Creating a New Annotation Package using SQLForge

Marc Carlson, Herve Pages, Nianhua Li

April 15, 2011

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 packages 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. To list all the supported organism .db0 packages directly from R you can use available.db0pkgs().

```
R>   available.db0pkgs()

 [1] "anopheles.db0"   "arabidopsis.db0" "bovine.db0"
 [4] "canine.db0"      "chicken.db0"     "chimp.db0"
 [7] "ecoliK12.db0"    "ecoliSakai.db0"  "fly.db0"
[10] "human.db0"       "malaria.db0"     "mouse.db0"
[13] "pig.db0"         "rat.db0"         "rhesus.db0"
[16] "worm.db0"        "xenopus.db0"     "yeast.db0"
[19] "zebrafish.db0"
```

Once you know the package name, you can then install the appropriate package with biocLite().

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> ##To see what chip packages are available:
```

```
R> available.chipdbschemas()
R> ##Then you can make a DB using that schema.
R> populateDB("HUMANCHIP_DB", 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> makeDBPackage("HUMANCHIP_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 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.

2.1 Installing your custom package

To install your package in Unix simply use R CMD INSTALL <packageName> at the command line. But if you are on Windows or Mac, you may have to instead use `install.packages` from within R. This will work because this kind of simple annotation package does not contain any code that has to be compiled. So you can simply call `install.packages` and set the repos parameter to NULL and the type parameter to "source". The final R command will look something like this:

```
R> install.packages("packageNameAndPath", repos=NULL, type="source")
```

Of course, you still have to type the path to your source directory correctly as the 1st argument. It is recommended that you use the autocomplete feature in R as you enter it so that you get the path specified correctly.

3 For Advanced users: How to add extra data into your packages

Sometimes you may find that you want to add extra supplementary data into the database for the package that you just created. In these cases, you will have to begin by using the SQL to add more data into the database. Before you can do that however, you will have to change the permissions on the sqlite database. The database will always be in the inst/extdata directory of your package source after you run SQLForge. Once you can edit your database, you will have to create a new table, and populate that table with new information using SQL statements. One good way to do this would be to use the *RSQLite* interface that is introduced in portions of the *AnnotationDbi* vignette. For a more thorough treatment of the RSQLite package, please see the vignette for that package at CRAN. Once you are finished editing the database with SQL, be sure to change the database file back to being a read only file.

However, adding the content to the database is only the 1st part of what has to be done. In order for the data to be exposed to the R layer as a mapping, you will have to also create and document a mapping object. To do this step we have added a simple utility function to AnnotationDbi that allows you to make a simple Bimap from a single table. The following example will make an additional mapping between the gene names and the gene symbols found in the gene_info table for the package *hgu95av2.db*. For this particular example, no additional SQL has to be inserted 1st into the

database since it is just adding a mapping onto data that already exists in the database (but is just not normally exposed as a mapping).

```
R> library(hgu95av2.db)
R> hgu95av2NAMESYMBOL <- createSimpleBimap("gene_info",
                                           "gene_name",
                                           "symbol",
                                           hgu95av2.db:::datacache,
                                           "NAMESYMBOL",
                                           "hgu95av2.db")
R> ##What is the mapping we just made?
R> hgu95av2NAMESYMBOL

NAMESYMBOL map for hgu95av2.db (object of class "AnnDbBimap")

R> ##Display the 1st 4 relationships in this new mapping
R> as.list(hgu95av2NAMESYMBOL)[1:4]

$`1-acylglycerol-3-phosphate O-acyltransferase 1 (lysophosphatidic acid acyltransfera
[1] "AGPAT1"

$`1-acylglycerol-3-phosphate O-acyltransferase 2 (lysophosphatidic acid acyltransfera
[1] "AGPAT2"

$`1-acylglycerol-3-phosphate O-acyltransferase 3`
[1] "AGPAT3"

$`1-acylglycerol-3-phosphate O-acyltransferase 4 (lysophosphatidic acid acyltransfera
[1] "AGPAT4"
```

If instead of creating a mapping on an existing example, you wanted to add a new mapping to your customized annotation package, you would need to call this function from zzz.R in your modified annotation package (and also expose it in the namespace). You will then want to be sure that your updated database has replaced the one in the inst/extdata directory that was originally generated by SQLForge. And finally, you will need to also put a man page into your package so that users will know how to make use of this new mapping.

4 Session Information

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

```
R version 2.13.0 (2011-04-13)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=C              LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
[1] GO.db_2.5.0          hgu95av2.db_2.5.0
[3] org.Hs.eg.db_2.5.0   RSQLite_0.9-4
[5] DBI_0.2-5            AnnotationDbi_1.14.1
[7] Biobase_2.12.1

loaded via a namespace (and not attached):
[1] tools_2.13.0
```