DATA 607 - Week 12

Joshua Sturm and Aryeh Sturm

11/16/2017

Task

For this assignment, you should take information from a relational database and migrate it to a NoSQL database of your own choosing.

For the relational database, you might use the flights database, the tb database, the "data skills" database your team created for Project 3, or another database of your own choosing or creation.

For the NoSQL database, you may use MongoDB (which we introduced in week 7), Neo4j, or another NoSQL database of your choosing.

Your migration process needs to be reproducible. R code is encouraged, but not required. You should also briefly describe the advantages and disadvantages of storing the data in a relational database vs. your NoSQL database.

Approach

We'll be using the flights database from week one. It will be imported to MySQL, and then transferred to a new MongoDB database. We chose to work with MongoDB simply because we've never used it before, so we wanted to try it out.

Import libraries

```
library(tidyverse)
library(RMySQL)
library(mongolite)
```

Import MySQL database

The database files are located in this project's GitHub repository.

```
#
# Connect to the database
#
flights.con <- dbConnect(MySQL(),
                         user = 'test',
                         password = 'thisisnotsecure',
                         host = 'localhost',
                         dbname = 'flights')
# Ensure proper connection
dbGetInfo(flights.con)
```

```
## $host
## [1] "localhost"
##
## $user
```

```
## [1] "test"
##
## $dbname
## [1] "flights"
##
## $conType
## [1] "localhost via TCP/IP"
##
## $serverVersion
## [1] "5.7.20-log"
##
## $protocolVersion
## [1] 10
##
## $threadId
## [1] 22
##
## $rsId
## list()
```

```
dbListTables(flights.con)
```

```
## [1] "airlines" "airports" "flights"  "planes"   "weather"
```

Assign tables to variables

```
#
# Two ways two do this: dbSendQuery() + dbFetch(), or in one step using dbGetQuery()
#
airlines <- dbGetQuery(flights.con,
                       'SELECT * FROM airlines')
airports <- dbGetQuery(flights.con,
                       'SELECT * FROM airports')
flights <- dbGetQuery(flights.con,
                      'SELECT * FROM flights')
planes <- dbGetQuery(flights.con,
                     'SELECT * FROM planes')
weather <- dbGetQuery(flights.con,
                      'SELECT * FROM weather')

#
# Sever database connection
#
dbDisconnect(flights.con)
```

```
## [1] TRUE
```

Create Mongo collection and insert documents

```
#
# Run 'mongod' to start a local mongo server
#
```

```
flights.mongo <- mongo(collection = "flights", db = "flightsdb")
flights.mongo$insert(flights)
```

```
## List of 5
##  $ nInserted  : num 336776
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
#
# We can also create individual tables (databases) for airlines, airports, etc...
#
airlines.mongo <- mongo(collection = "flights", db = "airlines")
airlines.mongo$insert(airlines)
```

```
## List of 5
##  $ nInserted  : num 16
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
airports.mongo <- mongo(collection = "flights", db = "airports")
airports.mongo$insert(airports)
```

```
## List of 5
##  $ nInserted  : num 1397
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
planes.mongo <- mongo(collection = "flights", db = "planes")
planes.mongo$insert(planes)
```

```
## List of 5
##  $ nInserted  : num 3322
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
weather.mongo <- mongo(collection = "flights", db = "weather")
weather.mongo$insert(weather)
```

```
## List of 5
##  $ nInserted  : num 8719
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
#
# Verify migration worked correctly
#
head(flights.mongo$find())
```

```
##   year month day dep_time dep_delay arr_time arr_delay carrier tailnum
## 1 2013     1   1      517         2      830        11      UA  N14228
## 2 2013     1   1      533         4      850        20      UA  N24211
## 3 2013     1   1      542         2      923        33      AA  N619AA
## 4 2013     1   1      544        -1     1004       -18      B6  N804JB
## 5 2013     1   1      554        -6      812       -25      DL  N668DN
## 6 2013     1   1      554        -4      740        12      UA  N39463
##   flight origin dest air_time distance hour minute
## 1   1545    EWR  IAH      227     1400    5     17
## 2   1714    LGA  IAH      227     1416    5     33
## 3   1141    JFK  MIA      160     1089    5     42
## 4    725    JFK  BQN      183     1576    5     44
## 5    461    LGA  ATL      116      762    6     54
## 6   1696    EWR  ORD      150      719    6     54
```

Conclusion

SQL is an intuitive, standardizard language. It's a lot more common, so it has much more community support. On the downside, it requires long statements for complex queries, and it's not so simple for quick edits.

NoSQL is more dynamic, and more scalable. It's also schema-less, so it can be deployed quickly, with lower maintenance costs. It is gaining in popularity (MongoDB recently went public). However, it's a non-standardized language, so it varies between brands. Since it's newer, it also doesn't have the community support that traditional relational databases have.