

Lecture 6

Brad McNeney

2019-01-31

Databases

Merging, selecting and filtering on data frames

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
 - ▶ Extract only what you need, when you need it.
 - ▶ See Stat 240

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)
```

```
## Warning: package 'DBI' was built under R version 3.4.4
```

```
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 example arguments to `dbConnect()` for connecting to a password-protected MySQL database.

```
con <- dbConnect(MySQL(), user='stataaccess',  
                  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's 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 AND City=='Denver'")
```

##	ID	City	State	Lat_N	Long_W
## 1	44	Denver	CO	40	105

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: `SELECT * from STATION, STATS WHERE STATION.ID=STATS.ID`

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

##	ID	City	State	Lat_N	Long_W	row	ID..7	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

DBI: Example joins

- ▶ With multiple tables it is safest to refer to variables by tablename.varname

```
queryp1 <- "SELECT STATION.City, STATION.State, STATS.Month, STATS.Rain_I"  
queryp2 <- "FROM STATION, STATS"  
queryp3 <- "WHERE STATION.ID=STATS.ID AND STATS.Month = 1"  
dbGetQuery(wdb,paste(queryp1,queryp2,queryp3))
```

```
##      City State Month Rain_I  
## 1 Phoenix   AZ     1   0.31  
## 2  Denver   CO     1   0.18  
## 3 Caribou   ME     1   2.10
```

SQL: Inner joins

- ▶ The above join is an example of an “inner” join, which returns only entries for IDs in both the STATION and STATS tables.
- ▶ Another more explicit way to do an inner join is with the INNER JOIN keyword.
 - ▶ The SELECT statement is modified to include only the first of the two tables.

```
queryp1 <- "SELECT STATION.*, STATS.Month, STATS.Rain_I FROM STATION"  
queryp2 <- "INNER JOIN STATS ON STATION.ID=STATS.ID WHERE STATS.Month=1"  
dbGetQuery(wdb,paste(queryp1,queryp2))
```

##	ID	City	State	Lat_N	Long_W	Month	Rain_I
## 1	13	Phoenix	AZ	33	112	1	0.31
## 2	44	Denver	CO	40	105	1	0.18
## 3	66	Caribou	ME	47	68	1	2.10

SQL: Left joins

- ▶ The inner join returns data for cities in **both** the STATION and STATS tables.
- ▶ If we want to return all cities in STATION, regardless of whether they have an entry in STATS, use a left join.
 - ▶ First add a station to 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..7	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] 0
```

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 inner-joins on data frames.

```
STATION <- rbind(STATION,miami)
merge(STATION,STATS,by="ID") # miami not in STATS so not in join
```

##	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)
```

```
## Warning: package 'bindrcpp' was built under R version 3.
```

```
##   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