SQLForge

Marc Carlson, Herve Pages, Nianhua Li

February 11, 2009

1   Introduction

The *AnnotationDbi* package provides a series of functions that can be used to build annotation packages for supported organisms. This collection of functions is called SQLForge.

In order to use SQLForge you really only need to have one kind of information and that is a list of paired IDs. These IDs are to be stored in a tab delimited file that is formatted in the same way that they used to be for the older AnnBuilder package. For those who are unfamiliar with the AnnBuilder package, this just means that there are two columns separated by a tab where the column on the left contains probe or probeset identifiers and the column on the right contains some sort of widely accepted gene accession. This file should NOT contain a header. SQLForge will then use these IDs along with it's own support databases to make an *AnnotationDbi* package for you. Here is how these IDs should look if you were to read them into R:

```
R> library(AnnotationDbi)
R> read.table(system.file("extdata", "hcg110_ID",
                          package="AnnotationDbi"),
             sep = "\t", header = FALSE, as.is = TRUE)[1:5,]

         V1      V2
1   1000_at X60188
2   1001_at X60957
3 1002_f_at X65962
4 1003_s_at X68149
5   1004_at X68149
```

In the example above, Genbank IDs are demonstrated. But it is also possible to use entrez gene IDs, refseq IDs or unigene accessions as the gene

identifiers. If refseq IDs are used, it is preferable to strip off the version extensions that can sometimes be added on by some vendors. The version extensions are digits that are sometimes tacked onto the end of a refseq ID and separated from the accession by a dot. As an example consider "NM_000193.2" . The "NM_000193" portion would be the actual accession number and the ".2" would be the version number. These version numbers are not used by these databases and their presence in your input can cause less than desirable results.

Alternatively, if you have an annotation file for an Affymetrix chip, you can use a parameter called affy that will automatically parse such a file and produce a similar mapping from that. It is important to understand however that despite that rather rich contents of an Affymetrix annotation file, almost none of these data are used in the making of an annotation package with SQLForge. Instead, the relevant IDs are stripped out, and then passed along to SQLForge as if you had created a file like is seen above. The option here to use such a file is offered purely as a convenience because the platform is so popular.

If you have additional information about your probes in the form of other kinds of supported gene IDs, you can pass these in as well by using the otherSrc parameter. These IDs must be formatted using the same two column format as described above, and if there are multiple source files, then you can pass them in as a list of strings that correspond to the file paths for these files.

Once you have your IDs ready, SQLForge will read them in, and use the gene IDs to compare to an intermediate database. The data from this database is what is used to make the specialized database that is placed inside of an annotation package.

At the present time, it is possible to make annotation packages for the most common model organisms. For each of these organisms another support package will be maintained and updated biannually which will include all the basic data gathered for this organism from sources such as NCBI, GO, KEGG and Flybase etc. These support packages will each be named after the organism they are intended for and will each include a large sqlite database with all the supporting information for that organism. Please note that support databases are not necessary unless you intend to actually make a new annotation package for one of the supported organisms. In the case where you want to make annotation packages, the support databases are only required for the organism in question. When SQLForge makes a new database, it uses the information supplied by the support database as the data source to make the annotation package. So the relevant support pack-

ages needs to be updated to the latest version in order to guarantee that the annotation packages you produce will be made with information from the last biannual update. These support packages are not meant to be annotation packages themselves and they come with no schema of their own. Instead these are merely a way to distribute the data to those who want to make custom annotation packages.

To check if your organism is supported simply look in the metadata packages repository on the bioconductor website for a .db0 package. Only special organism base packages will end with the .db0 extension. If you find a package that is named after the organism you are interested in, then your organism is supported, and you can use that database to make custom packages.

2   How to use SQLForge

To get the latest organism package you should only need to use biocLite.

Lets begin by making sure we have the latest organism package.

```
R>    source("http://bioconductor.org/biocLite.R")
R>    biocLite("human.db0")
```

Since each organism will have different kinds of data available, the schemas that will be needed for each organism will also change. SQLForge provides support functions for each of the model organisms that will create a sqlite database that complies with a specified database schema. To make an annotation package, these database populating functions are called along with additional code to wrap the database into a complete annotation package.

For each combination of organism and database schema, there must be a database populating function. As an example, the schema that defines chip packages for Homo sapiens is called HUMANCHIP_DB and the database populating function for that schema is called popHUMANCHIPDB(). Most of the metadata that is required by a database populating function is provided internally and is ultimately derived from the intermediate databases. But some information has to be supplied by the user such as the manufacturer etc. Additionally, the database populating functions have an option to output the schema that they use in the form of the SQL create statements that were declared internally. This allows the schema definitions to be kept synchronized with the code that generates the databases.

The following example will not only generate a database, but at the same time will also output a .sql file that will correspond to the HUMANCHIP_DB

database schema. We will begin by getting an example file that we have included in the AnnotationDbi package and then setting up the metadata to be passed in to the popHUMANCHIP() function.

```
R> hcg110_IDs = system.file("extdata",
                            "hcg110_ID",
                            package="AnnotationDbi")
R> myMeta = c("DBSCHEMA"="HUMANCHIP_DB",
              "ORGANISM"="Homo sapiens",
              "SPECIES"="Human",
              "MANUFACTURER"="Affymetrix",
              "CHIPNAME"="Human Cancer G110 Array ",
              "MANUFACTURERURL"="http://www.affymetrix.com")
```

For illustration purposes I will write this example to put the sqlite database into a temporary directory. I will also specify the type of the primary ID that is to be used by the databases populating function with the baseMapType parameter. In this case, "gb" is used to indicate genbank accesssions, but it could also have been "ug" for unigene, "eg" for Entrez gene, or "refseq" for refseq accessions. Additional details can be found in the man pages for these functions.

```
R> tmpout = tempdir()
R> popHUMANCHIPDB(affy = FALSE, prefix = "hcg110Test",
       fileName = hcg110_IDs, metaDataSrc = myMeta,
       baseMapType = "gb", outputDir = tmpout)
```

The preceeding code has generated a file in the working directory called hcg110.sqlite, which can be wrapped into an annotation package with the following:

```
R> seed <- new("AnnDbPkgSeed",
               Package = "hcg110Test.db",
               Version = "1.0.0",
               PkgTemplate = "HUMANCHIP.DB",
               AnnObjPrefix = "hcg110Test")
R> makeAnnDbPkg(seed,
                file.path(tmpout, "hcg110Test.sqlite"),
                dest_dir = tmpout)
```

Of course, most of the time you only want to make an annotation package for a particular chip. So we have made some wrapper functions to combine

all of the previous steps. The following shows how you could make the same
exact package as above but with a lot less hassle:

```
R> makeHUMANCHIP_DB(affy=FALSE,
    prefix="hcg110",
    fileName=hcg110_IDs,
    baseMapType="gb",
    outputDir = tmpout,
    version="1.0.0",
    manufacturer = "Affymetrix",
    chipName = "Human Cancer G110 Array",
    manufacturerUrl = "http://www.affymetrix.com")
```

Wrapper functions are provided for making all 6 of the different kinds of
chip based package types that are presently defined. These are named after
the schemas that they correspond to. So for example `makeHUMANCHIP_DB()`
corresponds to the HUMANCHIP_DB schema, and is used to produce chip
based annotation packages of that type.

3   Session Information

The version number of R and packages loaded for generating the vignette
were:

```
R version 2.8.1 (2008-12-22)
x86_64-unknown-linux-gnu

locale:
LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_MESSAGES=

attached base packages:
[1] tools      stats      graphics   grDevices utils      datasets
[7] methods    base

other attached packages:
[1] hgu95av2.db_2.2.5   RSQLite_0.7-1        DBI_0.2-4
[4] AnnotationDbi_1.4.3 Biobase_2.2.2
```