How to Use DBI: Connecting to Databases with R

Seth Falcon

January 17, 2006

1 Introduction

DBI stands for DataBase Interface. In this vignette you will learn how to use the *DBI* package to interact with database systems from within R.

The *DBI* package defines a generic interface for accessing an arbitrary RDBMS. For each RDBMS, a DBI compliant driver implements the generic interface. DBI drivers are available for Oracle, MySQL, and SQLite.

In theory, using DBI's generic interface allows the same code to work with different RDBMS's. In practice, this depends on how closely the particular RDBMS's adhere to the SQL standard and the degree to which developers are careful to avoid RDBMS-specific features.

For this vignette, we will use a SQLite database containing annotation data for the Affymetrix hgu95av2 microarray chip. We begin by loading the required packages.

```
> library(tools)
> library("RBioinf")
> library("DBI")
> library("RSQLite")
> dbPath <- "hgu95av2-sqlite.db"
```

2 Connecting to a SQLite database using DBI

Connecting to a RDBMS using DBI requires two steps. First, one creates an instance of a driver appropriate for the particular RDBMS using `dbDriver`. Then a connection to the database is established using `dbConnect`.

For SQLite, only the path to the file containing the database needs to be specified. For client/server RDBMS's such as MySQL and Oracle, one would need to specify network location, username, and password information.

```
> drv <- dbDriver("SQLite")
> db <- dbConnect(drv, dbPath)
> db
```

```
<SQLiteConnection:(2175,0)>
```

To disconnect from a RDBMS, use `dbDisconnect`. The call to `dbDisconnect` will be the same regardless of the RDBMS.

```
> dbDisconnect(db)
```

```
[1] TRUE
```

3 Exploring database structure using DBI

You can interogate the database and find out what tables are available and what columns individual tables contain using `dbListTables` and `dbListFields`.

```
> db <- dbConnect(drv, dbPath)
> dbListTables(db)
```

```
[1] "acc"         "go_evi"      "go_ont"      "go_ont_name" "go_probe"
[6] "pubmed"
```

```
> dbListFields(db, "go_probe")
```

```
[1] "affy_id" "go_id"   "evi"
```

4 Queries using DBI

You can execute any SQL statement using `dbSendQuery`. This function returns a instance of a subclass of *DBIResult* from which the results of the query can be obtained using `fetch`. In order to release the resources allocated to the result set, one must call `dbClearResult` when the result set is no longer of use.

The `fetch` function returns the result of the query as a *data.frame*. Information about the result set can be obtained from the functions `dbColumnInfo`, `dbGetRowsAffected`, and `dbGetRowCount`, all demonstrated below.

Here we query two columns from the `go_probe` table. `dbSendQuery` returns a *SQLiteResult* instance, a subclass of *DBIResult*.

```
> sql <- "select affy_id, evi from go_probe"
> rs <- dbSendQuery(db, sql)
> rs
```

```
<SQLiteResult:(2175,1,3)>
```

A *DBIResult* instance contains information about the query performed.

```
> dbColumnInfo(rs)
```

```
     name    Sclass                type len precision scale nullOK
1 affy_id character SQL92_TYPE_CHAR_VAR  -1        -1    -1   TRUE
2     evi character SQL92_TYPE_CHAR_VAR  -1        -1    -1   TRUE
```

```
> dbGetStatement(rs)

[1] "select affy_id, evi from go_probe"

> dbGetRowsAffected(rs)

[1] 71896
```

The actual results are retreived using `fetch` and returned as a *data.frame*.

```
> chunk <- fetch(rs, n = 5)
> chunk

  affy_id evi
1  986_at IEA
2  986_at TAS
3  986_at IEA
4  986_at TAS
5  986_at  NR
```

The `dbGetRowCount` function returns the number of rows fetched so far.

```
> dbGetRowCount(rs)

[1] 5
```

When a *DBIResult* instance is no longer needed, it must be cleared.

```
> dbClearResult(rs)

[1] TRUE

> rs

<Expired SQLiteResult:(2175,1,3)>
```

Exercise 1
Execute a query using `dbSendQuery` *without capturing the result (that is, without saving the return value). Use* `dbListResults` *to obtain a reference to the result set. Try closing the database connection with* `dbDisconnect` *before clearing the result set. Try making repeated calls to* `dbSendQuery` *without clearing any result sets.*

Exercise 2
Execute a query that only returns 10 rows using SQL's LIMIT directive. Call `fetch` *with* `n=6` *repeatedly.*

Exercise 3
Fetch the results for a query that returns no results.

5 Building R data structures from query results

```
> sql <- "select * from go_probe"
> rs <- dbSendQuery(db, sql)
> dat <- fetch(rs, n = 500)
> goList <- split(dat$go_id, dat$affy_id)
```

Exercise 4
Modify the `goList` *creation to include the evidence code information.*

Exercise 5
Write a function to retrieve descriptive information for a GO id.

6 Using ODBC

The *RODBC* package allows you to connect to ODBC data sources provided that you have an appropriate driver installed on your machine.

The *R Data Import/Export* manual gives some examples of using the *RODBC* package and includes an example of connecting to an Excel spreadsheet (this only works on Windows).

To connect to a RDBMS using ODBC, you need to configure an ODBC data service. How you do this is system dependent. You will also need an ODBC driver for the particular database you wish to connect to.

The *RBioinf* package includes AP-MS protein complex experiment results from Gavin and Ho as `pcomplex.txt` and `pcomplex.xls`. We will use this data to demonstrate the use of *RODBC*. The data is organized in three columns: bait-protein, experiment, and prey-protein. The rows matching a given bait-protein identifier give the prey-proteins identified in that experiment.

The following will only work if you are using Windows.

```
> library("RODBC")
> if (.Platform$OS.type == "Windows") {
+     pcomplexXls <- system.file("extdata/pcomplex.xls", package = "RBioinf")
+     db <- odbcConnectExcel(pcomplexXls)
+ }

> if (!is.null(db)) {
+     sqlTables(db)
+     sqlQuery(db, "select * from [Sheet1$] limit 5")
+ }
```

Exercise 6
If you are not using Windows, use the `pcomplex.txt` *to create a SQLite database, then use RSQLite for the remaining excercises.*

Exercise 7
Use the protein complex database to discover how many distinct bait proteins and distint prey proteins are listed in the database.

Exercise 8
Find all interactions with `TY1B_A` *as a bait.*