

Lecture 10: Databases in the Wild

This Cat Earns \$150 Per Hour Working at Home as the World's Grumpiest Data Scientist...



[<http://worldofwonder.net/grumpy-cat-lands-broadway-role-else-cats/>]

Introduction to Social Media

- Most of you likely know far more about social media than I do. But, just in case you don't...
- Social media is a technology that allows people to broadcast their own content online and view the content of others.
- It builds interconnected networks of content creators and consumers.

Almost As Big As It Gets

- More people use social media sites than e-mail, online shopping, or really anything online.
- People spend more time with social media than anything else online.
- Outside of watching television and, well, eating dinner, it's one of the most pervasive activities in the world.

Data Are Everywhere

- Social Media is also quite possibly the world's most pervasive data collection scheme.
- Everything innovative about social media is in how they made reporting data seem cool.
 1. *Connections and followers*
 2. *Speaking in #hashtags.*
 3. *Feedback buttons.*
- All of your activities on a social media site are recorded, collected, aggregated, analyzed, scrutinized, and ultimately sold.

Privacy On Social Media is a Myth

- The site may or may not have privacy settings that allow you to control who sees your page.
- However, in one way or another, the technology companies who run the sites are all selling access to the information on your page. There are some rare exceptions.
- In spite of this, many people freely share information that they might later wish they had kept private.

What Social Media Companies Want

- They want to have a complete profile of information on everyone.
- They want to turn that marketing information into opportunities to make money.
- Most of all, they want you to use the site so that they can show you advertisements or otherwise sell things to you.

Understand the Business to Understand the Designs

- Why are social media sites set up the way they are?
- What you see on the page is the result of decisions made based on a large degree of experimentation and data analysis.
- We can't really understand these choices without understanding the factors that drive the business of social media.

How We'll Examine Social Media

1. Understand the kind of data that they produce.
2. Investigating the activities, behaviors, and economic value of social media users.
3. Deriving some general insights about the customers and goals for the business.
4. Evaluating the design of these sites – and their inherent means of collecting data – in the context of fulfilling these goals.

Projects with a Real Social Media Company

- Working with a real social media company, I had an opportunity to help them understand the patterns and behaviors of their users.
- I helped to examine the effectiveness of advertising campaigns, marketing research on which customers were most likely to pay for subscription services.
- I also helped the engineering team to evaluate new features and determine whether these designs helped the company reach its goals.

About the Site

- A medium-sized social media platform.
- Earns money through a combination of advertising revenue and add-on subscription services.
- Millions of active users worldwide, including a sizable portion of subscribers.

Technologically Sophisticated

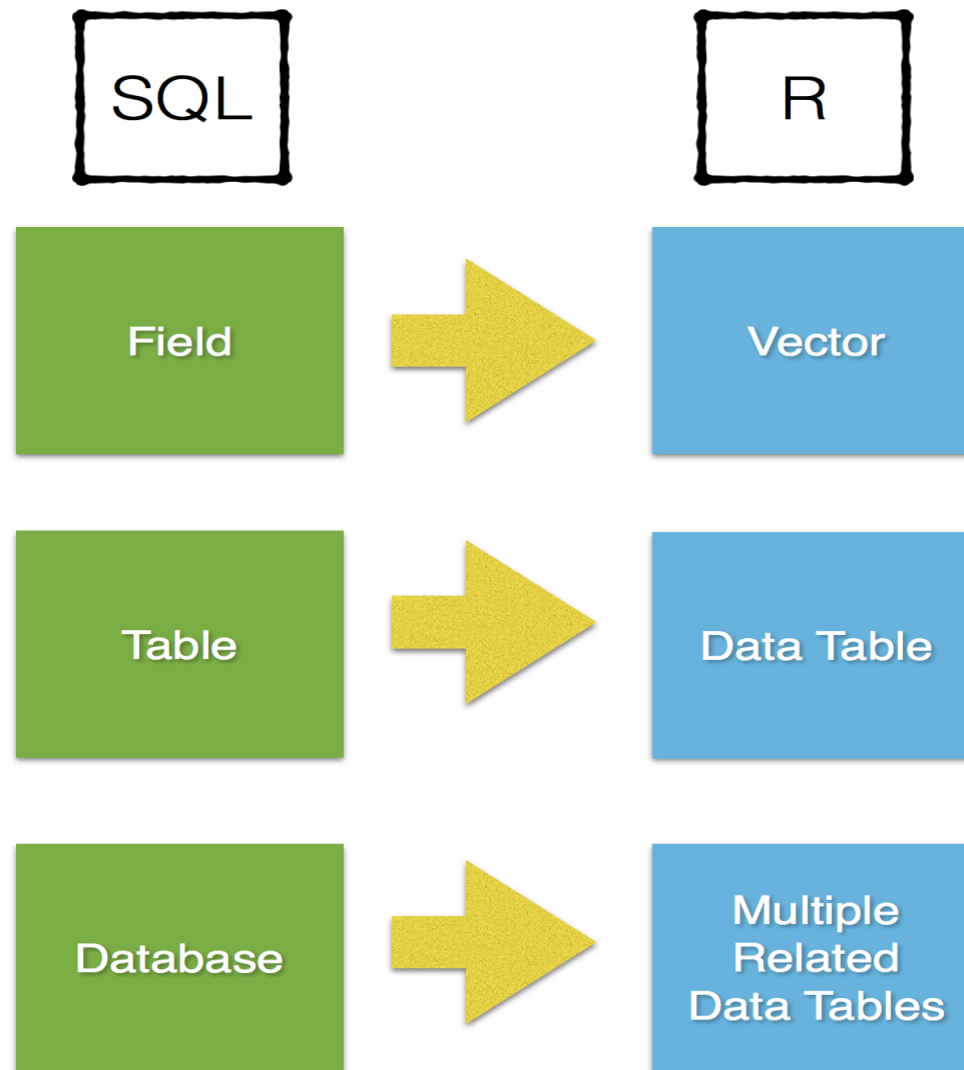
- Advanced engineering practices were used in every facet of the company.
- All data were securely stored in a database on a cloud-based platform.
- They were well positioned to make changes to their designs or strategy based on new information.

A Security Protocol

Accessing the data required a number of security measures:

- My computer's **hardware address** and home's **IP address** were registered in their systems. I could only access their systems from my machine, either in their office or at home.
- A secure Socket Shell (SSH) connection was required.
- Then a username and password were also requested to log in.

A Brief Introduction to Databases



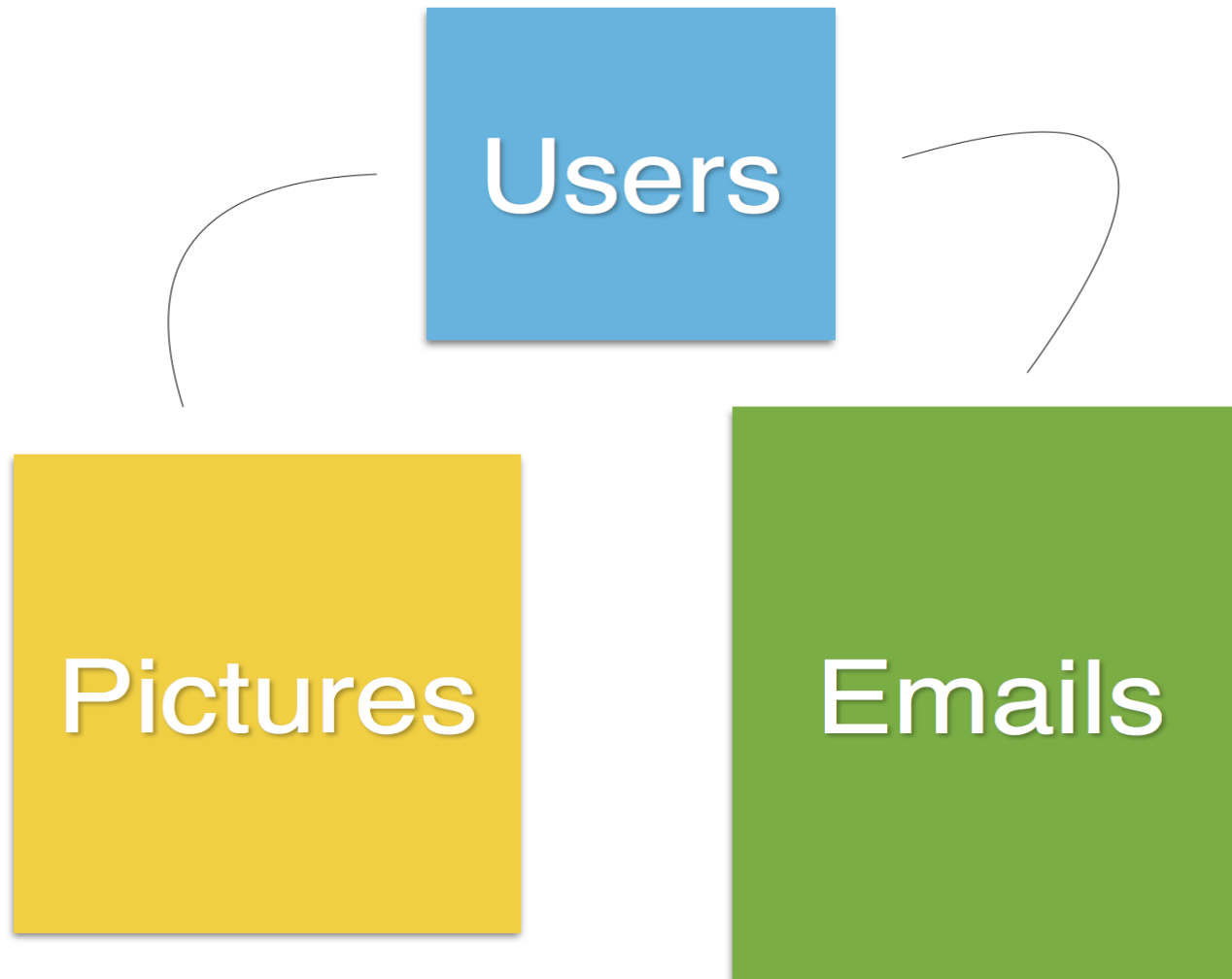
Most databases are built in some variation of Structured Query Language (SQL). In an SQL database, there is no such thing as a standalone variable. Every piece is

a table, which is akin to a `data.table` object in R.

A Relational Database

- The database is considered **relational** if multiple tables match together in some way. There is no requirement that a table must link to other tables.
- Each table should record unique information.
- Identifiers allow you bring the pieces together.

Example: Social Media Collections



The Users Table

This is a randomly generated collection of fictional social media users.

```
users <- fread(input = "Users.csv")
datatable(data = users[1:100, ], rownames = FALSE)
```

Show entries

Search:

id	Name	Age	State
xLlvq2PSdnn5osHg	Whipple, Alanna	32	FL
tMYkHda0MmRU2Gmu	Wilk, Brian	28	IL
yCmVrkukrY9qU0Wb	el-Amin, Siddeeqa	19	LA
gG7r9lqZBIxykjls	Baker, Skylar	64	MN
3z6yjCbwcyl3qBXK	Muya, Drake	40	WI
phVWQxOAK2QxYisz	Lim, Zong	29	KS
robaIseIGwsTD8IK	Phan, Sinh	36	TX
0yadyddUqsOjqUu2	Greene, Josie	70	WV

id	Name	Age	State
obU0szrugP588L70	Thomas, Garrett	23	CA
jny8tSWpCgaq5WHS	Rasmussen, Cory	50	NY

Showing 1 to 10 of 100 entries

Ambiguities Everywhere

- **Name:** This may or may not be the user's real name. In some settings, it's quite important to have it. In other cases, it doesn't matter.
- **Age:** Some users lie about their age. The site's limitations (e.g. age restrictions) may encourage lying.
- **State:** This could be **self-reported** or it may be based off of other information, e.g. based on the **IP address** of the user's most recent login.

The Pictures Table

This is a table recording all of these users' pictures.

```
pics <- fread(input = "Pictures.csv")
pics[, .N]
```

```
[1] 1724000
```

```
datatable(data = pics[1:100, ], rownames = FALSE)
```

Show entries

Search:

id	Owner	Date_Uploaded	File_Name	Deleted	Flagged
gSjAALvOQJMH	00dwK0vMEQImQmhP	2017-09-01T07:39:01Z	Pic_dKmt4yysOP.jpg	false	false
e9EtskrjSrTD	00dwK0vMEQImQmhP	2017-09-01T19:39:29Z	Pic_QNABtfO6oo.jpg	false	false
osZickQhGzfx	00dwK0vMEQImQmhP	2017-09-01T23:50:47Z	Pic_BXKcv4Eldj.jpg	false	false
lEnX9paBl t9B	00dwK0vMEQImQmhP	2017-09-02T06:36:22Z	Pic_8GEOJpd9FL.jpg	false	false

id	Owner	Date_Uploaded	File_Name	Deleted	Flagged
90rii47dzMHI	00dwK0vMEQImQmhP	2017-09-02T10:22:01Z	Pic_y8vzqd6qO7.jpg	false	false
hqjqdAbNAs6a	00dwK0vMEQImQmhP	2017-09-02T17:49:49Z	Pic_VWIBgFSG24.jpg	false	false
XPsnhlqPxcbl	00dwK0vMEQImQmhP	2017-09-03T00:00:32Z	Pic_UTre0F5ymP.jpg	false	false
IYI0JHUNH4A6	00dwK0vMEQImQmhP	2017-09-03T03:23:53Z	Pic_wIGDLb4MuM.jpg	false	false
9nKKb4Cltijl	00dwK0vMEQImQmhP	2017-09-03T09:07:56Z	Pic_bLJcdiUqK8.jpg	false	false
3zXl3KIjlgTK	00dwK0vMEQImQmhP	2017-09-03T09:36:49Z	Pic_zRfMFdYDc0.jpg	true	false
Showing 1 to 10 of 100 entries					
<div> Previous 1 2 3 4 5 ... 10 Next </div>					

Fields of the Pictures Table

- **id:** This is a unique identifier for the picture.
- **Owner:** This is the identifier for the owner of the picture. The value here matches up to a value of the **id** field in the **Users** table.
- **Date_Uploaded:** This is the time that the user uploaded the photo to the site.
- **File_Name:** The name of the picture's file, which is stored on the site's servers. When triggered by a click, the site uses this table to find and display the picture.
- **Deleted:** This indicates whether the user has deleted the photo or not. Notice that the photo **still exists in the company's database** but is no longer displayed on the user's page.
- **Flagged:** This indicates whether a picture was flagged for potentially violating the terms of the service.

The Emails Table

```
emails <- fread(input = "Emails.csv")
emails[, .N]
```

```
[1] 1280905
```

```
datatable(data = emails[1:100, ], rownames = FALSE)
```

Show entries

Search:

id	Sender	Receiver	Date_Sent	File
0L75ojy8hgNMuQKnLVZvgI ZA	00dwK0vMEQImQmhP	0ftMbD2Ny9roXRC6	2017-10-01T14:28:03Z	Message_nUPkzRW
IVzxjgQenwNAdcx8CtqLIU6w	00dwK0vMEQImQmhP	0hMFYgnhjB3pGXvB	2017-09-16T04:54:55Z	Message_aTrxKY0C
L4B9qFFCtImv444tdCV0vtmH	00dwK0vMEQImQmhP	0xC5jT8qv76oBzU	2017-09-03T07:11:26Z	Message_tjjL6Gq53z
LtNjpxomxSuj9ChxP62fGF9z	00dwK0vMEQImQmhP	2InIwNzZNcDEoK7I	2017-09-11T14:21:08Z	Message_ki9VIBsr17

id	Sender	Receiver	Date_Sent	File
J5o3OxYo2DFGgGWLVEDSAxBI	00dwK0vMEQImQmhP	4OfcIuUaTrCZcdbE	2017-09-15T09:49:57Z	Message_ZkIsXq0v
f3Uu8GaKsWbcDdJRiluIcBIG	00dwK0vMEQImQmhP	4OfcIuUaTrCZcdbE	2017-09-15T22:14:45Z	Message_dDgEZMy
JTmWVqPNLTtSFqusZrLjgvzh	00dwK0vMEQImQmhP	4OfcIuUaTrCZcdbE	2017-09-16T16:59:33Z	Message_3OXko3c8
xPEqTh7SimQ3Nkoy45AT5FjO	00dwK0vMEQImQmhP	4OfcIuUaTrCZcdbE	2017-09-16T17:20:28Z	Message_hLUdEyCz
a3QDaein8nQFsztLyhHgINQu	00dwK0vMEQImQmhP	62zkGg9Bt6rk08R5	2017-09-16T08:24:27Z	Message_sV3QGkln
FIzrQ6yTnYs72KWmFqIJJD5YB	00dwK0vMEQImQmhP	62zkGg9Bt6rk08R5	2017-09-17T06:09:10Z	Message_UYxgMAu
Showing 1 to 10 of 100 entries				
<div> Previous 1 2 3 4 5 ... 10 Next </div>				

Fields of the Emails Table

- **id:** This is a unique identifier for the message.
- **Sender:** This is the identifier for the sender of the message. The value here matches up to a value of **id** the **Users** table.
- **Receiver:** This is the identifier for the receiver of the message. The value here matches up to a value of **id** the **Users** table.
- **Date_Sent:** This is the date/time that the user Sender sent the message.
- **File_Name:** This is the name of the text file that stores the contents of the message on the site's servers.
- **Sender_Deleted:** TRUE/FALSE.
- **Receiver_Deleted:** TRUE/FALSE.

Notice that the email still exists in the company's database but is no longer displayed in the user's messages if it has been deleted.

Privacy Considerations

- Some countries have strong protections for the ownership and privacy of a user's data.
- In those regions, a user may demand that a company fully remove *all of the user's data** from its systems.
- In the case of an email, the message would have to be deleted **both from the sender's account** and **from the receiver's account** in order to fully comply.

The Time Frames

```
min.and.max <- function(x, na.rm = TRUE) {  
  require(data.table)  
  return(data.table(Min = min(x, na.rm = na.rm), Max = max(x,  
    na.rm = na.rm)))  
}  
pic.date.name <- "Date_Uploaded"  
pics[, min.and.max(get(pic.date.name))]
```

	Min	Max
1:	2017-09-01T04:00:00Z	2017-10-02T03:59:59Z

```
email.date.name <- "Date_Sent"  
emails[, min.and.max(get(email.date.name))]
```

	Min	Max
1:	2017-09-01T04:00:02Z	2017-10-04T01:50:38Z

All of the pictures and messages were added to the site in a time frame encompassing September and early October of 2017.

Linkages: Pics and Emails to Users

```
owner.name <- "Owner"  
user.id.name <- "id"  
pics[, mean(get(owner.name) %in% users[, get(user.id.name)])]
```

```
[1] 1
```

```
sender.name <- "Sender"  
receiver.name <- "Receiver"  
emails[, mean(get(sender.name) %in% users[, get(user.id.name)] &  
  get(receiver.name) %in% users[, get(user.id.name)])]
```

```
[1] 1
```

All of the pictures were owned by those in the **users** table, and all of the emails were between pairs in the **users** table.

Linkages: Users to Pics and Emails

```
users[, mean(get(user.id.name) %in% pics[, get(owner.name)])]
```

```
[1] 1
```

```
users[, mean(get(user.id.name) %in% emails[, get(sender.name)])]
```

```
[1] 0.9998
```

```
users[, mean(get(user.id.name) %in% emails[, get(receiver.name)])]
```

```
[1] 1
```

Every user in the cohort uploaded at least one picture and received at least one email. A few users did not send any emails among the messages recorded here.

This May or May Not Be Complete Information

- We don't necessarily know **why these users were identified**, which other **users were excluded**, or whether some of these users **emailed other users** who were not part of the cohort.
- Furthermore, these users may only represent a **sample** from the broader cohort.
- A total of **10000 users** uploaded **1.724 million pictures** and sent at least **1.280905 million emails** in about 5 weeks.
- We have a lot of data.

Getting the Data

- Large databases can store more information than we can reasonably work with.
- Some systems have interfaces that allow you to pull the information that you need.
- In other cases, it may be necessary for you to learn some amount of database programming (e.g. in **SQL** or one of its variants) to query what you need.

How Much SQL, and How Much R?

There is no magic formula for deciding on how to extract the data and analyze it. Your choices will depend on:

- The requirements of the setting;
- Your own abilities in using multiple languages;
- What makes the most technological sense. You could also employ a variety of other languages as needed.

My Approach to Data Extraction

- I used an SQL viewing program to understand the contents of the database. Everything that we've seen so far was visible through that program. I was not completely in the dark about the contents of the tables.
- I wrote the SQL queries needed to extract the data.
- Then I used R to run the query, extract the data, and perform all of my analyses.

This may or may not be the best approach, but it was one that provided a reasonable bridge between the structure of the database and my analyses in R.

Creating a Secure Connection

- I ran an SSH script on the command line of my computer:

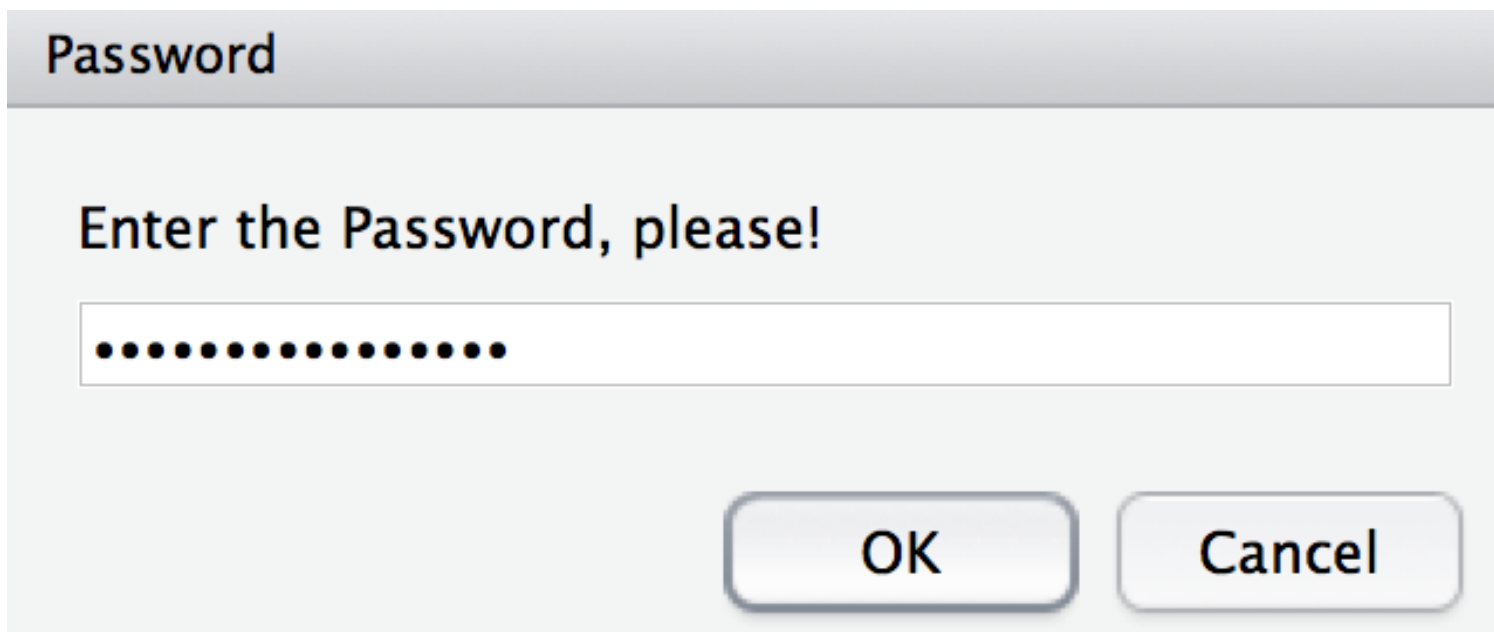
```
ssh -L 8080:the_complicated_cloud_domain_name  
my_laptop@vpn2.the_company_website.com -p 32722
```

When you run this kind of command, the server receives the information about the machine's name, hardware address, and IP address. The network's administrator has to register your account. If your information matches an account in the registry, then the secure connection is established. However, if I had tried to log in from a cafe, then the IP address would not match, and my access would be denied.

Entering Passwords in R

You can set up your program to allow a user to enter a password to proceed:

```
library(getPass)  
password <- getPass(msg = "Enter the Password, please!")
```



The image shows a standard R password prompt window. The title bar at the top is labeled 'Password'. Below the title bar, the text 'Enter the Password, please!' is displayed. Underneath the text is a rectangular input field containing 15 black dots, indicating that the password is being entered in a masked format. At the bottom right of the window, there are two buttons: 'OK' and 'Cancel'.

Connecting to the Database

```
require("RPostgreSQL")
drv <- dbDriver(drvName = "PostgreSQL")
on.exit(dbDisconnect(conn = conn))

library(getPass)
username <- getPass(msg = "Enter the Username, please!")
password <- getPass(msg = "Enter the Password, please!")

conn <- dbConnect(drv = drv, dbname = "the_social_media_database",
  host = "localhost", port = 8080, user = username, password = password)
rm(list = c("username", "password"))
```

This was connecting to a Postgre SQL database. Of course, there are many other varieties that may require a different library, driver, and connection protocol.

In practice, this is something that the organization's engineering team should help you with as you get started. Each setting is likely to have its own protocols.

Running SQL Through R

Connect to the Database

```
conn = dbConnect(drv = drv, dbname = "the_social_media_database", host =  
  'localhost', port = 8080, user = username, password = password)
```

Write a Query

```
sql.query.users = "SELECT * FROM USERS LIMIT 1000"
```

Extract the Data

```
Users = dbGetQuery(conn = conn, statement = sql.query.users)
```


An Extraction Function

```
extract.data.from.AWS <- function(query, n = NA, user = NA,
  pw = NA, returnTime = FALSE, dbname = "dbname=the_database sslmode=require",
  host = "localhost", port = 8090) {
  require(RPostgreSQL)
  require(data.table)
  drv <- dbDriver(drvName = "PostgreSQL")
  on.exit(dbDisconnect(conn = con))

  if (is.na(user)) {
    require(getPass)
    user <- getPass(msg = "Enter the username, please!")
  }
  if (is.na(pw)) {
    require(getPass)
    pw <- getPass(msg = "Enter the Password, please!")
  }
  con <- dbConnect(drv = drv, dbname = dbname, host = host,
    port = port, user = user, password = pw)
  rm(pw)
  toc <- Sys.time()
  if (!is.na(n)) {
    query <- sprintf("%s LIMIT %d", query, n)
  }
  dat <- dbGetQuery(conn = con, statement = query)
  tic <- Sys.time()

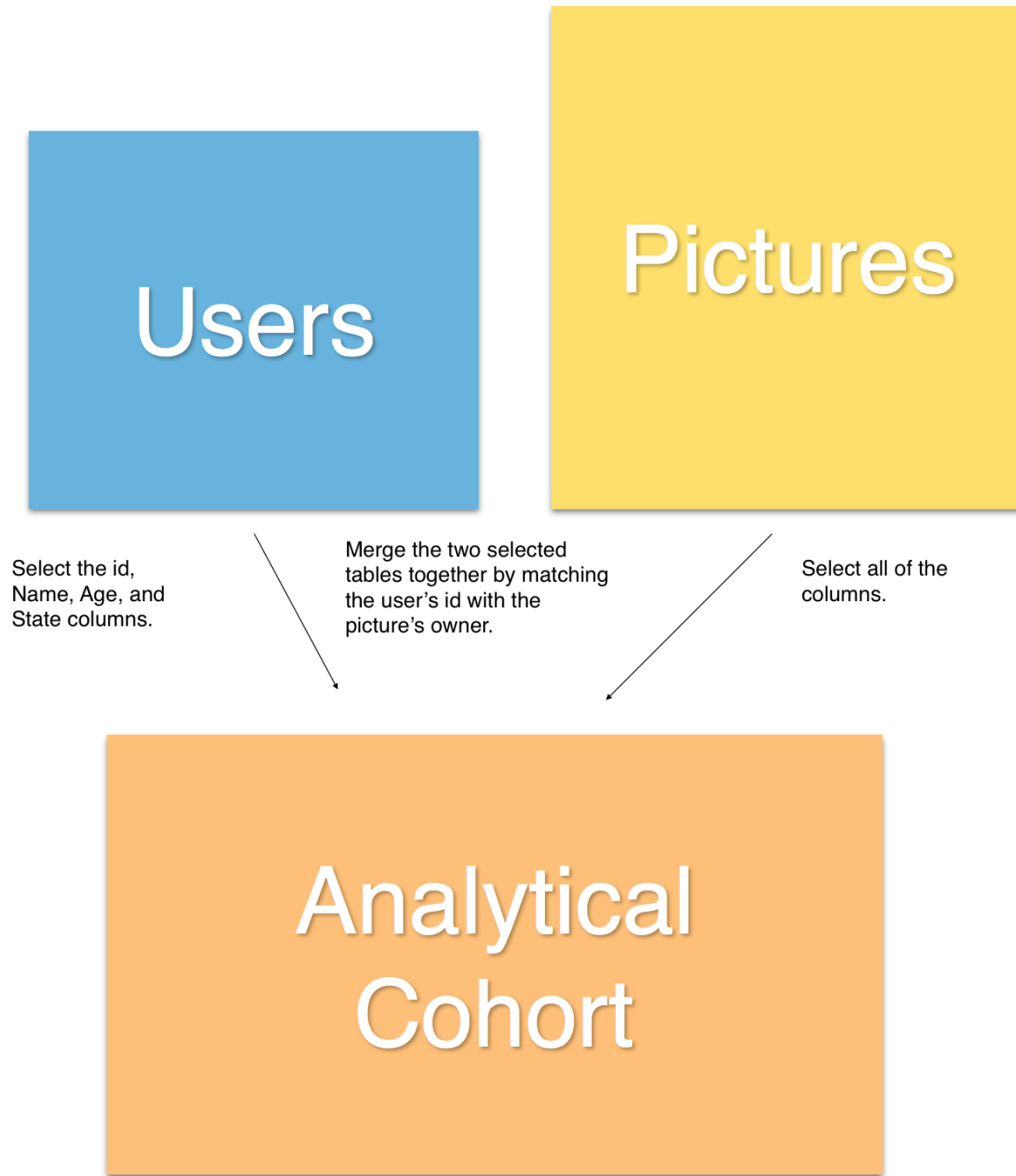
  if (returnTime == TRUE) {
    print(sprintf("Extracting %d items required %.0f seconds",
      nrow(dat), difftime(time1 = tic, time2 = toc,
        units = "secs")))
  }
  dat <- setDT(dat)
  return(dat)
}
```


High Level Overview of the Extraction

- The inputs provide the **location** and **credentials** to **connect to the online database**.
- The user also must **supply an SQL query** that will be used to extract the information.
- If **n** is supplied, then the query will be augmented to **only return n rows** of data; otherwise the full table is provided.
- A **data.table** object is returned.

We will **not be directly using this** extraction function or working with SQL in our class, but I have found it to be quite useful working with online databases. By generalizing the connection process, I can use this function to make any query. Then the bulk of my time can be spent focusing on the real problems – how to get the right table – instead of how to properly connect to the database.

Merging Information from Multiple Tables



Different Types of Merges

- **Inner Join:** Selects only the rows that have matches in both tables (users with pictures).
- **Left Join:** Includes all of the rows from the first table, whether or not a match exists in the second (users with or without pictures).
- **Right Join:** Includes all of the rows from the second table, whether or not a match exists in the first (pictures with or without users).
- **Full Join:** Includes all rows of both tables whether they have a match or not.

Option 2: Combining Tables in R

Any type of SQL Join can be performed in R using the **merge** command:

```
users_plus_pictures <- merge(x = users, y = pics, by.x = user.id.name,
                             by.y = owner.name, all.x = TRUE, all.y = TRUE)
datatable(data = users_plus_pictures[1:5], rownames = FALSE)
```

Show entries

Search:

id	Name	Age	State	id.y	Date_Uploaded	File_Name
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	gSjAALvOQJMH	2017-09-01T07:39:01Z	Pic_dKmt4yysOP.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	e9EtskrjSrTD	2017-09-01T19:39:29Z	Pic_QNABtfO6oo.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	osZickQhGzfx	2017-09-01T23:50:47Z	Pic_BXKcv4Eldj.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	lEnX9paBl t9B	2017-09-02T06:36:22Z	Pic_8GEOJpd9FL.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	90rii47dzMHI	2017-09-02T10:22:01Z	Pic_y8vzqd6qO7.jpg

Showing 1 to 5 of 5 entries

Previous



Next

Merging Options

- **x, y**: two `data.table` objects (or data frames).
- **by.x, by.y**: character vectors telling which columns in `x` to match to columns in `y`. Including multiple values will mean that the rows in `x` and `y` must match in all of the **by** columns.
- **all.x, all.y**: Will the outputs include all of the rows in `x` (TRUE) or only those with a match to `y` (FALSE), and likewise for `y`.

The combinations of **all.x** and **all.y** correspond to the different types of SQL Join operations.

Note: When **x** and **y** are **data.table** objects, then the **merge** function reverts to the **merge.data.table** function, which will perform the operation more efficiently.

Matching Join Types to Merging Types

- **Inner Join:** all.x and all.y are both FALSE.
- **Left Join:** all.x is TRUE, all.y is FALSE.
- **Right Join:** all.x is FALSE, all.y is TRUE.
- **Full Join:** all.x and all.y are both TRUE.

Multiple ID Columns

There is another problem with the result of the merge we performed. The resulting table had two different **id** columns.

Show entries

Search:

id	Name	Age	State	id.y	Date_Uploaded	File_Name
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	gSjAALvOQJMH	2017-09-01T07:39:01Z	Pic_dKmt4yysOP.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	e9EtskrjSrTD	2017-09-01T19:39:29Z	Pic_QNABtfO6oo.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	osZickQhGzfx	2017-09-01T23:50:47Z	Pic_BXKcv4Eldj.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	lEnX9paBl t9B	2017-09-02T06:36:22Z	Pic_8GEOJpd9FL.jpg
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	90rii47dzMHI	2017-09-02T10:22:01Z	Pic_y8vzqd6qO7.jpg

Showing 1 to 5 of 5 entries

Previous

1

Next

- The first **id** column is from x: the users' identifiers.
- The second **id** column is from y: the pictures' identifiers.
- It's not ideal to create two columns with the same name.

Unambiguous ID Columns

```

pic.id.name <- "pics_id"
setnames(x = pics, old = "id", new = pic.id.name)
age.name <- "Age"
state.name <- "State"

users_plus_pictures <- merge(x = users[, .SD, .SDcols = c(user.id.name,
  age.name, state.name)], y = pics, by.x = user.id.name,
  by.y = owner.name, all.x = TRUE, all.y = TRUE)
datatable(data = users_plus_pictures[1:5], rownames = FALSE)

```

Show entries

Search:

id	Age	State	pics_id	Date_Uploaded	File_Name	Deleted
00dwK0vMEQImQmhP	60	LA	gSjAALvOQJMH	2017-09-01T07:39:01Z	Pic_dKmt4yysOP.jpg	false
00dwK0vMEQImQmhP	60	LA	e9EtskrjSrTD	2017-09-01T19:39:29Z	Pic_QNABtfO6oo.jpg	false
00dwK0vMEQImQmhP	60	LA	osZickQhGzfx	2017-09-01T23:50:47Z	Pic_BXKcv4Eldj.jpg	false
00dwK0vMEQImQmhP	60	LA	lEnX9paBl t9B	2017-09-02T06:36:22Z	Pic_8GEOJpd9FL.jpg	false

id	Age	State	pics_id	Date_Uploaded	File_Name	Deleted
00dwK0vMEQImQmhP	60	LA	90rii47dzMHI	2017-09-02T10:22:01Z	Pic_y8vzqd6qO7.jpg	false

Showing 1 to 5 of 5 entries

Previous

1

Next

Databases: Fun with Counting

- So much of what you can do with databases involves **counting items by categories**.
- Most of the counts are either on the **number of records** or the **number of users** based on some qualifying criteria.

Counting Records

- Provide a summary of the number of pictures uploaded per user during the period of observation.

```
pic.counts.by.user <- users_plus_pictures[, .N, keyby = user.id.name]  
pic.counts.by.user[, summary(N)]
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
121.0	163.0	172.0	172.4	181.0	238.0

Daily Uploads

- How many **pictures** were uploaded by users **under 35 years old** in **California** on a **daily basis**?

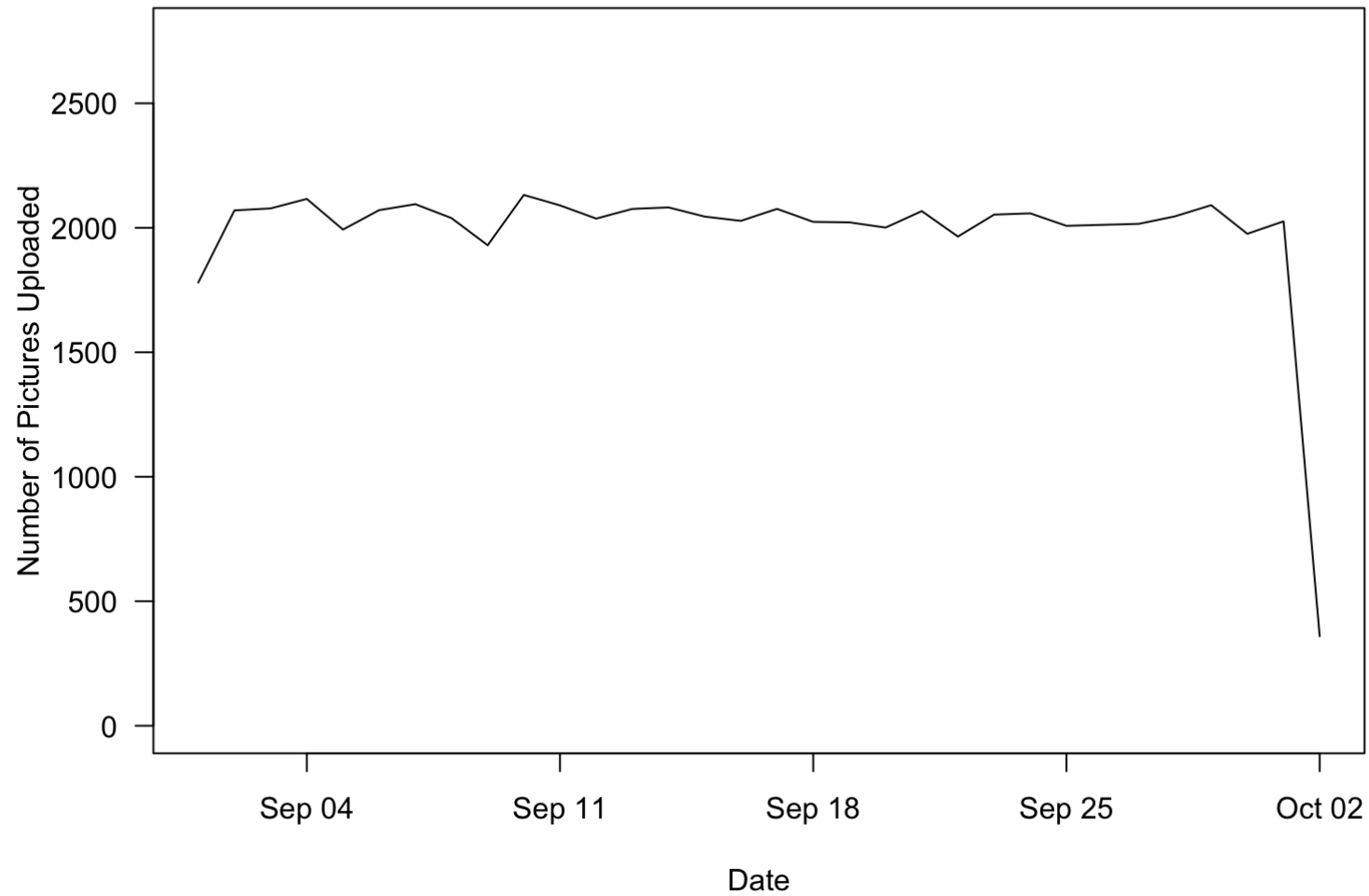
```
pic.date.name <- "Date_Uploaded"
calendar.date.name <- "Calendar Date"
users_plus_pictures[, eval(calendar.date.name) := as.Date(get(pic.date.name))]
```

```
the.counts <- users_plus_pictures[get(age.name) < 35 & get(state.name) == "CA", .N, keyby = calendar.date.name]
```

Plotting Daily Uploads

```
plot(as.formula(sprintf("N ~ `%s`", calendar.date.name)),  
     data = the.counts, type = "l", xlab = "Date", ylab = "Number of Pictures Uploaded",  
     ylim = c(0, 1.3 * max(the.counts[, N])), cex = 0.8,  
     las = 1, main = "Daily Uploaded Pics in CA under 35")
```

Daily Uploaded Pics in CA under 35



And What Happened on October 2nd?

- October 2nd was the last date included in the **pics** table. Let's take a look at the time of the last upload:

```
pics[, max(get(pic.date.name))]
```

```
[1] "2017-10-02T03:59:59Z"
```

- Because only about 4 hours of the day had passed when the data were downloaded, the number of pictures uploaded on October 2nd was considerably smaller.
- This is **not a point of alarm** – instead it's an artifact of how the system processed information in the middle of the night.
- Similarly, **monthly counts** may often see a similar drop. For this reason, it is important to **set a stopping date** on your calculations.

Daily Active Users

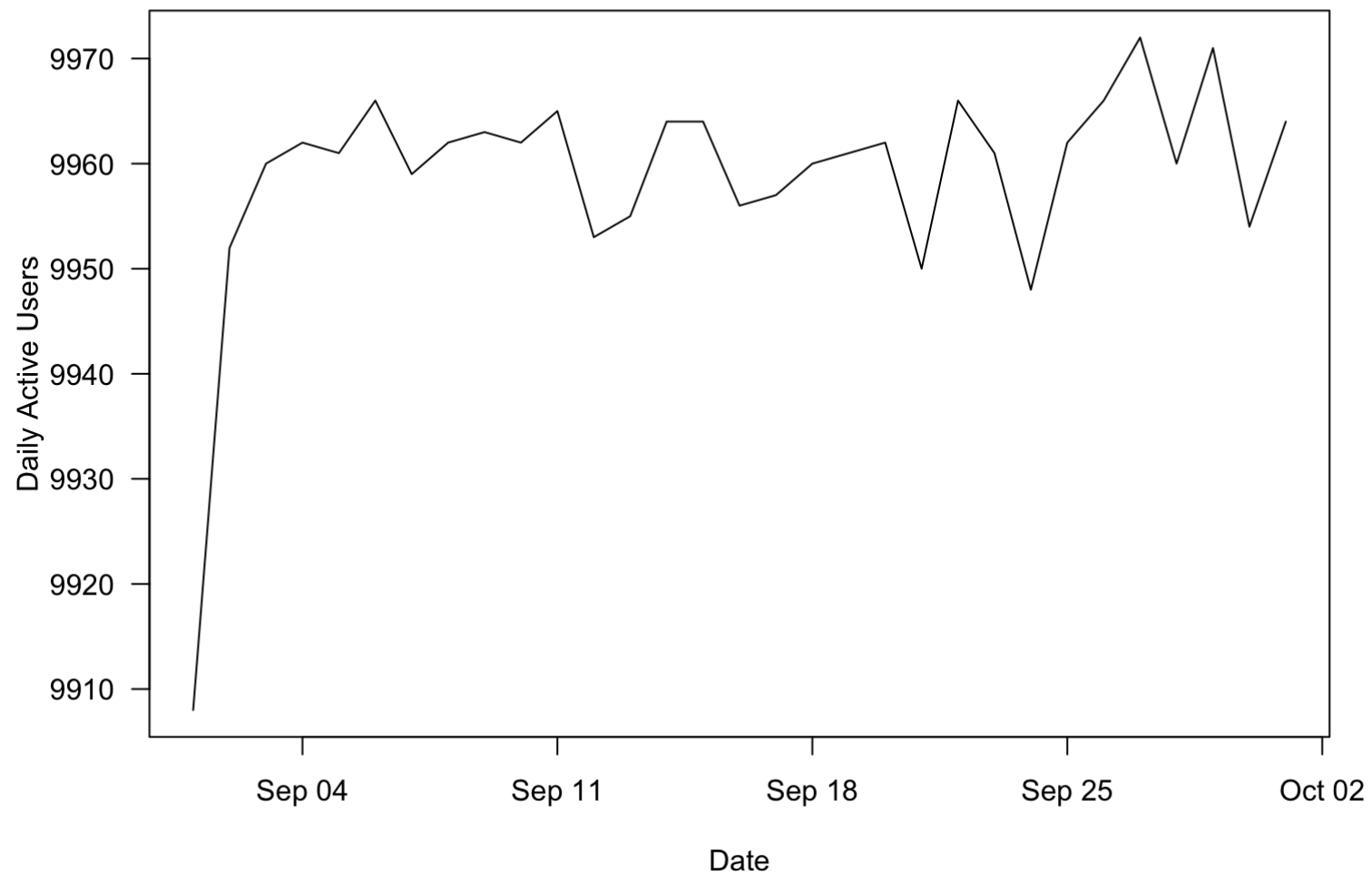
- How many of these users uploaded at least one picture? Show the number of daily active users.

```
daily.active.users <- users_plus_pictures[get(calendar.date.name) <
  max(get(calendar.date.name)), .(`Daily Active Users` = length(unique(get(user.id.name)))),
  keyby = calendar.date.name]
```


Plotting Daily Active Users

```
plot(as.formula(sprintf("`Daily Active Users` ~ `%s`", calendar.date.name)),  
     data = daily.active.users, type = "l", xlab = "Date",  
     ylab = "Daily Active Users", cex = 0.8, las = 1, main = "DAU: Uploaded Pictures")
```

DAU: Uploaded Pictures



Counting Unique Values

- Database tables can include enormous amounts of data.
- Using the most efficient techniques can really pay off in terms of the performance in these settings.
- Compared to **length(unique(...))**, the following function can be significantly faster for large data sets:

```
length.unique <- function(dat, value.name, by = NA) {  
  require(data.table)  
  this.dat <- setDT(x = dat)  
  if (length(by) > 0) {  
    if (!is.na(by[1])) {  
      if (by[1] == "") {  
        by[1] <- NA  
      }  
    }  
  }  
  if (is.na(by[1])) {  
    dat.u <- unique(this.dat, by = value.name)  
    ans <- dat.u[, .(N = .N)]  
  }  
  if (!is.na(by[1])) {  
    dat.u <- unique(this.dat, by = c(by, value.name))  
    ans <- dat.u[, .N, keyby = by]  
  }  
  return(ans)  
}
```


Day Active Users by Age Group

- For uploading pictures, how many **daily active users** were there in each state and in each age group (18-25, 25-35, 35-45, 45-65, 65+)?

```
age.group.name <- "Age Group"
library(Hmisc)
users_plus_pictures[, eval(age.group.name) := cut2(x = get(age.name), cuts = c(18, 25, 35, 45, 65, 120))]
dau.by.state <- length.unique(dat = users_plus_pictures, value.name = user.id.name, by = c(state.name,
  age.group.name, calendar.date.name))
datatable(data = dau.by.state[get(state.name) == "NY",][1:100,], rownames = FALSE)
```

Show entries

Search:

State	Age Group	Calendar Date	N
NY	[18, 25)	2017-09-01	82
NY	[18, 25)	2017-09-02	82
NY	[18, 25)	2017-09-03	82
NY	[18, 25)	2017-09-04	82
NY	[18, 25)	2017-09-05	82
NY	[18, 25)	2017-09-06	82

State	Age Group	Calendar Date	N
NY	[18, 25)	2017-09-07	82
NY	[18, 25)	2017-09-08	82
NY	[18, 25)	2017-09-09	82
NY	[18, 25)	2017-09-10	82

Counting Sent Emails

How many emails did the typical user send? Count the number for each user. Show a summary of the distribution. Keep in mind that some users may have sent zero emails (and would not appear as senders in the Emails table).

Summarizing the Counts of Sent Emails

```
email.count <- function(emails, by, id.name){  
  email.count.by.sender <- emails[, .(Num.Emails = .N), by = by]  
  
  the.emails.count <- merge(x = users[, .(id)], y = email.count.by.sender[, .(id = get(by), Num.Emails)], by =  
    id.name, all.x = TRUE, all.y = FALSE)  
  
  the.emails.count[is.na(Num.Emails), Num.Emails := 0]  
  return(the.emails.count)  
}  
email.id.name <- "id"  
sent.emails <- email.count(emails = emails, by = sender.name, id.name = email.id.name)  
summary(sent.emails[, Num.Emails])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0	87.0	127.0	128.1	168.0	289.0

Counting Received Emails

How many emails did the typical user receive? Count the number for each user. Show a summary of the distribution. Keep in mind that some users may have received zero emails (and would not appear as receivers in the Emails table).

Summarizing the Counts of Received Emails

```
received.emails <- email.count(emails = emails, by = receiver.name,  
                                id.name = email.id.name)  
summary(received.emails[, Num.Emails])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
45.0	102.0	127.0	128.1	153.0	244.0

Assessing Influence

Most people send about as many e-mails as they receive. One measure of a person's influence is how much attention they receive. Those who receive many more messages than they send are in demand. They are potentially influential. For each user, what is the difference between the number of messages received and the number sent? Show a summary of this difference as a measure of the user's influence.

Summarizing Influence

```
users.plus.sent <- merge(x = users[, .SD, .SDcols = user.id.name],  
  y = sent.emails[, .(id, Num.Emails.Sent = Num.Emails)],  
  by = user.id.name)  
users.email.counts <- merge(x = users.plus.sent, y = received.emails[,  
  .(id, Num.Emails.Received = Num.Emails)], by = user.id.name)
```

```
users.email.counts[, Received.Minus.Sent := Num.Emails.Received - Num.Emails.Sent]  
summary(users.email.counts[, Received.Minus.Sent])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-92	-18	0	0	19	83

Do Influential Email Recipients Post More Pictures?

What is the correlation between **receiving more emails than you send** and **posting more pictures**?

Here is a plan to answer this question:

- **Count how many pictures** each user posted. We already computed this with **pic.counts.by.user**.
- **Merge the tables** to line up the **counts of the uploads** with the **Received.Minus.Sent** score.
- Compute the **correlation**.

Merging the Counts and the Received_Minus_Sent

```
setnames(x = pic.counts.by.user, old = "N", new = "Num.Pics.Uploaded")
dat <- merge(x = pic.counts.by.user, y = users.email.counts,
            by = user.id.name, all = TRUE)

dat[, cor(x = Num.Pics.Uploaded, y = Received.Minus.Sent,
         use = "pairwise.complete.obs")]
```

```
[1] 0.00309792
```

In this case, simply uploading a lot of pictures does not mean that you will start to receive more messages than you send.

Problems with Relational Databases on the Cloud

1. New data and records are **constantly arriving**.
2. The information you want to connect **may not be directly related**. You may have to perform multiple JOIN operations to get what you're looking for.
3. Every **dynamic extraction requires downloading the results**, which can be time consuming.

We will discuss all of these problems in greater detail.

Issue #1: The Influx of New Data

- Any analysis you perform **will not include all of the data** because new records are arriving all of the time.
- New records will **affect your calculations**, making it difficult to verify the reliability of your results.
- Some **values may change over time**. A key finding about an example user would be interesting to report. However, it would be embarrassing if the special case no longer existed because the user's profile was changed at a later date.

Design Choices

- Some fields in a database can **record the history** of a user's changes.
- However, many fields **only save the current state**.
- Information may be changed for any number of reasons: the **user's preference**, from **quality assurance efforts**, or **changes from other users**, etc.

In short, a database always provides **a record of the current state of the data** in that moment. However, you may or may not have access to a full historical record of the changes that took place.

An Example: Early Usage Analyses

- How do the user's **responses in their profile** (e.g. **About Me**) impact their likelihood of **connecting with at least 100 other users** within 6 months?
- We could collect data on the **users' profiles** and the **number of connections** within 6 months.
- However, if the user **changed the profile** at a later point, we might not have accurate information to look back upon.
- If a specific set of information is important enough, then the engineering team can **record snapshots** of a user's profile to ensure that the original information can be retrieved.

Reproducibility in Databases

- Any analysis may be subject to **changing data**, which greatly **detracts from the potential** for reproducible research and reporting.
- In some cases, you may be able to **save a copy of the data** that you used for your report.
- However, **security protocols** may prevent you from keeping any records outside of the database.
- In some situations, the **best you can do** is to say **when and how you accessed the information**.

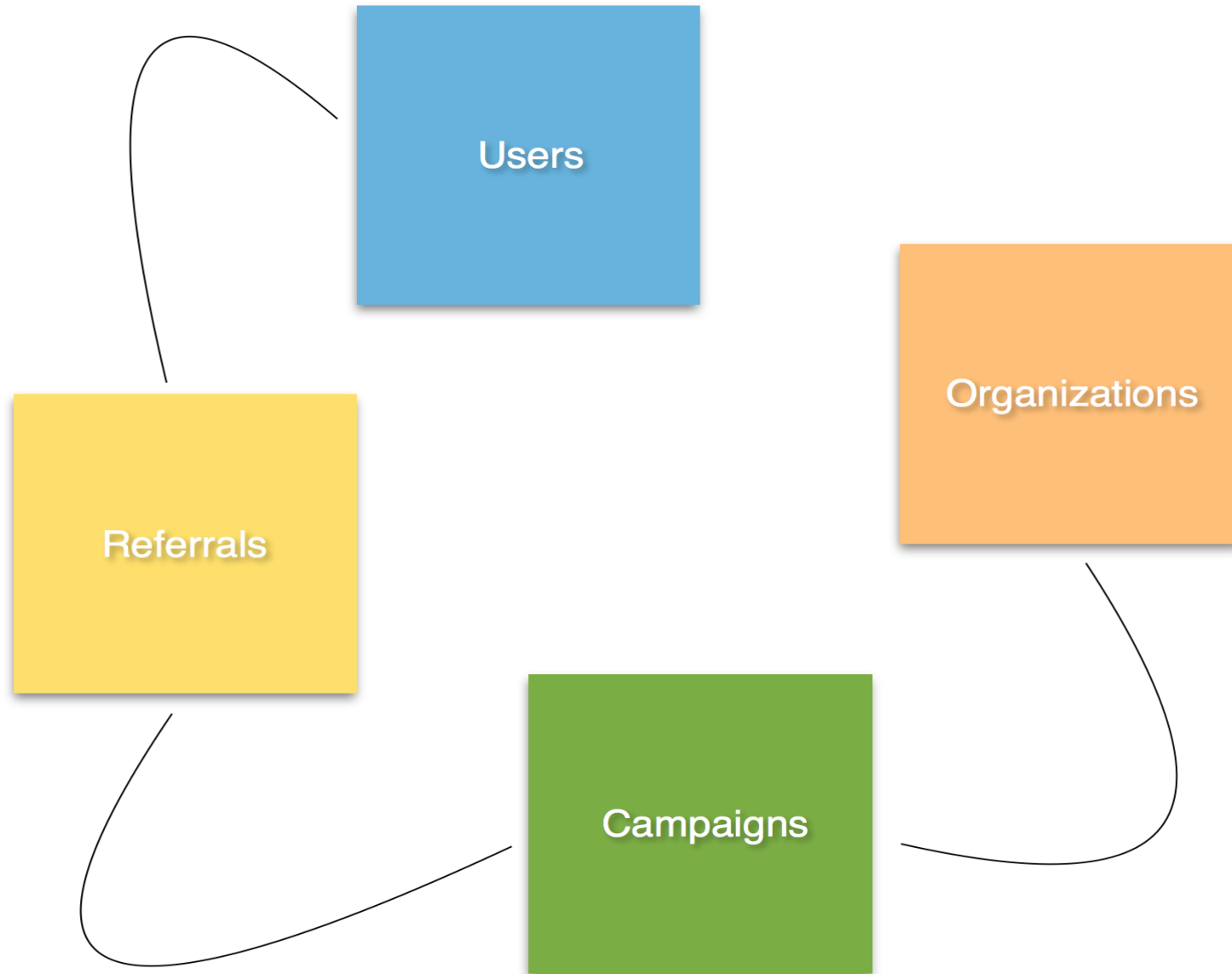
Issue #2: Indirectly Connected Information

- Not all tables are linked together.
- Some tables are only **indirectly** linked.
- Combining information from indirectly linked tables requires merging all of the intermediate tables together.

Example: Indirectly Linked Tables

The site has grown its community by advertising on other sites. How many new users have come from each site? It seems like a simple question.

Linking Organizations to Users



Perhaps this is not such a simple question after all.

The Referrals Table

```
referrals <- fread(input = "Referrals.csv")
datatable(data = referrals[1:100], rownames = FALSE)
```

Show entriesSearch:

User_id	Date	Campaign_id
xLlvq2PSdnn5osHg	2017-08-14T16:35:37Z	Campaign_39
tMYkHda0MmRU2Gmu	2017-09-29T09:21:05Z	Campaign_22
yCmVrkukrY9qU0Wb	2017-09-05T14:00:14Z	
gG7r9lqZBlXykjls	2017-08-09T10:12:37Z	
3z6yjCbwcyl3qBXK	2017-09-21T17:28:11Z	
phVWQxOAK2QxYisz	2017-09-28T04:29:47Z	
robaIseIGwsTD8IK	2017-09-24T16:58:52Z	Campaign_8
0yadyddUqsOjqUu2	2017-09-01T19:16:07Z	
obU0szrugP588L70	2017-09-19T05:29:35Z	
jny8tSWpCgaq5WHS	2017-09-10T05:37:15Z	

Showing 1 to 10 of 100 entries

Previous



2

3

4

5

...

10

Next

The Campaigns Table

```
campaigns <- fread(input = "Campaigns.csv")
datatable(data = campaigns[1:100, ])
```

Show entriesSearch:

	Campaign_id	Organization_id	Beginning_Date	Ending_Date
1	Campaign_1	pFuW7OYV	2017-06-17T07:32:16Z	2017-11-01T06:41:19Z
2	Campaign_2	LZV8GEwg	2017-07-25T04:09:58Z	2017-10-16T22:09:44Z
3	Campaign_3	NsiQvltb	2017-06-03T13:39:59Z	2017-11-01T14:31:17Z
4	Campaign_4	l4jPyUiQ	2017-06-14T00:56:55Z	2017-11-25T23:28:49Z
5	Campaign_5	UjwWgyPS	2017-06-28T06:28:52Z	2017-11-23T15:18:38Z
6	Campaign_6	77eepoPD	2017-07-26T00:02:43Z	2017-11-16T02:00:03Z
7	Campaign_7	lGT0L6N8	2017-06-16T01:27:41Z	2017-11-22T07:56:44Z
8	Campaign_8	sGufdX2l	2017-06-24T01:17:27Z	2017-11-12T00:22:09Z
9	Campaign_9	lGT0L6N8	2017-06-21T06:23:49Z	2017-10-06T08:26:22Z
10	Campaign_10	UjnlG5IE	2017-07-29T17:40:29Z	2017-10-29T07:23:45Z

Showing 1 to 10 of 100 entries

Previous



2

3

4

5

...

10

Next

The Organizations Table

```
organizations <- fread(input = "Organizations.csv")
datatable(data = organizations, rownames = FALSE)
```

Show entriesSearch:

id	Name
I4jPyUiQ	Organization_Name_1
UjwWgyPS	Organization_Name_2
IGT0L6N8	Organization_Name_3
IUYXcb9X	Organization_Name_4
4d2IhltT	Organization_Name_5
IkvDfywu	Organization_Name_6
H6Twef5U	Organization_Name_7
ifRDmhQ2	Organization_Name_8
77eopoPD	Organization_Name_9
SCKpqGFe	Organization_Name_10

Showing 1 to 10 of 20 entries

Previous



2

Next

Connecting Campaigns to Organizations

```
campaigns_plus_orgs_all <- merge(x = campaigns, y = organizations,  
  by.x = "Organization_id", by.y = "id", all.x = TRUE,  
  all.y = FALSE)  
campaigns_plus_orgs <- campaigns_plus_orgs_all[, .(Campaign_id,  
  organization = Name)]  
datatable(data = campaigns_plus_orgs, rownames = FALSE)
```

Show 10  entriesSearch:

Campaign_id	organization
Campaign_7	Organization_Name_3
Campaign_9	Organization_Name_3
Campaign_44	Organization_Name_3
Campaign_52	Organization_Name_3
Campaign_61	Organization_Name_3
Campaign_67	Organization_Name_3
Campaign_72	Organization_Name_3

Campaign_id	organization
Campaign_79	Organization_Name_3
Campaign_15	Organization_Name_6
Campaign_38	Organization_Name_6

Showing 1 to 10 of 100 entries

Previous

1

2

3

4

5

...

10

Next

Connecting Referrals to Organizations

```
referrals_plus_orgs_all <- merge(x = referrals, y = campaigns_plus_orgs,  
  by = "Campaign_id")  
referrals_plus_orgs <- referrals_plus_orgs_all[, .(User_id,  
  organization)]  
datatable(data = referrals_plus_orgs)
```

Show entriesSearch:

	User_id	organization
1	Wq2iEvaFcTQoH73r	Organization_Name_I9
2	U0pfiwK0pHp2r803	Organization_Name_I9
3	ExbJAv4HLTyZrRL5	Organization_Name_I9
4	pO3aUnLL2W3iu40N	Organization_Name_I9
5	9VCilWVHQ6VI9sHi	Organization_Name_I9
6	bR7M5ttjKoYOE9WO	Organization_Name_I9
7	0LxhVXzoIv9zVk6O	Organization_Name_I9
8	ag7GLZqvqQOBa5Xv	Organization_Name_I9

User_id		organization	
9	Ruus2fcQzeA5s0v4	Organization_Name_I9	
10	i5jBrmyGUJlkbzFx	Organization_Name_I9	

Showing 1 to 10 of 4,959 entries

Previous

1

2

3

4

5

...

496

Next

Connecting Users to Organizations

```
users_plus_orgs <- merge(x = users, y = referrals_plus_orgs,
  by.x = "id", by.y = "User_id")
datatable(data = users_plus_orgs, rownames = FALSE)
```

Show entries

Search:

id	Name	Age	State	organization
00dwK0vMEQImQmhP	Marquez, Magda	60	LA	Organization_Name_10
01ewzQlhIPwyFZCa	Behr, Joshua	69	WV	Organization_Name_8
03l8zDbLlx2mR7An	al-Azzi, Awda	48	OH	Organization_Name_1
03VyeGjzUq8sBgtL	Thompson, Soryn	42	WV	Organization_Name_2
03d5m7A8XeQc9PX7	Castro-Diaz, Izac	30	IL	Organization_Name_2
062u0l9wfVJVMl46	el-Noor, Mufeeda	22	NY	Organization_Name_8
06EeZwAftRxe30lZ	Barnett, Christina	20	TX	Organization_Name_5
07l9gfxniEaI lxXK	Lucero, Alyssa	62	CA	Organization_Name_17
07lAilmUtaGc5IBA	Mo, Hoang	39	VA	Organization_Name_5

id	Name	Age	State	organization
07qMbeRgMP0uzKHy	Johnson, Shazia	69	IL	Organization_Name_15

Showing 1 to 10 of 4,959 entries

Previous

1

2

3

4

5

...

496

Next

An Indirect Path

- Connecting users to the organizations that referred them required a **circuitous route** through a number of other tables.
- Meanwhile, we had to create a variety of intermediate tables along the way.
- It's worth examining whether we can be more direct in making these connections.

Option I: Nested Merges

We could do all of the merges in one very complicated step:

```
users_plus_orgs_2 <- merge(x = users, y = merge(x = referrals,  
  y = merge(x = campaigns, y = organizations, by.x = "Organization_id",  
    by.y = "id", all.x = TRUE, all.y = FALSE)[, .(Campaign_id,  
      organization = Name)], by = "Campaign_id")[, .(User_id,  
        organization)], by.x = "id", by.y = "User_id")  
setnames(x = users_plus_orgs_2, old = "id", new = "User_id")
```

Option 2: multi.merge

```
multi.merge <- function(objects, by = NULL, by.x = NULL,
  by.y = NULL, all = FALSE, all.x = all, all.y = all,
  sort = TRUE, suffixes = c(".x", ".y"), no.dups = TRUE,
  allow.cartesian = getOption("datatable.allow.cartesian"),
  ...) {
  require(data.table)
  num.objects <- length(objects)
  if (!is.null(by)) {
    by.x <- by
    by.y <- by
  }
  if (length(by.x) == 1) {
    by.x <- rep.int(x = by.x, times = num.objects -
      1)
  }
  if (length(by.y) == 1) {
    by.y <- rep.int(x = by.y, times = num.objects -
      1)
  }
  if (length(all.x) == 1) {
    all.x <- rep.int(x = all.x, times = num.objects -
      1)
  }
  if (length(all.y) == 1) {
    all.y <- rep.int(x = all.y, times = num.objects -
      1)
  }
  res <- setDT(objects[[1]])
  for (i in 2:num.objects) {
    res <- merge(x = res, y = setDT(objects[[i]]), by.x = by.x[i -
      1], by.y = by.y[i - 1], all.x = all.x[i - 1],
      all.y = all.y[i - 1], sort = sort, suffixes = suffixes,
      allow.cartesian = allow.cartesian, ...)
  }
}
```

```
    return(res)  
}
```

multi.merge Example

```
users_plus_orgs_3 <- multi.merge(objects = list(campaigns,  
  organizations[, .(id, organization = Name)], referrals,  
  users), by.x = c("Organization_id", "Campaign_id", "User_id"),  
  by.y = c("id", "Campaign_id", "id"), all.x = TRUE, all.y = FALSE)[,  
  .SD, .SDcols = names(users_plus_orgs_2)]  
  
mean(users_plus_orgs == users_plus_orgs_2)
```

```
[1] 1
```

```
mean(users_plus_orgs_2 == users_plus_orgs_3)
```

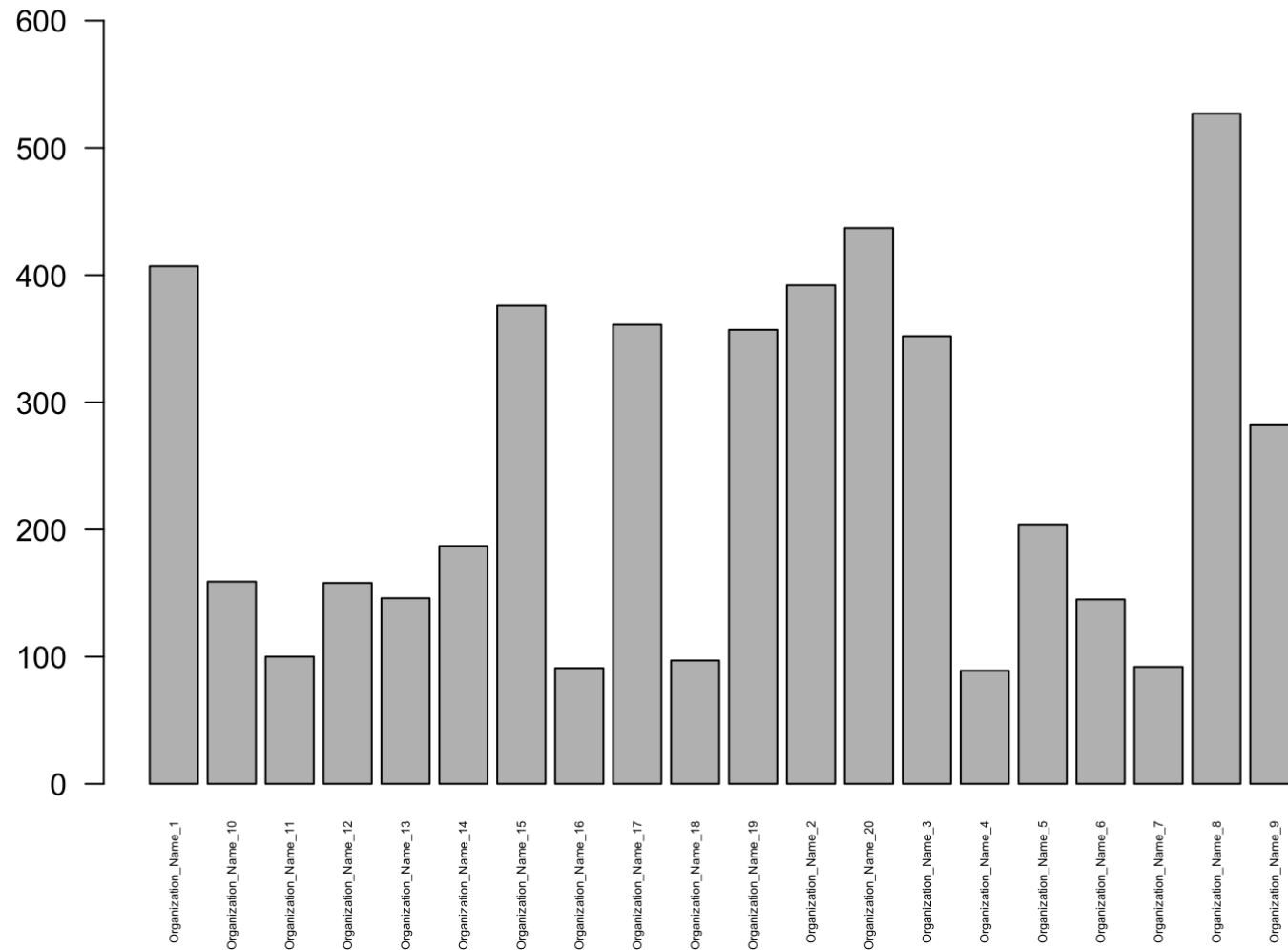
```
[1] 1
```

While not necessarily any faster, using **multi.merge** can significantly cut down on the work you have to do to structure nested merges.

Analytics on Referring Organizations

```
barplot.output <- users_plus_orgs[, barplot(table(organization),  
      las = 2, cex.names = 0.4, ylim = c(0, 600), main = "Referrals by Organization")]
```

Referrals by Organization



Characteristics of Users Referred by an Organization

```
users_plus_orgs[get(state.name) == "CA" & organization ==  
  "Organization_Name_1", .(Mean_Age_CA = round(x = mean(x = get(age.name),  
    na.rm = T), digits = 1))]
```

```
Mean_Age_CA  
1:         43.9
```

Answers Are Possible But Difficult to Generate

- Because of the **indirect links** between users and referring organizations, we had to do a lot of work to connect these two pieces.
- With K different tables, there are $K * (K - 1)/2$ **pairs of tables**. It may not be obvious how all of these pairs can be linked.
- Any simple, ad-hoc request can potentially **turn into a more involved investigation**.

A Disorienting Feeling

- It can be difficult enough to **fully understand one table** of data.
- Many databases might have **20-50 tables** that are routinely used.
- Understanding the **connections**, issues of quality, and **limitations** of a database can often require your full concentration.

Issue #3: Pitfalls of Cloud Computing

- Every extraction of data requires **processing plus downloading**.
- This can be **time consuming**: 15 minutes to an hour for a table that would load in seconds off of my local hard drive.
- Now imagine that you do download all of that data only to discover a **bug in your code**. Now you might have to download it all over again. This makes iterative analysis a time-consuming endeavor.

It is up to you to find a way to make your working environment practical and efficient. We'll discuss a few options for solving these problems.

Option #1: Periodic Downloads

- Periodic updates (daily, weekly, monthly, quarterly, etc.)
- De-identification.
- Security with passwords on the files, encrypted drives, etc.

Advantages to Periodic Downloads

- Speed of working with local files.
- Data cleaning and merging can be done as a pre-processing step.
- Greater consistency with using a single file for a period of time.

Disadvantages to Periodic Downloads

- The **institution's security is likely stronger** than your personal security measures.
- Your work **won't be in real time**. The most recent updates to the data will only be analyzed in the next period.
- Any **changes to the structure** of the database **won't be reflected** in your code. You may be developing analytical tools for a structure that has become obsolete.

In one example, I was developing some investigations while the data engineer was creating new variables to display in the tables. From time to time, the engineer would reorganize the data – changing the names of variables or moving information from one table to another. This would often happen without a process of informing me of the changes. Had I stuck with periodic downloads, I would have been developing my work on an obsolete structure.

Option #2: Working on the Cloud

- Periodic downloads bring the data from the cloud to your machine. Instead, you could **bring your machine to the cloud**.
- Where possible, an instance of R can be installed on the servers.
- Then, simply by logging in, you can program remotely much as you would on your own machine.

Advantages of Working on the Cloud

- You can work on the servers that store the data as if they're a local machine.
- The cloud's **processing capabilities** may exceed your system's power.
- You can most easily maintain all of the **security protocols**.

Disadvantages of Working on the Cloud

- **Not always possible:** many proprietary systems don't have a cloud processing environment for you to develop on.
- **Costly:** Processing hours on the cloud are vastly more expensive than using your own machine.
- Working on the cloud **does not eliminate the processing time** of merging tables in SQL. In some cases, this is the most costly step. However, it does eliminate the need for downloading the results of every query.

Option #3: Prototyping with Small Samples

- You can work with **limited samples of data** to speed up the development of your code.
- Then, you can run the analysis on a larger sample size when the program is ready to go.
- However, **beware of unanticipated cases!** The limited sample of data may not include every potential subgroup that your program needs to account for.

Making Do

- Security protocols may prevent you from storing data on your local hard drive.
- The computing environment may not be amenable to programming on the cloud.
- In this case, running the queries and only accessing data in RAM, with no saved copies, may be your only workable solution. In this case, you'll have to work within the constraints.

Option #4: Hire a Data Engineer

Hey, we've said all along that **data engineering** is **anything that a data scientist doesn't want to do** and **can persuade a data engineer to do** instead. If the databases are too challenging, it might impact the ability of your team to perform the intended analyses. Contributions from a data engineer who could handle some of these technical issues might free up the data scientists to derive insights for the organization. It's OK to ask for help, and it might be better for the project overall.

Periodic Updating Wins Out – When Possible

- Provided that it complies with your security protocols, periodic updates is the most practical solution to the challenges of working with relational databases.
- Much is gained by having one data set that is properly structured, organized, and well understood.
- Little is lost by not having the most recent updates. Most decisions are not so time sensitive.

But What About Real Time Systems?

- In extraordinary circumstances, creating real-time systems with the most updated information may be necessary.
- You may end up building real-time systems as part of a complex operation.
- However, by definition, real-time systems don't have much time for reflection. Even real-time systems require a high degree of investigation and knowledge. That usually starts with more sustained examinations of a fixed data set.

A Plan for Social Media Analytics

- Set a time period for updates (e.g. each month or quarter) depending on the requirements of the problem.
- Download all of the tables once per time period.
- Perform all of the necessary pre-processing work a single time: structuring, cleaning, quality checks, etc.
- Store the results in a secure local repository, preferably with password protections. If this is not possible, then revise your plans accordingly.
- Then perform all of your analysis on this version until the next update occurs.

Pre-Processing

- Once all of the tables have been extracted, perform any necessary data cleaning or structuring steps.
- Likewise, merge together any tables when connected information is required for the analyses.
- Do all of this work a single time per periodic update.

Saving the Image

- When pre-processing is complete, all of the tables will be loaded in memory.
- At this point, you can save the tables in a local drive. Your options include:
 1. Separate files for each table.
 2. A combined .RData file for the entire list of tables.

```
save(list = ls(), file = sprintf("Social_Media_Data_%s.RData",  
  Sys.Date()))
```

Using `list = ls()` saves all of the variables (equivalent to the **`save.image()`** function). But you can specify a smaller number of variables to be included in the list.

Maintaining Security

- This will depend on your technology and the requirements of your organization.
- Using **encrypted drives** is a great way to safeguard your files. The drive is password protected in your operating system. Once you've entered the password, you can read and write files as if they were unsecured.
- Likewise, individual files (e.g. spreadsheets) can be password protected. However, R's capabilities to read and write these files can be a bit limited. The **xlsx** package allows you to read a password protected spreadsheet.

An End Run Around Databases

- In some sense, databases are an excellent way to efficiently store information.
- Social media sites are constantly adding data to a wide variety of tables. There is no substitute for databases that interact with web designs so that information can be collected and displayed dynamically.
- In spite of this, databases are often structured in a way that makes analyses difficult. Where possible, it is better to use pre-processing to restructure the information in a manner that is more amenable to your work.

Now We are Set Up for Investigations

- In the coming lectures, we will explore the business of social media and how information can empower decisions.
- We will also investigate how the designs of social media pages are influenced by the goals of the underlying business.

And On and On

- Working with databases **is what it's really like** to be a data scientist.
- Your role is critical to **translating the information** into useful insights.
- However, **adapting to the structures** of each new database is a seemingly neverending challenge.

We will continue exploring these themes in the next lecture.