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

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	Kifisias 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;Kifisias 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.

```
# pylint: disable=invalid-name, anomalous-backslash-in-string
"""
    redisTableParser.py: Create a table in the Redis
    database.
"""

import argparse
import os.path
import redis

__author__ = "Stratos Gounidellis, Lamprini Koutsokera"
__copyright__ = "Copyright 2017, BDSMasters"


class RedisTableParser(object):
    """RedisTableParser: Implementation of the methods needed
        to successfuly create a table in the Redis database.
    """

    def sqlTableToRedis(self, tableFile):
        """Create a Redis Table parsing data from an SQL Table
        through a file.

        :param self: An instance of the class RedisTableParser.
        :param tableFile: A file that contains data from an SQL
            Table.
        """
        r = redis.StrictRedis(host='localhost', port=6379, db=0)
        with open(tableFile, "r") as inputFile:
            input_data = inputFile.readlines()
        try:
            flag_fields = True
            table = input_data.pop(0).replace("\n", "")
            tableId = table + "Id"
            if r.get(tableId) is None:
                r.set(tableId, 1)
            fields = []
            print

            for string in input_data:
                if not flag_fields and string.rstrip():
                    self.recordsInsertion(r, string, fields, table, tableId)
                if flag_fields and string.rstrip():
                    if string.replace("\n", "") == ";":
                        flag_fields = False
                    else:
                        fields.append(string.replace("\n", ""))

        except redis.exceptions.ConnectionError:
            print "\nRedis connection error! " + \
```

```
                "Check that redis server is on and working.\n"
            quit()
        except redis.exceptions.ResponseError:
            print "\nRedis response error! " + \
                "Check that redis' configuration!"
            quit()

    @staticmethod
    def recordsInsertion(r, string, fields, table, tableId):
        """Insert in redis database the records.

        :param r: An instance of connection to redis.
        :param string: A string delimited with ";",
            containing a record.
        :param fields: The attributes of the table.
        :param table: The name of the table to be inserted.
        :param tableId: The table counter.
        """
        counter = 1
        checkExists = False
        string = string.replace("\n", "")
        string = string.split(";")
        for field, record in zip(fields, string):
            if counter == 1:
                if record in r.smembers(table + "_PrimaryKeys"):
                    checkExists = True
                    print table + " with " + field + ": " + \
                        record + " already exists!"
                    break
                else:
                    r.sadd(table + "_PrimaryKeys", record)
                counter += 1
            record_key = table + "_" + field + "_" + r.get(tableId)
            r.set(record_key, record)
        if not checkExists:
            r.incr(tableId)


if __name__ == "__main__":

    parser = argparse.ArgumentParser(description="Insert relational data" +
                                     " in a redis database.",
                                     epilog="Go ahead and try it!")
    parser.add_argument("inputFile", type=str,
                        help="Input file with the sql table.")
    args = parser.parse_args()

    sqlTable = args.inputFile

    if os.path.isfile(sqlTable):
        instanceRedisTable = RedisTableParser()
        instanceRedisTable.sqlTableToRedis(sqlTable)
    else:
        raise Exception("\nInput file does not exist! \n")
```

```
105     print "\nRelational data have been successfuly 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
```

```python
# pylint: disable=invalid-name, anomalous-backslash-in-string
"""
    redisQueryParser.py: Implement an SQL query in the Redis
    database.
"""

import argparse
import os.path
import re
import sys
sys.tracebacklimit = 0

__author__ = "Stratos Gounidellis, Lamprini Koutsokera"
__copyright__ = "Copyright 2017, BDSMasters"

SPECIAL_CHARS = ["==", "!=", ">", "<", ">=", "<="]


class RedisQueryParser(object):
    """RedisQueryParser: Implementation of the methods needed
        to successfuly retrieve the expected results from the
        Redis database.
    """

    @staticmethod
    def checkNumeric(inputString):
        """Check whether a given string is numeric or not.

        :param inputString: A string from the query text file.
        :return: True, if the inputString is numeric.
            Otherwiser, return False.
        """
        try:
            float(inputString)
            return True
        except ValueError:
            pass

        try:
            import unicodedata
```

```
            unicodedata.numeric(inputString)
            return True
        except (TypeError, ValueError):
            pass

        return False

    @staticmethod
    def parseSqlQuery(queryFile):
        """Determine the clauses included in the query text file.

        :param queryFile: A file with the query clauses.
        :return: A tuple with the different clauses.
        """
        with open(queryFile, "r") as inputFile:
            input_data = inputFile.readlines()
        selectQuery = input_data.pop(0).replace("\n", "").replace(".", "_")
        fromQuery = input_data.pop(0).replace("\n", "")
        whereQuery = ""
        if len(input_data) >= 1:
            whereQuery = input_data.pop(0).replace("\n", "")
            if whereQuery.rstrip():
                whereQuery = whereQuery.replace(".", "_").strip()
                whereQuery = whereQuery.replace("(", "( ").replace(")", " )")
        orderQuery = ""
        if len(input_data) >= 1:
            orderQuery = input_data.pop(0).replace("\n", "")
            if orderQuery.rstrip():
                orderQuery = orderQuery.replace(".", "_").strip()
        limitQuery = None
        if len(input_data) >= 1:
            limitQuery = input_data.pop(0).replace("\n", "")
            if limitQuery.rstrip():
                limitQuery = limitQuery.strip()
            else:
                limitQuery = None
        return selectQuery, fromQuery, whereQuery, orderQuery, limitQuery

    @staticmethod
    def convertToRedisWhere(whereQuery, startString,
                            endString, flag=True, forCheck=None):
        """Tailor the WHERE clause according to the syntax and the logic
            of Python.

        :param whereQuery: A string with the WHERE clause.
        :param startString: A string with the character(-s) the
            search term should start.
        :param endString: A string with the character(-s) the
            search term should end.
        :param flag: Boolean variable to check whether the search term
            has already been tailored.
        :param forCheck: Either None or a List with the tables in
            FORM clause of the query.
        :return: A string with the transformed WHERE clause.
```

```
        """
        whereQuery = " " + whereQuery + " "
        if flag:
            indexesStart = sorted([m.start() for m
                                   in re.finditer(startString, whereQuery)])
        else:
            indexesStart = sorted([m.end() for m
                                   in re.finditer(startString, whereQuery)])
        indexesEnd = sorted([m.start() for m
                             in re.finditer(endString, whereQuery)])
        dictString = {}

        for start in indexesStart:
            for end in indexesEnd:
                flag = False
                if start < end:
                    newString = whereQuery[start:end].strip()
                    if (not re.search(r"\s", newString) and
                            len(newString) > 1 and not
                            re.search(r"r.get", newString)):
                        if forCheck is not None:
                            for clause in forCheck:
                                if clause in newString:
                                    flag = True
                                    break
                            if flag:
                                newQueryString = 'r.get(' + newString + ')'
                                dictString[newString] = newQueryString
                        else:
                            newQueryString = 'r.get(' + newString + ')'
                            dictString[newString] = newQueryString
        for key, value in dictString.iteritems():
            whereQuery = whereQuery.replace(key, value)
        return whereQuery.strip()

    def convertStringToNumber(self, whereQuery, startString, endString):
        """Tailor the WHERE clause according to the syntax and the logic
            of Python (numeric values).

        :param self: An instance of the class RedisQueryParser.
        :param whereQuery: A string with the WHERE clause.
        :param startString: A string with the character(-s) the
            search term should start.
        :param endString: A string with the character(-s) the
            search term should end.
        :return: A string with the transformed WHERE clause, based on the
            numeric values.
        """
        whereQuery = " " + whereQuery + " "
        indexesStart = sorted([m.end() for m
                               in re.finditer(startString, whereQuery)])
        indexesEnd = sorted([m.start() for m
                             in re.finditer(endString, whereQuery)])
        dictReplaceAfter = {}
```

```
        for start in indexesStart:
            for end in indexesEnd:
                if start < end:
                    newString = whereQuery[start:end].strip()
                    if (not re.search(r"\s", newString) and
                            len(newString) > 0):
                        if self.checkNumeric(newString):
                            if (newString not in dictReplaceAfter.keys() and
                                    not re.search(r"float", newString)):
                                dictReplaceAfter[start] = end
        counter = 0
        dictReplaceAfterNew = {}
        for i in sorted(dictReplaceAfter.keys()):
            whereQuery = whereQuery[0:i + counter] + "float(" + \
                whereQuery[i+counter:dictReplaceAfter.get(i)+counter] + ")" +
    \
                whereQuery[dictReplaceAfter.get(i)+counter:]
            dictReplaceAfterNew[i + counter] = dictReplaceAfter.get(i)+counter
            counter += 7

        return self.checkNumericBeforeOperator(dictReplaceAfterNew,
                                               whereQuery, startString)

    @staticmethod
    def checkNumericBeforeOperator(dictReplaceAfterNew, whereQuery,
                                   startString):
        """Tailor the WHERE clause according to the syntax and the logic
            of Python (numeric values).

        :param dictReplaceAfterNew: A dictionary with the indexes of the
            numeric values found in the WHERE clause.
        :param whereQuery: A string with the WHERE clause.
        :param startString: A string with the character(-s) the
            search term should start.
        :return: A string with the transformed WHERE clause, based on the
            numeric values.
        """
        dictReplaceBefore = {}
        for end in sorted(dictReplaceAfterNew.keys()):
            indexesStartNumeric = \
                sorted([m.start() for m
                        in re.finditer("r.get", whereQuery)])
            for startNumeric in indexesStartNumeric:
                if startNumeric < end - len(startString):
                    newStringNumeric = \
                            whereQuery[startNumeric:
                                       end - len(startString)].strip()
                    checkStringNumeric = \
                        whereQuery[(startNumeric - 6):
                                   end - len(startString)].strip()

                    if (not re.search(r"float",
                                      checkStringNumeric) and
                            not re.search(r"\s",
```

```
                                            newStringNumeric) and
                            len(newStringNumeric) > 0):
                        dictReplaceBefore[
                            startNumeric] = end - len(startString)
        counter = 0
        for i in sorted(dictReplaceBefore.keys()):
            whereQuery = whereQuery[0:i + counter] + "float(" + \
                whereQuery[i+counter:dictReplaceBefore.get(i)+counter] + \
                ") " + whereQuery[dictReplaceBefore.get(i)+counter:]
            counter += 7

        return whereQuery.strip()

    @staticmethod
    def selectFromToRedis(selectQuery, fromQuery, whereQuery,
                          selectQuerySplitOrder):
        """Parse and edit the SELECT and FROM clauses in order to be
    translated
            to python according to its syntax and logic rules.

        :param selectQuery: A string with the SELECT clause.
        :param fromQuery: A list with the tables in the FROM clause.
        :param whereQuery: A string with the WHERE clause.
        :param selectQuerySplitOrder: A list with the attributes included in
            the ORDER BY clause.
        :return: A tuple with the string including the lists to be created,
            the updated "SELECT" clause, the attributes that should be
            retrieved from redis (and their number) that are not included
            in the SELECT clause but they are included in the WHERE clause
            and the attributes that should be retrieved from redis.
        """
        selectFromString = ""
        selectQuerySplit = selectQuery.split(",")
        selectQuerySplit = map(str.strip, selectQuerySplit)
        for order in selectQuerySplitOrder:
            if order not in selectQuerySplit:
                selectQuerySplit.append(order)

        counterWhere = 0
        for i, _ in enumerate(fromQuery):
            pattern = r"(" + fromQuery[i] + ".)\w+"
            matches = re.finditer(pattern, whereQuery)
            for _, match in enumerate(matches):
                if match.group().replace(".", "_") not in selectQuerySplit:
                    selectQuerySplit.append(match.group().replace(".", "_"))
                    selectQuery += ", " + match.group().replace(".", "_")
                    counterWhere += 1

        keysList = ""
        for i, _ in enumerate(selectQuerySplit):
            if i == len(selectQuerySplit) - 1:
                keysList += selectQuerySplit[i].strip() + "_List"
                selectFromString = selectFromString + \
                    selectQuerySplit[i].strip() + \
```

```
                    "_List = sorted(r.keys(pattern='" + \
                    selectQuerySplit[i].strip() + "*'))\n"
            else:
                keysList += selectQuerySplit[i].strip() + "_List, "
                selectFromString = selectFromString + \
                    selectQuerySplit[i].strip() + \
                    "_List = sorted(r.keys(pattern='" + \
                    selectQuerySplit[i].strip() + "*'))\n\t"
        selectFromString += "\n\t"
        return selectFromString, selectQuery, keysList, counterWhere, \
            selectQuerySplit

    @staticmethod
    def orderQueryToRedis(orderQuery, selectQuery):
        """Parse and edit the ORDER clause in order to be translated
            to python according to its syntax and logic rules.

        :param orderQuery: A string with the ORDER clause.
        :param selectQuery: A string with the SELECT clause.

        :return: A tuple with the field according to which the results will
            be ordered, a variable to check whether the order will
            be ascending or descending, the updated "SELECT" clause and a
            variable to check whether the order field is included in the
    SELECT
            clause or not.
        """
        orderQuery = " " + orderQuery + " "
        orderTypes = ["asc", "desc"]
        orderFlag = 1
        for orderType in orderTypes:
            indexesStart = sorted(
                [m.start() for m in
                 re.finditer("(?i)" + orderType,
                             orderQuery)])
            for start in indexesStart:
                if orderQuery[start - 1:start] is " " \
                        and orderQuery[start + len(orderType):start +
                                       len(orderType) + 1] is " ":

                    if orderQuery[start:start + len(orderType)].lower() == \
                            "desc":
                        orderFlag = 0
                    orderQuery = orderQuery[0:start] + \
                        orderQuery[start + len(orderType):]

        orderField = orderQuery.strip().replace(".", "_")

        selectQuerySplit = []
        orderFieldExists = True
        if orderField not in selectQuery:
            selectQuerySplit.append(orderField)
            selectQuery += ", " + orderField
            orderFieldExists = False
```

```
        return orderField, orderFlag, selectQuery, selectQuerySplit, \
            orderFieldExists

    def whereToRedis(self, fromQuery, whereQuery):
        """Parse and edit the WHERE clause in order to be translated
            to python according to its syntax and logic rules.

        :param self: An instance of the class RedisQueryParser.
        :param fromQuery: A list with the tables in the FROM clause.
        :param whereQuery: A string with the WHERE clause.

        :return: A string with the python-like WHERE clause.
        """
        specialCharsWhere = []
        indexesStart = sorted([m.start() for m
                               in re.finditer("=", whereQuery)])
        counterEqual = 0
        for i in indexesStart:
            i += counterEqual
            if whereQuery[i - 1:i] is not "<" and whereQuery[i - 1:i] \
                    is not ">":
                whereQuery = whereQuery[0:i] + "==" + whereQuery[i+1:]
                counterEqual += 1
        whereQuery = whereQuery.replace("<>", "!=")
        for char in SPECIAL_CHARS:
            if char in whereQuery:
                specialCharsWhere.append(char)

        whereQuery = ' '.join(whereQuery.split())
        for char in specialCharsWhere:
            whereQuery = whereQuery.replace(" " + char + " ", char)
            whereQuery = whereQuery.replace(char + " ", char)
            whereQuery = whereQuery.replace(" " + char, char)

        for char in specialCharsWhere:
            whereQuery = self.convertToRedisWhere(whereQuery, " ", char)
            whereQuery = self.convertToRedisWhere(
                whereQuery, char, " ", False, fromQuery)

        for char in specialCharsWhere:
            whereQuery = self.convertStringToNumber(whereQuery, char, " ")
        whereQuery = ' '.join(whereQuery.split())
        whereQuery = re.sub(r'\b(?i)AND\b', ' and ', whereQuery)
        whereQuery = re.sub(r'\b(?i)OR\b', ' or ', whereQuery)
        whereQuery = re.sub(r'\b(?i)NOT\b', ' not ', whereQuery)
        whereQuery = whereQuery.replace("( ", "(").replace(") ", ")")
        for char in specialCharsWhere:
            whereQuery = whereQuery.replace(char, " " + char + " ")
        whereQuery = whereQuery.replace("< =", "<= ").replace("> =", ">= ") \
            .strip()
        whereQuery = ' '.join(whereQuery.split())
        whereString = "if " + whereQuery + ":\n\t\t"
        return whereString
```

```
    @staticmethod
    def pythonFileInitialize():
        """Initialize the python file to be created with some
            basic imports and methods' calls.

        :return: A string with initialization of the python file.
        """
        pythonFile = "import argparse\nimport numpy as np\nimport " + \
            "pandas as pd\nimport redis\n" + \
            "from tabulate import tabulate\n\n"
        pythonFile = pythonFile + \
            "r = redis.StrictRedis" + \
            "(host='localhost', port=6379, db=0)\n\n"
        pythonFile += "parser = argparse.ArgumentParser(description=" + \
            "'Execute a simple SQL query in a redis database and save" + \
            " output in a .csv file')\n"
        pythonFile += "parser.add_argument('outputFile', type=str," + \
            " help='Output .csv file with the query results.')\n"
        pythonFile += "args = parser.parse_args()\n" + \
            "resultsFile = args.outputFile\n"
        pythonFile += "if not resultsFile.endswith('.csv'):\n\t" + \
            "print '\\nOutput file should end with .csv!'\n\t" + \
            "quit()\n\ntry:\n\t"

        return pythonFile

    @staticmethod
    def pythonFileArrayResults(selectQuerySplit, whereQuery, counterTab):
        """Create the content of the python file responsible for
            saving the results properly in a numpy array.

        :param selectQuerySplit: A list with the attributes in the
            SELECT clause.
        :param whereQuery: A string with the WHERE clause.

        :return: A string with the content of the python file,
            which will save the results of the query in a numpy
            array.
        """
        resultsString = ("\t" * (counterTab - 1)) + "tempResults = np.array(["
        columnNames = ""
        for i, _ in enumerate(selectQuerySplit):
            if i == len(selectQuerySplit) - 1:
                if len(whereQuery) == 0:
                    resultsString = resultsString + "r.get(" + \
                        selectQuerySplit[i].strip() + \
                        ")])\n" + ("\t" * (counterTab + 1))
                else:
                    resultsString = resultsString + "r.get(" + \
                        selectQuerySplit[i].strip() + ")])\n" + \
                        ("\t" * (counterTab + 1))
                columnNames += "'" + selectQuerySplit[i].strip() + "'"
            else:
```

```
                resultsString += "r.get(" + selectQuerySplit[i].strip() + "),
    "
                columnNames += "'" + selectQuerySplit[i].strip() + "', "

        if counterTab == 0:
            resultsString += "\t"
        resultsString += "resultsArray = np.vstack((tempResults," + \
            " resultsArray))\n"
        resultsString = resultsString + "except NameError, e:\n\tprint" + \
            "'\\nCheck " + \
            "that all tables required are included in the FROM clause!\\n'" +
    \
            "\n\t" + \
            "print e.message\n\tquit()\n"
        resultsString = resultsString + "except ValueError, e:\n\tprint" + \
            " '\\nCheck that the value types of the WHERE clause are " + \
            "consistent with the value types of the attributes!\\n'\n\t" + \
            "print e.message\n\tquit()\n"
        resultsString += "except redis.exceptions.ConnectionError" + \
            ":\n\tprint '\\nRedis connection error! Check that " + \
            "Redis server is on and properly working!'\n\tquit()\n\n"
        resultsString = resultsString + "try:\n\tif resultsArray.size > " + \
            str(len(selectQuerySplit)) + ":\n\t\t"
        resultsString += "resultsArray = resultsArray[:-1, :]\n\t\t"

        return resultsString, columnNames

    @staticmethod
    def pythonFileForLoop(selectQuerySplit, selectQuery,
                          keysList, fromQuery):
        """Construct the main for loop of the output python file,
            in order to iterate over the results retrieved from
            the Redis database.

        :param selectQuerySplit: A list with the attributes in the
            SELECT clause.
        :param selectQuery: A string with the SELECT clause.
        :param counterWhere: The number of attributes contained in
            the WHERE clause but not in the SELECT clause.
        :param keysList: A string with the necessary content
            to iterate over the different attributess.

        :return: A string with the content of the python file,
            which will iterate over the results.
        """
        selectQuery = selectQuery.split(",")
        selectQuery = map(str.strip, selectQuery)
        forString = "resultsArray = np.zeros(" + \
            str(len(selectQuerySplit)) + ")\n\n"

        newKeysList = ''.join(map(str, keysList))
        newKeysList = newKeysList.split(",")
        newKeysList = map(str.strip, newKeysList)
        counterTab = 1
```

```
        for fromClause in fromQuery:
            forString += '\t' * counterTab
            forString += "for "
            for selectClause in selectQuery:
                if fromClause in selectClause:
                    forString += selectClause + ", "
            forString = forString.strip()
            forString = forString[:-1]

            keysForList = []
            for key in newKeysList:
                if fromClause in key:
                    keysForList.append(key)

            keysForString = ', '.join(map(str, keysForList))
            if len(keysForList) == 1:
                forString += " in " + keysForString + ":\n"
            else:
                forString += " in zip(" + keysForString + "):\n"
            counterTab += 1
        forString += '\t' * counterTab
        return forString, counterTab

    @staticmethod
    def pythonFileLimitOrderQuery(
            orderQuery, orderFlag, limitQuery,
            orderField, orderFieldExists, randomCheck):
        """Construct the main for loop of the ouput python file,
            in order to iterate over the results retrieved from
            the Redis database.

        :param orderQuery: A string with the ORDER clause.
        :param orderFlag: A boolean variable to check whether the
            ordering will be ascending or descending.
        :param limitQuery: A string with the LIMIT clause, i.e.
            the number of results to be printed.
        :param orderField: The field according to which the
            results will be ordered.
        :param orderFieldExists: A boolean variable to check whether the
            ordering field is included also in the SELECT clause or not.
        :param randomCheck: A boolean variable to check whether the
            results should be printed in random order.

        :return: A string with the content of the python file,
            related mainly with the formatting of the way the results
            are printed.
        """
        limitOrderString = ""
        if len(orderQuery) > 0 and not randomCheck:
            limitOrderString += "if dfResults['" + str(orderField) + \
                "'].dtype == 'object':\n\t\t\tdfResults['sortColumn'] " + \
                "= dfResults['" + str(orderField) + "'].str.lower()\n\t\t" + \
                "\tdfResults.sort_values(by='sortColumn', ascending=" + \
                str(orderFlag) + \
```

```
                ", inplace=True)\n\t\t\tdfResults.drop('" + \
                "sortColumn', axis=1, inplace=True)\n\t\t"
            limitOrderString += "else:\n\t\t\tdfResults.sort_values" + \
                "(by='" + orderField + "', ascending=" + str(orderFlag) + \
                ", inplace=True)\n\t\t"

            if not orderFieldExists:
                limitOrderString = limitOrderString + "dfResults.drop('" + \
                    orderField + "', axis=1, inplace=True)\n\t\t"
        if limitQuery is not None:
            limitOrderString += "dfResults = dfResults.head(n=" + \
                str(limitQuery) + ")\n\t\t"
        if randomCheck:
            if limitQuery is not None:
                limitOrderString = limitOrderString.replace(
                    "dfResults.head(n=" + str(limitQuery),
                    "dfResults.sample(n=min(" + str(limitQuery) +
                    ", dfResults.shape[0])")
            else:
                limitOrderString += \
                    "dfResults = dfResults.sample(n=dfResults.shape[0])\n\t\t"
        limitOrderString += "dfResults = dfResults.reset_index(drop=True" + \
            ")\n\t\t"

        limitOrderString += "try:\n\t\t\t"
        limitOrderString += \
            "print tabulate(dfResults, headers='keys', " + \
            "tablefmt='fancy_grid')\n\t\t"
        limitOrderString += "except UnicodeEncodeError:\n\t\t\t" + \
            "print\n\t\t\tprint dfResults\n\t\t\tpass\n\t\t"
        limitOrderString += "print '\\nTotal rows: ', dfResults.shape[0]\n\t\t
    "
        limitOrderString = limitOrderString + \
            "dfResults.to_csv(resultsFile, index=False, sep=';')\n\t\t"
        limitOrderString += "print 'The results have been saved in'"
        limitOrderString += ", resultsFile\n\t"
        limitOrderString += "else:\n\t\tprint '\\nNo results found. " + \
            "Try another query! \\nHint: Check the names of the attributes" +
    \
            " in the SELECT, the WHERE and the ORDER BY clauses ;)'\n"
        limitOrderString = limitOrderString + "except KeyError:\n\tprint" + \
            " 'Check that the ORDER BY clause contains only one field!'\n"

        return limitOrderString

    def sqlQueryToRedis(self, selectQuery, fromQuery, whereQuery, orderQuery,
                        limitQuery):
        """Call the methods required to build the output file.

        :param self: An instance of the class RedisQueryParser.
        :param selectQuery: A string with the SELECT clause.
        :param fromQuery: A string with the FROM clause.
        :param whereQuery: A string with the WHERE clause.
        :param orderQuery: A string with the ORDER clause.
```

```
        :param limitQuery: A string with the LIMIT clause.

        :return: A string with the final complete content of the
            python file.
        """
        pythonFile = self.pythonFileInitialize()
        fromQuery = fromQuery.split(",")
        fromQuery = map(str.strip, fromQuery)
        fromQuery = [s + "_" for s in fromQuery]

        selectQuerySplit = []
        orderField = ""
        orderFlag = 1
        orderFieldExists = True
        randomOrder = re.search("(?i)RAND\(\)", orderQuery.strip())
        if randomOrder is not None:
            randomOrder = randomOrder.group().upper()

        randomCheck = True
        if randomOrder != "RAND()":
            randomCheck = False
        if len(orderQuery) > 0 and randomOrder != "RAND()":
            orderField, orderFlag, selectQuery, selectQuerySplit, \
                orderFieldExists = self.orderQueryToRedis(
                    orderQuery, selectQuery)

        selectFromString, selectQuery, keysList, counterWhere, \
            selectQuerySplit = \
            self.selectFromToRedis(
                selectQuery, fromQuery, whereQuery, selectQuerySplit)
        pythonFile += selectFromString

        for _ in range(counterWhere):
            selectQuerySplit.pop(-1)

        forString, counterTab = self.pythonFileForLoop(
            selectQuerySplit, selectQuery, keysList, fromQuery)

        pythonFile += forString

        if len(whereQuery) > 0:
            pythonFile += self.whereToRedis(fromQuery, whereQuery)
        if len(whereQuery) == 0:
            counterTab = 0
        resultsString, columnNames = self.pythonFileArrayResults(
            selectQuerySplit, whereQuery, counterTab)
        pythonFile += resultsString
        if len(selectQuerySplit) == 1:
            pythonFile += "dfResults = pd.DataFrame(data=resultsArray)\n\t\t"
        else:
            pythonFile = pythonFile + "dfResults = pd.DataFrame(data=" + \
                "resultsArray, columns=(" + columnNames + "))\n\t\t"

        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 successfuly 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 
```

```
    sqlQuery = args.inputFile
    outputPython = args.outputFile

    if not os.path.isfile(sqlQuery):
        print "\nInput file does not exist!"
        quit()

    if not outputPython.endswith(".py"):
        print "\nOutput file should end with .py!"
        quit()

    instanceRedisQuery = RedisQueryParser()
    sqlClauses = instanceRedisQuery.parseSqlQuery(sqlQuery)
    pythonFileContent = instanceRedisQuery.sqlQueryToRedis(
        sqlClauses[0], sqlClauses[1], sqlClauses[2], sqlClauses[3],
        sqlClauses[4])
    instanceRedisQuery.writePythonFile(outputPython, pythonFileContent)
    instanceRedisQuery.checkSyntax(outputPython)
```

4 Unit testing

4.1 testRedisQueryParser.py

```
# pylint: disable=invalid-name, anomalous-backslash-in-string
"""
    testRedisQueryParser.py: Test the results' validity of the SQL
    Query Parsing.
"""

import unittest
from redisQueryParser import RedisQueryParser

__author__ = "Stratos Gounidellis, Lamprini Koutsokera"
__copyright__ = "Copyright 2017, BDSMasters"


class TestRredisQueryParser(unittest.TestCase):
    """TestRredisQueryParser: Implementation of the methods needed
        to successfuly test the expected results from the
        SQL Query Parsing.
    """

    def test_readSqlQuery(self):
        """Test whether a given query is read correctly or not.
        """
        instanceQueryParser = RedisQueryParser()
        fname = "redisQuery1.txt"
        clauses = instanceQueryParser.parseSqlQuery(fname)

        expectedClauses = ["Student_FName, Student_LName, Grade_Mark"]
        expectedClauses.append("Student, Grade")
        expectedClauses.append("Student_SSN=Grade_SSN")
```

```
        expectedClauses.append("")
        expectedClauses.append(None)

        self.assertEqual(clauses, tuple(expectedClauses))

    def test_selectFromToRedis(self):
        """Test whether the SELECT clause is converted correctly or not.
        """
        instanceQueryParser = RedisQueryParser()
        fname = "redisQuery1.txt"
        clauses = instanceQueryParser.parseSqlQuery(fname)
        selectQuery = clauses[0]
        fromQuery = clauses[1]
        fromQuery = fromQuery.split(",")
        fromQuery = map(str.strip, fromQuery)
        fromQuery = [s + "_" for s in fromQuery]
        whereQuery = clauses[2]
        selectQuerySplitOrder = []

        results = instanceQueryParser.selectFromToRedis(
            selectQuery, fromQuery, whereQuery, selectQuerySplitOrder)
        expectedClauses = "Student_FName_List, Student_LName_List," + \
            " Grade_Mark_List, Student_SSN_List, Grade_SSN_List"
        self.assertEqual(results[2], expectedClauses)

    def test_orderQueryToRedis(self):
        """Test whether the ORDER BY clause is converted correctly or not.
        """
        instanceQueryParser = RedisQueryParser()
        fname = "redisQuery.txt"
        clauses = instanceQueryParser.parseSqlQuery(fname)
        selectQuery = clauses[0]
        fromQuery = clauses[1]
        fromQuery = fromQuery.split(",")
        fromQuery = map(str.strip, fromQuery)
        fromQuery = [s + "_" for s in fromQuery]
        orderQuery = clauses[3]

        results = instanceQueryParser.orderQueryToRedis(
            orderQuery, selectQuery)
        results = results[:2]
        expectedClauses = ['Student_FName', 1]
        self.assertEqual(results, tuple(expectedClauses))

    def test_whereQueryToRedis(self):
        """Test whether the WHERE clause is converted correctly or not.
        """
        instanceQueryParser = RedisQueryParser()
        fname = "redisQuery.txt"
        clauses = instanceQueryParser.parseSqlQuery(fname)
        fromQuery = clauses[1]
        fromQuery = fromQuery.split(",")
        fromQuery = map(str.strip, fromQuery)
        fromQuery = [s + "_" for s in fromQuery]
```

```
        whereQuery = clauses[2]

        results = instanceQueryParser.whereToRedis(fromQuery, whereQuery)
        expectedClause = 'if r.get(Student_FName) < "Nikos1":\n\t\t'
        self.assertEqual(results, expectedClause)

    def test_exceptionSyntaxError(self):
        """Test whether the syntax of the created python file is correct.
        """
        instanceQueryParser = RedisQueryParser()
        fname = "redisQuery6.txt"

        sqlClauses = instanceQueryParser.parseSqlQuery(fname)
        pythonFileContent = instanceQueryParser.sqlQueryToRedis(
            sqlClauses[0], sqlClauses[1], sqlClauses[2], sqlClauses[3],
            sqlClauses[4])
        outputFile = "test.py"
        instanceQueryParser.writePythonFile(outputFile, pythonFileContent)

        with self.assertRaises(Exception) as context:
            instanceQueryParser.checkSyntax(outputFile)
        self.assertIn('\nERROR! Please check the syntax of the ' +
                      'query. Output python file is not created! :(',
                      "".join(context.exception))


if __name__ == "__main__":
    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*].