Athens University of Economics and Business

School of Business

Department of Management Science & Technology

Master of Science in Business Analytics

Program:	Full-time
Quarter:	2nd (Winter Quarter)
Course:	Big Data Systems and Architectures
Assignment:	Redis & MongoDB
Students (Registration №):	Sakellaris Emmanouil (f2822215), Souflas Eleftherios-Efthymios (f2822217)

Big Data Systems and Architectures
Redis & MongoDB Assignment

Students: Sakellaris Emmanouil, Souflas Eleftherios-Efthymios

Firstly, we installed REDIS and MongoDB on our workstations and downloaded the required datasets ("Recorded Actions" and "Bikes"). We used R programming language to answer all required tasks. The entire R code used for the assignment can be found in the "Code" folder.

Task 1 (REDIS)

We loaded R's REDUX library to implement REDIS commands through R code. We created a connection to the local instance of REDIS and loaded the relevant, for the completion of the tasks, csv files ("emails sent" and "modified listings"). Code:

```
# Load the library
library("redux")
# Create a connection to the local instance of REDIS
r <- redux::hiredis(redux::redis_config(host="127.0.0.1", port="6379"))
# Load csv
emails_sent <- read.csv(file.choose(), header = TRUE, sep=",")
modified_listings <- read.csv(file.choose(), header = TRUE, sep=",")
```

Task 1.1: How many users modified their listing on January?

From the "modified listings" csv file, we created a new vector with the modified listings of January that contained only the column "User ID". The new vector contained only the users who modified their listing in January. We created a loop to assign to a REDIS bitmap, named "Modifications January", the respective values. Then we counted the "1" values and found that 9,969 users modified their listing in January. Code:

```
# Create a new vector with the modified listings of January
ModificationsJanuary <- modified_listings[modified_listings$MonthID==1 &
                                  modified_listings$ModifiedListing==1,1]
# Create a loop to assign to a REDIS bitmap the respective values of each
# user
for (i in 1:length(ModificationsJanuary)){
  r$SETBIT("ModificationsJanuary",ModificationsJanuary[i],1)
}
# Find users who modified their listing in January
r$BITCOUNT("ModificationsJanuary") # 9969
```

Task 1.2: How many users did NOT modify their listing on January?

We performed logical negation on each bit of the previous bitmap, using the bitwise NOT operator, implemented by the "BITOP NOT" command of REDIS and stored the outcome to a new bitmap named "Non-Modifications January". We, then, used "BITCOUNT" to calculate the answer and got back 10,031 as an outcome. We added the Modifications and Non-modifications of January's listings (tasks 1.1 and 1.2) and got back as an answer 20,000. However, we know that there exist only 19,999 users. This happened because BITOP operations happen at byte-level increments. The 19,999 bits fit in 2,500 bytes = 2,500*8 bits = 19,999 + 1 bits. So, the last bit (20,000th) was used, although not depicting a User ID and had the value of 0. So, when the bitwise NOT operation was used, it performed a logical negation and was turned into 1. The true answer to the question is that 10,030 users did not

modify their listing in January. Code:

```
# Perform logical negation on each bit of the previous bitmap
r$BITOP("NOT","NonModificationsJanuary","ModificationsJanuary")
# Find users who did NOT modify their listing on January
r$BITCOUNT("NonModificationsJanuary") # 10031
# Sum users who modified and not modified their listing on January
r$BITCOUNT("ModificationsJanuary") +
  r$BITCOUNT("NonModificationsJanuary") # 20000
# Find bytes
20000/8
# Get the 20,000th user (do not really exist)
r$GETBIT("NonModificationsJanuary",19999) # 1
```

Task 1.3: How many users received at least one e-mail per month (at least one e-mail in January and at least one e-mail in February and at least one e-mail in March)?
We created a new data frame with all unique combinations of "User ID" and "Month ID" columns of the "emails sent" data frame. From this data frame, we created three new vectors with the emails sent per month. So, we had a vector for each month (January, February, and March) with the distinct users that was sent at least one e-mail. We then created three bitmaps ("Emails January", "Emails February" and "Emails March") filling them with the value of "1" on the users that was sent an email, using the "SETBIT" command. We then performed the logical AND operation on each group (three bitmaps) of the corresponding bits, using the "BITOP AND" command of REDIS and stored the result of it in a new bitmap, called "results_1_3". Finally, we counted the "1" values of the forementioned bitmap, using the "BITCOUNT" command and discovered that there were 2,668 users who received at least one e-mail per month (at least one e-mail in January and at least one e-mail in February and at least one e-mail in March). Code:

```
# Create three new vectors with the distinct users that was sent
# at least one e-mail per month
emails_sent_unique <- unique(emails_sent[c("UserID","MonthID")])
EmailsJanuary <- emails_sent_unique[emails_sent_unique$MonthID==1,1]
EmailsFebruary <- emails_sent_unique[emails_sent_unique$MonthID==2,1]
EmailsMarch <- emails_sent_unique[emails_sent_unique$MonthID==3,1]
# Create three bitmaps filling them with "1"s on the users that was sent
# email
for (i in 1:length(EmailsJanuary)){
  r$SETBIT("EmailsJanuary",EmailsJanuary[i],1)
}
for (i in 1:length(EmailsFebruary)){
  r$SETBIT("EmailsFebruary",EmailsFebruary[i],1)
}
for (i in 1:length(EmailsMarch)){
  r$SETBIT("EmailsMarch",EmailsMarch[i],1)
}
# Find users who received at least one e-mail per month
r$BITOP("AND","results_1_3",c("EmailsJanuary","EmailsFebruary",
"EmailsMarch"))
r$BITCOUNT("results_1_3") # 2668
```

Task 1.4: How many users received an e-mail on January and March but NOT on February?

We performed an inversion of “Emails February”, using “BITOP NOT” and stored the result in a new REDIS bitmap, named “No Emails February”. We then performed “BITOP AND” on “Emails January”, “No Emails February”, and “Emails March” and stored the result on a new bitmap, named “results_1_4”. We then counted the newly created bitmap’s “1” values and found that 2417 users received an e-mail on January and March but NOT on February. Code:

```
# Perform an inversion of "EmailsFebruary"
r$BITOP("NOT","NoEmailsFebruary","EmailsFebruary")
# Find users who received an e-mail in January and March but NOT in
# February
r$BITOP("AND","results_1_4",c("EmailsJanuary","NoEmailsFebruary",
"EmailsMarch"))
r$BITCOUNT("results_1_4") # 2417
```

Task 1.5: How many users received an e-mail on January that they did not open but they updated their listing anyway?

Firstly, we created a vector containing the users that opened an email in January. Then, we created a relevant bitmap in REDIS, named "Emails Opened January", that contained the forementioned users. Then we performed an inversion of “Emails Opened January”, using “BITOP NOT” and stored the result in a new REDIS bitmap, named “Emails Not Opened January”. The forementioned lastly created bitmap contained all users that did not open any e-mail in January, either they received or not an e-mail in that month. We then performed “BITOP AND” on “Emails January”, “Emails Not Opened January”, and “Modifications January” and stored the result on a new bitmap, named “results_1_5”. The forementioned newly created bitmap contained all users who received an e-mail in January, and they did not open any email in that month, and they modified their listing in that month. We then counted the newly created bitmap’s “1” values and found that 1961 users received an e-mail in January that they did not open but they updated their listing anyway. Code:

```
# Create a vector containing the users that was sent an email in January
# and they opened it
EmailsOpenedJanuary <-
unique(emails_sent[(emails_sent$MonthID==1) &
(emails_sent$EmailOpened==1),2])
# Create the relevant bitmap in REDIS
for (i in 1:length(EmailsOpenedJanuary)){
  r$SETBIT("EmailsOpenedJanuary",EmailsOpenedJanuary[i],1)
}
# Perform an inversion of previously created bitmap
r$BITOP("NOT","EmailsNotOpenedJanuary","EmailsOpenedJanuary")
# Find users who received an e-mail in January that they did not open but
# they updated their listing anyway
r$BITOP("AND","results_1_5",c("EmailsJanuary","EmailsNotOpenedJanuary",
                              "ModificationsJanuary"))
r$BITCOUNT("results_1_5") # 1961
```

Task 1.6: How many users received an e-mail on January that they did not open but they updated their listing anyway on January OR they received an e-mail on February that they did not open but they updated their listing anyway on February OR they received an e-mail on March that they did not open but they updated their listing anyway on March?

Firstly, we created two new REDIS bitmaps, named “Emails Not Opened February” and “Emails Not Opened March” containing all users that did not open any e-mail, either they received or not an e-mail in February and March respectively (with the same way as it was done for January in the previous task).
From the “modified listings” csv file, we created two new vectors with the modified listings of February and March, that contained only the User IDs. The new vectors contained 10,007 and 9,991 users, who modified their listing in February and March respectively. We then created two loops to assign to REDIS bitmaps, named "Modifications February" and "Modifications March", the respective values.
We then performed “BITOP AND” on “Emails <Month>”, “Emails Not Opened <Month>”, and “Modifications <Month>” (where <Month> was replaced with the respective month, February and March respectively) and stored the results on new bitmaps, named “results_1_6_b” and “results_1_6_c” respectively (we can consider that the bitmap “results_1_6_a” is the “results_1_5”, created in the previous task and containing the respective outcome for January). The forementioned newly created bitmaps contained all users that received an e-mail, and they did not open it, and they modified their listings in February and March respectively (like doing a separate task 1.5 for each of the forementioned months).
Lastly, we performed the logical inclusive OR operation on each group of corresponding bits of the bitmaps "results_1_5", "results_1_6_b", and "results_1_6_c", using “BITOP OR” command of REDIS and stored the outcome in a new bitmap, named "results_1_6". We then counted the newly created bitmap’s “1” values and found that 5249 users received an e-mail in January that they did not open but they updated their listing anyway in January OR they received an e-mail in February that they did not open but they updated their listing anyway in February OR they received an e-mail in March that they did not open but they updated their listing anyway in March. Code:

```
# Create two vectors containing the users that was sent to them an email
# that they opened in February and March respectively
EmailsOpenedFebruary <-
  unique(emails_sent[(emails_sent$MonthID==2) &
(emails_sent$EmailOpened==1),2])
EmailsOpenedMarch <-
  unique(emails_sent[(emails_sent$MonthID==3) &
(emails_sent$EmailOpened==1),2])
# Create the relevant bitmaps in REDIS
for (i in 1:length(EmailsOpenedFebruary)){
  r$SETBIT("EmailsOpenedFebruary",EmailsOpenedFebruary[i],1)
}
for (i in 1:length(EmailsOpenedMarch)){
  r$SETBIT("EmailsOpenedMarch",EmailsOpenedMarch[i],1)
}
# Invert previously created bitmaps
r$BITOP("NOT","EmailsNotOpenedFebruary","EmailsOpenedFebruary")
r$BITOP("NOT","EmailsNotOpenedMarch","EmailsOpenedMarch")
# Create two new vectors with the modified listings of February and March
ModificationsFebruary <- modified_listings[modified_listings$MonthID==2 &
                                  modified_listings$ModifiedListing==1,1]
ModificationsMarch <- modified_listings[modified_listings$MonthID==3 &
                                  modified_listings$ModifiedListing==1,1]
# Create loops to assign to REDIS bitmaps the respective values
for (i in 1:length(ModificationsFebruary)){
  r$SETBIT("ModificationsFebruary",ModificationsFebruary[i],1)
}
```

```
for (i in 1:length(ModificationsMarch)){
  r$SETBIT("ModificationsMarch",ModificationsMarch[i],1)
}
# Create bitmaps of each month's users who received an e-mail that they
# did not open but they updated their listing anyway
r$BITOP("AND","results_1_6_b",c("EmailsFebruary",
                        "EmailsNotOpenedFebruary","ModificationsFebruary"))
r$BITOP("AND","results_1_6_c",c("EmailsMarch","EmailsNotOpenedMarch",
                                "ModificationsMarch"))
# Find the answer using "BITOP OR" and counting "1"s
r$BITOP("OR","results_1_6",c("results_1_5","results_1_6_b",
                             "results_1_6_c"))
r$BITCOUNT("results_1_6") # 5249
```

Task 1.7: Does it make any sense to keep sending e-mails with recommendations to sellers? Does this strategy really work? How would you describe this in terms a business person would understand?

From all messages sent, without considering technical issues that sent more than one e-mail to a user in a month, the 58.8% were opened. From the above messages opened, the 49.9% can be linked to respective listing modifications, which we can further interpret that, from all messages sent, the 29.4% can be linked to respective listing modification. From this point of view, one can consider it relatively acceptable that if we send 100 e-mails, we can get back approximately 30 listing modifications.

However, from another point of view, from all listing modifications done, only the 28.2% can be linked to e-mail recipients who opened the e-mails and were influenced by the recommendations enlisted in them, whereas the vast majority (71.8%) of all listing modifications were done by users who either did not receive an e-mail or received one but never opened it.

So, we can consider that it makes no sense to keep sending e-mails with recommendations to sellers, as the vast majority of listing modifications are done by people that for sure did not take into account any recommendation e-mail. Also, those who received an e-mail, we cannot securely mention that the recommendations in the e-mail were the key factor that made them modify their listing and that they would not modify their listing either way. For the forementioned reasons, we state that the strategy of sending e-mails with recommendations to sellers does not really work as someone would desire and we should stop sending them. Code:

```
# 58.8% of messages sent, without taking account technical issues, were
# opened
(r$BITCOUNT('EmailsOpenedJanuary') + r$BITCOUNT('EmailsOpenedFebruary') +
  r$BITCOUNT('EmailsOpenedMarch')) /
  (r$BITCOUNT("EmailsJanuary") + r$BITCOUNT("EmailsFebruary") +
     r$BITCOUNT("EmailsMarch"))
# 49.9% of messages opened can be linked to respective listing
# modification, which is linked to the 29.4% of the total messages sent,
# without taking into account technical issues.
r$BITOP("AND","OpenModifyJan",c("EmailsOpenedJanuary",
                                "ModificationsJanuary"))
r$BITOP("AND","OpenModifyFeb",c("EmailsOpenedFebruary","ModificationsFebr
uary"))
r$BITOP("AND","OpenModifyMar",c("EmailsOpenedMarch","ModificationsMarch")
)
(r$BITCOUNT("OpenModifyJan") + r$BITCOUNT("OpenModifyFeb") +
```

```
  r$BITCOUNT("OpenModifyMar")) /
  (r$BITCOUNT('EmailsOpenedJanuary') + r$BITCOUNT('EmailsOpenedFebruary')
+
     r$BITCOUNT('EmailsOpenedMarch'))
(r$BITCOUNT("OpenModifyJan") + r$BITCOUNT("OpenModifyFeb") +
    r$BITCOUNT("OpenModifyMar")) /
  (r$BITCOUNT("EmailsJanuary") + r$BITCOUNT("EmailsFebruary") +
     r$BITCOUNT("EmailsMarch"))
# However, all listing modifications that can be considered to be a
# result of the opened emails account for the 28.2% of all listing
# modifications done, whereas all listing modifications that were done by
# users who received an e-mail that they did not open or by users that
# they did not receive an email account for the 71.8% of all listing
# modifications done.
nlistings <-
nrow(modified_listings[modified_listings$ModifiedListing==1,])
(r$BITCOUNT("OpenModifyJan") + r$BITCOUNT("OpenModifyFeb") +
    r$BITCOUNT("OpenModifyMar")) / nlistings
r$BITOP("AND","NotOpenModifyJan",c("EmailsNotOpenedJanuary",
                                   "ModificationsJanuary"))
r$BITOP("AND","NotOpenModifyFeb",c("EmailsNotOpenedFebruary",
                                   "ModificationsFebruary"))
r$BITOP("AND","NotOpenModifyMar",c("EmailsNotOpenedMarch",
                                   "ModificationsMarch"))
(r$BITCOUNT("NotOpenModifyJan") + r$BITCOUNT("NotOpenModifyFeb") +
  r$BITCOUNT("NotOpenModifyMar")) / nlistings
```

Task 1.8: (Optional Task) Do the previous subtasks again by using any type of relational or non-relational database. Compare the complexity of the solutions. Then benchmark the query execution time for the dataset that you have. At last, boost the number of entries to 1 billion rows (create your own dummy entries). Perform the benchmark again.

Firstly, we loaded the csv files in two tables named “emails sent” and “modified listings” of “BDSA” database and “DBO” schema of Microsoft SQL Server RDBMS. Then, we created the SQL queries to retrieve the same information required in the previous subtasks (1.1 – 1.6). The SQL queries are the following:

```
-- 1.1 : 9969
SELECT distinct count(UserID)
FROM BDSA.dbo.modified_listings as a
WHERE a.[ ModifiedListing] = 1 AND a.[ MonthID] = 1;

-- 1.2 : 1030
SELECT distinct count(UserID)
FROM BDSA.dbo.modified_listings as a
WHERE a.[ ModifiedListing] = 0 AND a.[ MonthID] = 1;

-- 1.3 : 2668
SELECT COUNT(a.[ UserID])
FROM
(SELECT [ UserID], COUNT(DISTINCT b.[ MonthID]) as Months
FROM BDSA.dbo.emails_sent as b
GROUP BY b.[ UserID]
```

```
HAVING COUNT(DISTINCT b.[ MonthID]) = 3) as a;

-- 1.4 : 2417
SELECT COUNT(a.[ UserID])
FROM
(SELECT [ UserID], COUNT(DISTINCT b.[ MonthID]) as Months
FROM BDSA.dbo.emails_sent b
WHERE b.[ MonthID] IN (1,3) AND NOT EXISTS
(SELECT 1 FROM BDSA.dbo.emails_sent c WHERE c.[ MonthID] = 2 AND c.[
UserID] = b.[ UserID])
GROUP BY b.[ UserID]
HAVING COUNT(DISTINCT b.[ MonthID]) = 2) as a;

-- 1.5 : 1961
SELECT COUNT(DISTINCT a.[ UserID])
FROM BDSA.dbo.emails_sent a
WHERE a.[ MonthID] = 1 AND a.[ EmailOpened] = 0 AND EXISTS
(SELECT 1 FROM BDSA.dbo.modified_listings b WHERE b.[ MonthID] = 1 AND
b.[ ModifiedListing] = 1 AND b.UserID = a.[ UserID])
AND NOT EXISTS
(SELECT 1 FROM BDSA.dbo.emails_sent c WHERE c.[ MonthID] = 1 AND c.[
EmailOpened] = 1 AND c.[ UserID] = a.[ UserID]);

-- 1.6 : 5249
SELECT COUNT(*) FROM (
SELECT a.[ UserID]
FROM BDSA.dbo.emails_sent a
WHERE a.[ MonthID] = 1 AND a.[ EmailOpened] = 0 AND EXISTS
(SELECT 1 FROM BDSA.dbo.modified_listings b WHERE b.[ MonthID] = 1 AND
b.[ ModifiedListing] = 1 AND b.UserID = a.[ UserID])
AND NOT EXISTS
(SELECT 1 FROM BDSA.dbo.emails_sent c WHERE c.[ MonthID] = 1 AND c.[
EmailOpened] = 1 AND c.[ UserID] = a.[ UserID])
UNION
SELECT a.[ UserID]
FROM BDSA.dbo.emails_sent a
WHERE a.[ MonthID] = 2 AND a.[ EmailOpened] = 0 AND EXISTS
(SELECT 1 FROM BDSA.dbo.modified_listings b WHERE b.[ MonthID] = 2 AND
b.[ ModifiedListing] = 1 AND b.UserID = a.[ UserID])
AND NOT EXISTS
(SELECT 1 FROM BDSA.dbo.emails_sent c WHERE c.[ MonthID] = 2 AND c.[
EmailOpened] = 1 AND c.[ UserID] = a.[ UserID])
UNION
SELECT a.[ UserID]
FROM BDSA.dbo.emails_sent a
WHERE a.[ MonthID] = 3 AND a.[ EmailOpened] = 0 AND EXISTS
(SELECT 1 FROM BDSA.dbo.modified_listings b WHERE b.[ MonthID] = 3 AND
b.[ ModifiedListing] = 1 AND b.UserID = a.[ UserID])
AND NOT EXISTS
(SELECT 1 FROM BDSA.dbo.emails_sent c WHERE c.[ MonthID] = 3 AND c.[
EmailOpened] = 1 AND c.[ UserID] = a.[ UserID])
) total;
```

Then we optimized the R code already produced, which was presented in the subtasks

above, in order to be executed faster. Instead of using the slow for loops to implement the SETBIT command of REDIS to iteratively set single bits of the bitmaps, we replaced multiple SETBIT calls with a single call to the variadic BITFIELD command and the use of fields of type “u1”. Because we were not able to implement the forementioned command with the REDUX library of R, we imported the RCPPREDIS library. Through the former library, we also found out that we could not set bits at offset 1,000,000 and more, because it produced an error, whereas not experiencing the same error message through the use of the latter library’s commands. We then separated the code that was creating the bitmaps and the code that was calculating the outcomes. The code prior to the forementioned optimization needed 33 seconds to complete. The optimized R code for the creation of a REDIS BITMAP is:

```
# Load the libraries
library('RcppRedis')
# Create a connection to the local instance of REDIS
redis <- new(Redis, "localhost")
# Function to create a REDIS BITMAP
Create.Redis.Bitmap <- function(subset, bitmapName) {
  comment <- paste('SETBIT', bitmapName, max(subset), '1', sep=" ")
  redis$exec(comment)
  for (i in seq(1, length(subset), by=200000)) {
    if (i+199999 < length(subset)){
      n <- i+199999
    } else {
      n <- length(subset)
    }
    comment <- subset[i:n]
    comment <- paste('SET u1', comment, '1', sep=" ", collapse=" ")
    comment <- paste('BITFIELD', bitmapName, comment)
    redis$exec(comment)
  }
}
# Then with many calls to the function we created all BITMAPS and then
# calculated all outcomes. For detailed report of the entire procedure
# followed, please see the “Assign_Redis_Faster.R” inside the “Code”
# folder.
```

The SQL queries above took less than 2 seconds (1.8 seconds) to produce the same outcome as was produced by the optimized REDIS through R code in less than 1 second (0.98 seconds). The solution provided by SQL seems to be easier to start using for someone who is familiar with SQL, however the bitwise operations used in REDIS are, objectively speaking, less complex, simpler to explain for a beginner learner of both “technologies” and more flexible to produce the desired outcome needed in each case.

Because the hard drive storage of our computer was not enough to fit 1 billion rows per table (1 billion rows in “emails sent” and 1 billion rows in “modified listings”), we created 500 million dummy rows per table (totalling 1 billion rows), which were fit in 26 GB inside SQL Server. We re-executed the above SQL script for the new tables, and it took 34 minutes and 40 seconds to run. The SQL script used to produce the dummy entries is:

```
-- Dummy entries of table emails_sent_2
WITH
```

```
  L0   AS (SELECT c FROM (SELECT 1 UNION ALL SELECT 1) AS D(c)), -- 2^1
  L1   AS (SELECT 1 AS c FROM L0 AS A CROSS JOIN L0 AS B),       -- 2^2
  L2   AS (SELECT 1 AS c FROM L1 AS A CROSS JOIN L1 AS B),       -- 2^4
  L3   AS (SELECT 1 AS c FROM L2 AS A CROSS JOIN L2 AS B),       -- 2^8
  L4   AS (SELECT 1 AS c FROM L3 AS A CROSS JOIN L3 AS B),       -- 2^16
  L5   AS (SELECT 1 AS c FROM L4 AS A CROSS JOIN L4 AS B),       -- 2^32
  Nums AS (SELECT ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS k FROM L5)
select 'abc' as EmailID, k/3 as UserID , (ABS(CHECKSUM(NewId())) % 3) + 1
as MonthID, ABS(CHECKSUM(NewId())) % 2 as EmailOpened
INTO BDSA.dbo.emails_sent_2
from nums
where k <= 150000000 -- number of rows

-- Dummy entries of table modified_listings_2
WITH
  L0   AS (SELECT c FROM (SELECT 1 UNION ALL SELECT 1) AS D(c)), -- 2^1
  L1   AS (SELECT 1 AS c FROM L0 AS A CROSS JOIN L0 AS B),       -- 2^2
  L2   AS (SELECT 1 AS c FROM L1 AS A CROSS JOIN L1 AS B),       -- 2^4
  L3   AS (SELECT 1 AS c FROM L2 AS A CROSS JOIN L2 AS B),       -- 2^8
  L4   AS (SELECT 1 AS c FROM L3 AS A CROSS JOIN L3 AS B),       -- 2^16
  L5   AS (SELECT 1 AS c FROM L4 AS A CROSS JOIN L4 AS B),       -- 2^32
  Nums AS (SELECT ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS k FROM L5)
select k/3 as UserID , (ABS(CHECKSUM(NewId())) % 3) + 1 as MonthID,
ABS(CHECKSUM(NewId())) % 2 as ModifiedListing
INTO BDSA.dbo.modified_listings_2
from nums
where k <= 150000000 -- number of rows
```

However, because our computer had only 16 GB of RAM, it was not feasible to fit the forementioned tables to be executed through REDIS and perform a benchmark comparison. So, we decided to reduce the number of rows to 300 million rows in total (150 million rows per table – as shown in the SQL script above). We executed the above SQL script for the 300 million rows tables, and it took 7 minutes and 43 seconds to complete, without using any index or other optimization performance method. Through R and REDIS commands, it required in total 56 minutes and 48 seconds to successfully complete its execution, without counting the time needed for the load of the data from the SQL Server database to R's memory. Most of the time spent was time needed for the creation of REDIS's BITMAPS, whereas the calculations needed for the answer of the subtasks completed in milliseconds (0.11 seconds) as we can see from the R output printed below.

```
> # Measure time taken to run the script
> print(total_redis_load_time)
Time difference of 56.80689 mins
>
> print(total_proc_time)
Time difference of 0.1165519 secs
```

Conclusions driven from the forementioned procedure undertaken are:

- REDIS BITFIELD command although it is faster than repetitive calls of SETBIT commands, it fails to load data more than 300,000 bits into a BITMAP with a single call, so we created a for loop per 200,000 rows to call the forementioned command.
- REDIS is slow at loading big data to its memory, but extremely fast to produce operations with them. It also requires smaller storage capacity for the load of the same data to its memory.

- REDIS can be used for caching queries. If the same SQL queries where ran to a productive transactional database that hundreds of thousand transactions are executed in it daily, then the execution time of them would certainly be higher, which is not acceptable. In that case, we could use REDIS during non-working hours to produce queries whose data are not changing fast or for creating reports at a daily basis, if we don't want to overload database servers with storage and execution time needed for the creation and refresh of materialized views (which is a database solution of caching queries instead of using REDIS).

Task 2 (MongoDB)

Firstly, we read all files of the "bikes" dataset from R. We generated a list with all the paths of the files and used this file as an index. This index file was created with Unix Terminal through the MS "Windows Subsystem for Linux" and the use of the command (after we changed working directory to the directory containing the files):

```
find * | grep json > files_list.txt
```

We then loaded R's JSONLITE library and with the use of the index file we loaded all JSONs into a list of characters (each element of the list was a JSON character). We then replaced empty JSON objects and arrays with "null" keyword and the Telephone field that had no name with a chosen name ("Tel"). Finally, we created a data frame per JSON document, appended them to a list and transformed the list to a single data frame, having as columns all possible fields that the JSON document could have and as rows all JSON documents.

```
# Load jsonlite
library("jsonlite")
# Save JSON to a variable
index_file_path <- paste('path/to/folder', sep="")
index_file <- paste(index_file_path, 'files_list.txt', sep = "/")
index_file <- readLines(index_file)
f <- file.path(index_file_path, index_file)
json_data <- lapply(f, readLines, encoding="UTF-8")
# Replace empty JSON objects or arrays with null
json_data <- gsub("{}", "null", json_data, perl=TRUE)
json_data <- gsub("[[]]", "null", json_data)
# Replace empty name of Tel field of add_seller field object
json_data <- gsub('\"\":', '\"Tel\":', json_data)
# Check it is OK
json_data[3]
# Number of JSON objects in the list
n <- length(json_data)
# Create empty list
l <- list()
# From character vector save all JSONs in a list of data frames
for (i in 1:n){
  lj <- fromJSON(json_data[i], flatten = TRUE, simplifyDataFrame=FALSE)
  fne <- Filter(Negate(purrr::is_empty), lj)
  df <- as.data.frame(fne)
  l[[length(l)+1]] <- df[1,]
}# Create a single data frame from the list of data frames
ldf <- dplyr::bind_rows(l, .id = "column_label")
```

We then "cleaned" the data frame though R. This included tasks of deletion of metric unit values (like kilometres, euros cubic centimetres and brake horsepower) from numeric fields, deletion of commas that implied separation of thousands, and deletion of words that implied not available values (NA) like "Ask for Price" in the price of a bike and "None" in the seller telephone field. We also deleted columns that provided the same information as others, like classified number and ad id columns. Finally, we transformed to numeric and date values all relevant and necessary "cleansed" fields. For thorough examination of the cleaning procedure, as well as the whole procedure followed, you can see the "Assign_Mongo.R" file in the "Code" folder.

Finally, we created the "ads" collection in MongoDB and loaded the "cleansed" data frame in it with the use of R's MONGOLITE library. Code:

```
# Load mongolite
library("mongolite")
# Open a connection to MongoDB
m <- mongo(collection = "ads",  db = "mydb", url = "mongodb://localhost")
# Insert this JSON object to MongoDB
m$insert(ldf)
```

We then answered the following subtasks:

Task 2.2: How many bikes are there for sale?
We counted all listings, using the "count" command and found that 29,701 bikes were inside the database for sale.
Code:

```
m$count() # 29701
```

Task 2.3: What is the average price of a motorcycle (give a number)? What is the number of listings that were used in order to calculate this average (give a number as well)? Is the number of listings used the same as the answer in 2.2? Why?
The average price of a motorcycle is approximately 2,963 € and was found using the "aggregate" command and, in the "group" stage of it, the "$avg" operator. The number of listings that were used to calculate the average price is 29,145, which is different from the total number of listings in the database. This is due to the fact that the "$avg" operator, that we used to compute the average price, ignores non-numeric values, including missing values. In the field "ad data Price" there exist 29,145 values of type "double" and 556 not available values, which number subtracted from the number of total listings returns the number of listings used for the calculation of the average price.
Code:

```
query <- '[{"$group":{"_id":{},"Avg_Price":{"$avg":"$ad_data_Price"}}}]'
m$aggregate(query) # 2962.701
query <- '[{"$group":{"_id":{},"Sum_Price":{"$sum":"$ad_data_Price"}}}]'
m$aggregate(query) # 86347921
query <-
'[{"$project":{"ad_data_Price":{"$type":"$ad_data_Price"},"_id":0}},
{"$match":{"ad_data_Price":{"$eq":"double"}}},
{"$group":{"_id":{},"Count_Listings_Used":{"$sum":1}}}]'
m$aggregate(query) # 29145
86347921/29145 # 2962.701
```

```
29701-29145 # 556
query <- '[{"$match":{"ad_data_Price":null}},
{"$group":{"_id":{},"Count_Ads_NA_Price":{"$sum":1}}}]'
m$aggregate(query) # 556
```

Task 2.4: What is the maximum and minimum price of a motorcycle currently available in the market?

The maximum price of a motorcycle in the database is 89,000 €. This value makes sense as it is for the BMW HP4, whose currently available price in the market is close to it. The minimum price of a motorcycle in the database is 1 €. This value does not make any sense, as it is not probable that a motorcycle can be bought for such a low value. This value exists, because in the database there also exist listings for spare parts of motorcycles. With a deep search, we found that the minimum reasonable price of a motorcycle, sold as a whole, is 150 €, which makes sense, as there existed two ads for the 1995 model of Yamaha JOG in the forementioned price. These motorcycles are already sold, and they were a bargain when their ads were online, as there exist similar bikes of the 1996 model of Yamaha JOG at the price of 250 € (https://www.car.gr/classifieds/bikes/view/325220425-yamaha-jog). The answer was produced using the "aggregate" command and in the "group" stage of it the "$max" and "$min" operators. Also, we used the "find" command to find the respective ads for the motorcycles' minimum and maximum price.

Code:

```
query <- '[{"$group":{"_id":{},"Max_Price":{"$max":"$ad_data_Price"},
"Min_Price":{"$min":"$ad_data_Price"}}}]'
m$aggregate(query) # max = 89,000 € and min = 1 €
m$find('{"ad_data_Price":89000}',
 '{"Make_Model":"$ad_data_Make_Model","Price":"$ad_data_Price","_id":0}')
m$find('{"ad_data_Make_Model" : "Yamaha JOG \'95","ad_data_Price" :
150}',
'{"URL" : "$query_url","Make_Model" : "$ad_data_Make_Model",
"Price" : "$ad_data_Price","Description" : "$description","_id" : 0}')
```

Task 2.5: How many listings have a price that is identified as negotiable?

The word "Negotiable" is found on the column "metadata model". On the cleansing phase of the data frame before importing it to MongoDB, we created a new column, named "metadata negotiable", with the information if the price of the ad is characterized by the seller as negotiable or not (1 or 0, respectively). So, with the use of "aggregate" command, we summed all "1" values to find that 1,348 listings have a price that is identified as negotiable. Alternatively, if we had not created a separate field for the "negotiable" information, we would have searched for the listings that contained this word in the initial field and we would have counted the ones that satisfied this condition.

Code:

```
# Query if the information of Negotiable ad prices were, as initially,
# on column metadata_model
query <-
'[{"$match":{"metadata_model":{"$regex":"Negotiable","$options":"i"}}},
{"$group":{"_id":{},"Negotiable":{"$sum":1}}}]'
# Query with information of Negotiable ad prices on created column
# metadata_Negotiable
query <- '[{"$group":{"_id":{},"Negotiable_Ads":{"$sum":
```

```
"$metadata_Negotiable"}}}]'
m$aggregate(query) # 1348
```

Task 2.6: (Optional) For each Brand, what percentage of its listings is listed as negotiable?

We grouped by the motorcycle brand and took the average value for the "metadata negotiable" field of each brand, providing also the total number of ads and the number of negotiable ads per brand in order for the user of the report to be able to assess high or low values of percentages provided (e.g. 0 or 100 percent), because there existed brands that had only a few listings registered in the database. For better presentation of the report, we rounded the percentages provided, transformed to integer the total number of negotiable ads per brand and sorted the result alphabetically per name of brand. In the appendix, you can find the list of the 198 motorcycle brands and their respective percentage of negotiable listings registered in the database.

Code:

```
query <- '[{"$group":{"_id":{"Brand":"$metadata_brand"},
"Negotiable_ads_Percentage":{"$avg":"$metadata_Negotiable"},
"Negotiable_ads":{"$sum":"$metadata_Negotiable"},"Total_ads":
{"$sum":1}}},{"$project":{"Brand":"$_id.Brand",
"Negotiable_ads_Percentage":{"$round":[{"$multiply":
["$Negotiable_ads_Percentage",100]},1]},"Negotiable_ads":{"$toInt":
"$Negotiable_ads"},"Total_ads":"$Total_ads","_id":0}},
{"$sort":{"Brand":1}}]'
m$aggregate(query)
```

Task 2.7: (Optional) What is the motorcycle brand with the highest average price?

If by motorcycle brand, buggy motorcycles (All-Terrain Vehicles) are also included, then with the use of the "aggregate" command, we firstly grouped listings by the brand, then projected the name of the brand and the average price of each brand, then sorted in a descending order of the price and lastly got the first row (maximum average price). With the forementioned method, we found that the motorcycle brand (including ATVs) with the highest average price is SEMOG. If by motorcycle brand, buggy motorcycles (All-Terrain Vehicles) are not implied, then in the forementioned "aggregate" command, we added a "match" step to exclude ATVs and we found that the motorcycle brand (excluding ATVs) with the highest average price is INDIAN. A picture of the two different types of motorcycle brands with the highest average price per category is included in the appendix (Figures 1 & 2). Code:

```
# If by motorcycle brand, ATVs or else Buggy Motorcycles are included
query <- '[{"$group":{"_id":{"Brand":"$metadata_brand"},
"Average_Price":{"$avg":"$ad_data_Price"}}},{"$project":{"Brand":
"$_id.Brand","Average_Price":"$Average_Price","_id":0}},
{"$sort":{"Average_Price":-1}},{"$project":{"_id":0,"Brand":
"$Brand"}},{"$limit":1}]'
# If by motorcycle brand, ATVs or else Buggy Motorcycles are not included
query <- '[{"$match":{"ad_data_Category":{"$ne":"Bike - Buggy"}}},
{"$group":{"_id":{"Brand":"$metadata_brand"},
"Average_Price":{"$avg":"$ad_data_Price"}}},{"$project":{"Brand":
"$_id.Brand","Average_Price":"$Average_Price","_id":0}},
{"$sort":{"Average_Price":-1}},{"$project":{"_id":0,"Brand":
"$Brand"}},{"$limit":1}]'
m$aggregate(query)
```

Appendix

	Brand	Negotiable_ads_Percentage	Negotiable_ads	Total_ads
1	AB	61.5	8	13
2	ACCESS	33.3	3	9
3	ACCESS MOTOR	28.6	2	7
4	ADIVA	80.0	4	5
5	ADLER	0.0	0	1
6	ADLY	0.0	0	7
7	AEON	3.0	1	33
8	AGM MOTORS	20.0	1	5
9	AIE	0.0	0	4
10	AJP	0.0	0	2
11	AMS	0.0	0	3
12	AMSTRONG	100.0	3	3
13	APOKOTOS	100.0	2	2
14	APRILIA	1.8	16	892
15	ARCTIC CAT	25.0	2	8
16	ARIEL	0.0	0	2
17	ASUS	20.0	1	5
18	BAJAJ	33.3	2	6
19	BAOTIAN	57.1	8	14
20	BAROSSA	0.0	0	1
21	BASHAN	52.4	11	21
22	BEELINE	0.0	0	1
23	BENELLI	7.3	3	41
24	BETA	27.5	11	40
25	BIMOTA	0.0	0	2
26	BMW	2.0	28	1394
27	BOATIAN	33.3	2	6
28	BOMBARDIER	100.0	1	1
29	BOOM-TRIKES	100.0	1	1
30	BRIXTON	0.0	0	18
31	BSA	33.3	5	15
32	BUELL	0.0	0	17
33	BUGGY MOTORS	100.0	3	3
34	BULTACO	50.0	1	2
35	CAGIVA	9.4	3	32
36	CAN-AM	9.1	2	22
37	CCM	0.0	0	6
38	CECTEK	0.0	0	2
39	CFMOTO	14.3	3	21
40	CHANG JIANG	0.0	0	1
41	CHEETAH	35.3	6	17
42	CPI	10.0	1	10
43	DAELIM	1.4	1	69
44	DAYANG	16.7	5	30
45	DAYTONA	13.2	52	393
46	DERBI	5.8	5	86
47	DIAS	100.0	3	3
48	DINLI	0.0	0	6
49	DIRT MOTOS	40.0	6	15
50	DKW	56.2	9	16
51	DUCATI	1.6	6	373
52	E-ATV	100.0	2	2
53	E-TON	50.0	1	2
54	EAGLE	0.0	0	3
55	EMB	0.0	0	1
56	EMW	50.0	1	2
57	ENFIELD	0.0	0	4
58	EUROMOTORS	0.0	0	6
59	EVOMOTO	0.0	0	5
60	FB MONDIAL	50.0	2	4
61	FEVER	100.0	1	1
62	FUXIN	25.0	1	4
63	FYM	0.0	0	1
64	G-FORCE	0.0	0	1
65	GAMAX	0.0	0	1
66	GARELLI	12.5	3	24
67	GAS-GAS	8.6	3	35
68	GEELY	0.0	0	1

69	GEMINI	66.7	4	6
70	GENATA	0.0	0	2
71	GENERIC	16.7	1	6
72	GILERA	3.1	18	590
73	GOES	0.0	0	1
74	HAOJIN	70.0	7	10
75	HARLEY DAVIDSON	3.9	12	309
76	HARLOW	0.0	0	1
77	HARTFORD	0.0	0	1
78	HEINKEL	40.0	2	5
79	HERCULES	33.3	1	3
80	HIGHPER	100.0	2	2
81	HONDA	4.0	247	6190
82	HOREX	16.7	1	6
83	HSUN	28.6	2	7
84	HUSABERG	3.3	1	30
85	HUSQVARNA	4.1	6	145
86	HYOSUNG	11.6	5	43
87	IMR	40.0	4	10
88	INDIAN	66.7	2	3
89	ITALJET	20.0	1	5
90	JAWA	43.8	7	16
91	JETMOTO	50.0	2	4
92	JIALING	5.0	1	20
93	JIANSHE	0.0	0	10
94	JINCHENG	28.6	2	7
95	JINLUN	100.0	1	1
96	JMSTAR	100.0	1	1
97	JONWAY	60.0	3	5
98	JOYNER	100.0	1	1
99	KAISAR	75.0	3	4
100	KAWASAKI	3.3	65	1953
101	KEEWAY	8.7	9	103
102	KINROAD	25.0	1	4
103	KL	0.0	0	1
104	KREIDLER	20.5	17	83
105	KTM	2.2	21	966
106	KUBERG	100.0	1	1
107	KXD	14.3	1	7
108	KYMCO	1.8	21	1148
109	LAMBRETTA	71.4	10	14
110	LAVERDA	0.0	0	2
111	LEM	25.0	3	12
112	LIFAN	24.0	25	104
113	LINGBEN	0.0	0	1
114	LINHAI	4.2	2	48
115	LINTEX	33.3	1	3
116	LML	22.2	6	27
117	LONCIN	12.0	3	25
118	MAICO	25.0	1	4
119	MALAGUTI	3.6	2	55
120	MASAI	0.0	0	2
121	MBK	40.0	2	5
122	MIKILON	0.0	0	1
123	MOBSTER	0.0	0	2
124	MODENAS	3.8	10	261
125	MONTESA	33.3	2	6
126	MORINI	0.0	0	3
127	MOTIVAS	33.3	1	3
128	MOTO GUZZI	2.0	1	50
129	MOTO MORINI	25.0	1	4
130	MOTOBI	100.0	1	1
131	MTG	100.0	4	4
132	MV AGUSTA	6.2	2	32
133	MZ	27.3	3	11
134	NEW FORCE MOTOR	0.0	0	1
135	NIPPONIA	11.1	1	9
136	NITRO MOTORS	100.0	1	1
137	NIU	100.0	1	1
138	NOMIK	50.0	1	2
139	NORTON	20.0	1	5
140	NOVA	0.0	0	2

```
141              NSU                0.0                0                9
142            ODESS              100.0                2                2
143          PEUGEOT                2.0                6              306
144              PGO               10.0                2               20
145          PIAGGIO                1.7               45             2685
146          POLARIS               17.1                6               35
147           POLINI                0.0                0                2
148             PUCH               22.2                2                9
149           QINGQI              100.0                2                2
150           QUADRO               40.0                2                5
151     REGAL-RAPTOR              100.0                2                2
152           REWACO               33.3                1                3
153    ROYAL ENFIELD               38.5                5               13
154            SACHS               30.6               19               62
155           SECKAM                0.0                0                3
156            SEMOG                0.0                0                1
157        SHAN YANG                0.0                0                1
158 SHANDONG LIANGZI                0.0                0                1
159           SHERCO              100.0                1                1
160         SHINERAY               10.7                3               28
161          SIAMOTO               20.0                1                5
162           SIMSON               27.3                3               11
163           SKYJET               50.0                5               10
164         SKYTEAM                 2.3                1               44
165              SMC               33.3                3                9
166           SOKUDO                0.0                0                4
167            SOLEX               26.7                4               15
168           SUBARU                0.0                0                1
169       SUPER MOTO               50.0                1                2
170           SUZUKI                2.2               52             2365
171              SWM               33.3                2                6
172              SYM                1.8               20             1090
173              TCB                9.1                4               44
174              TGB               14.3                1                7
175               TM               18.2                6               33
176          TRIUMPH                1.8                6              335
177          UNILLI                0.0                0                1
178            URAL               20.0                1                5
179           VEDIM                0.0                0                1
180         VEE ROAD              100.0                1                1
181            VESPA                5.1               14              274
182         VICTORIA                0.0                0                1
183          VICTORY              100.0                1                1
184            VMOTO               16.7                1                6
185              VOR               16.7                1                6
186              WSK              100.0                1                1
187         X-MOTORS                0.0                0               15
188            XGJAO               87.5                7                8
189          XINGYUE               37.5                3                8
190          XINLING                0.0                0                2
191           YAMAHA                2.8              156             5529
192              YMC               22.2                8               36
193          ZHONGYU              100.0                1                1
194             ZNEN               66.7                4                6
195         ZONGSHEN               19.2                5               26
196         ZUENDAPP               42.9                3                7
197          ZUNDAPP               32.3               20               62
198             ΑΛΛΟ               35.7              121              339

>
```

List of the 198 motorcycle brands and their respective percentage of negotiable listings registered in the database

Figure 1 - SEMOG brand is the brand with the highest average price if ATVs are included

Figure 2 - INDIAN brand is the brand with the highest average price if ATVs are excluded