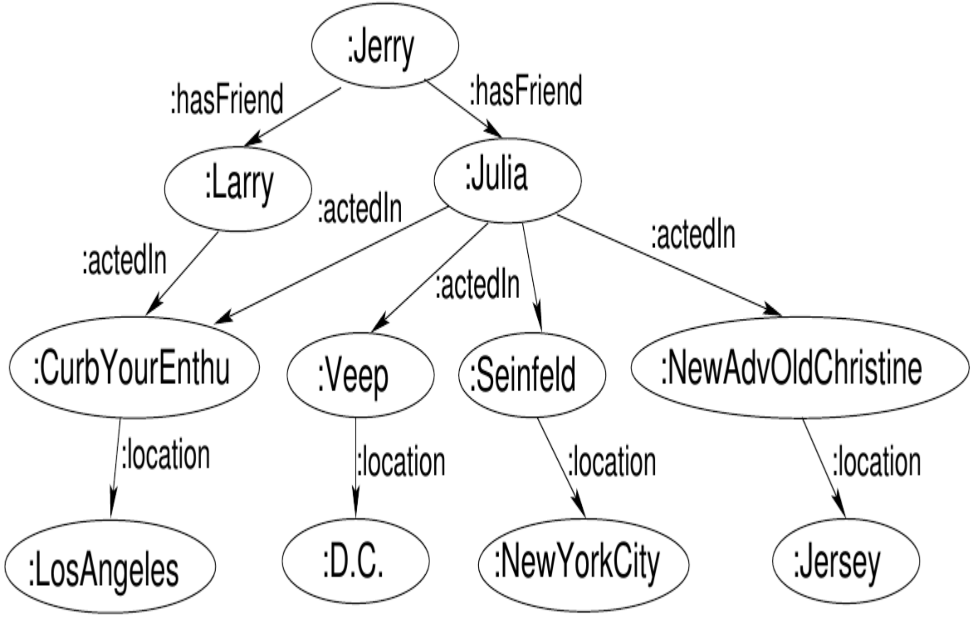
CS698F Advanced Data Management

Instructor: Medha Atre

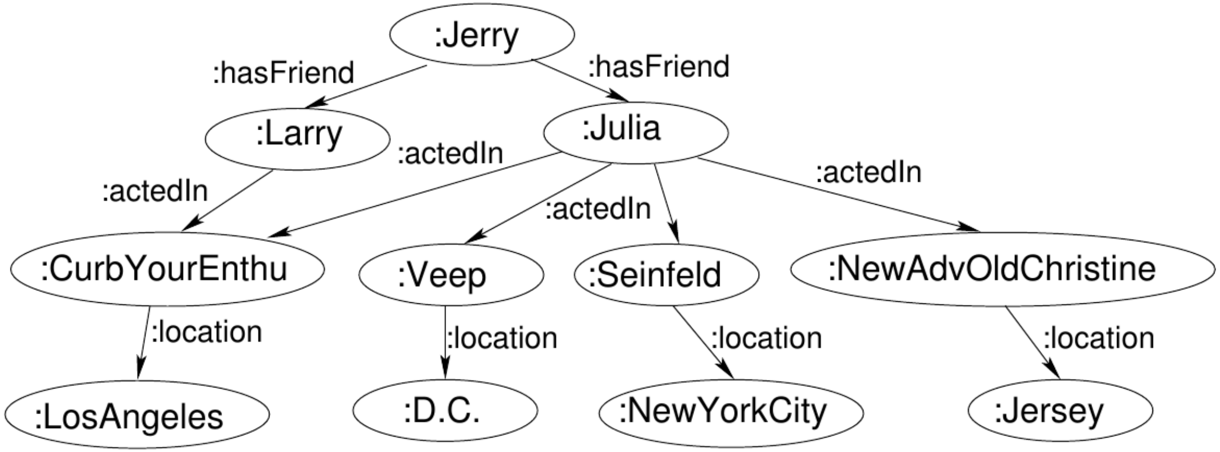
Graph as a table

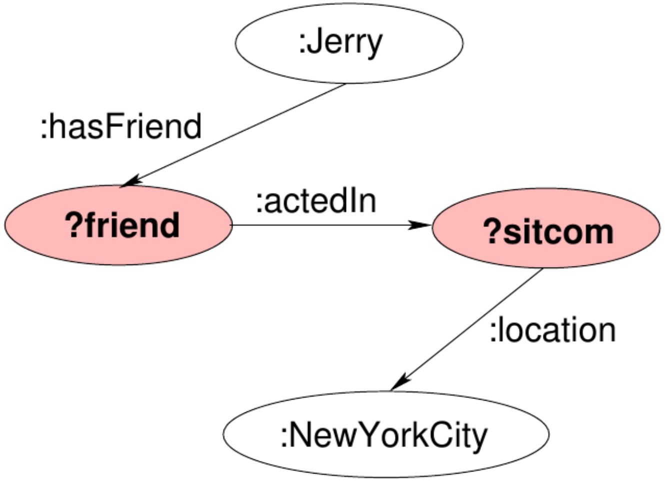


RDF table

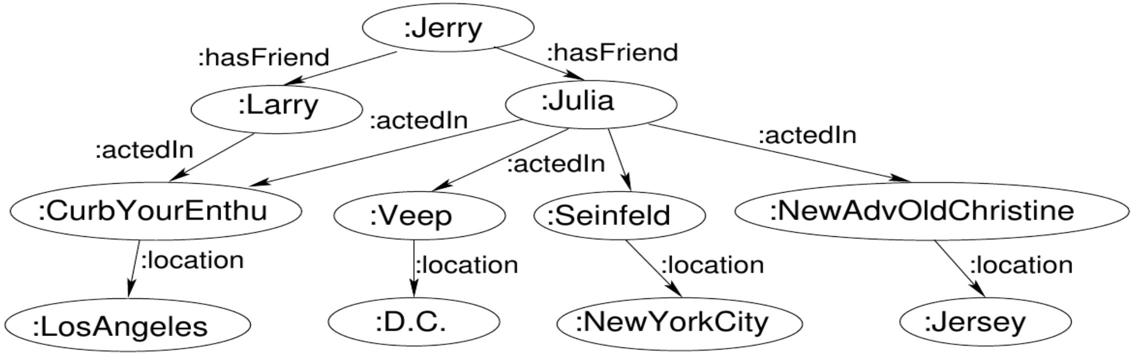
S	P	O
:Jerry	:hasFriend	:Larry
:Jerry	:hasFriend	:Julia
:Larry	:actedIn	:CurbYourEnthu
:Julia	:actedIn	:Seinfeld
:Julia	:actedIn	:Veep
:Julia	:actedIn	:NewAdvOldChristine
:Julia	:actedIn	:CurbYourEnthu
:Seinfeld	:location	:NewYorkCity
:Veep	:location	:D.C.
:CurbYourEnthu	:location	:LosAngeles
:NewAdvOldChristin	:location	:Jersey

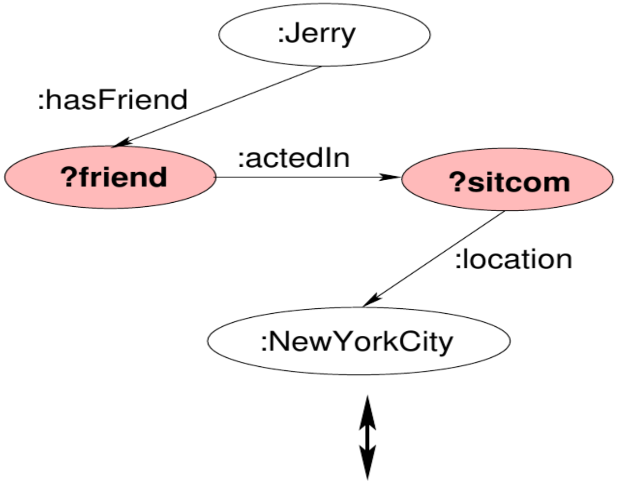
A graph pattern query





A graph pattern query

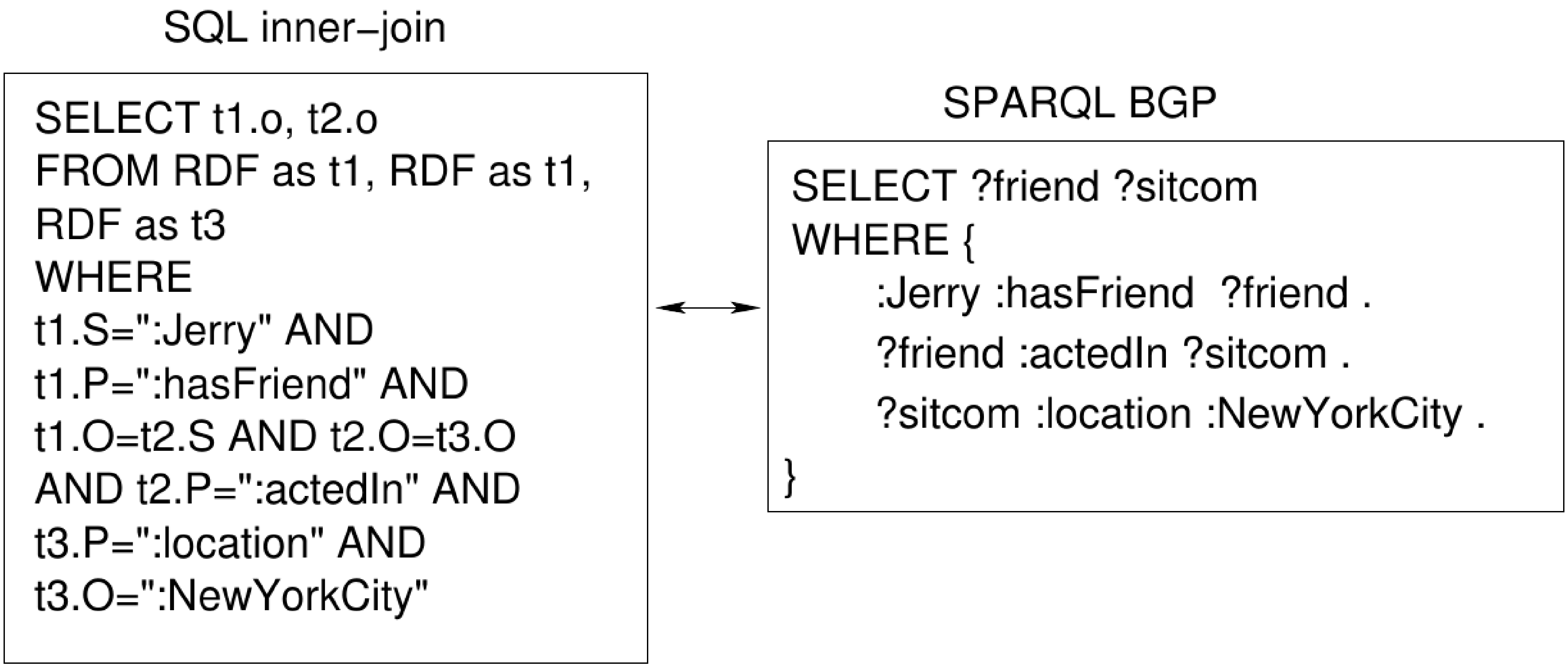




SPARQL BGP

```
SELECT ?friend ?sitcom
WHERE {
      :Jerry :hasFriend  ?friend .
      ?friend :actedIn ?sitcom .
      ?sitcom :location :NewYorkCity .
}
```

Pattern query as a self-join



SQL inner–join

```
SELECT t1.o, t2.o
FROM RDF as t1, RDF as t1,
RDF as t3
WHERE
t1.S=":Jerry" AND
t1.P=":hasFriend" AND
t1.O=t2.S AND t2.O=t3.O
AND t2.P=":actedIn" AND
t3.P=":location" AND
t3.O=":NewYorkCity"
```

SPARQL BGP

```
SELECT ?friend ?sitcom
WHERE {
        :Jerry :hasFriend  ?friend .
        ?friend :actedIn ?sitcom .
        ?sitcom :location :NewYorkCity .
}
```

Graph Indexing

- Assuming graph stored as a 3-column table
  - All possible permutations of 3-columns, 6 indexes
  - SPO, SOP, PSO, POS, OPS, OSP – with entire SPO, SOP etc as the search key.
  - Creates 6 copies of the graph
- Too much space wastage?
  - Data compression methods!

Graph Indexing

- Graph node/edge labels variable length strings
  - Not a good fit for *search-keys*
  - If using B+ trees, *order = page-size / (search-key-size + pointer-size)*
  - Map node/edge labels to fixed length IDs *label* → *ID*
  - Maintain a reverse mapping of *ID* → *label*

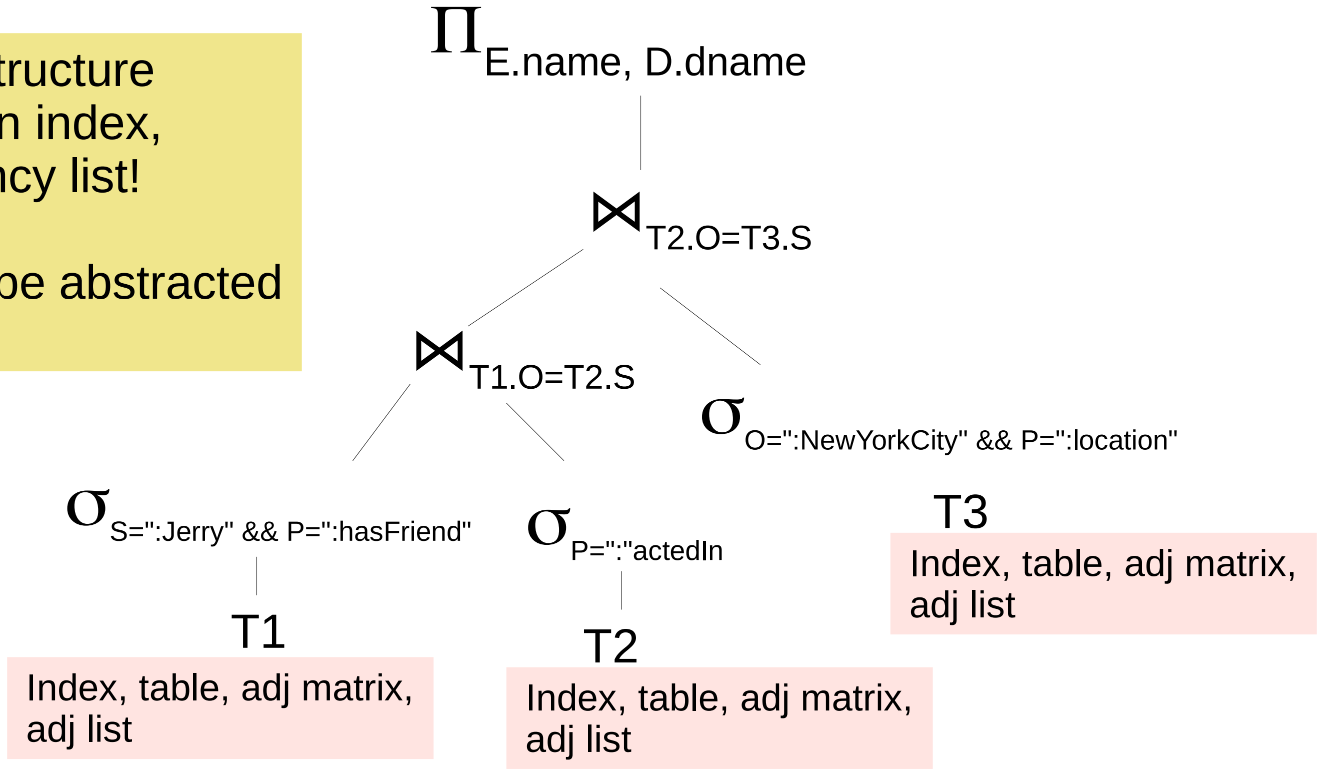
Graph Indexing

- Preprocessing steps
  - Map each unique label to fixed length ID
  - Represent all node and edge labels as Ids.
  - Create *label* → *ID* mapping – dictionary
  - Maintain a reverse mapping of *ID* → *label*
- Query processing steps
  - Convert pattern query to a join query
  - User join query optimization techniques
  - Run the query, get the results in *ID form*.
  - Use reverse mapping dictionary to map result IDs to labels.

Abstraction of join queries

The access data structure can be anything, an index, a table, an adjacency list!

Join methods can be abstracted out accordingly.



$\Pi_{\text{E.name, D.dname}}$

$\bowtie_{\text{T2.O=T3.S}}$

$\bowtie_{\text{T1.O=T2.S}}$

$\sigma_{\text{O=":NewYorkCity" \&\& P=":location"}}$

$\sigma_{\text{S=":Jerry" \&\& P=":hasFriend"}}$

$\sigma_{\text{P=":"actedIn}}$

T1

Index, table, adj matrix, adj list

T2

Index, table, adj matrix, adj list

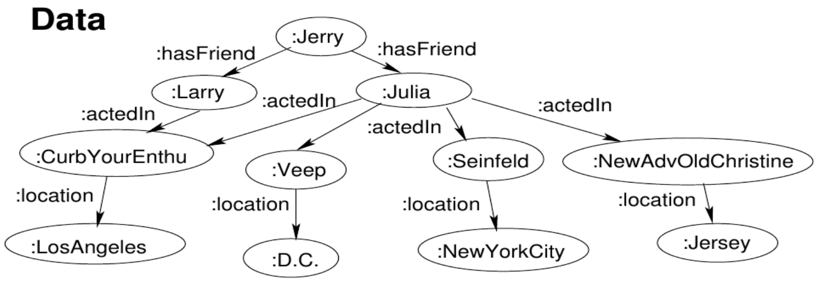
T3

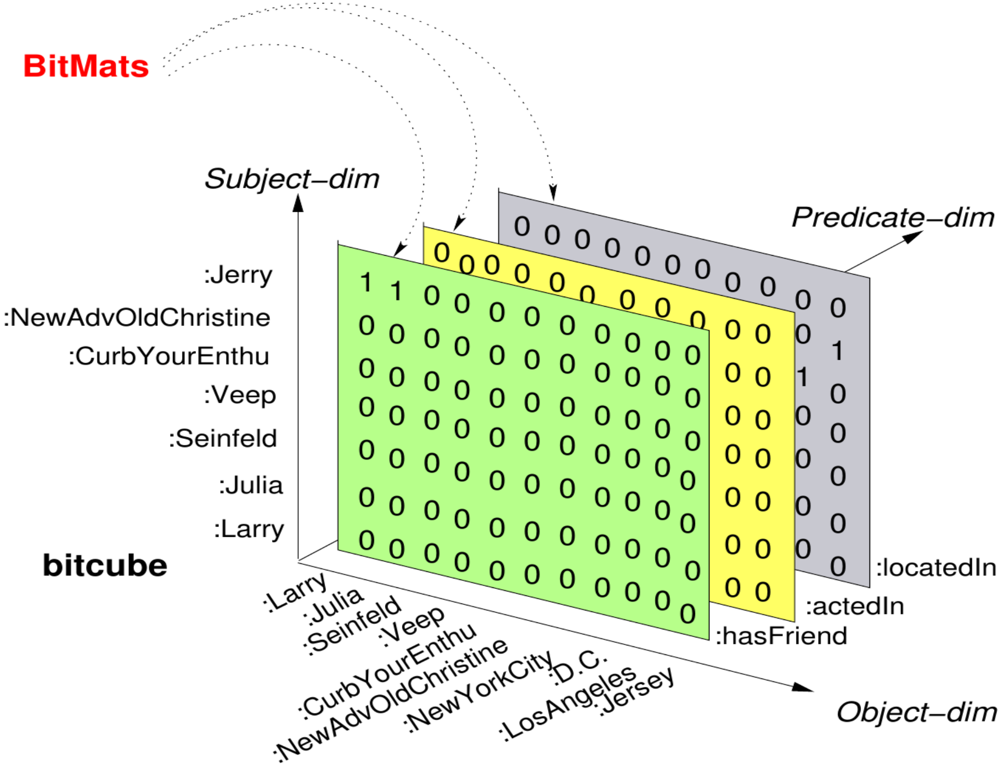
Index, table, adj matrix, adj list

Alternate indexing methods

- Let graphs be represented as adjacency matrix
- What to do with edge-labels?
  - One adjacency matrix for each unique edge label
  - Similar to splitting the original graph into multiple subgraphs
  - Each subgraph has only one type of edge-label
  - Build adjacency matrix for each edge-label

Indexing adjacency matrices





Joins with adjacency matrix

- Each matrix is like a 2-column table
  - So serialize the matrix as a table and join – no benefit of 2D matrix!
- Create a *row-vector* and *column-vector*
  - Column-vect = Boolean OR of all the rows
  - Row-vect = Boolean OR of all the columns
- Joining two matrices is equivalent to
  - *Intersection* of row/column vectors of two matrices
  - Removing matrix entries that have the values eliminated in the intersection!
  - This is called a *semi-join*!

Joins with adjacency matrices

	:Larry	:Julia									
:Jerry	1	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	0	0	0	0	0	0	0	0	0
:Larry	0	0	0	0	0	0	0	0	0	0	0

:hasFriend

We want
to join
columns
of left with
rows of right

			:Seinfeld	:Veep	:CurbUourEnthu						
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	1	1	1	1	0	0	0	0	0
:Larry	0	0	0	0	1	0	0	0	0	0	0

:actedIn

Joins with adjacency matrices

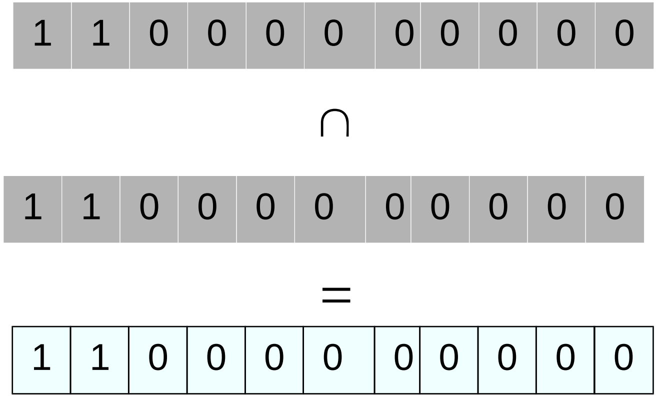
- Column values on LHS join with row values on RHS
  - *col-vect (mat1) AND row-vect(mat2) = partial-join-res*
  - For each 0 bit in *partial-join-res*, remove all matrix cells that contain 1 in the respective position

1	1	0	0	0	0	0	0	0	0	0

0
0
0
0
0
0
0
0
0
1
1

Joins with adjacency matrices

- Column values on LHS join with row values on RHS
  - *col-vect (mat1) AND row-vect(mat2) = partial-join-res*
  - For each 0 bit in *partial-join-res*, remove all matrix cells that contain 1 in the respective position



1	1	0	0	0	0	0	0	0	0	0

∩

1	1	0	0	0	0	0	0	0	0	0

Row-vect rotated by 90 deg

=

1	1	0	0	0	0	0	0	0	0	0

Nothing changes here, hence no matrix cells removed!

Joins with adjacency matrices

:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	1	0	0
.	0	0	0	0	0	0	0	1	0	0	0
.	0	0	0	0	0	0	1	0	0	0	0
:Julia	0	0	0	0	0	0	0	0	0	0	0
:Larry	0	0	0	0	0	0	0	0	0	0	0

:locatedIn

			:Seinfeld	:Veep	:CurbUourEnthu	:NewAdvOldChri					
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	1	1	1	1	0	0	0	0	0
:Larry	0	0	0	0	1	0	0	0	0	0	0

:actedIn

							:NewYorkCity				
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	1	0	0
.	0	0	0	0	0	0	0	1	0	0	0
:Seinfeld	0	0	0	0	0	0	1	0	0	0	0
:Julia	0	0	0	0	0	0	0	0	0	0	0
:Larry	0	0	0	0	0	0	0	0	0	0	0

:locatedIn

			:Seinfeld	:Veep	:CurbUourEnthu						
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	1	1	1	1	0	0	0	0	0
:Larry	0	0	0	0	1	0	0	0	0	0	0

:actedIn

0	0	1	0	0	0	0	0	0	0	0

∩

0	0	1	1	1	1	0	0	0	0	0

Row-vect rotated
by 90 deg

=

0	0	1	0	0	0	0	0	0	0	0

Bits corresponding to :Veep, :CurbYourEnthu, and :NewAdvOldChristine removed, hence remove respective matrix entries from both sides