Databases

James M. Flegal

Agenda

- Thanks to Prof. Cosma Shalizi (CMU Statistics) for this material
- What databases are, and why
- SQL
- Interfacing R and SQL

Databases

- A **record** is a collection of **fields**
- A **table** is a collection of records which all have the same fields (with different values)
- A **database** is a collection of tables

Databases vs. Dataframes

- R's dataframes are actually tables

R jargon	Database jargon
column	field
row	record
dataframe	table
types of the columns	table **schema**
bunch of related dataframes	database

Why Do We Need Database Software?

- *Size*
  - R keeps its dataframes in memory
  - Industrial databases can be much bigger
  - Work with selected subsets
- *Speed*
  - Clever people have worked very hard on getting just what you want fast
- *Concurrency*
  - Many users accessing the same database simultaneously
  - Lots of potential for trouble (two users want to change the same record at once)

The Client-Server Model

- Databases live on a **server**, which manages them
- Users interact with the server through a **client** program
- Lets multiple users access the same database simultaneously

SQL

- **SQL** (**structured query language**) is the standard for database software
- Mostly about **queries**, which are like doing a selection in R

```
debt[debt$Country=="France",c("growth","ratio")]
with(debt,debt[Country=="France",c("growth","ratio")])
subset(x=debt,subset=(Country=="France"),select=c("growth","ratio"))
```

- Let's look at how SQL does stuff like this

SELECT

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

- SELECT is the first word of a query, then modifiers say which fields/columns to use, and what conditions records/rows must meet, from which tables
- The final semi-colon is obligatory

SELECT

```
SELECT PlayerID,yearID,AB,H FROM Batting;
```

- Four columns from table `Batting`

```
SELECT * FROM Salaries;
```

- All columns from table `Salaries`

```
SELECT * FROM Salaries ORDER BY Salary;
```

- As above, but by ascending value of `Salary`

```
SELECT * FROM Salaries ORDER BY Salary DESC;
```

- Descending order

```
SELECT * FROM Salaries ORDER BY Salary DESC LIMIT 10;
```

- Top 10 salaries

SELECT

- Picking out rows meeting a condition

```
SELECT PlayerID,yearID,AB,H FROM Batting WHERE AB > 100 AND H > 0;
```

- Where R would use

```
Batting[Batting$AB>100 & Batting$H > 0, c("PlayerID","yearID","AB","H")]
```

Calculated Columns

- SQL knows about some simple summary statistics:

```
SELECT MIN(AB), AVG(AB), MAX(AB) FROM Batting;
```

- It can do arithmetic

```
SELECT AB,H,H/CAST(AB AS REAL) FROM Batting;
```

- Because `AB` and `H` are integers, and it won't give you a fractional part by default
- Calculated columns can get names

```
SELECT PlayerID,yearID,H/CAST(AB AS REAL) AS BattingAvg FROM Batting
  ORDER BY BattingAvg DESC LIMIT 10;
```

Aggregating

- We can do calculations on value-grouped subsets, like in `aggregate` or `d*ply`

```
SELECT playerID, SUM(salary) FROM Salaries GROUP BY playerID
```

Selecting Again

- First cut of records is with `WHERE`
- Aggregation of record with `GROUP BY`
- Post-aggregation selection with `HAVING`

```
SELECT playerID, SUM(salary) AS totalSalary FROM Salaries GROUP BY playerID
  HAVING totalSalary > 200000000
```

JOIN

- So far `FROM` has just been one table
- Sometimes we need to combine information from many tables

patient_last	patient_first	physician_id	complaint
Morgan	Dexter	37010	insomnia
Soprano	Anthony	79676	malaise
Swearengen	Albert	NA	healthy as a goddam horse
Garrett	Alma	90091	nerves
Holmes	Sherlock	43675	nicotine-patch addiction

physician_last	physician_first	physicianID	plan
Meridian	Emmett	37010	UPMC
Melfi	Jennifer	79676	BCBS
Cochran	Amos	90091	UPMC
Watson	John	43675	VA

JOIN

- Suppose we want to know which doctors are treating patients for insomnia
  - Complaints are in one table
  - Physicians are in the other
- In R, we'd use `merge` to link the tables up by `physicianID`
- Here, `physician_id` or `physicianID` is acting as the **key** or **unique identifier**

JOIN

- SQL doesn't have `merge`, it has `JOIN` as a modifier to `FROM`

```
SELECT physician_first, physician_last FROM patients INNER JOIN physicians
   ON patients.physician_id == physicians.physicianID WHERE condition=="insomnia"
```

- Creates a (virtual) table linking records where `physician_id` in one table matches `physicianID` in the other
- If the names were the same in the two tables, we could write (e.g.)

```
SELECT nameLast,nameFirst,yearID,AB,H FROM Master INNER JOIN Batting
  USING(playerID);
```

`INNER JOIN ... USING` links records with the same value of `playerID`

JOIN

- `LEFT OUTER JOIN` includes records from the first table which don't match any record in the 2nd
  - The "extra" records get `NA` in the 2nd table's fields
- `RIGHT OUTER JOIN` is just what you'd think
  - so is `FULL OUTER JOIN`

Updated Translation Table

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`

Connecting R to SQL

- SQL is a language; database management systems (DMBS) actually implement it and do the work
  - MySQL, SQLite, etc., etc.
- They all have somewhat different conventions
- The R package `DBI` is a unified interface to them
- Need a separate "driver" for each DBMS

Connecting R to SQL

```
install.packages("DBI", dependencies = TRUE) # Install DBI
install.packages("RSQLite", dependencies = TRUE) # Install driver for SQLite
library(RSQLite)
drv <- dbDriver('SQLite')
con <- dbConnect(drv, dbname="baseball.db")
```

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

Connecting R to SQL

```
dbListTables(con)          # Get tables in the database (returns vector)
dbListFields(con, name)  # List fields in a table
dbReadTable(con, name)   # Import a table as a data frame
```

Connecting R to SQL

```
dbGetQuery(conn, statement)
df <- dbGetQuery(con, paste(
  "SELECT nameLast,nameFirst,yearID,salary",
  "FROM Master NATURAL JOIN Salaries"))
```

Connecting R to SQL

Usual workflow:

- Load the driver, connect to the right database
- R sends an SQL query to the DBMS
- SQL **executes the query**, sending back a manageably small dataframe
- R does the actual statistics
- Close the connection when you're done

Going the Other Way

- The `sqldf` package lets you use SQL commands on dataframes
- Mostly useful if you already know SQL better than R...

Summary

- A database is basically a way of dealing efficiently with lots of potentially huge dataframes
- SQL is the standard language for telling databases what to do, especially what queries to run
- Everything in an SQL query is something we've practiced already in R
  - subsetting/selection, aggregation, merging, ordering
- Connect R to the database, send it an SQL query, analyse the returned dataframe
- More information at [http://www.stat.cmu.edu/~cshalizi/statcomp/14/]