
Redis Project: Relational databases & Key-Value systems

Efstratios Gounidellis
stratos.gounidellis [at] gmail.com

Lamprini Koutsokera
lkoutsokera [at] gmail.com

Course: "Big Data Management Systems"
Professor: Damianos Chatziantoniou

Department of Management Science & Technology

School of Business
Athens University of Economics & Business

April 19, 2017

Contents

1	Introduction	3
2	Relational data insertion in Redis database	3
2.1	redisTableParser.py	3
3	SQL query execution in Redis database	6
3.1	redisQueryParser.py	6
4	Unit testing	20
4.1	testRedisQueryParser.py	20
	References	22

1 Introduction

This assignment is a part of a project implemented in the context of the course "Big Data Management Systems" taught by Prof. Chatziantoniou in the Department of Management Science and Technology (AUEB). The aim of the project is to familiarize the students with big data management systems such as Hadoop, Redis, MongoDB and Neo4j.

In the context of this assignment on Redis, relational data are inserted into a redis database while sql queries are properly edited and transformed in order to retrieve information from the redis database.

2 Relational data insertion in Redis database

2.1 redisTableParser.py

A relation's schema and its contents are given in a text file in a specific format according to the following rules:

1. the first line contains only the table's name.
2. the second line contains the primary key's name, which is only a single attribute.
3. the rest of the attributes are in a single line each.
4. one line containing the character ";" follows.
5. the following line(s), represent records and are delimited by the character ";".

It is assumed that all attributes are of type string.

SQL Table - *Student*

SSN	FName	LName	Address	Age
12938	Nikos	Papadopoulos	Hydras 28, Athens	42
18298	Maria	Nikolaou	Kifisia 33, Marousi	34
81129	Dimitris	Panagiotou	Alamanas 44, Petralona	29

SQL Table in text file *Student*

```
Student
SSN
FName
LName
Address
Age
;
12938;Nikos;Papadopoulos;Hydras 28, Athens;42
18298;Maria;Nikolaou;Kifisia 33, Marousi;34
81129;Dimitris;Panagiotou;Alamanas 44, Petralona;29
```

The relational data will be inserted in the redis database using the following python script. The script is effective for the following cases:

1. The text file follows the structure described above.

2. The primary key is a single attribute.

```
1 # pylint: disable=invalid-name, anomalous-backslash-in-string
2 """
3     redisTableParser.py: Create a table in the Redis
4     database.
5 """
6
7 import argparse
8 import os.path
9 import redis
10
11 __author__ = "Stratos Gounidellis, Lamprini Koutsokera"
12 __copyright__ = "Copyright 2017, BDSMasters"
13
14
15 class RedisTableParser(object):
16     """RedisTableParser: Implementation of the methods needed
17         to successfully create a table in the Redis database.
18     """
19
20     def sqlTableToRedis(self, tableFile):
21         """Create a Redis Table parsing data from an SQL Table
22             through a file.
23
24             :param self: An instance of the class RedisTableParser.
25             :param tableFile: A file that contains data from an SQL
26                 Table.
27         """
28
29         r = redis.StrictRedis(host='localhost', port=6379, db=0)
30         with open(tableFile, "r") as inputFile:
31             input_data = inputFile.readlines()
32             try:
33                 flag_fields = True
34                 table = input_data.pop(0).replace("\n", "")
35                 tableId = table + "Id"
36                 if r.get(tableId) is None:
37                     r.set(tableId, 1)
38                 fields = []
39                 print
40
41                 for string in input_data:
42                     if not flag_fields and string.rstrip():
43                         self.recordsInsertion(r, string, fields, table, tableId)
44                     if flag_fields and string.rstrip():
45                         if string.replace("\n", "") == ";":
46                             flag_fields = False
47                         else:
48                             fields.append(string.replace("\n", ""))
49
50             except redis.exceptions.ConnectionError:
51                 print "\nRedis connection error! " + \
```

```

51         "Check that redis server is on and working.\n"
52     quit()
53 except redis.exceptions.ResponseError:
54     print "\nRedis response error! " + \
55         "Check that redis' configuration!"
56     quit()
57
58 @staticmethod
59 def recordsInsertion(r, string, fields, table, tableId):
60     """Insert in redis database the records.
61
62     :param r: An instance of connection to redis.
63     :param string: A string delimited with ";",
64                     containing a record.
65     :param fields: The attributes of the table.
66     :param table: The name of the table to be inserted.
67     :param tableId: The table counter.
68     """
69
70     counter = 1
71     checkExists = False
72     string = string.replace("\n", " ")
73     string = string.split(";")
74     for field, record in zip(fields, string):
75         if counter == 1:
76             if record in r.smembers(table + "_PrimaryKeys"):
77                 checkExists = True
78                 print table + " with " + field + ":" + \
79                     record + " already exists!"
80                 break
81             else:
82                 r.sadd(table + "_PrimaryKeys", record)
83                 counter += 1
84                 record_key = table + "_" + field + "_" + r.get(tableId)
85                 r.set(record_key, record)
86         if not checkExists:
87             r.incr(tableId)
88
89 if __name__ == "__main__":
90
91     parser = argparse.ArgumentParser(description="Insert relational data" +
92                                     " in a redis database.",
93                                     epilog="Go ahead and try it!")
94     parser.add_argument("inputFile", type=str,
95                         help="Input file with the sql table.")
96     args = parser.parse_args()
97
98     sqlTable = args.inputFile
99
100    if os.path.isfile(sqlTable):
101        instanceRedisTable = RedisTableParser()
102        instanceRedisTable.sqlTableToRedis(sqlTable)
103    else:
104        raise Exception("\nInput file does not exist! \n")

```

```
105     print "\nRelational data have been successfully inserted into Redis!"
```

3 SQL query execution in Redis database

3.1 redisQueryParser.py

A query will be given as a text file containing two to five lines:

1. first line (SELECT): a list of table_name.attribute_name, delimited by the character ",".
2. second line (FROM): a list of table names, delimited by the character ",".
3. third line (WHERE): a simple condition, consisting only of AND, OR, NOT, =, <>, >, <, <=, >= and parentheses.
4. fourth line (ORDER BY): a simple clause, containing either an attribute name and the way of ordering (ASC or DESC) or RAND().
5. fifth line (LIMIT): a number, specifying the number of rows to be displayed.

SQL Query - *Student, Grade*

```
SELECT Student.FName, Student.LName, Grade.Mark  
FROM Student, Grade  
WHERE Student.SSN=Grade.SSN  
ORDER BY Student.Age ASC  
LIMIT 2
```

SQL Query in text file - *Student, Grade*

```
Student.FName, Student.LName, Grade.Mark  
Student, Grade  
Student.SSN=Grade.SSN  
Student.Age ASC  
2
```

The sql query is transformed into proper python code using the following script. The script is effective for the following cases:

1. The text file follows the structure described above.
2. The ORDER BY clause contains only one attribute.
3. The sql query is correct according to the sql syntax.
4. The names of the tables and the attributes are correct.
5. In case a clause is skipped then the corresponding line remains blank, like the example below.

SQL Query without WHERE - *Student, Grade*

```

SELECT Student.FName, Student.LName, Grade.Mark
FROM Student, Grade
ORDER BY Student.Age ASC
LIMIT 2

```

SQL Query without WHERE in text file - *Student, Grade*

```

Student.FName, Student.LName, Grade.Mark
Student, Grade

```

```

Student.Age ASC
2

```

```

1 # pylint: disable=invalid-name, anomalous-backslash-in-string
2 """
3     redisQueryParser.py: Implement an SQL query in the Redis
4     database.
5 """
6
7 import argparse
8 import os.path
9 import re
10 import sys
11 sys.tracebacklimit = 0
12
13 __author__ = "Stratos Gounidellis, Lamprini Koutsokera"
14 __copyright__ = "Copyright 2017, BDSMasters"
15
16 SPECIAL_CHARS = ["==", "!=" , ">" , "<" , ">=" , "<="]
17
18
19 class RedisQueryParser(object):
20     """RedisQueryParser: Implementation of the methods needed
21         to successfully retrieve the expected results from the
22         Redis database.
23     """
24
25     @staticmethod
26     def checkNumeric(inputString):
27         """Check whether a given string is numeric or not.
28
29             :param inputString: A string from the query text file.
30             :return: True, if the inputString is numeric.
31                 Otherwise, return False.
32         """
33         try:
34             float(inputString)
35             return True
36         except ValueError:
37             pass
38
39         try:
40             import unicodedata

```

```

41         unicodedata.numeric(inputString)
42         return True
43     except (TypeError, ValueError):
44         pass
45
46     return False
47
48 @staticmethod
49 def parseSqlQuery(queryFile):
50     """Determine the clauses included in the query text file.
51
52     :param queryFile: A file with the query clauses.
53     :return: A tuple with the different clauses.
54     """
55
56     with open(queryFile, "r") as inputFile:
57         input_data = inputFile.readlines()
58         selectQuery = input_data.pop(0).replace("\n", "").replace(".", "_")
59         fromQuery = input_data.pop(0).replace("\n", "")
60         whereQuery = ""
61         if len(input_data) >= 1:
62             whereQuery = input_data.pop(0).replace("\n", "")
63             if whereQuery.rstrip():
64                 whereQuery = whereQuery.replace(".", "_").strip()
65             whereQuery = whereQuery.replace("(", "(").replace(")", ")")
66         orderQuery = ""
67         if len(input_data) >= 1:
68             orderQuery = input_data.pop(0).replace("\n", "")
69             if orderQuery.rstrip():
70                 orderQuery = orderQuery.replace(".", "_").strip()
71         limitQuery = None
72         if len(input_data) >= 1:
73             limitQuery = input_data.pop(0).replace("\n", "")
74             if limitQuery.rstrip():
75                 limitQuery = limitQuery.strip()
76             else:
77                 limitQuery = None
78     return selectQuery, fromQuery, whereQuery, orderQuery, limitQuery
79
80 @staticmethod
81 def convertToRedisWhere(whereQuery, startString,
82                         endString, flag=True, forCheck=None):
83     """Tailor the WHERE clause according to the syntax and the logic
84     of Python.
85
86     :param whereQuery: A string with the WHERE clause.
87     :param startString: A string with the character(-s) the
88         search term should start.
89     :param endString: A string with the character(-s) the
90         search term should end.
91     :param flag: Boolean variable to check whether the search term
92         has already been tailored.
93     :param forCheck: Either None or a List with the tables in
94         the FORM clause of the query.
95     :return: A string with the transformed WHERE clause.

```

```

95 """
96 whereQuery = " " + whereQuery + " "
97 if flag:
98     indexesStart = sorted([m.start() for m
99                         in re.finditer(startString, whereQuery)])
100 else:
101     indexesStart = sorted([m.end() for m
102                         in re.finditer(startString, whereQuery)])
103 indexesEnd = sorted([m.start() for m
104                     in re.finditer(endString, whereQuery)])
105 dictString = {}
106
107 for start in indexesStart:
108     for end in indexesEnd:
109         flag = False
110         if start < end:
111             newString = whereQuery[start:end].strip()
112             if (not re.search(r"\s", newString) and
113                 len(newString) > 1 and not
114                 re.search(r"r.get", newString)):
115                 if forCheck is not None:
116                     for clause in forCheck:
117                         if clause in newString:
118                             flag = True
119                             break
120             if flag:
121                 newQueryString = 'r.get(' + newString + ')'
122                 dictString[newString] = newQueryString
123             else:
124                 newQueryString = 'r.get(' + newString + ')'
125                 dictString[newString] = newQueryString
126         for key, value in dictString.items():
127             whereQuery = whereQuery.replace(key, value)
128     return whereQuery.strip()
129
130 def convertStringToNumber(self, whereQuery, startString, endString):
131     """Tailor the WHERE clause according to the syntax and the logic
132         of Python (numeric values).
133
134     :param self: An instance of the class RedisQueryParser.
135     :param whereQuery: A string with the WHERE clause.
136     :param startString: A string with the character(-s) the
137         search term should start.
138     :param endString: A string with the character(-s) the
139         search term should end.
140     :return: A string with the transformed WHERE clause, based on the
141         numeric values.
142     """
143     whereQuery = " " + whereQuery + " "
144     indexesStart = sorted([m.end() for m
145                         in re.finditer(startString, whereQuery)])
146     indexesEnd = sorted([m.start() for m
147                         in re.finditer(endString, whereQuery)])
148     dictReplaceAfter = {}

```

```

149     for start in indexesStart:
150         for end in indexesEnd:
151             if start < end:
152                 newString = whereQuery[start:end].strip()
153                 if (not re.search(r"\s", newString) and
154                     len(newString) > 0):
155                     if self.checkNumeric(newString):
156                         if (newString not in dictReplaceAfter.keys() and
157                             not re.search(r"float", newString)):
158                             dictReplaceAfter[start] = end
159             counter = 0
160             dictReplaceAfterNew = {}
161             for i in sorted(dictReplaceAfter.keys()):
162                 whereQuery = whereQuery[0:i + counter] + "float(" + \
163                 whereQuery[i+counter:dictReplaceAfter.get(i)+counter] + ")" + \
164                 whereQuery[dictReplaceAfter.get(i)+counter:]
165             dictReplaceAfterNew[i + counter] = dictReplaceAfter.get(i)+counter
166             counter += 7
167
168     return self.checkNumericBeforeOperator(dictReplaceAfterNew,
169                                         whereQuery, startString)
170
171 @staticmethod
172 def checkNumericBeforeOperator(dictReplaceAfterNew, whereQuery,
173                                 startString):
174     """Tailor the WHERE clause according to the syntax and the logic
175     of Python (numeric values).
176
177     :param dictReplaceAfterNew: A dictionary with the indexes of the
178         numeric values found in the WHERE clause.
179     :param whereQuery: A string with the WHERE clause.
180     :param startString: A string with the character(-s) the
181         search term should start.
182     :return: A string with the transformed WHERE clause, based on the
183         numeric values.
184     """
185     dictReplaceBefore = {}
186     for end in sorted(dictReplaceAfterNew.keys()):
187         indexesStartNumeric = \
188             sorted([m.start() for m
189                   in re.finditer("r.get", whereQuery)])
190         for startNumeric in indexesStartNumeric:
191             if startNumeric < end - len(startString):
192                 newStringNumeric = \
193                     whereQuery[startNumeric:
194                             end - len(startString)].strip()
195                 checkStringNumeric = \
196                     whereQuery[(startNumeric - 6):
197                             end - len(startString)].strip()
198
199                 if (not re.search(r"float",
200                                 checkStringNumeric) and
201                     not re.search(r"\s",

```

```

202                                     newStringNumeric) and
203                                         len(newStringNumeric) > 0):
204                                         dictReplaceBefore[
205                                             startNumeric] = end - len(startString)
206                                         counter = 0
207                                         for i in sorted(dictReplaceBefore.keys()):
208                                             whereQuery = whereQuery[0:i + counter] + "float(" + \
209                                                 whereQuery[i+counter:dictReplaceBefore.get(i)+counter] + \
210                                                 ") " + whereQuery[dictReplaceBefore.get(i)+counter:]
211                                         counter += 7
212
213                                         return whereQuery.strip()
214
215 @staticmethod
216 def selectFromToRedis(selectQuery, fromQuery, whereQuery,
217                         selectQuerySplitOrder):
218     """Parse and edit the SELECT and FROM clauses in order to be
219     translated
220         to python according to its syntax and logic rules.
221
222     :param selectQuery: A string with the SELECT clause.
223     :param fromQuery: A list with the tables in the FROM clause.
224     :param whereQuery: A string with the WHERE clause.
225     :param selectQuerySplitOrder: A list with the attributes included in
226         the ORDER BY clause.
227     :return: A tuple with the string including the lists to be created,
228             the updated "SELECT" clause, the attributes that should be
229             retrieved from redis (and their number) that are not included
230             in the SELECT clause but they are included in the WHERE clause
231             and the attributes that should be retrieved from redis.
232     """
233     selectFromString = ""
234     selectQuerySplit = selectQuery.split(",")
235     selectQuerySplit = map(str.strip, selectQuerySplit)
236     for order in selectQuerySplitOrder:
237         if order not in selectQuerySplit:
238             selectQuerySplit.append(order)
239
240     counterWhere = 0
241     for i, _ in enumerate(fromQuery):
242         pattern = r"(" + fromQuery[i] + ".)\w+"
243         matches = re.findall(pattern, whereQuery)
244         for _, match in enumerate(matches):
245             if match.group().replace(".", "_") not in selectQuerySplit:
246                 selectQuerySplit.append(match.group().replace(".", "_"))
247                 selectQuery += ", " + match.group().replace(".", "_")
248                 counterWhere += 1
249
250     keysList = ""
251     for i, _ in enumerate(selectQuerySplit):
252         if i == len(selectQuerySplit) - 1:
253             keysList += selectQuerySplit[i].strip() + "_List"
254             selectFromString = selectFromString + \
255                 selectQuerySplit[i].strip() + \

```

```

255         "_List = sorted(r.keys(pattern=' " + \
256             selectQuerySplit[i].strip() + "'))\n"
257     else:
258         keysList += selectQuerySplit[i].strip() + "_List, "
259         selectFromString = selectFromString + \
260             selectQuerySplit[i].strip() + \
261             "_List = sorted(r.keys(pattern=' " + \
262                 selectQuerySplit[i].strip() + "'))\n\t"
263     selectFromString += "\n\t"
264     return selectFromString, selectQuery, keysList, counterWhere, \
265         selectQuerySplit
266
267 @staticmethod
268 def orderQueryToRedis(orderQuery, selectQuery):
269     """Parse and edit the ORDER clause in order to be translated
270     to python according to its syntax and logic rules.
271
272     :param orderQuery: A string with the ORDER clause.
273     :param selectQuery: A string with the SELECT clause.
274
275     :return: A tuple with the field according to which the results will
276             be ordered, a variable to check whether the order will
277             be ascending or descending, the updated "SELECT" clause and a
278             variable to check whether the order field is included in the
279             SELECT
280             clause or not.
281     """
282     orderQuery = " " + orderQuery + " "
283     orderTypes = ["asc", "desc"]
284     orderFlag = 1
285     for orderType in orderTypes:
286         indexesStart = sorted(
287             [m.start() for m in
288                 re.finditer("(?i)" + orderType,
289                             orderQuery)])
290     for start in indexesStart:
291         if orderQuery[start - 1:start] is " " \
292             and orderQuery[start + len(orderType):start +
293                             len(orderType) + 1] is " ":
294             if orderQuery[start:start + len(orderType)].lower() == \
295                 "desc":
296                 orderFlag = 0
297             orderQuery = orderQuery[0:start] + \
298                         orderQuery[start + len(orderType):]
299
300     orderField = orderQuery.strip().replace(".", "_")
301
302     selectQuerySplit = []
303     orderFieldExists = True
304     if orderField not in selectQuery:
305         selectQuerySplit.append(orderField)
306         selectQuery += ", " + orderField
307         orderFieldExists = False

```

```

308
309     return orderField, orderFlag, selectQuery, selectQuerySplit, \
310         orderFieldExists
311
312 def whereToRedis(self, fromQuery, whereQuery):
313     """Parse and edit the WHERE clause in order to be translated
314         to python according to its syntax and logic rules.
315
316     :param self: An instance of the class RedisQueryParser.
317     :param fromQuery: A list with the tables in the FROM clause.
318     :param whereQuery: A string with the WHERE clause.
319
320     :return: A string with the python-like WHERE clause.
321     """
322     specialCharsWhere = []
323     indexesStart = sorted([m.start() for m
324                           in re.finditer("=", whereQuery)])
325     counterEqual = 0
326     for i in indexesStart:
327         i += counterEqual
328         if whereQuery[i - 1:i] is not "<" and whereQuery[i - 1:i] \
329             is not ">":
330             whereQuery = whereQuery[0:i] + "==" + whereQuery[i+1:]
331             counterEqual += 1
332     whereQuery = whereQuery.replace("<>", "!=")
333     for char in SPECIAL_CHARS:
334         if char in whereQuery:
335             specialCharsWhere.append(char)
336
337     whereQuery = ' '.join(whereQuery.split())
338     for char in specialCharsWhere:
339         whereQuery = whereQuery.replace(" " + char + " ", char)
340         whereQuery = whereQuery.replace(char + " ", char)
341         whereQuery = whereQuery.replace(" " + char, char)
342
343     for char in specialCharsWhere:
344         whereQuery = self.convertToRedisWhere(whereQuery, " ", char)
345         whereQuery = self.convertToRedisWhere(
346             whereQuery, char, " ", False, fromQuery)
347
348     for char in specialCharsWhere:
349         whereQuery = self.convertStringToNumber(whereQuery, char, " ")
350     whereQuery = ' '.join(whereQuery.split())
351     whereQuery = re.sub(r'\b(?i)AND\b', ' and ', whereQuery)
352     whereQuery = re.sub(r'\b(?i)OR\b', ' or ', whereQuery)
353     whereQuery = re.sub(r'\b(?i)NOT\b', ' not ', whereQuery)
354     whereQuery = whereQuery.replace("( ", "(").replace(")", ")")
355     for char in specialCharsWhere:
356         whereQuery = whereQuery.replace(char, " " + char + " ")
357         whereQuery = whereQuery.replace("< =", "<=").replace("> =", ">=") \
358             .strip()
359     whereQuery = ' '.join(whereQuery.split())
360     whereString = "if " + whereQuery + ":\\n\\t\\t"
361
362     return whereString

```

```

362
363     @staticmethod
364     def pythonFileInitialize():
365         """Initialize the python file to be created with some
366             basic imports and methods' calls.
367
368         :return: A string with initialization of the python file.
369         """
370
371         pythonFile = "import argparse\nimport numpy as np\nimport " + \
372             "pandas as pd\nimport redis\n" + \
373             "from tabulate import tabulate\n\n"
374         pythonFile += "r = redis.StrictRedis" + \
375             "(host='localhost', port=6379, db=0)\n\n"
376         pythonFile += "parser = argparse.ArgumentParser(description=" + \
377             "'Execute a simple SQL query in a redis database and save" + \
378             " output in a .csv file')\n"
379         pythonFile += "parser.add_argument('outputFile', type=str," + \
380             " help='Output .csv file with the query results. ')\n"
381         pythonFile += "args = parser.parse_args()\n" + \
382             "resultsFile = args.outputFile\n"
383         pythonFile += "if not resultsFile.endswith('.csv'): \n\t" + \
384             "print '\\nOutput file should end with .csv!'\n\t" + \
385             "quit()\n\ntry:\n\t"
386
387         return pythonFile
388
389     @staticmethod
390     def pythonFileArrayResults(selectQuerySplit, whereQuery, counterTab):
391         """Create the content of the python file responsible for
392             saving the results properly in a numpy array.
393
394         :param selectQuerySplit: A list with the attributes in the
395             SELECT clause.
396         :param whereQuery: A string with the WHERE clause.
397
398         :return: A string with the content of the python file,
399             which will save the results of the query in a numpy
400             array.
401         """
402
403         resultsString = ("\t" * (counterTab - 1)) + "tempResults = np.array([" + \
404             columnNames = ""
405         for i, _ in enumerate(selectQuerySplit):
406             if i == len(selectQuerySplit) - 1:
407                 if len(whereQuery) == 0:
408                     resultsString = resultsString + "r.get(" + \
409                         selectQuerySplit[i].strip() + \
410                         "])\n" + ("\t" * (counterTab + 1))
411             else:
412                 resultsString = resultsString + "r.get(" + \
413                     selectQuerySplit[i].strip() + "])\n" + \
414                     ("\t" * (counterTab + 1))
415             columnNames += "!" + selectQuerySplit[i].strip() + "!"
416         else:

```

```

416         resultsString += "r.get(" + selectQuerySplit[i].strip() + "), "
417         columnNames += " '" + selectQuerySplit[i].strip() + "', "
418
419     if counterTab == 0:
420         resultsString += "\t"
421     resultsString += "resultsArray = np.vstack((tempResults," + \
422         " resultsArray))\n"
423     resultsString = resultsString + "except NameError, e:\n\tprint" + \
424         "'\\nCheck " + \
425         "that all tables required are included in the FROM clause!\\n'" +
426         \
427         "\n\t" + \
428         "print e.message\n\tquit()\n"
429     resultsString = resultsString + "except ValueError, e:\n\tprint" + \
430         "'\\nCheck that the value types of the WHERE clause are " + \
431         "consistent with the value types of the attributes!\\n'\n\t" + \
432         "print e.message\n\tquit()\n"
433     resultsString += "except redis.exceptions.ConnectionError" + \
434         ":\n\tprint '\\nRedis connection error! Check that " + \
435         "Redis server is on and properly working!'\n\tquit()\n\n"
436     resultsString = resultsString + "try:\n\tif resultsArray.size > " + \
437         str(len(selectQuerySplit)) + ":\n\t\t"
438     resultsString += "resultsArray = resultsArray[:-1, :]\n\t\t"
439
440     return resultsString, columnNames
441
442 @staticmethod
443 def pythonFileForLoop(selectQuerySplit, selectQuery,
444                         keysList, fromQuery):
445     """Construct the main for loop of the output python file,
446     in order to iterate over the results retrieved from
447     the Redis database.
448
449     :param selectQuerySplit: A list with the attributes in the
450         SELECT clause.
451     :param selectQuery: A string with the SELECT clause.
452     :param counterWhere: The number of attributes contained in
453         the WHERE clause but not in the SELECT clause.
454     :param keysList: A string with the necessary content
455         to iterate over the different attributess.
456
457     :return: A string with the content of the python file,
458         which will iterate over the results.
459     """
460     selectQuery = selectQuery.split(",")
461     selectQuery = map(str.strip, selectQuery)
462     forString = "resultsArray = np.zeros(" + \
463         str(len(selectQuerySplit)) + ")\n\n"
464
465     newKeysList = ''.join(map(str, keysList))
466     newKeysList = newKeysList.split(",")
467     newKeysList = map(str.strip, newKeysList)
468     counterTab = 1

```

```

468     for fromClause in fromQuery:
469         forString += '\t' * counterTab
470         forString += "for "
471         for selectClause in selectQuery:
472             if fromClause in selectClause:
473                 forString += selectClause + ", "
474         forString = forString.strip()
475         forString = forString[:-1]
476
477     keysForList = []
478     for key in newKeysList:
479         if fromClause in key:
480             keysForList.append(key)
481
482     keysForString = ', '.join(map(str, keysForList))
483     if len(keysForList) == 1:
484         forString += " in " + keysForString + ":\n"
485     else:
486         forString += " in zip(" + keysForString + "):\n"
487         counterTab += 1
488     forString += '\t' * counterTab
489     return forString, counterTab
490
491 @staticmethod
492 def pythonFileLimitOrderQuery(
493     orderQuery, orderFlag, limitQuery,
494     orderField, orderFieldExists, randomCheck):
495     """Construct the main for loop of the ouput python file,
496     in order to iterate over the results retrieved from
497     the Redis database.
498
499     :param orderQuery: A string with the ORDER clause.
500     :param orderFlag: A boolean variable to check whether the
501         ordering will be ascending or descending.
502     :param limitQuery: A string with the LIMIT clause, i.e.
503         the number of results to be printed.
504     :param orderField: The field according to which the
505         results will be ordered.
506     :param orderFieldExists: A boolean variable to check whether the
507         ordering field is included also in the SELECT clause or not.
508     :param randomCheck: A boolean variable to check whether the
509         results should be printed in random order.
510
511     :return: A string with the content of the python file,
512         related mainly with the formatting of the way the results
513         are printed.
514 """
515     limitOrderString = ""
516     if len(orderQuery) > 0 and not randomCheck:
517         limitOrderString += "if dfResults['" + str(orderField) + \
518             "'].dtype == 'object':\n\t\t\tdfResults['sortColumn'] " + \
519             "= dfResults['" + str(orderField) + "'].str.lower()\n\t\t" + \
520             "\tdfResults.sort_values(by='sortColumn', ascending=" + \
521             str(orderFlag) + \

```

```

522     ", inplace=True)\n\t\t\tdfResults.drop('" + \
523     "sortColumn', axis=1, inplace=True)\n\t\t\t"
524 limitOrderString += "else:\n\t\t\tdfResults.sort_values" + \
525     "(by='" + orderField + "', ascending=" + str(orderFlag) + \
526     ", inplace=True)\n\t\t\t"
527
528     if not orderFieldExists:
529         limitOrderString = limitOrderString + "dfResults.drop('" + \
530             orderField + "', axis=1, inplace=True)\n\t\t\t"
531     if limitQuery is not None:
532         limitOrderString += "dfResults = dfResults.head(n=" + \
533             str(limitQuery) + ")\n\t\t\t"
534     if randomCheck:
535         if limitQuery is not None:
536             limitOrderString = limitOrderString.replace(
537                 "dfResults.head(n=" + str(limitQuery),
538                 "dfResults.sample(n=min(" + str(limitQuery) + \
539                 ", dfResults.shape[0]))")
540         else:
541             limitOrderString += \
542                 "dfResults = dfResults.sample(n=dfResults.shape[0])\n\t\t\t"
543     limitOrderString += "dfResults = dfResults.reset_index(drop=True" + \
544     ")\n\t\t\t"
545
546     limitOrderString += "try:\n\t\t\t"
547     limitOrderString += \
548         "print tabulate(dfResults, headers='keys', " + \
549         "tablefmt='fancy_grid')\n\t\t\t"
550     limitOrderString += "except UnicodeEncodeError:\n\t\t\t" + \
551         "print\n\t\t\tprint dfResults\n\t\t\ttpass\n\t\t\t"
552     limitOrderString += "print '\\nTotal rows: ', dfResults.shape[0]\n\t\t\t"
553
554     limitOrderString = limitOrderString + \
555         "dfResults.to_csv(resultsFile, index=False, sep=';')\n\t\t\t"
556     limitOrderString += "print 'The results have been saved in'" \
557         ", resultsFile\n\t\t\t"
558     limitOrderString += "else:\n\t\t\tprint '\\nNo results found. " + \
559         "Try another query! \\nHint: Check the names of the attributes" + \
560
561         " in the SELECT, the WHERE and the ORDER BY clauses ;)'\n\t\t\t"
562     limitOrderString = limitOrderString + "except KeyError:\n\t\t\t" + \
563         "'Check that the ORDER BY clause contains only one field!'\n\t\t\t"
564
565     return limitOrderString
566
567 def sqlQueryToRedis(self, selectQuery, fromQuery, whereQuery, orderQuery,
568                     limitQuery):
569     """Call the methods required to build the output file.
570
571     :param self: An instance of the class RedisQueryParser.
572     :param selectQuery: A string with the SELECT clause.
573     :param fromQuery: A string with the FROM clause.
574     :param whereQuery: A string with the WHERE clause.
575     :param orderQuery: A string with the ORDER clause.

```

```

574     :param limitQuery: A string with the LIMIT clause.
575
576     :return: A string with the final complete content of the
577             python file.
578     """
579
580     pythonFile = self.pythonFileInitialize()
581     fromQuery = fromQuery.split(",")
582     fromQuery = map(str.strip, fromQuery)
583     fromQuery = [s + " " for s in fromQuery]
584
585     selectQuerySplit = []
586     orderField = ""
587     orderFlag = 1
588     orderFieldExists = True
589     randomOrder = re.search("(?i)RAND\(\)", orderQuery.strip())
590     if randomOrder is not None:
591         randomOrder = randomOrder.group().upper()
592
593     randomCheck = True
594     if randomOrder != "RAND()":
595         randomCheck = False
596     if len(orderQuery) > 0 and randomOrder != "RAND()":
597         orderField, orderFlag, selectQuery, selectQuerySplit, \
598             orderFieldExists = self.orderQueryToRedis(
599                 orderQuery, selectQuery)
600
601     selectFromString, selectQuery, keysList, counterWhere, \
602         selectQuerySplit = \
603             self.selectFromToRedis(
604                 selectQuery, fromQuery, whereQuery, selectQuerySplit)
605     pythonFile += selectFromString
606
607     for _ in range(counterWhere):
608         selectQuerySplit.pop(-1)
609
610     forString, counterTab = self.pythonFileForLoop(
611         selectQuerySplit, selectQuery, keysList, fromQuery)
612
613     pythonFile += forString
614
615     if len(whereQuery) > 0:
616         pythonFile += self.whereToRedis(fromQuery, whereQuery)
617     if len(whereQuery) == 0:
618         counterTab = 0
619     resultsString, columnNames = self.pythonFileArrayResults(
620         selectQuerySplit, whereQuery, counterTab)
621     pythonFile += resultsString
622     if len(selectQuerySplit) == 1:
623         pythonFile += "dfResults = pd.DataFrame(data=resultsArray)\n\t\t"
624     else:
625         pythonFile = pythonFile + "dfResults = pd.DataFrame(data=" + \
626             "resultsArray, columns=(" + columnNames + "))\n\t\t"
627
628     if len(selectQuerySplit) == 1:

```

```

628     pythonFile = pythonFile + "dfResults.rename(columns={0:'" + \
629         str(selectQuerySplit[0]) + "'},inplace=True)\n\t\t"
630
631     limitOrderString = self.pythonFileLimitOrderQuery(
632         orderQuery, orderFlag, limitQuery, orderField,
633         orderFieldExists, randomCheck)
634
635     pythonFile += limitOrderString
636     return pythonFile.replace("\t", "    ")
637
638 @staticmethod
639 def checkSyntax(outputPython):
640     """Check the syntax of the created python file.
641
642     :param outputFile: The name of the output file to be created.
643     """
644     fileCompile = outputPython + "c"
645
646     if os.path.isfile(fileCompile):
647         os.remove(fileCompile)
648     os.popen('python -m py_compile ' + outputPython)
649     if not os.path.isfile(fileCompile):
650         os.remove(outputPython)
651         raise Exception('\nERROR! Please check the syntax of the ' +
652                         'query. Output python file is not created! :(')
653     print '\nSuccess! Python file has been successfully created!\n' + \
654         '\nRun it by typing:\n\t python ' + outputPython
655     os.remove(fileCompile)
656
657 @staticmethod
658 def writePythonFile(outputFile, sourceCode):
659     """Write the source code on the python file specified.
660
661     :param outputFile: The name of the output file to be created.
662     :param sourceCode: The source code to be written in the output
663         python file.
664     """
665     f = open(outputFile, "w+")
666     f.write(sourceCode)
667     f.close()
668
669
670 if __name__ == "__main__":
671
672     parser = argparse.ArgumentParser(description="Execute a simple SQL" +
673                                     " query in a redis database.",
674                                     epilog="Go ahead and try it at " +
675                                         " your own risk :)")
676     parser.add_argument("inputFile", type=str,
677                         help="Input file with the sql query.")
678     parser.add_argument("outputFile", type=str,
679                         help="Output python file executing the sql query.")
680     args = parser.parse_args()
681

```

```

682     sqlQuery = args.inputFile
683     outputPython = args.outputFile
684
685     if not os.path.isfile(sqlQuery):
686         print "\nInput file does not exist!"
687         quit()
688
689     if not outputPython.endswith(".py"):
690         print "\nOutput file should end with .py!"
691         quit()
692
693     instanceRedisQuery = RedisQueryParser()
694     sqlClauses = instanceRedisQuery.parseSqlQuery(sqlQuery)
695     pythonFileContent = instanceRedisQuery.sqlQueryToRedis(
696         sqlClauses[0], sqlClauses[1], sqlClauses[2], sqlClauses[3],
697         sqlClauses[4])
698     instanceRedisQuery.writePythonFile(outputPython, pythonFileContent)
699     instanceRedisQuery.checkSyntax(outputPython)

```

4 Unit testing

4.1 testRedisQueryParser.py

```

1 # pylint: disable=invalid-name, anomalous-backslash-in-string
2 """
3     testRedisQueryParser.py: Test the results' validity of the SQL
4     Query Parsing.
5 """
6
7 import unittest
8 from redisQueryParser import RedisQueryParser
9
10 __author__ = "Stratos Gounidellis, Lamprini Koutsokera"
11 __copyright__ = "Copyright 2017, BDSMasters"
12
13
14 class TestRedisQueryParser(unittest.TestCase):
15     """TestRedisQueryParser: Implementation of the methods needed
16         to successfully test the expected results from the
17         SQL Query Parsing.
18     """
19
20     def test_readSqlQuery(self):
21         """Test whether a given query is read correctly or not.
22         """
23         instanceQueryParser = RedisQueryParser()
24         fname = "redisQuery1.txt"
25         clauses = instanceQueryParser.parseSqlQuery(fname)
26
27         expectedClauses = ["Student_FName, Student_LName, Grade_Mark"]
28         expectedClauses.append("Student, Grade")
29         expectedClauses.append("Student_SSN=Grade_SSN")

```

```

30     expectedClauses.append(" ")
31     expectedClauses.append(None)
32
33     self.assertEqual(clauses, tuple(expectedClauses))
34
35 def test_selectFromToRedis(self):
36     """Test whether the SELECT clause is converted correctly or not.
37     """
38     instanceQueryParser = RedisQueryParser()
39     fname = "redisQuery1.txt"
40     clauses = instanceQueryParser.parseSqlQuery(fname)
41     selectQuery = clauses[0]
42     fromQuery = clauses[1]
43     fromQuery = fromQuery.split(",")
44     fromQuery = map(str.strip, fromQuery)
45     fromQuery = [s + "_" for s in fromQuery]
46     whereQuery = clauses[2]
47     selectQuerySplitOrder = []
48
49     results = instanceQueryParser.selectFromToRedis(
50         selectQuery, fromQuery, whereQuery, selectQuerySplitOrder)
51     expectedClauses = "Student_FName_List, Student_LName_List, " + \
52         " Grade_Mark_List, Student_SSN_List, Grade_SSN_List"
53     self.assertEqual(results[2], expectedClauses)
54
55 def test_orderQueryToRedis(self):
56     """Test whether the ORDER BY clause is converted correctly or not.
57     """
58     instanceQueryParser = RedisQueryParser()
59     fname = "redisQuery.txt"
60     clauses = instanceQueryParser.parseSqlQuery(fname)
61     selectQuery = clauses[0]
62     fromQuery = clauses[1]
63     fromQuery = fromQuery.split(",")
64     fromQuery = map(str.strip, fromQuery)
65     fromQuery = [s + "_" for s in fromQuery]
66     orderQuery = clauses[3]
67
68     results = instanceQueryParser.orderQueryToRedis(
69         orderQuery, selectQuery)
70     results = results[:2]
71     expectedClauses = ['Student_FName', 1]
72     self.assertEqual(results, tuple(expectedClauses))
73
74 def test_whereQueryToRedis(self):
75     """Test whether the WHERE clause is converted correctly or not.
76     """
77     instanceQueryParser = RedisQueryParser()
78     fname = "redisQuery.txt"
79     clauses = instanceQueryParser.parseSqlQuery(fname)
80     fromQuery = clauses[1]
81     fromQuery = fromQuery.split(",")
82     fromQuery = map(str.strip, fromQuery)
83     fromQuery = [s + "_" for s in fromQuery]

```

```

84     whereQuery = clauses[2]
85
86     results = instanceQueryParser.whereToRedis(fromQuery, whereQuery)
87     expectedClause = 'if r.get(Student_FName) < "Nikos1":\n\t\t'
88     self.assertEqual(results, expectedClause)
89
90 def test_exceptionSyntaxError(self):
91     """Test whether the syntax of the created python file is correct.
92     """
93     instanceQueryParser = RedisQueryParser()
94     fname = "redisQuery6.txt"
95
96     sqlClauses = instanceQueryParser.parseSqlQuery(fname)
97     pythonFileContent = instanceQueryParser.sqlQueryToRedis(
98         sqlClauses[0], sqlClauses[1], sqlClauses[2], sqlClauses[3],
99         sqlClauses[4])
100    outputFile = "test.py"
101    instanceQueryParser.writePythonFile(outputFile, pythonFileContent)
102
103   with self.assertRaises(Exception) as context:
104       instanceQueryParser.checkSyntax(outputFile)
105   self.assertIn('\nERROR! Please check the syntax of the ' +
106               'query. Output python file is not created! :(',
107               ''.join(context.exception))
108
109
110 if __name__ == "__main__":
111     unittest.main()

```

References

- [1] Peter Cooper. *Redis 101 - A whirlwind tour of the next big thing in NoSQL data storage.* <https://www.scribd.com/document/33531219/Redis-Presentation> [Accessed 12 Apr. 2017].
- [2] Redis.io. *Redis Quick Start.* <https://redis.io/topics/quickstart> [Accessed 12 Apr. 2017].