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

- 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?

• 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
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
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 withdraws $300 from his account, before the money is dispensed, the ATM goes down…
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
    - 5 seats are 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?

• Some applications do not ‘fit’!
## What’s in a DBMS?

<table>
<thead>
<tr>
<th>Data model</th>
<th>Query language</th>
<th>Transactions and crash recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical DB design</td>
<td>SQL, QBE, views</td>
<td>Transactions</td>
</tr>
<tr>
<td>Relational Model</td>
<td>XPath, XQuery</td>
<td></td>
</tr>
<tr>
<td>XML data model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map data to files</td>
<td>Query optimization</td>
<td>Locking</td>
</tr>
<tr>
<td>Clustering</td>
<td>Query evaluation</td>
<td>Concurrency control</td>
</tr>
<tr>
<td>Indexes</td>
<td></td>
<td>Recovery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logs</td>
</tr>
</tbody>
</table>
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, allergies)
    - 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 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, allergies)
prescription(patientId, date, medicine)

Requires a good understanding of the semantics of the application.
The Entity Relationship (ER) Data Model

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

**ER diagram**

**Keys**

- patId
- patName
- age

**Prescription**

- prescId
- prescName
- date

**Patient**

**“takes”**

**Cardinality constraints**
ER: Another Example

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

ER diagram

- Doctor
  - ssn
  - name
  - spec

- Department
  - dtld
  - dName

- works
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, prescId:int, prescDate:date)
  Prescription(prescId:int, presName:str)
ER to Relational

Patient(patId:int, patName:str, age: int)
Takes(patId:int, prescId:int, prescDate:date)
Prescription(prescId:int, prescName:str)
### Relational Model: Terminology

**Attributes**

<table>
<thead>
<tr>
<th>Patient-id</th>
<th>Patient-name</th>
<th>Patient-age</th>
</tr>
</thead>
<tbody>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>23</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>78</td>
</tr>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>5</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>Jones</td>
<td>14</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>55</td>
</tr>
</tbody>
</table>

**Constraints**

\[ \text{age} \geq 18 \text{ and } \text{age} \leq 45 \]

**Schema**

`Patient(patientId:int, patientName:str, age: int)`
### Attributes

<table>
<thead>
<tr>
<th>Patient-id</th>
<th>Patient-name</th>
<th>Patient-age</th>
</tr>
</thead>
<tbody>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>23</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>78</td>
</tr>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>5</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>Jones</td>
<td>14</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>55</td>
</tr>
</tbody>
</table>

### Constraints

- age $\geq 18$
- and $\text{age} \leq 45$

### Anything wrong with this table?

Yes, there are inconsistencies in the patient ages:

- Johnson: 23, 5
- Smith: 78, 55
- Jones: 14

The ages should be consistent for each patient.
Pitfalls in Relational Database Design

• Find a “good” collection of relation schemas
• Bad design may lead to
  – Repetition of information $\rightarrow$ inconsistencies!
    • E.g., keeping people and addresses in a single file
      John Doe, 100 John St. apt 7, New York, NY
      Mary Doe, 100 John St. apt 7, New York, NY
      Jane Doe, 100 John St. apt 7, New York, NY
      John Smith, 234 Bleecker St., apt 1A, New York NY
      Mary Smith, 234 Bleecker St., apt 1A, New York NY
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 languages:
  - 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.
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!*
Let’s dive now…

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

<table>
<thead>
<tr>
<th>Account</th>
<th>Branch</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-217</td>
<td>Brighton</td>
<td>750</td>
</tr>
<tr>
<td>A-101</td>
<td>Downtown</td>
<td>500</td>
</tr>
<tr>
<td>A-110</td>
<td>Downtown</td>
<td>600</td>
</tr>
<tr>
<td>A-215</td>
<td>Mianus</td>
<td>700</td>
</tr>
<tr>
<td>A-102</td>
<td>Perryridge</td>
<td>400</td>
</tr>
<tr>
<td>A-201</td>
<td>Perryridge</td>
<td>900</td>
</tr>
<tr>
<td>A-218</td>
<td>Perryridge</td>
<td>700</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
</tbody>
</table>
A small detour: Disk Accesses

• DMBS stores data on disk
  – READ: transfer data from disk to main memory (RAM)
  – WRITE: transfer data from RAM to disk
  – Both are high-cost operations!

• Disk access costs:
  – Seek time (milliseconds)
  – Rotational latency (milliseconds)
  – Transfer rate (100 MB/sec)

• RAM (nanoseconds): factor of $10^6$ faster than disk
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 $\rightarrow$ logical data model
  – Imperative plans $\rightarrow$ 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
```

```
Π_{number}
│
σ_{balance=350} use index(balance)
│
accounts
```
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: $40B+ 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 emerged:
  – Object-relational model: Informix Universal Server, UniSQL, O2, Oracle, DB2
  – XML-enabled databases: Oracle, DB2, SQLServer
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

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

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 $D_1, D_2, \ldots, D_n$ a relation $r$ is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  Thus a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ 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 $\text{Account}(\text{Number, Owner, Balance, Type})$

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

• $A_1, A_2, \ldots, A_n$ are attributes
• $R = (A_1, A_2, \ldots, A_n)$ 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
An instance of the relation...

the current contents or data in the relation.

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
</tr>
</tbody>
</table>
Another instance of the relation (two rows added, one (103) deleted)

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1,000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2,000.00</td>
<td>checking</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1,000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
<tr>
<td>107</td>
<td>W. Yu</td>
<td>7,500.00</td>
<td>savings</td>
</tr>
<tr>
<td>109</td>
<td>R. Jones</td>
<td>432.55</td>
<td>checking</td>
</tr>
</tbody>
</table>
Rows/Tuples

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

Order of tuples is irrelevant. Why?
Rows/Tuples

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**!

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Transaction-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Check-number</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>924</td>
<td>10/23/00</td>
<td>125.00</td>
</tr>
<tr>
<td>101</td>
<td>925</td>
<td>10/24/00</td>
<td>23.98</td>
</tr>
</tbody>
</table>
### Relational Database Example (cont.)

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit</th>
<th>Account</th>
<th>Transaction-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Check</th>
<th>Account</th>
<th>Check-number</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>924</td>
<td>10/23/98</td>
<td>125.00</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>925</td>
<td>10/24/99</td>
<td>23.98</td>
<td></td>
</tr>
</tbody>
</table>

Each relation has a key: key 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: $\{\text{account-number, account-owner}\}$ and $\{\text{account-number}\}$
    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: $\{\text{account-number}\}$ is a candidate key for $Account$ – it is a superkey and no subset of it is a superkey
### Relational Database Example (cont.)

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

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

Is this legal? If not, how do we prevent it from happening?
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.
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:
  ```sql
  CREATE TABLE <name> (
      <list of elements>
  )
  ```
  
• To delete a relation:
  ```sql
  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:
  \[
  \text{TIME 'hh:mm:ss'}
  \]
  with an optional decimal point and fractions of a second following.
  
  – Example: \text{TIME '17:00:02.5'} = \text{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:

  ```sql
  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:

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

NULLs are allowed for number
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:

```sql
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 in 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
Information in HTML

2. **The Advanced Html Companion**  
   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: $8.99 (20%)  
   Usually ships in 24 hours  
   Average Customer Review: ★★★★★

How to bridge the DB and Web approaches?

---

Big Data – Spring 2016  
Juliana Freire
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

<table>
<thead>
<tr>
<th>ID</th>
<th>GRAA_HUMAN</th>
<th>STANDARD;</th>
<th>PRT;</th>
<th>262 AA.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>P12544;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>01-OCT-1989 (REL. 12, CREATED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>01-OCT-1989 (REL. 12, LAST SEQUENCE UPDATE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>15-DEC-1996</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>GRANZYME A PRECURSOR (EC 3.4.21.78) (CYTOTOXIC T-LYMPHOCYTE PROTEINASE 1) (HANUKKAH FACTOR) (H FACTOR) (HF) (GRANZYME 1) (CTL TRYPTASE) (FRAGMENTIN 1).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>GZMA OR CTLA3 OR HFSP.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>HOMO SAPIENS (HUMAN).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>Eukaryota; Metazoa; Chordata; Vertebrata; Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RN</td>
<td>[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>SEQUENCE FROM N.A.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>TISSUE=T-CELL;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>MEDLINE; 88125000.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>GERSHENFELD H.K., HERSHBERGER R.J., SHOWS T.B., WEISSMAN I.L.;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>&quot;Cloning and chromosomal assignment of a human cDNA encoding a T cell- and natural killer cell-specific trypsin-like serine protease.&quot;;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RN</td>
<td>[2]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>SEQUENCE OF 29-53.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>MEDLINE; 88330824.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>POE M., BENNETT C.D., BIDDISON W.E., BLAKE J.T., NORTON G.P.;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>&quot;Human cytotoxic lymphocyte tryptase. Its purification from granules and the characterization of inhibitor and substrate specificity.&quot;;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RN</td>
<td>[3]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>SEQUENCE OF 29-40, AND CHARACTERIZATION.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>MEDLINE; 89009866.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>&quot;Characterization of three serine esterases isolated from human IL-2 activated killer cells.&quot;;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RN</td>
<td>[4]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>SEQUENCE OF 29-39, AND CHARACTERIZATION.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>MEDLINE; 89035468.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>KRAEHENBUHL O., REY C., JENNE D.E., LANZAVECCHIA A., GROSCURTH P.;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>CARREL S., TSCHOPP J.;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| [..] | }
XML In Action

2. **The Advanced Html Companion**
   
   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 Bencit 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**
   
   <Book>
   <Title>The Advanced Html Companion</Title>
   <Author>Keith Schengili-Roberts</Author>
   <Author>Kim Silk-Copeland</Author>
   <Price>35.96</Price>
   ...
   </Book>
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

<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 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>
  </show>

  <show year="1994"> <!-- Example Television Show -->
    <title>X Files, The</title>
    <seasons>4</seasons>
  </show>
</imdb>

XML Document = Tagged elements + Attributes + Text
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 is the total amount of a contract?
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 (...)?
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
THANKS!