Databases, cont'd.
Pandas
Large Survey Database

Before we start

- HW #1: I'll e-mail the assignment tomorrow
- Is everyone on the mailing list?
- Survey talk assignments
- Wednesday vs. Data Science talk on visualization

SELECT Statement

- SELECT ra, dec, psfMag_r FROM photo
- SELECT ra, dec, psfMag_r FROM photo WHERE psfMag_r < 21.5
- SELECT ra, dec, psfMag_r FROM photo WHERE psfMag_r < 21.5 LIMIT 5
- SELECT ra, dec, psfMag_r FROM photo WHERE psfMag_r < 21.5 LIMIT 5
- SELECT COUNT(psfMag_r), AVG(psfMag_r) FROM photo WHERE psfMag_r < 21.5
- SELECT COUNT(*), run FROM photo GROUP BY run
- SELECT COUNT(*), run FROM photo GROUP BY run ORDER BY run
- SELECT COUNT(*) as ct , run FROM photo GROUP BY run ORDER BY ct

JOIN: Joining tables

- Example:
  - Each row in the 'photo' table has a `run' entry – the ID of the SDSS run where this object was observed
  - Each entry in the 'runs' table has a 'mjdstart' entry, indicating the time when the observing for this run started
  - How can we find the mjdstart for each object? An algorithm for doing it by hand:
    - For each row in the photo table:
      - Read off the value of 'run'
      - Find the corresponding row in the 'runs' table
      - Read off the value of mjdstart

JOIN: Joining tables

- SELECT

  photo.ra, photo.dec, photo.run, mjdstart

  FROM

  photo

  JOIN runs ON photo.run = runs.run

The columns we're interested in.

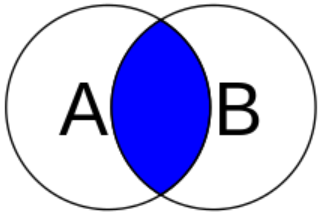
Those appearing in more than one table need to be prefixed by the table name.

The table we're querying

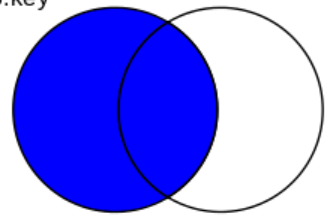
Instructions how to join the runs table onto the photo table.

SQL JOINS

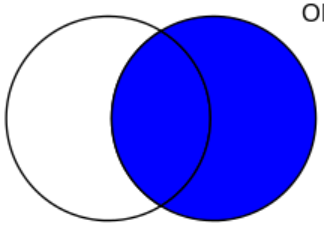
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key



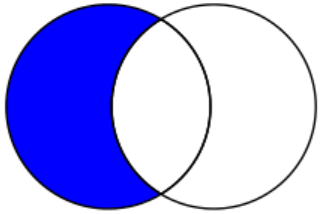
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key



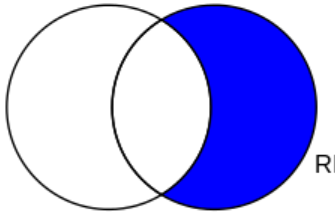
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key



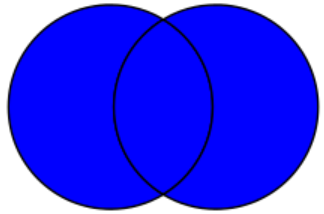
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL



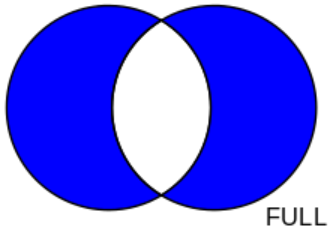
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL



More information

- http://en.wikipedia.org/wiki/Join_%28SQL%29
- http://blog.codinghorror.com/a-visual-explanation-of-sql-joins/

Pandas

What is Pandas?

- pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for Python

When to use Pandas?

- When you need to analyze/manipulate heterogeneous, indexed, potentially sparse, tabular data
  - I.e., tables, time series
  - This is in contrast to arrays, where numpy usually suffices

- When the dataset is small enough to fit in memory

What do you get with Pandas?

- A set of labeled array data structures (Series/TimeSeries, DataFrame, Panel)
- Index objects enabling both simple axis indexing and multi-level / hierarchical axis indexing
- An integrated group by engine for aggregating and transforming data sets
- Date range generation (date_range) and custom date offsets enabling the implementation of customized frequencies
- Input/Output tools: loading tabular data from flat files (CSV, delimited, Excel 2003), and saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
- Memory-efficient "sparse" versions of the standard data structures for storing data that is mostly missing or mostly constant (some fixed value)
- Moving window statistics (rolling mean, rolling standard deviation, etc.)
- Static and moving window linear and panel regression

http://pandas.pydata.org/pandas-docs/stable/overview.html

Key Data Structures

- Series
  - Labeled, 1-dimensional, data of homogenous type
- TimeSeries
  - Time-indexed, 1-dimensional, data of homogenous type
- DataFrame
  - Labeled, 2-dimensional data, where columns can be of different types
- Panel
  - Labeled, 3-dimensional data
  - Less frequently used

NULL in Pandas: NaN

- Pandas uses NaN to represent NULLs

Live Demo!

http://www.gregreda.com/2013/10/26/intro-to-pandas-data-structures/

Finding out more

- Pandas
  - 10 minute tutorial: http://pandas.pydata.org/pandas-docs/stable/10min.html
  - 10 minute tutorial video: http://vimeo.com/59324550
  - Pandas Tutorials: http://pandas.pydata.org/pandas-docs/stable/tutorials.html
  - Intro to Pandas data structures: http://www.gregreda.com/2013/10/26/intro-to-pandas-data-structures/

Large Survey Database

The Trouble With Web Archives

- What do most astronomical web archives do for users?
  - Data storage
  - Cone searches
    - Give me all objects at position (α, δ) within radius r
  - Execution of complex queries against the database (e.g., you can in principle compute correlation functions from within an SQL query), a few at a time

- However, they are quite poor at supporting complex, whole-dataset, operations!

But that is where the science is!

- With large data sets, the name of the game is complex data mining – we need to go through the entire dataset (repeatedly), to discover interesting things.

- Corollary: we will either
  a) Want to push the computing to the data (LSST)
  b) Pull the data to your computing at home (now (and LSST!))

Examples

- Galactic structure: density/proper motion maps of the Galaxy
  - => forall stars, compute distance, bin, create 5D map
- Galactic structure: dust distribution
  - => forall stars, compute g-r color, bin, find blue tip edge, infer dust distribution
- Near-field cosmology: MW satellite searches
  - => forall stars, compute colors, convolve with spatial filters, report any satellite-like peaks
- Variability: Bayesian classification of transients and discovery of variables
  - => forall stars, get light curves, compute likelihoods, alert if interesting
- …

Large Survey Database

- A distributed Python framework for storing, querying, and distributing computation on large datasets
- http://lsddb.org
- Data access: SQL-like queries (Python)

- Current release:
  - SQL-like query language
  - Local caching of data
  - On-the-fly cross-matching of catalogs
  - ACID (Atomicity, Consistency, Isolation, Durability) transactions
  - MapReduce engine and Python API
  - Multi-core aware query engine
  - Distributed query engine (experimental)

Limitations

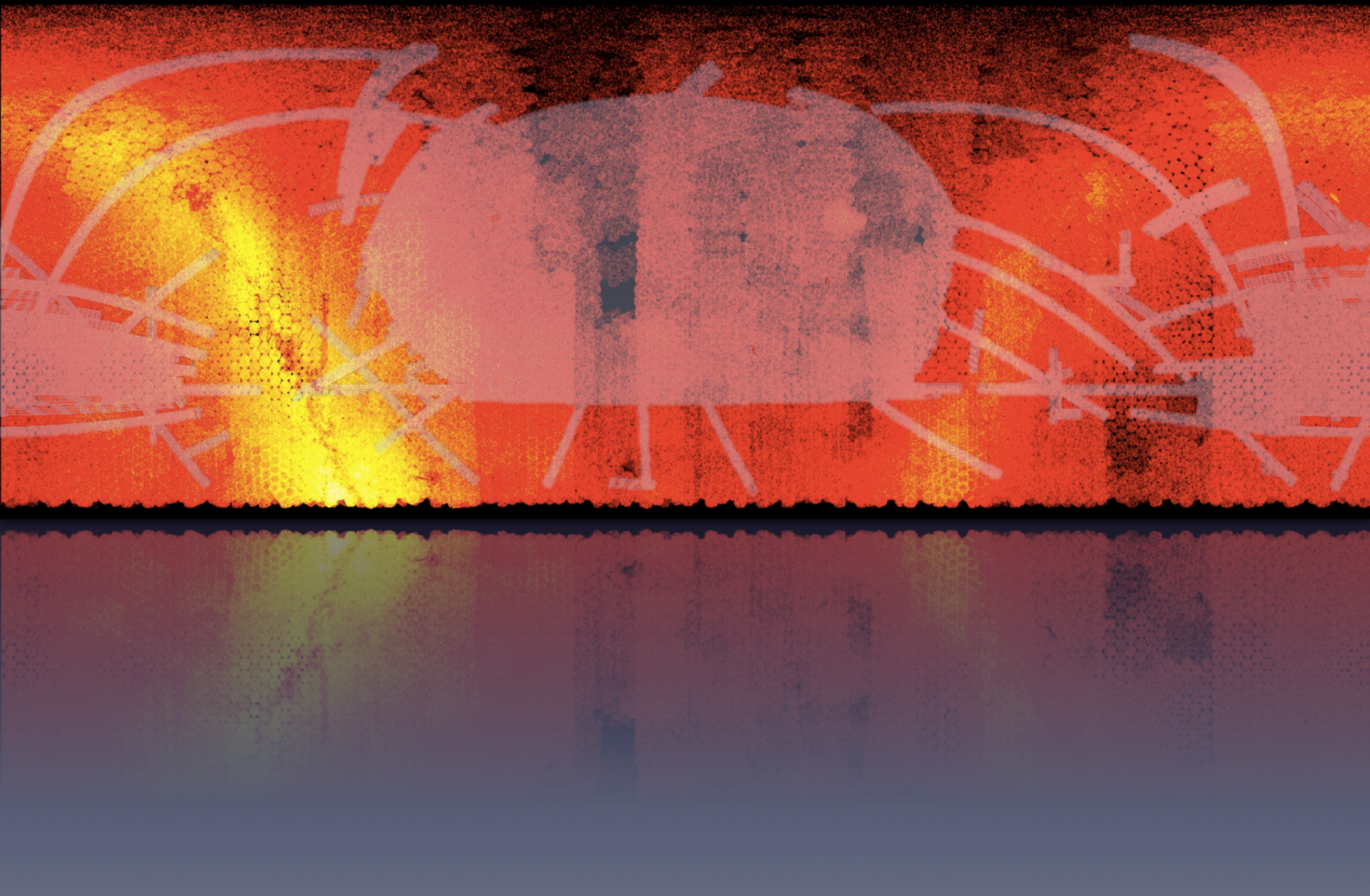
- A made-up, non-standard, language
- Limited ability to do joins
- May have strange, buggy, corner cases
- Only a few active developers

… but, it's unfortunately one of the rare things that address our needs!

What and Where

- Catalogs:
  - Pan-STARRS PS1 (~14B rows)
  - SDSS (~300M rows)
  - GALEX (~40M rows)
  - LSST mocks (13B rows)
  - PTF
  - …

14 billion detections of PS1



LSD @ ASTR 597b

- We will primarily be using LSD to extract subsets from large data sets (mostly SDSS, WISE)
- We will use it to do catalog cross-matching
- We will be importing catalogs to Survey Science Group's LSD server
  - This will be your final project: writing the table schemas and code to load a survey into LSD
  - Note: still procuring the machine, should be here by the end of the quarter.
  - Result: one of the more capable survey archives!

Quick LSD Tutorial

1. Installing LSD
2. Where to find help
3. Setting up the LSD environment
4. A simple query
5. LSD architecture
   - Overall layout (tables are directories)
   - In-directory Layout
   - Query Execution
6. Query syntax
7. Joins
8. Cross-match joins
9. Defining new tables and importing new catalogs
10. Using LSD from Python
11. MapReduce with LSD
12. The Future

Documentation:

http://lsddb.org

Installing

- wget -N -nv http://research.majuric.org/lsd-setup/scripts/go.sh

- bash go.sh

Examples

export LSD_DB=/ssd/mjuric/lsd/db

lsd-admin remote follow table http://faun.rc.fas.harvard.edu/mjuric/db/public sdss

lsd-query --format=fits --bounds='beam(200, 40, 1)' 'select ra, dec from sdss'

lsd-query --format=fits --bounds='beam(200, 40, 1)' 'select ra, dec from sdss where g - r > 0.5'

lsd-query --format=fits --bounds='beam(200, 40, 1)' 'select ra, dec from sdss where (g - r > 0.5) & (g - r < 0.6)'

lsd-admin desc table sdss

lsd-admin remote list http://faun.rc.fas.harvard.edu/mjuric/db/public

lsd-admin remote follow table http://faun.rc.fas.harvard.edu/mjuric/db/public usnob

lsd-query --format=fits --bounds='beam(200, 40, 1)' 'select ra, dec from usnob'

lsd-query --bounds='beam(200, 40, .1)' 'select ra, dec, usnob.ra, usnob.dec, usnob._DIST*3600 from sdss, usnob(matchedto=sdss) '

lsd-query --bounds='beam(200, 40, .1)' 'select ra, dec, usnob.ra, usnob.dec, usnob._DIST*3600 from sdss, usnob(matchedto=sdss, outer) '

lsd-query --bounds='beam(200, 40, .1)' 'select ra, dec, usnob.ra, usnob.dec, usnob._DIST*3600, usnob._ISNULL from sdss, usnob(matchedto=sdss, outer) '

Using LSD from Python

```
import os
import lsd
db = lsd.DB(os.environ["LSD_DB"])

import lsd.bounds
b = [ (lsd.bounds.rectangle(180, 30, 190, 40, coordsys="gal"), None) ]
rows = db.query('select ra, dec, r FROM sdss').fetch(bounds=b)

rows.as_ndarray()
data = pd.DataFrame(rows.as_ndarray())
```

Importing new catalogs

- Write a schema in YAML
- Create a table using 'lsd-admin create table'
- Use the lsd-import utility to populate the table

```
    I    ./src/schemas/usnob.yaml
# Schema for USNO-B table
filters: {complevel: 5, complib: blosc}
schema:
  common:
    primary_key: usnob_id
    spatial_keys: [ra, dec]
    columns:
    - [usnob_id, u8]
    - [ra, f8]
    - [dec, f8]
    - [sra, i2]
    - [sde, i2]
    - [epoch, f4]
    - [muRa, i2]
    - [muDec, i2]
    - [muprob, u1]
    - [muflag, u1]
    - [smura, i2]
    - [smude, i2]
    - [sfitra, i2]
    - [sfitde, i2]
    - [nfitpt, u1]
    - [mag, 5f4]
    - [fldid, 5i2]
    - [sg, 5u1]
    - [fldepoch, 5f4]
```

Creation

[mjuric@lsst-dev [anaconda-updates] ~/lsd/opt/lsd/lsd]$ lsd-admin create table --schema ./src/schemas/galex.yaml galex

-------- committing 20150126223147.284667 [galex] ---------

[galex] Updating tablet catalog: [256 el.]::::::::::::::::::> 0.18 sec

[galex] Updating neighbors: Already up to date.

[galex] Updating tablet catalog: [256 el.]::::::::::::::::::> 0.19 sec

[galex] Updating stats: [0 el.]> 0.00 sec

[galex] Marking tablets read-only...

---------- success 20150126223147.284667 [galex] ---------

Table 'galex' created.

[mjuric@lsst-dev [anaconda-updates] ~/lsd/opt/lsd/lsd]$

Import

lsd-import text

--force -d , galex_gr5

--cols-file=schemas/galex.map

.../GALEX/AIS/.csv.gz