
Wes McKinney (pandas)
Francesc Alted (Señor pytables)
Fernando Perez (ipython)
Travis Oliphant (numpy, Continuum)
Paul Ivanov (matplotlib)
IZZE

Databases with Python

AY250 Fall 2013

authors: J. Bloom

Overview of Today's Lecture

- What are DBs? Why DBs?
- Generic DB concepts:
  relational table architecture SQL
- sqllite3 & Python
- MySQL and Postgresql
- HDF5 and NetCDF4

What are DBs?

Organized collection of data, managed by a Database Management System (DBMS)

DBMSs (generally) offer:

- storage
- creation & manipulation of data
- search
- (high-level data operations)

Whaa? Doesn't Python already do this for me?

creation
manipulation
search
storage

```
>>> db = range(1,1000,2)
>>> db[10] = "spamalot"
>>> db.index(31)
15
>>> import cPickle ; cPickle.dump(db,open("my.py.db","w"))
```

Limitations:

- Searches on lists are slow, O(n)
- db can only be as big as the available RAM
- efficiently storage files (i.e., pickles)
  are python-only (portability issue)
- single-instance use only (no concurrent use by others)

dictionaries solve some of simplest the searching issues (using a hash value for the key), but any realistic search is still painfully slow on big data:

```
db = {"scenes": \
        [{"cheeseshop": {"actors": ["Cleese", "Palin"]}},  \
         {"spam":       {"actors": ["Cleese","Idle"]}}],  \
      "actors": \
         {"Palin":  {"age": 54, "how funny": 6}, \
          "Cleese": {"age": 62, "how funny": 9},  \
          "Idle":   {"age": 61, "how funny": None}}}
```

what's the average age of the actors in the cheeseshop scene?

(concept of a JOIN operation...see more later)

Python "Solution" is to 3rd-Party It

provides built-in access (as of 2.5) to sqlite3
& APIs for all other major DBMS

General Purpose Database Systems

- IBM DB2
- Firebird (and Interbase)
- Informix
- Ingres
- MySQL
- Oracle
- PostgreSQL
- SAP DB (also known as "MaxDB")
- Microsoft SQL Server
- Sybase

Record-based Databases

Databases working on flat files or fixed records.

- MetaKit
- ZODB
- BerkeleyDB
- KirbyBase
- Durus
- atop
- buzhug

XML Databases

- 4Suite server
- Oracle/Sleepycat DB XML

somewhat outdated: http://wiki.python.org/moin/DatabaseInterfaces

Understanding DBMS Frameworks

Three levels

1. External
   what "users" of the DB sees (i.e., the Python API)
   usually different for different users

2. Conceptual
   logical structure of the DB, relationships between the data, security details, etc. "DB Scheme"

3. Physical
   how the data are stored, managed at a low level (from compiled-language code to bytes)

Implementation

How the conceptual level views data, the "data model":

Relational

Think of data organized as 2D tables, with connections between them

[most]

Object-Oriented

Think of data as attributes in model classes, with methods & inheritance

db4o
~BigTable

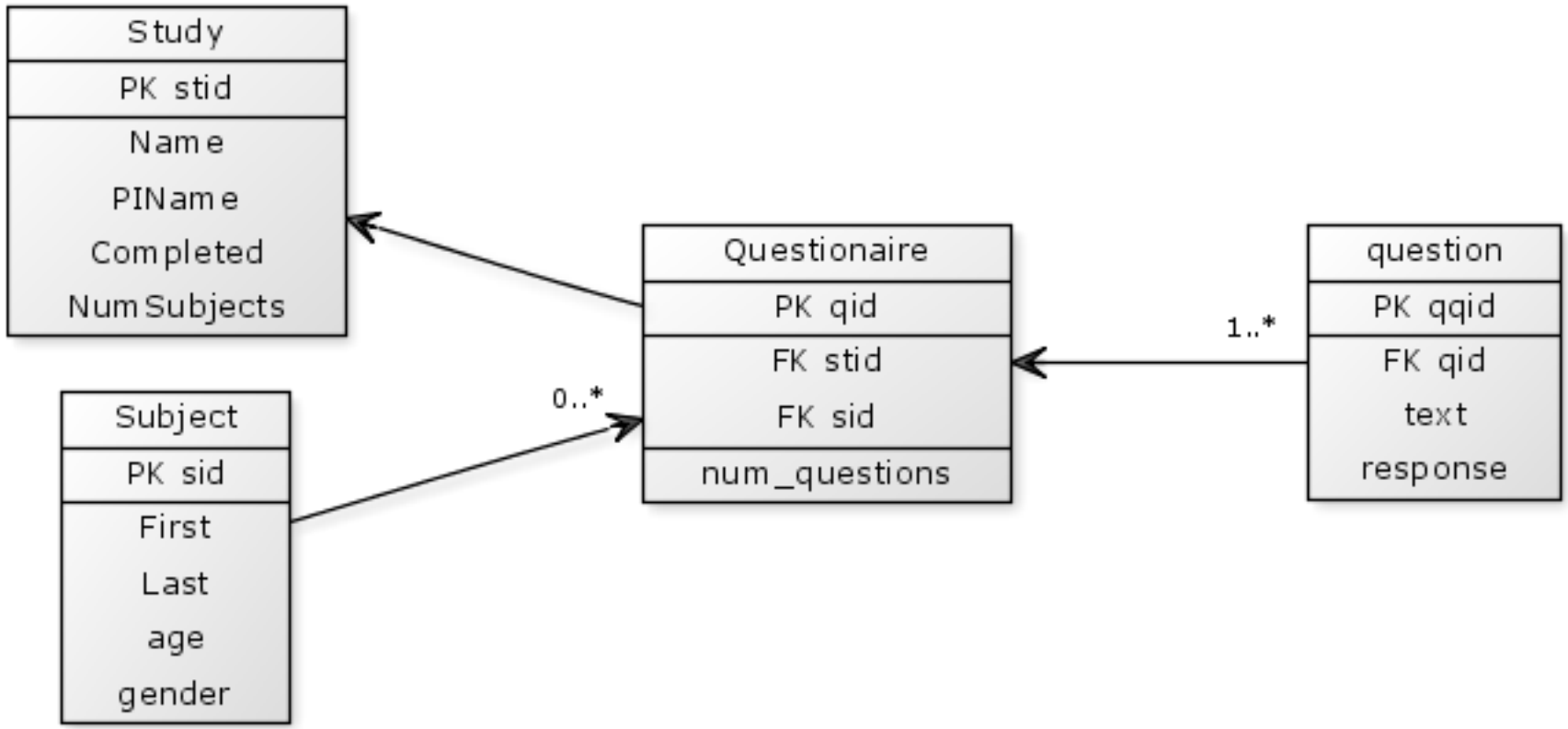
http://www.odbms.org/downloads.aspx

Hierarchical

Tree- or document-like. XML/JSON DBs and "NoSQL" DBs

eXist, RethinkDB
DynamoDB,
mongoDB
couchDB, ...

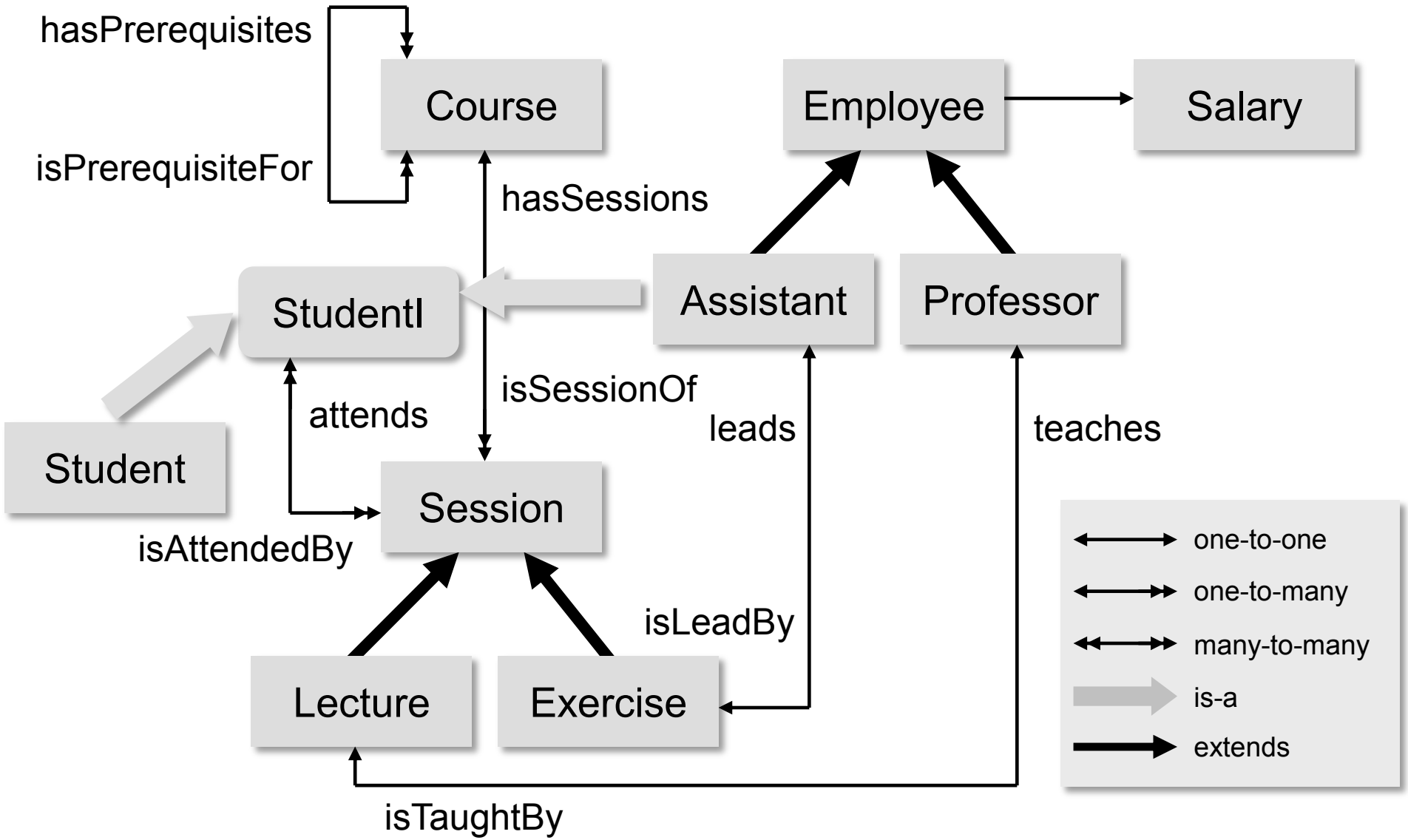
Relational



PK: Primary Key (Hashed)

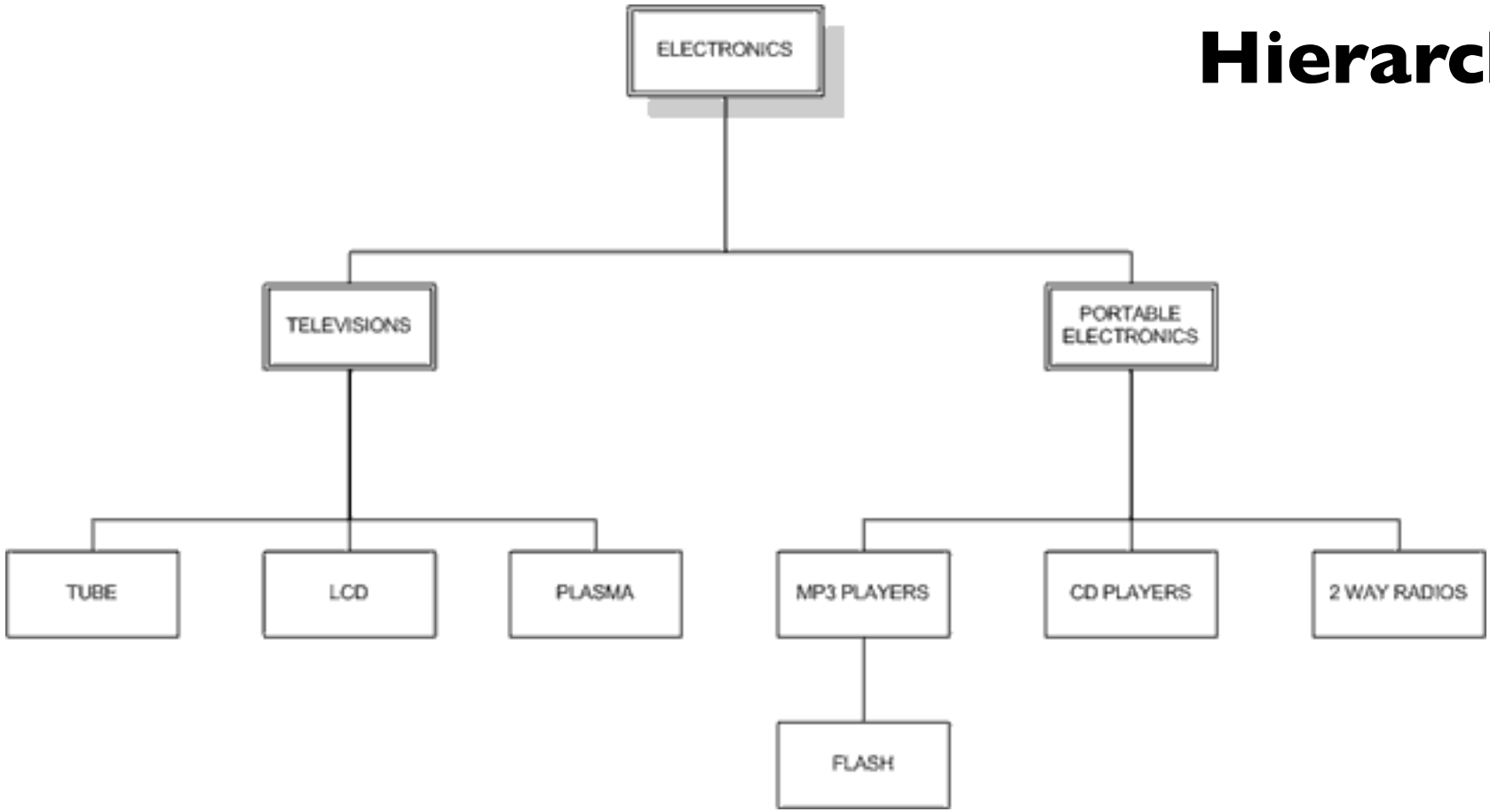
FK: Foreign Key

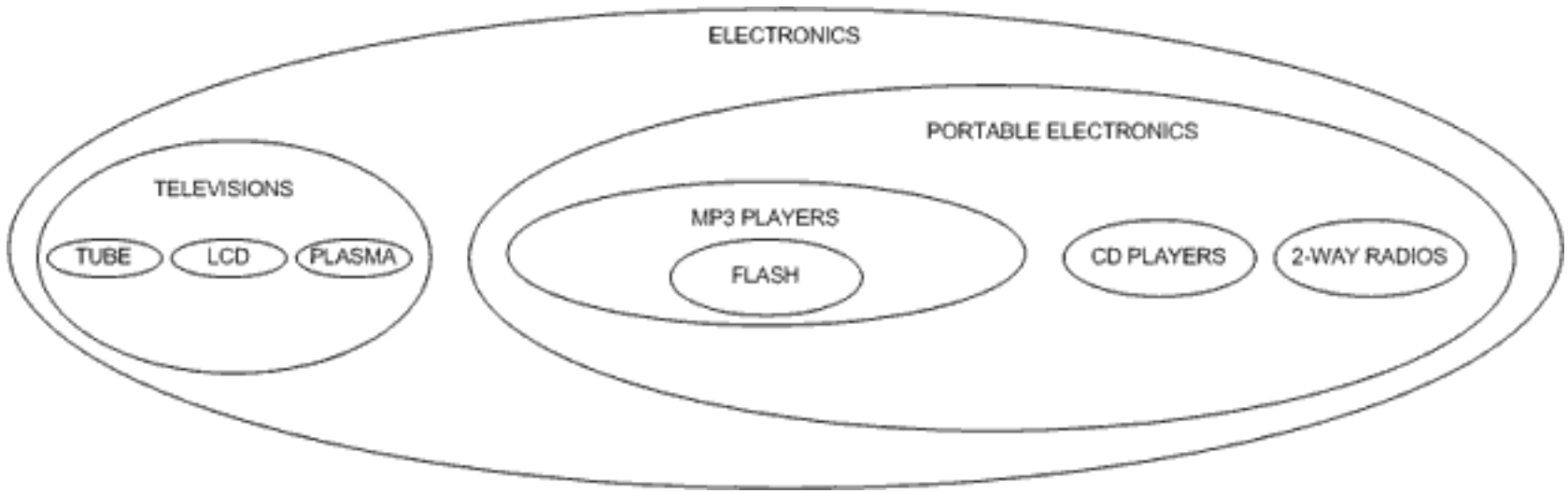
Object-Oriented



http://www.odbms.org/download/lecture_12_objectDatabases.pdf

Hierarchical





Implementation

How the DBMS operates:

client-server - persistent server managing and accepting connections from multiple clients. (could have a distributed implementation)

MySQL, Postgresql, ...

embedded - no server. Clients connect to the DB directly.

sqlite3

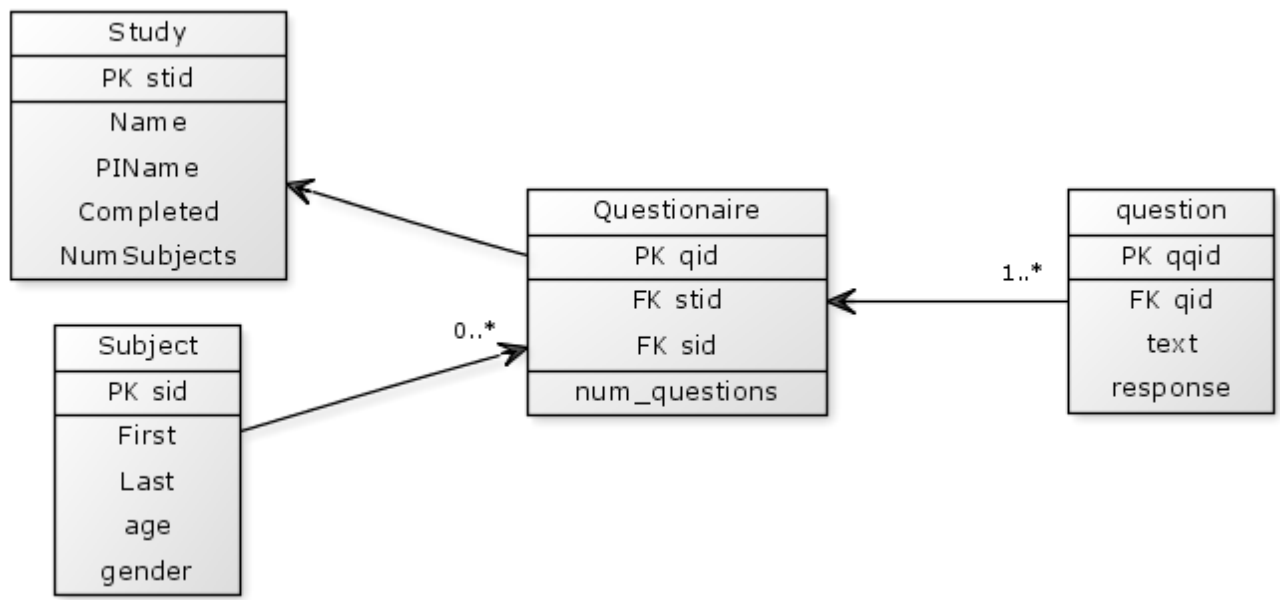
All modern DBMSs are meant to be
fast, safe, very scalable:
large, complex queries can be efficient

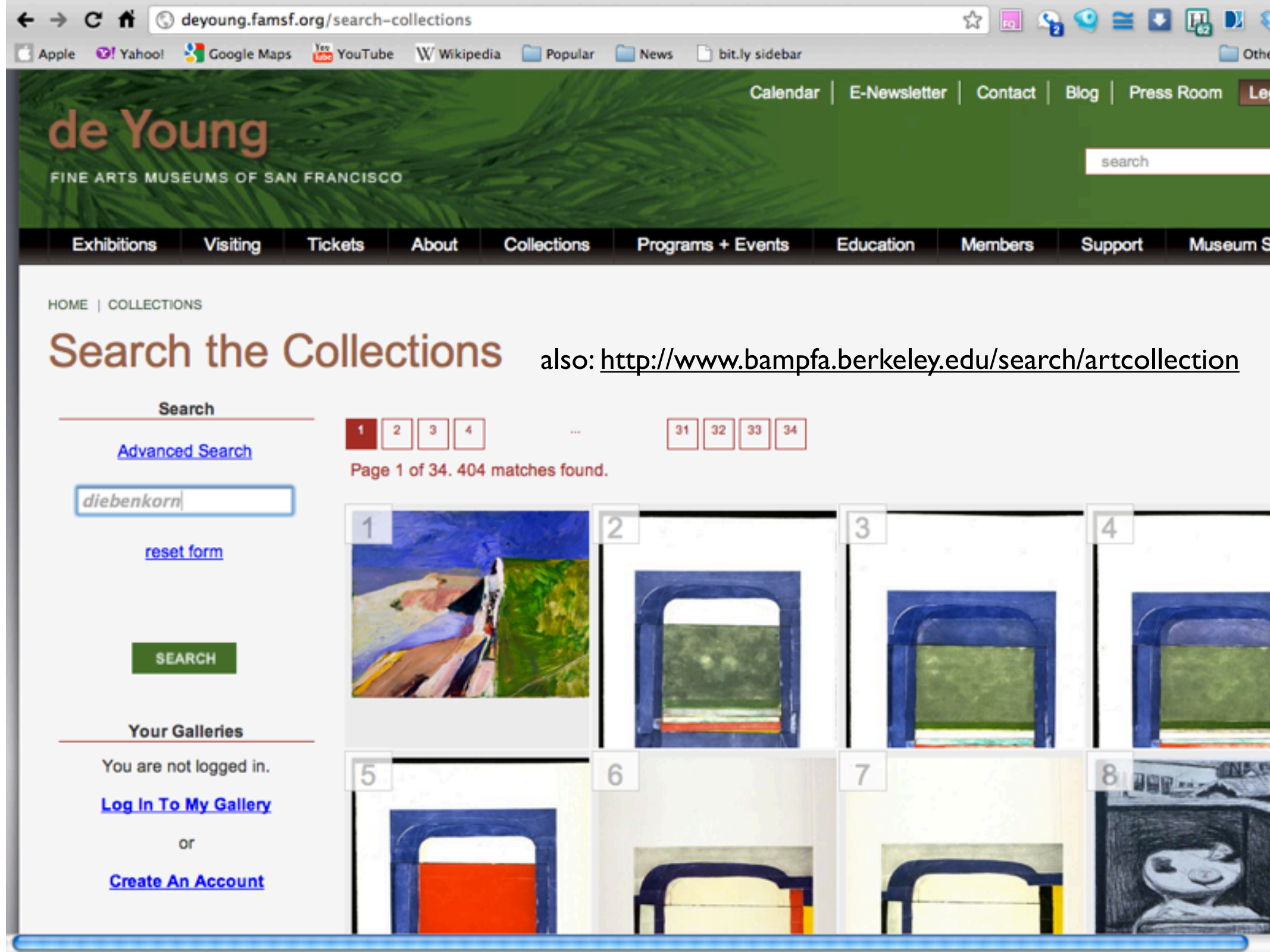
Data Model Concepts

database - contains tables, actions ("triggers"), permissions/security

table - data with fixed number of columns and rows of data that you add to (and query)

keys - hashed columns (or combinations of columns) used for efficiently joining of tables. "Primary" & "Foreign"




deyoung.famsf.org/search-collections
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
Popular
News
bit.ly sidebar
Calendar
E-Newsletter
Contact
Blog
Press Room
de Young
FINE ARTS MUSEUMS OF SAN FRANCISCO
search
Exhibitions
Visiting
Tickets
About
Collections
Programs + Events
Education
Members
Support
HOME | COLLECTIONS
Search the Collections
also: http://www.bampfa.berkeley.edu/search/artcollection
Search
Advanced Search
diebenkorn
reset form
SEARCH
Your Galleries
You are not logged in.
Log In To My Gallery
or
Create An Account
1 2 3 4 ... 31 32 33 34
Page 1 of 34. 404 matches found.
1
2
3
4
5
6
7
8

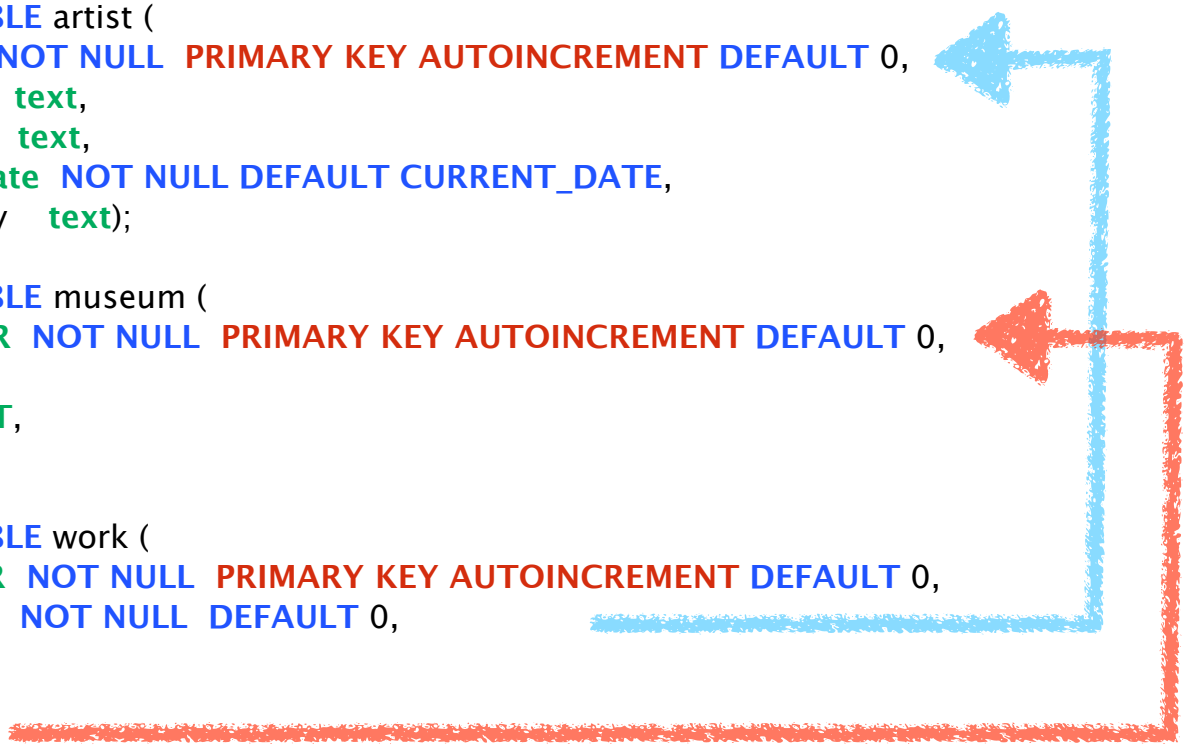
Simple (Command-Line) DB Creation

```
PySeminar> sqlite3 art.db
SQLite version 3.7.7.1 2011-06-28 17:39:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .read art-create.sql
sqlite> .schema
```

```
CREATE TABLE artist (
aid integer  NOT NULL  PRIMARY KEY AUTOINCREMENT DEFAULT 0,
first_name     text,
last_name      text,
birth_date date  NOT NULL DEFAULT CURRENT_DATE,
birth_country    text);

CREATE TABLE museum (
mid INTEGER  NOT NULL  PRIMARY KEY AUTOINCREMENT DEFAULT 0,
name TEXT,
country TEXT,
city TEXT);

CREATE TABLE work (
wid INTEGER  NOT NULL  PRIMARY KEY AUTOINCREMENT DEFAULT 0,
aid INTEGER  NOT NULL  DEFAULT 0,
title TEXT,
type TEXT,
mid integer,
finish_date Date);
```



"Data Definition Language (DDL)"

Loading in some data

```
PySeminar> sqlite3 art.db
SQLite version 3.7.7.1 2011-06-28 17:39:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .read art-load.sql
sqlite> .header on
sqlite> .mode column
sqlite> select * from work;
wid         aid         title       type        mid         finish_date
----------  ----------  ----------  ----------  ----------  -----------
1           3           Seawall     Painting    2           1957-06-01
2           3           Spreading   Print       1           1981-02-30
3           1           Bedroom at  Painting    2           1889-09-03
4           2           Campbell's  screenprin  2           1968-09-08
5           2           Liz         lithograph  1           1966-02-22
```

"Data Manipulation Language (DML)":

```
select * from tablename;
```

Basic Queries in SQL

```
select * from work;
```

gets all rows from table work and prints out all columns

```
select title,type from work;
```

gets all rows from table work and prints selected columns

```
select title,type as "work type" from work;
```

gets all rows from table work and prints selected columns, renaming the second one (view only)

```
select title,DATE('NOW') - finish_date as age
from work where
finish_date < DATE("1970-01-01")
order by age desc;
```

get only those rows where the work was finished before 1970, and show the name and age in years, descending order by age

Basic Queries in SQL

```
sqlite> select * from work where type = "painting";
sqlite>
sqlite> select * from work where type like "painting";
wid         aid         title       type        mid         finish_date
----------  ----------  ----------  ----------  ----------  -----------
1           3           Seawall     Painting    2           1957-06-01
3           1           Bedroom at  Painting    2           1889-09-03
sqlite> select * from work where type glob "P*";
wid         aid         title       type        mid         finish_date
----------  ----------  ----------  ----------  ----------  -----------
1           3           Seawall     Painting    2           1957-06-01
2           3           Spreading   Print       1           1981-02-30
3           1           Bedroom at  Painting    2           1889-09-03
sqlite> select * from work where type glob "*[p|P]rint*";
wid         aid         title              type        mid         finish_date
----------  ----------  ---------------  ----------  ----------  -----------
2           3           Spreading Spade  Print       1           1981-02-30
4           2           Campbell's Toma  screenprin  2           1968-09-08
```

Small exercise:

What work was created less than 35 years ago and starts with the letter "S"?

What if I want to view all works in Berkeley by American artists?

This information is distributed across three different tables

artist

aid	first_name	last_name	birth_date	birth_country	death_date
1	Vincent	Van Gogh	1853-03-30	Netherlands	1890-07-29
2	Andy	Warhol	1928-08-06	USA	1987-02-22
3	Richard	Diebenkorn	1922-04-22	USA	1993-03-30

work

wid	aid	title	type	mid	finish_date
1	3	Seawall	Painting	2	1957-06-01
2	3	Spreading	Print	1	1981-02-30
3	1	Bedroom at	Painting	2	1889-09-03
4	2	Campbell's	screenprin	2	1968-09-08
5	2	Liz	lithograph	1	1966-02-22

museum

mid	name	country	city
1	Berkeley Art	USA	Berkeley
2	de Young	USA	San Franci

SQL: JOIN (after SELECT, before WHERE)

What if I want to view all works in Berkeley by American artists?

This information is distributed across three different tables

```
sqlite> select museum.name as museum,artist.last_name as
artist,work.title,work.type,work.finish_date as date from work
 JOIN artist on artist.aid = work.aid
 JOIN museum on museum.mid = work.mid
 where artist.birth_country = 'USA'
   and museum.city = 'Berkeley';

museum        artist      title             type        date
------------  ----------  ----------------  ----------  ----------
Berkeley Art  Diebenkorn  Spreading Spade   Print       1981-02-30
Berkeley Art  Warhol      Liz               lithograph  1966-02-22
```

http://zetcode.com/databases/sqlitetutorial/

More Advanced Concepts

Views

"virtual tables" that are the result of a (complex)query on the tables. Like "Smart Folders"

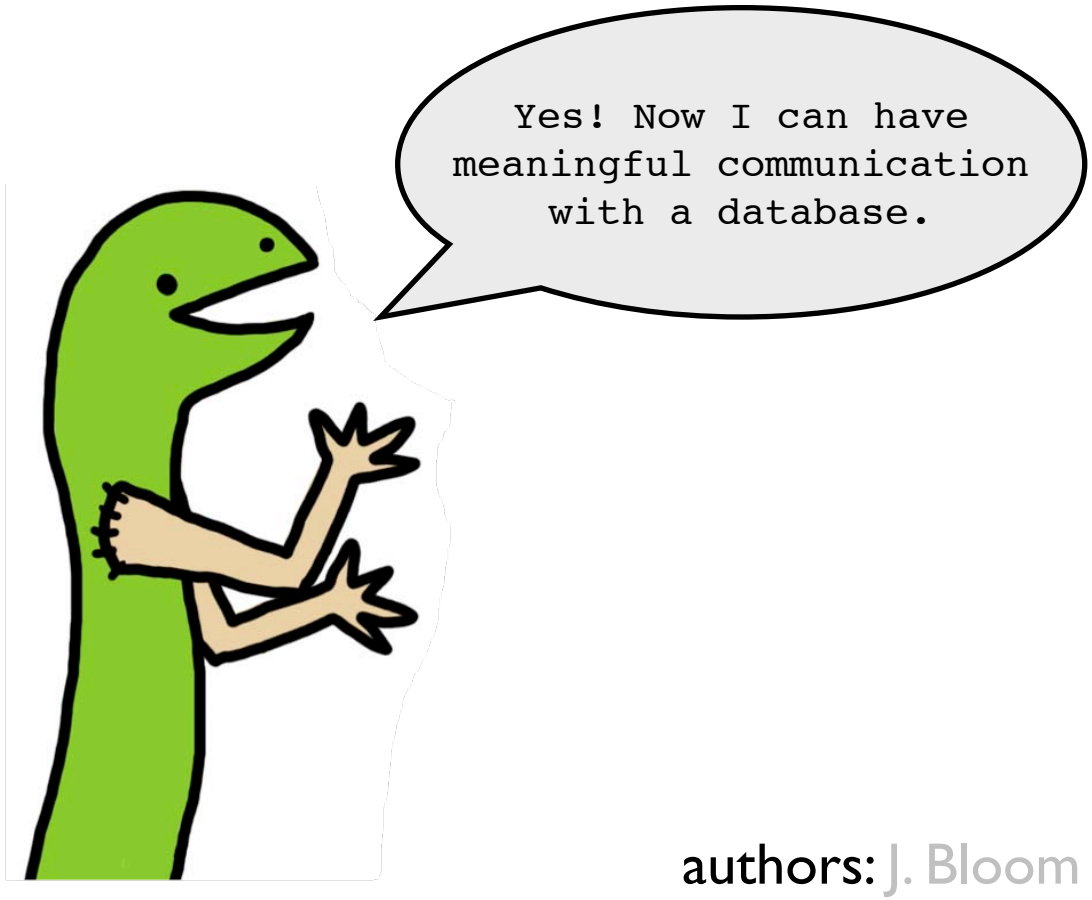
Triggers

built in actions that the DB takes on itself when something it done to it. E.g., log all the insert actions in a separate log table

Transactions

series of DB interactions that can be committed "all at once" or "rolledback" if there is a problem

Practical Database Usage in Python



authors: J. Bloom

Outline

1) Built-in Database Module
    - sqlite3
    - Plotting from a database
2) MySQL, PostgreSQL
    - MySQLdb, PyGreSQL, others
3) Breakout Exercise
    - Plotting seismograph stations
4) HDF
    - tables, netCDF4

sqlite3

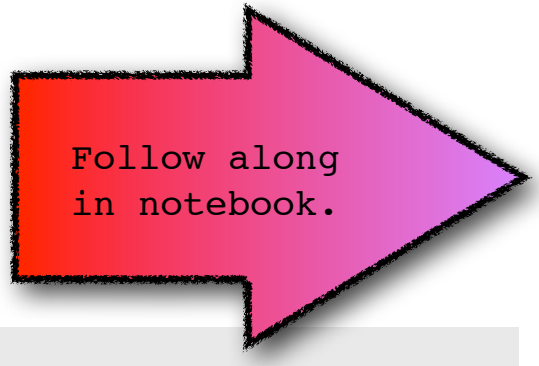
- Built-in SQL database access
- Database is stored as a file (or in RAM)
- Syntax uses Structured Query Language
- Attempts to protect the database from corruption

The next slides will show how to

- create a database
- create a table
- insert some data
- join specific columns of tables
- left join tables

sqlite3

create a database & table




```
>>> import sqlite3
>>> connection = sqlite3.connect("/tmp/example.db")
>>> cursor = connection.cursor()
>>> sql_cmd = """CREATE TABLE dan_aykroyd (id INTEGER PRIMARY KEY AUTOINCREMENT,
...     skit_title TEXT, air_date DATE, season INT, ep INT, role TEXT)"""
>>> cursor.execute(sql_cmd)
```

example.db

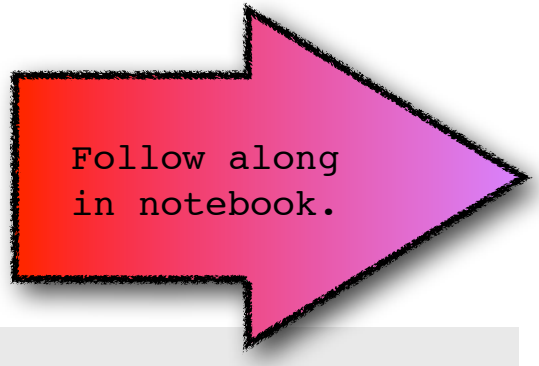
dan_aykroyd

id	skit_title	air_date	season	ep	role

[continue interpreter session next slide]

sqlite3

insert some data




```
>>> skit_data = [
...     ("Trojan Horse Home Security", "10/11/75", 1, 1, "Kenny Vorstrather"),
...     ("E. Buzz Miller's Animal Kingdom", "2/25/78", 3, 12, "E. Buzz Miller"),
...     ("The Coneheads at Home", "10/21/78", 4, 3, "Beldar Conehead")]
>>> for role in skit_data:
...     sql_cmd = ("INSERT INTO dan_aykroyd (skit_title, air_date, season, " +
...         "ep, role) VALUES " + str(role))
>>>     cursor.execute(sql_cmd)
```

example.db

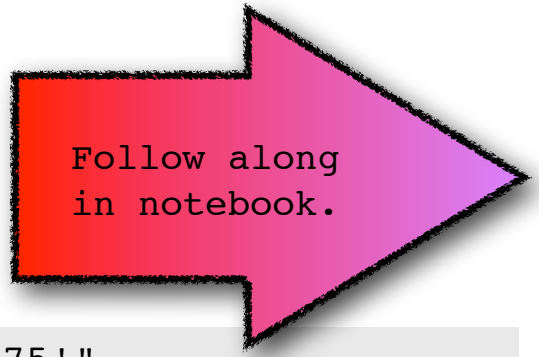
dan_aykroyd

id	skit_title	air_date	season	ep	role
1	"Trojan Horse Home Security"	"10/11/75"	1	1	"Kenny Vorstrather"
2	"E. Buzz Miller's Animal Kingdom"	"2/25/78"	3	12	"E. Buzz Miller"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Beldar Conehead"

[continue interpreter session next slide]

sqlite3

run simple query



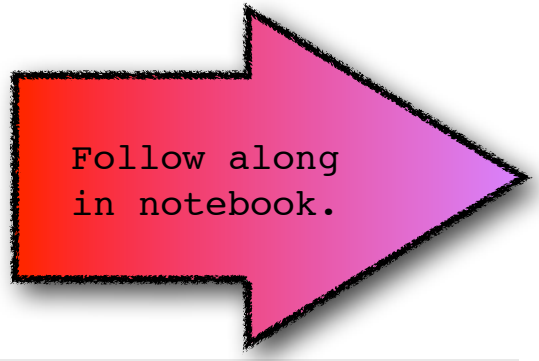

```
>>> sql_cmd = "SELECT * FROM dan_aykroyd WHERE air_date > '10/12/75'"
>>> cursor.execute(sql_cmd)
>>> db_info = cursor.fetchall()
>>> for entry in db_info: print entry
    (2, u"E. Buzz Miller's Animal Kingdom", u'2/25/78', 3, 12, u'E. Buzz Miller')
    (3, u'The Coneheads at Home', u'10/21/78', 4, 3, u'Beldar Conehead')
>>> connection.commit()
>>> connection.close()
```

example.db

dan_aykroyd

id	skit_title	air_date	season	ep	role
1	"Trojan Horse Home Security"	"10/11/75"	1	1	"Kenny Vorstrather"
2	"E. Buzz Miller's Animal Kingdom"	"2/25/78"	3	12	"E. Buzz Miller"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Beldar Conehead"

sqlite3



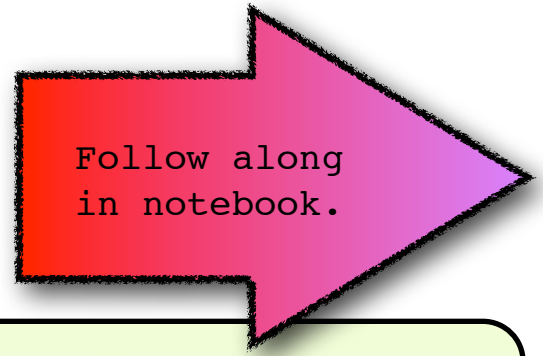
create another table, enter data, run simple join on columns

```
>>> import sqlite3
>>> connection = sqlite3.connect("/tmp/example.db")
>>> cursor = connection.cursor()
>>> sql_cmd = """CREATE TABLE jane_curtin (id INTEGER PRIMARY KEY AUTOINCREMENT,
...     skit_title TEXT, air_date DATE, season INT, episode INT, role TEXT)"""
>>> cursor.execute(sql_cmd)
>>> skit_data = [
...     ("Teen Talk", "7/24/76", 1, 23, "Jane"),
...     ("The Snakehandling O'Sheas", "9/25/76", 2, 2, "Jane O'Shea"),
...     ("The Coneheads at Home", "10/21/78", 4, 3, "Prymaat Conehead")]
>>> for role in skit_data:
...     sql_cmd = ("INSERT INTO jane_curtin (skit_title, air_date, season, " +
...         "episode, role) VALUES " + str(role))
...     cursor.execute(sql_cmd)
>>> sql_cmd = """SELECT dan_aykroyd.skit_title, dan_aykroyd.air_date,
...     dan_aykroyd.season, dan_aykroyd.episode, dan_aykroyd.role, jane_curtin.role
...     FROM dan_aykroyd, jane_curtin WHERE
...     dan_aykroyd.skit_title = jane_curtin.skit_title AND
...     dan_aykroyd.air_date = jane_curtin.air_date"""
>>> cursor.execute(sql_cmd)
>>> db_info = cursor.fetchall()
>>> for entry in db_info: print entry
(u'The Coneheads at Home', u'10/21/78', 4, 3, u'Beldar Conehead', u'Prymaat Conehead')
```

[continue interpreter session next code slide]

sqlite3

join specific columns



example.db

dan_aykroyd

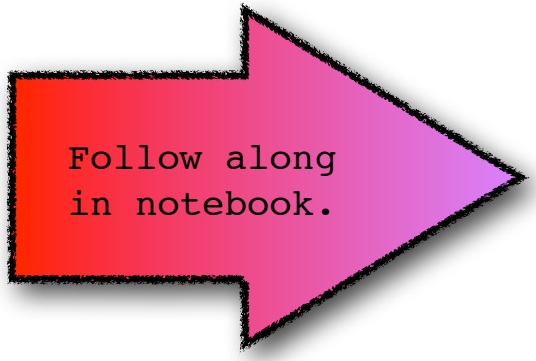
id	skit_title	air_date	season	ep	role
1	"Trojan Horse Home Security"	"10/11/75"	1	1	"Kenny Vorstrather"
2	"E. Buzz Miller's Animal Kingdom"	"2/25/78"	3	12	"E. Buzz Miller"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Beldar Conehead"

jane_curtin

id	skit_title	air_date	season	ep	role
1	"Teen Talk"	"7/24/76"	1	23	"Jane"
2	"The Snakehandling O'Sheas"	"9/25/76"	2	2	"Jane O'Shea"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Prymaat Conehead"

sqlite3

run left join



[interpreter session continued from previous code slide]

```
>>> sql_cmd = """SELECT dan_aykroyd.skit_title, dan_aykroyd.air_date,
...     dan_aykroyd.season, dan_aykroyd.ep, dan_aykroyd.role, jane_curtin.role
...     FROM dan_aykroyd LEFT JOIN jane_curtin ON
...     dan_aykroyd.skit_title = jane_curtin.skit_title AND
...     dan_aykroyd.air_date = jane_curtin.air_date"""
>>> cursor.execute(sql_cmd)
>>> db_info = cursor.fetchall()
>>> for entry in db_info: print entry
(u'Trojan Horse Home Security', u'10/11/75', 1, 1, u'Kenny Vorstrather', None)
(u"E. Buzz Miller's Animal Kingdom", u'2/25/78', 3, 12, u'E. Buzz Miller', None)
(u'The Coneheads at Home', u'10/21/78', 4, 3, u'Beldar Conehead', u'Prymaat Conehead')
>>> connection.commit()
>>> connection.close()
```

sqlite3

left join

dan_aykroyd

id	skit_title	air_date	season	ep	role
1	"Trojan Horse Home Security"	"10/11/75"	1	1	"Kenny Vorstrather"
2	"E. Buzz Miller's Animal Kingdom"	"2/25/78"	3	12	"E. Buzz Miller"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Beldar Conehead"

jane_curtin

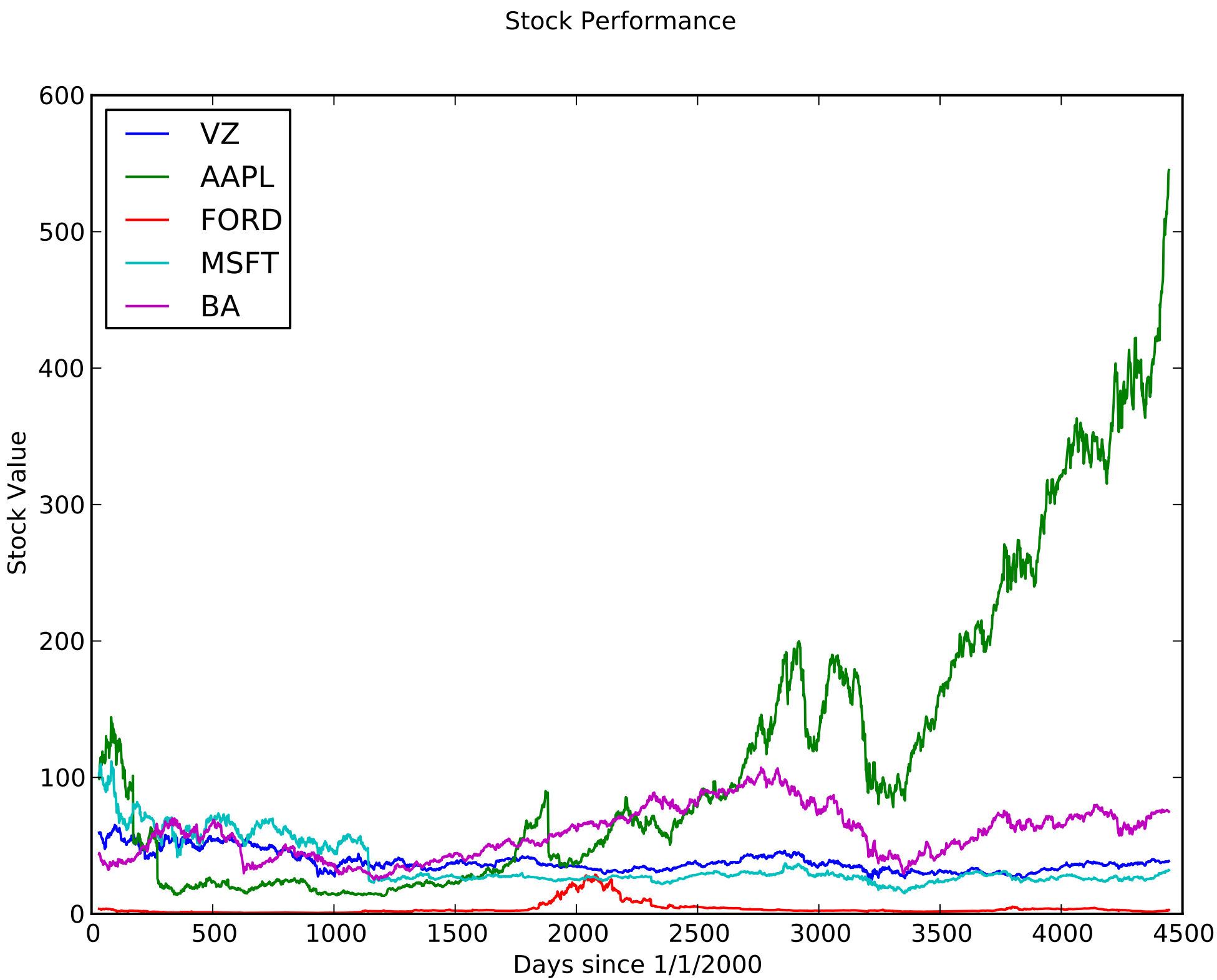
id	skit_title	air_date	season	ep	role
1	"Teen Talk"	"7/24/76"	1	23	"Jane"
2	"The Snakehandling O'Sheas"	"9/25/76"	2	2	"Jane O'Shea"
3	"The Coneheads at Home"	"10/21/78"	4	3	"Prymaat Conehead"

left-joined table

id	skit_title	air_date	season	ep	dan_aykroyd.role	jane_curtin.role
1	"Trojan Horse Home Security"	"10/11/75"	1	1	"Kenny Vorstrather"	None
2	"E. Buzz Miller's Animal Kingdom"	"2/25/78"	3	12	"E. Buzz Miller"	None
3	"The Coneheads at Home"	"10/21/78"	4	3	"Beldar Conehead"	"Prymaat Conehead"

sqlite3

example with plotting
data retrieval


Stock Performance
VZ
AAPL
FORD
MSFT
BA
Stock Value
Days since 1/1/2000
0
100
200
300
400
500
600
0
500
1000
1500
2000
2500
3000
3500
4000
4500

MySQLdb

"**MySQLdb** is a thread-compatible interface to the popular MySQL database server that provides the Python database API."

Not built-in; get it from:
http://sourceforge.net/projects/mysql-python/

Project page:
http://mysql-python.sourceforge.net/MySQLdb.html

Need MySQL installed, can be tricky on some distributions.
Common syntax with **sqlite3**.

```
sudo brew install mysql; pip install mysql-python
```

MySQLdb

exposing the database

```
import MySQLdb
conn = MySQLdb.connect(host="host", user="user",
    passwd="password", db="database",
    connect_timeout=60)
cursor = conn.cursor()
query_command = "SELECT * FROM table"
cursor.execute(query_command)
db_info = cursor.fetchall()
cursor.close()
conn.close()
```

MySQLdb translates the data-types

Above example is a query, but any MySQL command can be sent
(respecting user's permissions)

PyGreSQL and Others

Analogous to MySQLdb for PostgreSQL.

PyGreSQL - Unix/Windows, Python 2.3-2.6
pg8000 - any OS, Python 2.5+ & 3.0+
py-postgresql - any OS, Python 3.0+

Breakout Exercise

1) Plot Mercator world map using latitude and longitude points from “world.txt”. Polygons are separated by rows containing “nan nan”.

2) Import data from “stations_list.txt” into a local database. From now on, retrieve data from the local database before plotting.

3) Plot the stations on the Mercator map. Use different colors for Open, Reserved, and Closed. Explore plotting different station networks with different colors.

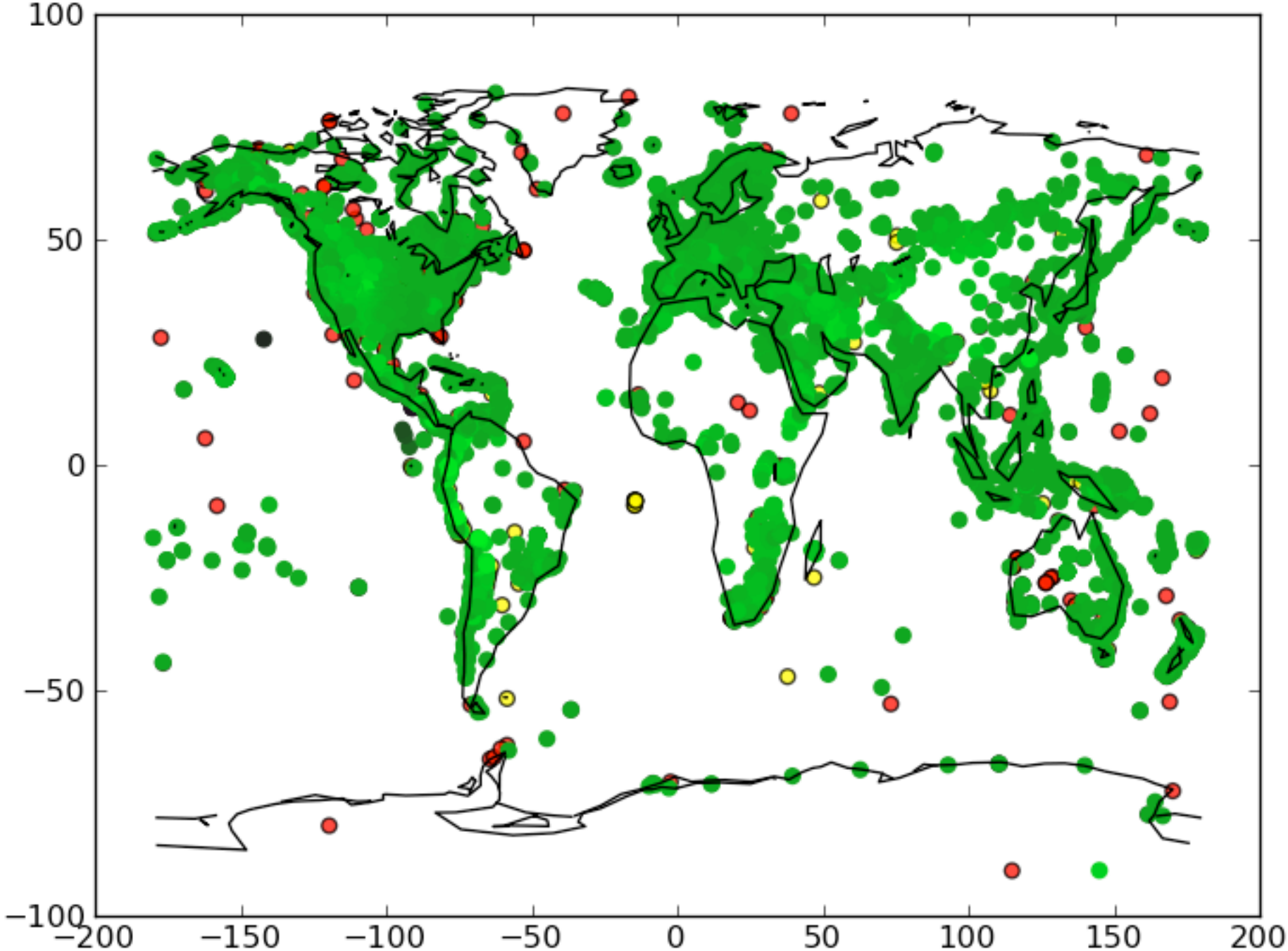
Extra Credit) Colorize Open stations as a function of elevation.

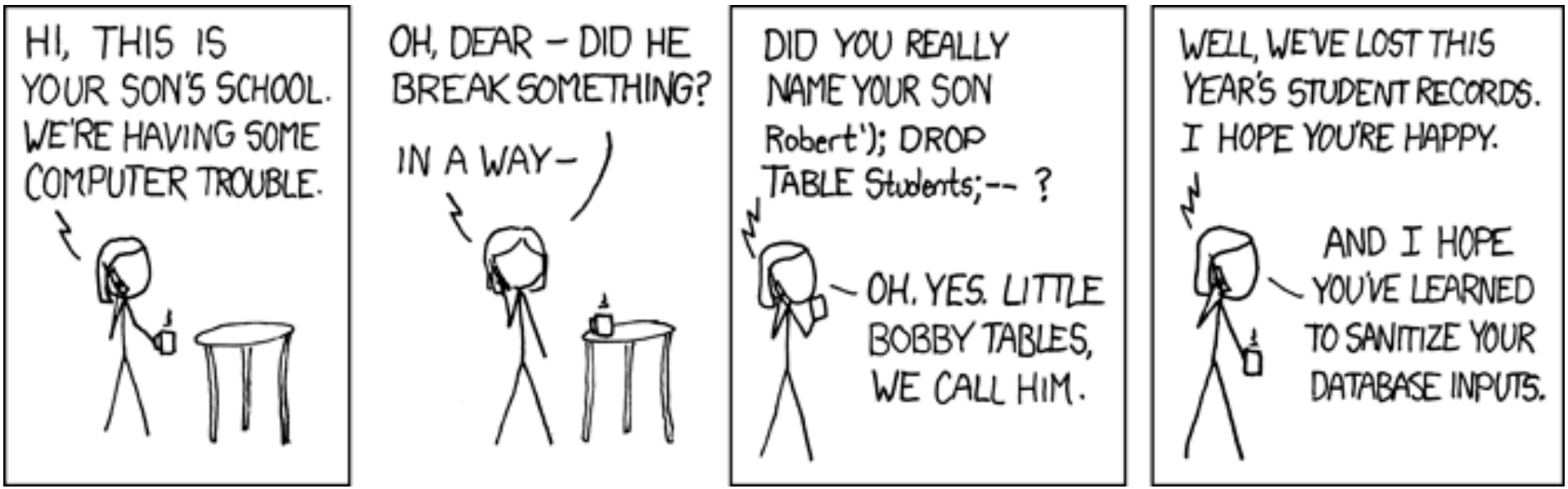
Breakout Solution

Using a database allows you to avoid the work of separating out the data in python before plotting in different colors or styles. Naively, this could be done with a loop and logic test

```
for station in station_list:
  if “closed”:
    color=“red”
```

but, we can be smarter with a database.


100
50
0
−50
−100
−200
−150
−100
−50
0
50
100
150
200



http://xkcd.com/327/

SQLAlchemy

"Object Relational Mapper"

Drizzle | Firebird | Informix | MaxDB | Microsoft Access | Microsoft SQL Server | MySQL | Oracle | PostgreSQL | SQLite | Sybase

- associates Python classes with database tables, and instances of those classes (objects) with rows in their corresponding tables.

- "high level and abstracted pattern of usage"

-essentially, making the Python usage almost entirely independent of the database engine

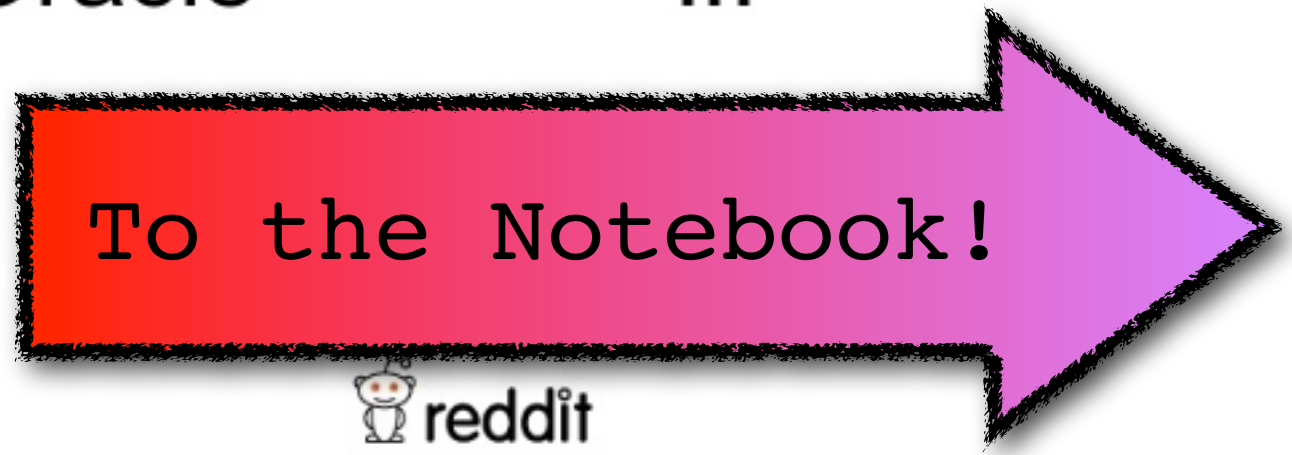
SQLAlchemy

http://docs.sqlalchemy.org/en/latest/index.html

http://techspot.zzzeek.org/

Supports

- SQLite
- Postgresql
- MySQL
- Oracle
- MS-SQL
- Firebird
- Sybase
- ...



reddit

yelp

Dropbox

Analytics: Scientific Data Management

Scientific Data Formats

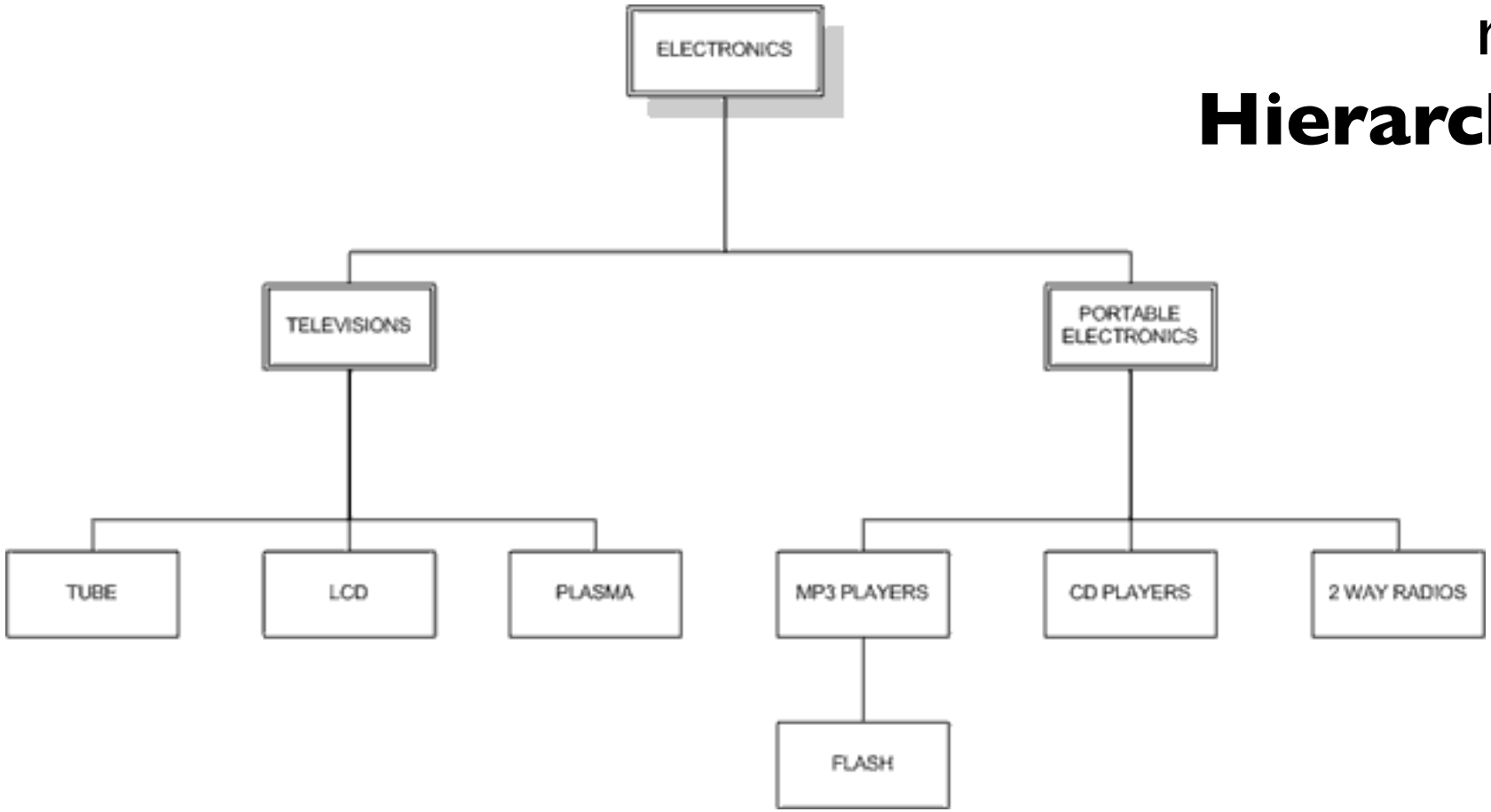
Scientific data is often based on numerical arrays that are stored in scientific data formats such as NetCDF (network common data format), HDF5 (hierarchical data format) and FITS (Flexible Image Transport System). The data formats support efficient mechanisms for accessing and manipulating arrays and allow machine-independent data exchange. They also provide mechanisms for storing metadata information such as simulation parameters.

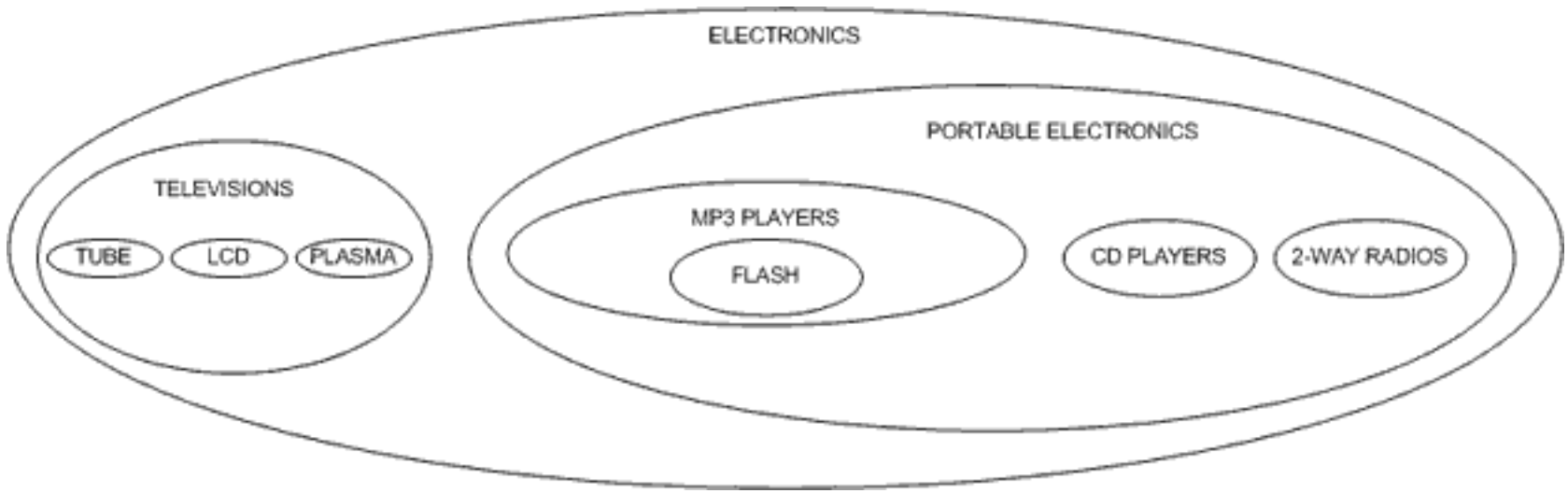
HDF5 supports parallel I/O which allows multiple processors to read and write files. This is particularly important for large-scale data intensive applications. NetCDF, on the other hand, only supports serial I/O. Parallel I/O is provided by ParallelNetCDF.

Currently there is no common scientific data format that is used across different scientific disciplines. For instance, NetCDF commonly is used in the climate modeling community. FITS is the preferred format for storing astronomy data, while HDF5 is used, for instance, in the combustion and fusion simulation communities.

recall:

Hierarchical





C-based API "database," developed for scientific applications by NCSA

Views data like a file system (with paths /food/spam/ingredients). You connect to the data on disk (like in sqlite3)

Two types of objects in HDF5:

- Datasets - multidimensional homogeneous arrays. They can have attributes (metadata).
  - Table: like record-arrays in numpy
  - Array
- Groups - containers holding datasets and other groups (like "folders")

Several Nice Things about HDF5 itself:

- portability - document is self-describing and platform independent (endian issues are dealt with client-side)
- hugely scalable datamodel (2**63-sized tables)
- Clients made to run massively parallel

`pytables` is a Pythonic API of HDF5

- happy to use numpy arrays
- works on compressed datafiles
- open-sourced

/ attr

`tables.openFile`

/ is the root

/Datasets attr attr

`tables.createGroup`

/Datasets attr attr
myarray attr

`tables.createArray`

note: different flavors of arrays
unlike numpy arrays, HDF5 arrays can be extended

/Datasets attr attr
myarray attr
mytable attr

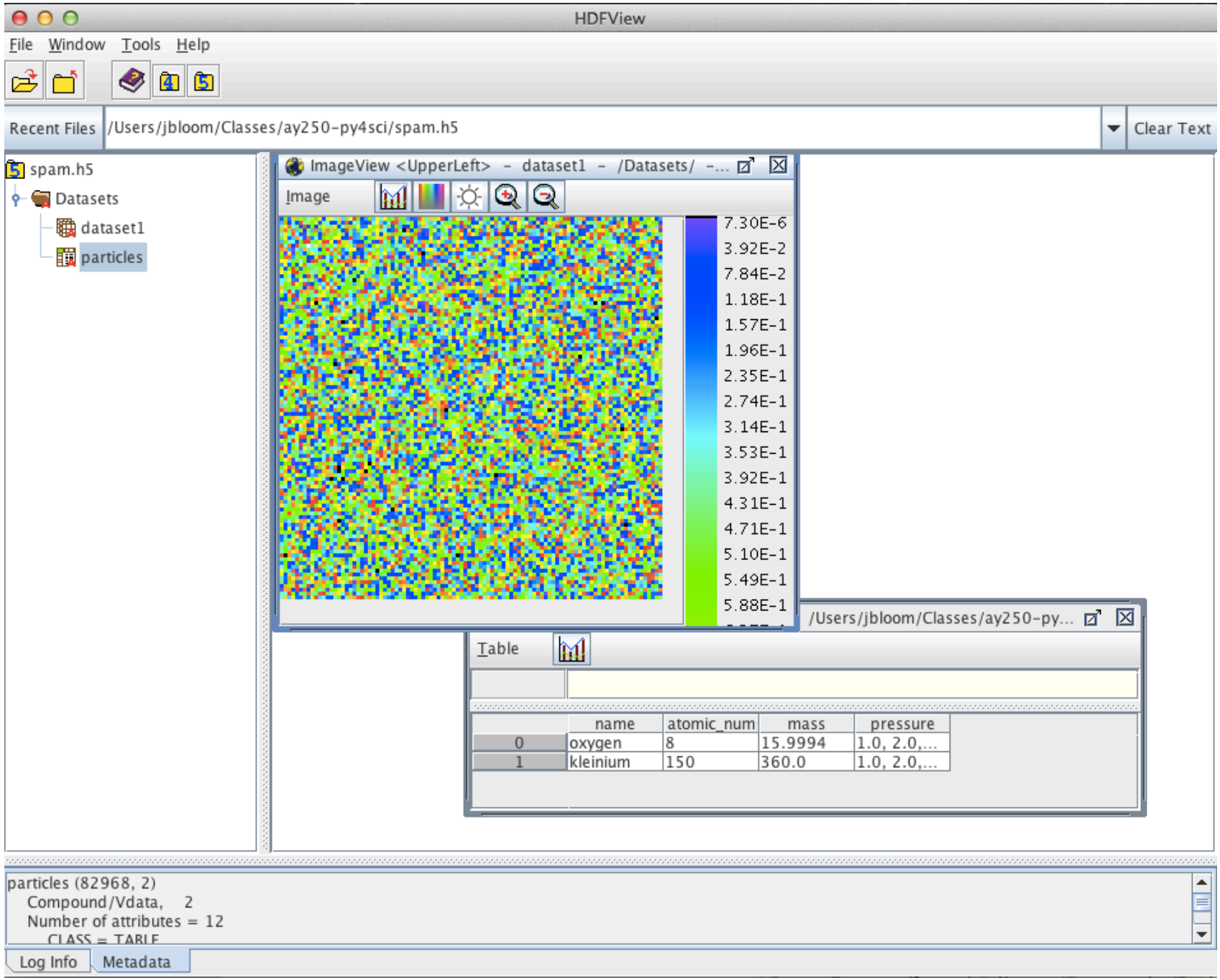
`tables.createTable`

Also `createExternalLink` creates a link to another HDF5

Making an HDF5 file

```
>>> from tables import *
>>> h5file = openFile("spam.h5",mode = "w", title = "PyTables/HDF5 test file")
>>> h5file
>>> datasets = h5file.createGroup(h5file.root, "Datasets", "Test data sets")
>>> h5file.createArray(datasets, 'dataset1', np.random.random((100,100)), "Test data set #1")
>>> class Particle(IsDescription):
    name        = StringCol(16, pos=1) # 16-character String
    atomic_num  = IntCol(pos=2)        # integer
    mass        = FloatCol(pos=3)      # double (double-precision)
    pressure    = Float32Col(shape=(2,3))
>>> table1 = h5file.createTable(datasets, "particles", Particle)
>>> row = table1.row
>>> row
/Datasets/particles.row (Row), pointing to row #0
>>> row["name"] = "oxygen"
>>> row["atomic_num"] = 8
>>> row["mass"] = 15.9994
>>> row["pressure"] = [[1,2,3],[-1,1,3]]
>>> row.append() ; table1.flush()
>>> h5file.root.Datasets.particles[0]
('oxygen', 8, 15.9994, [[1.0, 2.0, 3.0], [-1.0, 1.0, 3.0]])
>>> [row['name'] for row in table1.where('atomic_num > 5 and mass > 5.0')]
['oxygen']

>>> h5file.close()
```



http://www.hdfgroup.org/hdf-java-html/hdfview/

“Python and HDF5 - Fast Storage for Large Data”

https://us.pycon.org/2012/schedule/presentation/231/

netCDF4

another hierarchical dataformat
(used primarily in climate modeling)

OPeNDAP

Open-source Project for a Network Data Access Protocol

protocols for interacting with remote sites serving
netCDF4, HDF5, etc. data.
Python's netCDF4 module allows you to open
URLs and interact with the data as if it was local

Fast, Distributed DBs (e.g., HBase)

APACHE HBASE

Search wiki, mailing lists & more Search

Built by: maven

Welcome to Apache HBase!

HBase is the Hadoop database. Think of it as a distributed scalable Big Data store.

When Would I Use HBase?

Use HBase when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware. HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

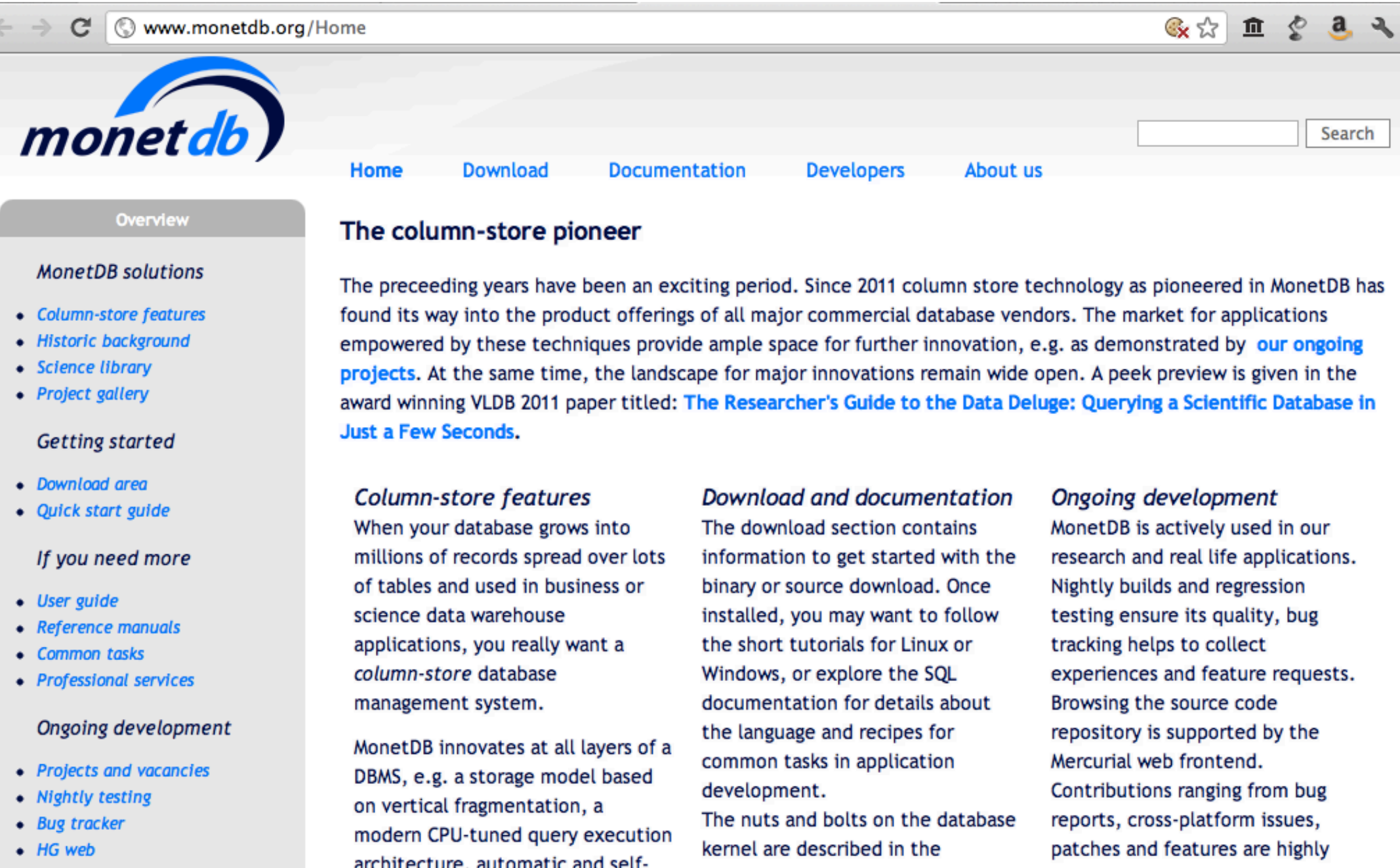
Features

HBase provides:

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with HBase tables.
- Easy to use Java API for client access.
- Block cache and Bloom Filters for real-time queries.
- Query predicate push down via server side Filters
- Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
- Extensible jruby-based (JIRB) shell
- Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX

Column-store DBs

large data amenable to map-reduce problems



SciDB

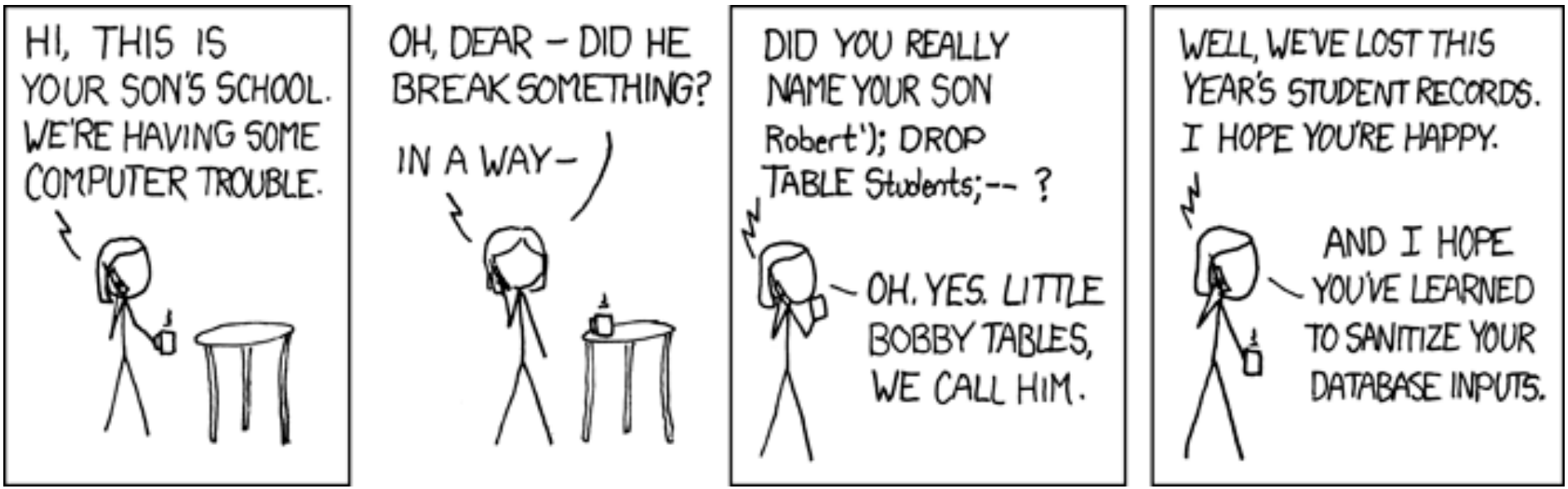
Get Latest SciDB SHARE

ABOUT | USE CASES | GET INVOLVED | DOWNLOAD | NEWS & EVENTS | FORUM

HEART BEAT: **Oct 23, 2011:** Slides **from the 2nd SciDB Community Meeting have been posted**

June 15, 2011: Release 11.06 is Available

Website Design by MicroArts Creative Agency



http://xkcd.com/327/