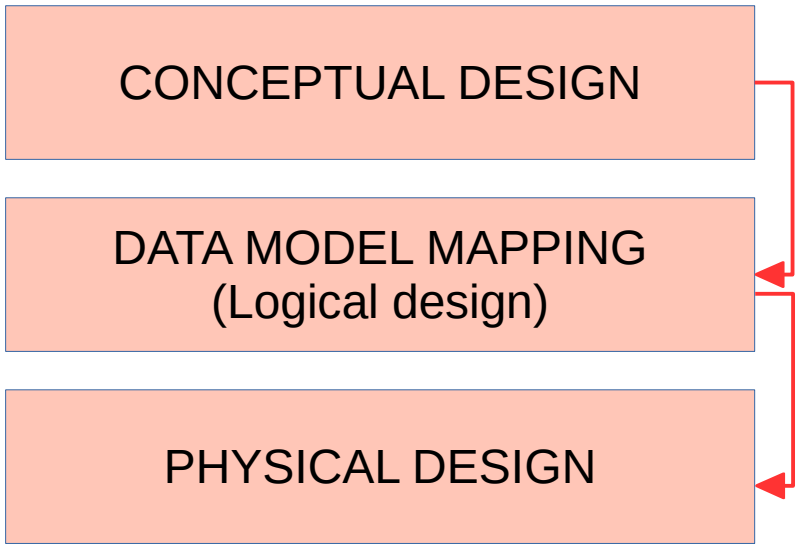
Information System:

So we are going to assume that what we need to do is to design an information system.

INFORMATION SYSTEM (APPLICATION) DESIGN



DATA MODEL



The key idea here is MODEL

What we do is a model of the actual data that is used (needed) in real life.

If in real-life there are 57 students enrolled in CS 336 (summer 2016), then an Information System used by the university to model enrollment should contain that data somewhere.

One of the advantages of using a model is that we can obtain data from the model without having to “observe” real life every time we need information.

The operations performed by the system should mimic those on real life.

DATA STORAGE:

Once we have our model, we need to store its data somewhere.

The first idea that comes to mind is to store the data in files.

- Sequential (access) files: Data can only be read/stored sequentially. In order to read element 10, I would have to read first elements from 1 through 9. It is not a very effective way of storing data.

- Random access files. In order to be able to effectively use random access files, we need to have efficient algorithms for insertion, deletion, and searching.

How would we modify an element in a sequential file?
How would we modify an element in a random access file?

CONCURRENCY

In any case. There is always going to be a problem with concurrency. It might happen that two users want to access the same file location at the same time.

Example:
Alice and Bob have a joint bank account.
Alice wants to withdraw $30 and Bob wants to withdraw $40. The account balance is $45. Both go to a different ATM machine at the same time and do the following:

Alice:

Request account balance: $45
Withdraw $30 (local)
Compute new account balance: $15 (local)
Store new account balance: $15

Bob:

Request account balance: $45
Withdraw $40 (local)
Compute new account balance: $5 (local)
Store new account balance: $5

What could the operations on a centralized system be?

Request account balance $45 (Alice)
Request account balance $45 (Bob)
Store new account balance: $5 (Bob)
Store new account balance: $15 (Alice)

The final balance stored in the account is $15 after withdrawing $70 from a account with an initial balance of $45

DBMS (database management system)

We need to keep our data in a system that:

- Can take care of concurrency issues,
- Stores the data in such a way that the user does not have to be concerned (or know) where or how the data is being stored
- Provides efficient algorithms for insertion, deletion, search and update.
- Should be general enough so that I can define the structure of my data (schema).
- Should be reliable.
- Should not have limits on the amount of data stored.

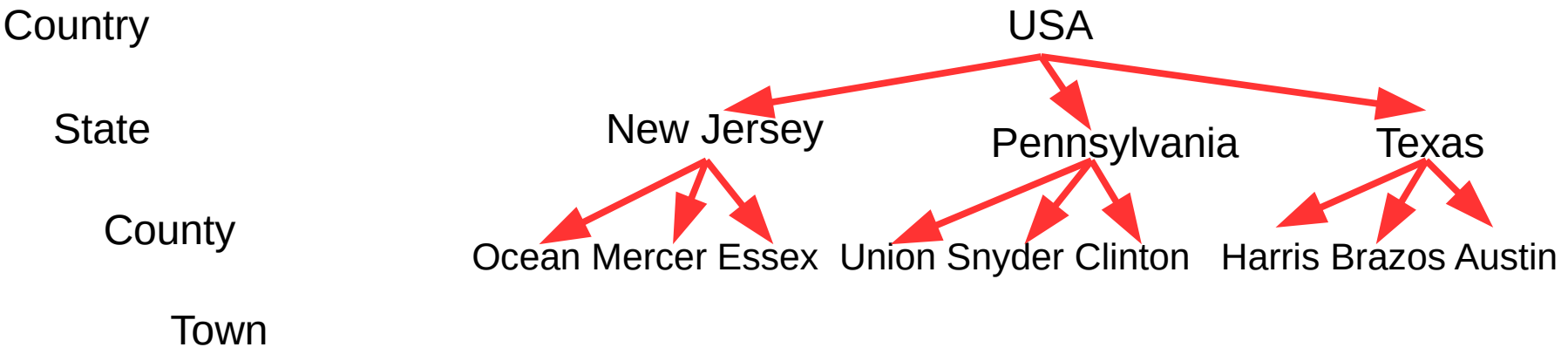
DBMS (a definition):

efficient, reliable, convenient, and safe !
multi-user storage of and access to massive!
amounts of persistent data

TYPES OF DATABASE MODELS:

Hierarchical Model (the 60's):

Assumes that all the data that is stored is organized hierarchically.
Example:



The Hierarchical model looks like a tree, and elements are found by following the links.

Every node has just one parent

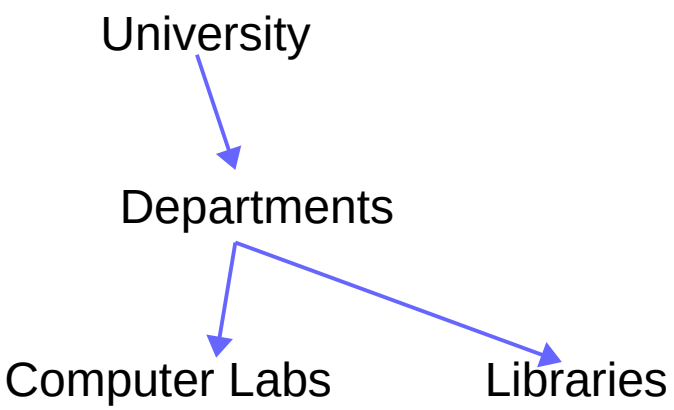
This model is still being used today in:

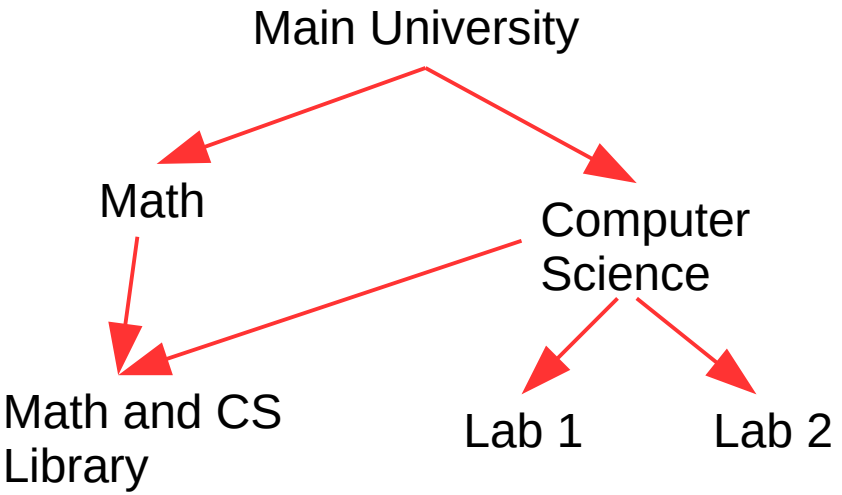
- File Systems
- Geographical Information Systems

Network Model (the 70's):

Assumes that **a given node might have more than one parent.**

Example:





Search by following pointers.

Update/Delete/Insert require lots of pointer operations to be performed

Relational Model

Developed in 1970 (Codd)
Used during the 80's

Top Relational DBMS in use:

MySQL
Oracle
SQL server (Microsoft)
DB2 (IBM)

Search by content… not by following links or pointers.

Basic structure: Relational Table

Abstraction for design: Entity Relationship Diagrams
(next topic)

Object Oriented Model

With the emergence and use of OO languages such as C++, Java, etc.

Used during the 90's

Lots of problems when communicating OO applications with databases in relational model.

Emergence of Post-Relational no-SQL content:

XML
AJAX

In order to try to circumvent the problems of OO – Database connectivity and work on a common set of data.

Used during the beginning of the 21st century.

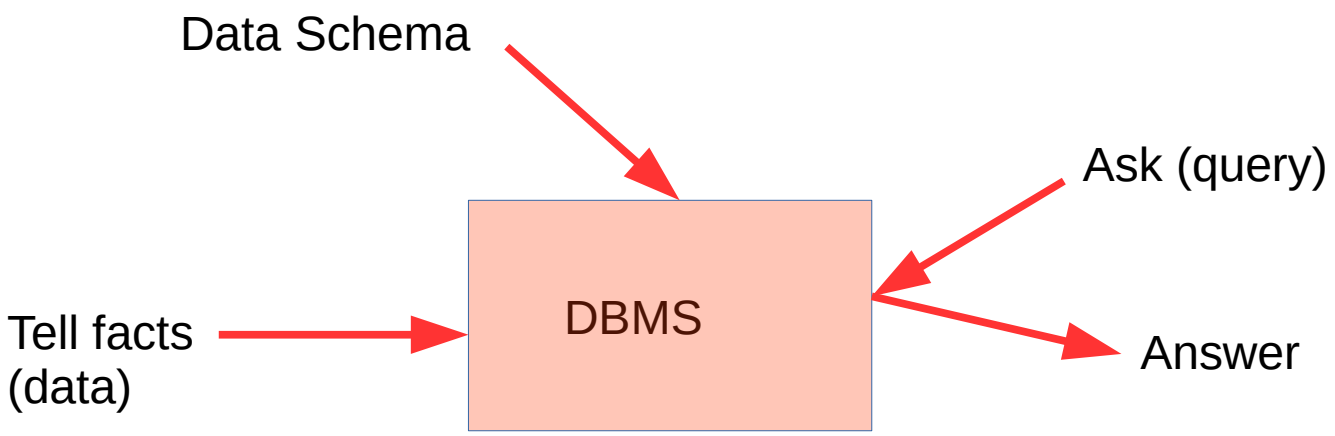
Theoretical vs Technical

there is a difference between training and education.!

If computer science is a fundamental discipline, then university education in this field should emphasize enduring fundamental principles rather than transient current technology.!

-Peter Wegner, Three Computer Cultures. 1970.!

Functional view of Information Management (1)



Need for languages:

- A language to provide the schema (L_{SCHEMA})
- A language to tell facts (L_{TELL})
- A language to ask questions or query language (L_{QUERY})
- A language in which the answer is provided (L_{ANSWER})

In more general terms we will call the DBMS just an Information Manager. Keep in mind that a DBMS is an Information Manager but not every Information Manager is a DBMS.

Functional view of Information Management (2)

There are three functions that act on an Information Manager (including its corresponding data):

SCHEMA
TELL
QUERY

There are functions that will modify the *IM* (information manager). And functions that will not modify the *IM*.

Can you tell which ones are which?

The QUERY function only reads information from the *IM*, so it does not modify it. The other two functions will modify the data stored in the *IM*.

Functional view of Information Management (3)

The functions will take the following inputs:

SCHEMA(*s*,*M*):

- $s \in L_{SCHEMA}$ is an element of the language used to provide the schema to the IM.
- *M* is an instance of the IM, including the state of the database a the time the SCHEMA operation is executed.

TELL(*t*,*M*):

- $t \in L_{TELL}$ is an element of the language used to tell facts.
- *M* is an instance of the IM, including the state of the database a the time the TELL operation is executed.

QUERY(*q*,*M*):

- $q \in L_{QUERY}$ is an element of the language used to query the IM.
- *M* is an instance of the IM, including the state of the database a the time the QUERY operation is executed.

Functional view of Information Management (3)

Formally we define the following functions:

SCHEMA: L_{SCHEMA} x IM $\longrightarrow$ IM

TELL: L_{TELL} x IM $\longrightarrow$ IM

QUERY: L_{QUERY} x IM $\longrightarrow$ L_{ANSWER}

EXAMPLE:

Assume that we want our IM to just store Strings.
We will query the IM to determine if a given String is stored in the IM.

What would the Languages (L_{SCHEMA}, L_{TELL}, L_{QUERY}, and L_{ANSWER}) be?

- Assume that L_{SCHEMA} looks like variable declarations in a Programming Language.
- $L_{TELL} = \Sigma^*$
- $L_{QUERY} = \Sigma^*$
- $L_{ANSWER} = \{yes, no\}$

Implementation of:

Insertion,
Searching

SAMPLE PROGRAM:

```
IM = new InformationManager()

IM = SCHEMA("String word",IM)

IM = TELL("dog",IM)
IM = TELL("cat",IM)
IM = TELL("cow",IM)
IM = TELL("hen",IM)

answer=QUERY("cow",IM)   // answer is "yes"
answer=QUERY("pig",IM)   // answer is "no"
```

Features of IM and DBMS

Desirable services provided by many IM

- **persistence** (maintain information even after program stops)!
- **convenient access** (ability to ask questions declaratively rather than programmatically; hide and change implementation; queries optimized to speed up answering)!
- **deal with massive amounts of facts told** (terabytes not uncommon; cannot be stored in main memory)!
- **performance** (high speed even in the presence of many operations and much data)!
- **maintain some notion of consistency in presence of multiple concurrent access**

Other features common to Database Management Systems (specialized IM):

- **resilience** (ability to survive hardware, software, power failures)!
- **reliability** (almost always up – phone companies use dbms!)!
- **scalability** (a recent phenomenon: data can grow incredibly fast: search engines cache the web) **"**

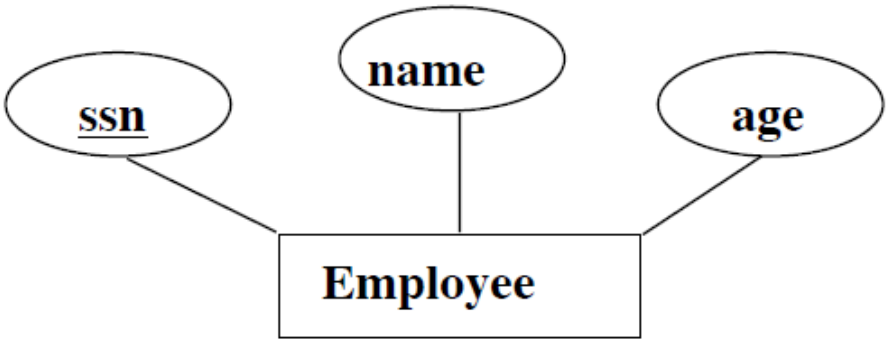
THE RELATIONAL MODEL

Entity Relationship Diagram

Example: Company

- The company is organized into departments. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager.
- Each department *controls* a number of projects. Each project has a name, number and a budget.
- For each employee we keep a social security number, name, age, address, salary, sex, and birthdate. Each employee *works in* one department but may *work on* several projects. We keep track of the time when the employee started working for the department. We also keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

ER Model Basics

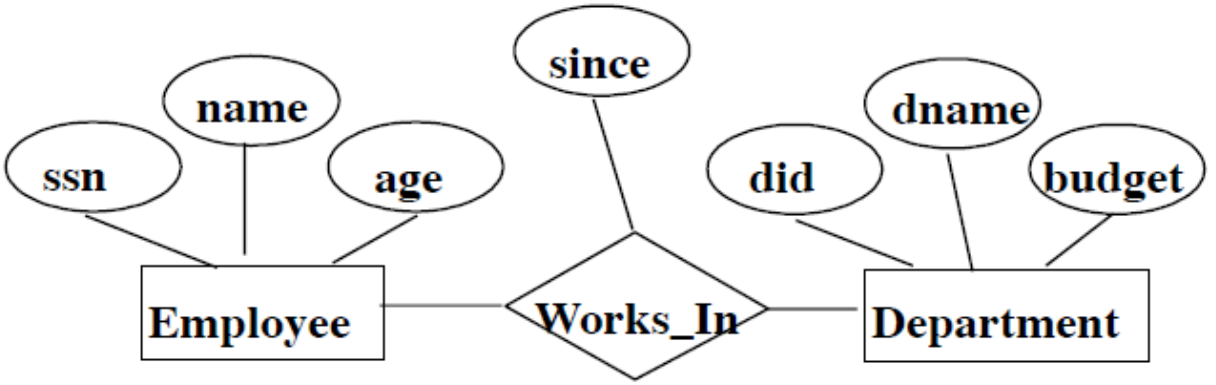


- *"Entity Set"*: A collection of similar individuals. E.g., all employees. (Like a *class* in ObjOriented programming language)
- *"Attribute"*: a simple value associated with the entity, usually describing it
  - **All entities in an entity set have the same set of attributes.**
- Each attribute has a "*domain* "
  - **base (value) type: int, float,... string, date,**
  - **<< some versions of ER allow composite attributes:**

    name[first,last]

    address[phone[area_code,local_phone],...]
  - **< < some versions of ER allow multi-valued attributes (e.g., color for Car**

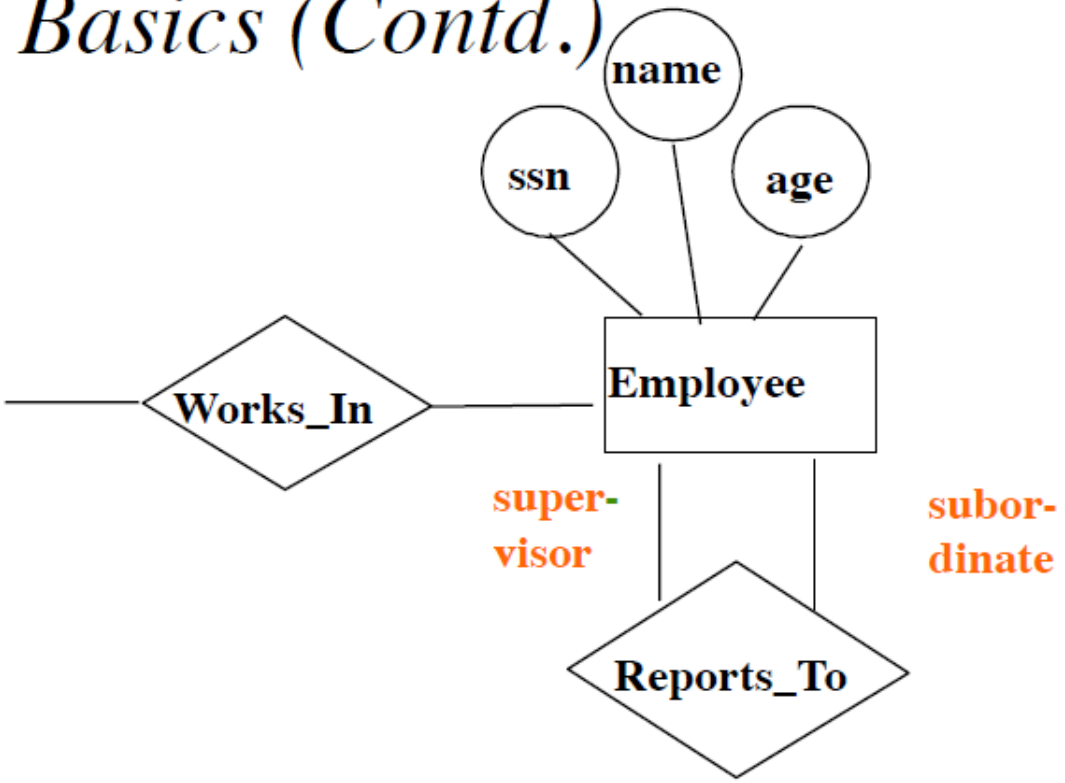
ER Model Basics (Contd.)



- *Relationship set*: Set of specific relationships/tuples between individuals in the entities linked to it. (e.g., *(mary, sales_dept)*)
  - **An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 : E1, ..., en : En**
  - (*Think of it as the specification of the predicate* ***worksIn***)
- *Relationship set attributes*: qualify nature of relationship (e.g. time)

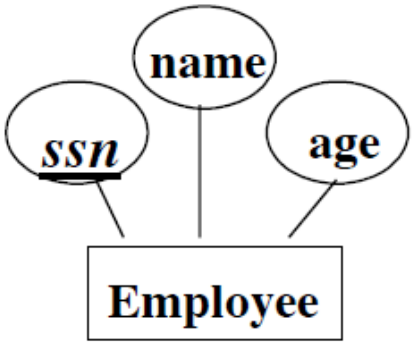
  ***since*(12/3/2002,*worksIn*(marge,sales))**

ER Model Basics (Contd.)



- Same entity set could participate in different relationship set, or in multiple ways in the same relationship set.
  - **To distinguish the participants, use "role names" as in *Reports_To***

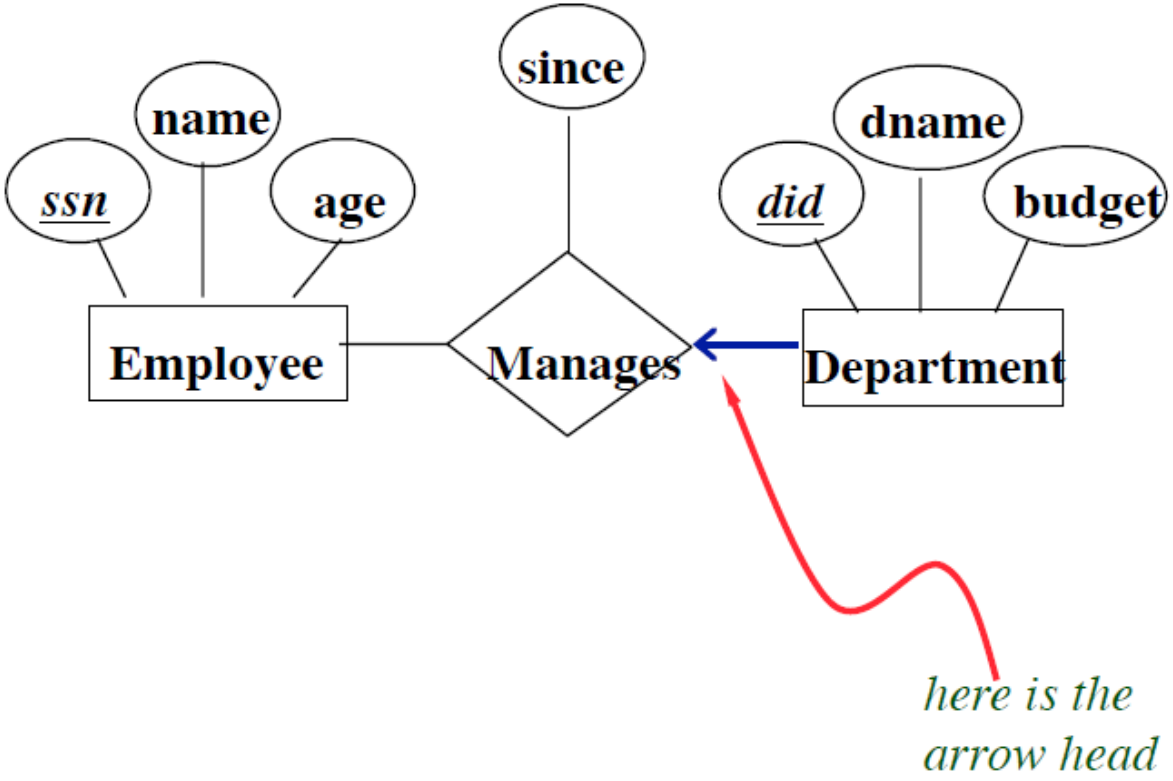
Constraints in ER diagrams (1): "key" for entities



- An entity key is a minimal set of attributes that identifies every individual in that entity set
- e.g., ***ssn*** uniquely identifies an employee
- if you needed both *ssn* and *name* to uniquely identify an employee, they would form a *composite* key
- Note that the set of all attributes must always uniquely identify the entity but it may not be minimal, hence not a key
- ***notation: underscore the attribute name(s) forming the key***

Constraints in ER diagrams (1): "key" for relationships

- suppose each department has at most one manager So a department can be thought of as key constraint on relationship ***Manages*** because there is at most one tuple with any department in the ***manages***(,) binary relation. *[marked as edge with an arrow on it in your text]*



(According to this diagram can an Employee manage more than one Dept?)