Introduction to the rmongodb Package

MongoDB (www.mongodb.org) is a scalable, high-performance, document-oriented NoSQL database. The **rmongodb** package provides an interface from the statistical software R (www.r-project.org) to MongoDB and back using the mongodb-C library.

This vignette will provide a first introduction to the **rmongodb** package and offer a lot of code to get stared. If you need anyhelp getting started with MongoDB please check the resources provided by MongoDB: http://docs.mongodb.org/manual/tutorial/getting-started/

Installing and loading the rmongodb package

There is a stable CRAN version of **rmongodb** available:

```
install.packages("rmongodb")
```

You can also install the latest development version from the GitHub repository:

```
library(devtools)
install_github("rmongodb", "mongosoup")
```

The installation should be very simple and straightforward. No local MongoDB installation is required. Only if you install from source, the RUnit tests will need a local MongoDB installation.

After you've installed the the **rmongodb** package you can load it just like any other package:

```
library(rmongodb)
```

Connecting R to MongoDB

First of all we have to create a connection to a MongoDB installation. If no paramters are provided we connect to the MongoDB installation running on localhost. Parameters for external installations and user authentication are implemented and documented.

```
help("mongo.create")
mongo <- mongo.create()
```

```
## Unable to connect to 127.0.0.1:27017, error code = 2
```

```
mongo
```

```
## [1] 0
## attr(,"mongo")
## <pointer: 0x10be2e160>
## attr(,"class")
## [1] "mongo"
## attr(,"host")
## [1] "127.0.0.1"
## attr(,"name")
## [1] ""
## attr(,"username")
## [1] ""
## attr(,"password")
## [1] ""
## attr(,"db")
## [1] "admin"
## attr(,"timeout")
## [1] 0
```

```
mongo.is.connected(mongo)
```

```
## [1] FALSE
```

It's always a good idea to check if there is a working connection to your MongoDB to avoid errors.

Getting databases and collections

Get all databases of your MongoDB connection:

```
if (mongo.is.connected(mongo) == TRUE) {
    mongo.get.databases(mongo)
}
```

Get all collections in a specific database of your MongoDB (in this case, the “rmongodb” database)

```
if (mongo.is.connected(mongo) == TRUE) {
    db <- "rmongodb"
    mongo.get.database.collections(mongo, db)
}
coll <- "rmongodb.zips"
```

We will use the 'zips' collection in the following examples. The 'zips' collection holds the MongoDB example data set called “Zip Code Data Set” (http://docs.mongodb.org/manual/tutorial/aggregation-zip-code-data-set/). This data set is available as JSON and contains zip code data from the US.

Getting the size of collections, a sample

document and values for a key

Using the command *mongo.count*, we can check how many documents are in the collection or in the result of a specific query. More information for all functions can be found in the help files.

```
if (mongo.is.connected(mongo) == TRUE) {
    help("mongo.count")
    mongo.count(mongo, coll)
}
```

In order to run queries it is important to know some details about the available data. First of all you can run the command *mongo.find.one* to get one document from your collection.

```
if (mongo.is.connected(mongo) == TRUE) {
    mongo.find.one(mongo, coll)
}
```

The command *mongo.distinct* is going to provide a list of all values for a specific key.

```
if (mongo.is.connected(mongo) == TRUE) {
    res <- mongo.distinct(mongo, coll, "city")
    head(res)
}
```

Finding some first data

Now we can run the first queries on our MongoDB. In this case we ask for one document providing zip code data for the city “COLORADO CITY”. Please be aware that the output of *mongo.find.one* is a BSON object, which can not be used directly for further analysis in R. Using the command *mongo.bson.to.list*, an R list object will be created from the BSON object.

```
if (mongo.is.connected(mongo) == TRUE) {
    cityone <- mongo.find.one(mongo, coll, "{\"city\":\"COLORADO CITY\"}")
    print(cityone)
    mongo.bson.to.list(cityone)
}
```

Creating BSON objects

Until recently, working with **rmonbodb** was all about creating BSON objects in R. This was a very difficult and error-prone task. Since **rmongodb** version 1.2 you can use JSON directly, and the syntax of the **rmongodb** package is much more similar to the MongoDB shell.

```
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.append(buf, "city", "COLORADO CITY")
```

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

```
query <- mongo.bson.from.buffer(buf)
query
```

```
##  city : 2     COLORADO CITY
```

The same BSON object can be created using one line of code and JSON:

```
mongo.bson.from.JSON("{\"city\":\"COLORADO CITY\"}")
```

```
##  city : 2     COLORADO CITY
```

Finding more data

For real analyses it is important to get more than one document of data from MongoDB. As an example, we first use the command *mongo.distict* to get an overview about the population distribution. Then we check for all cities with less than two inhabitants (errors in the data set?).

```
if (mongo.is.connected(mongo) == TRUE) {
    pop <- mongo.distinct(mongo, coll, "pop")
    hist(pop)
    boxplot(pop)

    nr <- mongo.count(mongo, coll, "{\"pop\":{\"$lte\":2}}")
    print(nr)
    pops <- mongo.find.all(mongo, coll, "{\"pop\":{\"$lte\":2}}")
    print(dim(pops))
    head(pops)
}
```

Finding more data with a more complex query

The analysis gets more interesting when creating a more complex query with two arguments. Using the package **jsonlite** you can check and visualize your JSON syntax first. Afterwards we query MongoDB with this JSON query.

```
library(jsonlite)
json <- "{\"pop\":{\"$lte\":2}, \"pop\":{\"$gte\":1}}"
cat(prettify(json))
```

```
## {
##  "pop" : {
##      "$lte" : 2
##  },
##  "pop" : {
##      "$gte" : 1
##  }
## }
```

```
validate(json)
```

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

```
if (mongo.is.connected(mongo) == TRUE) {
    pops <- mongo.find.all(mongo, coll, json)
    print(dim(pops))
    head(pops)
}
```

In this case you will get a warning because the construction of the data.frame object is still inefficient and error-prone due to the NoSQL data structure. We are currently working on a solution.

```
mongo.cursor.to.data.frame
```

```
## function (cursor, nullToNA = TRUE, ...)
## {
##     warning("This fails for most NoSQL data structures. I am working on a new solution")
##     res <- data.frame()
##     while (mongo.cursor.next(cursor)) {
##         val <- mongo.bson.to.list(mongo.cursor.value(cursor))
##         if (nullToNA == TRUE)
##             val[sapply(val, is.null)] <- NA
##         val <- val[sapply(val, class) != "mongo.oid"]
##         res <- rbind.fill(res, as.data.frame(val, ...))
##     }
##     return(as.data.frame(res))
## }
## <environment: namespace:rmongodb>
```

Inserting some data into MongoDB

Another interesting point is inserting data into MongoDB.

```
# insert data
a <- mongo.bson.from.JSON("{\"ident\":\"a\", \"name\":\"Markus\", \"age\":33}")
b <- mongo.bson.from.JSON("{\"ident\":\"b\", \"name\":\"MongoSoup\", \"age\":1}")
c <- mongo.bson.from.JSON("{\"ident\":\"c\", \"name\":\"UseR\", \"age\":18}")

if (mongo.is.connected(mongo) == TRUE) {
    icoll <- paste(db, "test", sep = ".")
    mongo.insert.batch(mongo, icoll, list(a, b, c))

    dbs <- mongo.get.database.collections(mongo, db)
    print(dbs)
    mongo.find.all(mongo, icoll)
}
```

Updating documents and creating indices for efficient queries

You can also update your data in MongoDB from R and add indices for more efficient queries.

```
if (mongo.is.connected(mongo) == TRUE) {
    mongo.update(mongo, icoll, "{\"ident\":\"b\"}", "{\"$inc\":{\"age\":3}}")

    res <- mongo.find.all(mongo, icoll)
    print(res)

    # Creating an index for the field 'ident'
    mongo.index.create(mongo, icoll, "{\"ident\":1}")
    # check mongoshell!
}
```

Dropping/removing collections and databases and closing the connection to MongoDB

Of course there are also commands to drop databases and collections in MongoDB. After you finished all your analyses it's a good idea to destroy the connection to your MongoDB.

```
if (mongo.is.connected(mongo) == TRUE) {
    mongo.drop(mongo, icoll)
    mongo.drop.database(mongo, db)
    res <- mongo.get.database.collections(mongo, db)
    print(res)

    # close connection
    mongo.destroy(mongo)
}
```

Feedback and Issues

Please do not hesitate to contact us if there are any issues using **rmongodb**. Issues or pull requests can be submitted via github: https://github.com/mongosoup/rmongodb