Logistics

- Pop quiz (canvas, 5min)
- Lab 0 (this Wed/Friday)
- No office hours this week
- TA office hours:
  - Xiaofeng Zhou: Mondays 4-6pm
  - Miguel Rodriguez: Thursdays 3-5pm

Review

- How to do data science
  - Data Wrangling
- Big picture: data sources, ETL, data warehouse, data analytics/BI tools
- Data Types
  - Tabular – relational databases
  - Semi structured
  - Text, video, images
  - Graph data

Outline

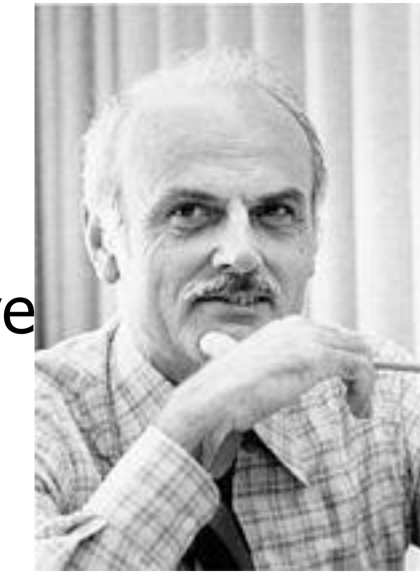
- Big Picture
- Data Types and Sources
- Data Models
- Data Preparation

Key Concept: Structured Data

A *data model* is a collection of concepts for describing data.

A *schema* is a description of a particular collection of data, using a given data model.

The Relational Model*



E. F., “Ted” Codd
Turing Award 1981

- The Relational Model is Ubiquitous:
  - MySQL, PostgreSQL, Oracle, DB2, SQLServe
  - Foundational work done at
    - IBM - System R
    - UC Berkeley - Ingres
- Object-oriented concepts have been merged in
    - Early work: POSTGRES research project at Berkeley
    - Informix, IBM DB2, Oracle 8i
- Also has support for XML (semi-structured data)
  – Adding support for full text search (unstructured data)

**Codd, E. F. (1970). "A relational model of data for large shared data banks". Communications of the ACM 13 (6): 37*

Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation:* made up of 2 parts:
  - *Schema* : specifies name of relation, plus name and type of each column
    - **Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)**
  - *Instance* : the actual data at a given time
    - #rows = *cardinality*
    - #fields = *degree / arity*
- Abstractly… A relation is a mathematical object (from set theory) which is true for certain arguments.
- An instance defines the set of arguments for which the relation is true.

Ex: Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, arity = 5 , all rows distinct

- The relation is true for these tuples and false for others (a.k.a, the closed world assumption)

SQL - A language for Relational DBs*

- SQL = Structured Query Language
- Data Definition Language (DDL)
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- Data Manipulation Language (DML)
  - Queries: Specify queries to find tuples that satisfy criteria (read) – relational operators?
  - Updates: add, modify, remove tuples (write) – operators?
- The DBMS is responsible for efficient evaluation.

* Developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the 1970s. Used to be *SEQUEL* (*Structured English QUEry Language*)

Example Database Implementation of Tabular Data Model

- SQLite
  - Table: fixed number of named columns of specified type
  - 5 storage classes for columns
    - NULL
    - INTEGER
    - REAL
    - TEXT
    - BLOB
  - Data stored on disk in a single file in row-major order
  - Operations performed via sqlite3 shell
- Other relational transaction databases?

OTHER "TABLE-LIKE" DATA MODELS

Pandas/Python

- **Series**: a named, ordered array/dictionary
  - Values can be any Numpy data type object
  - The keys of the dictionary are the **indexes**
  - Built on NumPy's **ndarray**

- **DataFrame**: a table with named columns
  - Represented as a map Dict (col_name -> series)
  - Each Series object represents a column

Operations

- map() functions
- filter (apply predicate to rows)
- sort/group by
- aggregate: sum, count, average, max, min
- Pivot or reshape
- Relational:
  - union, intersection, difference, cartesian product (CROSS JOIN)
  - select/filter, project
  - join: natural join (INNER JOIN), theta join, semi-join, etc.
  - Rename
- **More in Lab 0-part 2 and Lab 1**

Matrices vs Databases

- Tools like Pandas give up some of the important safety features of RDBMS (e.g. ACID), but can be much faster.

A

row	col	value
1	1	5.7
3	1	3.2
2	2	-5

B

row	col	value
2	1	12.0
3	3	5.1

SELECT A.row, B.col, SUM(A.value * B.value)
FROM A JOIN B
ON A.col = B.row
GROUP BY A.row, B.col

- What does this query do?

→ Matrix multiply in SQL (array/matrix extended data types)

- You probably never want to do this, but the ***opposite*** direction (relational aggregate query → matrix mult.) can be very useful.

What’s Wrong with Tables?

- Too limited in structure?
- Too rigid?
- Too costly up front?
- Examples… ??

RDBMS tables – row based

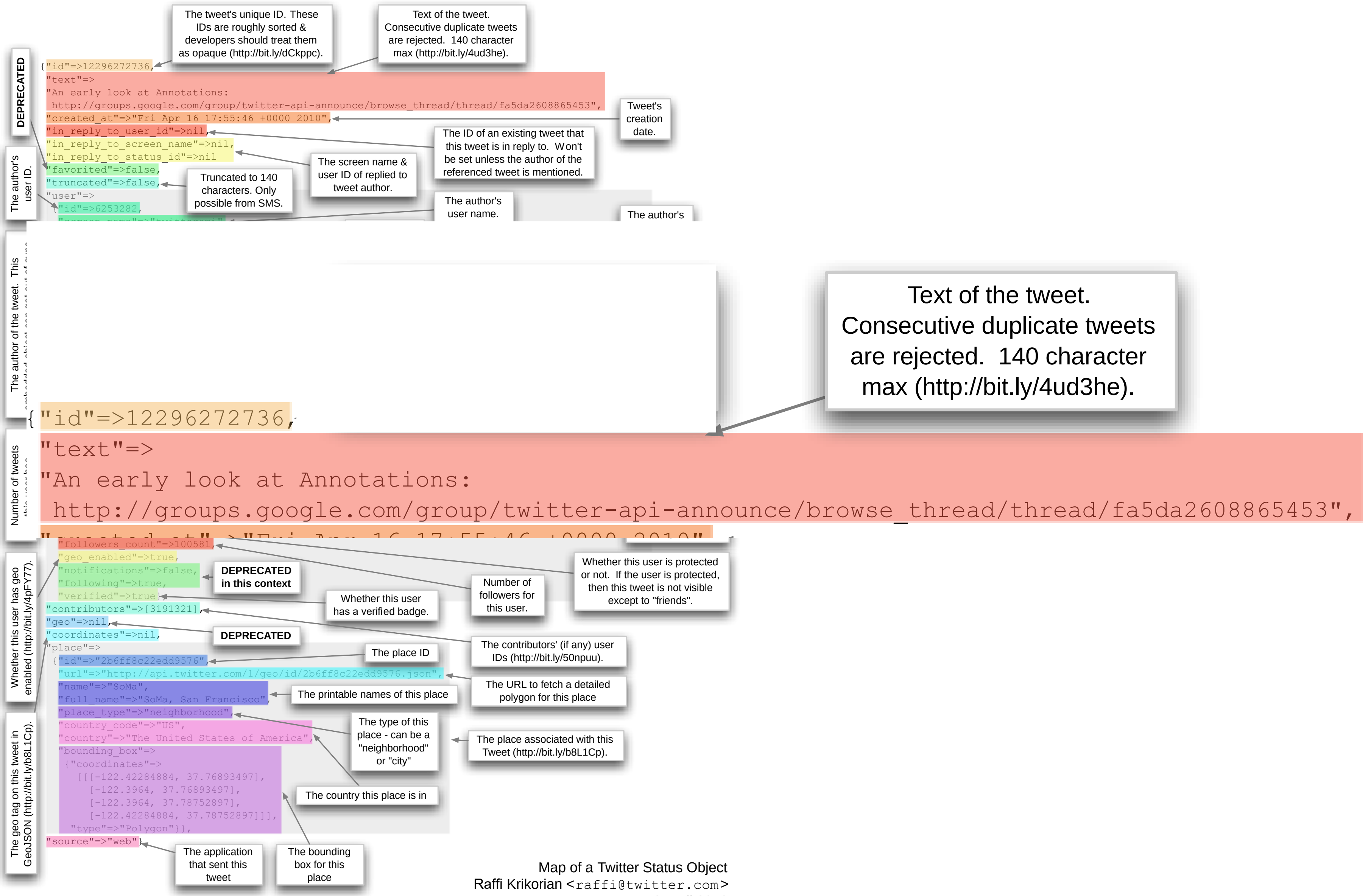
Table:

sid	name	login	age	gpa
53831	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

Represente

53831	Jones	jones@cs	18	3.4
53831	Smith	smith@ee	18	3.2

Tweet JSON Format



Map of a Twitter Status Object
Raffi Krikorian <raffi@twitter.com>
18 April 2010

Prerequisites for "Schemaless" DBs

- Need **external** and **internal** representations for all data types that will be used.
- **Internal:** a dynamically-typed, object-oriented language (like Java)
- **External:** an extensible data description language: JSON or XML

- **For Performance:** Fast SerDe (Serialization and DeSerialization) so internal data structures can be efficiently pushed or extracted from disk or network.

JSON format --> tree structure

```
{ "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
   "age": 25,
   "height_cm": 167.6,
   "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

Prerequisites for "Schemaless" DBs

- JSON includes named fields in a tree structure. Primitive types (e.g. string, number, boolean,…) are implicit.
- We can read JSON data (or XML) and automatically create internal representations for complex data.
- Using the field names and object structure, we can query these objects once loaded.

Other problems with (RDBMS) Tables?

- **Indices:** Typical RDBMS table storage is mostly indices
  - Can't afford this overhead for large datastores
- **Transactions:**
  - Safe state changes require journals etc., and are slow
- **Relations:**
  - **Checking** relations adds further overhead to updates
- **Sparse Data Support:**
  - RDBMS Tables are very wasteful when data is very sparse
  - Very sparse data is common in modern data stores, ex?
  - RDBMS tables might have dozens of columns, modern data stores might have many thousands.

RDBMS tables – row based

Table:

R

ID	name	login	loc	locid	LAT	LONG	ALT	State
52841	Jones	jones@cs	NULL	NULL	NULL	NULL	NULL	NULL
53831	Smith	smith@ee	NULL	NULL	NULL	NULL	NULL	NULL
55541	Brown	brown@ee	NULL	NULL	NULL	NULL	NULL	NULL
52841	Jones	jones@cs	NULL	NULL	NULL	NULL	NULL	NULL
53831	Smith	smith@ee	NULL	NULL	NULL	NULL	NULL	NULL
55541	Brown	brown@ee	NULL	NULL	NULL	NULL	NULL	NULL

Column-based store

Table:

ID	name	login	loc	locid	LAT	LONG	ALT	State
52841	Jones	jones@cs	Albany	2341	38.4	122.7	100	CA
53831	Smith	smith@ee	NULL	NULL	NULL	NULL	NULL	NULL
55541	Brown	brown@ee	NULL	NULL	NULL	NULL	NULL	NULL

ID	name
52841	Jones
53831	Smith
55541	Brown

ID	login
52841	jones@cs
53831	smith@ee
55541	brown@ee

ID	loc
52841	Albany

ID	locid
52841	2341

ID	LAT
52841	38.4

ID	LONG
52841	122.7

…

NoSQL Storage Systems

	Data Model
Cassandra	Columnfamily
CouchDB	Document
HBase	Columnfamily
MongoDB	Document
Neo4J	Graph
Redis	Collection
Riak	Document
Scalaris	Key/value
Tokyo Cabinet	Key/value
Voldemort	Key/value

Outline

- Big Picture
- Data Types and Sources
- Data Models
- Data Preparation

Data Preparation (I)

- ETL
  - We need to **extract** data from the **source(s)**
  - We need to **load** data into the **sink**
  - We need to **transform** data at the source, sink, or in a **staging area**

  - Sources: file, database, event log, web site, HDFS…
  - Sinks: Python, R, SQLite, RDBMS, NoSQL store, files, HDFS…

Data Preparation (II)

- Process model
  - The construction of a new data preparation process is done in many phases
    - Data **characterization**
    - Data **cleaning**
    - Data **integration**
  - We must efficiently move data around in space and time
    - Data **transfer**
    - Data **serialization** and **deserialization** (for files or network)

Data Preparation (III)

- Workflow
  - The transformation **pipeline** or **workflow** often consists of many steps
    - For example: Unix pipes and filters
    - $ cat data_science.txt | wc | mail -s "word count” myname@some.com
  - Hands-on experience in Lab0-part1
  - Parallel data preparation (e.g., Map-Reduce, Pig)
  - If the workflow is to be used more than once, it can be **scheduled**
    - Scheduling can be time-based or event-based
    - Use publish-subscribe to register interest (e.g. Twitter feeds)
  - Recording the execution of a workflow is known as capturing **lineage** or **provenance**

Schema-on-Read vs. Schema-on-Write

- Schema-on-Write: Traditional data systems require users to create a schema before loading any data into the system.

- Schema-on-Read: In Hadoop ecosystem, data can start flowing into the system in its original form, then the schema is parsed at read time (each user can apply their own "data-lens‟ to interpret the data).

Not “IF” But “WHEN”?

- “Schema on Write”
  - Traditional Approach
- “Schema on Read”
  - Data is simply copied to the file store, no transformation is needed.
  - A SerDe (Serializer/Deserlizer) is applied during read time to extract the required columns (late binding)
  - New data can start flowing anytime and will appear retroactively once the SerDe is updated to parse it.

	Pros	
• Read is Fast • Standards/Governance	⟷	• Load is Fast • Flexibility/Agility

Summary

- Data Models, Tables, Structure, etc.
  - SQL
  - NoSQL
  - Schema on Read vs. Schema on Write

- Unix data preparation and Pandas data manipulation lab 0 this Wednesday & Friday