Representing Data in Julia: The Design of DataArrays and DataFrames

John Myles White

June 26, 2014

Tabular Data

Senator	Bill 1	Bill 2	Bill 3
Byrd (D WV)	1	1	0
Chambliss (R GA)	NA	0	1

The Big Question

How should we represent tabular data in Julia?

Sub-Question 1

How do we represent a single missing value?

Starting from Scratch

Base Julia has no concept of `NULL` or `NA`

What Should `NA` Mean?

- Epistemological missingness
  - Absence of knowledge of value, but value exists
- Ontological missingness
  - Value does not exist

Epistemological Missingness

- `true || NA` should evaluate to `true`
- `false && NA` should evaluate to `false`

3 Ways to Implement `NA`

- Sentinel values
  - Treat `NaN` or similar value as `NA`
- Option types
  - Augment single values of other types with a Boolean mask
- Database strategy
  - Add one `NA` singleton value
  - Add `NA`-compatible data structures

The Sentinel Values Approach

- The approach taken by R
- Use special `Float64` value for `NA`:
  - `reinterpret(Float64, 0x7ff00000000007a2)`
- Use special `Int32` value for `NA`:
  - `-2147483648`
- Similar tricks for other types

Pros and Cons of Sentinel Values

- Pros
  - Type-stable
  - Imposes no indirection
    - Directly use machine ops
    - No additional memory requirements
  - `NaN` already implements `NA`-like semantics
    - Existing functions like + behave correctly
- Cons
  - Requires a sentinel value for every new type
  - Potential binary incompatibility with other systems
  - Vulnerable to compiler optimizations
    - `NaN + NA != NA + NaN` in some R builds
  - Discards a potentially useful value

Option Types

```
immutable Option{T}
    val::T
    na::Bool
end
```

Pros and Cons of Option Types

- Pros
  - Trivially extensible to new types
  - Well-understood behavior
- Cons
  - Array storage format is not appropriate for BLAS, etc.

Database Strategy

- Add a singleton value, `NA`, of `NAtype`
- Add an `Array{T}`-like type that accepts `Union(T, NAtype)`

DataArray Type

```
type DataArray{T}
    vals::Array{T}
    nas::BitArray
end
```

Pros and Cons of the Database Strategy

- Pros
  - Trivially extensible to new types
  - Can treat `NA`-free `DataArray` as an `Array`
- Cons
  - **Type-instability**
  - Reimplements a lot of `Array{T}` functionality

Type-Instability in DataArrays.jl

```
da = @data([1, 2, NA])
da[1] # The type this call returns is uncertain
      # Could be Int
      # Could be NAtype
```

What Should Julia Do?

- We've pursued the database strategy
- Going forward, I'd love to see experiments with Option types
- But ability to pass arrays with no copies to BLAS is a big win

Sub-Question 2

Should we store tables as rows or as columns?

Column-Oriented Database

```
{
    :Senator => ["Byrd (D WV)", "Chambliss (R GA)"],
    :Bill1 => [1, NA],
}
```

Row-Oriented Database

```
[
    {:Senator => "Byrd (D WV)", :Bill1 => 1},
    {:Senator => "Chambliss (R GA)", :Bill1 => NA},
}
```

Pros and Cons of Column-Oriented Storage

- Pros
  - Potentially faster (esp. for heterogeneous data)
  - Easier to integrate with Julia's type system
- Cons
  - Reading rows requires non-sequential memory access

Pros and Cons of Row-Oriented Storage

- Pros
  - Easier to serialize in human-readable format
  - Required by some systems (e.g. Vega)
- Cons
  - Seek time can be prohibitive
  - Must deal with all columns in every interaction

The DataFrame Type

```
type DataFrame
    cols::Vector{Any}
    index::Index
end
```

What Should Julia Do?

- DataFrames.jl is column-oriented
- Row-oriented architecture doesn't seem as appealing
- But we mostly read in data from row-oriented sources

Sub-Problem 3

Should we store data in memory or on disk?

Pros and Cons of Storing Data in Memory

- Pros
  - Very fast
  - No caching required anywhere
- Cons
  - Limited to size of main memory

Pros and Cons of Storing Data on Disk

- Pros
  - Limited only by size of disk store
- Cons
  - Data needs to get into memory at some point
  - Quality of caching becomes a bottleneck

What Should Julia Do?

- DataFrames are entirely in memory
- DataFrames currently use DataArrays for columns
- Could transition to using cached columns

Our Solution to Representing Data

- Database strategy for representing missingness
- Column-oriented database-like structure
- All data is held in-memory

Moving Forward

- Better on-disk data structures
- Better streaming data support
- Better indexing of DataFrames