Ratcl & Rasql

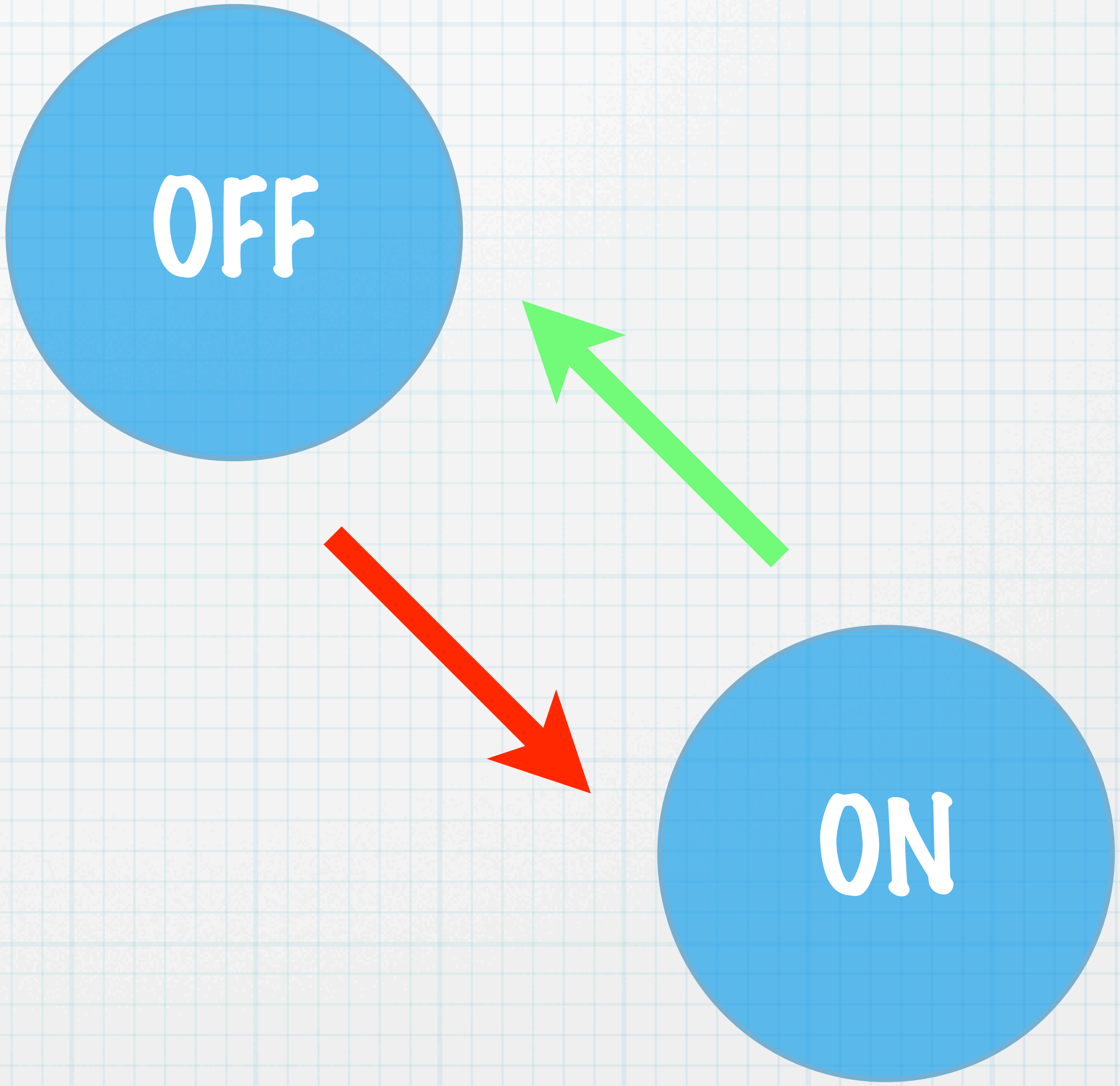
Jean-Claude Wippler

EQUI4 software

The Netherlands

The Data Dilemma

- data on disk
  - permanent
  - passive
- data in memory
  - running apps
  - modifiable



Everything in Tcl

- application startup and exit:

```
array set mydata [read $fd]
puts $fd [array get mydata]
```

- sluggish - doesn't scale
- risky - lose all if write fails
- structure? search? share? speed?

Everything in a DB

- learn a new language & mindset
- which DB, pick ONE and stick with it
- startup - can take a lot more code
- copy, copy, copy data from DB to Tcl
- copy, copy, copy data from Tcl to DB
- design structure first - or be sorry later

Relational Algebra

- best intro I've seen:

  `http://en.wikipedia.org/wiki/Relational_algebra`

- can do everything with 6 primitives:

  select, project, product,
  union, difference, rename

- could RA be what SQL should have been?

Ratcl

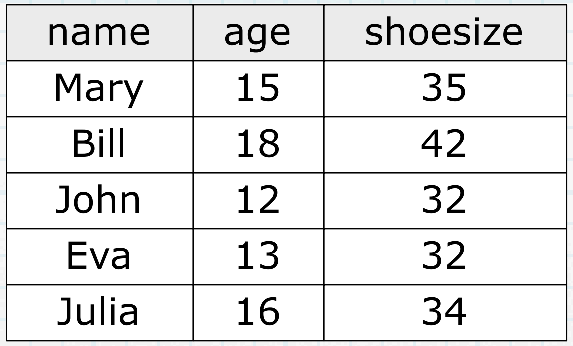
- stay in Tcl, think in Tcl, code in Tcl
- manipulate data, still on disk
  - access managed by Ratcl
- lots of data manipulation operators
  - relational, set-wise, vectors
- transactions - commit/rollback, ACID

Implications

- you control the data, but don't own it
  - learn to work with "views"
- stop writing loops to find & process
  - think relational, set-wise, "wham!"
- large speed & memory-use benefits
- no hostages - can always import/export

Views

- A view:



name	age	shoesize
Mary	15	35
Bill	18	42
John	12	32
Eva	13	32
Julia	16	34

- rectangular “array”, “table”
- named columns uniform type
- rows 0 .. N-1 vertical: efficient

The “wham” mindset

- Relational product of views A and B:

A:

name	age
Mary	15
Bill	18
John	12

B:

shoe	size
left	32
right	38

#A = 3

#B = 2

name	age
Mary	15
Mary	15
Bill	18
Bill	18
John	12
John	12

+

shoe	size
left	32
right	38
left	32
right	38
left	32
right	38

=

A x B:

name	age	shoe	size
Mary	15	left	32
Mary	15	right	38
Bill	18	left	32
Bill	18	right	38
John	12	left	32
John	12	right	38

Views are virtual

- the "product" example uses NO memory
- it doesn't read any data
  - data is read when accessed
  - memory-mapped files, no copying
  - cached by the O/S, same as "paging"
- combined operations are also virtual

Data exchange

- Tcl to view - “real data”

```
set r [vdef name age shoesize {Paul 15 32}]
set v [vdef name age [array get mydata]]
```

- view to Tcl - “real processing”

```
puts [view $v sort | get]
view $v each c { puts $c(name) }
array set a [view $v where {age >= 16} | get]
```

Meta-views

- Every view has a meta-view ...
- ... which describes its structure
- View: Meta-view:

name	age	shoesize
Mary	15	35
Bill	18	42
John	12	32
Eva	13	32
Julia	16	34

→

type	name	subv
S	name	-
I	age	-
I	shoesize	-

- #columns in view = #rows in meta-view

Repeating data

- let's add a field to list their friends:

name	age	shoesize	friends
Mary	15	35	Eva, Bill
Bill	18	42	Mary
John	12	32	Mary, Eva, Julia
Eva	13	32	John
Julia	16	34	Mary

- how do you represent this?

Repeat the rows?

- store each friend in a row copy:

name	age	shoesize	friend
Mary	15	35	Eva
Mary	15	35	Bill
Bill	18	42	Mary
John	12	32	Mary
John	12	32	Eva
John	12	32	Julia
Eva	13	32	John
Julia	16	34	Mary

- can (will!) become inconsistent - BAD

Relational: normalize

- use two relations, link by common key:
  - Master: Detail:

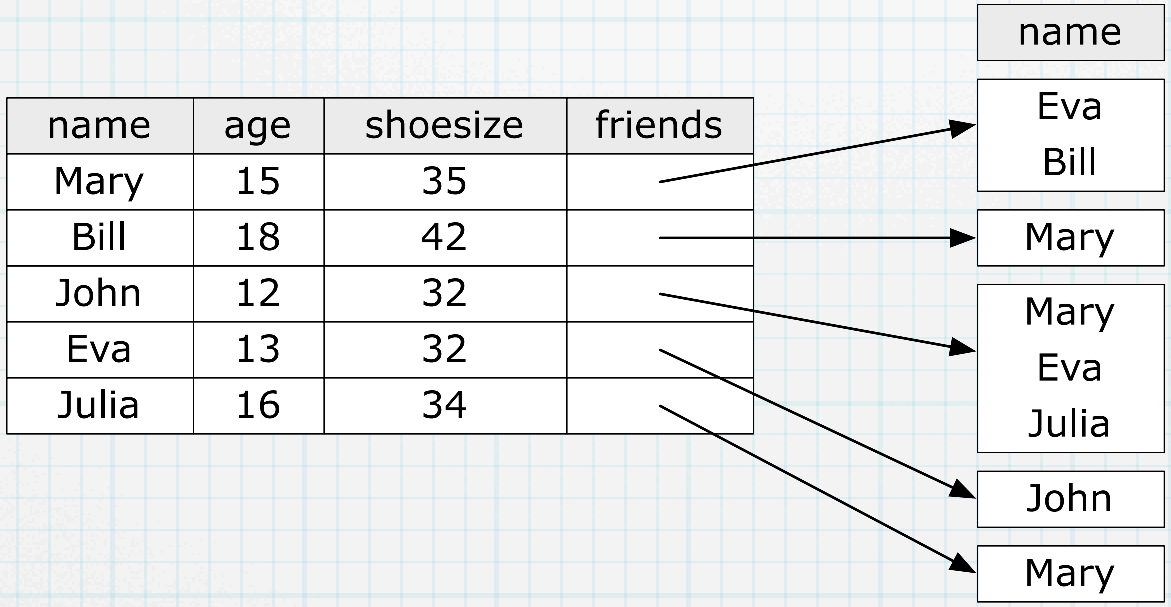
name	age	shoesize
Mary	15	35
Bill	18	42
John	12	32
Eva	13	32
Julia	16	34

name	friend
Mary	Eva
Mary	Bill
Bill	Mary
John	Mary
John	Eva
John	Julia
Eva	John
Julia	Mary

  - simple & consistent
  - keys may require a lot of space

Sub-views

- embed 1:N in a hierarchical way:



name	age	shoesize	friends
Mary	15	35	
Bill	18	42	
John	12	32	
Eva	13	32	
Julia	16	34	

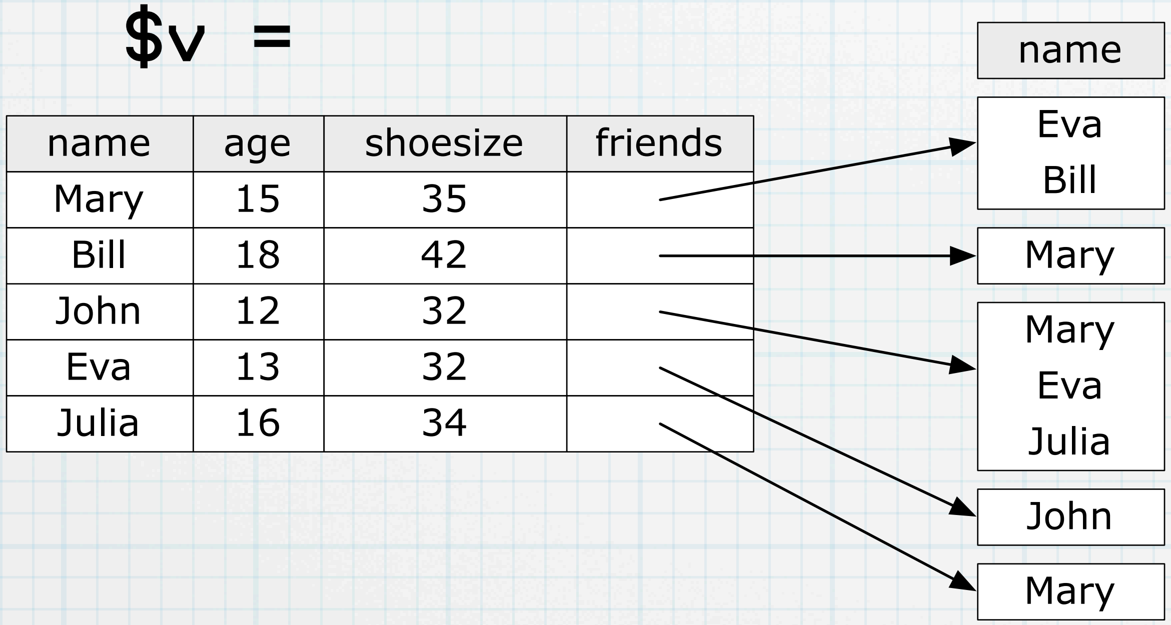
name
Eva Bill
Mary
Mary Eva Julia
John
Mary

- still clean & tidy
- more efficient in time & space

Ratcl can "flatten" ...

- subviews and expanded are equivalent:

```
$v =
```



name	age	shoesize	friends
Mary	15	35	
Bill	18	42	
John	12	32	
Eva	13	32	
Julia	16	34	

name
Eva Bill
Mary
Mary Eva Julia
John
Mary

- `view $v flatten friends =`

name	age	shoesize	friend
Mary	15	35	Eva
Mary	15	35	Bill
Bill	18	42	Mary
John	12	32	Mary
John	12	32	Eva
John	12	32	Julia
Eva	13	32	John
Julia	16	34	Mary

… and go back: "group"

- grouping is inverse of flattening:

$v =

name	age	shoesize	friend
Mary	15	35	Eva
Mary	15	35	Bill
Bill	18	42	Mary
John	12	32	Mary
John	12	32	Eva
John	12	32	Julia
Eva	13	32	John
Julia	16	34	Mary

- `view $v group name age shoesize =`

name	age	shoesize	friends
Mary	15	35	
Bill	18	42	
John	12	32	
Eva	13	32	
Julia	16	34	

name
Eva Bill
Mary
Mary Eva Julia
John
Mary

Relational Join

- the workhorse for normalized data:

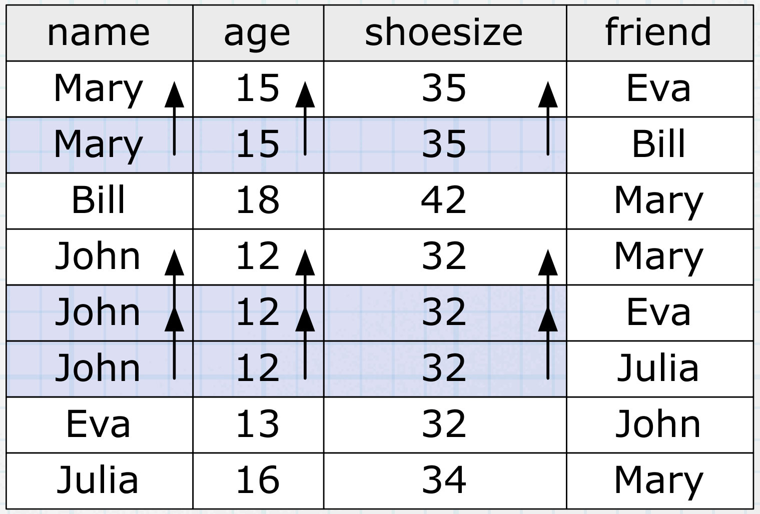
$v =

name	age	shoesize
Mary	15	35
Bill	18	42
John	12	32
Eva	13	32
Julia	16	34

$w =

name	friend
Mary	Eva
Mary	Bill
Bill	Mary
John	Mary
John	Eva
John	Julia
Eva	John
Julia	Mary

- "classical" join result =



name	age	shoesize	friend
Mary	15	35	Eva
Mary	15	35	Bill
Bill	18	42	Mary
John	12	32	Mary
John	12	32	Eva
John	12	32	Julia
Eva	13	32	John
Julia	16	34	Mary

- physical = "v & w" versus logical = joined

Joins

- a join is “N lookups in parallel”
- joins produce subviews in Ratcl
- no NULLs, yet equivalent
- rely on flattening & grouping
- think in very high-level: data shapes!

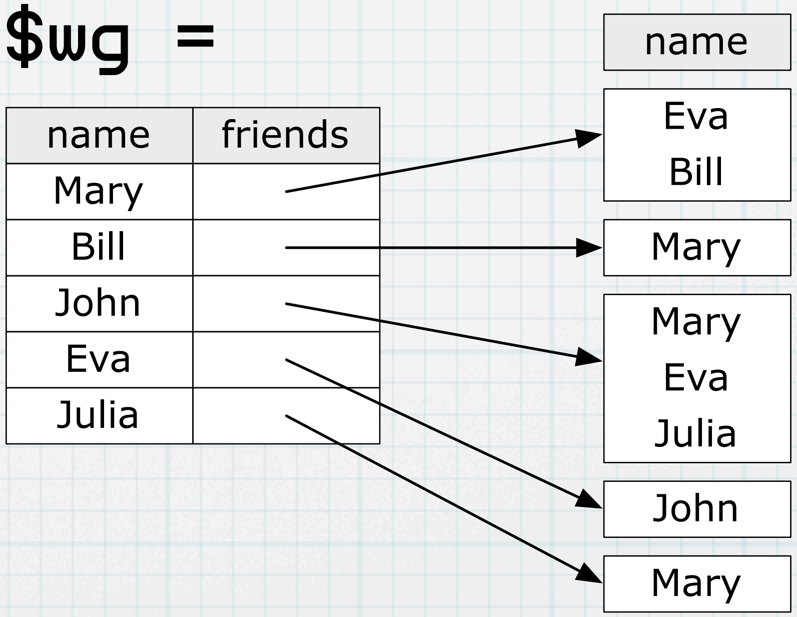
Ratcl's Join

- first, group repeated field to subviews ...

$w =

name	friend
Mary	Eva
Mary	Bill
Bill	Mary
John	Mary
John	Eva
John	Julia
Eva	John
Julia	Mary

$wg =



name	friends
Mary	Eva Bill
Bill	Mary
John	Mary Eva Julia
Eva	John
Julia	Mary

- `view $w group name =`

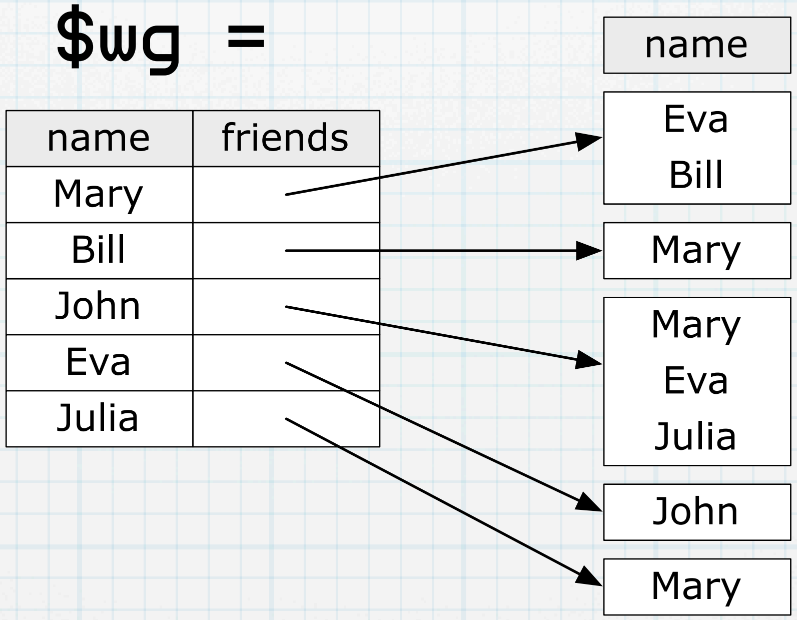
Ratcl's Join - part 2

- ... then connect corresponding rows

```
$v =
```

name	age	shoesize
Mary	15	35
Bill	18	42
John	12	32
Eva	13	32
Julia	16	34

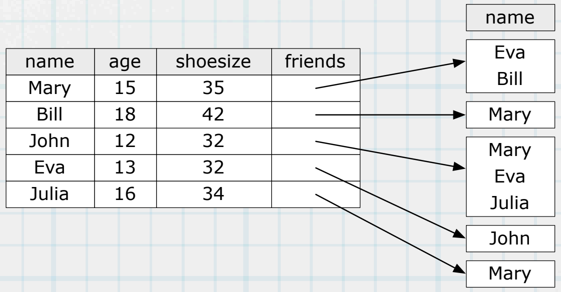
```
$wg =
```



name

name	friends
Mary	Eva Bill
Bill	Mary
John	Mary Eva Julia
Eva	John
Julia	Mary

- `view $v join $wg = (same result)`



name

name	age	shoesize	friends
Mary	15	35	Eva Bill
Bill	18	42	Mary
John	12	32	Mary Eva Julia
Eva	13	32	John
Julia	16	34	Mary

It’s all Relational

- SQL

```
SELECT * FROM data
   WHERE name = ‘John’
   ORDER BY age
```

- Ratcl

```
view $data where {name = ‘John’} | sort age
```

- or maybe

```
view $data where {name == “John”} | sort age
```

SQL & Rasql

- (S)tructured - tables & joins
- (Q)uery - "what", not "how"
- (L)anguage - standard notation

- Rasql translates SQL to Ratcl view ops
  - thin layer to create an "access plan"

SQL?

- 1 standard? - N dialects!
- optimization, trial and error
- NULL, 3-valued logic
- half a programming language

- Rasql doesn't try to be "big" SQL system

Inside Ratcl

- guided by simplicity and performance
  - lessons from Forth, APL, and Metakit
- "obsessively vector-oriented" design
  - tiny special-purpose virtual machine
  - portable implementation language

Minimalism

- Forth & APL show that “less is more”
- built on a very uniform data structure
  - tiny and fast mark/sweep GC
- VM is < 1000 lines of C code
  - 1000 more for vector ops
- it’s all under the hood - Tcl is the API

How small?

- VM is a 30 Kb C extension
- Tcl wrapper is another 40 Kb
- as starkit - which uses compression

  `http://www.equi4.com/pub/vq/ratcl.kit`

  - 100 Kb for complete system
  - includes binaries for 5 platforms

Speed: think again

- **SQL:** `SELECT * FROM data WHERE name = ‘John’`
  - “*” often reads too much
- **Ratcl:** `set v [view $data where {name = ‘John’}]`
  - USE determines I/O: later & lazily
  - column-wise “inverted” storage
  - like having indexes on everything

How Rasql works

-
  ```
  select name from students
    where     age > 15
    group by  shoesize
    having    count(shoesize) > 1
  ```

- 1. map to groups
  2. collect counts
  3. omit some counts
  4. flatten result
  5. omit some ages
  6. done!

Why it's fast

1. load column of shoe sizes:	1 read
2. locate duplicates via hash:	O(N)
3. load column of ages:	1 read
4. select specific age range:	O(N)
5. logical AND, bitmaps:	fast
6. Done!	

How fast?

- Join 161,127 x 47,079 on 1 int:

```
Ratcl: 0.08 s, Metakit: 3.16 s
```

- Find unique IP's in 1,077,106 entries

```
Ratcl 0.15 s, Tcl 3.7 s (lsort -unique)
 (~ 4 Mb)      (~ 28 Mb)
```

- Find 3 matches in 1,077,106 values

```
Ratcl 1.66 s, Metakit 2.18 s, SQLite 3.85 s
Ratcl 28 µS, SQLite 316 µS - indexed
               (create 37s, drop 3.2s)
```

Current status

- Ratcl 0.92
  - it works, many operators
  - API has not been frozen yet
  - it's not very robust or fast right now
  - maps MK datafiles, and writes dumps
- Rasql - only an older preview

Progress

- on the web as "Vlerq" research project

  `http://www.vlerq.org/`

- good software is like good wine
  - consumed quickly just gets you drunk
  - take your time to enjoy its richness
- most of my 2005 time goes into Vlerq