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
- Parse & Check Query
- Translate query string into internal representation
- Check syntax, access control, table names, etc.
- Logical plan -> physical plan
- Decide how best to answer query: query optimization
- Return Results
- 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
- Move from WHAT to HOW is query optimization
  - SQL -> Relational Algebra -> Physical Plan
  - Relational Algebra = Logical Plan

Relational Algebra
Sets vs. 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?)

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 \(\tau\)

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 that 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 =, \(<\), \(\leq\), \(>\), \(\geq\), \(<>\) 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 salary Example:

<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:
  \[ \Pi_{\text{SSN}, \text{Name}}(\text{Employee}) \]
  \[ \text{Answer(SSN, Name)} \]

Different semantics over sets or bags! Why?

Composing RA Operators

Patient

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

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

Cartesian Product

- Each tuple in \( R_1 \) with each tuple in \( R_2 \)
  \[ R_1 \times R_2 \]

- Rare in practice; mainly used to express joins

Cross-Product Example

Employee x Dependent

\[ \pi_{\text{Name}, \text{SSN}}(\text{Employee}) \times \pi_{\text{EmpSSN}, \text{DepName}}(\text{Dependent}) \]

Renaming

- Changes the schema, not the instance
  \[ \rho_{B_1, \ldots, B_n}(R) \]

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

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

\[ R_1 \bowtie R_2 \]

• Meaning: \( R_1 \bowtie R_2 = \pi_\alpha (R_1 \times R_2) \)

• Where:
  – Selection \( \sigma \) checks equality of all common attributes
  – Projection \( \pi \) eliminates duplicate common attributes

\[ \text{Natural Join Example} \]

\[ \begin{array}{ccc}
A & B & C \\
X & Y & U \\
X & Z & V \\
Y & Z & U \\
z & v & w
\end{array} \]

\[ \begin{array}{ccc}
\text{A} & \text{B} & \text{C} \\
X & Z & U \\
X & Z & V \\
Y & Z & U \\
Y & Z & V \\
z & v & w
\end{array} \]

\[ R \bowtie S = \pi_{A,B,C}(\sigma_{R.B=S.B}(R \times S)) \]

\[ \text{Natural Join Example 2} \]

<table>
<thead>
<tr>
<th>AnonPatient P</th>
<th>Voters V</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>
</tbody>
</table>

\[ P \bowtie V \]

| P | V |
|-----------------|
| age | zip | disease | name |
| 54 | 98125 | heart | p1 |
| 20 | 98120 | flu | p2 |

\[ \text{Theta Join} \]

• 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

\[ P \bowtie_\theta P \bowtie V, \text{zip and age} \geq V \text{age} -1 \text{ and age} \leq V \text{age} +1 \]

\[ \text{Equijoin} \]

• 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>

\[
P \times_{P.age = V.age} V
\]

- **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 \( \theta \)
  - **Equijoin**: \( R \bowtie_{=\theta} S = \sigma_{=\theta}(R \times S) \)
    - Join condition \( \theta \) consists only of equalities
  - **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.

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>
</tbody>
</table>

AnnonJob 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>

\[
P \bowtie J
\]

- **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
  \[
  \pi_{sname}(\sigma_{psize > 10}(Supplier \bowtie Supply))
  \]

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