Recap: Datalog
- Facts and Rules
- Selection, projection, join
- Recursive rules
- Grouping, aggregates
- Negation
- Safe vs unsafe rules
- Stratification

Class Overview
- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
  - Data models, SQL, Datalog, Relational Algebra
- Unit 3: Non-relational data
- Unit 4: RDBMS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions

Relational Algebra
- Set-based algebra that manipulates relations
  - We will extend it to multisets / bags
- In SQL & Datalog we say what we want
- In RA we can express how to get it
- Every DBMS implementations converts a SQL query to RA in order to execute it
- An RA expression is called a query plan

Why study yet another relational query language?
- RA is how SQL is implemented in DBMS
  - We will see more of this in a few weeks
- RA opens up opportunities for query optimization

Basics
- Relations and attributes
- Functions that are applied to relations
  - Return relations
  - Can be composed together
  - Often displayed using a tree rather than linearly
  - Use Greek symbols: \( \sigma \), \( \pi \), \( \delta \), etc

Introduction to Database Systems
CSE 414

Lecture 10: Relational Algebra

Class Overview
- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
  - Data models, SQL, Datalog, Relational Algebra
- Unit 3: Non-relational data
- Unit 4: RDBMS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions

Relational Algebra
- Set-based algebra that manipulates relations
  - We will extend it to multisets / bags
- In SQL & Datalog we say what we want
- In RA we can express how to get it
- Every DBMS implementations converts a SQL query to RA in order to execute it
- An RA expression is called a query plan

Why study yet another relational query language?
- RA is how SQL is implemented in DBMS
  - We will see more of this in a few weeks
- RA opens up opportunities for query optimization
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 $\#$

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

Union and Difference

$R_1 \cup R_2$
$R_1 - R_2$

Only make sense if $R_1, R_2$ have the same schema

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 (as we will see later)
  $$R_1 \cap R_2 = R_1 \Join R_2$$

Selection

- Returns all tuples which satisfy a condition
  $$\sigma_c(R)$$
- Examples
  - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
  - $\sigma_{\text{Salary} >= 50000}(\text{Employee})$
- The condition $c$ can be $=, <, <=, >, >=, <>$ combined with AND, OR, NOT

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

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

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

### Renaming
- Changes the schema, not the instance
- Example:
  - Given Employee(Name, SSN)
  - \( \rho_{N, S}(\text{Employee}) \rightarrow \text{Answer(N, S)} \)

### Natural Join
- Meaning: \( R1 \bowtie R2 = \Pi_A(\sigma_0(R1 \times R2)) \)
- Where:
  - Selection \( \sigma \) checks equality of all common attributes (i.e., attributes with same names)
  - Projection \( \Pi_A \) eliminates duplicate common attributes
Natural Join Example

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

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>Alice</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

Theta Join

- A join that involves a predicate
  \[ R1 \bowtie_\theta R2 = \sigma_\theta(R1 \times R2) \]
- Here \( \theta \) can be any condition
- No projection in this case!
- For our voters/patients example:
  \[ P \bowtie_\theta P.zip = V.zip \text{ and } P.age \geq V.age - 1 \text{ and } P.age \leq V.age + 1 \]

Equijoin

- A theta join where \( \theta \) is an equality predicate
  \[ R1 \bowtie_\theta R2 = \sigma_\theta(R1 \times R2) \]
- By far the most used variant of join in practice
- What is the relationship with natural join?
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 \( \theta \)
  - No projection

• **Equijoin:** \( R \bowtie \delta S = \sigma_{\delta} (R \times S) \)
  - Join condition \( \delta \) consists only of equalities
  - No projection

• **Natural join:** \( R \Join S = \pi_A (\sigma_{\theta} (R \times S)) \)
  - 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