Interacting with Databases

- SQL
- DBI
- SQLite
- AnnotationDbi
- SQLForge

SQL Basics:

- SELECT and FROM
  - SELECT * FROM table;

table

id	value
1	bar
2	bar2

1	bar
2	bar2

SQL Basics:

- WHERE
  - SELECT * FROM table WHERE value=”bar”;

table

id	value
1	bar
2	bar2

1	bar

SQL Basics:

- AS
  - SELECT id AS ID_stuff FROM table;

table

id	value
1	bar
2	bar2

ID_Stuff
1
2

SQL Basics:

- AS
  - SELECT id FROM table AS Tab1;

table

id	value
1	bar
2	bar2

Tab1

id
1
2

SQL Basics:

- DISTINCT
  - SELECT DISTINCT id FROM table;

table

id	value
1	bar
2	bar2
2	bar3

id

1

2

DBI

- DBI provides a series of classes and functions for interacting with databases from R.
- DBI is a base package for RSQLite and other database interfaces
- One of the interfaces to build off of DBI is RSQLite

A Database Connection with DBI

```
> library("RSQLite")
> m = dbDriver("SQLite")
> con = dbConnect(m)
> data(USArrests)
> dbWriteTable(con, "USArrests", USArrests, overwrite = TRUE)
[1] TRUE
> dbListTables(con)
[1] "USArrests"
```

Using fetch to get a result set

```
> rs = dbSendQuery(con, "select * from USArrests")
> d1 = fetch(rs, n = 5)
> d1
```

	row_names	Murder	Assault	UrbanPop	Rape
1	Alabama	13.2	236	58	21.2
2	Alaska	10.0	263	48	44.5
3	Arizona	8.1	294	80	31.0
4	Arkansas	8.8	190	50	19.5
5	California	9.0	276	91	40.6

Using a result set

```
> rs = dbSendQuery(con, "select * from USArrests")
> d1 = fetch(rs, n = 5)
> dbHasCompleted(rs)
[1] FALSE
> dbListResults(con)
[[1]]
<SQLiteResult:(13692,0,7)>
> d2 = fetch(rs, n = -1)
> d2[1:3,]
```

	row_names	Murder	Assault	UrbanPop	Rape
6	Colorado	7.9	204	78	38.7
7	Connecticut	3.3	110	77	11.1
8	Delaware	5.9	238	72	15.8

Clearing a result set

We have now fetched ALL of the result set, so it has completed

> dbHasCompleted(rs)

[1]TRUE

#You must always clean up your result sets.

> dbClearResult(rs)

[1]TRUE

dbGetQuery vs dbSendQuery

- dbSendQuery() returns a result set that has to subsequently be cleared.
- dbGetQuery() automatically fetches any results and then clears your result set each time.
- DO NOT mix these functions in a session.

Some simple examples

```
> dbListTables(con)
[1] "USArrests"
> dbListFields(con, "USArrests")
[1] "row_names" "Murder"    "Assault"   "UrbanPop"  "Rape"



> query = paste("SELECT name FROM sqlite_master WHERE",
+   "type='table' ORDER BY name;")
> rs = dbSendQuery(con, query)
> fetch(rs, n= -1)
       name
1 USArrests
```

More simple examples

```
> rs = dbSendQuery(con,
+    "SELECT * FROM USArrests WHERE Murder > 10")
> fetch(rs, n= 2)
```

	row_names	Murder	Assault	UrbanPop	Rape
1	Alabama	13.2	236	58	21.2
2	Alaska	10.0	263	48	44.5

AnnotationDbi

- For interacting with and creating new annotation packages.
- Old environment annotations are emulated
- All data is now stored in sqlite files
- Schemas for these databases are standardized
- Code is in place to create new annotation packages by making databases according to standard schemas and then wrapping these into annotation packages

Basic info about AnotationDBi pkgs

```
> library("hgu95av2.db")
> ls("package:hgu95av2.db")
 [1] "hgu95av2"              "hgu95av2ACCNUM"        "hgu95av2ALIAS2PROBE"
 [4] "hgu95av2CHR"           "hgu95av2CHRLENGTHS"    "hgu95av2CHRLOC"
 [7] "hgu95av2_dbconn"       "hgu95av2_dbfile"       "hgu95av2_dbInfo"
[10] "hgu95av2_dbschema"     "hgu95av2ENSEMBL"       "hgu95av2ENSEMBL2PROBE"
[13] "hgu95av2ENTREZID"      "hgu95av2ENZYME"        "hgu95av2ENZYME2PROBE"
[16] "hgu95av2GENENAME"      "hgu95av2GO"            "hgu95av2GO2ALLPROBES"
[19] "hgu95av2GO2PROBE"      "hgu95av2MAP"           "hgu95av2MAPCOUNTS"
[22] "hgu95av2OMIM"          "hgu95av2ORGANISM"      "hgu95av2PATH"
[25] "hgu95av2PATH2PROBE"    "hgu95av2PFAM"          "hgu95av2PMID"
[28] "hgu95av2PMID2PROBE"    "hgu95av2PROSITE"       "hgu95av2REFSEQ"
[31] "hgu95av2SYMBOL"        "hgu95av2UNIGENE"
> mycon = hgu95av2_dbconn()
> colnames(hgu95av2GO)
[1] "probe_id"     "go_id"     "Evidence"     "Ontology"
```

Basic info about AnotationDBi maps

- You can look at the maps in the usual ways:

```
> as.list(hgu95av2ENTREZID)[1:2]
$`1000_at`
[1] "5595"
$`1001_at`
[1] "7075"
> mget(c("1000_at","1001_at"), hgu95av2ENTREZID)
 $`1000_at`
[1] "5595"
$`1001_at`
[1] "7075"
```

You can list what is in a map

```
> toTable(hgu95av2GO)[1:4,]
     probe_id  go_id              Evidence Ontology
1    1000_at   GO:0000074        NAS      BP
2    1000_at   GO:0006468        IDA      BP
3    1000_at   GO:0006468        IEA      BP
4    1000_at   GO:0007049        IEA      BP
> links(hgu95av2GO)[1:4,]
     probe_id  go_id              Evidence
1    1000_at   GO:0000074        NAS
2    1000_at   GO:0006468        IDA
3    1000_at   GO:0006468        IEA
4    1000_at   GO:0007049        IEA
```

You can reverse maps

```
> is(hgu95av2MAP, "Bimap")
> rmMAP = revmap(hgu95av2MAP)
> rmMAP$"13q14.2"
[1] "1570_f_at" "1571_f_at" "1672_f_at" "1900_at"   "2044_s_at"
"39405_at"
[7] "38649_at"


> myl=list(a="w", b="x")
> revmap(myl)
$w
[1] "a"
$x
[1] "b"
```

Basic structure of the underlying DB

- Portable
- Gene centric
- Uses an internal “_id”
- “_id” has no meaning outside of an individual database

Using RSQLite to interface with an Annotation Database

```
#You could do this:
> library("RSQLite")
> m = dbDriver("SQLite")
> testDB = system.file("extdata/hgu95av2-sqlite.db",
+  package="RBioinf")
> con = dbConnect(m, dbname = testDB)


#then:
> tabs = dbListTables(con)
> tabs
> dbListFields(con, tabs[2])
```

Simple example of an inner Join I

```
> mycon = hgu95av2_dbconn()
> tabs = dbListTables(mycon)
> tabs
 [1] "alias"  "chrlengths"  "chromosome_locations"
 [4] "chromosomes"  "cytogenetic_locations"  "ec"
 [7] "ensembl"  "gene_info"  "genes"
[10] "go_bp"  "go_bp_all"  "go_cc"
[13] "go_cc_all"  "go_mf"  "go_mf_all"
[16] "kegg"  "map_counts"  "map_metadata"
[19] "metadata"  "omim"  "pfam"
[22] "probes"  "prosite"  "pubmed"
[25] "refseq"  "sqlite_stat1"  "unigene"
```

Simple example of an inner Join II

```
> dbListFields(mycon, tabs[16])
[1] "_id"     "path_id"

> dbListFields(mycon, tabs[22])
[1] "probe_id"  "accession" "_id"

#Could also have just used (more on this later):
> hgu95av2_dbschema()
```

Simple example of an inner Join III

```
> SQL <- "SELECT probe_id, path_id from probes, kegg WHERE probes._id=kegg._id;"
> tab = dbGetQuery(mycon,SQL)
> tab[1:3,]
     probe_id      path_id
1    36512_at      00623
2    36512_at      00650
3    36512_at      00960
```

Schema information

```
> hgu95av2_dbschema()


-- The "genes" table is the central table.
CREATE TABLE genes (
  _id INTEGER PRIMARY KEY,
  gene_id VARCHAR(10) NOT NULL UNIQUE          -- Entrez Gene ID
);

-- Data linked to the "genes" table.
CREATE TABLE probes (
  probe_id VARCHAR(80) PRIMARY KEY,            -- manufacturer ID
  accession VARCHAR(20) NULL,                  -- GenBank accession number
  _id INTEGER NULL,                            -- REFERENCES genes
  FOREIGN KEY (_id) REFERENCES genes (_id)
);
```

QC Data

> hgu95av2()

This package has the following mappings:

hgu95av2ACCNUM has 12625 mapped keys (of 12625 keys)

hgu95av2ALIAS2PROBE has 36251 mapped keys (of 36251 keys)

hgu95av2CHR has 12056 mapped keys (of 12625 keys)

hgu95av2CHRLENGTHS has 25 mapped keys (of 25 keys)

...

Additional Information about this package:

DB schema: HUMANCHIP_DB

DB schema version: 1.0

Organism: Homo sapiens

Date for NCBI data: 2007-Oct7

Date for GO data: 200709

Date for KEGG data: 2007-Oct7

Date for Golden Path data: 2006-Apr14

Date for IPI data: 2007-Aug08

Date for Ensembl data: 2007-Oct24

SQLite Bonus Feature:

- ATTACH
  - ATTACH “foo.sqlite” AS bar;

Results in the database being attached to the current session.

All the tables inside of “foo.sqlite” are now available as bar.table

SELECT * FROM bar.table1;

SELECT _id FROM bar.table2;

etc

Joining across databases

```
> mycon = hgu95av2_dbconn()
> GOdbloc = system.file("extdata", "GO.sqlite", package="GO.db")
> attachSql = paste("ATTACH '", GOdbloc, "' as go;", sep = "")
> dbGetQuery(mycon, attachSql)
> sql = paste("SELECT  DISTINCT a.go_id AS 'hgu95av2.go_id',",
+             "a._id AS 'hgu95av2._id',",
+             "g.go_id AS 'GO.go_id', g._id AS 'GO._id',",
+             "g.ontology",
+             "FROM go_bp_all AS a, go.go_term AS g",
+             "WHERE a.go_id = g.go_id LIMIT 10;")
> data = dbGetQuery(mycon, sql)
> data
```

Joining across databases

	hgu95av2.go_id	hgu95av2._id	GO.go_id	GO._id	ontology
1	GO:0000002	258	GO:0000002	11	BP
2	GO:0000002	1638	GO:0000002	11	BP
3	GO:0000002	3827	GO:0000002	11	BP
4	GO:0000002	4707	GO:0000002	11	BP
5	GO:0000003	41	GO:0000003	12	BP
6	GO:0000003	43	GO:0000003	12	BP
7	GO:0000003	81	GO:0000003	12	BP
8	GO:0000003	83	GO:0000003	12	BP
9	GO:0000003	105	GO:0000003	12	BP
10	GO:0000003	106	GO:0000003	12	BP

SQLForge

- Makes SQLite DBs and packages them into standardized annotation packages
- Uses standard templates to make packages and standard schemas to make the databases underlying these
- The data are obtained from intermediate databases. (human.db0, mouse.db0 etc.)

Making a sqlite database I

```
#requirement #1
> source("http://bioconductor.org/biocLite.R")
> biocLite("human.db0")

#requirement #2
> hgu95av2_IDs = system.file("extdata",
+                            "hgu95av2_ID",
+                            package="AnnotationDbi")

> myMeta = c("DBSCHEMA" = "HUMANCHIP_DB",
+            "ORGANISM" = "Homo sapiens",
+            "SPECIES" = "Human",
+            "MANUFACTURER" = "Affymetrix",
+            "CHIPNAME" = "Affymetrix Human Genome U95 Set Version 2",
+            "MANUFACTURERURL" = "http:www.affymetrix.com")
```

Making a sqlite database II

```
> popHUMANCHIPDB(affy = FALSE,
+                prefix = "hgu95av2Test",
+                fileName = hgu95av2_IDs,
+                metaDataSrc = myMeta,
+                baseMapType = "gb",
+                outputDir = getwd(),
+                printSchema = TRUE)
```

Making an annotation package

```
> seed <- new("AnnDbPkgSeed",
+                     Package = "hgu95av2Test.db",
+                     Version = "2.1.0",
+                     PkgTemplate = "HUMANCHIP.DB",
+                     AnnObjPrefix = "hgu95av2Test")

> makeAnnDbPkg(seed,
+                         "hgu95av2Test.sqlite",
+                         dest_dir = getwd())
```

Making a DB package with 1 call

```
> makeHUMANCHIP_DB(affy=FALSE,
+                  prefix="hgu95av2",
+                  fileName=hgu95av2_IDs,
+                  baseMapType="gb",
+                  outputDir = getwd(),
+                  version="2.1.0",
+                  manufacturer = "Affymetrix",
+                  chipName = "Affymetrix Human Genome U95 Set Version 2",
+                  manufacturerUrl = "http://www.affymetrix.com")
```

END

“cut” slides with other examples that seemed too complicated for presentation follow...

Another join

```
> con = GO_dbconn()
> query = paste("SELECT go_ont.go_id, go_ont.ont,",
 +     "go_ont_name.ont_name FROM go_ont,",
 +    "go_ont_name WHERE (go_ont.ont = go_ont_name.ont)")
> rs = dbSendQuery(con, query)
> f3 = fetch(rs, n=3)
> f3
       go_id ont           ont_name
1 GO:0004497  MF Molecular Function
2 GO:0005489  MF Molecular Function
3 GO:0005506  MF Molecular Function


> dbClearResult(rs)
```

A more complex example

```
> query = paste("SELECT g1.*, g2.evi FROM go_probe g1,",
+    "go_probe g2 WHERE  (g1.go_id = 'GO:0005737' ",
+    "and g2.go_id = 'GO:0005737') AND (g1.affy_id = g2.affy_id)",
+     "AND (g1.evi = 'IDA' AND g2.evi = 'ISS')")
> rs = dbSendQuery(con, query)
> fetch(rs)
```

	affy_id	go_id	evi	evi
1	41306_at	GO:0005737	IDA	ISS
2	1069_at	GO:0005737	IDA	ISS
3	38704_at	GO:0005737	IDA	ISS
4	39501_f_at	GO:0005737	IDA	ISS

A more complex example for mapping Symbols

```
queryAlias = function(x) {
  it = paste("('", paste(x, collapse="', '"), "')", sep="")
  paste("select _id, alias_symbol from alias",
        "where alias_symbol in", it, ");")
}

queryGeneinfo = function(x) {
  it = paste("('", paste(x, collapse="', '"), "')", sep="")
  paste("select _id, symbol from gene_info where",
        "symbol in", it, ");")
}
```

A more complex example for mapping Symbols

```
queryAlias = function(x) {
  it = paste("('", paste(x, collapse="', '"), "')", sep="")
  paste("select _id, alias_symbol from alias",
        "where alias_symbol in", it, ");")
}

queryGeneinfo = function(x) {
  it = paste("('", paste(x, collapse="', '"), "')", sep="")
  paste("select _id, symbol from gene_info where",
        "symbol in", it, ");")
}
```