Relational Query Languages

- Languages for describing queries on a relational database
- **Structured Query Language (SQL)**
  - Predominant application-level query language
  - Declarative
- **Relational Algebra**
  - Intermediate language used within DBMS
  - Procedural
What is an Algebra?

- A language based on operators and a domain of values
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations (and the operators are as described later), we get the relational algebra
- We refer to the expression as a query and the value produced as the query result

Relational Algebra

- **Domain**: set of relations
- **Basic operators**: select, project, union, set difference, Cartesian product
- **Derived operators**: set intersection, division, join

**Procedural**
- Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression
The Role of Relational Algebra in a DBMS

Select Operator

- Produce table containing subset of rows of argument table satisfying condition

\[ \sigma_{\text{condition}}(\text{relation}) \]

- Example:

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John</td>
<td>123 Main</td>
<td>stamps</td>
</tr>
<tr>
<td>1123</td>
<td>John</td>
<td>123 Main</td>
<td>coins</td>
</tr>
<tr>
<td>5556</td>
<td>Mary</td>
<td>7 Lake Dr</td>
<td>hiking</td>
</tr>
<tr>
<td>9876</td>
<td>Bart</td>
<td>5 Pine St</td>
<td>stamps</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{Hobby='stamps'}}(\text{Person}) \]
Selection Condition

- Operators: <, ≤, ≥, >, =, ≠
- Simple selection condition:
  - `<attribute> operator <constant>`
  - `<attribute> operator <attribute>`
  - `<condition> AND <condition>`
  - `<condition> OR <condition>`
  - NOT <condition>

Selection Condition - Examples

- \(\sigma\) Id>3000 OR Hobby='hiking' (Person)
- \(\sigma\) Id>3000 AND Id<3999 (Person)
- \(\sigma\) NOT(Hobby='hiking') (Person)
- \(\sigma\) Hobby='hiking' (Person)
Project Operator

- Produces table containing subset of columns of argument table

\[ \pi_{\text{attribute list}}(\text{relation}) \]

- Example:

<table>
<thead>
<tr>
<th>Person</th>
<th>[ \pi_{\text{Name,Address}}(\text{Person}) ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>1123</td>
<td>John</td>
</tr>
<tr>
<td>1123</td>
<td>John</td>
</tr>
<tr>
<td>5556</td>
<td>Mary</td>
</tr>
<tr>
<td>9876</td>
<td>Bart</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>stamps</td>
</tr>
<tr>
<td>John</td>
<td>coins</td>
</tr>
<tr>
<td>Mary</td>
<td>hiking</td>
</tr>
<tr>
<td>Bart</td>
<td>stamps</td>
</tr>
</tbody>
</table>

- Result is a table (no duplicates); can have fewer tuples than the original
Expressions

- $\pi_{Id, Name} (\sigma_{\text{Hobby}='stamps' \lor \text{Hobby}='coins'} (\text{Person}))$

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John</td>
<td>123 Main</td>
<td>stamps</td>
</tr>
<tr>
<td>1123</td>
<td>John</td>
<td>123 Main</td>
<td>coins</td>
</tr>
<tr>
<td>5556</td>
<td>Mary</td>
<td>7 Lake Dr</td>
<td>hiking</td>
</tr>
<tr>
<td>9876</td>
<td>Bart</td>
<td>5 Pine St</td>
<td>stamps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>John</td>
</tr>
<tr>
<td>9876</td>
<td>Bart</td>
</tr>
</tbody>
</table>

Result

Set Operators

- Relation is a set of tuples, so set operations should apply: $\cap$, $\cup$, $-$ (set difference)
- Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- Hence, scope of set operations limited to union compatible relations
Union Compatible Relations

- Two relations are **union compatible** if
  - Both have the same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using **union, intersection, and set difference**

Example

Tables:
- Person (SSN, Name, Address, Hobby)
- Professor (Id, Name, Office, Phone)
are **not** union compatible.

But
- $\pi_{Name}(Person)$ and $\pi_{Name}(Professor)$
  are union compatible so
- $\pi_{Name}(Person) - \pi_{Name}(Professor)$
  makes sense.
Cartesian Product

- If R and S are two relations, $R \times S$ is the set of all concatenated tuples $<x,y>$, where $x$ is a tuple in $R$ and $y$ is a tuple in $S$
- $R$ and $S$ need not be union compatible

$R \times S$ is expensive to compute:
- Factor of two in the size of each row
- Quadratic in the number of rows

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x2</td>
<td>y1</td>
<td>y2</td>
</tr>
<tr>
<td>x3</td>
<td>x4</td>
<td>y3</td>
<td>y4</td>
</tr>
</tbody>
</table>

$R$  

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x2</td>
<td>y1</td>
<td>y2</td>
</tr>
<tr>
<td>x3</td>
<td>x4</td>
<td>y3</td>
<td>y4</td>
</tr>
</tbody>
</table>

$S$  

$R \times S$

Renaming

- Result of expression evaluation is a relation
- Attributes of relation must have distinct names. This is not guaranteed with Cartesian product
  - e.g., suppose in previous example $a$ and $c$ have the same name
- Renaming operator tidies this up. To assign the names $A_1, A_2, \ldots, A_n$ to the attributes of the $n$-column relation produced by expression $expr$ use:
  \[ expr [A_1, A_2, \ldots, A_n] \]
Example

Transcript (StudId, CrsCode, Semester, Grade)
Teaching (ProfId, CrsCode, Semester)

\[
\pi_{\text{StudId, CrsCode}}(\text{Transcript}) \left[ \text{StudId, CrsCode1} \right] \\
\times \pi_{\text{ProfId, CrsCode}}(\text{Teaching}) \left[ \text{ProfId, CrsCode2} \right]
\]

This is a relation with 4 attributes:
StudId, CrsCode1, ProfId, CrsCode2

Derived Operation: Join

- A (general or theta) join of R and S is the expression
  \[
  R \bowtie_{\text{join-condition}} S
  \]
  where join-condition is a conjunction of terms:
  \[A_i \text{ oper } B_i\]
  in which \(A_i\) is an attribute of \(R\); \(B_i\) is an attribute of \(S\); and
  oper is one of \(=, <, >, \geq, \leq\).

- The meaning is:
  \[
  \sigma_{\text{join-condition}'} (R \times S)
  \]
  where join-condition and join-condition' are the same, except for possible renaming of attributes
Join and Renaming

- **Problem:**
  - R and S might have attributes with the same name – in which case the Cartesian product is not defined

- **Solutions:**
  - Rename attributes prior to forming the product and use new names in join-condition.
  - Qualify common attribute names with relation names (thereby disambiguating the names).
    - e.g., Transcript.CrsCode or Teaching.CrsCode
  - This solution is nice, but doesn’t always work: consider
    - R \join \text{join-condition} R
    - In R.A, how do we know which R is meant?

Theta Join – Example

- Output the names of all employees that earn more than their managers.

\[ ?_{\text{Employee.Name}} (\text{Employee} \bowtie \text{Manager}) \]

- The join yields a table with attributes:
  - Employee.Name, Employee.Id, Employee.Salary, Mngrid,
  - Manager.Name, Manager.Id, Manager.Salary
Equijoin Join - Example

**Equijoin**: Join condition is a conjunction of equalities.

\[ \pi_{\text{Name}, \text{CrsCode}}(\text{Student} \bowtie_{\text{Id} = \text{StudId}} \sigma_{\text{Grade} = 'A'}(\text{Transcript})) \]

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Addr</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>John</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>222</td>
<td>Mary</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>333</td>
<td>Bill</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>444</td>
<td>Joe</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudId</th>
<th>CrsCode</th>
<th>Sem</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>CSE305</td>
<td>S00</td>
<td>B</td>
</tr>
<tr>
<td>222</td>
<td>CSE306</td>
<td>S99</td>
<td>A</td>
</tr>
<tr>
<td>333</td>
<td>CSE304</td>
<td>F99</td>
<td>A</td>
</tr>
</tbody>
</table>

The equijoin is used very frequently since it combines related data in different relations.

Natural Join

**Special case of equijoin:**
- join condition equates all and only those attributes with the same name (condition doesn’t have to be explicitly stated)
- duplicate columns eliminated from the result

\[ \text{Transcript} \bowtie \text{Teaching} = \]

\[ \overline{\text{StudId, Transcript.CrsCode, Transcript.Sem, Grade, ProfId}} \\
\{ \text{Transcript} \bowtie \sigma_{\text{CrsCode} = \text{CrsCode} \text{ AND } \text{Sem} = \text{Sem}}(\text{Teaching}) \} \\
\{ \text{StudId, CrsCode, Sem, Grade, ProfId} \} \]
Natural Join (cont’d)

More generally:

\[ R \bowtie S = \pi_{\text{attr-list}} (\sigma_{\text{join-cond}} (R \times S) ) \]

where

1. \text{attr-list} = \text{attributes} (R) \cup \text{attributes} (S)
   (duplicates are eliminated) and
2. \text{join-cond} has the form:
   \[ A_1 = A_1 \text{ AND } \ldots \text{ AND } A_n = A_n \]
   where
   \[ \{ A_1 \ldots A_n \} = \text{attributes}(R) \cap \text{attributes}(S) \]

Natural Join Example

List all Ids of students who took at least two different courses:

\[ \pi_{\text{StudId}} (\sigma_{\text{CrsCode} \neq \text{CrsCode2}} (\sigma_{\text{Transcript} \bowtie \text{Transcript}} [\text{StudId, CrsCode2, Sem2, Grade2}] )) \]

We don’t want to join on \text{CrsCode}, \text{Sem}, and \text{Grade} attributes, hence renaming!
Outer Join

- Three types
  - Left outer join / Right outer join / Full outer join

- Given two relations $r$ and $s$, the tuples in $r \bowtie_{\text{cond}} s$ consist of three categories
  1. The tuples that appear in the regular join of $r$ and $s$, $r \bowtie_{\text{cond}} s$
  2. The tuples of $r$ that do not join with any tuple in $s$
  3. The tuples of $s$ that do not join with any tuple in $r$

- For left outer join, $1 \cup 2$
- For right outer join, $1 \cup 3$
- For full outer join, $1 \cup 2 \cup 3$

Division

- Goal: Produce the tuples in one relation, $r$, that match all tuples in another relation, $s$
  - $r (A_1, \ldots, A_n, B_1, \ldots, B_m)$
  - $s (B_1, \ldots, B_m)$
  - $r/s$, with attributes $A_1, \ldots, A_n$, is the set of all tuples $<a>$ such that for every tuple $<b>$ in $s$, $<a, b>$ is in $r$

- Can be expressed in terms of projection, set difference, and cross-product
Division (cont’d)

Goal: Produce the tuples in one relation, \( r \), that match all tuples in another relation, \( s \)

- \( r (A_1, \ldots, A_n, B_1, \ldots, B_m) \)
- \( s (B_1, \ldots, B_m) \)
- \( r/s \), with attributes \( A_1, \ldots, A_n \), is the set of all tuples \( <a> \) such that for every tuple \( <b> \) in \( s \), \( <a,b> \) is in \( r \)

Can be expressed in terms of projection, set difference, and cross-product

\[
T_1 = \pi_A (R) \times S \\
T_2 = \pi_A (T_1 - R) \\
T_3 = \pi_A (R) - T_2
\]

Division
Division - Example

- List the Ids of students who have passed *all* courses that were taught in spring 2006

- *Numerator:*
  - *StudId* and *CrsCode* for every course passed by *every* student:
    \[ \pi_{\text{StudId}, \text{CrsCode}}(\sigma_{\text{Grade} \neq 'F'}(\text{Transcript})) \]

- *Denominator:*
  - *CrsCode* of *all* courses taught in spring 2006
    \[ \pi_{\text{CrsCode}}(\sigma_{\text{Semester}='S2006'}(\text{Teaching})) \]

- Result is *numerator/denominator*