Creating an annotation package with a new database schema

Gábor Csárdi

November 29, 2012

Contents

1 Introduction

BioConductor includes many databases and chip annotations, in a form that can be used conveniently from R, e.g. the *GO.db* package contains the Gene Ontology database, the *KEGG.db* (part of) the KEGG Pathway database, the *hgu133a.db* package contains the annotation data for the Affymetrix Human Genome 133a microarray chip, etc. These packages are called annotation packages, as they don't contain code that you can run, only data. (Or at least the emphasis is on the data, not the code.)
If one uses BioConductor heavily, together with some database, then it is convenient to create an annotation package of that database, and then use it the standard, BioConductor way. I will show how to do this, using the TargetScan μRNA target prediction database (`http://www.targetscan.org`) as an example.

2 Creating the template of your package

First, install and load the *AnnotationForge* package, this package is able to build annotation packages.

```
> library(AnnotationForge)
```

Our new package will be named `targetscan.Hs.eg.db` and we will create it in the temporary directory of the current R session.

```
> package <- "targetscan.Hs.eg.db"
> packagedir <- paste(tempdir(), sep=.Platform$file.sep, "targetscan")
> unlink(packagedir, recursive=TRUE)
```

Next, you need to create a template for your package. This basically means creating all the files of your package, except for the SQLite file containing the data itself. (That we will add later.) It is easiest to use a template for an already existing annotation package as a starting point, we will use the template for the *KEGG.db* package.

```
> start.template <- system.file("AnnDbPkg-templates/KEGG.DB",
                                package="AnnotationForge")
> tmp.keggdir <- paste(tempdir(), sep=.Platform$file.sep, "KEGG.DB")
> unlink(tmp.keggdir, recursive=TRUE)
> file.copy(start.template, tempdir(), recursive=TRUE)

[1] TRUE
```

```
> file.rename(tmp.keggdir, packagedir)

[1] TRUE
```

2.1 Update DESCRIPTION file

Now we update the package template for our purposes. Make sure that you update the DESCRIPTION file. We just modify two fields here, the rest is OK.

```
> desc <- packageDescription("targetscan", tempdir())
> desc$License <- "BSD"
> desc$Description <-
    "@PKGTITLE@ assembled using data from TargetScan"
> write.dcf(unclass(desc), attr(desc, "file"))
```

2.2 Create the *Bimap* objects

In the annotation packages the SQL databases are hidden below different kind of *Bimap* R objects. These objects are created on the fly, when you load the package. For the built in annotation packages, the *AnnotationForge* package takes care of creating them. For us this is not an option, because we want our package to work with the standard, unmodified *AnnotationDbi* package. So we put the required code to build the R objects into our package itself. This involves modifying the zzz.R file in our package template. First we create the "seeds" object that is used to translate the *Bimap* operations to SQL queries. It is basically a list of entries, one entry for one *Bimap* object. Note, that it is not necessarily true, that each *Bimap* object corresponds to a SQL table. You can have two *Bimaps* querying the same table(s), as you will see in my examples below.
You can find some documentation about the syntax in this email from the BioConductor mailing list: https://stat.ethz.ch/pipermail/bioconductor/2009-March/026740.html

```
> zzzfile <- paste(packagedir, sep=.Platform$file.sep,
                   "R", "zzz.R")
> bimap.code <- '
  ### Mandatory fields: objName, Class and L2Rchain
  TARGETSCAN_DB_AnnDbBimap_seeds <- list(
      list(objName="MIRBASE2FAMILY",
           Class="AnnDbBimap",
           L2Rchain=list(
             list(tablename="mirbase",
```

```
            Lcolname="mirbase_id",
            Rcolname="family"
            ),
       list(
            tablename="mirna_family",
            Lcolname="_id",
            Rcolname="name"
            )
       )
     ),
list(objName="MIRNA",
     Class="miRNAAnnDbBimap",
     L2Rchain=list(
       list(tablename="mirbase",
            Lcolname="mirbase_id",
            Rcolname="mirbase_id",
            Rattribnames=c(
              MiRBase_Accession="{mirbase_accession}",
              Seed_m8="{seed_m8}",
              Species_ID="species.name",
              Mature_sequence="{mature_sequence}",
              Family_conservation="{family_conservation}"),
            Rattrib_join=
              "LEFT JOIN species ON {species}=species.id"
            )
       )
     ),
 list(objName="TARGETS",
      Class="AnnDbBimap",
      L2Rchain=list(
        list(tablename="genes",
             Lcolname="gene_id",
             Rcolname="_id"
             ),
        list(tablename="targets",
             Lcolname="target",
             Rcolname="family"
             ),
        list(tablename="mirna_family",
             Lcolname="_id",
             Rcolname="name"
             )
        )
      ),
 list(objName="TARGETSFULL",
```

```
              Class="miRNATargetAnnDbBimap",
              L2Rchain=list(
                list(tablename="genes",
                     Lcolname="gene_id",
                     Rcolname="_id"
                     ),
                list(tablename="targets",
                     Lcolname="target",
                     Rattribnames=c(
                       UTR_start="{utr_start}",
                       UTR_end="{utr_end}",
                       MSA_start="{msa_start}",
                       MSA_end="{msa_end}",
                       Seed_match="seed_match.name",
                       PCT="{pct}"),
                     Rattrib_join=paste(sep="",
                       "LEFT JOIN seed_match ON {seed_match}=seed_match._id ",
                       "LEFT JOIN mirna_family AS _R ON {family}=_R._id"),
                     Rcolname="name"
##                     ),
##                list(tablename="mirna_family",
##                     Lcolname="_id",
##                     Rcolname="name"
                     )
                )
              )
  )
  '
> cat(bimap.code, file=zzzfile, append=TRUE)
```

You can see more examples in the `R/createAnnObjs.*` files in the source code of the *AnnotationDbi* package.
Most of the *Bimap* objects are of subtype *AnnDbBimap*, this is actually a subclass of *Bimap*. If you want to do something special, e.g. I need a lot of meta data for each μRNA, target gene prediction, then you can create subclasses for yourself, e.g. *miRNATargetAnnDbBimap* is such a subclass. Then, you need the actual function that creates the mapping using the seed. For me this is called `createAnnObjs.TARGETSCAN_DB`, so I have added this code to `zzz.R`:

```
> create.code <- '
  createAnnObjs.TARGETSCAN_DB <- function(prefix, objTarget,
                                          dbconn, datacache)
  {
```

```
      ## checkDBSCHEMA(dbconn, "TARGETSCAN_DB")

      ## AnnDbBimap objects
      seed0 <- list(
          objTarget=objTarget,
          datacache=datacache
      )
      ann_objs <- AnnotationDbi:::createAnnDbBimaps(
                    TARGETSCAN_DB_AnnDbBimap_seeds, seed0)

      ## Reverse maps
      revmap2 <- function(from, to)
      {
          map <- revmap(ann_objs[[from]], objName=to)
          L2Rchain <- map@L2Rchain
          tmp <- L2Rchain[[1]]@filter
          L2Rchain[[1]]@filter <-
            L2Rchain[[length(L2Rchain)]]@filter
          L2Rchain[[length(L2Rchain)]]@filter <- tmp
          map@L2Rchain <- L2Rchain
          map
      }
      ann_objs$FAMILY2MIRBASE <- revmap2("MIRBASE2FAMILY",
                                         "FAMILY2MIRBASE")

      ## 1 special map that is not an AnnDbBimap object
      ## (just a named integer vector)
      ann_objs$MAPCOUNTS <-
        AnnotationDbi:::createMAPCOUNTS(dbconn, prefix)

      AnnotationDbi:::prefixAnnObjNames(ann_objs, prefix)
  }
  '
> cat(create.code, file=zzzfile, append=TRUE)
```

Because of a recent change in *AnnotationDbi* package, we need to do workaround to get our function called. The `createAnnObjs.Schemachoice` function call that was introduced can only handle the built in database schemas, so we need to replace it with an explicit call to our function.

```
> zzz <- readLines(zzzfile)
> zzz <- sub('createAnnObjs.SchemaChoice("@DBSCHEMA@",',
             'createAnnObjs.@DBSCHEMA@(',
             zzz, fixed=TRUE)
> writeLines(zzz, zzzfile)
```

2.3 Implement your new *Bimap* classes

If you needed your own *Bimap* classes, then you need to implement them. I did this by adding the following code to the `.onLoad` function of the `zzz.R` file, right after the `require("methods")` line:

```
> class.code <- '
  setClass("miRNAAnnDbBimap", contains="AnnDbBimap")
  setClass("miRNATargetAnnDbBimap", contains="AnnDbBimap")

  setMethod("as.list", "miRNATargetAnnDbBimap",
            function(x, ...)
            {
              y <- AnnotationDbi:::flatten(x, fromKeys.only=TRUE)
              makemiRNATargetNode <- function(name, UTR_start, UTR_end,
                                              MSA_start, MSA_end,
                                              Seed_match, PCT, ...)
                {
                  l <- mapply(list, SIMPLIFY=FALSE,
                              miR.Family=name,
                              UTR.start=UTR_start,
                              UTR.end=UTR_end,
                              MSA.start=MSA_start,
                              MSA.end=MSA_end,
                              Seed.match=Seed_match,
                              PCT=PCT)
                  for (i in seq_along(l)) {
                    class(l[[i]]) <- "targetscan.Target"
                  }
                  l
                }
              AnnotationDbi:::.toListOfLists(y, mode=1,
                                             makemiRNATargetNode)
            }
            )

  setMethod("as.list", "miRNAAnnDbBimap",
            function(x, ...)
            {
              y <- AnnotationDbi:::flatten(x, fromKeys.only=TRUE)
              makemiRNANode <- function(mirbase_id,
                                        MiRBase_Accession,
                                        Seed_m8, Species_ID,
                                        Mature_sequence,
                                        Family_conservation, ...)
```

```
                {
                  l <- list(MiRBase.ID=mirbase_id,
                            MiRBase.Accession=MiRBase_Accession,
                            Seed.m8=Seed_m8,
                            Species=Species_ID,
                            Mature.sequence=Mature_sequence,
                            Family.conservation=Family_conservation,
                            ...)
                  class(l) <- "targetscan.MiRBase"
                  l
                }
              AnnotationDbi:::.toListOfLists(y, mode=1, makemiRNANode)
            }
            )
  '
> zzz <- readLines(zzzfile)
> insert.class <- grep('require[(]"methods', zzz)[1]
> zzz <- c(zzz[1:insert.class],
           paste("   ", strsplit(class.code, "\n")[[1]]),
           zzz[(insert.class+1):length(zzz)])
> writeLines(zzz, zzzfile)
```

Note that the new classes create S3 objects instead of S4, this is just me disliking S4. If you want yo see an S4 example, look at the `as.list` method of the *GOTermsAnnDbBimap* class in the `R/BimapFormatting.R` file of the *AnnotationDbi* package.
We also add some methods to print the objects of the new classes in a nice way. For simplicity we append everything to the `zzz.R` file.

```
> print.code <- '
  print.targetscan.MiRBase <- function(x, ...) {
    for (name in names(x)) {
      name2 <- sub("\\\\.", " ", name)
      s <- paste(sep="", name2, ": ", x[[name]])
      cat(strwrap(s, exdent=4), sep="\n")
    }
    invisible(x)
  }

  print.targetscan.Target <- function(x, ...) {
    for (name in names(x)) {
      name2 <- sub("\\\\.", " ", name)
      s <- paste(sep="", name2, ": ", x[[name]])
      cat(strwrap(s, exdent=4), sep="\n")
```

```
    }
    invisible(x)
  }
  '
> cat(print.code, file=zzzfile, append=TRUE)
```

The new classes and the methods need to be exported, so some extra lines needed to be added to the `NAMESPACE` file:

```
> namespace.file <- zzzfile <- paste(packagedir, sep=.Platform$file.sep,
                                     "NAMESPACE")
> namespace.text <- '
  exportClasses("miRNAAnnDbBimap",
                "miRNATargetAnnDbBimap")

  export(print.targetscan.MiRBase, print.targetscan.Target)
  S3method("print", targetscan.MiRBase)
  S3method("print", targetscan.Target)
  '
> cat(namespace.text, file=namespace.file, append=TRUE)
```

2.4 Write documentation

Remove the unneeded `.Rd` files from your package template and write the ones you need. In reality you might want to wait with this, until the package is actually built and everything works well, because you might want to redesign your set of *Bimap* objects a couple of times, before settling down with the perfect solution.
For this demonstration package we don't write any manual pages, instead we remove the `man` directory completely.

```
> mandir <- paste(tempdir(), sep=.Platform$file.sep,
                  "targetscan", "man")
> unlink(mandir, recursive=TRUE)
```

3 Adding the new database schema

Add the `inst/DBschemas/schemas_1.0/TARGETSCAN_DB.sql` file, this contains basically the SQL commands to create your tables and indexes. For standard annotation packages the schema files are in the *AnnotationDbi* package, but we don't want to modify that, so we just put it into our package:

```
> schema.text <- '
  --
  -- TARGETSCAN_DB schema
  -- ====================
  --

  CREATE TABLE genes (
    _id INTEGER PRIMARY KEY,
    gene_id VARCHAR(10) NOT NULL UNIQUE          -- Entrez Gene ID
  );

  CREATE TABLE mirna_family (
    _id INTEGER PRIMARY KEY,
    name VARCHAR(255) NOT NULL                   -- miRNA family
  );

  CREATE TABLE species (
    id VARCHAR(10) PRIMARY KEY,                  -- species ID from NCBI
    name VARCHAR(100) NOT NULL                   -- species name
  );

  CREATE TABLE seed_match (
    _id INTEGER PRIMARY KEY,
    name VARCHAR(10)
  );

  CREATE TABLE mirbase (
    mirbase_id VARCHAR(50) PRIMARY KEY,          -- MiRBase ID
    mirbase_accession CHAR(12) NOT NULL UNIQUE,  -- MiRBase accession
    family INTEGER NOT NULL,                     -- REFERENCES family
    seed_m8 CHAR(7) NOT NULL,                    -- seed m8
    species VARCHAR(10) NOT NULL,                -- REFERENCES species
    mature_sequence VARCHAR(100) NOT NULL,       -- mature sequence
    family_conservation VARCHAR(3) NOT NULL,     -- family convervation
    FOREIGN KEY (family) REFERENCES mirna_family (_id),
    FOREIGN KEY (species) REFERENCES species (id)
  );

  CREATE TABLE targets (
    family INTEGER NOT NULL,                     -- REFERENCES family
    target INTEGER NOT NULL,                     -- REFERENCES genes
    species VARCHAR(10) NOT NULL,                -- REFERENCES species
    utr_start INTEGER NOT NULL,                  -- UTR start
    utr_end INTEGER NOT NULL,                    -- UTR end
    msa_start iNTEGER NOT NULL,                  -- MSA start
```

```
  msa_end INTEGER NOT NULL,                    -- MSA end
  seed_match INTEGER NOT NULL,                 -- REFERENCES seed_match
  pct VARCHAR(10) NOT NULL,                    -- PCT
  FOREIGN KEY (family) REFERENCES mirna_family (_id),
  FOREIGN KEY (target) REFERENCES genes (_id),
  FOREIGN KEY (species) REFERENCES species (id),
  FOREIGN KEY (seed_match) REFERENCES seed_match (_id)
);

-- Metadata tables.
CREATE TABLE metadata (
  name VARCHAR(80) PRIMARY KEY,
  value VARCHAR(255)
);
CREATE TABLE map_counts (
  map_name VARCHAR(80) PRIMARY KEY,
  count INTEGER NOT NULL
);
CREATE TABLE map_metadata (
  map_name VARCHAR(80) NOT NULL,
  source_name VARCHAR(80) NOT NULL,
  source_url VARCHAR(255) NOT NULL,
  source_date VARCHAR(20) NOT NULL
);
-- Indexes
'
> dir.create(paste(packagedir, sep=.Platform$file.sep,
                   "inst", "DBschemas", "schemas_1.0"),
             recursive=TRUE, mode="0755")
> cat(schema.text, file=paste(packagedir, sep=.Platform$file.sep,
                     "inst", "DBschemas", "schemas_1.0",
                     "TARGETSCAN_DB.sql"))
```

Make sure that you read the schema guidelines in the file `DBschemas/SchemaGuidelines.txt` in the *AnnotationDbi* package. The `DBschemas/schemas_1.0/DataTypes.txt` file is also useful, this contains the types of the columns of the already existing annotation packages and you want to be consistent with these, i.e. the column storing Entrez Gene IDs should be the same in all annotation packages.

4 Creating the SQLite database file

We start creating the database itself now. An SQLite database is a single file. We will simply put it into the temporary directory of the current R session.

```
> dbfile <- file.path(tempdir(), "targetscan.Hs.eg.sqlite")
> unlink(dbfile)
```

4.1 Download the data

Next, we download the data. We need the data from the TargetScan website and the NCBI taxonomy data file, to translate species identifiers to species names. We put all the files into the temporary R directory. If they are already there, then they are not downloaded again.

```
> ## Download TargetScan data
> targetfile <- file.path(tempdir(), "Predicted_Targets_Info.txt")
> targetfile.zip <- paste(targetfile, sep="", ".zip")
> if (!file.exists(targetfile)) {
    data.url <- paste(sep="", "http://www.targetscan.org/vert_50/",
           "vert_50_data_download/Predicted_Targets_Info.txt.zip")
    download.file(data.url, destfile=targetfile.zip)
    targetfile.tmp <-
      zip.file.extract(targetfile, basename(targetfile.zip))
    file.copy(targetfile.tmp, targetfile)
  }
> familyfile <- file.path(tempdir(), "miR_Family_Info.txt")
> familyfile.zip <- paste(familyfile, sep="", ".zip")
> if (!file.exists(familyfile)) {
    data.url <- paste(sep="", "http://www.targetscan.org/vert_50/",
                      "vert_50_data_download/miR_Family_Info.txt.zip")
    download.file(data.url, destfile=familyfile.zip)
    familyfile.tmp <-
      zip.file.extract(familyfile, basename(familyfile.zip))
    file.copy(familyfile.tmp, familyfile)
  }
> taxfile <- file.path(tempdir(), "names.dmp")
> taxfile.zip <- file.path(tempdir(), "taxdmp.zip")
> if (!file.exists(taxfile)) {
    data.url <- "ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdmp.zip"
    download.file(data.url, destfile=taxfile.zip)
    taxfile.tmp <-
      zip.file.extract(taxfile, basename(taxfile.zip))
```

```
  file.copy(taxfile.tmp, taxfile)
}
```

4.2 Read in the data

Next, read in the data, and filter it a bit.

```
> family <- read.delim(familyfile)
> targets <- read.delim(targetfile)
> tax <- read.delim(taxfile, header=FALSE, quote="")
> tax <- tax[,-c(2,4,6,8)]
> species <- unique(family$Species.ID)
> names <- tax[,2][ match(species, tax[,1]) ]
> species <- data.frame(id=species, name=names)
```

4.3 Create the database and the tables

The code for creating the tables is taken from the definition of the new database schema.

```
> ## Create the database file
> library(RSQLite)
> drv <- dbDriver("SQLite")
> db <- dbConnect(drv, dbname=dbfile)
> ## Create tables
> create.sql <- strsplit(schema.text, "\n")[[1]]
> create.sql <- paste(collapse="\n", create.sql)
> create.sql <- strsplit(create.sql, ";")[[1]]
> create.sql <- create.sql[-length(create.sql)] # nothing to run here
> tmp <- sapply(create.sql, function(x) sqliteQuickSQL(db, x))
```

4.4 Put the data into the tables

Now we insert the data into the tables, line by line. This can be quite slow if you have big tables. Unfortunately SQLite cannot insert more than one line with a single `INSERT` command. (TODO)

```
> ## Append data
> ## Species
> dbBeginTransaction(db)
> dbGetPreparedQuery(db, 'INSERT INTO "species" VALUES(:id, :name);',
                     bind.data=species)
> dbCommit(db)
> ## miRNA families
```

```
> family$miR.family <- sub("^miR-141/200$", "miR-141/200a",
                           family$miR.family, fixed=FALSE)
> family$miR.family <- sub("^miR-200bc/420$", "miR-200bc/429",
                           family$miR.family, fixed=FALSE)
> fam <- unique(as.character(family$miR.family))
> fam <- cbind(id=seq_along(fam), fam)
> dbBeginTransaction(db)
> dbGetPreparedQuery(db,
                     "INSERT INTO 'mirna_family' VALUES(:id, :fam);",
                     bind.data=as.data.frame(fam))
> dbCommit(db)
> ## mirbase table
> mirbase <- family[,c("MiRBase.ID", "MiRBase.Accession",
                       "miR.family", "Seed.m8", "Species.ID",
                       "Mature.sequence",
                       "Family.Conservation.")]
> mirbase$miR.family <- fam[,1][ match(family$miR.family, fam[,2]) ]
> colnames(mirbase) <- letters[1:7]
> dbBeginTransaction(db)
> dbGetPreparedQuery(db, bind.data=mirbase,
                     'INSERT INTO "mirbase" VALUES(:a,:b,:c,:d,:e,:f,:g)')
> dbCommit(db)
> ## keep entries for human only
> targets2 <- targets
> targets2 <- targets2[ targets2$Species.ID == 9606, ]
> targets2$Gene.ID[ targets2$Gene.ID == 934] <- 100133941
> ## genes
> gs <- unique(targets2$Gene.ID)
> gs <- cbind(seq_along(gs), gs)
> dbBeginTransaction(db)
> dbGetPreparedQuery(db, bind.data=data.frame(a=gs[,1],
                           b=as.integer(gs[,2])),
                     "INSERT INTO 'genes' VALUES(:a,:b);")
> dbCommit(db)
> ## seed_match
> sm <- sort(unique(as.character(targets$Seed.match)))
> sm <- cbind(seq_along(sm), sm)
> dbBeginTransaction(db)
> dbGetPreparedQuery(db, bind.data=data.frame(a=sm[,1], b=sm[,2]),
                     "INSERT INTO 'seed_match' VALUES(:a,:b);")
> dbCommit(db)
> ## targets, human only :(
> targets2$miR.Family <- fam[,1][ match(targets2$miR.Family, fam[,2]) ]
> targets2$Gene.ID <- gs[,1][ match(targets2$Gene.ID, gs[,2]) ]
> targets2$Seed.match <- sm[,1][ match(targets2$Seed.match, sm[,2]) ]
```

```
> colnames(targets2) <- sub(".", "_", colnames(targets2), fixed=TRUE)
> dbBeginTransaction(db)
> dbGetPreparedQuery(db, bind.data=targets2,
                     paste(sep="",
                           "INSERT INTO targets VALUES(:miR_Family,",
                           ":Gene_ID, :Species_ID,",
                           ":UTR_start, :UTR_end,",
                           ":MSA_start, :MSA_end,",
                           ":Seed_match, :PCT);"))
> dbCommit(db)
```

4.5 Append the metadata

Every annotation package must contain a `metadata` table, with some information about the package. Moreover, there is also a `map_counts` table, that provides some quality control.

```
> ## metadata
> metadata <- rbind(c("DBSCHEMA", "TARGETSCAN_DB"),
                    c("ORGANISM", "Homo sapiens"),
                    c("SPECIES", "Human"),
                    c("DBSCHEMAVERSION", "1.0"),
                    c("TARGETSCANSOURCENAME", "TargetScan"),
                    c("TARGETSCANSOURCEURL",
                      paste(sep="", "http://www.targetscan.org/cgi-bin/",
                            "targetscan/data_download.cgi?db=vert_50")),
                    c("TARGETSCANSOURCEDATE",
                      format(Sys.time(), "%Y-%b%d")),
                    c("TARGETSCANVERSION", "5.0"))
> q <- paste(sep="", "INSERT INTO 'metadata' VALUES('", metadata[,1],
             "','", metadata[,2], "');")
> tmp <- sapply(q, function(x) sqliteQuickSQL(db, x))
> ## map_counts
> map.counts <- rbind(c("FAMILY2MIRBASE", nrow(fam)),
                      c("MIRBASE2FAMILY", nrow(mirbase)),
                      c("MIRNA", nrow(mirbase)),
                      c("TARGETS", nrow(gs)),
                      c("TARGETSFULL", nrow(gs))
                      )
> q <- paste(sep="", "INSERT INTO 'map_counts' VALUES('", map.counts[,1],
             "',", map.counts[,2], ");")
> tmp <- sapply(q, function(x) sqliteQuickSQL(db, x))
```

4.6 Check that everything was added properly

Some quick checks that the SQLite database has the right number of rows and columns.

```
> if (dbGetQuery(db, "SELECT COUNT(*) FROM species") != nrow(species)) {
    stop("FOOBAR")
  }
> if (dbGetQuery(db, "SELECT COUNT(*) FROM mirna_family") != nrow(fam)) {
    stop("FOOBAR")
  }
> if (dbGetQuery(db, "SELECT COUNT(*) FROM mirbase") != nrow(mirbase)) {
    stop("FOOBAR")
  }
> if (dbGetQuery(db, "SELECT COUNT(*) FROM genes") != nrow(gs)) {
    stop("FOOBAR")
  }
> if (dbGetQuery(db, "SELECT COUNT(*) FROM seed_match") != nrow(sm)) {
    stop("FOOBAR")
  }
> if (dbGetQuery(db, "SELECT COUNT(*) FROM targets") != nrow(targets2)) {
    stop("FOOBAR")
  }
```

Finally, we can disconnect from the SQLite database file. If you want to change things in the database, then you can reopen it any number of times, wither from R, or from another tool, e.g. the command line utility called `sqlite3`.

```
> ## Disconnect
> dbGetQuery(db, "VACUUM")

NULL

> dbDisconnect(db)

[1] TRUE
```

5 Build the new annotation package

```
> seed <- new("AnnDbPkgSeed",
              Package = package,
              Version = "5.0-1",
              PkgTemplate = packagedir,
```

```
                AnnObjPrefix = "targetscan.Hs.eg",
                Title = "TargetScan miRNA target predictions for human",
                Author = "Gabor Csardi <Gabor.Csardi@foo.bar>",
                Maintainer = "Gabor Csardi <Gabor.Csardi@foo.bar>",
                organism = "Homo sapiens",
                species = "Human",
                biocViews = "AnnotationData, FunctionalAnnotation",
                DBschema = "TARGETSCAN_DB",
                AnnObjTarget = "TargetScan (Human)",
                manufacturer = "None",
                manufacturerUrl = "None"
                )
> unlink(paste(tempdir(), sep=.Platform$file.sep, package), recursive=TRUE)
> makeAnnDbPkg(seed, dbfile, dest_dir = tempdir())
```

6 Install and try the new package

```
> install.packages(paste(tempdir(), sep=.Platform$file.sep, "targetscan.Hs.eg.db"),
                   repos=NULL)

> library(targetscan.Hs.eg.db)
> mget(c("346389", "54715"), targetscan.Hs.egTARGETS)
> mget("miR-328", revmap(targetscan.Hs.egTARGETS))
> mget("346389", targetscan.Hs.egTARGETSFULL)
> toTable(targetscan.Hs.egTARGETS)[1:5,]
> toTable(targetscan.Hs.egTARGETSFULL)[1:5,]
> mget("hsa-let-7a", targetscan.Hs.egMIRNA)
> checkMAPCOUNTS("targetscan.Hs.eg.db")
```

7 Session Information

- R version 2.15.2 (2012-10-26), `i386-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: AnnotationDbi 1.20.3, AnnotationForge 1.0.3, Biobase 2.18.0, BiocGenerics 0.4.0, DBI 0.2-5, RSQLite 0.11.2, org.Hs.eg.db 2.8.0

- Loaded via a namespace (and not attached): IRanges 1.16.4, parallel 2.15.2, stats4 2.15.2, tools 2.15.2