RELATIONAL ALGEBRA

CS 564- Fall 2015
RELATIONAL QUERY LANGUAGES

• manipulation and retrieval of data from a database
• 2 types:
  – **Declarative**: Tuple Relational Calculus (**TRC**) Domain Relational Calculus (**DRC**)  
    • describe what a user wants, rather than how to compute it  
  – **Procedural**: Relational Algebra (**RA**)  
    • operational, useful for representing execution plans
QUERY VS PROGRAMMING LANGUAGES

Query Languages:
• are not “Turing complete”
• are not intended to be used for complex calculations
• support easy, efficient access to large data sets
What is Relational Algebra?

• **algebra**: mathematical system consisting of
  – **operands**: variables or values from which new values can be constructed
  – **operators**: symbols denoting procedures that construct new values from given values

• **relational algebra**: an algebra whose operands are relations or variables that represent relations
  – operators do the most common things that we need to do with relations in a database
  – can be used as a query language for relations
Relational Algebra Prelim

- **Query:**
  - Input: relational instances
  - Output: relational instances
  - specified using the schemas
    - may produce different results for different instances
    - the schema of the result is fixed

- Positional **vs** named-field notation:
  - C.name or
  - 2
Relational Algebra Prelim

• Basic operations:
  – *Selection* \( \{ \sigma \} \): selects a subset of rows
  – *Projection* \( \{ \pi \} \): deletes columns
  – *Cross-product* \( \{ \times \} \): combines two relations
  – *Set-difference* \( \{ - \} \), *Union* \( \{ \cup \} \)

• When the relations have named fields:
  – *Renaming* \( \{ \rho \} \)

• Additional operations:
  – *Intersection, join, division*
BASIC OPERATIONS
Selection

Notation: \( \sigma_C(R) \)
- C is a condition that refers to the attributes of R
- outputs the \text{rows} of R that satisfy C
- output schema: same as input schema

Example
- \( \sigma_{\text{age}>24}(Person) \)
- \( \sigma_{\text{age}>24 \text{ and } \text{age} \leq 28}(Person) \)
- \( \sigma_{\text{age}>24 \text{ and } \text{name} = \text{"Paris"}}(Person) \)
**SELECTION EXAMPLE**

**Person**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>25</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{age} > 24}(\text{Person}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>25</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>
**Projection**

Notation: $\pi_{A_1, A_2, \ldots, A_n}(R)$

- outputs only the columns $A_1, A_2, \ldots, A_n$
- removes any duplicate tuples (why?)
- output schema: $R(A_1, A_2, \ldots, A_n)$

**Example**

- $\pi_{SSN, age}(Person)$
- $\pi_{SSN, phoneNumber, age}(Person)$
### Projection Example

**Person**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

\[ \pi_{SSN, name}(\text{Person}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
</tr>
</tbody>
</table>
**Union**

Notation: $R_1 \cup R_2$

- outputs all tuples in $R_1$ or $R_2$
- both relations must have the same schema!
- output schema: same as input

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
</tr>
</tbody>
</table>

$\cup$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
</tr>
<tr>
<td>a₃</td>
<td>b₁</td>
</tr>
<tr>
<td>a₄</td>
<td>b₄</td>
</tr>
</tbody>
</table>

= 

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
</tr>
<tr>
<td>a₃</td>
<td>b₁</td>
</tr>
<tr>
<td>a₄</td>
<td>b₄</td>
</tr>
</tbody>
</table>
**DIFFERENCE**

Notation: \( R_1 - R_2 \)

- outputs all tuples in \( R_1 \) and not in \( R_2 \)
- both relations must have the same schema!
- output schema: same as input

\[
\begin{array}{cc}
A & B \\
\hline
a_1 & b_1 \\
a_2 & b_1 \\
a_2 & b_2 \\
\end{array}
\quad - \quad
\begin{array}{cc}
A & B \\
\hline
a_1 & b_1 \\
a_3 & b_1 \\
a_4 & b_4 \\
\end{array}
= \begin{array}{cc}
A & B \\
\hline
a_2 & b_1 \\
a_2 & b_2 \\
\end{array}
\]
**CROSS-PRODUCT**

Notation: $R_1 \times R_2$

- matches each tuples in $R_1$ with each tuple in $R_2$
- input schema: $R_1 (A_1, A_2, \ldots, A_n), R_2 (B_1, B_2, \ldots, B_m)$
- output schema: $R (A_1, \ldots, A_n, B_1, \ldots, B_m)$
# Cross-Product Example

<table>
<thead>
<tr>
<th>Person</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>name</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
</tr>
</tbody>
</table>

Person $\times$ Dependent
RENAMING

Notation: $\rho_{A_1, A_2, \ldots, A_n}(R)$
- does not change the instance, only the schema!
- input schema: $R(B_1, B_2, \ldots, B_n)$
- output schema: $R(A_1, \ldots, A_n)$

Why is it necessary?
# Renaming Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>depSSN</th>
<th>depname</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>934729837</td>
<td>Helen</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>934729837</td>
<td>Bob</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>934729837</td>
<td>Helen</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>934729837</td>
<td>Bob</td>
</tr>
</tbody>
</table>
Derived Operations
INTERSECTION

Notation: $R_1 \cap R_2$

- outputs all tuples in $R_1$ and $R_2$
- output schema: same as input
- can be expressed as: $R_1 - (R_1 - R_2)$

\[
\begin{array}{c|c}
A & B \\
\hline
a_1 & b_1 \\
\hline
a_2 & b_1 \\
\hline
a_2 & b_2 \\
\end{array}
\quad \cap \quad
\begin{array}{c|c}
A & B \\
\hline
a_1 & b_1 \\
\hline
a_3 & b_1 \\
\hline
a_4 & b_4 \\
\end{array}
= \begin{array}{c|c}
A & B \\
\hline
a_1 & b_1 \\
\end{array}
\]
JOIN (THETA JOIN)

Notation: $R_1 \bowtie_C R_2 = \sigma_C (R_1 \times R_2)$

• cross-product followed by a selection
• $C$ can be any boolean-valued condition
• might have less tuples than the cross-product!
**Theta Join Example**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>26</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dSSN</th>
<th>dname</th>
<th>dage</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Helen</td>
<td>23</td>
</tr>
<tr>
<td>934729837</td>
<td>Bob</td>
<td>28</td>
</tr>
</tbody>
</table>

\[ \text{Person} \bowtie_{\text{Person.age}>\text{Dependent.dage}} \text{Dependent} \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>dSSN</th>
<th>dname</th>
<th>dage</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>26</td>
<td>934729837</td>
<td>Helen</td>
<td>23</td>
</tr>
</tbody>
</table>
**EQUI-JOIN**

Notation: $R_1 \bowtie_C R_2$

- special case of join where the condition C contains only equalities!
- output schema: similar to cross-product, but only one copy for the fields in the equality

**Example** for $R(A, B), S(C, D)$

- $R \bowtie_{B=C} S$
- output schema: $T(A, B, D)$
**NATURAL JOIN**

Notation: $R_1 \bowtie R_2$

- equi-join on all the common fields

**Person**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>26</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>22</td>
</tr>
</tbody>
</table>

**Dependent**

<table>
<thead>
<tr>
<th>SSN</th>
<th>dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Helen</td>
</tr>
<tr>
<td>934729837</td>
<td>Bob</td>
</tr>
</tbody>
</table>

\[ \text{Person} \bowtie \text{Dependent} \]
Natural Join $R \bowtie S$

- **Input schema:** $R(A, B, C, D), S(A, C, E)$
  - Output schema?
- **Input schema:** $R(A, B, C), S(D, E)$
  - Output schema?
- **Input schema:** $R(A, B, C), S(A, B, C)$
  - Output schema?
**Semi-Join**

Notation: $R_1 \bowtie R_2$

- natural join followed by projection on the attributes of $R_1$

**Example:**

- $R(A, B, C), S(B, D)$
- $R \bowtie S = \pi_{A,B,C} (R \bowtie S)$
- output schema?
DIVISION

Notation: $R_1 / R_2$

• suppose $R_1 (A, B)$ and $R_2 (B)$
• the output contains all A-tuples such that for every B-tuple in $R_2$, there exists an (A,B) tuple in $R_1$
• output schema: $R(A)$
**Division Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
</tr>
<tr>
<td>a₁</td>
<td>b₂</td>
</tr>
<tr>
<td>a₁</td>
<td>b₃</td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>b₂</td>
</tr>
<tr>
<td>b₃</td>
</tr>
<tr>
<td>b₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>b₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/B₁</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/B₂</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td></td>
</tr>
<tr>
<td>a₂</td>
<td></td>
</tr>
</tbody>
</table>
**COMBINING RA OPERATORS**

- We can build more complex queries by combining RA operators together
  
  e.g. standard algebra: $(x + 1) * y - z^2$

- There are 3 different notations:
  - sequence of assignment statements
  - expressions with operators
  - expression trees
**Combining RA Operators**

**Input schema:** $R(B, C), S(A, B)$

- expressions with operators
  
  \[ \pi_A(\sigma_{C=1}(R) \bowtie S) \]

- sequence of assignment statements
  
  \[
  
  R' = \sigma_{C=1}(R) \\
  R'' = R' \bowtie S \\
  R''' = \pi_A(R'') 
  
  \]

- expression trees
EXPRESSIVE POWER

• RA cannot express transitive closure!

Edges

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Transitive closure computes all pairs of nodes connected by a directed path
EXAMPLES

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

Q1: What are the names of the sailors who have reserved boat #100?

Q2: What are the names of the sailors who have reserved a red boat?
EXAMPLES

Sailors \((\text{sid}, \text{sname}, \text{rating}, \text{age})\)
Reserves \((\text{sid}, \text{bid}, \text{day})\)
Boats \((\text{bid}, \text{bname}, \text{color})\)

Q3: Find the names of the sailors who have reserved in a green or red boat

Q4: Find the names of the sailors who have reserved in a green and a red boat
EXAMPLES

Sailors \((\text{sid}, \text{sname}, \text{rating}, \text{age})\)
Reserves \((\text{sid}, \text{bid}, \text{day})\)
Boats \((\text{bid}, \text{bname}, \text{color})\)

**Q5:** Find the names of the sailors who have reserved all ‘470’ boats
MORE EXAMPLES

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, country)
Person (ssn, name, phone, city)

Q6: Find the phone numbers of people who bought iPads from Fred (the salesman)

Q7: Find the names of people who bought products from the USA
More Examples

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, country)
Person (ssn, name, phone, city)

Q8: Find the names of people who bought products from the USA, but not from Greece

Q9: Find the names of people who bought products from the USA, and live in Madison
RECAP

• Relational Algebra
  – query language for relations

• Basic Operations
  – selection, projection
  – difference, union
  – cross-product, renaming

• Derived Operations
  – join, natural join, equi-join, division, etc