Lectures 6: Relational Algebra
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)

• 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**
   - Logical plan → physical plan

3. **Query Execution**

4. **Return Results**

---

SQL query

- Logical plan → physical plan
- Query evaluation

---

Translate query string into internal representation

Check syntax, access control, table names, etc.

Logical plan → physical plan

Query Evaluation
The WHAT and the HOW

• SQL = **WHAT** we want to get from the data

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

• The passage from **WHAT** to **HOW** is called query optimization
  – SQL -> Relational Algebra -> Physical Plan
  – Relational Algebra = Logical Plan
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

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’

Execution order is now clearly specified

Logical plan
Many physical details are still left open!
Relational Algebra
Edgar Frank “Ted” Codd

"A Relational Model of Data for Large Shared Data Banks"
1970

Turing Award 1981
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 = 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 $\tau$

Extended RA
Union and Difference

\[ R_1 \cup R_2 \]

\[ R_1 - R_2 \]
What about Intersection?

- Derived operator using minus
  \[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]

- Derived using join (will explain later)
  \[ R_1 \cap R_2 = R_1 \bowtie R_2 \]
Select

• Returns all tuples which satisfy a condition

\[ \sigma_c(R) \]

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

• The condition c can be \( =, <, \leq, >, \geq, <> \)
### 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>

\[ \sigma_{\text{Salary} > 40000} \ (\text{Employee}) \]

<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>
Projection

• Eliminates columns

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

• Example: project social-security number and names:
  - \[ \Pi \text{SSN, Name} \ (\text{Employee}) \]
  - Answer(SSN, Name)
### 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>John</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

\[ \Pi_{\text{Name}, \text{Salary}} (\text{Employee}) \]

**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>
Composing RA Operators

<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>

\( \sigma_{\text{disease=’heart’}}(\text{Patient}) \)

<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>

\( \pi_{\text{zip,disease}}(\sigma_{\text{disease=’heart’}}(\text{Patient})) \)

<table>
<thead>
<tr>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>98120</td>
<td>heart</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>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</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 × Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</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:
  – \( \rho_{N, S}(\text{Employee}) \) \( \rightarrow \) \( \text{Answer}(N, S) \)

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

\[ R_1 \bowtie R_2 \]

• Meaning: \[ R_1 \bowtie R_2 = \Pi_A (\sigma_\theta (R_1 \times R_2)) \]

• Where:
  – Selection \( \sigma \) checks equality of all common attributes (attributes with same names)
  – Projection eliminates duplicate common attributes
Natural Join Example

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

\[ R1 \Join_{\theta} R2 = \sigma_{\theta} (R1 \times R2) \]

- Here \( \theta \) can be any condition

- For our voters/patients example:

  \[ P \Join P.zip = V.zip \text{ and } P.age \geq V.age -1 \text{ and } P.age \leq V.age +1 \]
Equijoin

- A theta join where \( \theta \) is an equality predicate
- Projection drops all redundant attributes

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

- 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>

$$P \bowtie_{P\text{.age}=V\text{.age}} V$$

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

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

• **Natural join**: $R \bowtie S = \pi_{A} (\sigma_{\theta}(R \times S))$
  - Equijoin
  - Equality on all fields with same name in $R$ and in $S$
  - Projection $\pi_{A}$ drops all redundant attributes
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
  – Does not eliminate duplicate columns

• **Variants**
  – Left outer join
  – Right outer join
  – Full outer join
## Outer Join 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>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
</tr>
</tbody>
</table>

**AnonJob J**

<table>
<thead>
<tr>
<th>job</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>disease</th>
<th>job</th>
<th>J.age</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
<td>33</td>
<td>98120</td>
</tr>
</tbody>
</table>
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
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part})) \]

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

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

```sql
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'
```
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'

δ

Π

σ

price>100 and city='Seattle'

Customer

Product

Purchase

pid=pid

cid=cid

x.name,z.name
An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions

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'

[Diagram of a tree with nodes labeled with conditions and operations, illustrating the query optimization process.]
Extended RA: Operators on Bags

- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$