Introduction to Database Systems
CSE 414

Lectures 9-10: Relational Algebra
Announcements

• Webquiz 3 due Monday night, 11 pm

• HW3 due Wednesday night, 11 pm
  – Log on to your Azure account by now

• Today’s lecture: secs. 2.4 and 5.1
Where We Are

• Motivation for using a DBMS for managing data
• SQL, SQL, SQL
  – Declaring the schema for our data (CREATE TABLE)
  – Inserting data one row at a time or in bulk (INSERT/.import)
  – Modifying the schema and updating the data (ALTER/UPDATE)
  – Querying the data (SELECT)
  – Tuning queries (CREATE INDEX)

• Next step: More knowledge of how DBMSs work
  – Client-server architecture
  – Relational algebra and query execution
Query Evaluation Steps

1. **Parse & Check Query**
   - Translate query string into internal representation
   - Check syntax, access control, table names, etc.

2. **Decide how best to answer query: query optimization**

3. **Query Execution**

4. **Return Results**

**Query Evaluation**

---

*Note: The diagram illustrates the flow of query evaluation steps.*
The WHAT and the HOW

- SQL = **WHAT** we want to get form the data

- Relational Algebra = **HOW** to get the data we want

- The passage from **WHAT** to **HOW** is called query optimization
Overview: 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
Overview: Relational Algebra = HOW

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

δ

Π

x.name, z.name

price > 100 and city = 'Seattle'

T1(pid, name, price, pid, cid, store)
T2(....)
T4(name, name)
T3(....)

Final answer

Execution order is now clearly specified

Customer

Product

Purchase

But a lot of physical details are still left open!
Relational Algebra
Sets v.s. Bags

• Sets: \{a, b, c\}, \{a, d, e, f\}, \{\}\ldots
• Bags: \{a, a, b, c\}, \{b, b, b, b, b\}\ldots

Relational Algebra has two semantics:
• Set semantics = standard Relational Algebra
• Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)
Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\Pi$
- Cartesian product $\times$, join $\Join$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$

CSE 414 - Spring 2015
Union and Difference

\[ R_1 \cup R_2 \]
\[ R_1 - R_2 \]

What do they mean over bags?
What about Intersection?

• Derived operator using minus

\[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]

• Derived using join (will explain later)

\[ R_1 \cap R_2 = R_1 \Join R_2 \]
Selection

• Returns all tuples which satisfy a condition

\[ \sigma_c(R) \]

• Examples
  - \( \sigma_{\text{Salary} > 40000} (\text{Employee}) \)
  - \( \sigma_{\text{name} = "Smith"} (\text{Employee}) \)

• The condition c can be =, <, ≤, >, ≥, <>
\[ \sigma_{\text{Salary} > 40000} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>
Projection

- Eliminates columns

\[ \Pi_{A_1, \ldots, A_n}(R) \]

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

Different semantics over sets or bags! Why?
### Employee Table

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

### Bag semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

### Set semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>

Which is more efficient?
Composing RA Operators

Patient

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{zip,disease}}(\text{Patient}) \]

<table>
<thead>
<tr>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{disease}='\text{heart}'}(\text{Patient}) \]

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{zip}}(\sigma_{\text{disease}='\text{heart}'}(\text{Patient})) \]

<table>
<thead>
<tr>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>98120</td>
</tr>
<tr>
<td>98125</td>
</tr>
</tbody>
</table>
Cartesian Product

• Each tuple in R1 with each tuple in R2

\[ R1 \times R2 \]

• Rare in practice; mainly used to express joins
# Cross-Product Example

## Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

## Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

## Employee X Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Renaming

• Changes the schema, not the instance

\[ \rho_{B_1, \ldots, B_n}(R) \]

• Example:
  - \( \rho_{N, S}(Employee) \rightarrow \text{Answer}(N, S) \)

Not really used by systems, but needed on paper
Natural Join

\[ R_1 \bowtie R_2 \]

• **Meaning:** \( R_1 \bowtie R_2 = \Pi_A(\sigma(R_1 \times R_2)) \)

• **Where:**
  - Selection \( \sigma \) checks equality of all common attributes
  - Projection eliminates duplicate common attributes
Natural Join Example

\[
R \Join S = \Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))
\]

<table>
<thead>
<tr>
<th>R</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
</tbody>
</table>
## Natural Join Example 2

### AnonPatient P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

### Voters V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

### \( P \bowtie V \)

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
</tr>
</tbody>
</table>
Natural Join

• Given schemas $R(A, B, C, D), S(A, C, E)$, what is the schema of $R \bowtie S$?

• Given $R(A, B, C), S(D, E)$, what is $R \bowtie S$?

• Given $R(A, B), S(A, B)$, what is $R \bowtie S$?
Theta Join

- A join that involves a predicate

\[ R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2) \]

- Here \( \theta \) can be any condition
- For our voters/disease example:

\[ P \bowtie_{P.zip = V.zip \text{ and } P.age < V.age + 5 \text{ and } P.age > V.age - 5} V \]
Equijoin

• A theta join where $\theta$ 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
## Equijoin Example

### AnonPatient $P$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

### Voters $V$

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

The equijoin $P \bowtie_{P.age = V.age} V$ results in:

<table>
<thead>
<tr>
<th>age</th>
<th>P.zip</th>
<th>disease</th>
<th>name</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>98120</td>
</tr>
</tbody>
</table>
Join Summary

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

• **Equijoin**: $\text{R} \bowtie_\theta S = \pi_A (\sigma_\theta(R \times S))$
  – Join condition $\theta$ consists only of equalities
  – Projection $\pi_A$ drops all redundant attributes

• **Natural join**: $\text{R} \bowtie S = \pi_A (\sigma_\theta(R \times S))$
  – Equijoin
  – Equality on **all** fields with same name in $R$ and in $S$
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

<table>
<thead>
<tr>
<th>AnonPatient P</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
</tr>
</tbody>
</table>

| P ⊙ J |

<table>
<thead>
<tr>
<th>AnonJob J</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
<td>age</td>
</tr>
<tr>
<td>lawyer</td>
<td>54</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
</tr>
</tbody>
</table>
Some Examples

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

Q2: Name of supplier of parts with size greater than 10
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10} (\text{Part}))) \]

Q3: Name of supplier of red parts or parts with size greater than 10
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10} (\text{Part}) \cup \sigma_{pcolor='red'} (\text{Part}))) \]
From SQL to RA
From SQL to RA

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'
```
From SQL to RA

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

δ

Π x.name, z.name

σ price>100 and city='Seattle'

δ

cid=cid

Product

Purchase

Customer
An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions

```
δ

Π x.name,z.name

cid=cid

pid=pid

σ price>100

Product

Purchase

Customer

σ city='Seattle'
```
Extended RA: Operators on Bags

• Duplicate elimination $\delta$
• Grouping $\gamma$
• Sorting $\tau$
Logical Query Plan

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

T1, T2, T3 = temporary tables

T1(city, p, c) \( \gamma \) city, sum(price)→p, count(*)→c

T2(city, p, c) \( \sigma \) p > 100

T3(city, c) \( \Pi \) city, c

sales(product, city, price)
Typical Plan for Block (1/2)

SELECT-PROJECT-JOIN
Query
Typical Plan For Block (2/2)

\[ \text{having}_\text{condition} \]

\[ \gamma \text{ fields, sum/count/min/max(fields)} \]

\[ \pi \text{ fields} \]

\[ \sigma \text{ selection condition} \]

\[ \text{join condition} \]

\[ \cdots \]
How about Subqueries?

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
        (SELECT *
         FROM  Supply P
         WHERE P.sno = Q.sno
         and P.price > 100)
```
How about Subqueries?

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

Correlation!
How about Subqueries?

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

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

- Supplier(sno,sname,scity,sstate)
- Part(pno,pname,psize,pcolor)
- Supply(sno,pno,price)
How about Subqueries?

\[
\begin{align*}
\text{(SELECT} & \text{ Q.sno} \\
\text{FROM} & \text{ Supplier Q} \\
\text{WHERE} & \text{ Q.sstate} = \text{‘WA’}) \\
\text{EXCEPT} & \text{ (SELECT} \text{ P.sno} \\
\text{FROM} & \text{ Supply P} \\
\text{WHERE} & \text{ P.price} \text{ > 100}) \\
\text{EXCEPT} & \text{ set difference}
\end{align*}
\]

Un-nesting

\[
\begin{align*}
\text{SELECT} & \text{ Q.sno} \\
\text{FROM} & \text{ Supplier Q} \\
\text{WHERE} & \text{ Q.sstate} = \text{‘WA’} \text{ and Q.sno not in} \\
\text{(SELECT} & \text{ P.sno} \\
\text{FROM} & \text{ Supply P} \\
\text{WHERE} & \text{ P.price} \text{ > 100})
\end{align*}
\]

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)
How about Subqueries?

(\text{\textbf{SELECT} } Q.sno \\
\text{\textbf{FROM} } \text{Supplier} \ Q \\
\text{\textbf{WHERE} } Q.sstate = 'WA')
\text{\textbf{EXCEPT}}
(\text{\textbf{SELECT} } P.sno \\
\text{\textbf{FROM} } \text{Supply} \ P \\
\text{\textbf{WHERE} } P.price > 100)

\text{Supplier}(sno, sname, scity, sstate)
\text{Part}(pno, pname, psize, pcolor)
\text{Supply}(sno, pno, price)
From Logical Plans to Physical Plans
Example

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Give a relational algebra expression for this query
Relational Algebra

\[ \pi_{\text{sname}} \left( \sigma_{\text{scity} = \text{Seattle} \land \text{sstate} = \text{WA} \land \text{pno} = 2} (\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply}) \right) \]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Relational Algebra

Relational algebra expression is also called the “logical query plan”
Physical Query Plan 1

\[
\sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land pno = 2} \pi_{\text{sname}}
\]

A physical query plan is a logical query plan annotated with physical implementation details.
Suppliers \((\text{sid, sname, scity, sstate})\) 

Validations \(\text{Supply}(\text{sid, pno, quantity})\)

Physical Query Plan 2

- (On the fly) \(\pi_{\text{sname}}\) (d)
- (Sort-merge join) \(\text{sid} = \text{sid}\) (c)
- (Scan write to T1) \((\text{Scan write to T2})\)
  \(\sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA'}\) (a)
  \(\sigma_{\text{pno} = 2}\) (b)

\(\text{Supplier (File scan)}\)
\(\text{Supply (File scan)}\)

Different but equivalent logical query plan; different physical plan
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Query Plan 3

(On the fly) (d) \( \pi_{sname} \)

(On the fly)

(On the fly)

(c) \( \sigma_{scity= 'Seattle'} \land sstate= 'WA' \)

(b) \( \text{sid} = \text{sid} \) (Index nested loop)

(Use index)

(a) \( \sigma_{pno=2} \)

Supply (Index lookup on pno )
Assume: clustered

Supplier (Index lookup on sid)
 Doesn’t t matter if clustered or not

Another logical plan that produces the same result and is implemented with a different physical plan
Physical Data Independence

• Means that applications are insulated from changes in physical storage details
  – E.g., can add/remove indexes without changing apps
  – Can do other physical tunings for performance

• SQL and relational algebra facilitate physical data independence because both languages are “set-at-a-time”: Relations as input and output