

# Big Data: Introduction to Databases

Juliana Freire



# Today

- Why study databases?
- Why use databases?
- Introduction to Relational Databases
- Representing structured data with the Relational Model
- Accessing and querying data using SQL

# Why study databases?

- Databases used to be *specialized applications*, now they are a *central component* in computing environments
- Knowledge of database concepts is *essential* for computer scientists and for anyone who needs to *manipulate* data

# Why study databases?

- Databases are everywhere, even when you don't see them: most activities involve data
  - Banking + credit cards: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Telecommunications/networks
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
  - **Web sites**: front end to information stored in databases; e.g., Google, YouTube, Flickr, Amazon...
  - Scientific research, e.g., studying the environment, cities, ...
- Data needs to be *managed*

# Why study databases?

- Data is valuable:
  - E.g., bank account records, tax records, student records, your videos and photos...
  - It must be protected - no matter what happens whether we have machine crashes, disk crashes, hurricanes/floods;
  - It also needs to be protected from **people**

# Why study databases?

- Data is often structured

Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address
1	26361767	09/24/2013 02:37:36 AM	09/24/2013 03:09:54 AM	NYPD	New York City Police Depa Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10304	645 BAY STREET
2	26361929	09/24/2013 02:34:20 AM	09/24/2013 03:17:14 AM	NYPD	New York City Police Depa Blocked Driveway	No Access	Street/Sidewalk	11205	31 ELLIOTT PLACE
3	26366604	09/24/2013 02:20:22 AM	09/24/2013 02:32:46 AM	NYPD	New York City Police Depa Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk	10012	620 BROADWAY
4	26362779	09/24/2013 02:14:43 AM	09/24/2013 03:15:10 AM	NYPD	New York City Police Depa Blocked Driveway	Partial Access	Street/Sidewalk	11218	562 EAST 9 STREET
5	26364390	09/24/2013 02:10:33 AM		NYPD	New York City Police Depa Blocked Driveway	Partial Access	Street/Sidewalk	11379	58-15 86 STREET
6	26366039	09/24/2013 02:04:47 AM		DOHMH	Department of Health and Indoor Air Quality	Chemical Vapors/Gases/Odors	3+ Family Apartment Building	10031	541 WEST 142 STREET
7	26365752	09/24/2013 02:04:40 AM	09/24/2013 02:28:36 AM	NYPD	New York City Police Depa Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk	10458	2357 CROTONA AVENUE
8	26364370	09/24/2013 02:03:19 AM		DOHMH	Department of Health and Rodent	Condition Attracting Rodents	Commercial Building	10460	605 MORRIS PARK AVENUE
9	26361893	09/24/2013 01:56:32 AM	09/24/2013 03:04:45 AM	NYPD	New York City Police Depa Blocked Driveway	No Access	Street/Sidewalk	11370	20-15 74 STREET
10	26364368	09/24/2013 01:55:45 AM		DOHMH	Department of Health and Rodent	Condition Attracting Rodents	1-2 Family Dwelling	10460	584 MORRIS PARK AVENUE
11	26364997	09/24/2013 01:54:56 AM		DOHMH	Department of Health and Rodent	Condition Attracting Rodents	1-2 Family Dwelling	10460	582 MORRIS PARK AVENUE
12	26363537	09/24/2013 01:46:30 AM		NYPD	New York City Police Depa Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10304	645 BAY STREET
13	26367406	09/24/2013 01:42:14 AM		DOHMH	Department of Health and Rodent	Condition Attracting Rodents	Commercial Building	10460	605 MORRIS PARK AVENUE
14	26361809			IMH	Department of Health and Rodent	Condition Attracting Rodents	Commercial Building	10460	605 MORRIS PARK AVENUE
15	26362539			IMH	Department of Health and Rodent	Condition Attracting Rodents	Commercial Building	10460	605 MORRIS PARK AVENUE
16	26364157			IMH	Department of Health and Rodent	Condition Attracting Rodents	Commercial Building	10460	605 MORRIS PARK AVENUE

## Steven Spielberg

From Wikipedia, the free encyclopedia

**Steven Allan Spielberg** (born December 18, 1946)<sup>[1]</sup> is an American **film director**, **screenwriter**, and **film producer**. In a career spanning six decades, Spielberg's films have taken up many themes and genres. Spielberg's early **science-fiction** and **adventure films** were seen as an archetype of modern **Hollywood blockbuster** filmmaking. In later years, his films began addressing such issues as the **Holocaust**, **slavery**, war and **terrorism**.


Spielberg won the **Academy Award for Best Director** for *Schindler's List* (1993) and *Saving Private Ryan* (1998). Three of Spielberg's films - *Jaws* (1975), *E.T. the Extra-Terrestrial* (1982), and *Jurassic Park* (1993) - achieved **box office** records, each becoming the highest-grossing film made at the

Steven Spielberg	
	
<b>Born</b>	Steven Allan Spielberg December 18, 1946 (age 63) Cincinnati, Ohio, U.S.
<b>Occupation</b>	Film director, producer, screenwriter
<b>Years active</b>	1964–present
<b>Spouse(s)</b>	Amy Irving (m. 1985–1989) Kate Capshaw (m. 1991–present)



See larger photo

## Hitachi Deskstar 7K500 - hard drive - 500 GB - SATA-300




**\$53** and up (6 stores)  **cashback** · 2%  
★★★★★ **user reviews** (1)

The Hitachi Deskstar 7K500 hard disk drive extends the company's long-standing tradition of performance and reliability leadership. Hitachi's standardized features in desktop solutions enable fast transfer rates, low power utilization and quiet acoustics.... [more...](#)

Share  [Facebook](#)  [Twitter](#)  [Email](#)

See also: [Product Summary](#) · [Where to Buy](#) · [User Reviews](#) · [Expert Reviews](#) · [Specifications](#)

## WHERE TO BUY »

PRODUCT	SELLER	PRICE	
 <b>Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal</b>	ServerSupply.com	<b>\$53.00</b>	<a href="#">Go to store</a>
 <b>Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal</b>	ALLHDD.COM	<b>\$64.00</b>	<a href="#">Go to store</a>
 <b>Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal</b>	Assembly Alliance Electronics	<b>\$69.69</b>	<a href="#">Go to store</a>

# Why study databases?

- Data is often structured
- We can exploit this regular structure
  - To retrieve data in useful ways (that is, we can use a *query* language)
  - To store data efficiently

# Why study Databases?

- Because the database field has made a number of contributions to basic computer science
  - Databases are behind many of important contributions and impact that CS has had
  - Find, gather, analyze and understand data, e.g., Banks, human genome, ecommerce, Web:
- *Understand concepts and apply to different problems and different areas, e.g., Big Data*
- Because DBMS software is highly successful as a commercial technology (Oracle, DB2, MS SQL Server...)
- Because DB research is highly active and **\*\*very\*\*** interesting!
  - Lots of opportunities to have practical impact



# Database Systems: The Basics



# A Simple Data Management Problem: Address Book

- Solution
  - Create a text file
- Advantages
  - Easy to add and modify
  - Easily copied (e.g., for backup, or paper dump)
  - Shareable (as a unit)
  - Substring searchable
  - Powerful, programmable tools
- But there can be complications...

# Complication 1: File Gets Very Large

- Problem:
  - Searching gets slow and imprecise
  - Search for “Elm Street” yields “Wilhelm Streeter”
- Solution
  - Add indexes over fields commonly searched upon
  - Structure data into fields
    - Search for street=“Elm Street”

## **Database Concepts:**

- *Record organization*
- *Indexes*

## Complication 2: Data Redundancy

- Why?
  - Large families, frequent moves
  - Might forget to update addresses of some family members
  - Want space economy, single point of update
  - Importance of residence as separate entity: 1 Xmas card each
- Solution:
  - Separate residences from names: 2 files, one for persons, one for residence
  - But how do we associate a residence with a person?
  - How many residences can a person have? 0? 1? Several?

### **Database Concepts:**

- ***Consistency***
- ***Normalization***
- ***Foreign keys***

## Complication 3: Multiple Associations Of Persons and Residences

- Meaning:
  - People can own, rent, manage, visit residences
  - May want constraints on numbers of residences per person
- Examples:
  - many-one (single family), many-many (rich folks), one-many (builder)

### **Database Concepts:**

- ***Relationships***
- ***Cardinality constraints***
- ***Consistency***

# Complication 4: Need To Add Information For New Purposes

- Examples:
  - Xmas cards sent and received
  - Post office gives big discount for using Zip+4 addressing
- Requirements:
  - Adding fields and/or new tables

**Database Concept:**  
• *Schema evolution*

# Complication 5: Doing Ad Hoc Analysis and Retrieval

- Example:
  - “Who have we sent cards to each of the past 5 years, but received 2 or fewer cards in return?”
- Requires:
  - Language for expressing analysis and retrieval
  - Implementation that performs analysis and retrieval correctly and efficiently

## **Database Concepts:**

- *Query languages*
- *Query optimization and execution*

# Complication 6: Want To Organize The Data Differently For Some Users

- Examples:
  - Other family members want to see names and residences together
  - You don't want your kids to see your business entries
- Solution:
  - Use stored queries as “windows” onto the database
  - Data not selected by query is “not there”

## **Database Concepts:**

- ***Joins***
- ***Views***
- ***Security***



## Complication 7: Required Existence Of Associated Data

**Database Concept:**

• *Referential integrity*

- Examples:
  - Can't send a Xmas card to someone without an address
  - Names are not unique unless qualified by residence: the John Jones living at 123 Elm Street
- Solutions:
  - Refuse to insert a name unless it is associated with an address
  - Refuse to delete an address if it is associated with a name
  - Or, tolerate multiple non-unique names

# Complication 8: Want Programmed Access To Data

- Meaning:
  - Want to write a Java program to search, display, update entries
- Solution:
  - Use data organization to define corresponding datatypes
  - Use access library to open, retrieve, update data

## **Database Concepts:**

- ***Database schemas***
- ***API***
- ***Embedded querying***

# Complication 9: Multiple Updates On All Or None Basis

- Examples:
  - Two households merge
  - Requires changing residences of several persons
  - What if your computer crashes during updates?
- Solution:
  - Present illusion that all updates are done simultaneously
  - Implemented by commit or rollback of entire piece of work

## **Database Concept:**

- *Transactions*
- *Atomicity*

# Complication 10: Your Computer Crashes (Again)

- Will your data still be present
  - Uncorrupted?
  - In what state, given that a transaction was in progress?
- Solution:
  - Make sure old data are safely accessible until latest commit

## **Database Concept:**

- *Data durability*
- *Recovery*

# Complication 11: Two Computers In Your Household

- How can data be shared?
  - USB key? Ugh, multiple version headaches
  - Dropbox – changes can be overwritten
  - Let's assume the database is shared somehow
  - What if one user is merging households, another is splitting one up?
  - What are meaningful results?
- A common policy:
  - Transactions are atomic
  - They appear to run one after the other, in some order

## Database Concepts:

- *Transaction isolation*
- *Concurrency control*
- *Transaction serializability*

# Complication 12: A Home Computer And A Business Computer

- Is there one database or two?
  - Want speed, reliability of local data at each site
  - But logically, one database for maintenance and querying
  - Data communication between them (most of the time ... )
  - Want some capability for independent operation (robustness)
- Solutions:
  - Personal data on the home computer
  - Business data on the business computer
  - Common logical view

## Database Concepts:

- *Distributed databases*
- *Data partitioning*
- *Data replication*

# Complication 13: Your Uncle Louie Gets The Genealogy Bug

- His grand vision:
  - All family members pool their databases over the Internet
  - Together, all genealogy relationships can be recorded
- But:
  - Aunt Sarah is paranoid: will not reveal birthdates
  - You are too: you don't want your business associates in the genealogy database
  - Everyone wants complete control over safety of their own data
  - People use different formats for records, and different name abbreviations for entries

## **Database Concepts:**

- ***Federated databases***
- ***Data integration***

# Complication 14: You Become President

- Of USA, of University, of a large organization
  - Your address list grows to hundreds of thousands or more
  - You realize it contains useful information *in the large*
- Examples
  - Which are top 10 zip codes on the list?
  - Which zip codes have addresses that are most likely to send cards to you when you send cards to them?
  - Which of those zip codes are in states that had less than 5% difference in Republican / Democratic presidential votes in 2004?

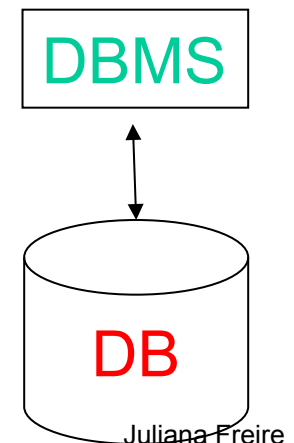
## **Database Concepts:**

- ***Data mining***
- ***Online analytical processing***



# Databases and Database Management Systems

- **Database (DB)** is an integrated collection of data
  - Models real-world objects
    - Entities (e.g., people, residence, Christmas cards)
    - Relationships (e.g., John Doe lives on 123 Elm St)
  - Captures *structure* – allows data to be **queried**
- A **Database Management System (DBMS)** is a software suite designed to store and manage databases
  - Provides environment that is both *convenient* and *efficient* to use.
  - Address all *complications* discussed



# Storing Data: Database vs File System

- Once upon a time database applications were built on top of file systems...
- But this has many drawbacks:
  - Data redundancy, inconsistency and isolation
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task, e.g., search people by zip code or last name; update telephone number
  - Integrity problems
    - Integrity constraints (e.g., num\_residence = 1) become part of program code -- hard to add new constraints or change existing ones
  - Failures may leave database in an inconsistent state with partial updates carried out, e.g., John and Mary get married, add new residence, update John's entry, and database crashes while Mary's entry is being updated...

# Storing Data: Database vs File System (cont.)

- Concurrent access by multiple users
  - Needed for performance: can you imagine if only 1 person at a time could buy a ticket from Delta?
  - Uncontrolled concurrent access can lead to inconsistencies, e.g., the same seat could be sold multiple times...
    - There are 3 seats left; I buy 2 seats; John buys 3 seats at the same time
    - If I hit enter 1<sup>st</sup> there will be 1 seat left; if John is faster there will be 0; but 5 seats have been sold and we will fight at the airport!

**Database systems offer solutions to all the above problems**

# Why use Database Systems?

- Data independence and efficient access
  - Easy + efficient access through declarative query languages and optimization
- Data integrity and security
  - Preventing inconsistencies, safeguarding data from failures and malicious access
- Concurrent access
- Reduced application development time
- Uniform data administration

# When not to use Database Systems?

# What's in a DBMS?

“above  
the water”

“below  
the water”

<i>Data model</i>	<i>Query language</i>	<i>Transactions and crash recovery</i>
Logical DB design Relational Model XML data model	SQL, QBE, views XPath, XQuery	Transactions
Map data to files Clustering Indexes	Query optimization Query evaluation	Locking Concurrency control Recovery Logs

# Designing a database: The Conceptual Model

- What are the *entities* and *relationships* among these entities in the application?
- What information about these entities and relationships should we store in the database?
- What are the *integrity constraints* or *business rules* that hold?
- Different applications have different needs, and different perspectives – even to model the *same* object
  - billing department: patient(id, name, insurance, address)  
visit(patientId, procedure, date, charge)
  - inpatient: patient(id,name,age,address)  
alergies(id,alergies)  
prescription(patientId,date,medicine)

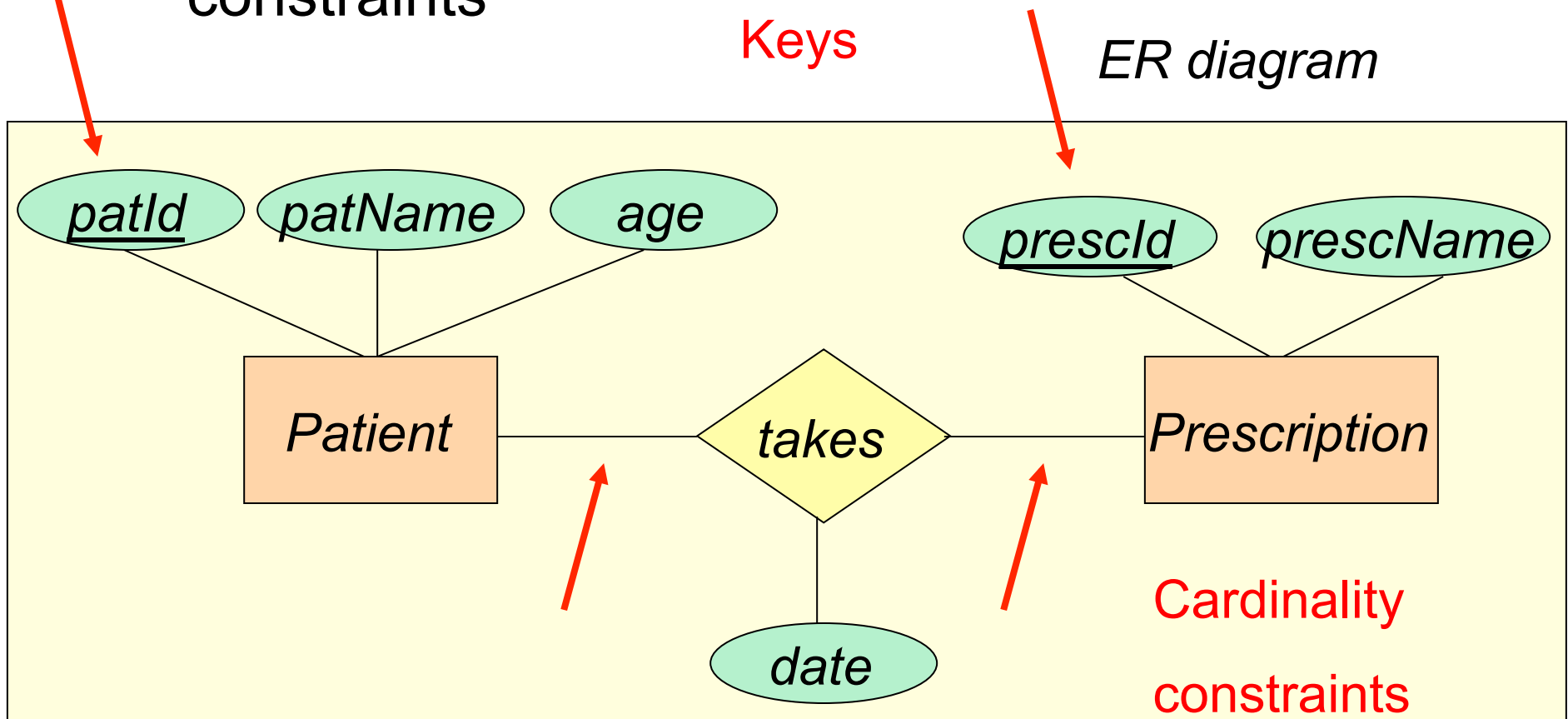
# Designing a database: The Conceptual Design

- What are the *entities* and *relationships* among these entities in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - Different *semantics* of the application from different perspectives – even to model the same object
- billing department: patient(id, name, insurance, address)  
visit(patientId, procedure, date, charge)  
inpatient: patient(id,name,age,address)  
allergies(id,allergies)  
prescription(patientId,date,medicine)



# The Entity Relationship (ER) Data Model

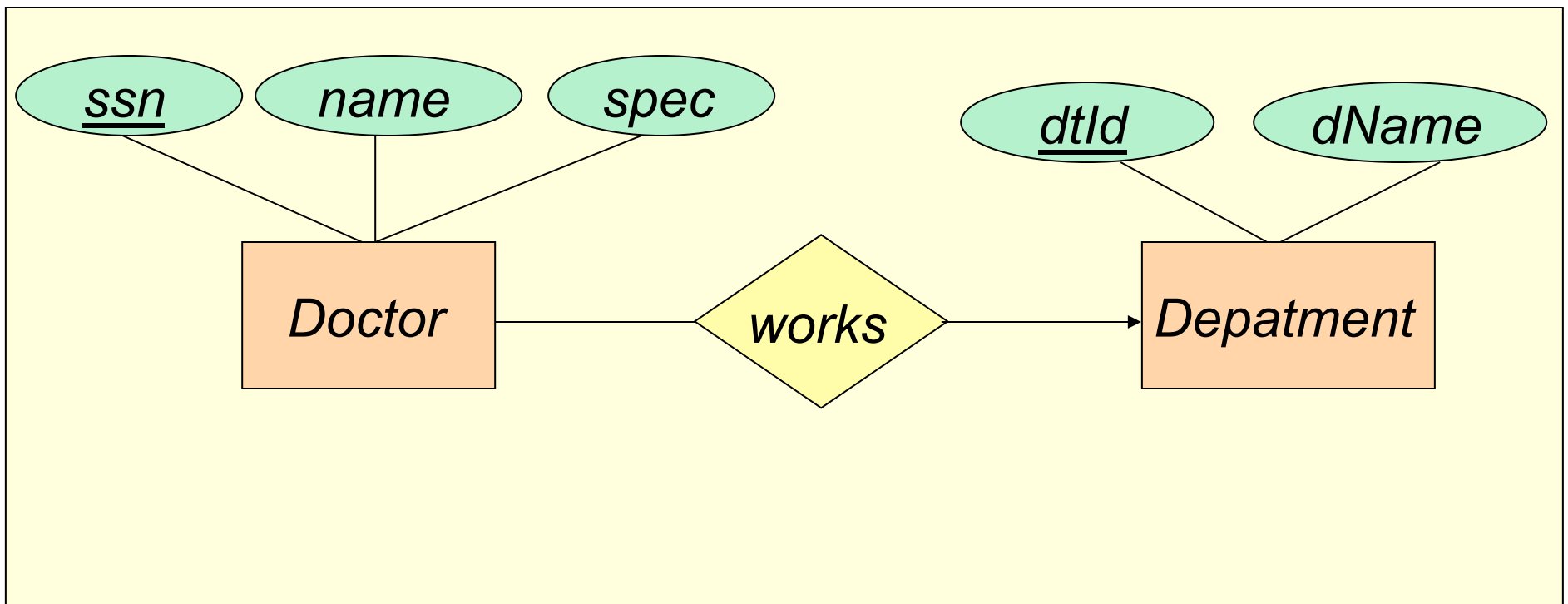
- A *data model* is a collection of concepts for describing data, relationships, semantics and constraints



## ER: Another Example

- A department has many doctors, but a doctor can only work in one department

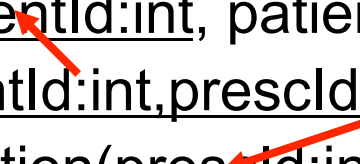
*ER diagram*



# Relational Data Model

- ER used for conceptual design is then mapped into the relational model
- The *relational model of data* is the most widely used model today
  - Main concept: *relation*, basically a table with rows and columns
  - Every relation has a *schema*, which describes the columns, or fields
- A *schema* is a description of a particular collection of data, using a given data model

Patient(patientId:int, patientName:str, age: int)  
Takes(patientId:int,presId:int,prescDate:date)  
Prescription(presId:int, presName:str)

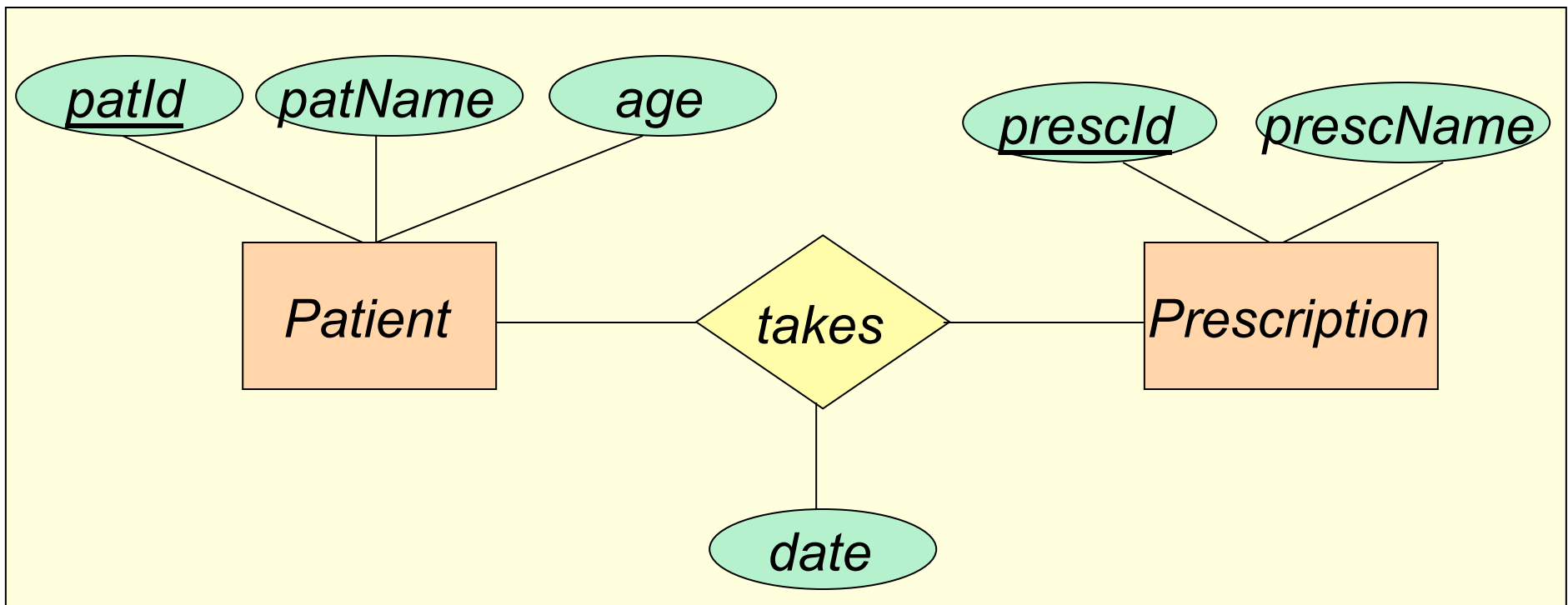


## ER to Relational

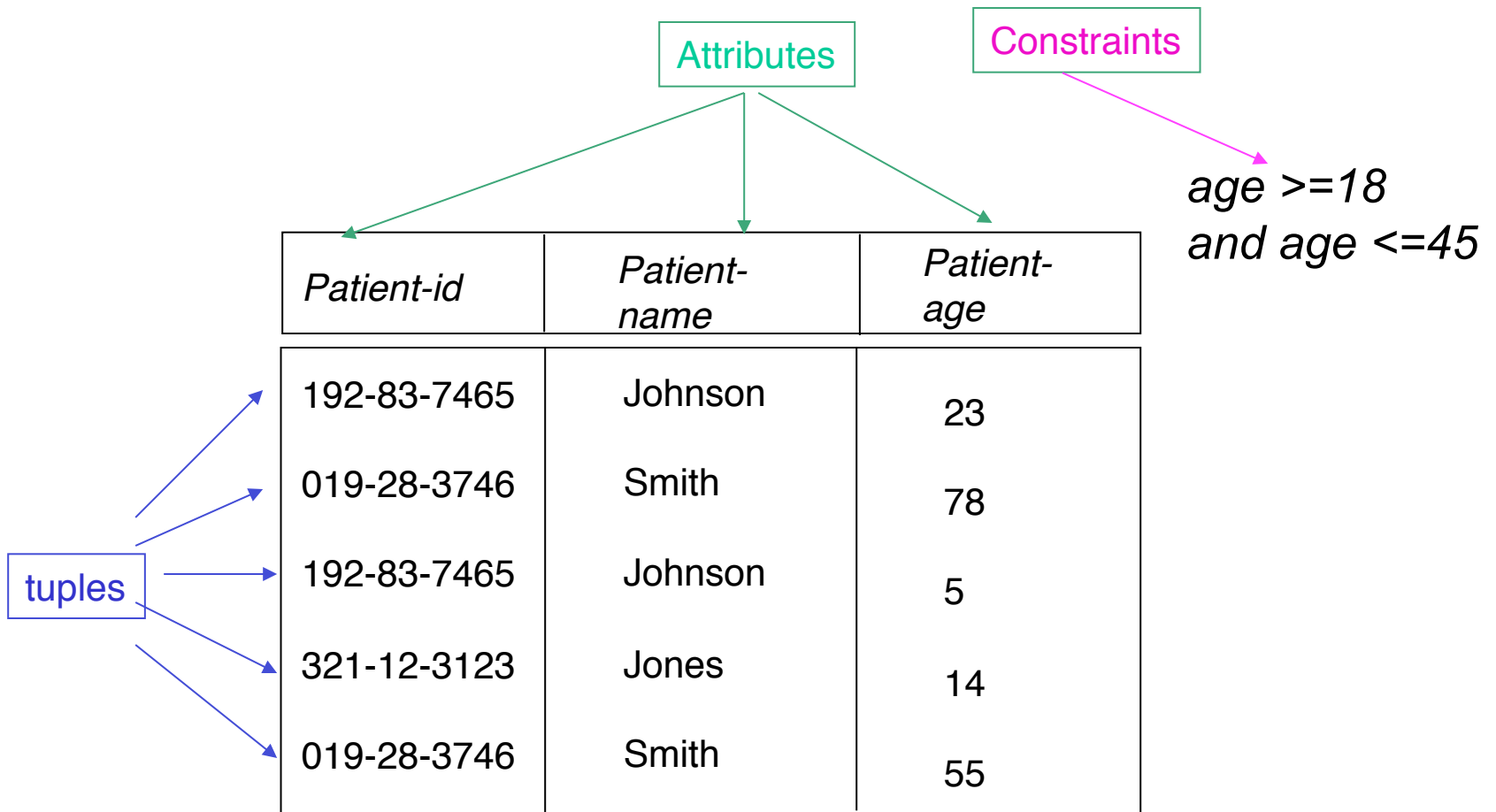
Patient(patientId:int, patientName:str, age: int)

Takes(patientId:int,prescId:int,prescDate:date)

Prescription(prescId:int, presName:str)



# Relational Model: Terminology



Patient(patientId:int, patientName:str, age: int)

# Pitfalls in Relational Database Design

- Find a “good” collection of relation schemas
- Bad design may lead to
  - Repetition of information → inconsistencies!
    - E.g., keeping people and addresses in a single file
  - Inability to represent certain information
    - E.g., a doctor that is both a cardiologist and a pediatrician
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Ensure constraints are properly modeled: updates check for violation of database integrity constraints

# Query Languages

- *Query languages*: Allow *manipulation* and *retrieval* of data from a database
- Queries are posed wrt **data model**
  - Operations over objects defined in data model
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for optimization
- Query Languages **!=** programming languages
  - QLs support easy, efficient access to large data sets
  - QLs not expected to be “Turing complete”
  - QLs not intended to be used for complex calculations

# Query Languages

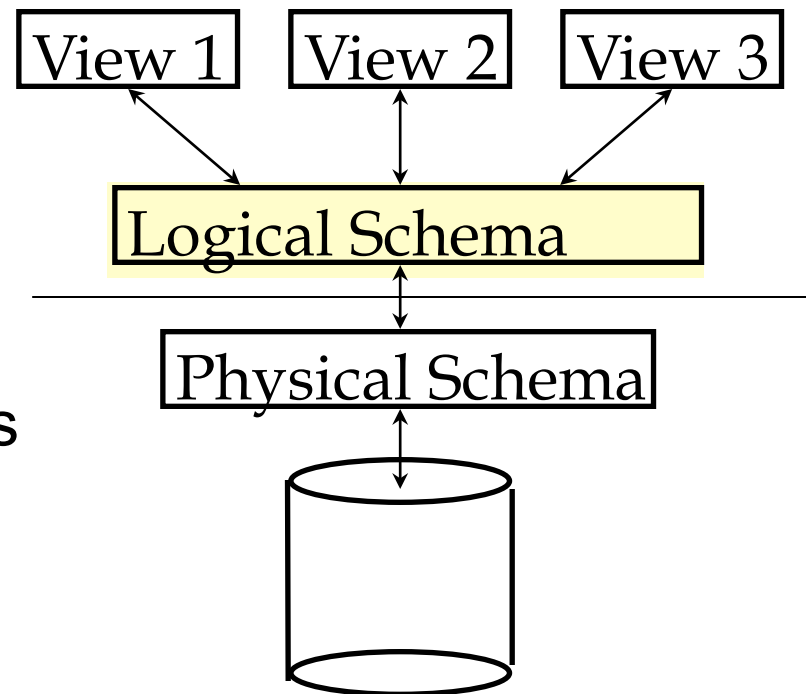
- *Query languages*: Allow *manipulation* and *retrieval* of data from a database.
- Relational model supports simple powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages **!=** programming
  - QLs not expected to be “Turing complete”.
  - QLs support easy, efficient access to large data sets.
  - QLs not intended to be used for complex calculations.

a language that can  
compute anything  
that can be  
computed



# Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*
  - Views describe how users see the data
  - Logical schema defines logical structure
  - Physical schema describes the files and indexes used



*Key to good performance*

# Example: University Database

- Physical schema:
  - Students stored in id order
  - Index on last name
- Logical 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)*
- External Schema (View):
  - *Course\_info(cid:string,enrollment:integer)*

# Data Independence

- Applications insulated from how data is structured and stored
- *Logical data independence*: Protection from changes in *logical* structure of data
  - Changes in the logical schema do not affect users as long as their *views* are still available
- *Physical data independence*: Protection from changes in *physical* structure of data
  - Changes in the physical layout of the data or in the indexes used do not affect the *logical* relations

*One of the most important benefits of using a DBMS!*

*“above  
the water”*

*“below  
the water”*

<i>Data model</i>	<i>Query language</i>	<i>Transactions and crash recovery</i>
Logical DB design Relational Model XML data model	SQL, QBE, views XPath, XQuery	Transactions
Map data to files Clustering Indexes	Query optimization Query evaluation	Locking Concurrency control Recovery Logs


Let' s dive now...

---

# Storage and Indexing

- The *DB administrator* designs the physical structures
- Nowadays, database systems can do (some of) this automatically: autoadmin, index advisors
- File structures: sequential, hashing, clustering, single or multiple disks, etc.
- Example – Bank accounts
  - Good for:  
List all accounts in the Downtown branch
  - What about:  
List all accounts with balance = 350

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	



# Storage and Indexing

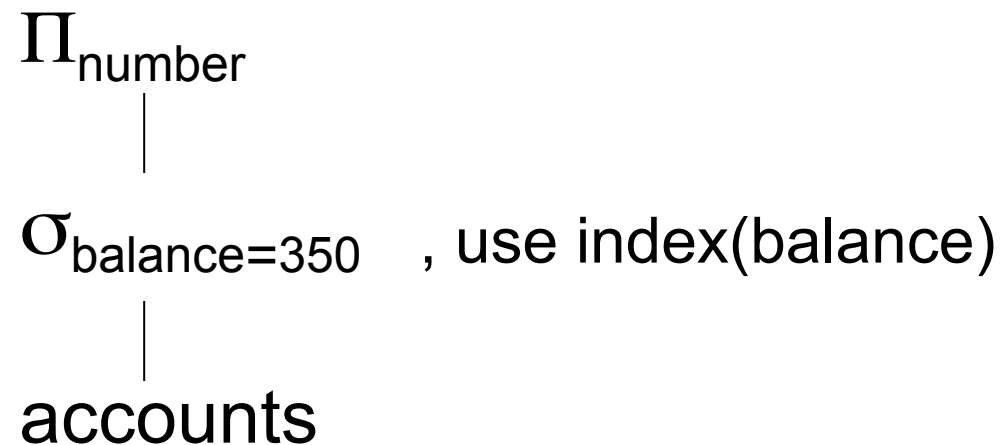
- Indexes:
  - Select attributes to index
  - Select the type of index
- **Storage manager** is a module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system:
  - interaction with the file manager
  - efficient storing, retrieving and updating of data

# Query Optimization and Evaluation

- DBMS must provide efficient access to data
  - In an emergency, can't wait 10 minutes to find patient allergies
- **Declarative** queries are translated into **imperative** query plans
  - Declarative queries → logical data model
  - Imperative plans → physical structure
- Relational **optimizers** aim to find the best imperative plans (i.e., shortest execution time)
  - In practice they avoid the worst plans...

## Example: Query Optimization

select number  
from accounts  
where balance = 350





# Transaction: An Execution of a DB Program

- Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes)
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins
  - Ensuring that a transaction (run alone) preserves consistency is ultimately the programmer's responsibility!
- **Transaction-management** component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures
  - DBMS ensures *atomicity* (all-or-nothing property)

# Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance
- But interleaving actions of different user programs can lead to inconsistency
  - e.g., nurse and doctor can simultaneously edit a patient record
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system

## Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a transaction
  - If there is power outage, will the patient database become inconsistent?
- **Idea:** Keep a *log* (history) of all actions carried out by the DBMS while executing a set of transactions
  - **Before** a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are *undone* using the log; and if log entry wasn't saved before the crash, corresponding change was not applied to database!

## Databases make these folks happy ...

- End users
- DBMS vendors: \$20B+ industry
- DB application programmers
- Database administrator (DBA)
  - Designs logical /physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve



# Summary

- DBMS used to maintain and query (*large*) structured datasets
- Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security
- Levels of abstraction give data independence

# The Relational Model

Juliana Freire

# Database Model

- Provides the means for
  - *specifying particular data structures*
  - *constraining the data sets* associated with these structures, and
  - *manipulating the data*
- Data *definition* language (DDL): define structures and constraints
- Data *manipulation* language (DML): specify manipulations/operations over the data

# Different Data Models

- **Relational**
- **Semi-structured/XML**
- Object-oriented
- Object-relational
- Hierarchical
- Network
- ...

*Will be covered in this course*



# Why Study the Relational Database Model?

- Extremely useful and simple
  - Single data-modeling concept: relations = 2-D tables
  - Allows clean yet powerful manipulation languages
- Most widely used model
  - Vendors: IBM, Microsoft, Oracle
  - Open source: MySQL
- Some competitors: object-oriented model, **semi-structured (XML) model**
- A synthesis emerging:
  - *Object-relational model*: Informix Universal Server, UniSQL, O2, Oracle, DB2
  - XML-enabled databases: Oracle, DB2, SQLServer

## Example: A Relation

*Relation  
name*

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

The *Account* relation keeps track of bank accounts.

Facts about real-world entities:

*J. Smith owns a checking account whose number is 101 and balance is 1000.00*

## Why do we ‘need’ this model?

- Why not use tables defined in Java or C?
- The relational model provides *physical independence*
  - Tables can be stored in many different ways, but they have the same *logical* representation
- Operations can be expressed in *relational algebra*
  - Table-oriented operations---simple
  - Limited set of operations is a strength: queries can be automatically optimized

## Example: Attributes of a Relation

The name of the attributes (columns)

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Attribute domains:

**Number** must be a 3-digit number

**Owner** must be a 30-character string


**Type** must be “checking” or “savings”

# Basic Structure

- Given sets  $D_1, D_2, \dots, D_n$  a **relation**  $r$  is a subset of  $D_1 \times D_2 \times \dots \times D_n$   
Thus a relation is a **set** of n-tuples  $(a_1, a_2, \dots, a_n)$  where each  $a_i \in D_i$
- Each attribute of a relation has a name
- Set of allowed values for attribute is called the **domain** of the attribute
- Attribute values are required to be **atomic**, that is, indivisible
  - multi-valued attribute values **are not atomic**
  - composite attribute values **are not atomic**
- The special value *null* is a member of every domain
  - Phone number = null
  - Number is unknown; number does not exist

# Example: Relation Schema

The schema for the relation



Account			
Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

**Account(Number, Owner, Balance, Type)**

The **schema** sets the structure of the relation--it is the **definition** of the relation.

*(Note: the schema specifies more information than what is shown, e.g., types, keys, constraints...)*


# Relation Schema

- $A_1, A_2, \dots, A_n$  are *attributes*
- $R = (A_1, A_2, \dots, A_n)$  is a *relation schema*  
E.g. Account-schema =  
 $(number, owner, balance, type)$
- $r(R)$  is a *relation* on the *relation schema*  $R$   
E.g.,  $account$  (Account-schema)
  - Says “account is a relation conforming to Account-schema”
- Attributes of a relation form a *set*, not a list!
  - We often must specify a standard order

# Relation Instance

An instance of the relation...

the current contents or data in the relation.



Account			
Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking



## Relation Instance (cont.)

Another instance of the relation  
(two rows added, one (103) deleted)

Account			
Number	Owner	Balance	Type
101	J. Smith	1,000.00	checking
102	W. Wei	2,000.00	checking
103	M. Jones	1,000.00	checking
104	H. Martin	10,000.00	checking
105	W. Yu	7,500.00	savings
107	R. Jones	432.55	checking
109			

103 deleted

new

# Terminology for Relational Databases

The intension of the table

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

The extension of the table

# Rows/Tuples

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

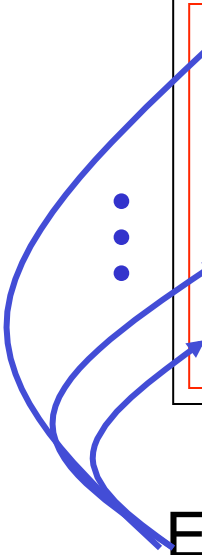
Each entry in the relation is called a **row**, or a **tuple**, or a **record**.

Order of tuples is irrelevant. Why?

# Rows/Tuples

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking



Each entry in the relation is called a **row**, or a **tuple**, or a **record**.

**Order** of tuples is irrelevant. Why?  
relation is a **set**, not a **list**!

# Relation Schema and Attributes

Account( Number, Owner, Balance, Type )  
=  
Account( Owner Number, Balance, Type )

**Order** of attributes is irrelevant: attributes in a relation schema form a set

Often choose a *standard* order

## Challenge Question

- How many different ways are there to represent a relation instance if instance has:
  - 3 attributes and 3 tuples?
  - $N$  attributes and  $M$  tuples?

# Degree and Cardinality

Degree or **arity** of a relation is the number of attributes

Degree of this relation (or table) is 4  
because there are 4 attributes

Cardinality of this instance is 5 (because there are 5 rows)

Account	Number	Owner	Balance	Type
→	101	J. Smith	1000.00	checking
→	102	W. Wei	2000.00	checking
→	103	J. Smith	5000.00	savings
→	104	M. Jones	1000.00	checking
→	105	H. Martin	10,000.00	checking

**Cardinality** of a relation = the number of rows in the current instance

# Relational Database

- A database consists of multiple relations
- Information is broken up---each relation store one part of the information  
E.g.: *account* : information about accounts  
      *deposit*: information about deposits into accounts  
      *check* : information about checks
- What would happen if we stored all information in a single table?
  - E.g., *bank(account-number, balance, customer-name, deposit-date, deposit-amount..)*



# Relational Database

- A database consists of multiple relations
- Information is broken up---each relation store one part of the information  
E.g.: *account* : information about accounts  
*deposit*: information about deposits into accounts  
*check* : information about checks
- What would happen if we stored all information in a single table?
  - repetition of information (e.g., two customers own an account, two deposits on the same account)
  - the need for null values (e.g., represent a customer without an account)

To avoid these problems we *normalize* databases

# Relational Database Example

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/00	125.00
	101	925	10/24/00	23.98

## Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/99	23.98

Each Relation has a key.... where the values must be unique.

# Keys

- Let  $K \subseteq R$
- $K$  is a **superkey** of  $R$  if values for  $K$  are sufficient to identify a unique tuple of each possible relation  $r(R)$ 
  - Example:  $\{account-number, account-owner\}$  and  $\{account-number\}$  are both superkeys of *Account*, if no two accounts can possibly have the same number.
- $K$  is a **candidate key** if  $K$  is minimal  
Example:  $\{account-number\}$  is a candidate key for *Account* – it is a superkey and no subset of it is a superkey

## Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00
	106	5	12/5/00	555.00



Is this legal?

If not, how do we prevent it from happening?

## Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00
	<del>106</del>	<del>5</del>	<del>12/5/00</del>	<del>555.00</del>

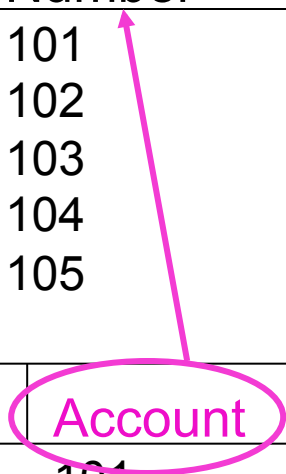
We say that **Deposit.Account** is a foreign key that references **Account.Number**. If the DBMS enforces this constraint we say we have **referential integrity**.

## Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98



Similarly, Check.Account is a foreign key that references Account.Number.

# Specification of a Relational Schema

- Select the relations, with a **name for each table**
- Select **attributes for each relation** and give the **domain for each attribute**
- Specify the **key(s)** for each relation
- Specify all appropriate **foreign keys** and **integrity constraints**
- *Database schema* is the set of schemas for the set of relations



## Another Example: A University Database

(keys are underlined, domains omitted)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

## Example Database (cont.)

(with foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

What foreign keys are present in the Completed table?

## Example Database (cont.)

(with foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

Foreign keys in the Completed table are shown above.

# **CREATING A TABLE IN A RELATIONAL DATABASE**

# Creating a Relation in SQL

- Simplest form is:

```
CREATE TABLE <name> (  
    <list of elements>  
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

## Elements of Table Declarations

- Most basic element: an attribute and its type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR( $n$ ) = fixed-length string of  $n$  characters.
  - VARCHAR( $n$ ) = variable-length string of up to  $n$  characters.

## Example: Create Table

```
CREATE TABLE Account (  
    number    DECIMAL (20, 0) ,  
    owner     VARCHAR (30) ,  
    balance   REAL ,  
    type      CHAR (10)  
);
```

# SQL Values

- Integers and reals are represented as you would expect
- Strings are too, except they require single quotes
  - Two single quotes = real quote, e.g., 'Joe's Bookstore'.
- Any value can be NULL



## Dates and Times

- DATE and TIME are types in SQL.
- The form of a date value is:

DATE 'yyyy-mm-dd'

– **Example:** DATE '2007-09-30' for Sept. 30, 2007

```
CREATE TABLE Check (  
    account    CHAR(20) ,  
    check      INT ,  
    cDate      DATE ,  
    amount     REAL) ;
```

## Times as Values

- The form of a time value is:

TIME 'hh:mm:ss'

with an optional decimal point and fractions of a second following.

- **Example:** TIME '17:00:02.5' = two and a half seconds after 5:00PM.

## Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE
- Either says that *no two tuples of the relation may agree in all the attribute(s) on the list*
- There are a few distinctions to be mentioned later

## Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Account (  
  number    CHAR(20) PRIMARY KEY,  
  owner     VARCHAR(30),  
  balance   REAL,  
  type      CHAR(10)  
);
```

## Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute
- Example:

```
CREATE TABLE Account (  
  number      CHAR(20)  UNIQUE,  
  owner       VARCHAR(30), NULLs are allowed  
  balance     REAL,      for number  
  type        CHAR(10)  
);
```

## Declaring Multi-attribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement
- This form is essential if the key consists of more than one attribute
  - May be used even for one-attribute keys

## Example: Single-Attribute Keys

```
CREATE TABLE Account (  
  number    CHAR(20) ,  
  owner     VARCHAR(30) ,  
  balance   REAL ,  
  type      CHAR(10) ,  
  PRIMARY KEY (number)  
);
```

## Example: Multiattribute Key

- The account and check together are the key for Check:

```
CREATE TABLE Check (  
    account CHAR(20) ,  
    check INT ,  
    cDate DATE ,  
    amount REAL ,  
    PRIMARY KEY (account, check)  
);
```



# PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

# Semi-structured Data Model

- Another data model, based on trees
- Motivation:
  - flexible representation of data
  - sharing of *documents* among systems and databases

# How did it all start?

- Data on the Web
  - Web provides a universal standard for information exchange
  - Publish and share (HTML) files in fixed addresses (URL) through a std protocol (HTTP)
  - Document structure does not reflect structure of data
- Data on the Web vs. Databases
  - Structure: relational schemas
  - Sharing data → query languages, mechanisms for concurrency control and recovery. Need to preserve data integrity!
  - Physical independence: logical view to query the data; physical for efficiency

# Data on the Web: A Limitation

- Suppose an investment bank publishes financial data
  - Data lives in a RDBMS
  - HTML pages are generated using a combination of SQL queries and published on the Web
- Another organization needs to access this information
  - **PROBLEM:** can only access the HTML source ☹️
  - Solution: write a wrapper to extract structured information from HTML
    - Labor intensive, brittle
    - Inefficient: may need to download a whole DB in order to get a single interest rate! (*Would be a lot easier to query for this value*)

# Information in HTML

## 2. [The Advanced Html Companion](#)

by Keith Schengili-Roberts, Kim Silk-Copeland. Paperback (August 1998)

Our Price: \$35.96

You Save: \$8.99 (20%)

Usually ships in 24 hours

Average Customer Review: ★★★★★

## 3. [Applied XML Solutions \(Sams Professional Publishing\)](#)

by Benoit Marchal. Paperback (August 29, 2000)

Our Price: \$35.99

You Save: \$8.00 (20%)

Usually ships in 24 hours

Average Customer Review: ★★★★★

# How to bridge the DB and Web approaches?

```
<span class=small>
<table border=0 cellpadding=3 width=100%> <tr valign=top> <td><font size=-1><b>2.</b></font></td> <td align=center width="60">
<a href=/exec/obidos/tg/detail/-/0126235422/qid=1053550181/sr=1-2/ref=sr_1_2/103-8080549-6058246?v=glance&s=books>
</a> </td> <td width=100% valign=top> <font size=-1> <b>
<a href=/exec/obidos/tg/detail/-/0126235422/qid=1053550181/sr=1-2/ref=sr_1_2/103-8080549-6058246?v=glance&s=books>
The Advanced Html Companion</a></b> <br>
<font face=verdana,arial,helvetica size=-1>by Keith Schengili-Roberts, Kim Silk-Copeland</font> (<b>Paperback</b> - August 1998) <br>
Avg. Customer Rating:  <br> <font color=990000> </font> <span class=small><b>Editions: </b> Paperback </span> <br> </td> </tr> </table> <span class=tiny> Usually ships in 24
hours<br> </span> <table border=0 cellpadding=0 cellspacing=0 width=100%><tr><td width=50% valign=top> <table border=0 cellpadding=0
cellspacing=0> <tr> <td align=right valign=top class=small nowrap>List Price:</td> <td>&nbsp;</td> <td class=small> $44.95 </td></tr> <tr>
<td align=right valign=top class=small><a href=/exec/obidos/tg/detail/-/0126235422/103-8080549-6058246?v=glance&s=books>Buy new</
a>:</td> <td>&nbsp;</td> <td class=small> <b class=price>$35.96</b> </td></tr> </table> </td> <td width=50% valign=top> <a href=http://
www.amazon.com/exec/obidos/tg/stores/offering/list/-/0126235422/all/ref=sr_pb_a/103-8080549-6058246><span class=small>Used &amp;
new</span></a> <span class=tiny>from</span> <span class=price>$5.85</span> </td></tr></table> <br><br> </span>
```

# Data Exchange: A Limitation

- Suppose an investment bank exchanges financial data with a partner
  - Data lives in a RDBMS
  - Serialize data in CSV, DB dump, ...
- Another organization needs to access this information
  - **PROBLEMS:**
    - CSV does not have a schema
    - DB dump is incompatible
    - Bank's schema is different from the partner's schema
  - Solution: agreement between partners
  - *But what happens when there are many partners?*

# XML: A First Step Towards Convergence

- XML = *Extensible Markup Language*.
- XML is syntactically related to HTML
- Goal is different:
  - HTML describes document structure
  - XML transmits textual data
- XML *solves* the data exchange problem
  - No need to write specialized wrappers.
  - The schema of an XML document serves as a contract
- XML *does not solve* the problem of efficient access
  - DB research is doing that!
  - Storage techniques, mechanisms for maintaining data integrity and consistency,...

## Why XML?

- Lingua franca of the Web
- Web's secret sauce
- Next silver bullet
- <Your favorite motto here/>



# XML In Action

## SwissProt data

# XML encoding of SwissProt

ID GRAA\_HUMAN STANDARD; PRT; 262 AA.  
AC P12544;  
DT 01-OCT-1989 (REL. 12, CREATED)  
DT 01-OCT-1989 (REL. 12, LAST SEQUENCE UPDATE)  
DT 15-DEC-1998  
DE GRANZYME RA GERSHENFELD H.K., HERSHBERGER R.J., SHOWS T.B., WEISSMAN I.L.;  
DE 1) (HANUKKIN TRYPTASE) (TRYPTASE) (TRYP) (GRANZYME 1) (CYTOTOXIC PROTEINASE);  
DE (FRAGMENTIN 1).  
GN GZMA OR CTLA3 OR HFSP.  
OS HOMO SAPIENS (HUMAN).  
OC Eukaryota; Metazoa; Chordata; Vertebrata; Mammalia; Eutheria;  
OC Primates; Catarrhini; Hominidae; Homo.  
RN [1]  
RP SEQUENCE FROM N.A.  
RC TISSUE=T-CELL;  
RX MEDLINE; 88125000.  
RA GERSHENFELD H.K., HERSHBERGER R.J., SHOWS T.B., WEISSMAN I.L.;  
RT "Cloning and chromosomal assignment of a human cDNA encoding a T  
cell- and natural killer cell-specific trypsin-like serine  
protease.";  
RL PROC. NATL. ACAD. SCI. U.S.A. 85:1184-1188(1988).  
RN [2]  
RP SEQUENCE OF 29-53.  
RX MEDLINE; 88330824.  
RA POE M., BENNETT C.D., BIDDISON W.E., BLAKE J.T., NORTON G.P.,  
RA RODKEY J.A., SIGAL N.H., TURNER R.V., WU J.K., ZWEERINK H.J.;  
RT "Human cytotoxic lymphocyte tryptase. Its purification from granules  
and the characterization of inhibitor and substrate specificity.";  
RL J. BIOL. CHEM. 263:13215-13222(1988).  
RN [3]  
RP SEQUENCE OF 29-40, AND CHARACTERIZATION.  
RX MEDLINE; 89009866.  
RA HAMEED A., LOWREY D.M., LICHTENHELD M., PODACK E.R.;  
RT "Characterization of three serine esterases isolated from human IL-2  
activated killer cells.";  
RL J. IMMUNOL. 141:3142-3147(1988).  
RN [4]  
RP SEQUENCE OF 29-39, AND CHARACTERIZATION.  
RX MEDLINE; 89035468.  
RA KRAEHENBUHL O., REY C., JENNE D.E., LANZAVECCHIA A., GROSCURTH P.,  
RA CARREL S., TSCHOPP J.;  
[...]

```
<?xml version="1.0" ?>
<cml title="SwissProtTree">
<molecule title="SwissProt file" scheme="SWISSPROT">
<string title="ID" scheme="SWISSPROT">GRAA_HUMAN</string>
</molecule>
</cml>
</?xml>
<list dictname="DT" title="Revision" scheme="SWISSPROT">
<string title="Comments">REL. 12, CREATED</string>
<date>1989-10-01</date>
</list>
<list dictname="DT" title="Revision" scheme="SWISSPROT">
<string title="Comments">REL. 12, LAST SEQUENCE UPDATE</string>
<date>1989-10-01</date>
</list>
<list title="AUTHORS">
<person>
<initials>H.K.</initials>
<surname>GERSHENFELD</surname>
</person>
<person>
<initials>R.J.</initials>
<surname>HERSHBERGER</surname>
</person>
<person>
<initials>T.B.</initials>
<surname>SHOWS</surname>
</person>
...
</list>
```

# XML In Action

2. **The Advanced Html Companion**

by Keith Schengili-Roberts, Kim Silk-Copeland. Paperback (August 1998)

Our Price: \$35.96

You Save: \$8.99 (20%)

Usually ships in 24 hours

Average Customer Review: ★★☆☆☆

3. **Applied XML Solutions (Sams Professional Publishing)**

by Benoit Marchal. Paperback (August 29, 2000)

Our Price: \$35.99

You Save: \$9.00 (20%)

Usually ships in 24 hours

Average Customer Review: ★★☆☆☆

4. **Applied XML: A Toolkit for Programmers**

by Thomas R. E. F. ... (2000)

Our

You

<Book>

<Title>The Advanced Html Companion</Title>

<Author> Keith Schengili-Roberts </Author>

<Author> Kim Silk-Copeland</Author>

<Price> 35.96</Price>

...

</Book>

XML

# What Are Benefits?

- Tags
  - Easier for machine & humans to parse
- Tree structure
  - Nodes with parent/child relationships
  - Easier to understand
  - Easier to enforce
  - Easier to navigate

RA GERSHENFELD H.K., HERSHBERGER  
R.J., SHOWS T.B., WEISSMAN I.L.;

```
<list title="AUTHORS">
  <person>
    <initials>H.K.</initials>
    <surname>GERSHENFELD</surname>
  </person>
  <person>
    <initials>R.J.</initials>
    <surname>HERSHBERGER</surname>
  </person>
  <person>
    <initials>T.B.</initials>
    <surname>SHOWS</surname>
  </person>
  ...
</list>
```

# IMDB Example : Data

XML Document =  
Tagged elements +  
Attributes + Text

```
<?xml version="1.0" standalone="yes"?>
<imdb>
  <show year="1993"> <!-- Example Movie -->
    <title>Fugitive, The</title>
    <review>
      <suntimes>
        <reviewer>Roger Ebert</reviewer> gives <rating>two thumbs
        up</rating>! A fun action movie, Harrison Ford at his best.
      </suntimes>
    </review>
    <review>
      <nyt>The standard Hollywood summer movie strikes back.</nyt>
    </review>
    <box_office>183,752,965</box_office>
  </show>
  <show year="1994"> <!-- Example Television Show -->
    <title>X Files, The</title>
    <seasons>4</seasons>
  </show>
</imdb>
```

XML vs. HTML?

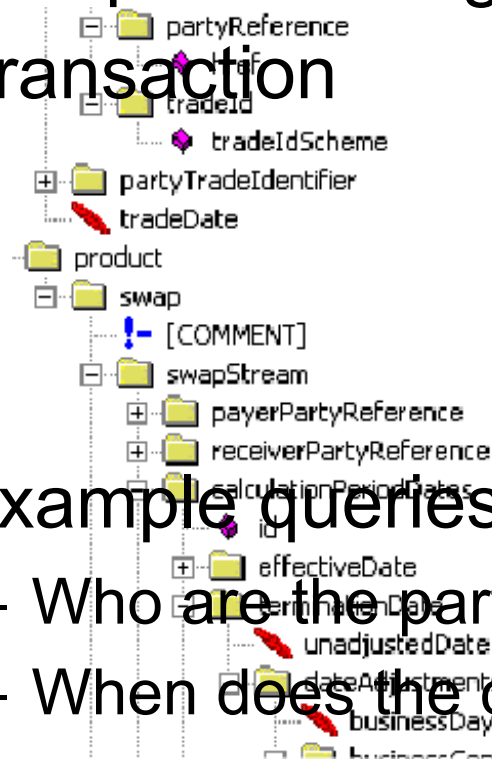
# IMDB Example : Schema

```
<element name="show">
  <complexType>
    <sequence>
      <element name="title" type="xs:string"/>
      <sequence minoccurs="0" maxoccurs="unbounded">
        <element name="review" mixed="true"/>
      </sequence>
      <choice>
        <element name="box_office" type="xs:integer"/>
        <element name="seasons" type="xs:integer"/>
      </choice>
    </sequence>
    <attribute name="year" type="xs:integer" use="optional"/>
  </complexType>
</element>
```

# FpML (finance)

- Complex nesting

- Transaction



- Example queries

- Who are the parties involved in a given contract?
- When does the contract expire?
- What are the various components of a contract?
- What the is the total amount of a contract?

#CHASE
TW9235
<a href="http://www.chase.com/swaps/trade-id">http://www.chase.com/swaps/trade-id</a>
1994-12-12
Chase pays the floating rate every 6 months,
FloatingCalcPeriodDates
1999-12-14
MODFOLLOWING

# HL7 (healthcare)

- Stream data (when coming from medical devices)
- Legacy data
- Transaction
- Temporal queries
- Example queries
  - Who is the patient?
  - When did the measurement take place?
  - For the duration of the measurement, what were the max and min value for the following vital signals (...)?



2-687005
Evans
Carolyn
1962-03-24
F
903 Diane Circle
Phoenixville
PA
19460
(610)555-1212

# Key Concepts in Databases

- Data Model: general conceptual way of structuring data
  - Relational: attributes, tuples, relations, SQL
  - XML: attributes nodes, trees, characters, XPATH/XQuery
- Schema: structure of a particular database under a certain data model
  - Relational: Definition of a set of relations + constraints
  - XML: Grammar for the structure of a document + constraints
- Instance: actual data conforming to a schema
  - Relational: Set of tables (instances of relations)
  - XML: Ordered tree



# Relational Model versus XML: Fundamental Differences

- Relations: Schema must be fixed in advance  
XML: Does not require predefined, fixed schema
- Relations: Rigid flat table structure  
XML: Flexible hierarchical structure (defined by regular expressions)
- Relations: simple structure, simple query language  
XML: complex structure, more complex query language

# Relational Model versus XML: Additional Differences

- Relations: Ordering of data not relevant (tuple ordering or attribute ordering)  
XML: Ordering forced by document format, may or may not be relevant
- Relations: Transmission and sharing can be problematic  
XML: Designed for easy representation and exchange
- Relations: "Native" data model for all current widely-used commercial DBMSs  
XML: "Add-on," often implemented on top of relations