Data munging with SQL and R

Joshua Reich

josh@i2pi.com

Jan 7, 2010
Overview of SQL

The same query in R

There are better ways

data.table
SQL

- Structured Query Language
- Relational Data Model
- Procedural Logic Support
- PostgreSQL is great
josh=# \d series

Table "fred.series"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>series_id</td>
<td>integer</td>
<td>not null default nextval.</td>
</tr>
<tr>
<td>fred_id</td>
<td>character varying(16)</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>frequency</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>units</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>adjustment</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>notes</td>
<td>text</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

"series_pkey" PRIMARY KEY, btree (series_id)
"series_fred_id" UNIQUE, btree (fred_id)
"series_series_id" UNIQUE, btree (series_id)
josh=# SELECT * FROM series LIMIT 1;
- [ RECORD 1 ]--------------------------------------------------------

series_id | 196839
fred_id    | AAA
title      | Moody’s Seasoned Aaa Corporate Bond Yield
frequency  | Monthly
units      | Percent
adjustment | Not Applicable
notes      | Averages of daily data. Reprinted with permission from Moody’s Investors Services. Copyright. Moody’s tries to include bonds with...
josh=# \d data

Table "fred.data"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>series_id</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>timestamp without time zone</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>double precision</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

"data_series_id" btree (series_id)
```
joah=# SELECT * 
    FROM data 
    WHERE series_id = 196839 
    ORDER BY date;

<table>
<thead>
<tr>
<th>series_id</th>
<th>date</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>196839</td>
<td>1919-01-01 00:00:00</td>
<td>5.35</td>
</tr>
<tr>
<td>196839</td>
<td>1919-02-01 00:00:00</td>
<td>5.35</td>
</tr>
<tr>
<td>196839</td>
<td>1919-03-01 00:00:00</td>
<td>5.39</td>
</tr>
<tr>
<td>196839</td>
<td>1919-04-01 00:00:00</td>
<td>5.44</td>
</tr>
<tr>
<td>196839</td>
<td>1919-05-01 00:00:00</td>
<td>5.39</td>
</tr>
<tr>
<td>196839</td>
<td>1919-06-01 00:00:00</td>
<td>5.4</td>
</tr>
<tr>
<td>196839</td>
<td>1919-07-01 00:00:00</td>
<td>5.44</td>
</tr>
<tr>
<td>196839</td>
<td>1919-08-01 00:00:00</td>
<td>5.56</td>
</tr>
<tr>
<td>196839</td>
<td>1919-09-01 00:00:00</td>
<td>5.6</td>
</tr>
<tr>
<td>196839</td>
<td>1919-10-01 00:00:00</td>
<td>5.54</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
```
SELECT
    substring(title from 22),
    min(value)::numeric(4,2),
    avg(value)::numeric(4,2),
    max(value)::numeric(4,2)
FROM
    series s INNER JOIN data d ON (s.series_id = d.series_id)
WHERE
    LENGTH(fred_id) = 5 AND
    fred_id LIKE '%URN' AND
    fred_id NOT IN ('D8URN', 'DSURN')
GROUP BY title
ORDER BY avg(value) DESC;
<table>
<thead>
<tr>
<th>substring</th>
<th>min</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Virginia</td>
<td>4.00</td>
<td>8.60</td>
<td>19.60</td>
</tr>
<tr>
<td>Alaska</td>
<td>5.10</td>
<td>8.11</td>
<td>12.80</td>
</tr>
<tr>
<td>Michigan</td>
<td>3.10</td>
<td>7.91</td>
<td>17.50</td>
</tr>
<tr>
<td>Mississippi</td>
<td>4.40</td>
<td>7.75</td>
<td>15.10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>North Dakota</td>
<td>1.70</td>
<td>4.14</td>
<td>8.10</td>
</tr>
<tr>
<td>South Dakota</td>
<td>2.10</td>
<td>3.76</td>
<td>7.10</td>
</tr>
<tr>
<td>Nebraska</td>
<td>1.80</td>
<td>3.46</td>
<td>7.80</td>
</tr>
</tbody>
</table>
An aside...

- INNER vs. OUTER joins
- median() ?
- PL/R is the bees knees
- SQL sucks for Time Series + Graphs (among others)
- RODBC vs. sqldf
# Cheat sheet

<table>
<thead>
<tr>
<th>SQL</th>
<th>R (base)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN</td>
<td>merge()</td>
</tr>
<tr>
<td>WHERE</td>
<td>subset()</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>aggregate()</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>order()</td>
</tr>
</tbody>
</table>

...kinda.
I say 'kinda' because while it is possible to shoe-horn a one-to-one mapping of SQL clauses with R functions, R generally has better ways of going about things. In the example that we are about to walk through, experienced R users will think of more R-esque ways of doing things, but the goal here is to get as close to one-to-one with methodology and output.
library(RODBC)
ch <- odbcConnect('josh')

series <- sqlFetch(ch, 'fred.series')
data <- sqlFetch(ch, 'fred.data')

dim(series)
[1] 19539 7

dim(data)
[1] 3878182 3

system.time(
  m<-merge(series, data)
)
  user  system elapsed
33.250 0.930  34.267
unemployment <- m[ grepl('^([A-Z]{2})URN\', m$fred_id) & !grepl('DSURN', m$fred_id), ]

nrow(unemployment)
[1] 19584

colnames(unemployment)
[1] "series_id"  "fred_id"   "title"
[4] "frequency"  "units"    "adjustment"
[7] "notes"      "date"      "value"
aMean  <- aggregate(list(mean=unemployment$value),
                  list(state=unemployment$title), mean)
apMin  <- aggregate(list(min=unemployment$value),
                  list(state=unemployment$title), min)
apMax  <- aggregate(list(max=unemployment$value),
                  list(state=unemployment$title), max)

tada  <- merge(merge(aMin,aMean),aMax)
tada$state <- substring(tada$state, 22)
tada  <- tada[order(tada$mean, decreasing=T),]
<table>
<thead>
<tr>
<th>State</th>
<th>Min</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Virginia</td>
<td>4.0</td>
<td>8.600781</td>
<td>19.6</td>
</tr>
<tr>
<td>Alaska</td>
<td>5.1</td>
<td>8.105208</td>
<td>12.8</td>
</tr>
<tr>
<td>Michigan</td>
<td>3.1</td>
<td>7.910938</td>
<td>17.5</td>
</tr>
<tr>
<td>Mississippi</td>
<td>4.4</td>
<td>7.748437</td>
<td>15.1</td>
</tr>
<tr>
<td>North Dakota</td>
<td>1.7</td>
<td>4.139844</td>
<td>8.1</td>
</tr>
<tr>
<td>South Dakota</td>
<td>2.1</td>
<td>3.758594</td>
<td>7.1</td>
</tr>
<tr>
<td>Nebraska</td>
<td>1.8</td>
<td>3.460938</td>
<td>7.8</td>
</tr>
</tbody>
</table>
That worked, but sucked.

- 617 keystrokes for R vs. 355 for SQL
- 42 sec runtime for R, 0.026 sec for Postgres
That worked, but sucked.

- 617 keystrokes for R vs. 355 for SQL → pplyr!
- 42 sec runtime for R, 0.026 sec for Postgres → data.table!
data.table is great for big data

- SQL is a query language atop a relational data model
- Explicit relations lead to indices
- Indices makes joins **fast**
- R, in general, (& sqldf, plyr, in particular) have no concept of indices
- data.table brings indices to data frames
**data.table basics**

```r
dt <- DT(df, key='colname1,colname2')
```

Takes a data frame \( (df) \) and creates a data.table \( (dt) \) which is **indexed** by `colname1` then `colname2` (for ties)

```r
dt[i, j]
```

- If `i` is not another data.table, then it works exactly like a data.frame for subsetting rows.
- If `i` is a **data.table**, then does a fast \( (O(\log(n))) \) join of `i`’s keys with `dt`’s. Returns intersection.
- If `j` is a **single column** index, then it works exactly like a data.frame for selecting a column.
- If `j` is a **data.table**, then it performs expressions in the scope of the data.table.
data.table options

dt[i, j, mult={'first', 'last', 'all'},
nomatch={0, NA},
roll={FALSE, TRUE},
by='colname']

• When performing a join (using a data.table as i), the default behavior is to only return the first match. You can tweak this with the mult option.

• Likewise, the default join behavior is an INNER JOIN. For an OUTER JOIN, specify nomatch = NA.

• When doing joins on time series, roll lets you join against imperfectly matched times by using the closest prior time.

• by lets you GROUP BY colname, for each group the j expressions are evaluated separately. (Inefficient according to docs)
Syntax abuse at its finest

library(data.table)

# Turn our data frames into tables, indexed by # series_id

sdt <- DT(series, key=’series_id’)
ddt <- DT(data, key=’series_id’)

# Create a data.table, u, with only the series # relating to state unemployment


# NB:
# - Our i clause is the same for subsetting a # data.frame.
# - j clause is optional.
# Syntax abuse at its finest

# Create a data.table by joining ddt against u, 
# grouping by series, and calculating our 
# stats on each group.

d <- ddt[ u, 
           DT(min=min(value),
               mean=mean(value),
               max=max(value)),
           by='series_id',
           mult='all' ]

# Now apply the same cleaning as we did in the 
# data.frame version.

data <- merge(d,series)[,c('title','min','mean','max')]
data$title <- substring(data$title, 22)
data <- data[order(data$mean, decreasing=T),]
## Tada!

<table>
<thead>
<tr>
<th>title</th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>West Virginia</td>
<td>4.0</td>
<td>8.600781</td>
<td>19.6</td>
</tr>
<tr>
<td>Alaska</td>
<td>5.1</td>
<td>8.105208</td>
<td>12.8</td>
</tr>
<tr>
<td>Michigan</td>
<td>3.1</td>
<td>7.910938</td>
<td>17.5</td>
</tr>
<tr>
<td>Mississippi</td>
<td>4.4</td>
<td>7.748437</td>
<td>15.1</td>
</tr>
<tr>
<td>North Dakota</td>
<td>1.7</td>
<td>4.139844</td>
<td>8.1</td>
</tr>
<tr>
<td>South Dakota</td>
<td>2.1</td>
<td>3.758594</td>
<td>7.1</td>
</tr>
<tr>
<td>Nebraska</td>
<td>1.8</td>
<td>3.460938</td>
<td>7.8</td>
</tr>
</tbody>
</table>

Even including the time taken to build the indices, this runs in **0.03** sec, compared to 42 sec when we used merge().
Learn More

- http://cran.r-project.org/web/packages/RODBC/
- http://cran.r-project.org/web/packages/data.table/
- http://cran.r-project.org/web/packages/sqldf/
- http://cran.r-project.org/web/packages/plr/
- http://www.joeconway.com/plr/
- http://research.stlouisfed.org/fred2/