RSQLAlchemy: OOP abstraction of a SQLite database

by Awo Ashiabor

March 19, 2014

1 Introduction

`RSQLAlchemy` is a package that abstracts sqlite databases. Sqlite objects such as databases, tables and records are abstracted in an object oriented programming (OOP) framework. To process records, one invokes methods associated with the OOP representations. This package provides database read and write functionalities and is equivalent to the object oriented abstraction component (ORM) of the python package, SQLAlchemy[1].

2 RSQLAlchemy Concepts

`RSQLAlchemy` aims to provide an object oriented programming alternative to SQL[2]-based database management systems (DBMSs) such as SQLite. With `RSQLAlchemy`, a sqlite database connection is abstracted as an *engine*. Tables are abstracted as reference classes using the *mapTable* function. The *session* provides a workspace for running applications. It is in the session that one can query the engine for data and perform complex data processing such as data filtering, joins and so on. The session also provides methods to write to the sqlite database.

Start an RSQLAlchemy session

Begin an RSQLAlchemy application by creating a session.

```
> require(RSQLAlchemy)
> mySession=session()
```

[1] See http://www.sqlalchemy.org/

[2] SQL is a language for querying and managing data in databases—

Connect to a database

To connect to a database, create an engine with a database location and bind the engine to the sesion. The example below connects to the sqlite database enclosed in the package `RSQLAlchemy`

```
> testDB
```

```
[1] "C:/Users/Awo/R/win-library/3.0/RSQLAlchemy/exec/test.db"
```

```
> testEngine=engine(databaseName=testDB)
> mySession$bind(testEngine)
```

Query database

Query contents of a table in the database by calling the *query()* method. The *query()* method returns the entire set of query results.

```
> mySession$query("snp") #prints entire table
```

```
Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
1   SNP01   1     T     C
2   SNP02   1     A     T
3   SNP03   5     A     C
4   SNP04   5     A     T
5   SNP05   5     C     G
6   SNP06   7     T     G
7   SNP07   7     A     C
8   SNP08  11     G     A
9   SNP09  11     A     C
10  SNP10  16     G     T
11  SNP01   1     A     C
12  SNP02   2     T     C
```

```
> #mySession$query("snp.snp_id")#prints snp_id column in snp table
> #mySession$query("snp","snp.snp_id")
```

Limit query results

To limit query results use the *limit()* method or the *offset()* method.

```
> mySession$query("snp.snp_id")$limit()[1:5,]
```

```
[1] "SNP01" "SNP02" "SNP03" "SNP04" "SNP05"
```

```
> mySession$query("snp.snp_id")$offset()[5:10,]
```

```
[1] "SNP05" "SNP06" "SNP07" "SNP08" "SNP09" "SNP10"
```

Sort query results

To sort query results, use the *orderBy()* method.

```
> mySession$query("snp")$orderBy("all_A")

Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
2   SNP02   1     A     T
3   SNP03   5     A     C
4   SNP04   5     A     T
7   SNP07   7     A     C
9   SNP09  11     A     C
11  SNP01   1     A     C
5   SNP05   5     C     G
8   SNP08  11     G     A
10  SNP10  16     G     T
1   SNP01   1     T     C
6   SNP06   7     T     G
12  SNP02   2     T     C

> mySession$query("snp")$orderBy("all_A","all_B")#sort by multiple columns

Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
3   SNP03   5     A     C
7   SNP07   7     A     C
9   SNP09  11     A     C
11  SNP01   1     A     C
2   SNP02   1     A     T
4   SNP04   5     A     T
5   SNP05   5     C     G
8   SNP08  11     G     A
10  SNP10  16     G     T
1   SNP01   1     T     C
12  SNP02   2     T     C
6   SNP06   7     T     G
```

Filter query results

RSQLAlchemy permits various filter conditions. The *filterBy()* method can interprete most basic comparison functions such as *==*, *!=* and *is.null()*. RSQLAlchemy also provides functions equivalent to the *not like* and *not in* sql filters. It allows combination of filter clauses and nested filters.

```
> mySession$query("snp")$filterBy(all_A=="A") #equals

Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
2   SNP02   1     A     T
3   SNP03   5     A     C
4   SNP04   5     A     T
7   SNP07   7     A     C
9   SNP09  11     A     C
11  SNP01   1     A     C

> mySession$query("snp")$filterBy(all_A!="A") #not equals

Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
1   SNP01   1     T     C
5   SNP05   5     C     G
6   SNP06   7     T     G
8   SNP08  11     G     A
10  SNP10  16     G     T
12  SNP02   2     T     C

> #mySession$query("snp")$filterBy(snp_id %like% "1") #like
> #mySession$query("snp")$filterBy(snp_id %!like% "1") #not like
> #mySession$query("snp")$filterBy(all_A %in% c("A","C")) #in
> #mySession$query("snp")$filterBy(all_A %!in% c("A","C")) #not in
> #mySession$query("snp")$filterBy(is.null(all_A)) #is null
> #mySession$query("snp")$filterBy(!is.null(all_A)) #is not null
> #mySession$query("snp")$filterBy(all_A=="A" | all_A=="C")#or
> #mySession$query("snp")$filterBy(all_A=="A" & all_B=="C")#and
> #mySession$query("snp","genotype")$filterBy(snp.snp_id==genotype.snp_id)#multiple
```

3 Joins

To join multiple tables, use any of the 4 RSQLAlchemy join methods: innerjoin, leftjoin, rightjoin and outerjoin (abbreviated as join).

```
> mySession$join("snp","genotype")$limit()[1:3,] #outerjoin

  snp.snp_id snp.chr snp.all_A snp.all_B genotype.id genotype.snp_id
1      SNP01       1         T         C        S001           SNP01
2      SNP02       1         A         T        S001           SNP01
3      SNP03       5         A         C        S001           SNP01
```

```
  genotype.genotype
1                AB
2                AB
3                AB
```

```
> #mySession$outerjoin("snp","genotype.genotype")
> #mySession$join("snp","genotype")$filterBy(snp.snp_id==genotype.snp_id)
> #mySession$innerjoin("snp","genotype",joinOn="snp.snp_id==genotype.snp_id")
> #mySession$leftjoin("snp","genotype",joinOn="snp.snp_id==genotype.snp_id")
> #mySession$rightjoin("snp","genotype",joinOn="snp.snp_id==genotype.snp_id")
```

4 Abstract a database table structure as an OOP object (reference class)

Map a database table structure to a reference class by running the *mapTable* function. *tableName* provides the link between the reference class and the sqlite table. In the case of the example below, the **snp class** is mapped to the **snp table** in the database.

```
> snp<-mapTable(tableName="snp",
+                columns=snp_id="character",chr="character",all_A="character",all_B="
+               primaryKey="snp_id");
> snp

Generator for class "snp":

Class fields:

Name:             primaryKey                snp_id                   chr
Class:                   ANY             character             character

Name:                  all_A                 all_B             dataFrame
Class:             character             character activeBindingFunction

 Class Methods:
    "callSuper", "copy", "export", "field", "getClass", "getRefClass", "import",
"initFields", "initialize", "show", "trace", "untrace", "usingMethods"


 Reference Superclasses:
    "envRefClass"

> #create an instance of snp
> SNP50 <- snp(snp_id=SNP50, chr="1", all_A="T", all_B="C")
```

```
> #access the attributes of instances like any other reference classes
> SNP50$snp_id

[1] "SNP50"

> SNP50$all_A

[1] "T"

> SNP50$primaryKey

[1] "snp_id"
```

5 Add and delete objects from session

One can save objects temporarily in the session before commiting them to the database. Similarly, RSQLAlchemy provides methods to delete objects saved in session.

```
> #check contents of session
> length(mySession$objects);names(mySession$objects);mySession$objects

[1] 0

NULL

list()

> #add an object to session
> mySession$add(SNP50)
> length(mySession$objects);names(mySession$objects);mySession$objects

[1] 1

[1] "SNP50"

$SNP50
Reference class object of class "snp"
Field "primaryKey":
[1] "snp_id"
Field "snp_id":
[1] "SNP50"
Field "chr":
[1] "1"
Field "all_A":
[1] "T"
```

```
Field "all_B":
[1] "C"
Field "dataFrame":
     snp_id  chr all_A all_B
[1,] "SNP50" "1" "T"   "C"

> #modify object and resave to session
> SNP50$snp_id<-"SNPCHANGE"
> mySession$add(SNP50)#object previously added to session is replaced
> length(mySession$objects);names(mySession$objects);mySession$objects

[1] 1

[1] "SNP50"

$SNP50
Reference class object of class "snp"
Field "primaryKey":
[1] "snp_id"
Field "snp_id":
[1] "SNPCHANGE"
Field "chr":
[1] "1"
Field "all_A":
[1] "T"
Field "all_B":
[1] "C"
Field "dataFrame":
     snp_id      chr all_A all_B
[1,] "SNPCHANGE" "1" "T"   "C"

> #add multiple objects at once
> SNPA <- snp(snp_id=SNPA, chr="1", all_A="T", all_B="C")
> SNPB <- snp(snp_id=SNPB, chr="1", all_A="T", all_B="C")
> mySession$add_all(SNPA,SNPB)
> length(mySession$objects);names(mySession$objects);mySession$objects

[1] 3

[1] "SNP50" "SNPA"  "SNPB"

$SNP50
Reference class object of class "snp"
Field "primaryKey":
[1] "snp_id"
```

```
Field "snp_id":
[1] "SNPCHANGE"
Field "chr":
[1] "1"
Field "all_A":
[1] "T"
Field "all_B":
[1] "C"
Field "dataFrame":
     snp_id      chr all_A all_B
[1,] "SNPCHANGE" "1" "T"   "C"

$SNPA
Reference class object of class "snp"
Field "primaryKey":
[1] "snp_id"
Field "snp_id":
[1] "SNPA"
Field "chr":
[1] "1"
Field "all_A":
[1] "T"
Field "all_B":
[1] "C"
Field "dataFrame":
     snp_id chr all_A all_B
[1,] "SNPA" "1" "T"   "C"

$SNPB
Reference class object of class "snp"
Field "primaryKey":
[1] "snp_id"
Field "snp_id":
[1] "SNPB"
Field "chr":
[1] "1"
Field "all_A":
[1] "T"
Field "all_B":
[1] "C"
Field "dataFrame":
     snp_id chr all_A all_B
[1,] "SNPB" "1" "T"   "C"
```

```
> #delete object(s) from session
> length(mySession$objects);names(mySession$objects)

[1] 3

[1] "SNP50" "SNPA"  "SNPB"

> mySession$delete(SNP50)
> length(mySession$objects);names(mySession$objects)

[1] 2

[1] "SNPA" "SNPB"

> mySession$delete_all(SNPA,SNPB)
> length(mySession$objects);names(mySession$objects)

[1] 0

character(0)
```

6 Write to database

To write to database, run the commit() method

```
> mySession$query("snp")

Reference class object of class "tableRecords"
Field "dataFrame":
   snp_id chr all_A all_B
1   SNP01   1     T     C
2   SNP02   1     A     T
3   SNP03   5     A     C
4   SNP04   5     A     T
5   SNP05   5     C     G
6   SNP06   7     T     G
7   SNP07   7     A     C
8   SNP08  11     G     A
9   SNP09  11     A     C
10  SNP10  16     G     T
11  SNP01   1     A     C
12  SNP02   2     T     C

> SNPINSERT <- snp(snp_id=SNPINSERT, chr="1", all_A="T", all_B="C")
> mySession$add(SNPINSERT)
> mySession$commit()
```

```
[1] "Records committed"

> mySession$query("snp")

Reference class object of class "tableRecords"
Field "dataFrame":
      snp_id chr all_A all_B
1      SNP01   1     T     C
2      SNP02   1     A     T
3      SNP03   5     A     C
4      SNP04   5     A     T
5      SNP05   5     C     G
6      SNP06   7     T     G
7      SNP07   7     A     C
8      SNP08  11     G     A
9      SNP09  11     A     C
10     SNP10  16     G     T
11     SNP01   1     A     C
12     SNP02   2     T     C
13 SNPINSERT   1     T     C
```