Reading mySQL

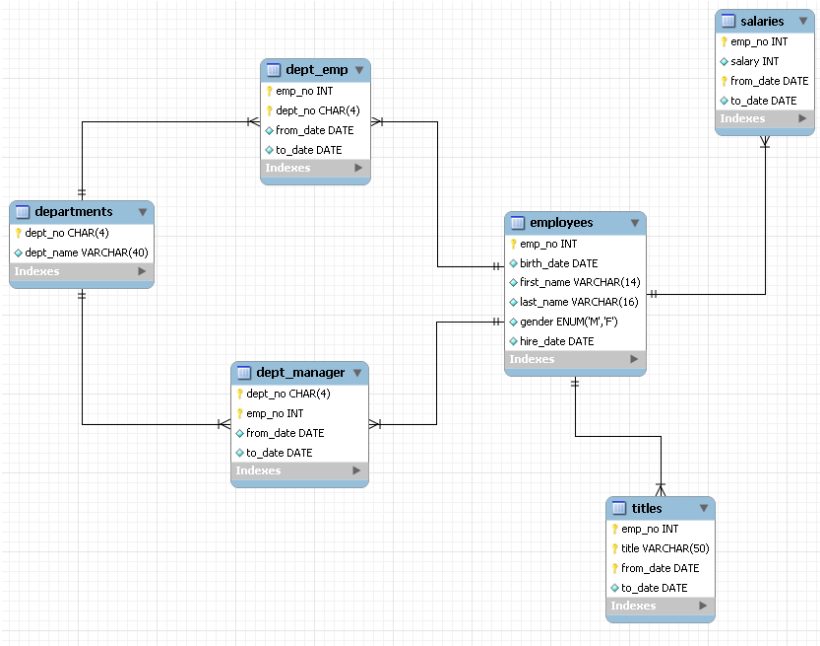
Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

mySQL

- Free and widely used open source database software
- Widely used in internet based applications
- Data are structured in
  - Databases
  - Tables within databases
  - Fields within tables
- Each row is called a record

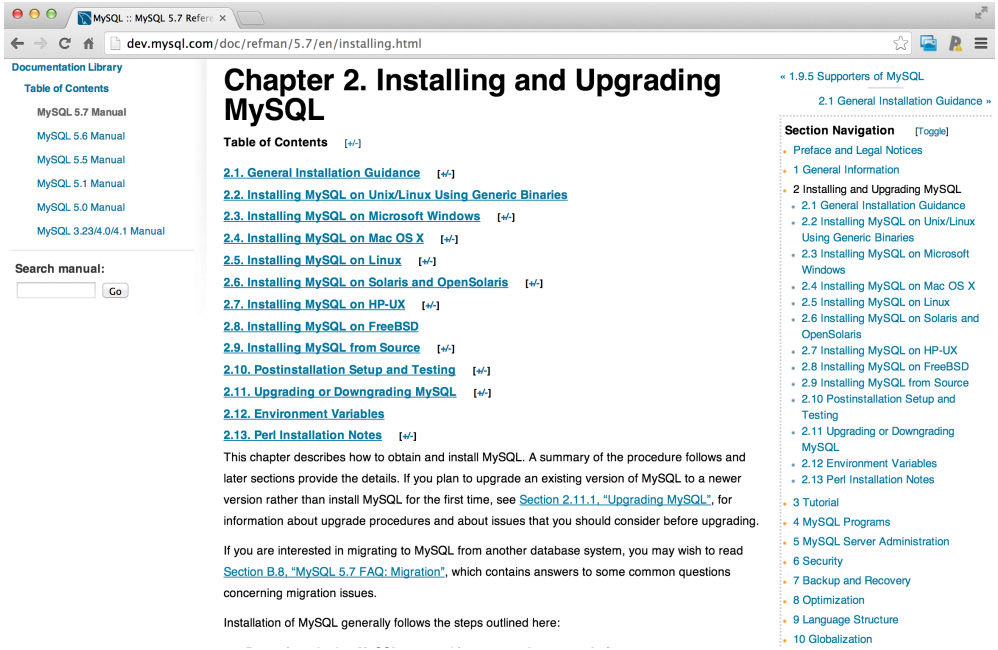
http://en.wikipedia.org/wiki/MySQL http://www.mysql.com/

Example structure



http://dev.mysql.com/doc/employee/en/sakila-structure.html

Step 1 - Install MySQL

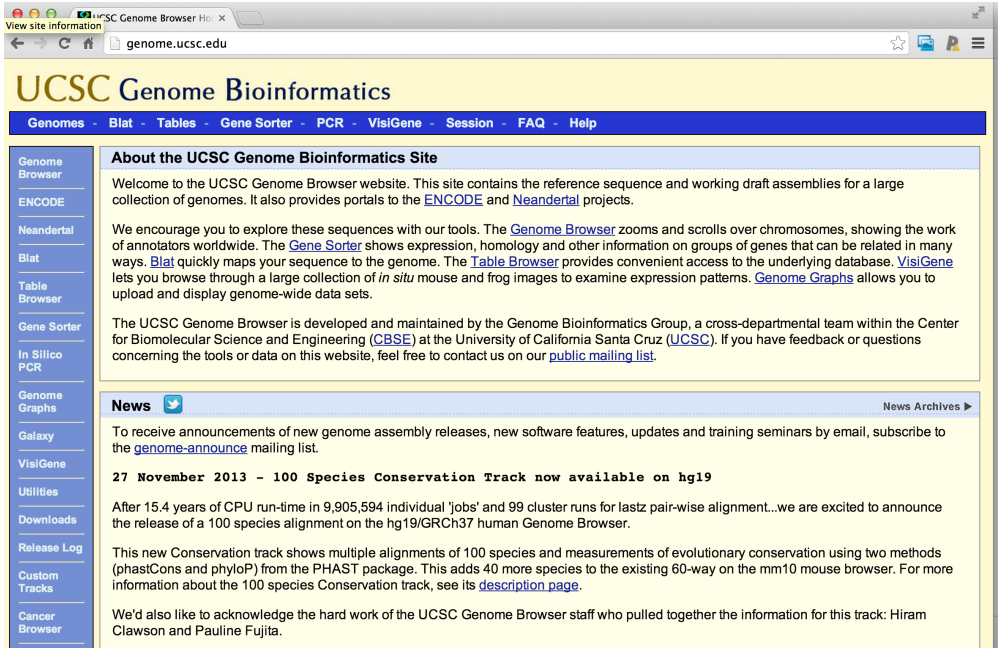


http://dev.mysql.com/doc/refman/5.7/en/installing.html

Step 2 - Install RMySQL

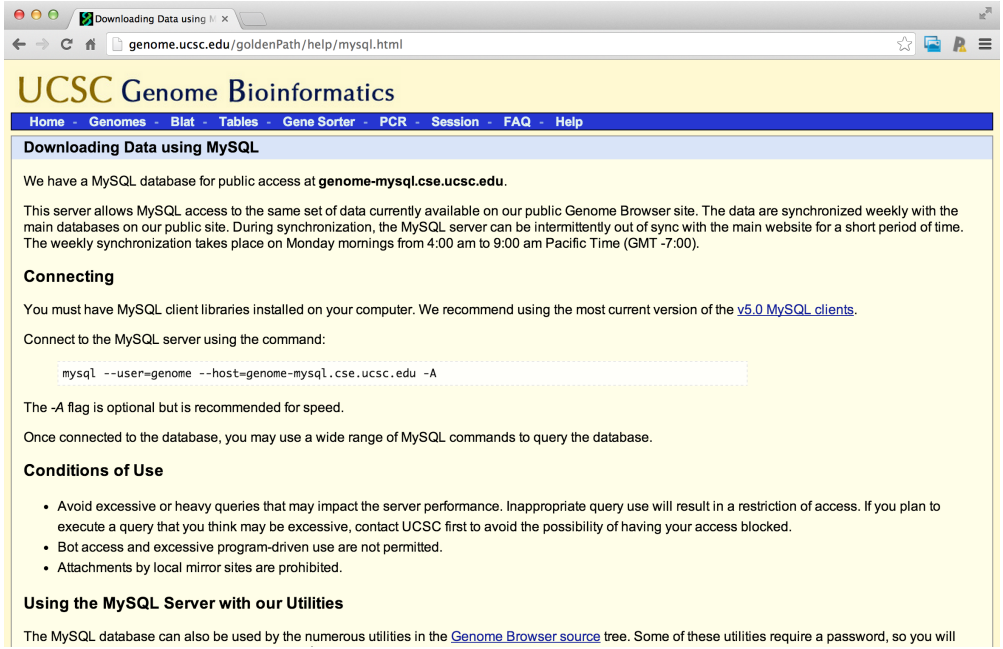
- On a Mac: `install.packages("RMySQL")`
- On Windows:
  - Official instructions - http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL (may be useful for Mac/UNIX users as well)
  - Potentially useful guide - http://www.ahschulz.de/2013/07/23/installing-rmysql-under-windows/

Example - UCSC database



http://genome.ucsc.edu/

UCSC MySQL



http://genome.ucsc.edu/goldenPath/help/mysql.html

Connecting and listing databases

```
ucscDb <- dbConnect(MySQL(),user="genome",
                    host="genome-mysql.cse.ucsc.edu")
result <- dbGetQuery(ucscDb,"show databases;"); dbDisconnect(ucscDb);
```

```
[1] TRUE
```

```
result
```

```
             Database
1   information_schema
2              ailMel1
3              allMis1
4              anoCar1
5              anoCar2
6              anoGam1
7              apiMel1
8              apiMel2
```

Connecting to hg19 and listing tables

```
hg19 <- dbConnect(MySQL(),user="genome", db="hg19",
                  host="genome-mysql.cse.ucsc.edu")
allTables <- dbListTables(hg19)
length(allTables)
```

```
[1] 10949
```

```
allTables[1:5]
```

```
[1] "HInv"         "HInvGeneMrna" "acembly"      "acemblyClass" "acemblyPep"
```

Get dimensions of a specific table

```
dbListFields(hg19,"affyU133Plus2")
```

```
 [1] "bin"         "matches"     "misMatches"  "repMatches"  "nCount"      "qNumInsert"
 [7] "qBaseInsert" "tNumInsert"  "tBaseInsert" "strand"      "qName"       "qSize"
[13] "qStart"      "qEnd"        "tName"       "tSize"       "tStart"      "tEnd"
[19] "blockCount"  "blockSizes"  "qStarts"     "tStarts"
```

```
dbGetQuery(hg19, "select count(*) from affyU133Plus2")
```

```
  count(*)
1    58463
```

Read from the table

```
affyData <- dbReadTable(hg19, "affyU133Plus2")
head(affyData)
```

```
  bin matches misMatches repMatches nCount qNumInsert qBaseInsert tNumInsert tBaseInsert strand
1 585     530          4          0     23          3          41          3         898      -
2 585    3355         17          0    109          9          67          9       11621      -
3 585    4156         14          0     83         16          18          2          93      -
4 585    4667          9          0     68         21          42          3        5743      -
5 585    5180         14          0    167         10          38          1          29      -
6 585     468          5          0     14          0           0          0           0      -
         qName qSize qStart qEnd tName     tSize tStart  tEnd blockCount
1  225995_x_at   637      5  603  chr1 249250621  14361 15816          5
2  225035_x_at  3635      0 3548  chr1 249250621  14381 29483         17
3  226340_x_at  4318      3 4274  chr1 249250621  14399 18745         18
4 1557034_s_at  4834     48 4834  chr1 249250621  14406 24893         23
5    231811_at  5399      0 5399  chr1 249250621  19688 25078         11
6    236841_at   487      0  487  chr1 249250621  27542 28029          1
                                                                blockSizes
1                                                          93,144,229,70,21,
2                 73,375,71,165,303,360,198,661,201,1,260,250,74,73,98,155,163,
```

Select a specific subset

```
query <- dbSendQuery(hg19, "select * from affyU133Plus2 where misMatches between 1 and 3")
affyMis <- fetch(query); quantile(affyMis$misMatches)
```

```
  0%  25%  50%  75% 100%
   1    1    2    2    3
```

```
affyMisSmall <- fetch(query,n=10); dbClearResult(query);
```

```
[1] TRUE
```

```
dim(affyMisSmall)
```

```
[1] 10 22
```

Don't forget to close the connection!

```
dbDisconnect(hg19)
```

```
[1] TRUE
```

Further resources

- RMySQL vignette http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf
- List of commands http://www.pantz.org/software/mysql/mysqlcommands.html
  - **Do not, do not, delete, add or join things from ensembl. Only select.**
  - In general be careful with mysql commands
- A nice blog post summarizing some other commands http://www.r-bloggers.com/mysql-and-r/

Reading HDF5

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

HDF5

- Used for storing large data sets
- Supports storing a range of data types
- Heirarchical data format
- *groups* containing zero or more data sets and metadata
  - Have a *group header* with group name and list of attributes
  - Have a *group symbol table* with a list of objects in group
- *datasets* multidmensional array of data elements with metadata
  - Have a *header* with name, datatype, dataspace, and storage layout
  - Have a *data array* with the data

http://www.hdfgroup.org/

R HDF5 package

```
source("http://bioconductor.org/biocLite.R")
biocLite("rhdf5")
```

```
library(rhdf5)
created = h5createFile("example.h5")
created
```

```
[1] TRUE
```

- This will install packages from Bioconductor http://bioconductor.org/, primarily used for genomics but also has good "big data" packages
- Can be used to interface with hdf5 data sets.
- This lecture is modeled very closely on the rhdf5 tutorial that can be found here http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf

Create groups

```
created = h5createGroup("example.h5","foo")
created = h5createGroup("example.h5","baa")
created = h5createGroup("example.h5","foo/foobaa")
h5ls("example.h5")
```

```
  group    name      otype dclass dim
0     /     baa H5I_GROUP
1     /     foo H5I_GROUP
2  /foo foobaa H5I_GROUP
```

Write to groups

```
A = matrix(1:10,nr=5,nc=2)
h5write(A, "example.h5","foo/A")
B = array(seq(0.1,2.0,by=0.1),dim=c(5,2,2))
attr(B, "scale") <- "liter"
h5write(B, "example.h5","foo/foobaa/B")
h5ls("example.h5")
```

```
        group   name       otype  dclass       dim
0           /    baa   H5I_GROUP                  
1           /    foo   H5I_GROUP                  
2        /foo      A H5I_DATASET INTEGER     5 x 2
3        /foo foobaa   H5I_GROUP                  
4 /foo/foobaa      B H5I_DATASET   FLOAT 5 x 2 x 2
```

Write a data set

```
df = data.frame(1L:5L,seq(0,1,length.out=5),
  c("ab","cde","fghi","a","s"), stringsAsFactors=FALSE)
h5write(df, "example.h5","df")
h5ls("example.h5")
```

```
        group   name       otype   dclass       dim
0           /    baa   H5I_GROUP                  
1           /     df H5I_DATASET COMPOUND         5
2           /    foo   H5I_GROUP                  
3        /foo      A H5I_DATASET  INTEGER     5 x 2
4        /foo foobaa   H5I_GROUP                  
5 /foo/foobaa      B H5I_DATASET    FLOAT 5 x 2 x 2
```

Reading data

```
readA = h5read("example.h5","foo/A")
readB = h5read("example.h5","foo/foobaa/B")
readdf= h5read("example.h5","df")
readA
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

Writing and reading chunks

```
h5write(c(12,13,14),"example.h5","foo/A",index=list(1:3,1))
h5read("example.h5","foo/A")
```

```
     [,1] [,2]
[1,]   12    6
[2,]   13    7
[3,]   14    8
[4,]    4    9
[5,]    5   10
```

Notes and further resources

- hdf5 can be used to optimize reading/writing from disc in R
- The rhdf5 tutorial:
  - http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf
- The HDF group has informaton on HDF5 in general http://www.hdfgroup.org/HDF5/

Reading data from the web

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

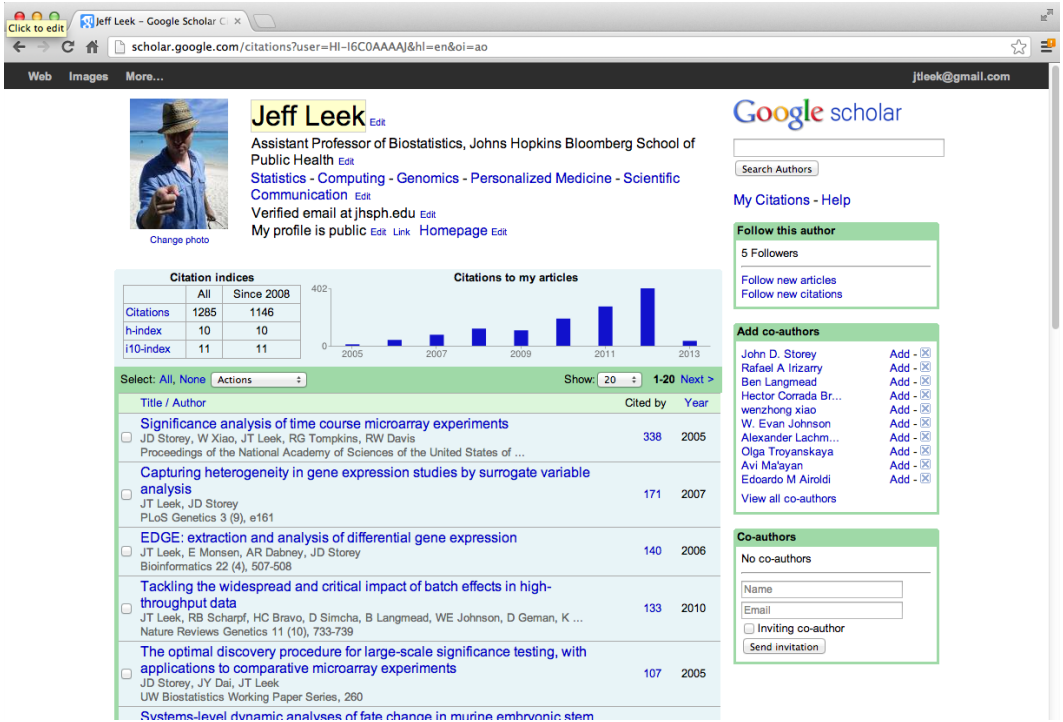
Webscraping

Webscraping: Programatically extracting data from the HTML code of websites.

- It can be a great way to get data How Netflix reverse engineered Hollywood
- Many websites have information you may want to programaticaly read
- In some cases this is against the terms of service for the website
- Attempting to read too many pages too quickly can get your IP address blocked

http://en.wikipedia.org/wiki/Web_scraping

Example: Google scholar



http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en

Getting data off webpages - readLines()

```
con = url("http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en")
htmlCode = readLines(con)
close(con)
htmlCode
```

```
[1] "<!DOCTYPE html><html><head><title>Jeff Leek - Google Scholar Citations</title><meta name=\"robots\
```

Parsing with XML

```
library(XML)
url <- "http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en"
html <- htmlTreeParse(url, useInternalNodes=T)

xpathSApply(html, "//title", xmlValue)
```

```
[1] "Jeff Leek - Google Scholar Citations"
```

```
xpathSApply(html, "//td[@id='col-citedby']", xmlValue)
```

```
 [1] "Cited by" "397"      "259"      "237"      "172"      "138"      "125"      "122"
 [9] "109"      "101"      "34"       "26"       "26"       "24"       "19"       "13"
[17] "12"       "10"       "10"       "7"        "6"
```

GET from the httr package

```
library(httr); html2 = GET(url)
content2 = content(html2,as="text")
parsedHtml = htmlParse(content2,asText=TRUE)
xpathSApply(parsedHtml, "//title", xmlValue)
```

```
[1] "Jeff Leek - Google Scholar Citations"
```

Accessing websites with passwords

```
pg1 = GET("http://httpbin.org/basic-auth/user/passwd")
pg1
```

```
Response [http://httpbin.org/basic-auth/user/passwd]
  Status: 401
  Content-type:
```

http://cran.r-project.org/web/packages/httr/httr.pdf

Accessing websites with passwords

```
pg2 = GET("http://httpbin.org/basic-auth/user/passwd",
    authenticate("user","passwd"))
pg2
```

```
Response [http://httpbin.org/basic-auth/user/passwd]
  Status: 200
  Content-type: application/json
{
  "authenticated": true,
  "user": "user"
}
```

```
names(pg2)
```

```
[1] "url"         "handle"      "status_code" "headers"     "cookies"     "content"
[7] "times"       "config"
```

Using handles

```
google = handle("http://google.com")
pg1 = GET(handle=google,path="/")
pg2 = GET(handle=google,path="search")
```

http://cran.r-project.org/web/packages/httr/httr.pdf

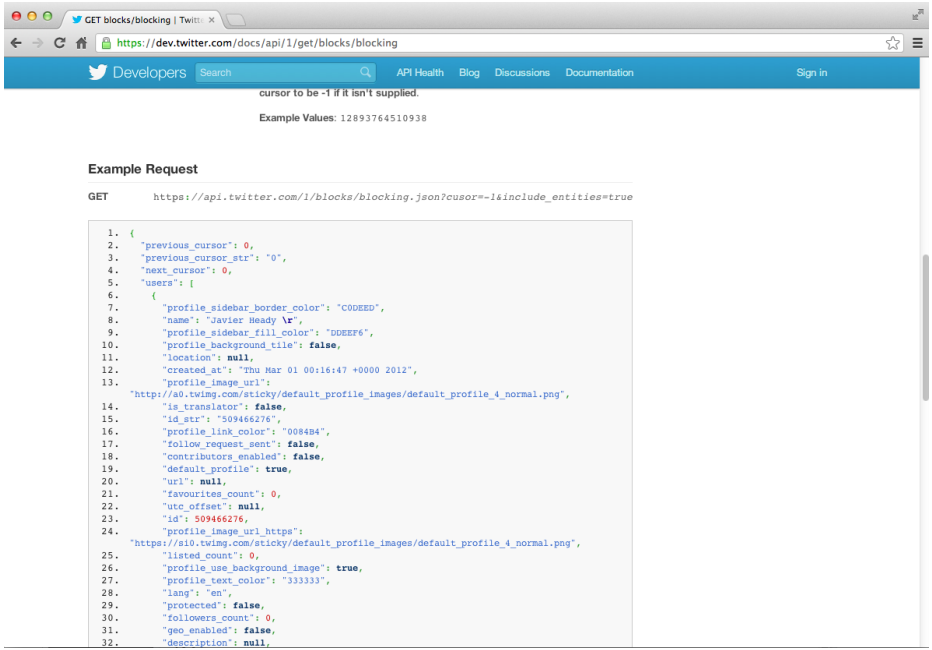
Notes and further resources

- R Bloggers has a number of examples of web scraping http://www.r-bloggers.com/?s=Web+Scraping
- The httr help file has useful examples http://cran.r-project.org/web/packages/httr/httr.pdf
- See later lectures on APIs

Reading data from APIs

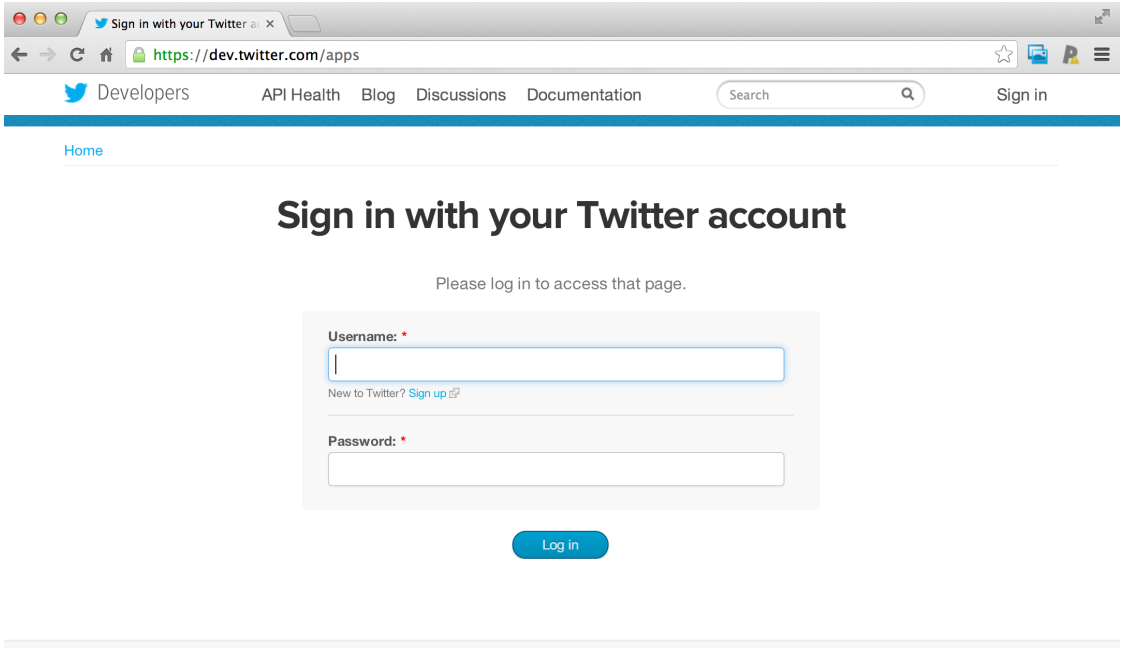
Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Application programming interfaces



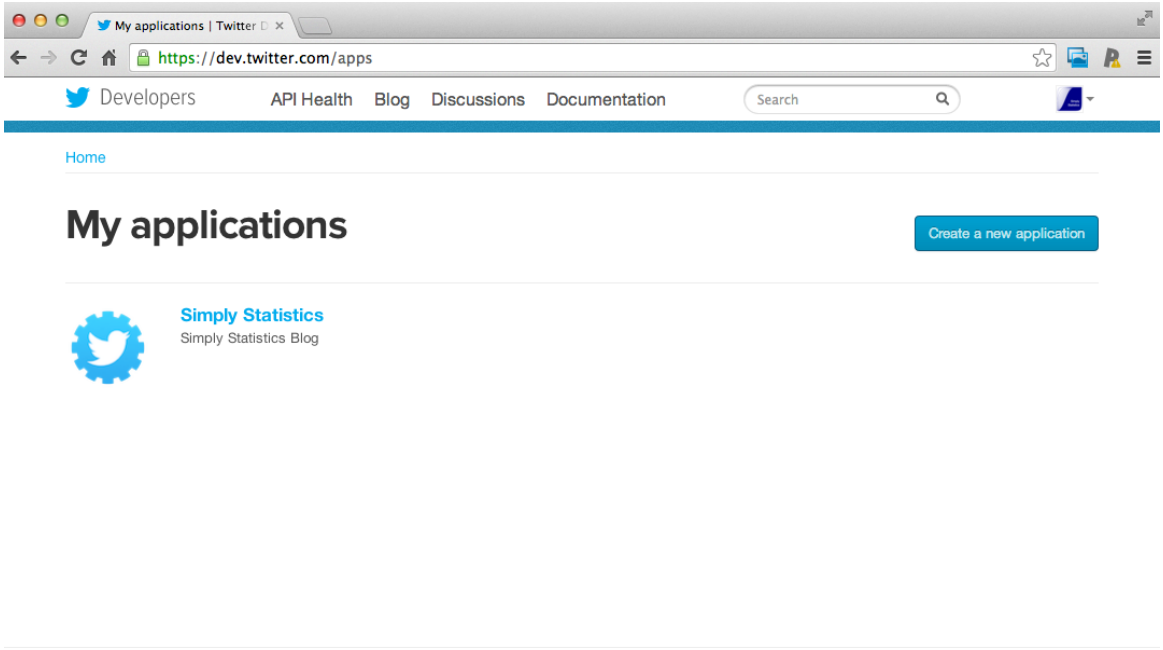
https://dev.twitter.com/docs/api/1/get/blocks/blocking

Creating an application

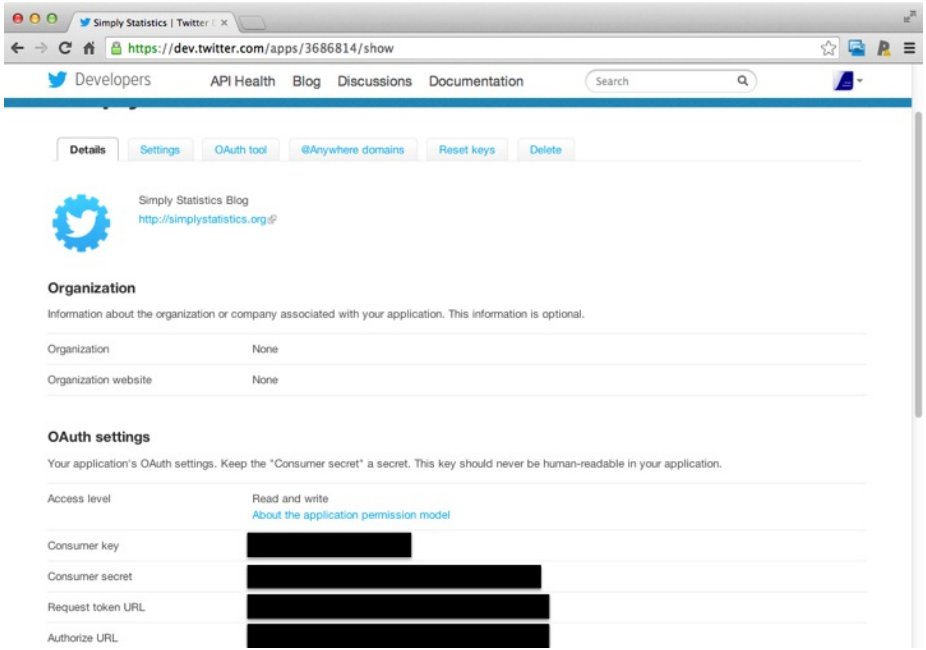


https://dev.twitter.com/apps)

Creating an application



Creating an application



Accessing Twitter from R

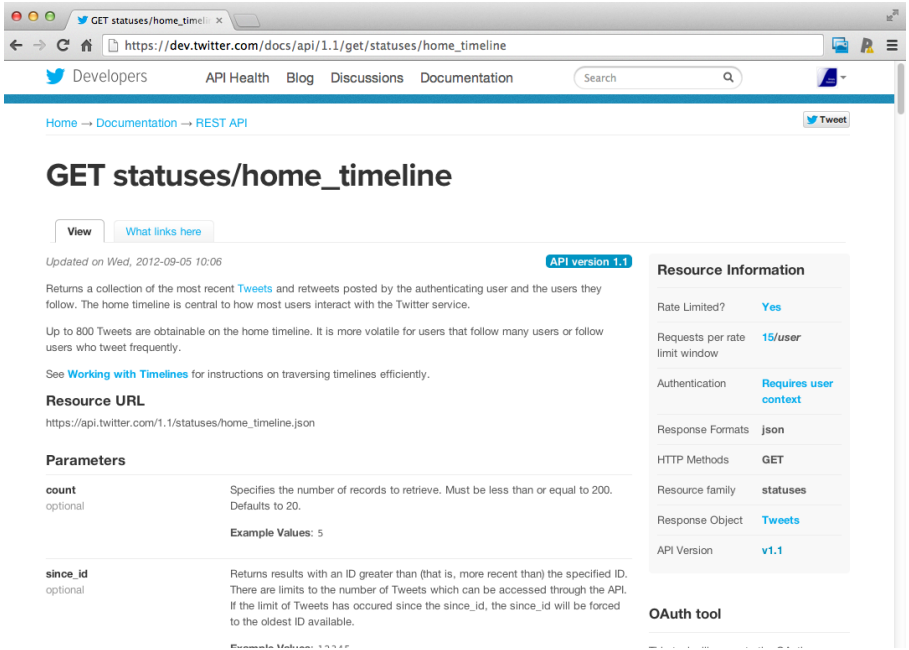
```
myapp = oauth_app("twitter",
                  key="yourConsumerKeyHere",secret="yourConsumerSecretHere")
sig = sign_oauth1.0(myapp,
                    token = "yourTokenHere",
                     token_secret = "yourTokenSecretHere")
homeTL = GET("https://api.twitter.com/1.1/statuses/home_timeline.json", sig)
```

Converting the json object

```
json1 = content(homeTL)
json2 = jsonlite::fromJSON(toJSON(json1))
json2[1,1:4]
```

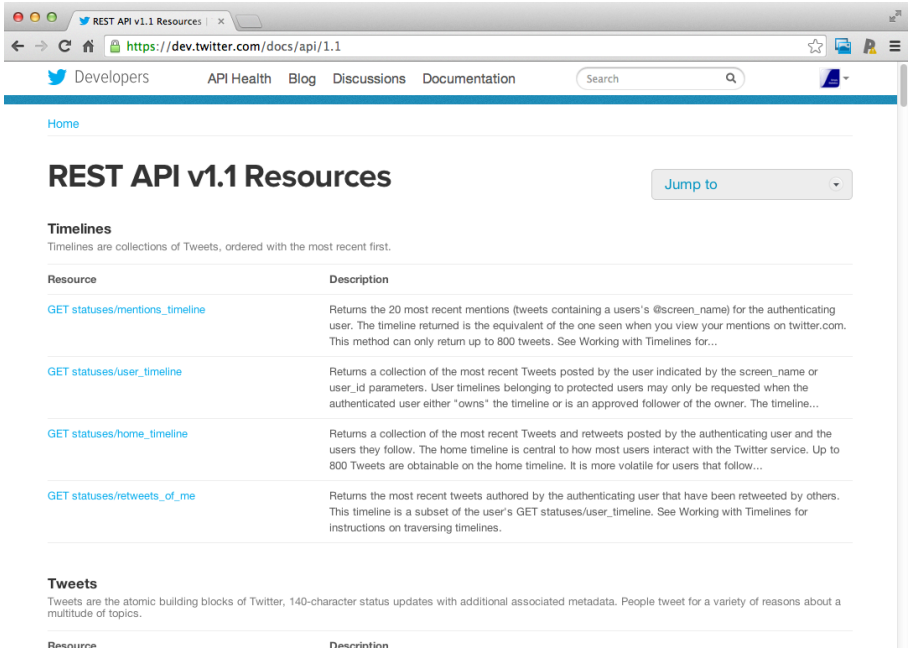
```
                    created_at           id             id_str
1 Mon Jan 13 05:18:04 +0000 2014 4.225984e+17 422598398940684288

1 Now that P. Norvig's regex golf IPython notebook hit Slashdot, let's see if our traffic spike tops th
```

How did I know what url to use?



https://dev.twitter.com/docs/api/1.1/get/search/tweets

In general look at the documentation



https://dev.twitter.com/docs/api/1.1/overview

In general look at the documentation

- httr allows `GET`, `POST`, `PUT`, `DELETE` requests if you are authorized
- You can authenticate with a user name or a password
- Most modern APIs use something like oauth
- httr works well with Facebook, Google, Twitter, Githb, etc.

Reading from other sources

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

There is a package for that

- Roger has a nice video on how there are R packages for most things that you will want to access.
- Here I'm going to briefly review a few useful packages
- In general the best way to find out if the R package exists is to Google "data storage mechanism R package"
    - For example: "MySQL R package"

Interacting more directly with files

- file - open a connection to a text file
- url - open a connection to a url
- gzfile - open a connection to a .gz file
- bzfile - open a connection to a .bz2 file
- *?connections* for more information
- Remember to close connections

foreign package

- Loads data from Minitab, S, SAS, SPSS, Stata,Systat
- Basic functions *read.foo*
  - read.arff (Weka)
  - read.dta (Stata)
  - read.mtp (Minitab)
  - read.octave (Octave)
  - read.spss (SPSS)
  - read.xport (SAS)
- See the help page for more details http://cran.r-project.org/web/packages/foreign/foreign.pdf

Examples of other database packages

- RPostresSQL provides a DBI-compliant database connection from R. Tutorial-https://code.google.com/p/rpostgresql/, help file-http://cran.r-project.org/web/packages/RPostgreSQL/RPostgreSQL.pdf
- RODBC provides interfaces to multiple databases including PostgreQL, MySQL, Microsoft Access and SQLite. Tutorial - http://cran.r-project.org/web/packages/RODBC/vignettes/RODBC.pdf, help file - http://cran.r-project.org/web/packages/RODBC/RODBC.pdf
- RMongo http://cran.r-project.org/web/packages/RMongo/RMongo.pdf (example of Rmongo http://www.r-bloggers.com/r-and-mongodb/) and rmongodb provide interfaces to MongoDb.

Reading images

- jpeg - http://cran.r-project.org/web/packages/jpeg/index.html
- readbitmap - http://cran.r-project.org/web/packages/readbitmap/index.html
- png - http://cran.r-project.org/web/packages/png/index.html
- EBImage (Bioconductor) - http://www.bioconductor.org/packages/2.13/bioc/html/EBImage.html

Reading GIS data

- rdgal - http://cran.r-project.org/web/packages/rgdal/index.html
- rgeos - http://cran.r-project.org/web/packages/rgeos/index.html
- raster - http://cran.r-project.org/web/packages/raster/index.html

Reading music data

- tuneR - http://cran.r-project.org/web/packages/tuneR/
- seewave - http://rug.mnhn.fr/seewave/