Introduction to Database Systems CSE 444

Lecture 17: Relational Algebra

Outline

- Motivation and sets vs. bags
- Relational Algebra
- Translation from SQL to the Relational Algebra

- Read Sections 2.4, 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**

SQL = WHAT

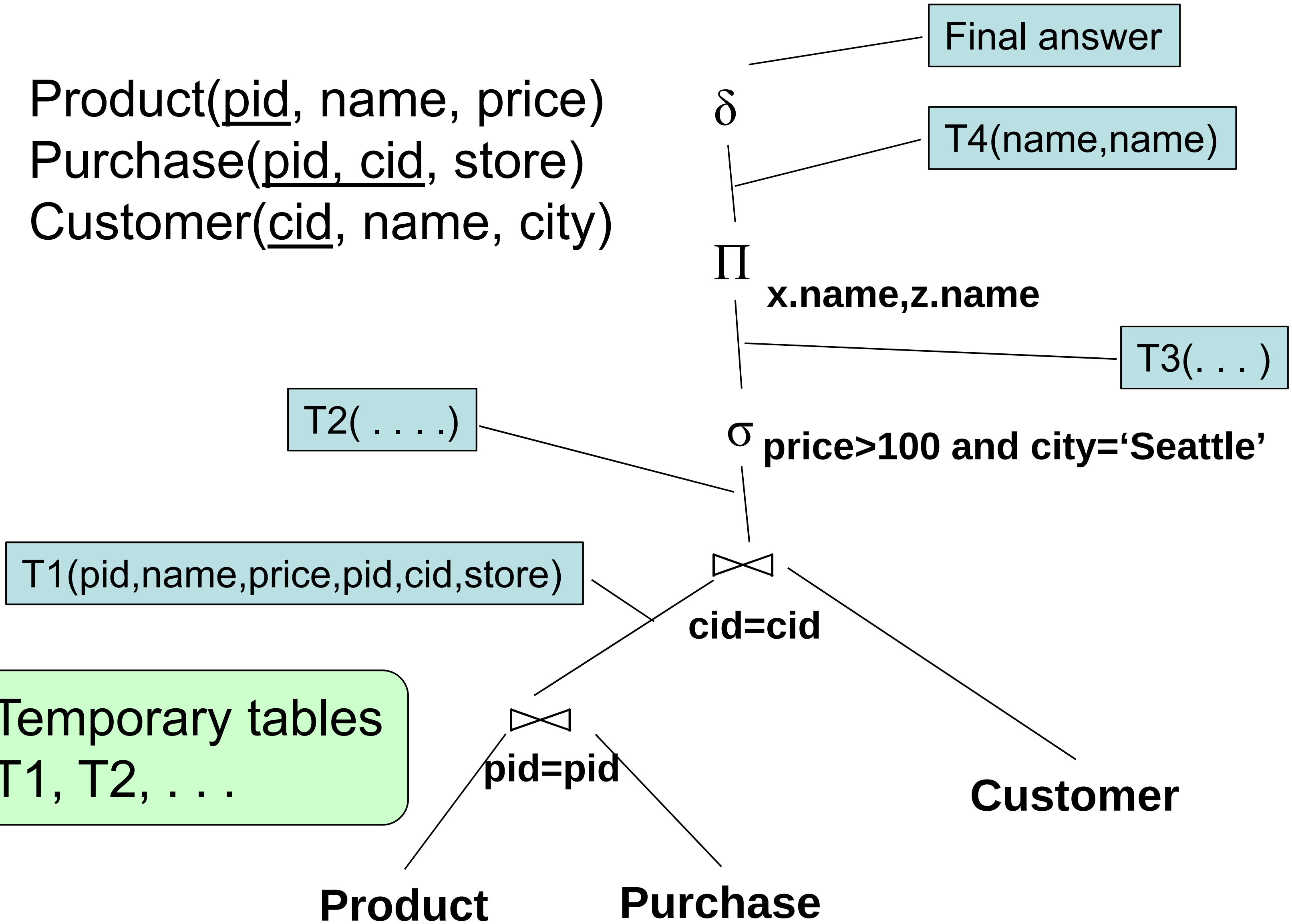
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.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)



Temporary tables
T1, T2, . . .

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 flavors:

- Over sets: theoretically elegant but limited
- Over bags: needed for SQL queries + more efficient
  - Example: Compute average price of all products

We discuss set semantics

- We mention bag semantics only where needed

Relational Algebra

- **Query language** associated with relational model
- **Queries specified in an operational manner**
  - A query gives a step-by-step procedure
- **Relational operators**
  - Take one or two relation instances as argument
  - Return one relation instance as result
  - Easy to **compose** into **relational algebra expressions**

Relational Algebra (1/3)

Five basic operators:

- Union ($\cup$) and Set difference (–)
- Selection: : $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge$,$\vee$) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- Cross-product or cartesian product ($\times$)

Relational Algebra (2/3)

Derived or auxiliary operators:

- Intersection ($\cap$), Division (R/S)
- Join: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- Variations of joins
  - Natural, equijoin, theta-join
  - Outer join and semi-join
- Rename $\rho_{B1,\ldots,Bn}(S)$

Relational Algebra (3/3)

Extensions for bags

- Duplicate elimination: δ
- Group by: γ [Same symbol as aggregation]
  - Partitions tuples of a relation into “groups”
- Sorting: τ

Other extensions

- Aggregation: γ (min, max, sum, average, count)

Union and Difference

- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees

- R1 – R2
- Example:
  - AllEmployees – RetiredEmployees

Be careful when applying to bags!

What about Intersection ?

- It is a derived operator
- $R1 \cap R2 = R1 – (R1 – R2)$
- Also expressed as a join (will see later)
- Example
  – UnionizedEmployees $\cap$ RetiredEmployees

Selection

- Returns all tuples that satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{Salary > 40000}$ (Employee)
  - $\sigma_{name = \text{“Smith”}}$ (Employee)
- The condition c can be
  - Boolean combination ($\wedge$,$\vee$) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >

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
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

Semantics differs over set or over bags

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

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

Name	Salary
John	20000
John	60000

Set semantics: duplicate elimination automatic

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

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

Name	Salary
John	20000
John	60000
John	20000

Bag semantics: no duplicate elimination; need explicit δ

Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: R1 $\times$ R2
- Example:
  - Employee $\times$ Dependents
- Very rare in practice; mainly used to express joins

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B1,\ldots,Bn}$ (R)
- Example:
  - $\rho_{LastName,\ SocSocNo}$ (Employee)
  - Output schema: Answer(LastName, SocSocNo)

Different Types of Join

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ

- **Equijoin**: $R \bowtie_\theta S = \pi_A (\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes
  - By far most used join in practice

- **Natural join**: $R \bowtie S = \pi_A (\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** common attributes (names) in R and in S
  - Projection drops duplicate common attributes

Theta-Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$$P \bowtie_{P.age=J.age \wedge P.zip=J.zip \wedge P.age < 50} J$$

P.age	P.zip	disease	job	J.age	J.zip
20	98120	flu	cashier	20	98120

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$P \bowtie_{P.age=J.age} J$

age	P.zip	disease	job	J.zip
54	98125	heart	lawyer	98125
20	98120	flu	cashier	98120

Natural Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$P \bowtie J$

age	zip	disease	job
54	98125	heart	lawyer
20	98120	flu	cashier

So Which Join Is It ?

- When we write R $\bowtie$ S we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$P \bowtie^{o} V$

age	zip	disease	job
54	98125	heart	lawyer
20	98120	flu	cashier
33	98120	lung	null

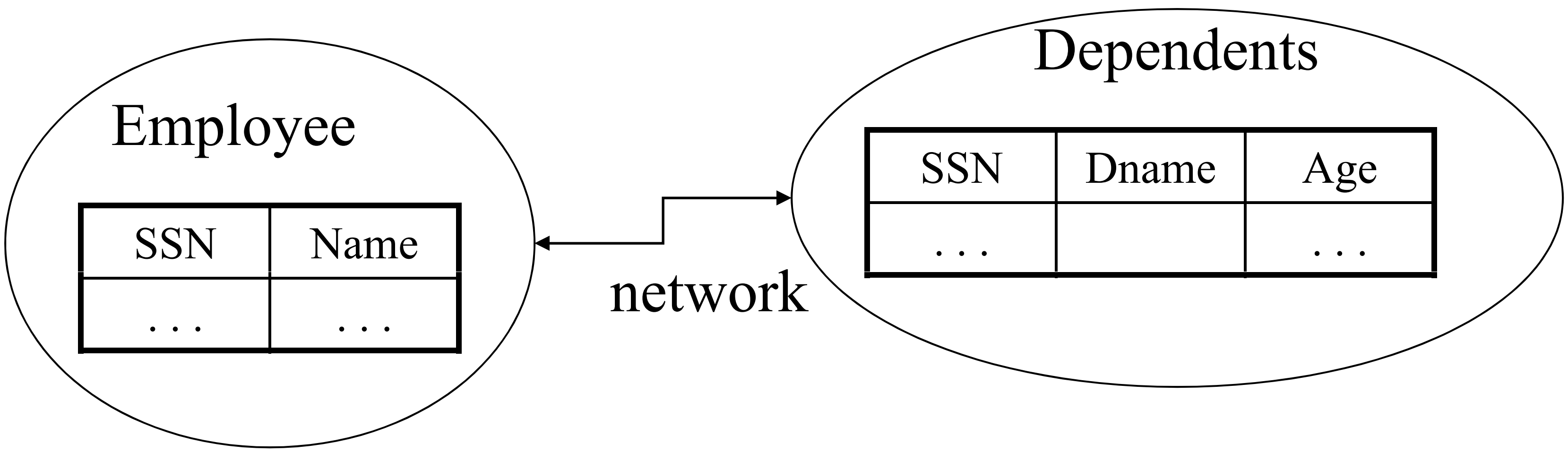
Semijoin

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:
  - Employee $\ltimes$ Dependents

- Particularly useful in distributed databases
  - Compute the query with minimum amount of data transfer

Semijoins in Distributed Databases

- Semijoins are used in distributed databases



$$\text{Employee} \bowtie_{ssn=ssn} (\sigma_{age>71} (\text{Dependents}))$$

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

$$R = \text{Employee} \ltimes T$$

$$\text{Answer} = R \bowtie \text{Dependents}$$

Operators on Bags

- Duplicate elimination δ

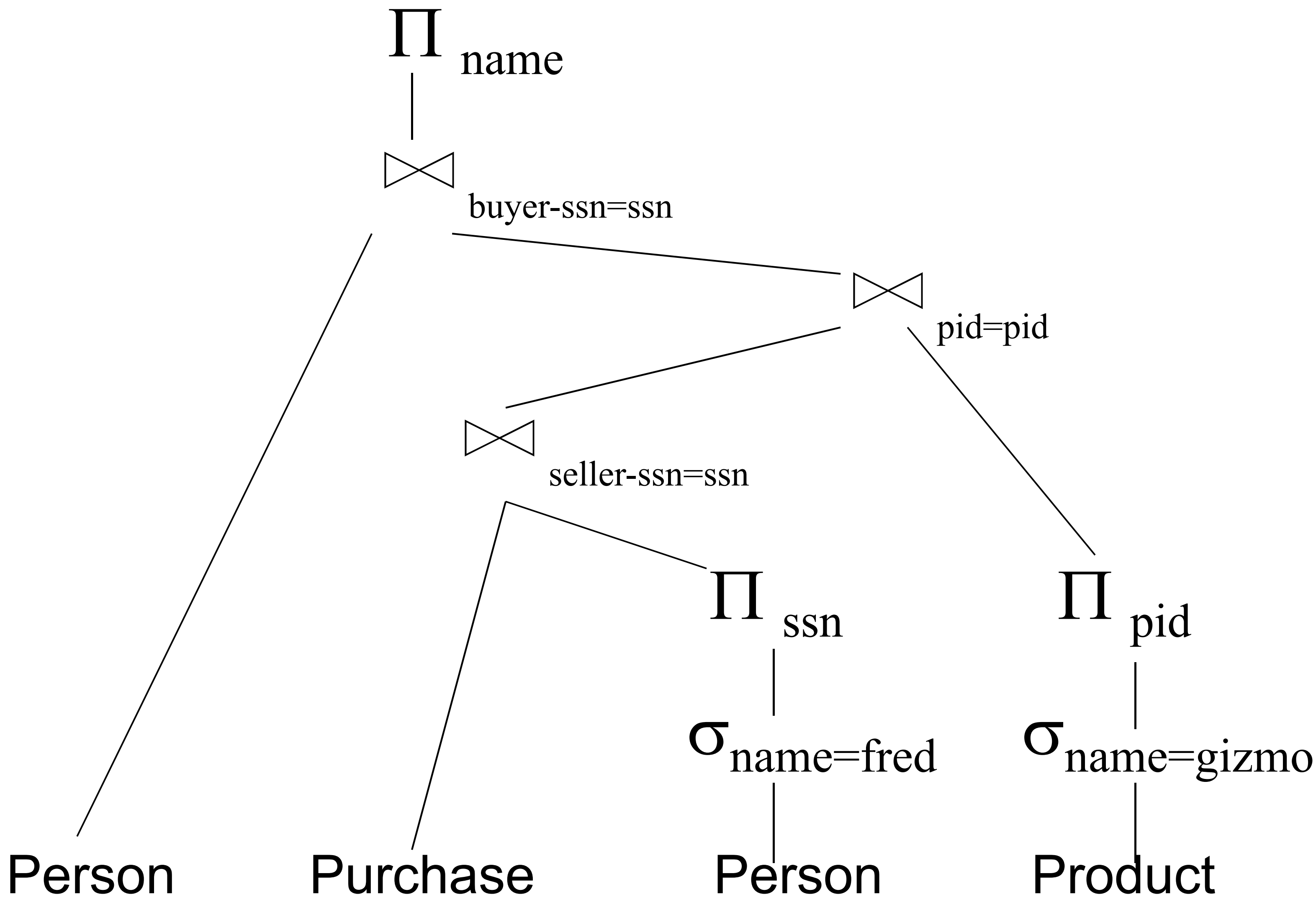
  δ(R) = select distinct * from R

- Grouping γ

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

- Sorting τ

Complex RA Expressions



RA = Dataflow Program

- An Algebra Expression is like a program
  - Several operations
  - Strictly specified order
- But Algebra expressions have limitations

RA and Transitive Closure

- 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