Lecture 18: More Databases

Statistical Computing, 36-350

Monday November 23, 2015

Outline

- Recap of databases, and SQL
- Recap of interfacing R and SQL
- Operations across multiple tables

Databases versus data frames

- Data frames in R are actually tables in database lingo

R jargon	Database jargon
column	field
row	record
dataframe	table
types of the columns	table schema
bunch of dataframes	database
selections, `subset`	`SELECT ... FROM ... WHERE ... HAVING`
`aggregate`, `d*ply`	`GROUP BY`
`merge`	`JOIN`
`order`	`ORDER BY`

Databases are very useful for performing manipulations across different tables

They don't require the tables to be stored in memory, which allows for better speed and efficiency, also handles concurrency issues

The use a **client-server** model, and the standard language for database programming is called **SQL**

Connecting R to SQL

Before running the following, install the packages: `DBI`, `RSQLite`. Also, download the database file http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/baseball.db, and save it in your working directory.

```r
library(DBI)
library(RSQLite)
drv = dbDriver("SQLite")
con = dbConnect(drv, dbname="baseball.db")
```

`con` is now a persistent connection to the database `baseball.db`

```
dbListTables(con)                    # Get tables in the database
```

```
##  [1] "AllstarFull"         "Appearances"         "AwardsManagers"
##  [4] "AwardsPlayers"       "AwardsShareManagers" "AwardsSharePlayers"
##  [7] "Batting"             "BattingPost"         "Fielding"
## [10] "FieldingOF"          "FieldingPost"        "HallOfFame"
## [13] "Managers"            "ManagersHalf"        "Master"
## [16] "Pitching"            "PitchingPost"        "Salaries"
## [19] "Schools"             "SchoolsPlayers"      "SeriesPost"
## [22] "Teams"               "TeamsFranchises"     "TeamsHalf"
## [25] "sqlite_sequence"     "xref_stats"
```

```
dbListFields(con, "Batting")          # List fields in table Batting
```

```
##  [1] "playerID"  "yearID"    "stint"     "teamID"    "lgID"
##  [6] "G"         "G_batting" "AB"        "R"         "H"
## [11] "2B"        "3B"        "HR"        "RBI"       "SB"
## [16] "CS"        "BB"        "SO"        "IBB"       "HBP"
## [21] "SH"        "SF"        "GIDP"      "G_old"
```

```
batting = dbReadTable(con, "Batting") # Import a table as a data frame
dim(batting)
```

```
## [1] 93955    24
```

SELECT

The main tool in SQL is SELECT, which allows you to perform queries on a particular table in a database. It has the form:

```
SELECT columns or computations
  FROM table
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC|DESC]
  LIMIT offset,count;
```

Example: picking out columns

Pick out columns playerID, yearID, AB, H, HR from Batting, order the rows by yearID (increasing order by default), limit display to 10 rows:

```
dbGetQuery(con, paste("SELECT playerID, yearID, AB, H, HR",
                      "FROM Batting",
                      "ORDER BY yearID",
                      "LIMIT 10"))
```

```
##     playerID yearID  AB  H HR
## 1  abercda01   1871   4  0  0
## 2   addybo01   1871 118 32  0
## 3  allisar01   1871 137 40  0
## 4  allisdo01   1871 133 44  2
## 5  ansonca01   1871 120 39  0
## 6  armstbo01   1871  49 11  0
## 7  barkeal01   1871   4  1  0
## 8  barnero01   1871 157 63  0
## 9  barrebi01   1871   5  1  0
## 10 barrofr01   1871  86 13  0
```

Note that the R equivalent is:

```
head(batting[order(batting$yearID), c("playerID", "yearID", "AB", "H", "HR")],10)
```

```
##       playerID yearID  AB  H HR
## 142  abercda01   1871   4  0  0
## 472   addybo01   1871 118 32  0
## 1085 allisar01   1871 137 40  0
## 1106 allisdo01   1871 133 44  2
## 1894 ansonca01   1871 120 39  0
## 2172 armstbo01   1871  49 11  0
## 3711 barkeal01   1871   4  1  0
## 3833 barnero01   1871 157 63  0
## 3911 barrebi01   1871   5  1  0
## 4026 barrofr01   1871  86 13  0
```

Example: computing homerun averages

In R, to compute the average the homeruns `HR` per year, and sort in descending order:

```
library(plyr)
sort(daply(batting, .(yearID), function(df) { mean(df$HR, na.rm=TRUE)} ), dec=TRUE)[1:10]
```

```
##     1987     1996     1986     1999     1977     1985     2000     2004
## 5.300832 5.073620 4.730769 4.692699 4.601010 4.588535 4.525437 4.490115
##     1979     2001
## 4.487582 4.412288
```

How we would do this in SQL:

```
dbGetQuery(con, paste("SELECT yearID, AVG(HR) as AvgHR",
                      "FROM Batting",
                      "GROUP BY yearID",
                      "ORDER BY AvgHR DESC",
                      "LIMIT 10"))
```

```
##   yearID    AvgHR
## 1   1987 5.300832
## 2    1996 5.073620
## 3    1986 4.730769
## 4    1999 4.692699
## 5    1977 4.601010
## 6    1985 4.588535
## 7    2000 4.525437
## 8    2004 4.490115
## 9    1979 4.487582
## 10   2001 4.412288
```

Few notes:

- here we dynamically renamed the computed column `AVG(HR)` to be `AvgHR`
- the order of commands matters; try switching the order of `GROUP BY` and `ORDER BY`, and you'll get an error

Example: limiting records we average

Only compute homerun averages past 1990:

```r
dbGetQuery(con, paste("SELECT yearID, AVG(HR) as AvgHR",
                      "FROM Batting",
                      "WHERE yearID >= 1990",
                      "GROUP BY yearID",
                      "ORDER BY AvgHR DESC",
                      "LIMIT 10"))
```

```
##    yearID    AvgHR
## 1    1996 5.073620
## 2    1999 4.692699
## 3    2000 4.525437
## 4    2004 4.490115
## 5    2001 4.412288
## 6    2006 4.336554
## 7    2002 4.283658
## 8    1993 4.273595
## 9    2003 4.271534
## 10   1995 4.189938
```

```r
# Exercise: what is the R equivalent?
```

Example: limiting records we display

Only show homerun averages bigger than 4:

```r
dbGetQuery(con, paste("SELECT yearID, AVG(HR) as AvgHR",
                      "FROM Batting",
                      "WHERE yearID >= 1990",
                      "GROUP BY yearID",
                      "HAVING AvgHR >= 4",
                      "ORDER BY AvgHR DESC"))
```

```
##    yearID    AvgHR
## 1    1996 5.073620
## 2    1999 4.692699
## 3    2000 4.525437
## 4    2004 4.490115
## 5    2001 4.412288
## 6    2006 4.336554
## 7    2002 4.283658
## 8    1993 4.273595
## 9    2003 4.271534
## 10   1995 4.189938
## 11   1998 4.181668
## 12   2005 4.132619
## 13   1994 4.111940
## 14   1997 4.084507
## 15   2009 4.036829
```

```
# Exercise: what is the R equivalent?
```

JOIN

So far FROM has just been one table. Sometimes we need to combine information from many tables. SQL can do this without needing to read them each into memory

```
dbListFields(con, "Salaries")
```

```
## [1] "yearID"   "teamID"   "lgID"     "playerID" "salary"
```

```
dbGetQuery(con, "SELECT * FROM Salaries LIMIT 5")
```

```
##   yearID teamID lgID  playerID salary
## 1   1980    TOR   AL stiebda01  55000
## 2   1981    NYA   AL jacksre01 588000
## 3   1981    TOR   AL stiebda01  85000
## 4   1982    TOR   AL stiebda01 250000
## 5   1983    TOR   AL stiebda01 450000
```

```
dbGetQuery(con, "SELECT yearID, teamID, lgID, playerID, HR FROM Batting LIMIT 5")
```

```
##   yearID teamID lgID  playerID HR
## 1   2004    SFN   NL aardsda01  0
## 2   2006    CHN   NL aardsda01  0
## 3   2007    CHA   AL aardsda01  0
## 4   2008    BOS   AL aardsda01  0
## 5   2009    SEA   AL aardsda01  0
```

Suppose we want to figure out the average salaries of the players with the top 10 highest homerun averages. We'd have to combine these two data frames

- In R, we'd use merge() to link the tables up by playerID (or do it manually)
- SQL doesn't have merge(), but it has JOIN as a modifier to FROM

Practice problems

Enter your unique ID here:

Work through the following problems (go ahead and fill in code where needed)

```
# 1. Our first attempt using JOIN
dbGetQuery(con, paste("SELECT playerID, salary, HR",
                      "FROM Batting JOIN Salaries USING(playerID)",
                      "LIMIT 10"))
```

```
##     playerID  salary HR
## 1  aardsda01  300000  0
## 2  aardsda01  387500  0
## 3  aardsda01  403250  0
## 4  aardsda01  419000  0
## 5  aardsda01 2750000  0
## 6  aardsda01  300000  0
## 7  aardsda01  387500  0
## 8  aardsda01  403250  0
## 9  aardsda01  419000  0
## 10 aardsda01 2750000  0
```

```
# You can see that we've attempted to join the two tables using playerID

# What happened? We can see that there are (at least) 10 records of the player
# aardsda01 ...
# Write SQL code below to confirm that there are only 6 records of this player
# in Salaries, and 5 of this player in the Salaries table. Hence there are far
# too many in the joined table ...
# (Hint: use WHERE, and note SQL requires ' ' to define a string)
```

```
# 2. Our second attempt using JOIN
dbGetQuery(con, paste("SELECT yearID, playerID, salary, HR",
                      "FROM Batting JOIN Salaries USING(yearID, playerID)",
                      "LIMIT 10"))
```

```
##    yearID  playerID salary HR
## 1    2004 aardsda01 300000  0
## 2    2007 aardsda01 387500  0
## 3    2008 aardsda01 403250  0
## 4    2009 aardsda01 419000  0
## 5    1986  aasedo01 600000 NA
## 6    1987  aasedo01 625000 NA
## 7    1988  aasedo01 675000 NA
## 8    1989  aasedo01 400000  0
## 9    2006  abadan01 327000  0
## 10   1998 abbotje01 175000 12
```

```
# You can see that we've attempted to join the two tables using yearID and
# playerID, both. This makes sense because each player has a different record
# (row) for each year of their career

# What happened now? There are only 5 records for the player aardsda01. Write
# SQL code to confirm that these are the properly merged records for this player.
# That is, look at the records in the Batting and Salaries data tables for this
# player, and look at the yearID field
```

```
# 3. Modify the code below which derives the merged data table (same as above),
# to list the names of the players with the top 10 average homeruns, along with
# their average salaries
# (Hint: building on top of this code, you can just pretend like you have one big
# table with fields from both Batting and Salaries; use AVG, GROUP BY, ORDER BY)

dbGetQuery(con, paste("SELECT playerID, salary, HR",
                      "FROM Batting JOIN Salaries USING(yearID, playerID)",
                      "LIMIT 10"))
```

```
##     playerID salary HR
## 1  aardsda01 300000  0
## 2  aardsda01 387500  0
## 3  aardsda01 403250  0
## 4  aardsda01 419000  0
## 5   aasedo01 600000 NA
## 6   aasedo01 625000 NA
## 7   aasedo01 675000 NA
## 8   aasedo01 400000  0
## 9   abadan01 327000  0
## 10 abbotje01 175000 12
```

```
# 4. Modify the code below which derives the merged data table (same as above),
# to list the names of the players with the top 10 average salaries, along with
# their average homeruns

dbGetQuery(con, paste("SELECT playerID, salary, HR",
                      "FROM Batting JOIN Salaries USING(yearID, playerID)",
                      "LIMIT 10"))
```

```
##     playerID salary HR
## 1  aardsda01 300000  0
## 2  aardsda01 387500  0
## 3  aardsda01 403250  0
## 4  aardsda01 419000  0
## 5   aasedo01 600000 NA
## 6   aasedo01 625000 NA
## 7   aasedo01 675000 NA
## 8   aasedo01 400000  0
## 9   abadan01 327000  0
## 10 abbotje01 175000 12
```

```
# 5. Write SQL code to list the 10 worst seasons in terms of errors committed E
# by an individual player. You should display the year and player name, along with
# their error count E. Also, restrict your search to years 1990 and later
# (Hint: the error column E is available as part of the Fielding table; use WHERE
# to restrict your search)

dbListFields(con,"Fielding")
```

```
##  [1] "playerID"    "yearID"      "stint"       "teamID"      "lgID"
##  [6] "POS"         "G"           "GS"          "InnOuts"     "PO"
## [11] "A"           "E"           "DP"          "PB"          "WP"
## [16] "SB"          "CS"          "pickoffs"    "ZR"          "missing_g_c"
## [21] "missing_g"
```

```
# 6. Write SQL code to appropriately JOIN the Fielding and Salaries data tables,
# and list the salaries for the player / year combinations you extracted in the
# previous question
```

```
# 7. Write SQL code to answer the following question: what was the highest salary
# paid to a player who made more than 30 errors in a season, after 1990?
```