Announcements

- WQ4 is posted and due on Nov. 3, 11pm
- HW2 will be due next Monday 11pm

Where We Are

- Motivation for using a DBMS for managing data
- 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

SQL query

1. Parse & Check Query
   - Check syntax, access control, table names, etc.

2. Decide how best to answer query: query optimization
   - Logical plan → physical plan

3. Query Execution
   - Return Results

The WHAT and the HOW

- SQL = WHAT we want to get from the data
- Relational Algebra = HOW to get the data we want
- Move from WHAT to HOW is query optimization
  - SQL → Relational Algebra → Physical Plan
  - Relational Algebra = Logical Plan

Relational Algebra
Sets v.s. Bags

- Sets: \{a, b, c\}, \{a, d, e, f\}, \{\}\ldots
- Bags: \{a, a, b, c\}, \{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?)

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)
  \[ R_1 \cap R_2 = R_1 \bowtie R_2 \]

Selection

- Returns all tuples which satisfy a condition
  \[ \sigma_c(R) \]
- Examples
  - \[ \sigma_{\text{salary} > 40000}(\text{Employee}) \]
  - \[ \sigma_{\text{name} = \text{"Smith"}}(\text{Employee}) \]
- The condition \( c \) can be =, <, <=, >, >=, <> combined with AND, OR, NOT

Selection Examples

<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>

Employee Selection

<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:
  - \( \pi_{\text{SSN, Name}}(\text{Employee}) \)
  - \( \text{Answer}(\text{SSN, Name}) \)

Different semantics over sets or bags! Why?

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>

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

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

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

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>

\( \pi_{\text{Name, Salary}}(\text{Employee}) \)

<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>

Bag semantics

Set semantics

Which is more efficient?

Cartesian Product

- Each tuple in R1 with each tuple in R2
  \( R_1 \times R_2 \)
- 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>99999999</td>
</tr>
<tr>
<td>Tony</td>
<td>77777777</td>
</tr>
</tbody>
</table>

Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999999</td>
<td>Emily</td>
</tr>
<tr>
<td>77777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

Employee \times Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>99999999</td>
<td>99999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>99999999</td>
<td>77777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>77777777</td>
<td>99999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>77777777</td>
<td>77777777</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 \( \pi \) eliminates duplicate common attributes

\[
\begin{array}{c|c|c|c}
\text{AnonPatient} P & \text{Voters} V \\
\hline
\text{name} & \text{age} & \text{zip} & \text{disease} \\
\hline
p1 & 54 & 98126 & \text{heart} \\
p2 & 20 & 98120 & \text{flu} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{age} & \text{zip} & \text{disease} \\
\hline
54 & 98125 & \text{heart} \\
20 & 98120 & \text{flu} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{name} & \text{age} & \text{zip} \\
\hline
p1 & 54 & 98126 \\
p2 & 20 & 98120 \\
\end{array}
\]

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/patients example:
  \[
  P \bowtie_\theta P.\text{zip} = V.\text{zip} \text{ and } P.\text{age} \geq V.\text{age} \text{ and } P.\text{age} \leq V.\text{age} + 1 \\
  \]

Equijoin

- A theta join where \( \theta \) is an equality predicate
- 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 ≈\_age=\_age V

So Which Join Is It?

When we write R ⨝ S, we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.

JOIN SUMMARY

- **Theta-join**: R ⨝_{\theta} S = \sigma_{\theta}(R \times S)
  - Join of R and S with a join condition \( \theta \)
  - Cross-product followed by selection \( \theta \)
- **Equijoin**: R ⨝_{\theta} S = \sigma_{\theta}(R \times S)
  - Join condition \( \theta \) consists only of equalities
- **Natural join**: R ⨝ 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

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

MORE EXAMPLES

Name of supplier of parts with size greater than 10
\[ \pi_{sname}(\sigma_{psize>10}(Part)) \]

Name of supplier of red parts or parts with size greater than 10
\[ \pi_{sname}(\sigma_{pcolor='red'}(Part) \cup \sigma_{psize>10}(Part)) \]