Introduction to rsolr

Michael Lawrence

May 17, 2022

Contents

1 Introduction

The `rsolr` package provides an idiomatic (R-like) and extensible interface between R and Solr, a search engine and database. Like an onion, the interface consists of several layers, along a gradient of abstraction, so that simple problems are solved simply, while more complex problems may require some peeling and perhaps tears. The interface is idiomatic, syntactically but also in terms of *intent*. While Solr provides a search-oriented interface, we recognize it as a document-oriented database. While not entirely schemaless, its schema is extremely flexible, which makes Solr an effective database for prototyping and adhoc analysis. R is designed for manipulating data, so `rsolr` maps common R data manipulation verbs to the Solr database and its (limited) support for analytics. In other words, `rsolr` is for analysis, not search, which has presented some fun challenges in design. Hopefully it is useful — we had not tried it until writing this document.

We have interfaced with all of the Solr features that are relevant to data analysis, with the aim of implementing many of the fundamental data munging operations. Those operations are listed in the table below, along with how we have mapped those operations to existing and well-known functions in the base R API, with some important extensions. When called on `rsolr` data structures, those functions should behave analogously to the existing implementations for `data.frame`. Note that more complex operations, such as joining and reshaping tables, are best left to more sophisticated frameworks, and we encourage others to implement our extended base R API on top of such systems. After all, Solr is a search engine. Give it a break.

Operation	R function
Filtering	`subset`
Transformation	`transform`
Sorting	`sort`
Aggregation	`aggregate`

2 Demonstration: nycflights13

2.1 The Dataset

As part demonstration and part proof of concept, we will attempt to follow the introductory workflow from the `dplyr` vignette. The dataset describes all of the airline flights departing New York City in 2013. It is provided by the `nycflights13` package, so please see its documentation for more details.

```
library(nycflights13)
dim(flights)

## [1] 336776     19

head(flights)

## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
```

```
## 4  2013     1     1      544            545         -1     1004            1022
## 5  2013     1     1      554            600         -6      812             837
## 6  2013     1     1      554            558         -4      740             728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

2.2 Populating a Solr core

The first step is getting the data into a Solr *core*, which is what Solr calls a database. This involves writing a schema in XML, installing and configuring Solr, launching the server, and populating the core with the actual data. Our expectation is that most use cases of `rsolr` will involve accessing an existing, centrally deployed, usually read-only Solr instance, so those are typically not major concerns. However, to conveniently demonstrate the software, we need to violate all of those assumptions. Luckily, we have managed to embed an example Solr installation within `rsolr`. We also provide a mechanism for autogenerating a Solr schema from a `data.frame`. This could be useful in practice for producing a template schema that can be tweaked and deployed in shared Solr installations. Taken together, the process turns out to not be very intimidating.

We begin by generating the schema and starting the demo Solr instance. Note that this instance is really only meant for demonstrations. You should not abuse it like the people abused the poor built-in R HTTP daemon.

```
library(rsolr)

## Loading required package:  BiocGenerics
##
## Attaching package:  'BiocGenerics'
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, basename, cbind, colnames, dirname, do.call,
##     duplicated, eval, evalq, get, grep, grepl, intersect, is.unsorted,
##     lapply, mapply, match, mget, order, paste, pmax, pmax.int,
```

```
pmin,
##     pmin.int, rank, rbind, rownames, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which.max, which.min
##
## Attaching package:  'rsolr'
## The following object is masked from 'package:stats':
##
##     ftable
## The following object is masked from 'package:base':
##
##     grouping

schema <- deriveSolrSchema(flights)
solr <- TestSolr(schema)

## Starting Solr...
## Use options(verbose=TRUE) to diagnose any problems.
## Solr started at:  http://localhost:8983/solr/flights
```

Next, we need to populate the core with our data. This requires a way to interact with the core from R. **rsolr** provides direct access to cores, as well as two high-level interfaces that represent a dataset derived from a core (rather than the core itself). The two interfaces each correspond to a particular shape of data. *SolrList* behaves like a list, while *SolrFrame* behaves like a table (data frame). *SolrList* is useful for when the data are ragged, as is often the case for data stored in Solr. The Solr schema is so dynamic that we could trivially define a schema with a virtually infinite number of fields, and each document could have its own unique set of fields. However, since our data are tabular, we will use *SolrFrame* for this exercise.

```
sr <- SolrFrame(solr$uri)
```

Finally, we load our data into the Solr dataset:

```
sr[] <- flights
```

This takes a while, since Solr has to generate all sorts of indices, etc.

As *SolrFrame* behaves much like a base R data frame, we can retrieve the dimensions and look at the head of the dataset:

```
dim(sr)

## [1] 336776     19

head(sr)

## DocDataFrame (6x19)
##   year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
## 1 2013     1   1      517            515         2      830            819
## 2 2013     1   1      533            529         4      850            830
## 3 2013     1   1      542            540         2      923            850
## 4 2013     1   1      544            545        -1     1004           1022
## 5 2013     1   1      554            600        -6      812            837
## 6 2013     1   1      554            558        -4      740            728
##   arr_delay carrier flight tailnum origin dest air_time distance hour minute
## 1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
## 2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
## 3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
## 4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
## 5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
## 6        12      UA   1696  N39463    EWR  ORD      150      719    5     58
##             time_hour
## 1 2013-01-01 10:00:00
## 2 2013-01-01 10:00:00
## 3 2013-01-01 10:00:00
## 4 2013-01-01 10:00:00
## 5 2013-01-01 11:00:00
## 6 2013-01-01 10:00:00
```

Comparing the output above the that of the earlier call to `head(flights)` reveals that the data are virtually identical. As Solr is just a search engine (on steroids), a significant amount of engineering was required to achieve that result.

2.3 Restricting by row

The simplest operation is filtering the data, i.e., restricting it to a subset of interest. Even a search engine should be good at that. Below, we use `subset` to restrict to the flights to those departing on January 1 (2013).

```r
subset(sr, month == 1 & day == 1)

## 'flights' (ndoc:842, nfield:19)
##     year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   1 2013     1   1      517            515         2      830            819
##   2 2013     1   1      533            529         4      850            830
##   3 2013     1   1      542            540         2      923            850
##   4 2013     1   1      544            545        -1     1004           1022
##   5 2013     1   1      554            600        -6      812            837
## ...  ...   ... ...      ...            ...       ...      ...            ...
## 838 2013     1   1     2356           2359        -3      425            437
## 839 2013     1   1     <NA>           1630      <NA>     <NA>           1815
## 840 2013     1   1     <NA>           1935      <NA>     <NA>           2240
## 841 2013     1   1     <NA>           1500      <NA>     <NA>           1825
## 842 2013     1   1     <NA>            600      <NA>     <NA>            901
##     arr_delay carrier flight tailnum origin dest air_time distance hour minute
##   1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
##   2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
##   3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
##   4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
##   5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
## ...       ...     ...    ...     ...    ...  ...      ...      ...  ...    ...
## 838       -12      B6    727  N588JB    JFK  BQN      186     1576   23     59
## 839      <NA>      EV   4308  N18120    EWR  RDU     <NA>      416   16     30
## 840      <NA>      AA    791  N3EHAA    LGA  DFW     <NA>     1389   19     35
## 841      <NA>      AA   1925  N3EVAA    LGA  MIA     <NA>     1096   15      0
## 842      <NA>      B6    125  N618JB    JFK  FLL     <NA>     1069    6      0
##               time_hour
##   1 2013-01-01 10:00:00
##   2 2013-01-01 10:00:00
##   3 2013-01-01 10:00:00
##   4 2013-01-01 10:00:00
##   5 2013-01-01 11:00:00
## ...                 ...
## 838 2013-01-02 04:00:00
## 839 2013-01-01 21:00:00
## 840 2013-01-02 00:00:00
## 841 2013-01-01 20:00:00
## 842 2013-01-01 11:00:00
```

Note how the records at the bottom contain missing values. Solr does not provide any facilities for missing value representation, but we mimic it by excluding those fields from those documents.

We can also extract ranges of data using the canonical `window()` function:

```
window(sr, start=1L, end=10L)

## DocDataFrame (10x19)
##    year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##  1 2013     1   1      517            515         2      830            819
##  2 2013     1   1      533            529         4      850            830
##  3 2013     1   1      542            540         2      923            850
##  4 2013     1   1      544            545        -1     1004           1022
##  5 2013     1   1      554            600        -6      812            837
##  6 2013     1   1      554            558        -4      740            728
##  7 2013     1   1      555            600        -5      913            854
##  8 2013     1   1      557            600        -3      709            723
##  9 2013     1   1      557            600        -3      838            846
## 10 2013     1   1      558            600        -2      753            745
##    arr_delay carrier flight tailnum origin dest air_time distance hour minute
##  1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
##  2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
##  3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
##  4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
##  5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
##  6        12      UA   1696  N39463    EWR  ORD      150      719    5     58
##  7        19      B6    507  N516JB    EWR  FLL      158     1065    6      0
##  8       -14      EV   5708  N829AS    LGA  IAD       53      229    6      0
##  9        -8      B6     79  N593JB    JFK  MCO      140      944    6      0
## 10         8      AA    301  N3ALAA    LGA  ORD      138      733    6      0
##              time_hour
##  1 2013-01-01 10:00:00
##  2 2013-01-01 10:00:00
##  3 2013-01-01 10:00:00
##  4 2013-01-01 10:00:00
##  5 2013-01-01 11:00:00
##  6 2013-01-01 10:00:00
##  7 2013-01-01 11:00:00
##  8 2013-01-01 11:00:00
```

```
##  9 2013-01-01 11:00:00
## 10 2013-01-01 11:00:00
```

Or, as we have already seen, the more convenient:

```
head(sr, 10L)

## DocDataFrame (10x19)
##    year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##  1 2013     1   1      517            515         2      830            819
##  2 2013     1   1      533            529         4      850            830
##  3 2013     1   1      542            540         2      923            850
##  4 2013     1   1      544            545        -1     1004           1022
##  5 2013     1   1      554            600        -6      812            837
##  6 2013     1   1      554            558        -4      740            728
##  7 2013     1   1      555            600        -5      913            854
##  8 2013     1   1      557            600        -3      709            723
##  9 2013     1   1      557            600        -3      838            846
## 10 2013     1   1      558            600        -2      753            745
##    arr_delay carrier flight tailnum origin dest air_time distance hour minute
##  1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
##  2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
##  3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
##  4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
##  5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
##  6        12      UA   1696  N39463    EWR  ORD      150      719    5     58
##  7        19      B6    507  N516JB    EWR  FLL      158     1065    6      0
##  8       -14      EV   5708  N829AS    LGA  IAD       53      229    6      0
##  9        -8      B6     79  N593JB    JFK  MCO      140      944    6      0
## 10         8      AA    301  N3ALAA    LGA  ORD      138      733    6      0
##              time_hour
##  1 2013-01-01 10:00:00
##  2 2013-01-01 10:00:00
##  3 2013-01-01 10:00:00
##  4 2013-01-01 10:00:00
##  5 2013-01-01 11:00:00
##  6 2013-01-01 10:00:00
##  7 2013-01-01 11:00:00
##  8 2013-01-01 11:00:00
##  9 2013-01-01 11:00:00
```

```
## 10 2013-01-01 11:00:00
```

We could also call `:` to generate a contiguous sequence:

```
sr[1:10,]

## 'flights' (ndoc:10, nfield:19)
##    year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##  1 2013     1   1      517            515         2      830            819
##  2 2013     1   1      533            529         4      850            830
##  3 2013     1   1      542            540         2      923            850
##  4 2013     1   1      544            545        -1     1004           1022
##  5 2013     1   1      554            600        -6      812            837
##  6 2013     1   1      554            558        -4      740            728
##  7 2013     1   1      555            600        -5      913            854
##  8 2013     1   1      557            600        -3      709            723
##  9 2013     1   1      557            600        -3      838            846
## 10 2013     1   1      558            600        -2      753            745
##    arr_delay carrier flight tailnum origin dest air_time distance hour minute
##  1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
##  2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
##  3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
##  4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
##  5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
##  6        12      UA   1696  N39463    EWR  ORD      150      719    5     58
##  7        19      B6    507  N516JB    EWR  FLL      158     1065    6      0
##  8       -14      EV   5708  N829AS    LGA  IAD       53      229    6      0
##  9        -8      B6     79  N593JB    JFK  MCO      140      944    6      0
## 10         8      AA    301  N3ALAA    LGA  ORD      138      733    6      0
##              time_hour
##  1 2013-01-01 10:00:00
##  2 2013-01-01 10:00:00
##  3 2013-01-01 10:00:00
##  4 2013-01-01 10:00:00
##  5 2013-01-01 11:00:00
##  6 2013-01-01 10:00:00
##  7 2013-01-01 11:00:00
##  8 2013-01-01 11:00:00
##  9 2013-01-01 11:00:00
## 10 2013-01-01 11:00:00
```

Unfortunately, it is generally infeasible to randomly access Solr records by index, because numeric indexing is a foreign concept to a search engine. Solr does however support retrieval by a key that has a unique value for each document. These data lack such a key, but it is easy to add one and indicate as such to `deriveSolrSchema()`.

2.4 Sorting

To sort the data, we just call `sort()` and describe the order by passing a formula via the `by` argument. For example, we sort by year, breaking ties with month, then day:

```
sort(sr, by = ~ year + month + day)

## 'flights' (ndoc:336776, nfield:19)
##        year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##      1 2013     1   1      517            515         2      830            819
##      2 2013     1   1      533            529         4      850            830
##      3 2013     1   1      542            540         2      923            850
##      4 2013     1   1      544            545        -1     1004           1022
##      5 2013     1   1      554            600        -6      812            837
##    ...  ...   ... ...      ...            ...       ...      ...            ...
## 336772 2013    12  31     <NA>            705      <NA>     <NA>            931
## 336773 2013    12  31     <NA>            825      <NA>     <NA>           1029
## 336774 2013    12  31     <NA>           1615      <NA>     <NA>           1800
## 336775 2013    12  31     <NA>            600      <NA>     <NA>            735
## 336776 2013    12  31     <NA>            830      <NA>     <NA>           1154
##        arr_delay carrier flight tailnum origin dest air_time distance hour
##      1        11      UA   1545  N14228    EWR  IAH      227     1400    5
##      2        20      UA   1714  N24211    LGA  IAH      227     1416    5
##      3        33      AA   1141  N619AA    JFK  MIA      160     1089    5
##      4       -18      B6    725  N804JB    JFK  BQN      183     1576    5
##      5       -25      DL    461  N668DN    LGA  ATL      116      762    6
##    ...       ...     ...    ...     ...    ...  ...      ...      ...  ...
## 336772      <NA>      UA   1729    <NA>    EWR  DEN     <NA>     1605    7
## 336773      <NA>      US   1831    <NA>    JFK  CLT     <NA>      541    8
## 336774      <NA>      MQ   3301  N844MQ    LGA  RDU     <NA>      431   16
## 336775      <NA>      UA    219    <NA>    EWR  ORD     <NA>      719    6
## 336776      <NA>      UA    443    <NA>    JFK  LAX     <NA>     2475    8
##        minute           time_hour
```

```
##      1      15 2013-01-01 10:00:00
##      2      29 2013-01-01 10:00:00
##      3      40 2013-01-01 10:00:00
##      4      45 2013-01-01 10:00:00
##      5       0 2013-01-01 11:00:00
##    ...     ...                 ...
## 336772       5 2013-12-31 12:00:00
## 336773      25 2013-12-31 13:00:00
## 336774      15 2013-12-31 21:00:00
## 336775       0 2013-12-31 11:00:00
## 336776      30 2013-12-31 13:00:00
```

To sort in decreasing order, just pass `decreasing=TRUE` as usual:

```
sort(sr, by = ~ arr_delay, decreasing=TRUE)

## 'flights' (ndoc:336776, nfield:19)
##          year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##        1 2013     1   9      641            900      1301     1242           1530
##        2 2013     6  15     1432           1935      1137     1607           2120
##        3 2013     1  10     1121           1635      1126     1239           1810
##        4 2013     9  20     1139           1845      1014     1457           2210
##        5 2013     7  22      845           1600      1005     1044           1815
##      ...  ...   ... ...      ...            ...       ...      ...            ...
## 336772 2013     5   4     1816           1820        -4     2017           2131
## 336773 2013     5   2     1947           1949        -2     2209           2324
## 336774 2013     5   6     1826           1830        -4     2045           2200
## 336775 2013     5  20      719            735       -16      951           1110
## 336776 2013     5   7     1715           1729       -14     1944           2110
##          arr_delay carrier flight tailnum origin dest air_time distance hour
##        1      1272      HA     51  N384HA    JFK  HNL      640     4983    9
##        2      1127      MQ   3535  N504MQ    JFK  CMH       74      483   19
##        3      1109      MQ   3695  N517MQ    EWR  ORD      111      719   16
##        4      1007      AA    177  N338AA    JFK  SFO      354     2586   18
##        5       989      MQ   3075  N665MQ    JFK  CVG       96      589   16
##      ...       ...     ...    ...     ...    ...  ...      ...      ...  ...
## 336772         -74      AS      7  N551AS    EWR  SEA      281     2402   18
## 336773         -75      UA    612  N851UA    EWR  LAX      300     2454   19
## 336774         -75      AA    269  N3KCAA    JFK  SEA      289     2422   18
## 336775         -79      VX     11  N840VA    JFK  SFO      316     2586    7
```

```
## 336776        -86      VX     193  N843VA    EWR  SFO      315      2565   17
##        minute           time_hour
##      1      0 2013-01-09 14:00:00
##      2     35 2013-06-15 23:00:00
##      3     35 2013-01-10 21:00:00
##      4     45 2013-09-20 22:00:00
##      5      0 2013-07-22 20:00:00
##    ...    ...                 ...
## 336772     20 2013-05-04 22:00:00
## 336773     49 2013-05-02 23:00:00
## 336774     30 2013-05-06 22:00:00
## 336775     35 2013-05-20 11:00:00
## 336776     29 2013-05-07 21:00:00
```

2.5 Restricting by field

Just as we can use `subset` to restrict by row, we can also use it to restrict by column:

```
subset(sr, select=c(year, month, day))

## 'flights' (ndoc:336776, nfield:3)
##        year month day
##      1 2013     1   1
##      2 2013     1   1
##      3 2013     1   1
##      4 2013     1   1
##      5 2013     1   1
##    ...  ...   ... ...
## 336772 2013     9  30
## 336773 2013     9  30
## 336774 2013     9  30
## 336775 2013     9  30
## 336776 2013     9  30
```

The `select` argument is analogous to that of `subset.data.frame`: it is evaluated to set of field names to which the dataset is restricted. The above example is static, so it is equivalent to:

```r
sr[c("year", "month", "day")]

## 'flights' (ndoc:336776, nfield:3)
##         year month day
##       1 2013     1   1
##       2 2013     1   1
##       3 2013     1   1
##       4 2013     1   1
##       5 2013     1   1
##     ...  ...   ... ...
## 336772 2013     9  30
## 336773 2013     9  30
## 336774 2013     9  30
## 336775 2013     9  30
## 336776 2013     9  30
```

But with **subset** we can also specify dynamic expressions, including ranges:

```r
subset(sr, select=year:day)

## 'flights' (ndoc:336776, nfield:3)
##         year month day
##       1 2013     1   1
##       2 2013     1   1
##       3 2013     1   1
##       4 2013     1   1
##       5 2013     1   1
##     ...  ...   ... ...
## 336772 2013     9  30
## 336773 2013     9  30
## 336774 2013     9  30
## 336775 2013     9  30
## 336776 2013     9  30
```

And exclusion:

```r
subset(sr, select=-(year:day))

## 'flights' (ndoc:336776, nfield:16)
```

```
##        dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##      1      517            515         2      830            819        11
##      2      533            529         4      850            830        20
##      3      542            540         2      923            850        33
##      4      544            545        -1     1004           1022       -18
##      5      554            600        -6      812            837       -25
##    ...      ...            ...       ...      ...            ...       ...
## 336772     <NA>           1455      <NA>     <NA>           1634      <NA>
## 336773     <NA>           2200      <NA>     <NA>           2312      <NA>
## 336774     <NA>           1210      <NA>     <NA>           1330      <NA>
## 336775     <NA>           1159      <NA>     <NA>           1344      <NA>
## 336776     <NA>            840      <NA>     <NA>           1020      <NA>
##        carrier flight tailnum origin dest air_time distance hour minute
##      1      UA   1545  N14228    EWR  IAH      227     1400    5     15
##      2      UA   1714  N24211    LGA  IAH      227     1416    5     29
##      3      AA   1141  N619AA    JFK  MIA      160     1089    5     40
##      4      B6    725  N804JB    JFK  BQN      183     1576    5     45
##      5      DL    461  N668DN    LGA  ATL      116      762    6      0
##    ...     ...    ...     ...    ...  ...      ...      ...  ...    ...
## 336772      9E   3393    <NA>    JFK  DCA     <NA>      213   14     55
## 336773      9E   3525    <NA>    LGA  SYR     <NA>      198   22      0
## 336774      MQ   3461  N535MQ    LGA  BNA     <NA>      764   12     10
## 336775      MQ   3572  N511MQ    LGA  CLE     <NA>      419   11     59
## 336776      MQ   3531  N839MQ    LGA  RDU     <NA>      431    8     40
##                  time_hour
##      1 2013-01-01 10:00:00
##      2 2013-01-01 10:00:00
##      3 2013-01-01 10:00:00
##      4 2013-01-01 10:00:00
##      5 2013-01-01 11:00:00
##    ...                 ...
## 336772 2013-09-30 18:00:00
## 336773 2013-10-01 02:00:00
## 336774 2013-09-30 16:00:00
## 336775 2013-09-30 15:00:00
## 336776 2013-09-30 12:00:00
```

Solr also has native support for globs:

```r
sr[c("arr_*", "dep_*")]

## 'flights' (ndoc:336776, nfield:4)
##         arr_time arr_delay dep_time dep_delay
##       1      830        11      517         2
##       2      850        20      533         4
##       3      923        33      542         2
##       4     1004       -18      544        -1
##       5      812       -25      554        -6
##     ...      ...       ...      ...       ...
## 336772     <NA>      <NA>     <NA>      <NA>
## 336773     <NA>      <NA>     <NA>      <NA>
## 336774     <NA>      <NA>     <NA>      <NA>
## 336775     <NA>      <NA>     <NA>      <NA>
## 336776     <NA>      <NA>     <NA>      <NA>
```

While we are dealing with fields, we should mention that renaming is also (in principle) possible:

```r
### FIXME: broken in current Solr CSV writer
### rename(sr, tail_num = "tailnum")
```

2.6 Transformation

To compute new columns from existing ones, we can, as usual, call the `transform` function:

```r
sr2 <- transform(sr,
                 gain = arr_delay - dep_delay,
                 speed = distance / air_time * 60)
sr2[c("gain", "speed")]

## 'flights' (ndoc:336776, nfield:1)
##         gain
##       1    9
##       2   16
##       3   31
##       4  -17
##       5  -19
```

```
##    ...  ...
## 336772 <NA>
## 336773 <NA>
## 336774 <NA>
## 336775 <NA>
## 336776 <NA>
```

2.6.1 Advanced note

The `transform` function essentially quotes and evaluates its arguments in the given frame, and then adds the results as columns in the return value. Direct evaluation affords more flexibility, such as constructing a table with only the newly computed columns. By default, evaluation is completely eager — each referenced column is downloaded in its entirety. But we can make the computation lazier by calling `defer` prior to the evaluation via `with`:

```
with(defer(sr), data.frame(gain = head(arr_delay - dep_delay),
                           speed = head(distance / air_time * 60)))

##   gain    speed
## 1    9 370.0440
## 2   16 374.2731
## 3   31 408.3750
## 4  -17 516.7213
## 5  -19 394.1379
## 6   16 287.6000
```

Note that this approach, even though it is partially deferred, is potentially less efficient than `transform` two reasons:

1. It makes two requests to the database, one for each column,
2. The two result columns are downloaded eagerly, since the result must be a `data.frame` (and thus practicalities required us to take the `head` of each promised column prior to constructing the data frame).

We can work around the second limitation by using a more general form of data frame, the *DataFrame* object from S4Vectors:

```
with(defer(sr),
     S4Vectors::DataFrame(gain = arr_delay - dep_delay,
                          speed = distance / air_time * 60))

## DataFrame with 336776 rows and 2 columns
##                           gain                 speed
##          <SolrFunctionPromise> <SolrFunctionPromise>
## 1                            9             370.04404
## 2                           16             374.27313
## 3                           31               408.375
## 4                          -17              516.7213
## 5                          -19             394.13794
## ...                        ...                   ...
## 336772                      NA                    NA
## 336773                      NA                    NA
## 336774                      NA                    NA
## 336775                      NA                    NA
## 336776                      NA                    NA
```

Note that we did not need to take the `head` of the individual columns, since *DataFrame* does not require the data to be stored in-memory as a base R vector.

2.7 Summarization

Data summarization is about reducing large, complex data to smaller, simpler data that we can understand.

A common type of summarization is aggregation, which is typically defined as a three step process:

1. Split the data into groups, usually by the the interaction of some factor set,
2. Summarize each group to a single value,
3. Combine the summaries.

Solr natively supports the following types of data aggregation:

- `mean`,
- `min`, `max`,

- `median`, `quantile`,
- `var`, `sd`,
- `sum`,
- count (`table`),
- counting of unique values (for which we introduce `nunique`).

The rsolr package combines and modifies these operations to support high-level summaries corresponding to the R functions `any`, `all`, `range`, `weighted.mean`, `IQR`, `mad`, etc.

A prerequisite of aggregation is finding the distinct field combinations that correspond to each correspond to a group. Those combinations themselves constitute a useful summary, and we can retrieve them with `unique`:

```
unique(sr["tailnum"])

## DocDataFrame (4044x1)
##      tailnum
##    1  D942DN
##    2  N0EGMQ
##    3  N10156
##    4  N102UW
##    5  N103US
##  ...     ...
## 4040  N998AT
## 4041  N998DL
## 4042  N999DN
## 4043  N9EAMQ
## 4044    <NA>

unique(sr[c("origin", "tailnum")])

## DocDataFrame (7944x2)
##      origin tailnum
##    1    EWR  N0EGMQ
##    2    EWR  N10156
##    3    EWR  N102UW
##    4    EWR  N103US
```

```
##    5    EWR  N104UW
##  ...    ...     ...
## 7940    LGA  N998AT
## 7941    LGA  N998DL
## 7942    LGA  N999DN
## 7943    LGA  N9EAMQ
## 7944    LGA    <NA>
```

Solr also supports extracting the top or bottom N documents, after ranking by some field, optionally by group.

The convenient, top-level function for aggregating data is `aggregate`. To compute a global aggregation, we just specify the computation as an expression (via a named argument, mimicking `transform`):

```
aggregate(sr, delay = mean(dep_delay, na.rm=TRUE))

##      delay
## 1 12.63907
```

It is also possible to specify a function (as the `FUN` argument), which would be passed the entire frame.

As with `stats::aggregate`, we can pass a grouping as a formula:

```
delay <- aggregate(~ tailnum, sr,
                       count = TRUE,
                       dist = mean(distance, na.rm=TRUE),
                       delay = mean(arr_delay, na.rm=TRUE))
delay <- subset(delay, count > 20 & dist < 2000)
```

The special `count` argument is a convenience for the common case of computing the number of documents in each group.

Here is an example of using `nunique` and `ndoc`:

```
head(aggregate(~ dest, sr,
               nplanes = nunique(tailnum),
               nflights = ndoc(tailnum)))

##   dest nplanes nflights
## 1  ABQ     108      254
## 2  ACK      58      265
```

```
## 3  ALB     172      439
## 4  ANC       6        8
## 5  ATL    1180    17215
## 6  AUS     993     2439
```

There is limited support for dynamic expressions in the aggregation formula. At a minimum, the expression should evaluate to logical. For example, we can condition on whether the distance is more than 1000 miles.

```
head(aggregate(~ I(distance > 1000) + tailnum, sr,
               delay = mean(arr_delay, na.rm=TRUE)))

##   I(distance > 1000) tailnum     delay
## 1              FALSE  D942DN 31.500000
## 2              FALSE  N0EGMQ  8.986755
## 3              FALSE  N10156 13.701149
## 4              FALSE  N102UW  2.937500
## 5              FALSE  N103US -6.934783
## 6              FALSE  N104UW  1.804348
```

It also works for values naturally coercible to logical, such as using the modulus to identify odd numbers. For clarity, we label the variable using **`transform`** prior to aggregating.

```
head(aggregate(~ odd + tailnum, transform(sr, odd = distance %% 2),
               delay = mean(arr_delay, na.rm=TRUE)))

##     odd tailnum     delay
## 1 FALSE  D942DN 31.500000
## 2 FALSE  N0EGMQ  8.589520
## 3 FALSE  N10156  7.797753
## 4 FALSE  N102UW 19.000000
## 5 FALSE  N103US -7.285714
## 6 FALSE  N104UW 20.700000
```

Aggregate and subset in the same command, as with data.frame:

```
head(aggregate(~ tailnum, sr,
               subset = distance > 500,
               delay = mean(arr_delay, na.rm=TRUE)))
```

```
##   tailnum     delay
## 1  D942DN 31.500000
## 2  N0EGMQ  8.919580
## 3  N10156 12.009174
## 4  N102UW  2.937500
## 5  N103US -6.934783
## 6  N104UW  1.804348
```

Aggregate the entire dataset:

```
aggregate(sr, delay = mean(arr_delay, na.rm=TRUE))

##      delay
## 1 6.895377
```

3 Cleaning up

Having finished our demonstration, we kill our Solr server:

```
solr$kill()
```