Introduction to Database Systems
CSE 414

Lecture 11: More Relational Algebra

Announcements

- WQ4/HW4 released
  - Both due next Tuesday
- Please make sure you get your AWS set up!
  - Will need for HW6
- Do not use seaquill for data storage
  - Machine gets wiped out periodically

Relational Algebra Operators

- Union ∪, ~~intersection ∩~~, difference -
- Selection σ
- Projection π
- Cartesian product X, join ⋈
- (Rename ρ)
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

RA

Extended RA

All operators take in 1 or more relations as inputs and return another relation

---

Composing RA Operators

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}$(Patient)

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease='heart'}$(Patient)

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

$\pi_{zip,disease}(\sigma_{disease='heart'}$(Patient))

zip	disease
98125	heart
98120	heart

Natural Join

R1 ⋈ R2

- Meaning: $R1 \bowtie R2 = \Pi_A(\sigma_\theta(R1 \times R2))$
- Where:
  - Selection σ_θ checks equality of all common attributes (i.e., attributes with same names)
  - Projection Π_A eliminates duplicate common attributes

Join Summary

- **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
  - No projection
- **Equijoin**: $R \bowtie_\theta S = \sigma_\theta (R \times S)$
  - Join condition θ consists only of equalities
  - No projection
- **Natural join**: $R \bowtie S = \pi_A (\sigma_\theta (R \times S))$
  - Equality on **all** fields with same name in R and in S
  - Projection π_A drops all redundant attributes

Some Examples

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie (\text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}))))$

Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie (\text{Supply} \bowtie (\sigma_{\ psize>10}\ (\text{Part}) \cup \sigma_{pcolor='red'}\ (\text{Part})\ )\ )\ )$

$\pi_{sname}(\text{Supplier} \bowtie (\text{Supply} \bowtie (\sigma_{\ psize>10\ \vee\ pcolor='red'}\ (\text{Part})\ )\ )\ )$

Some Examples

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

Name of supplier of parts with size greater than 10

```
 Project[sname](Supplier Join[sno=sno]
              (Supply Join[pno=pno] (Select[psize>10](Part))))
```

Name of supplier of red parts or parts with size greater than 10

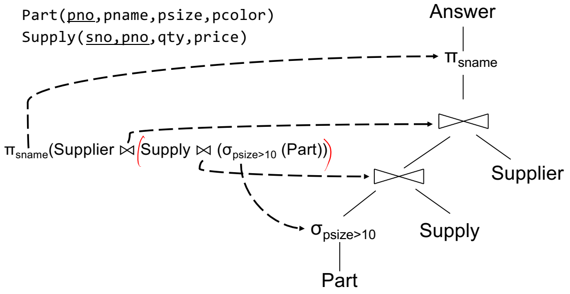
```
Project[sname](Supplier Join[sno=sno]
              (Supply Join[pno=pno]
              ((Select[psize>10](Part)) Union
                                         (Select[pcolor='red'](Part)))

Project[sname](Supplier Join[sno=sno] (Supply Join[pno=pno]
                                    (Select[psize>10 OR pcolor='red'](Part))))
```

Can be represented as trees as well

Representing RA Queries as Trees

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```



Relational Algebra Operators

- Union ∪, ~~intersection ∩,~~ difference -
- Selection σ
- Projection π
- Cartesian product X, join ⋈
- (Rename ρ)
- Duplicate elimination δ
- Grouping and aggregation ɣ
- Sorting **τ**

RA

Extended RA

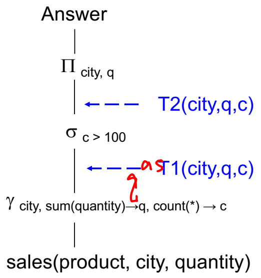
All operators take in 1 or more relations as inputs and return another relation

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Sorting τ
  - Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.

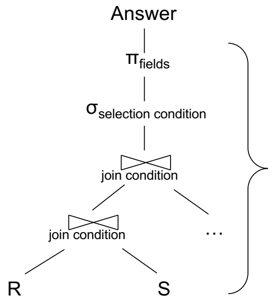
Using Extended RA Operators

```
SELECT city, sum(quantity)
FROM sales
GROUP BY city
HAVING count(*) > 100
```



T1, T2 = temporary tables

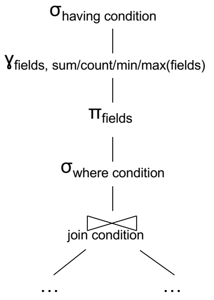
Typical Plan for a Query (1/2)



```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN
Query

Typical Plan for a Query (1/2)



```
SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING condition
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

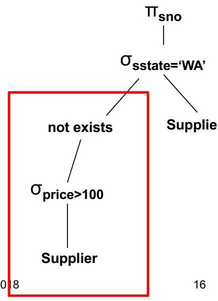
```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = ‘WA’
  and not exists
  (SELECT *
   FROM Supply AS P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

Option 1: create nested plans

```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = ‘WA’
  and not exists
  (SELECT *
   FROM Supply AS P
   WHERE P.sno = Q.sno
         and P.price > 100)
```



Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = ‘WA’
  and not exists
  (SELECT *
   FROM Supply AS P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

Correlation !

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = ‘WA’
  and not exists
  (SELECT *
   FROM Supply AS P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = ‘WA’
  and Q.sno not in
  (SELECT P.sno
   FROM Supply AS P
   WHERE P.price > 100)
```

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

How about Subqueries?

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier AS Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply AS P
   WHERE P.price > 100)
```

```
SELECT  Q.sno
FROM Supplier AS Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
    FROM Supply AS P
    WHERE P.price > 100)
```

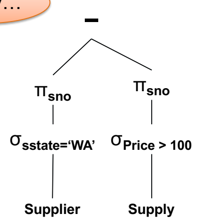
EXCEPT = set difference

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

How about Subqueries?

```
(SELECT  Q.sno
 FROM Supplier AS Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply AS P
   WHERE P.price > 100)
```

Finally…



Summary of RA and SQL

- SQL = a declarative language where we say *what* data we want to retrieve
- RA = an algebra where we say *how* we want to retrieve the data
- **Theorem**: SQL and RA can express exactly the same class of queries

RDBMS translate SQL → RA, then optimize RA

Summary of RA and SQL

- SQL (and RA) cannot express ALL queries that we could write in, say, Java
- Example:
  - Parent(p,c):   find all descendants of 'Alice'
  - No RA query can compute this!
  - This is called a *recursive query*
  - *Use Datalog!*

Datalog v.s. RA (and SQL)

- Datalog without recursion, but with negation and aggregates expresses the same queries as RA: next slides

---

R(A,B,C)
S(A,B,C)
T(G,H)

RA to Datalog by Examples

Union:
R(A,B,C) ∪ S(A,B,C)

U(x,y,z) :- R(x,y,z).
U(x,y,z) :- S(x,y,z).

R(A,B,C)
S(A,B,C)
T(G,H)

RA to Datalog by Examples

Intersection:

$R(A,B,C) \cap S(A,B,C)$

I(x,y,z) :- R(x,y,z), S(x,y,z).

R(A,B,C)
S(D,E,F)
T(G,H)

RA to Datalog by Examples

Selection: $\sigma_{A>100 \text{ and } B=\text{'foo'}}(R)$

L(x,y,z) :- R(x,y,z), x > 100, y='foo'.

Selection: $\sigma_{A>100 \textbf{ or } B=\text{'foo'}}(R)$

L(x,y,z) :- R(x,y,z), x > 100.

L(x,y,z) :- R(x,y,z), y='foo'.

R(A,B,C)
S(D,E,F)
T(G,H)

RA to Datalog by Examples

Equi-join: $R \bowtie_{\boxed{R.A=S.\underline{D} \text{ and } R.B=S.\underline{E}}} S$

!= natural join

J(x,y,z,q) :- R($\underline{x,y}$,z), S($\underline{x,y}$,q).

R(A,B,C)
S(D,E,F)
T(G,H)

RA to Datalog by Examples

Projection: $\Pi_A(R)$

P(x) :- R(x,y,z).

R(A,B,C)
S(D,E,F)
T(G,H)

RA to Datalog by Examples

To express difference, we add negation

$R - S$

D(x,y,z) :- R(x,y,z), NOT S(x,y,z).

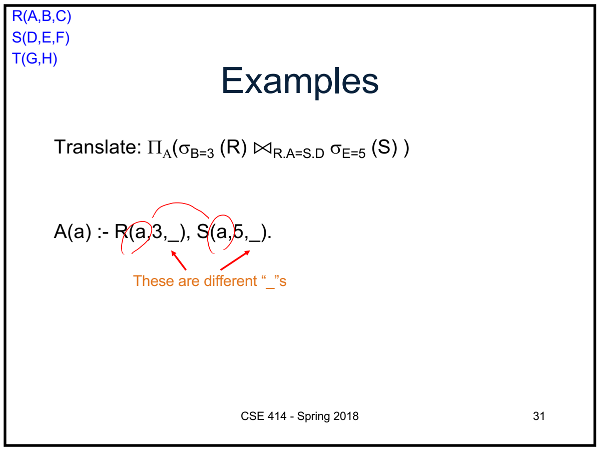
R(A,B,C)
S(D,E,F)
T(G,H)

Examples

Translate: $\Pi_A(\sigma_{B=3}(R))$

A(a) :- R(a,3,_).

Underscore used to denote an "anonymous variable"
Each such variable is unique



R(A,B,C)
S(D,E,F)
T(G,H)

Examples

Translate: $\Pi_A(\sigma_{B=3}(R) \bowtie_{R.A=S.D} \sigma_{E=5}(S))$

A(a) :- R(a,3,_), S(a,5,_).

These are different "_"s