Introduction to Data Management
CSE 344

Lecture 15: NoSQL and JSon

Announcements

- Current assignments:
  - Homework 4 due tonight
  - Web Quiz 6 due next Wednesday
  - [There is no Web Quiz 5]
- Today's lecture:
  - JSon
  - The book covers XML instead (11.1-11.3, 12.1)

The New Hipster: NoSQL

NoSQL Motivation

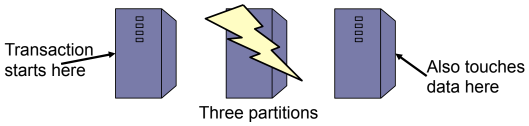
- Originally motivated by Web 2.0 applications
- Goal is to scale simple OLTP-style workloads to thousands or millions of users
- Users are doing both updates and reads

What is the Problem?

- Single server DBMS are too small for Web data
- Solution: scale out to multiple servers
- This is hard for the *entire* functionality of DMBS
- NoSQL: reduce functionality for easier scale up
  - Simpler data model
  - Simpler transactions

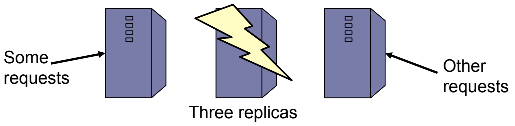
Scale Through Partitioning

- Partition the database across many machines in a cluster
  - Database now fits in main memory
  - Queries spread across these machines
- Can increase throughput
- Easy for reads but writes become expensive!



Scale Through Replication

- Create multiple copies of each database partition
- Spread queries across these replicas
- Can increase throughput and lower latency
- Can also improve fault-tolerance
- Easy for reads but writes become expensive!



Data Models

Taxonomy based on data models:

- ☞ Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Key-Value Stores Features

- **Data model**: (key,value) pairs
  - Key = string/integer, unique for the entire data
  - Value = can be anything (very complex object)
- **Operations**
  - Get(key), Put(key,value)
  - Operations on value not supported
- **Distribution / Partitioning**
  - No replication: key k is stored at server h(k)
  - 3-way replication: key k stored at h1(k),h2(k),h3(k)

How does get(k) work? How does put(k,v) work?

Flights(fid, date, carrier, flight_num, origin, dest, ...)
Carriers(cid, name)

Example

- How would you represent the Flights data as key, value pairs?
- Option 1: key=fid, value=entire flight record
- Option 2: key=date, value=all flights that day
- Option 3: key=(origin,dest), value=all flights between

How does query processing work?

Key-Value Stores Internals

- Data remains in main memory
- One type of impl.: distributed hash table
- Most systems also offer a persistence option
- Others use replication to provide fault-tolerance
  - Asynchronous or synchronous replication
  - Tunable consistency: read/write one replica or majority
- Some offer ACID transactions others do not
- Multiversion concurrency control or locking

Data Models

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- ☞ Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Document Stores Features

- **Data model**: (key,document) pairs
  - Key = string/integer, unique for the entire data
  - Document = JSon, or XML
- **Operations**
  - Get/put document by key
  - Limited, non-standard query language on JSon
- **Distribution / Partitioning**
  - Entire documents, as for key/value pairs

We will discuss JSon today

Data Models

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB
- ☞ Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS

Extensible Record Stores

- Based on Google's BigTable
- Data model is rows and columns
- Scalability by splitting rows and columns over nodes
  - Rows partitioned through sharding on primary key
  - Columns of a table are distributed over multiple nodes by using "column groups"
- HBase is an open source implementation of BigTable

JSon and Semistructured Data

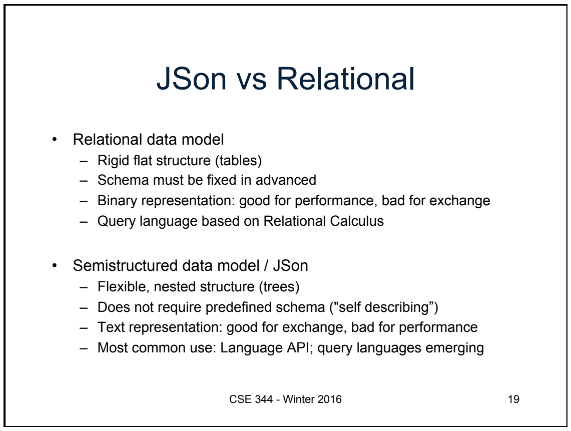
The Semistructured Data Model

- So far we have studied the ***relational data model***
  - Data is stored in tables(=relations)
  - Queries are expressions in the relational calculus (or relational algebra, or datalog, or SQL…)
- Today: Semistructured data model
  - Popular formats today: XML, JSon, protobuf

JSON - Overview

- JavaScript Object Notation = lightweight text-based open standard designed for human-readable data interchange. Interfaces in C, C ++, Java, Python, Perl, etc.
- The filename extension is .json.

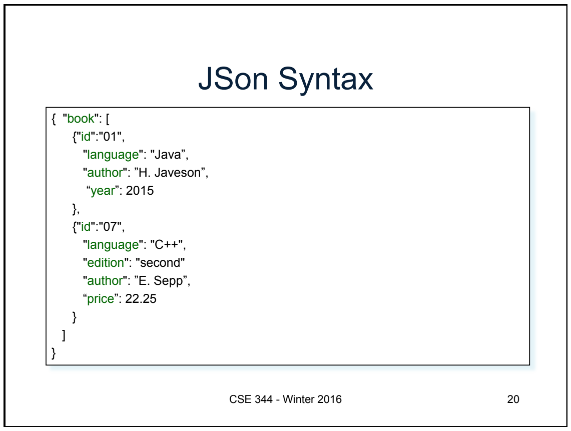
We will emphasize JSon as semi-structured data



JSon vs Relational

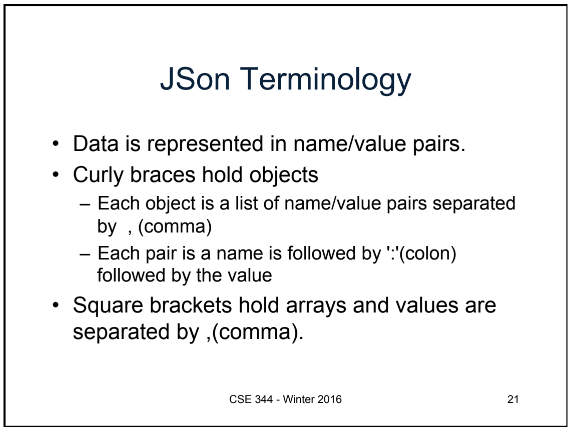
- Relational data model
  - Rigid flat structure (tables)
  - Schema must be fixed in advanced
  - Binary representation: good for performance, bad for exchange
  - Query language based on Relational Calculus

- Semistructured data model / JSon
  - Flexible, nested structure (trees)
  - Does not require predefined schema ("self describing”)
  - Text representation: good for exchange, bad for performance
  - Most common use: Language API; query languages emerging



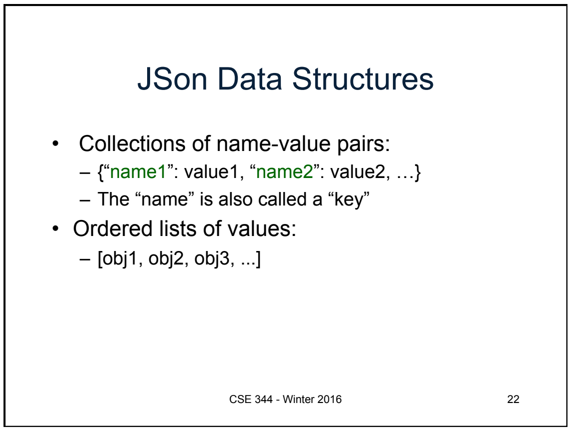
JSon Syntax

```
{  "book": [
     {"id":"01",
        "language": "Java”,
        "author": ”H. Javeson”,
         “year”: 2015
     },
     {"id":"07",
        "language": "C++",
        "edition": "second"
        "author": ”E. Sepp”,
        “price”: 22.25
     }
  ]
}
```



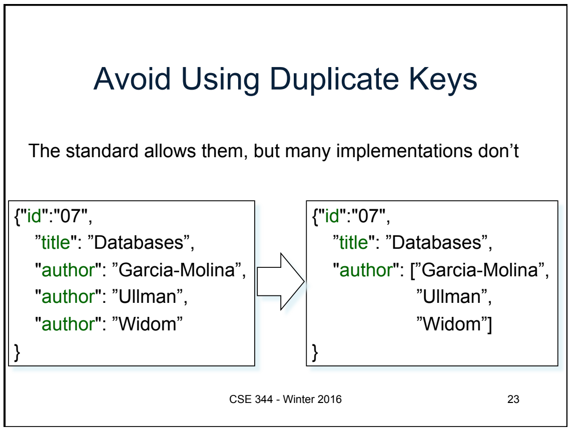
JSon Terminology

- Data is represented in name/value pairs.
- Curly braces hold objects
  - Each object is a list of name/value pairs separated by , (comma)
  - Each pair is a name is followed by ':'(colon) followed by the value
- Square brackets hold arrays and values are separated by ,(comma).



JSon Data Structures

- Collections of name-value pairs:
  - {“name1”: value1, “name2”: value2, …}
  - The “name” is also called a “key”
- Ordered lists of values:
  - [obj1, obj2, obj3, ...]

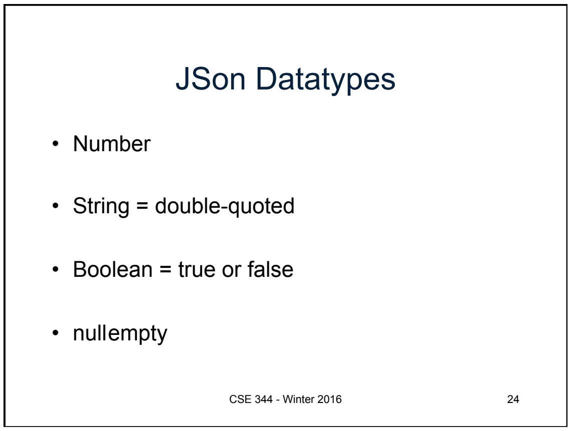


Avoid Using Duplicate Keys

The standard allows them, but many implementations don’t

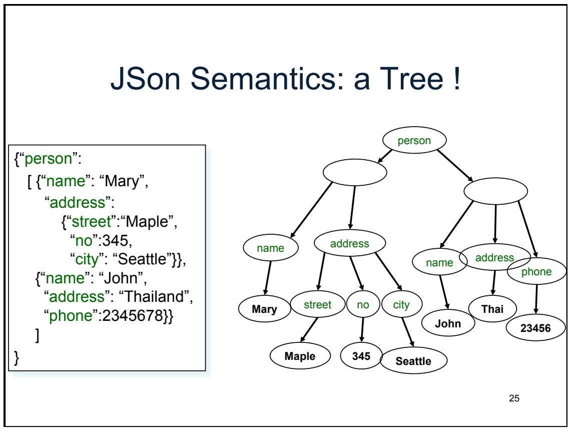
```
{"id":"07",
   ”title": ”Databases”,
   "author": ”Garcia-Molina”,
   "author": ”Ullman”,
   "author": ”Widom”
}
```

```
{"id":"07",
   ”title": ”Databases”,
   "author": [”Garcia-Molina”,
               ”Ullman”,
               ”Widom”]
}
```



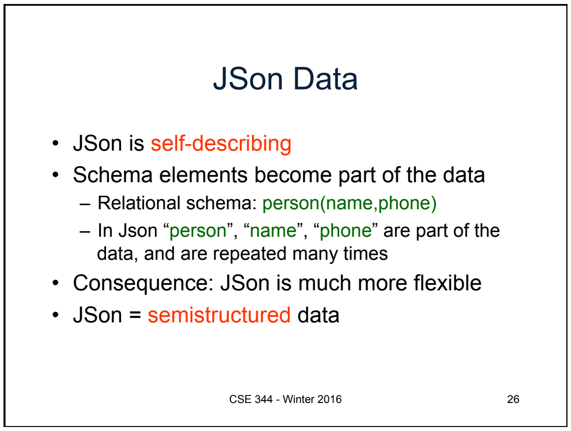
JSon Datatypes

- Number
- String = double-quoted
- Boolean = true or false
- nullempty



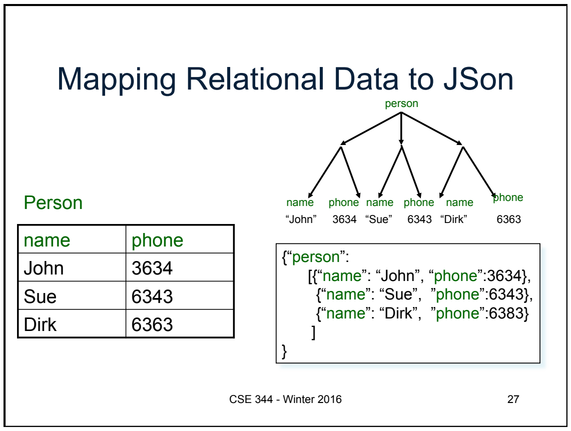
JSon Semantics: a Tree !

```
{"person":
  [ {"name": "Mary",
     "address":
        {"street":"Maple",
         "no":345,
         "city": "Seattle"}},
    {"name": "John",
     "address": "Thailand",
     "phone":2345678}}
  ]
}
```



JSon Data

- JSon is self-describing
- Schema elements become part of the data
  – Relational schema: person(name,phone)
  – In Json "person", "name", "phone" are part of the data, and are repeated many times
- Consequence: JSon is much more flexible
- JSon = semistructured data

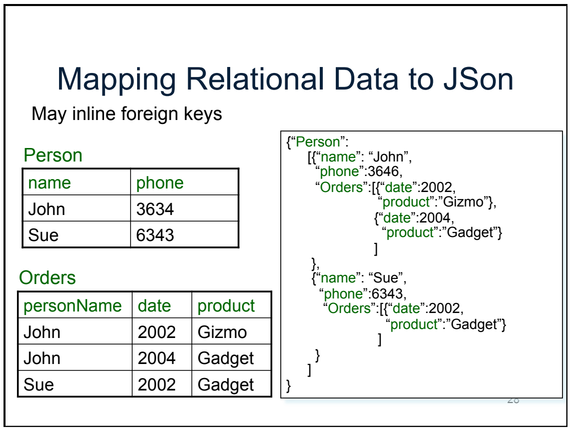


Mapping Relational Data to JSon

Person

name	phone
John	3634
Sue	6343
Dirk	6363

```
{"person":
    [{"name": "John", "phone":3634},
     {"name": "Sue",  "phone":6343},
     {"name": "Dirk",  "phone":6383}
    ]
}
```



Mapping Relational Data to JSon

May inline foreign keys

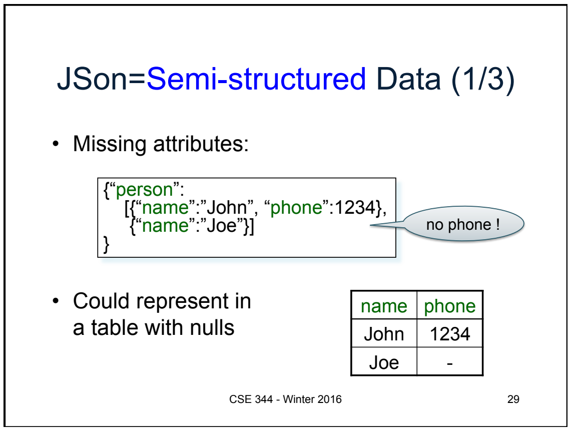
Person

name	phone
John	3634
Sue	6343

Orders

personName	date	product
John	2002	Gizmo
John	2004	Gadget
Sue	2002	Gadget

```
{"Person":
    [{"name": "John",
      "phone":3646,
      "Orders":[{"date":2002,
                 "product":"Gizmo"},
                {"date":2004,
                 "product":"Gadget"}
                ]
     },
     {"name": "Sue",
      "phone":6343,
       "Orders":[{"date":2002,
                  "product":"Gadget"}
                 ]
      }
    ]
}
```



JSon=Semi-structured Data (1/3)

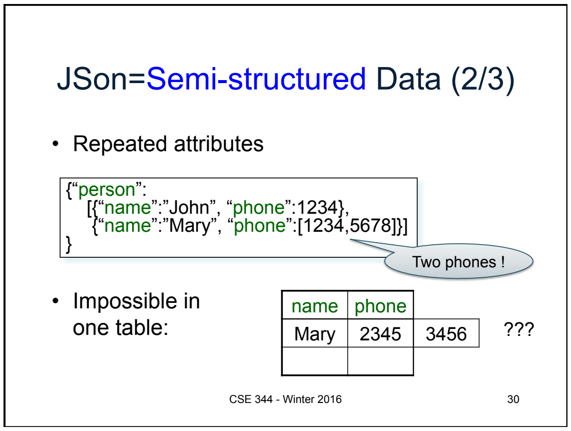
- Missing attributes:

```
{"person":
    [{"name":"John", "phone":1234},
     {"name":"Joe"}]
}
```

no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-



JSon=Semi-structured Data (2/3)

- Repeated attributes

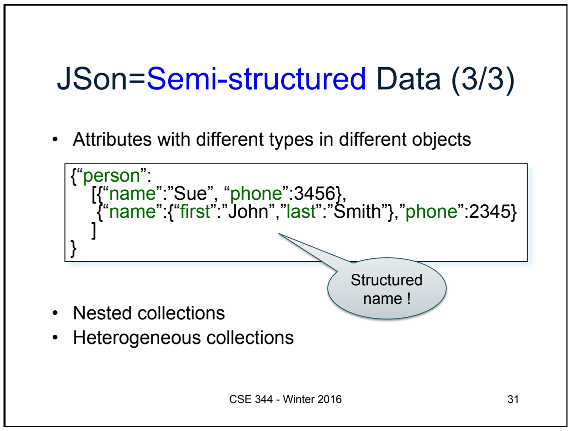
```
{"person":
    [{"name":"John", "phone":1234},
     {"name":"Mary", "phone":[1234,5678]}]
}
```

Two phones !

- Impossible in one table:

name	phone	
Mary	2345	3456

???



JSon=Semi-structured Data (3/3)

- Attributes with different types in different objects

```
{“person”:
   [{“name”:”Sue”, “phone”:3456},
    {“name”:{“first”:”John”,”last”:”Smith”},”phone”:2345}
   ]
}
```

Structured name !

- Nested collections
- Heterogeneous collections