TareaNoSQL

Milena Motta

2023-10-30

Introducción

En esta práctica se verá como R puede ser utilizado para conectarnos a una base de datos NoSQL. En particular Mongo DB.Para la realización de la práctica se utilizará Atlas un servicio en la nube gratuito para manegar bases de datos Mongo DB. MongoDB Atlas es fácil de configurar y tiene conjuntos de datos de muestra para ejemplos de R con Mongo DB. Puede cargar conjuntos de datos de muestra usando el "..." junto al botón de colecciones en la página de su clúster. No obstante, aunque se puede crear un clúster específico en Atlas, esta práctica parte de uno ya creado y que puede ser utilizado para la resolución de los ejercicios.

- Adicionalmente, se recomienda utilizar un cliente Mongo DB para conectarse a la base de datos e inspeccionar los datos contenidos. Esto será extemadamente útil para realizar las consultas. Puede considerarse Robo 3T (https://robomongo.org/download). Además, si se crea el propio cluster en Atlas, este tiene una interfaz amigable para inspeccionar los datos.
- Además de estas funciones, existe documentación de las colecciones y la información contenida en esta base de datos de ejemplo en https://docs.atlas.mongodb.com/sample-data/sample-training/
- Para la resolución de ejercicios puede consultar el Manual de Mongo DB que contiene ejemplos y explicación de la sintáxis de Mongo DB

Paquetes R utilizados

El controlador R MongoDB preferido, mongolite, es rápido y tiene una sintaxis similar a la del shell MongoDB. Mongolite es la que se utilizará en los siguientes ejemplos. Los otros paquetes enumerados aquí no han estado tan activos en Github recientemente. Los paquetes más populares para conectar MongoDB y R son:

- **mongolite**: un controlador R MongoDB más reciente, mongolite puede realizar varias operaciones como indexación, canalizaciones de agregación, cifrado TLS y autenticación SASL, entre otras. Está basado en el paquete jsonlite para R y mongo-c-driver. Podemos instalar mongolite desde CRAN o desde RStudio (explicado en una sección posterior). RMongo: RMongo fue el primer controlador R MongoDB con una sencilla interfaz R MongoDB. Tiene una sintaxis como la del shell MongoDB. RMongo ha quedado obsoleto a partir de ahora.
- **rmongodb**: rmongodb tiene funciones para crear pipelines, manejar objetos BSON, etc. Su sintaxis es muy compleja en comparación con mongolite. Al igual que RMongo, rmongodb ha quedado obsoleto y no está disponible ni se mantiene en CRAN.

Instalación Mongolite

Para poder usar el paquete mongolite debemos instalarlo previamente con el comando siguiente, además de importar la librería posteriormente

```
library(mongolite)
```

Conectarse a Mongo DB

Esta es la cadena de conexión a MongoDB en Atlas. Se podría sustituir por otro servidor o clúster en Atlas si se desea.

```
cadena_conexion = 'mongodb+srv://user01:user01@cluster0.mcblc3z.mongodb.net/test'
```

Opciones de no validación de certificados SSL. Sin esta opción puede haber error de conexión a Atlas.

```
opciones_conexion = ssl_options(weak_cert_validation = F)
```

Después de establecer la conexión a MongoDB se recupera la colección 'trips' usando la función mongo() en código R para obtener la colección de viajes de la base de datos sample_training. Esta recopilación contiene datos de viajes realizados por los usuarios de un servicio de bicicletas compartidas con sede en la ciudad de Nueva York.

```
viajes = mongo(collection="trips", db="sample_training", url=cadena_conexion, options = opciones_conexi
```

Puede verificar que su código ahora esté conectado a la colección MongoDB verificando el número total de documentos en esta base de datos. Para hacerlo, use la función count().

```
viajes$count()
```

```
## [1] 10000
```

Ahora que tiene una conexión establecida con la base de datos, podrá leer los datos de la misma para ser procesados por R.

Cómo obtener datos en R desde MongoDB

En esta sección, aprenderá cómo recuperar datos de MongoDB y mostrar los mismos. Continuemos con trips_collection de la sección anterior.

Puede usar la interfaz de usuario de MongoDB Atlas para ver los documentos de trip_collection o RStudio para visualizarlos.

Obtenga cualquier documento de muestra de la colección usando el método *iterate*().one() para examinar la estructura de los datos de esta colección.

```
viajes$iterate()$one()
```

```
## $tripduration
## [1] 396
##
## $`start station id`
## [1] 317
##
## $`start station name`
## [1] "E 6 St & Avenue B"
##
## $`end station id`
## [1] 317
##
```

```
## $`end station name`
## [1] "E 6 St & Avenue B"
##
## $bikeid
## [1] 23082
##
## $usertype
## [1] "Subscriber"
##
## $`birth year`
## [1] 1981
##
## $`start station location`
## $`start station location`$type
## [1] "Point"
##
## $`start station location`$coordinates
## $`start station location`$coordinates[[1]]
## [1] -73.98185
##
## $`start station location`$coordinates[[2]]
## [1] 40.72454
##
##
##
## $`end station location`
## $`end station location`$type
## [1] "Point"
##
## $`end station location`$coordinates
## $`end station location`$coordinates[[1]]
## [1] -73.98185
##
## $`end station location`$coordinates[[2]]
## [1] 40.72454
##
##
##
## $`start time`
## [1] "2016-01-01 01:17:55 CET"
##
## $`stop time`
## [1] "2016-01-01 01:24:32 CET"
```

Ahora que conoce la estructura de los documentos, puede realizar consultas más avanzadas, como buscar los cinco viajes más largos a partir de los datos de recopilación de viajes. Y luego enumerar la duración en orden descendente.

```
viajes$find(sort = '{"tripduration" : -1}' , limit = 5)
```

```
##   tripduration start station id          start station name end station id
## 1       326222              391         Clark St & Henry St            310
## 2       279620             3165 Central Park West & W 72 St           3019
## 3       173357             3155     Lexington Ave & E 63 St           3083
## 4       152023              340     Madison St & Clinton St           2009
```

```
## 5       146099              368          Carmine St & 6 Ave            238
##           end station name bikeid   usertype birth year
## 1      State St & Smith St  18591 Subscriber       1979
## 2        NYCBS Depot - DEL  17547   Customer
## 3 Bushwick Ave & Powers St  15881   Customer
## 4 Catherine St & Monroe St  22678 Subscriber       1992
## 5  Bank St & Washington St  15553   Customer
##   start station location.type start station location.coordinates
## 1                       Point                -73.99345, 40.69760
## 2                       Point                -73.97621, 40.77579
## 3                       Point                -73.96649, 40.76440
## 4                       Point                -73.98776, 40.71269
## 5                       Point                -74.00215, 40.73039
##   end station location.type end station location.coordinates
## 1                     Point              -73.98913, 40.68927
## 2                     Point              -73.98193, 40.71663
## 3                     Point              -73.94100, 40.71248
## 4                     Point              -73.99683, 40.71117
## 5                     Point              -74.00859, 40.73620
##            start time           stop time
## 1 2016-01-01 01:58:20 2016-01-04 20:35:23
## 2 2016-01-02 17:07:26 2016-01-05 22:47:46
## 3 2016-01-02 15:25:36 2016-01-04 15:34:53
## 4 2016-01-02 00:31:59 2016-01-03 18:45:42
## 5 2016-01-01 02:59:07 2016-01-02 19:34:07
```

La consulta anterior utiliza operadores de clasificación y límite para producir este conjunto de resultados.

Cómo analizar datos de MongoDB en R

Para analizar MongoDB con R con más detalle, puede usar el marco de agregación de MongoDB. Este marco permite a los operadores crear canalizaciones de agregación que ayudan a obtener los datos exactos con una sola consulta.

Suponga que desea verificar cuántos suscriptores realizaron viajes de una duración > 240 segundos y regresaron a la misma estación donde comenzaron. La consulta usa MongoDB [$expr] (https://docs.mongodb.com/manual/reference/operator/query/expr/) para comparar dos campos en el mismo documento.

```
query = viajes$find('{"usertype":"Subscriber","tripduration":{"$gt":240},"$expr": {"$eq": ["$start stati
```

Combinando estos operadores con algún código R, también puede ver qué tipo de usuarios son más comunes: suscriptores o clientes únicos. Para ello, se puede agrupar usuarios por tipo de usuario campo.

```
tipos_usuario = viajes$aggregate('[{"$group":{"_id":"$usertype", "Count": {"$sum":1}}}]')

print(tipos_usuario)
```

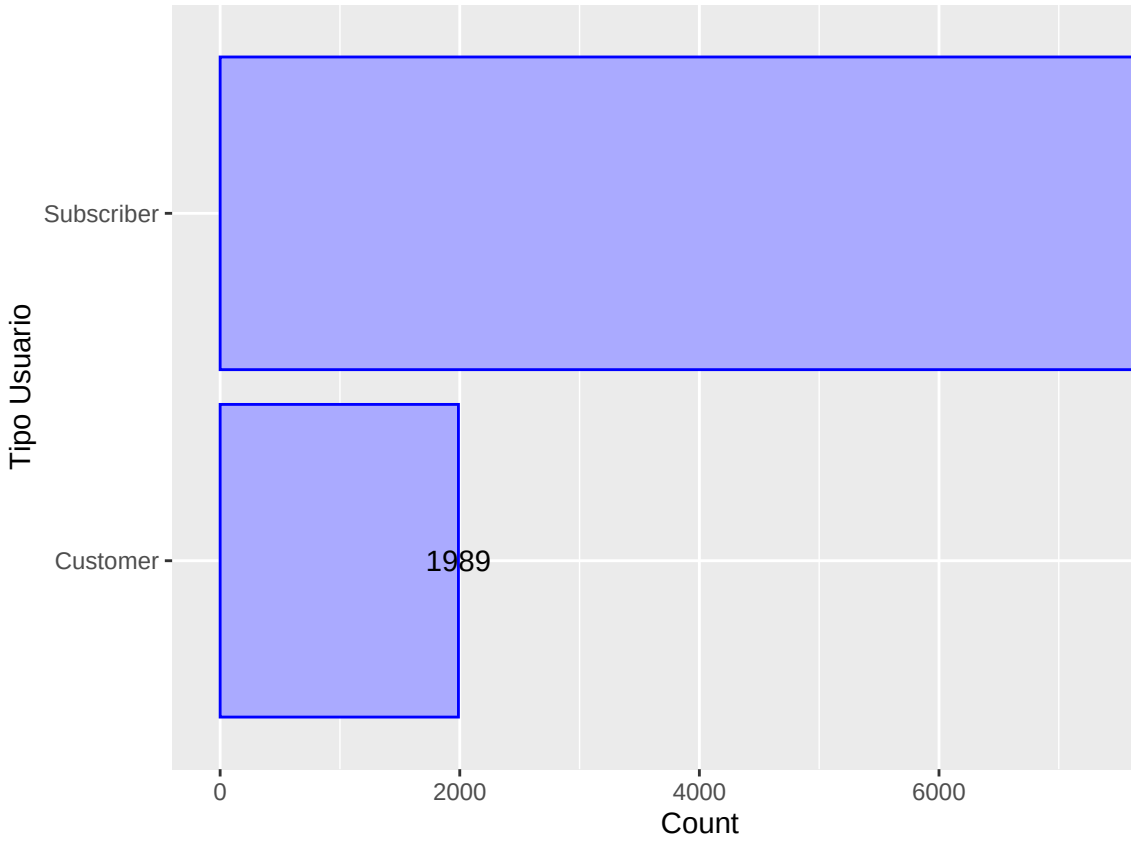
```
##          _id Count
## 1 Subscriber  8011
## 2   Customer  1989
```

Para comparar los resultados, puede visualizar los datos. Es conveniente convertir los datos obtenidos de mongolite en un marco de datos y usar ggplot2 para trazar.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(lubridate)
library(ggplot2)

df <- as.data.frame(tipos_usuario)

ggplot(df,aes(x=reorder(`_id`,Count),y=Count))+
geom_bar(stat="identity",color='blue',fill='#aaaaff')+geom_text(aes(label = Count), color = "black") +c
```



consulta suscriptores-1.pdf

viajes = mongo(collection="trips", db="sample_training", url=cadena_conexion, options = opciones_conexion)

Tarea a realizar

A continuación se proponen 4 ejercicios a completar. Se recomienda completaros en script markdown de R que se proporciona.

Tarea 1

Exploremos otro diagrama de barras con una colección diferente - inspections. Esta recopilación contiene datos sobre las inspecciones de edificios de la ciudad de Nueva York y si pasan o no. Recupere dicha colección en R.

```
library(mongolite)

# Recuperar la colección 'inspections' de la base de datos 'sample_training'
inspections_collection = mongo(collection="inspections", db="sample_training", url=cadena_conexion, opti



inspections_data <- inspections_collection$find

inspections_collection$count()
```

```
## [1] 80047
```

Tarea 2

Suponga que desea verificar el número de empresas que no aprobaron las inspecciones en 2015 en comparación con 2016.

Si ve los datos obtenidos de la colección, notará que el campo de fecha es una Cadena. Convertirlo en tipo de fecha y luego extraer el año requerirá algún procesamiento. Pero, con la canalización de agregación de Mongodb, puede hacer todo en una sola consulta. Para manipular el campo de fecha, use el operador $addFields.

Además, agregue las deficiencias encontradas en las inspecciones por año

```
library(mongolite)

# Conectar a la colección 'inspections'
inspections_collection = mongo(collection="inspections", db="sample_training", url=cadena_conexion, opti



inspections_collection$aggregate('[
  {"$addFields": {"format_year": {"$year": {"$toDate": "$date"}}}},
  {"$match": {"result": "Fail", "format_year": { "$in": [2015, 2016]}}},
  {"$group": { "_id": "$format_year", "count": { "$sum": 1 }}}
]')
```

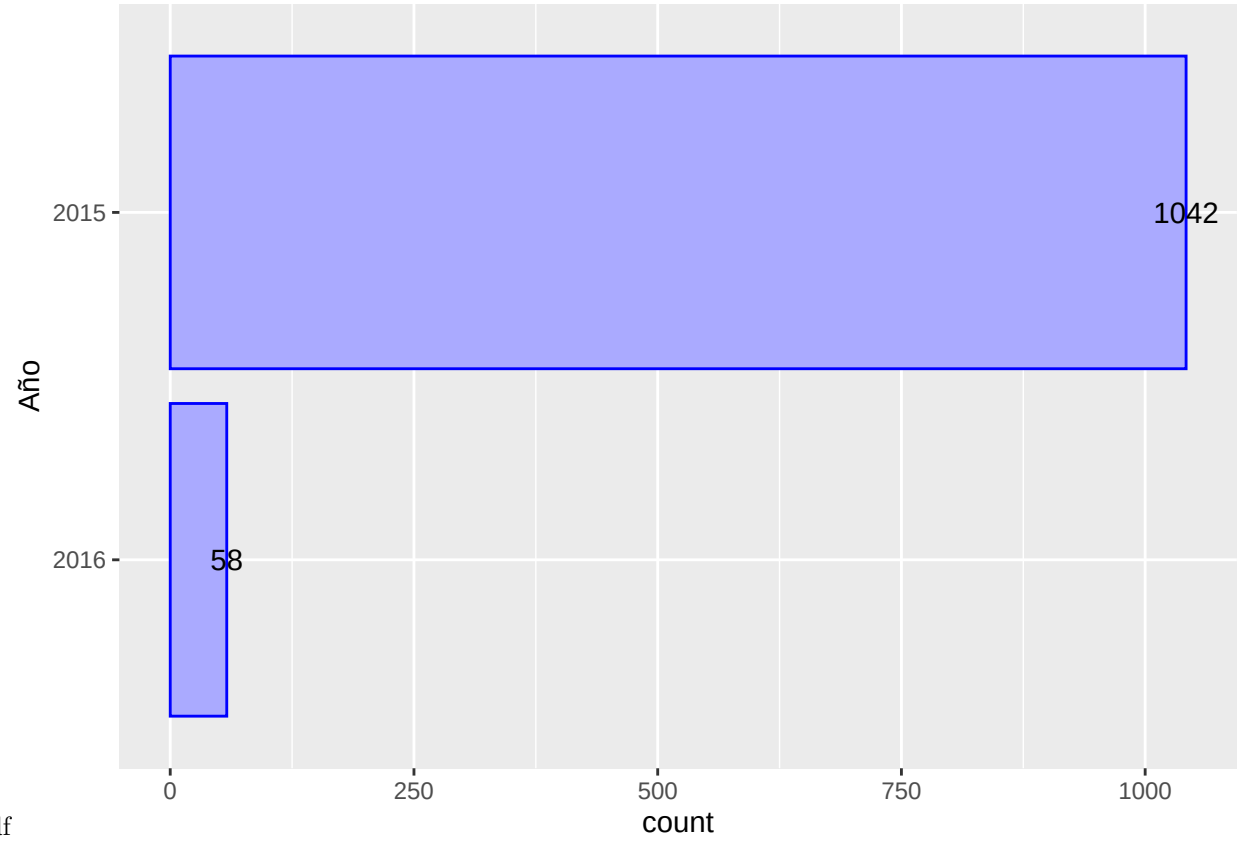
```
##    _id count
## 1 2015  1042
## 2 2016    58
```

Tarea 3

Teniendo en cuenta que el resultado de la tarea anterior está agrupando los resultados por año, cree un gráfico de barras.

```
library(mongolite)
library(tidyverse)
library(lubridate)
library(ggplot2)
```

```r
# Conectar a la colección 'inspections'
inspections_collection = mongo(collection="inspections", db="sample_training", url=cadena_conexion, opti


pipeline <- '[
  {"$addFields": {"format_year": {"$year": {"$toDate": "$date"}}}},
  {"$match": {"result": "Fail", "format_year": { "$in": [2015, 2016] } }},
  {"$group": { "_id": "$format_year", "count": { "$sum": 1 } }}
]'

result <- inspections_collection$aggregate(pipeline)

# Convertir el resultado en un marco de datos
result_df <- as.data.frame(result)

# Crear un gráfico de barras
df <- as.data.frame(result_df)

ggplot(df,aes(x=reorder(`_id`,count),y=count))+
geom_bar(stat="identity",color='blue',fill='#aaaaff')+geom_text(aes(label = count), color = "black") +c
```



3-1.pdf

Tarea 4

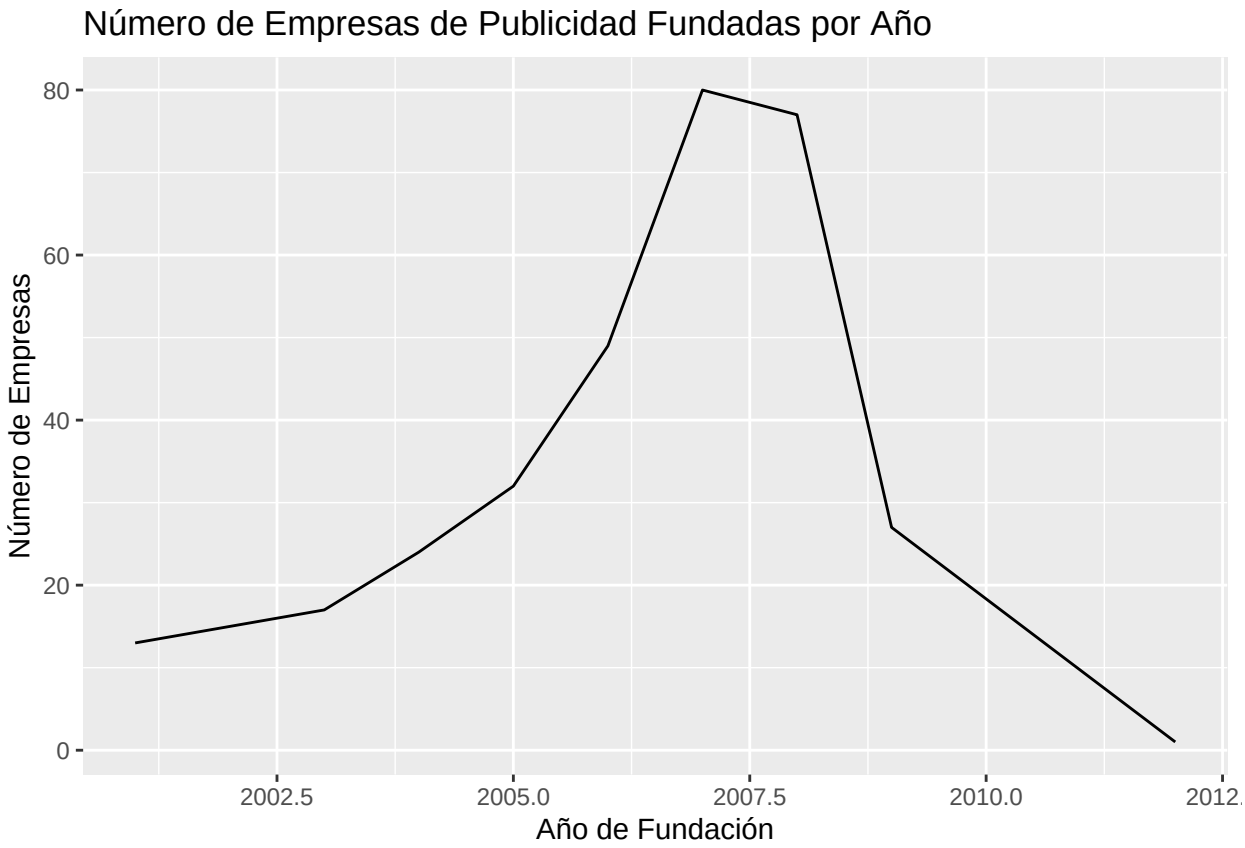
A continuación, se utilizará la colección 'companies', que contiene información sobre empresas, como su año de fundación y la dirección de su sede.

Supongamos que desea conocer la tendencia del número de empresas de publicidad (category_code = 'advertising') fundadas a partir de 2000 en adelante. Para ello, utilice el operador relacional $gt, agrupe los resultados por año de creación ('founded_year') y ordénelos para que se muestren posterioremente en un gráfico de líneas por año.

```
# Cargar bibliotecas
library(mongolite)
library(ggplot2)

# Conectar a la colección 'companies'
companies_collection = mongo(collection="companies", db="sample_training", url=cadena_conexion, options

# Definir la canalización de agregación
pipeline <- '[
  { "$match": { "category_code": "advertising", "founded_year": { "$gt": 2000 } } },
  { "$group": { "_id": "$founded_year", "count": { "$sum": 1 } } },
  { "$sort": { "_id": 1 } }
]'

# Ejecutar la canalización de agregación
result <- companies_collection$aggregate(pipeline)

print(result)
```

```
##     _id count
## 1  2001    13
## 2  2002    15
## 3  2003    17
## 4  2004    24
## 5  2005    32
## 6  2006    49
## 7  2007    80
## 8  2008    77
## 9  2009    27
## 10 2012     1
## 11 2013     1
```

```
# Crear un marco de datos con tus datos
data <- data.frame(
  ano = c(2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2012),
  count = c(13, 15, 17, 24, 32, 49, 80, 77, 27, 1)
)

# Crear el gráfico de líneas
ggplot(data, aes(x = ano, y = count)) +
  geom_line() +
  labs(title = "Número de Empresas de Publicidad Fundadas por Año",
       x = "Año de Fundación",
       y = "Número de Empresas")
```


Número de Empresas de Publicidad Fundadas por Año
Número de Empresas
0
20
40
60
80
2002.5
2005.0
2007.5
2010.0
2012.5
Año de Fundación

4-1.pdf