

Lecture 6

Brad McNeney

2018-02-08

Databases

Relational database

- ▶ A relational database is a collection of tables.
 - ▶ As statisticians, we think of rows (a.k.a. records) as sampled units, and columns as the variables measured on the units.
- ▶ Rows of one table are related to rows of others by keys
 - ▶ Each table has a key (primary key) that uniquely identifies the rows.
 - ▶ Including the key of another table (a foreign key) allows us to link (relate) records of the two tables.

Simple example database

Example from [http://www.itl.nist.gov/div897/ctg/dm/sql_examples.htm]

- ▶ STATION table (ID is primary key)

ID	City	State	Lat_N	Long_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105
66	Caribou	ME	47	68

- ▶ STATS table (primary key suppressed; ID is foreign key)

ID	Month	Temp_F	Rain_I
13	1	57.4	0.31
13	7	91.7	5.15
44	1	27.3	0.18
44	7	74.8	2.11
66	1	6.7	2.1
66	7	65.8	4.52

Database software terminology

- ▶ Software referred to as relational database management systems (RDBMS).
 - ▶ Example implementations: MySQL, SQLite
- ▶ The language for querying the database is structured query language (SQL).
 - ▶ Similar implementations in all major RDBMS
- ▶ The database resides on a server, which we access with a client.
 - ▶ Large databases will be on special-purpose remote servers.
 - ▶ Client runs on your computer.
 - ▶ However, client/server model usually involves layers of security that make access difficult.
 - ▶ We'll work mostly with SQLite, for which the server runs on your computer.

Why databases in R?

- ▶ The data is stored in a RDBMS.
 - ▶ Our focus today
- ▶ Work with datasets that are too large to be stored in memory

SQLite databases in R

- ▶ The RSQLite package contains an SQLite client *and* server we can use from R
 - ▶ SQLite is an open-source RDBMS engine bundled with RSQLite
 - ▶ Interface with DB engine (connect, write to, etc.) *via* a common interface called DBI
 - ▶ Install RSQLite and DBI before starting
 - ▶ Load DBI; access functions from RSQLite with the `::` operator.

```
library(DBI)
mydb <- dbConnect(RSQLite::SQLite(), "my-db.sqlite")
dbDisconnect(mydb)
#> [1] TRUE
unlink("my-db.sqlite")
```

Notes

- ▶ The first line, `mydb <- dbConnect(RSQLite::SQLite(), "my-db.sqlite")`, creates a “connection” to an SQLite database stored in the file “my-db.sqlite”
 - ▶ `SQLite()` is called the “driver”. It handles all the RDBMS-specific details of how the client and server communicate.
 - ▶ The following argument is the file name for the database. Other drivers need details like username and password to connect.
 - ▶ Initially the database is empty
- ▶ The second line, `dbDisconnect(mydb)`, disconnects from the database, but does not remove it.
- ▶ Remove with `unlink()` (a base R command).
- ▶ `dbDisconnect()` and all other commands we will use of the form `db*` are generics from DBI

Aside: MySQL database driver in R

- ▶ Wasn't able to get easy access to a MySQL database on campus.
- ▶ In case it is of later use, here are the arguments to `dbConnect()` for connecting to a password-protected database for Stat 341.

```
con <- dbConnect(MySQL(), user='stataccess',  
                  password='<password goes here>',  
                  dbname='stat341',  
                  host='muncho.its.sfu.ca')  
dbListTables(con)
```

Creating STATION and STATS tables

```
wdb <- dbConnect(RSQLite::SQLite(), "wdb.sqlite")
STATION <- data.frame(ID=c(13,44,66),
  City = c("Phoenix","Denver","Caribou"),
  State = c("AZ","CO","ME"),
  Lat_N = c(33,40,47),
  Long_W = c(112,105,68))
STATS <- data.frame(row = 1:6,
  ID = c(13,13,44,44,66,66),
  Month = c(1,7,1,7,1,7),
  Temp_F = c(57.4,91.7,27.3,74.8,6.7,65.8),
  Rain_I = c(0.31,5.15,0.18,2.11,2.1,4.52))
dbWriteTable(wdb,name='STATION', value = STATION, overwrite=TRUE)
dbWriteTable(wdb,name='STATS', value = STATS, overwrite=TRUE)
dbListTables(wdb)
```

```
## [1] "STATION" "STATS"
```

Notes

- ▶ The `overwrite=TRUE` argument to the `dbWriteTable()` commands is necessary for this demo, to allow re-knitting the document, but is not necessary in general.
- ▶ Now that we have an example database, we can see how to use SQL to extract data.
- ▶ Will interleave SQL tutorial and DBI.

SQL: Retrieve data from a single table

- ▶ The SQL SELECT statements:
 - ▶ `SELECT * FROM STATION;`
 - ▶ `SELECT City, Lat_N FROM STATION;`
 - ▶ `SELECT Month, Temp_F, Rain_I FROM STATS;`
- ▶ Filtering with the WHERE clause:
 - ▶ `SELECT * from STATION WHERE Lat_N >= 40;`
- ▶ Most SQL interfaces require a `;` at the end of each statement, but not R DBI.

DBI: Example data retrieval.

```
dbGetQuery(wdb, "SELECT * from STATION")
```

```
##      ID      City State Lat_N Long_W
## 1 13 Phoenix    AZ    33    112
## 2 44  Denver    CO    40    105
## 3 66 Caribou    ME    47     68
```

```
dbGetQuery(wdb, "SELECT City, Lat_N from STATION")
```

```
##      City Lat_N
## 1 Phoenix    33
## 2  Denver    40
## 3 Caribou    47
```

```
dbGetQuery(wdb, "SELECT * from STATION WHERE Lat_N >= 40")
```

```
##      ID      City State Lat_N Long_W
## 1 44  Denver    CO    40    105
## 2 66 Caribou    ME    47     68
```

DBI: Notes on dbGetQuery()

- ▶ `dbGetQuery()` calls three functions:
 1. `dbSendQuery()` sends the query to the DB,
 2. `dbFetch()` fetches the “result set”, and
 3. `dbClearResult()` frees memory and other resources associated with the result set.
- ▶ If the result set is too large to fit in memory, you can split the fetching into batches.
 - ▶ Then need to call `dbSendQuery()`, `dbFetch()` and `dbClearResult()` yourself.

DBI: Batched queries

```
rs <- dbSendQuery(wdb, "SELECT * FROM STATS")
while (!dbHasCompleted(rs)) {
  df <- dbFetch(rs, n = 2) # use n to set size of subset
  print(df)
}
```

```
##   row ID Month Temp_F Rain_I
## 1   1 13     1   57.4   0.31
## 2   2 13     7   91.7   5.15
##   row ID Month Temp_F Rain_I
## 1   3 44     1   27.3   0.18
## 2   4 44     7   74.8   2.11
##   row ID Month Temp_F Rain_I
## 1   5 66     1    6.7   2.10
## 2   6 66     7   65.8   4.52
```

```
dbClearResult(rs)
```

DBI: Parametrized queries

- ▶ Can pass the same query with several different values of a parameter `x`.
 - ▶ Create a parameter in an SQL statement with `:<name>`
 - ▶ Bind a value to the parameter with `dbBind()`

```
rs <- dbSendQuery(wdb,"SELECT * FROM STATION WHERE Lat_N >= :x")
dbBind(rs,param = list(x=40))
dbFetch(rs)
```

##	ID	City	State	Lat_N	Long_W
## 1	44	Denver	CO	40	105
## 2	66	Caribou	ME	47	68

```
dbBind(rs,param=list(x=45))
dbFetch(rs)
```

##	ID	City	State	Lat_N	Long_W
## 1	66	Caribou	ME	47	68

```
dbClearResult(rs)
```


SQL: Joining tables

- ▶ The purpose of related tables is to reduce redundancy.
 - ▶ For example, all the info on the stations appears once in the STATION table, and need not be repeated in the STATS table.
- ▶ But what if we need the info on the stations and the weather data?
Need to “join” tables.
- ▶ Simplest join (inner join): `SELECT * from STATION, STATS
WHERE STATION.ID=STATS.ID`

DBI: Example joins

```
dbGetQuery(wdb,"SELECT * from STATION, STATS WHERE STATION.ID=STATS.ID")
```

##	ID	City	State	Lat_N	Long_W	row	ID	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	13	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	13	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	44	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	44	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	66	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	66	7	65.8	4.52

```
queryp1 <- "SELECT City, State, Month, Rain_I from STATION, STATS"  
queryp2 <- "WHERE STATION.ID=STATS.ID AND Month = 1 AND Lat_N >= 40"  
dbGetQuery(wdb,paste(queryp1,queryp2))
```

##	City	State	Month	Rain_I
## 1	Denver	CO	1	0.18
## 2	Caribou	ME	1	2.10

SQL: Left joins

- ▶ The inner join returns data for cities in **both** the STATION and STATS tables.
- ▶ If we want the union of all stations, use a left join.

- ▶ First add another station with no data in STATS
- ▶ Miami, FL is at Lat 26 and Long 80.
- ▶ Give Miami station ID 77.
- ▶ In SQL we'd add Miami and do the left join with

```
INSERT INTO STATION VALUE (77,'Miami','FL',26,80)
SELECT * FROM STATION
LEFT JOIN STATS ON STATION.ID = STATS.ID
```

- ▶ Many other types of SQL joins. See [\[https://www.tutorialspoint.com/sqlite/sqlite_using_joins.htm\]](https://www.tutorialspoint.com/sqlite/sqlite_using_joins.htm) for a summary of joins in SQLite.

DBI: Adding to a table and left join

```
miami <- data.frame(ID=77, City="Miami", State="FL", Lat_N=26, Long_W=80)
dbWriteTable(wdb, name='STATION', value = miami, append=TRUE)
qq<-"SELECT * FROM STATION LEFT JOIN STATS ON STATION.ID = STATS.ID"
dbGetQuery(wdb, qq)
```

##	ID	City	State	Lat_N	Long_W	row	ID	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	13	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	13	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	44	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	44	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	66	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	66	7	65.8	4.52
## 7	77	Miami	FL	26	80	NA	NA	NA	NA	NA

- Note: append=TRUE adds to the current table

SQL: Table indices

- ▶ A query like `SELECT * FROM STATION WHERE Lat_N >= 40` requires that the RDBMS read the `Lat_N` value in every row of `STATION` and return the rows where `Lat_N` is 40 or more
- ▶ Such a query can be made much faster by creating an “index” on `Lat_N`.
 - ▶ An index is a table in the database, sorted on the indexed variable.
 - ▶ See [<http://www.sqlite.org/queryplanner.html>] for a nice description of how indexing columns speeds up searches.

DBI: Create an index with dbExecute()

- Use dbExecute() to execute queries that do not return tabular data.

```
dbExecute(wdb, "CREATE INDEX indx ON STATION(Lat_N)")
```

```
## [1] 1
```

Exercises for Reading Week

1. Select the stations from the state of Colorado
2. Add Vancouver as a new station in the STATION table (Use BC as the "State" and the latitude 49 and longitude 123)
3. Do an inner join that returns a table with city, state, and temperatures for July from cities at north latitude 40 or less.
4. Repeat (3) as a left join.

More Exercises

5. Do a left join that returns a table with city, state, and temperatures for July from cities at north latitude 40 or more. (Vancouver excluded because we filter on month, and month is missing for Vancouver.)
6. Do a left join that returns a table with city, state, and temperatures from cities at north latitude 40 or less.

```
vancouver <- data.frame(ID=88, City="Vancouver", State="BC", Lat_N=49, Long_W=123)
dbWriteTable(wdb, name='STATION', value = vancouver, append=TRUE)
qq1<-"SELECT City, State, Temp_F FROM STATION"
qq2<-"LEFT JOIN STATS ON STATION.ID = STATS.ID WHERE Lat_N >= 40"
dbGetQuery(wdb, paste(qq1, qq2))
```

##		City	State	Temp_F
## 1		Denver	CO	27.3
## 2		Denver	CO	74.8
## 3		Caribou	ME	6.7
## 4		Caribou	ME	65.8
## 5		Vancouver	BC	NA

Clean up

```
dbDisconnect(wdb)  
unlink("wdb.sqlite")
```

Merging, selecting and filtering on data frames

Inner join data frames with merge()

- ▶ The merge() function in R does SQL-like joins on data frames.

```
STATION <- rbind(STATION,miami)
merge(STATION,STATS,by="ID")
```

##	ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	7	65.8	4.52

Left joining data frames with merge()

```
merge(STATION, STATS, by="ID", all.x=TRUE)
```

##	ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	7	65.8	4.52
## 7	77	Miami	FL	26	80	NA	NA	NA	NA

Join functions in dplyr

- In dplyr the functions for joining are more explicitly named

```
library(dplyr)
inner_join(STATION, STATS, by="ID")
```

##	ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	7	65.8	4.52

Left join function in dplyr

```
left_join(STATION, STATS, by="ID")
```

##	ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
## 1	13	Phoenix	AZ	33	112	1	1	57.4	0.31
## 2	13	Phoenix	AZ	33	112	2	7	91.7	5.15
## 3	44	Denver	CO	40	105	3	1	27.3	0.18
## 4	44	Denver	CO	40	105	4	7	74.8	2.11
## 5	66	Caribou	ME	47	68	5	1	6.7	2.10
## 6	66	Caribou	ME	47	68	6	7	65.8	4.52
## 7	77	Miami	FL	26	80	NA	NA	NA	NA

select() to select columns

- ▶ select() from dplyr can be used to select columns.
 - ▶ Can use different “helper” functions to select variables (help(select_helpers))

```
select(STATION, City, State)
```

```
##      City State
## 1 Phoenix   AZ
## 2  Denver   CO
## 3 Caribou   ME
## 4  Miami    FL
```

```
select(STATION, matches("L."))
```

```
##   Lat_N Long_W
## 1    33    112
## 2    40    105
## 3    47     68
## 4    26     80
```

Using filter() like WHERE

```
select(STATION,matches("L.")) %>% filter(Lat_N>=40)
```

##	Lat_N	Long_W
## 1	40	105
## 2	47	68

Combining join/select/filter with %>%

```
inner_join(STATION, STATS, by="ID") %>%  
  select(matches("._.")) %>% filter(Lat_N >= 40)
```

##	Lat_N	Long_W	Temp_F	Rain_I
## 1	40	105	27.3	0.18
## 2	40	105	74.8	2.11
## 3	47	68	6.7	2.10
## 4	47	68	65.8	4.52

Exercises with dplyr

1. Select the stations from the state of Colorado (`filter()`)
2. Select the City and State from the stations using the `select()` command from dplyr: `select(STATION, City, State)`
3. Add Vancouver as a new station in the STATION data frame. Use BC as the "State" and the latitude 49 and longitude 123. (`rbind()` or `bind_rows()`)
4. Do an inner join that returns a table with city, state, and temperatures for July from cities at north latitude 40 or less. (`inner_join()` and `select()`)
5. Repeat (3) as a left join. (`left_join()` and `select()`)