Lecture 16: Relational Algebra

Monday, May 10, 2010

Outline

Relational Algebra:

- Chapters 5.1 and 5.2

The WHAT and the HOW

- In SQL we write WHAT we want to get form the data

- The database system needs to figure out HOW to get the data we want

- The passage from WHAT to HOW goes through the Relational Algebra

Data Independence

SQL = WHAT

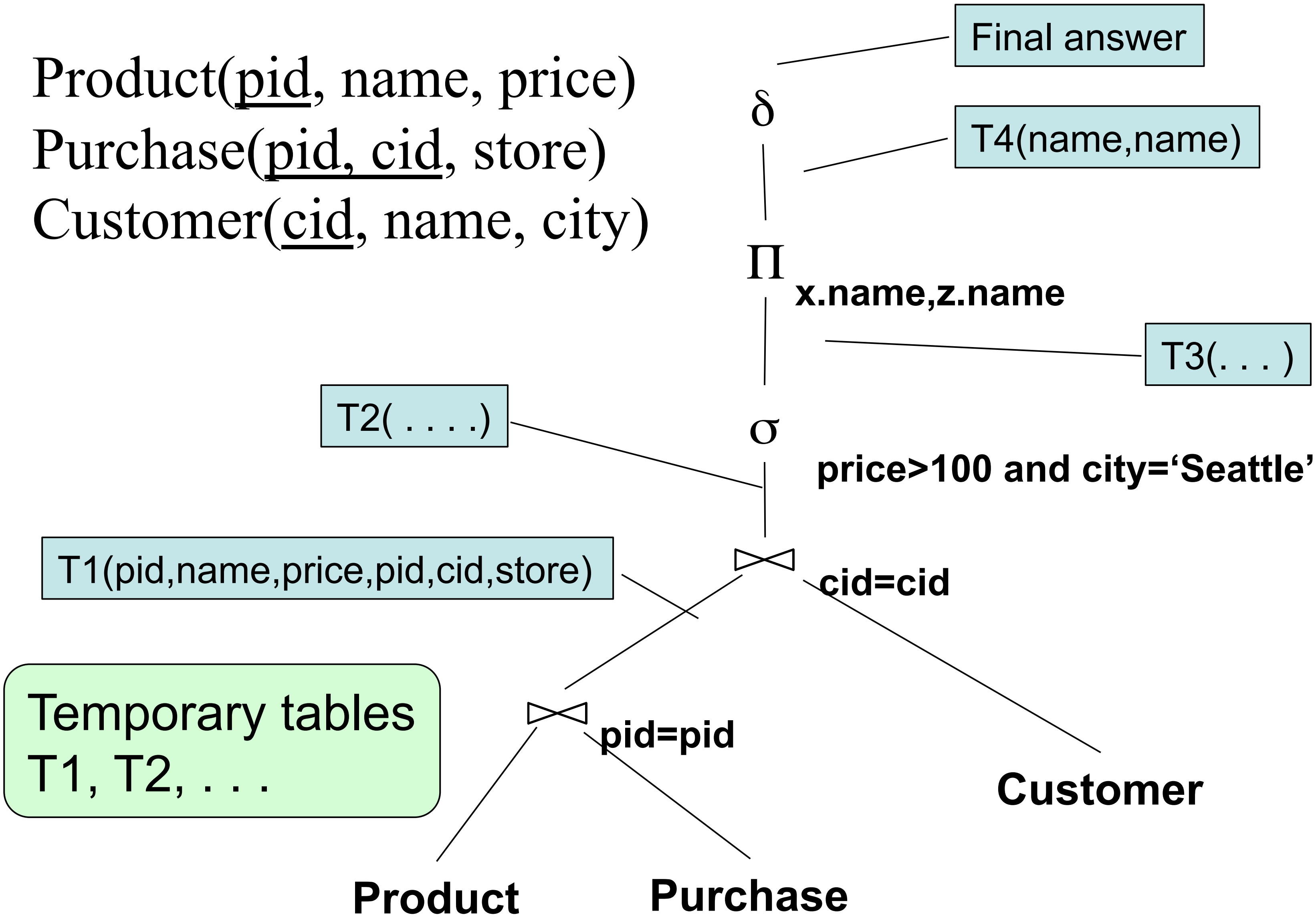
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```

It's clear WHAT we want, unclear HOW to get it

Relational Algebra = HOW

Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)



Relational Algebra = HOW

The order is now clearly specified:

Iterate over PRODUCT…
…join with PURCHASE…
…join with CUSTOMER…
…select tuples with Price>100 and City=‘Seattle’…
…eliminate duplicates…
…and that’s the final answer !

Sets v.s. Bags

- Sets: {a,b,c}, {a,d,e,f}, { }, . . .
- Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

Relational Algebra has two semantics:

- Set semantics
- Bag semantics

Extended Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection σ
- Projection Π
- Join $\bowtie$
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

Relational Algebra (1/3)

The Basic Five operators:

- Union: $\cup$
- Difference: -
- Selection: σ
- Projection: Π
- Join: $\bowtie$

Relational Algebra (2/3)

Derived or auxiliary operators:

- Renaming: ρ
- Intersection, complement
- Variations of joins
  - natural, equi-join, theta join, semi-join, cartesian product

Relational Algebra (3/3)

Extensions for bags:

- Duplicate elimination: δ
- Group by: γ
- Sorting: τ

Union and Difference

$R1 \cup R2$

$R1 - R2$

What do they mean over bags ?

What about Intersection ?

- Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

- Derived using join (will explain later)

$$R1 \cap R2 = R1 \bowtie R2$$

Selection

- Returns all tuples which satisfy a condition

$$\sigma_c(R)$$

- Examples
  - $\sigma_{\text{Salary > 40000}}$ (Employee)
  - $\sigma_{\text{name = “Smith”}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Employee

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

Projection

- Eliminates columns

$$\Pi_{A1,\ldots,An}(R)$$

- Example: project social-security number and names:
  - $\Pi_{\text{SSN, Name}}$ (Employee)
  - Answer(SSN, Name)

Semantics differs over set or over bags

Employee

SSN	Name	Salary
1234545	John	200000
5423341	John	600000
4352342	John	200000

$\Pi_{Name,Salary}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient to implement ?

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$R1 \times R2$$

- Very rare in practice; mainly used to express joins

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee ✕ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Changes the schema, not the instance

$$\rho_{B1,\ldots,Bn}(R)$$

- Example:
  - $\rho_{N,\,S}$(Employee) → Answer(N, S)

Natural Join

$$R1 \bowtie R2$$

- Meaning: $R1 \bowtie R2 = \Pi_A(\sigma(R1 \times R2))$
- Where:
  - The selection σ checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

Natural Join

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

$\mathbf{R} \bowtie \mathbf{S} =$
$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join

- Given the schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?

- Given R(A, B, C), S(D, E), what is R ⋈ S ?

- Given R(A, B), S(A, B), what is R ⋈ S ?

Theta Join

- A join that involves a predicate

$$R1 \bowtie_{\theta} R2 \;=\; \sigma_{\theta}(R1 \times R2)$$

- Here θ can be any condition

Eq-join

- A theta join where θ is an equality

$$R1 \bowtie_{A=B} R2 \;=\; \sigma_{A=B}\,(R1 \times R2)$$

- This is by far the most used variant of join in practice

So Which Join Is It ?

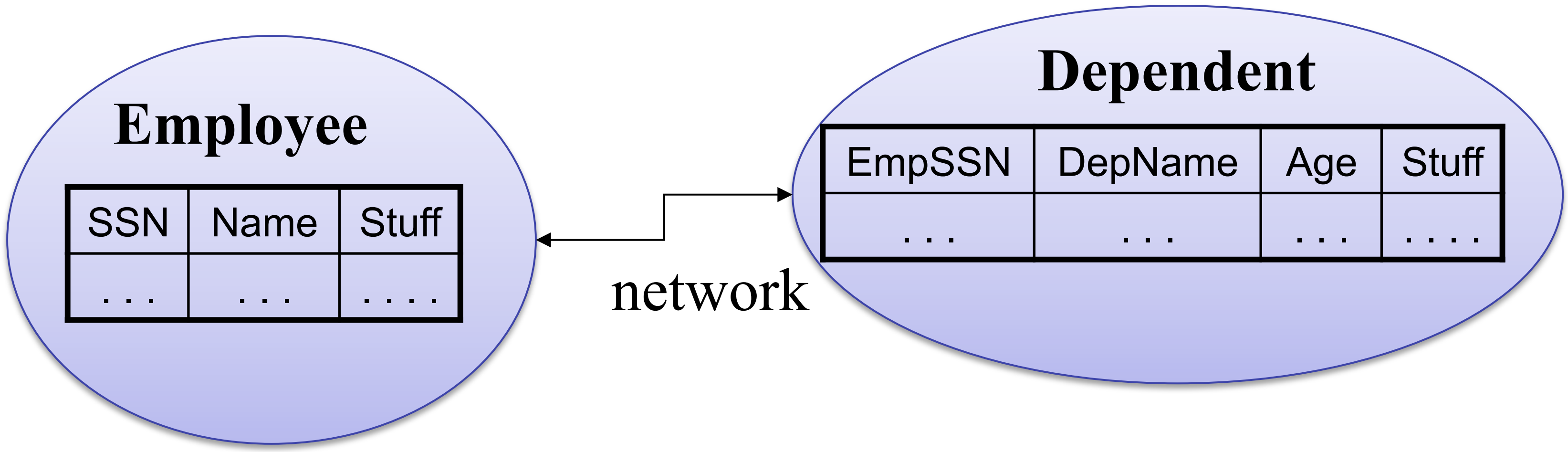
- When we write R ⋈ S we usually mean an eq-join, but we often omit the equality predicate when it is clear from the context

Semijoin

$$R \ltimes_C S = \Pi_{A1,\ldots,An} (R \bowtie_C S)$$

- Where $A_1, \ldots, A_n$ are the attributes in R

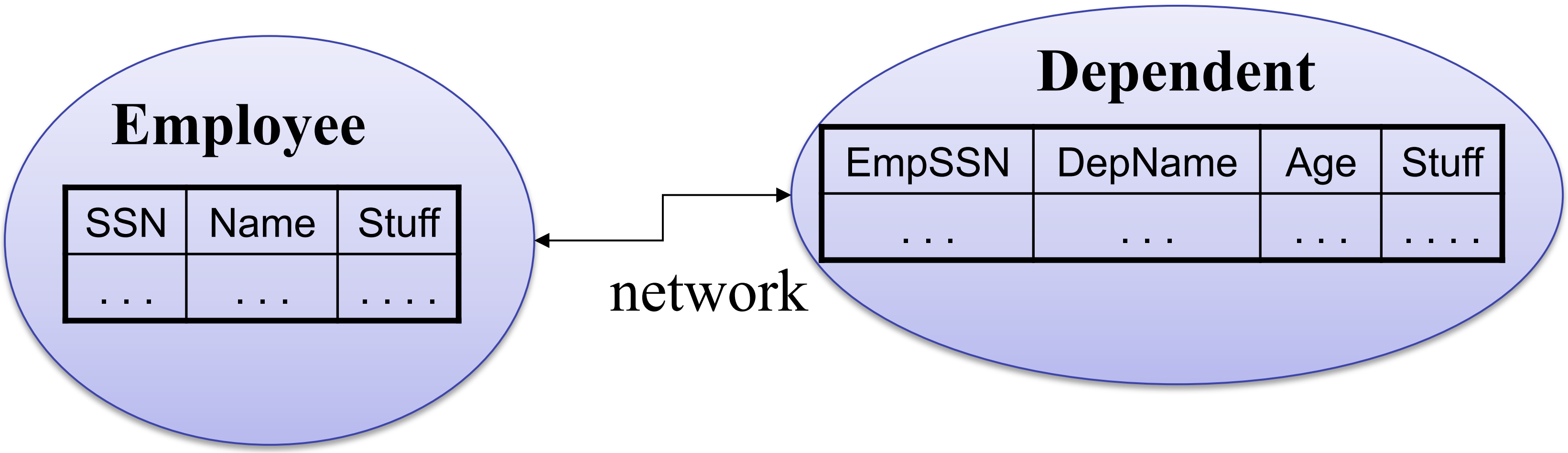
Semijoins in Distributed Databases



$$\text{Employee} \bowtie_{SSN=EmpSSN} (\sigma_{age>71} (\text{Dependent}))$$

Task: compute the query with minimum amount of data transfer

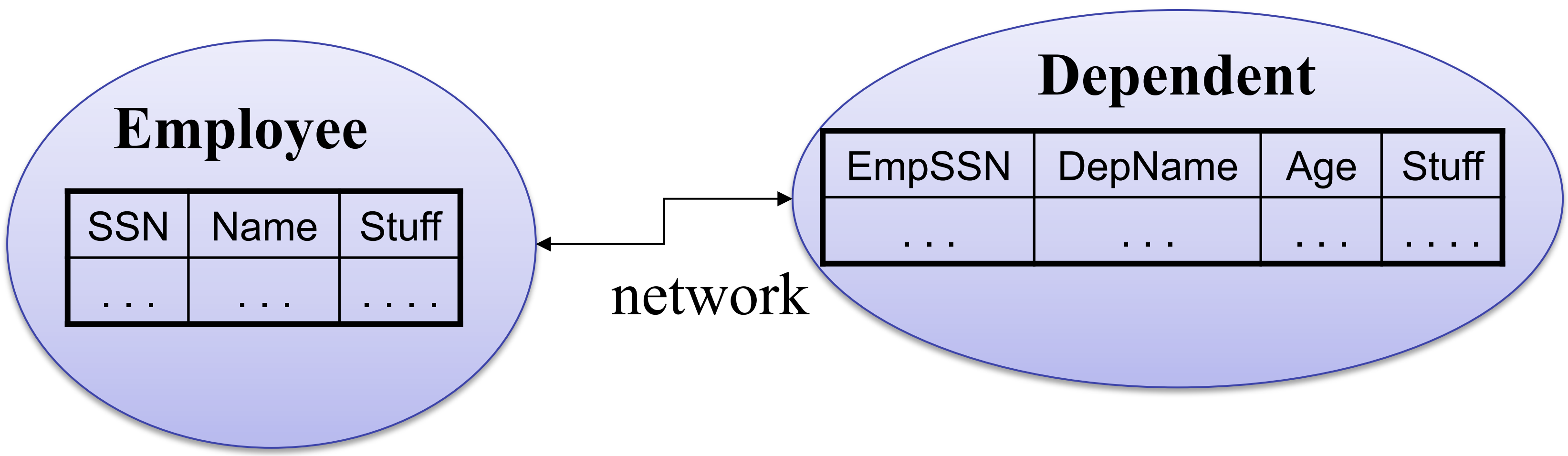
Semijoins in Distributed Databases



$$\text{Employee} \bowtie_{SSN=EmpSSN} (\sigma_{age>71} (\text{Dependent}))$$

$$T(SSN) = \Pi_{SSN}\, \sigma_{age>71} (\text{Dependents})$$

Semijoins in Distributed Databases

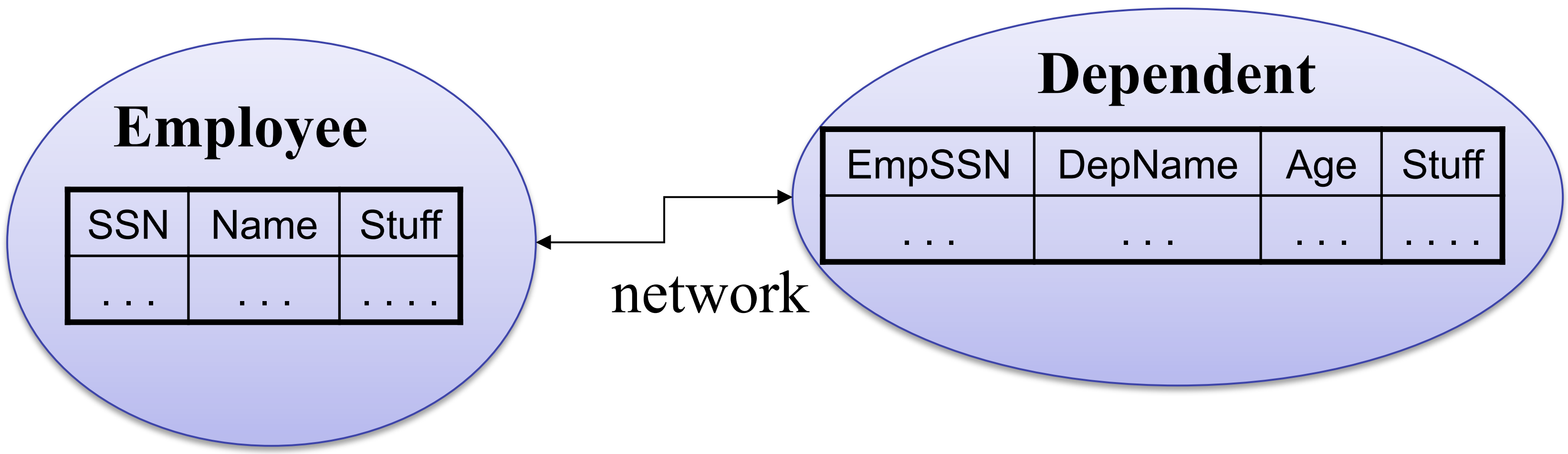


$$\text{Employee} \bowtie_{\text{SSN=EmpSSN}} (\sigma_{\text{age>71}} (\text{Dependent}))$$

$$T(\text{SSN}) = \Pi_{\text{SSN}}\, \sigma_{\text{age>71}} (\text{Dependents})$$

$$R = \text{Employee} \bowtie_{\text{SSN=EmpSSN}} T$$
$$= \text{Employee} \ltimes_{\text{SSN=EmpSSN}} (\sigma_{\text{age>71}} (\text{Dependents}))$$

Semijoins in Distributed Databases



$$\text{Employee} \bowtie_{SSN=EmpSSN} (\sigma_{age>71} (\text{Dependent}))$$

$$T(SSN) = \Pi_{SSN}\, \sigma_{age>71} (\text{Dependents})$$

$$R = \text{Employee} \ltimes_{SSN=EmpSSN} T$$

$$\text{Answer} = R \bowtie_{SSN=EmpSSN} \text{Dependents}$$

Joins R US

- The join operation in all its variants (eq-join, natural join, semi-join, outer-join) is at the *heart* of relational database systems
- WHY ?

Operators on Bags

- Duplicate elimination δ

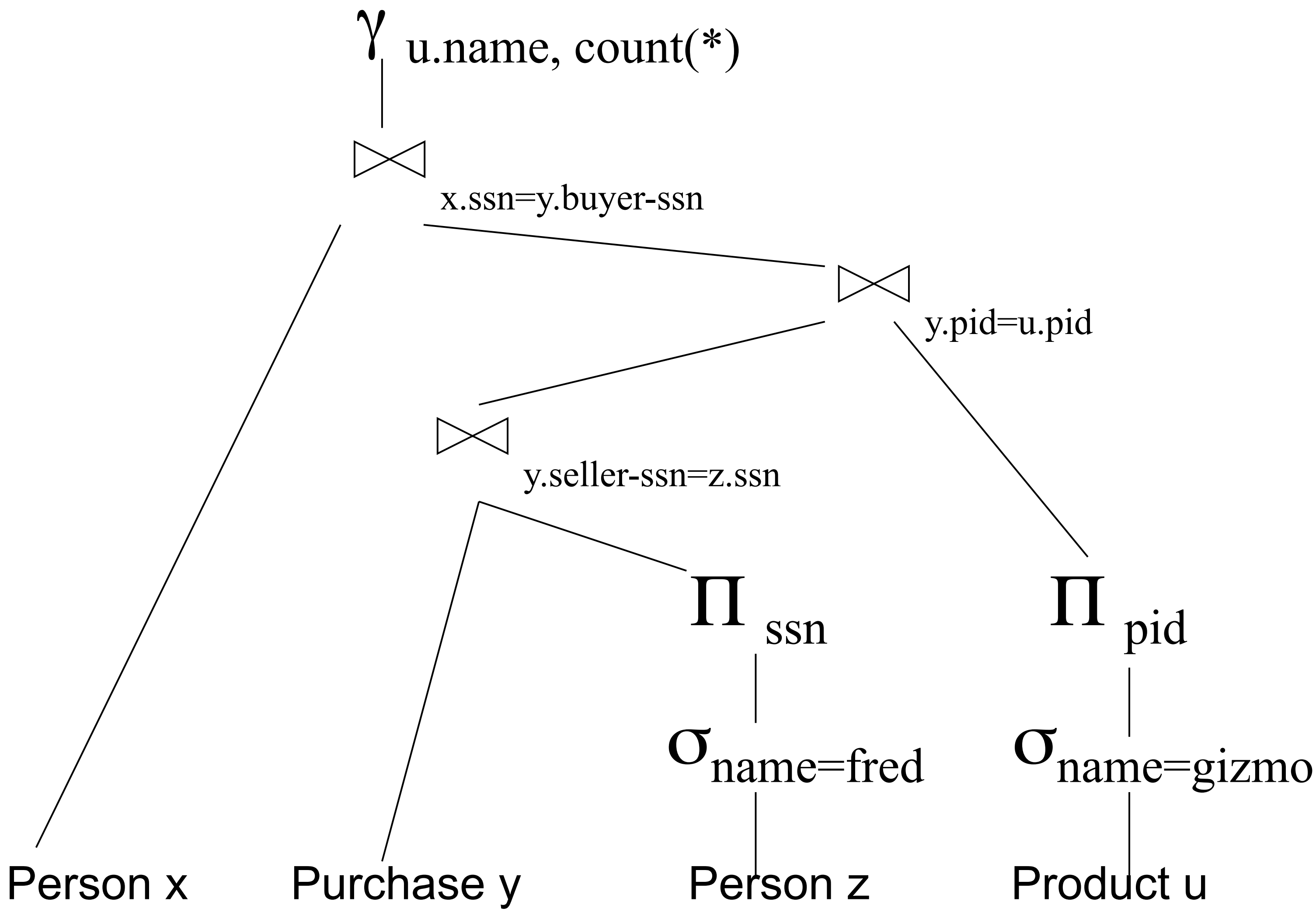
  $\delta(R)$ = select distinct * from R

- Grouping γ

  $\gamma_{A,sum(B)}$ = select A,sum(B) from R group by A

- Sorting τ

Complex RA Expressions



RA = Dataflow Program

- Several operations, plus strictly specified order
- In RDBMS the dataflow graph is always a tree
- Novel applications (s.a. PIG), dataflow graph may be a DAG

Limitations of RA

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write Java program
- Remember *the Bacon number* ? Needs TC too !