Course Notes on Relational Algebra

Relational Algebra: Summary

- Operators
  - Selection
  - Projection
  - Union, Intersection, Difference
  - Cartesian Product
  - Join
  - Division
- Equivalences
- Outer Join, Outer Union
- Transitive Closure

What is the Relational Algebra?

- Relational algebra =
  - a collection of operations each acting on one or two relations and producing one relation as result, and
  - a language for combining those operations
- The algebra has played a central role in the relational model: algebraic operations characterize high-level set-at-a-time access
- The algebra in practice
  - it was never a real user language (calculus-based languages and SQL are simpler)
  - its semantics is clear and a de facto standard
  - a precise syntax for the algebra is more complicated than its semantics

Relational Schema for the Company Example

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>FName</th>
<th>LName</th>
<th>BDate</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>SuperSSN</th>
<th>DNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department</td>
<td>SSN</td>
<td>DName</td>
<td>DMgr</td>
<td>MgrStartDate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeptLocations</td>
<td>SSN</td>
<td>DNumber</td>
<td>DLocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>SSN</td>
<td>PName</td>
<td>PLocation</td>
<td>DNumber</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WorkOn</td>
<td>SSN</td>
<td>PNo</td>
<td>Hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependent</td>
<td>SSN</td>
<td>DependantName</td>
<td>Sex</td>
<td>BDate</td>
<td>Relationship</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Selection (or Restriction)

- Select tuples satisfying a condition

\[ \sigma_{\text{condition}}(R) = \{ r \in R \mid \text{condition}(r) \} \]

- Example: \( \sigma_{\text{DNo}=5}(\text{Employee}) \)

### Possible Forms of Conditions in Selection

- **Simple Condition** = \( \{ \text{attribute} \mid \text{comparison} \mid \text{attribute} \} \)
- \( \{ \text{attribute} \mid \text{comparison} \mid \text{constant} \} \)

- Comparisons: =, \( \neq \), <, \( \leq \), >, \( \geq \)

- Condition: combination of simple conditions with AND, OR, NOT

- Simple conditions are the most frequent
Projection

- Retain a subset of the attributes (columns) of a relation

\( \pi_{\text{attributes}}(\text{relation}) \)

- Example: \( \pi_{\text{LName}, \text{FName}, \text{Salary}}(\text{Employee}) \)

<table>
<thead>
<tr>
<th>LName</th>
<th>FName</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>John</td>
<td>30000</td>
</tr>
<tr>
<td>Wong</td>
<td>Franklin</td>
<td>40000</td>
</tr>
<tr>
<td>Zelaya</td>
<td>Alicia</td>
<td>25000</td>
</tr>
<tr>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td>Narayan</td>
<td>Ramesh</td>
<td>38000</td>
</tr>
<tr>
<td>English</td>
<td>Joyce</td>
<td>25000</td>
</tr>
<tr>
<td>Jabbar</td>
<td>Ahmad</td>
<td>25000</td>
</tr>
<tr>
<td>Borg</td>
<td>James</td>
<td>55000</td>
</tr>
</tbody>
</table>

- Intuition: projection is a "vertical" slice of relation Employee

"Duplicate Removal"

- The result of a projection is a relation \( \Rightarrow \) projection involves duplicate removal

- Example: \( \pi_{\text{DNo}}(\text{Employee}) \)

<table>
<thead>
<tr>
<th>DNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

- What is wrong with duplicate tuples?
  - "If something is true, then saying it twice does not make it more true" (Codd)
    Even if repetition is a proven pedagogical technique :-)
  - Identical things are indistinguishable, there is no need to represent them twice
  - "Objects in the real world have only one thing in common: they are all different" (anonymous, A. Taivalsaari, JOOP, Nov. 1997)
  - Distinct things should have some value to distinguish them
  - It is difficult to manipulate "duplicate tuples" unless for counting, averaging, etc
  - The obvious modeling of identical objects is as a common description plus the number of copies of the object

- Duplicate removal (from a multiset to a set) can be expensive; it can be done with
  - nested loops
  - sort/merge
  - hashing

A Precise Definition of the Relational Algebra

- Algebraic operations:
  - operate on one or two relations and produce a relation as result
  - the result relation has no name
  - rules are needed to specify the attribute names in the result of algebraic operations with two operands

- For a precise syntax and semantics of the algebra, see *A Precise Definition of Basic Relational Notions and the Relational Algebra*, A. Pirotte, ACM SIGMOD Record, 13-1, 1982, pp. 30-45.
Combining Algebraic Operations

List the name and salary of employees in department 5

(1) Nested form: $\pi_{\text{FirstName, LastName, Salary}}(\sigma_{\text{DNo}=5}(\text{Employee}))$

(2) Sequential form: $\text{Temp} \leftarrow \sigma_{\text{DNo}=5}(\text{Employee})$

<table>
<thead>
<tr>
<th>SSN</th>
<th>FirstName</th>
<th>Initial</th>
<th>LastName</th>
<th>Date</th>
<th>Sex</th>
<th>Salary</th>
<th>SuperSSN</th>
<th>DNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>09-Jan-55</td>
<td>M</td>
<td>30000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>333445555</td>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>08-Dec-45</td>
<td>M</td>
<td>40000</td>
<td>888665555</td>
<td>5</td>
</tr>
<tr>
<td>666884444</td>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>15-Sep-52</td>
<td>M</td>
<td>40000</td>
<td>888665555</td>
<td>5</td>
</tr>
<tr>
<td>453453453</td>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>31-Jul-62</td>
<td>F</td>
<td>25000</td>
<td>333445555</td>
<td></td>
</tr>
</tbody>
</table>

R(FirstName, LastName, Salary) ← $\pi_{\text{FirstName, LastName, Salary}}(\text{Temp})$

Nesting or Sequencing Operations

- Several relational algebra operations may be needed to express a given request:
  - by nesting several algebraic operations within a single relational algebra expression
  - by applying operations one at a time in a sequence of steps and creating named intermediate relations by assignment operations
  - the correspondence between nested form and sequential form is immediate

- Nesting and closure
  - the result of an algebraic operation is a relation
  - algebraic operations can be nested like functions
  - closure is essential for the full power of the algebra

Set-Theoretical Operations

- Standard operations on sets are automatically applicable to relations
- Union compatibility
  - relations have to be defined on the same domains
  - type compatibility would be more adequate
  - more precise definition: two relations $R$ and $S$ are union-compatible if there is a one-to-one correspondence between attributes of $R$ and attributes of $S$ such that corresponding attributes are associated with the same domain
  - A mechanism for defining the attribute names of the result is needed

- A relation is a set of tuples
- Union, intersection, and difference on union-compatible relations $R$ and $S$ have their usual meaning

Nesting

- nesting = classical functional composition:
  - $Y_5 \rightarrow f_5(f_1(X_1, X_2), f_2(X_3))$, is equivalent to the sequence
    - $Y_1 \rightarrow f_1(X_1, X_2)$
    - $Y_2 \rightarrow f_2(X_3)$
    - $Y_4 \rightarrow f_4(Y_1, X_4, Y_2)$

Closure

- closure = make nesting freely usable for combining operations, i.e., the result of every algebraic operation is a relation and it can be used as operand of another algebraic operation
- closure can be violated by
  - the definition of language structure (SQL does)
  - operations whose result is not a relation (e.g., “relations” with duplicate tuples)
- What is really needed is “compositionality”
  - the result of a query can be used as argument for another query
  - this is a version of what is called orthogonality (i.e., the generality of combining pieces of the definition of a language)
Union, Intersection, Difference

<table>
<thead>
<tr>
<th>Student</th>
<th>Prof</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>LN</td>
</tr>
<tr>
<td>Susan</td>
<td>Yao</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Shah</td>
</tr>
<tr>
<td>Barbara</td>
<td>Jones</td>
</tr>
<tr>
<td>Amy</td>
<td>Ford</td>
</tr>
<tr>
<td>Jimmy</td>
<td>Wang</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student ∪ Prof</th>
<th>Student ∩ Prof</th>
<th>Student − Prof</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>LN</td>
<td>FN</td>
</tr>
<tr>
<td>Susan</td>
<td>Yao</td>
<td>Susan</td>
</tr>
</tbody>
</table>

- Rules must be stated to specify the attribute names of the result; for example
  - the attributes of the first operand
  - the attributes of the second operand
  - explicitly-specified attributes, e.g., R₁(FN, LN) ← Student ∪ Prof

- The Cartesian product associates every tuple of the first relation with every tuple of the second one
- The result relation has all the attributes of the operand relations: if the attributes of the operand relations are not all distinct, some of them must be renamed
- The relational algebra thus needs a renaming operation, with as a possible syntax:
  - rename (relation name. (oldname → newname, ... ) )
  - in the example: rename(DeptLocations,(DNumber→DNo))
Semantics of Cartesian Product

- The Cartesian product associates every tuple of one relation with every tuple of the other (this is not a very useful operation)
- The following operation is more useful

$$\pi_{D\text{Number}, D\text{Name}, Mgr\text{SSN}, Mgr\text{StartDate}, D\text{Location}}(\sigma_{D\text{Number}=D\text{No}}(\text{Temp}))$$

<table>
<thead>
<tr>
<th>DNumber</th>
<th>DName</th>
<th>MgrSSN</th>
<th>MgrStartDate</th>
<th>DLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
<td>19-Jun-71</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
<td>01-Jan-85</td>
<td>Stafford</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Bellaire</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Sugarland</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
<td>01-Jan-85</td>
<td>Stafford</td>
</tr>
<tr>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
<td>19-Jun-71</td>
<td>Houston</td>
</tr>
</tbody>
</table>

- The result is a join of relations Department and DeptLocations, that associates each department with its locations

- The restriction retains only the tuples of the Cartesian product where the values of DNumber and of DNo are equal
- The projection suppresses the DNo attribute
- All these operations (Cartesian product, selection, and projection) can be expressed as a single algebra operation: the join
- The join is a fundamental operation for meaningfully creating bigger relations from smaller ones: but it is not always the inverse of projection (see later)

Join

- Join combines two relations into one on the basis of a condition
- $R \bowtie_{\text{condition}} S = \{\text{concat}(r, s) \mid r \in R \land s \in S \land \text{condition}(r, s)\}$
- Add the location information to the information about departments

```
DeptLocs ← Department \bowtie_{\text{DNumber}=\text{DNo}, \text{DLocation}}
```

<table>
<thead>
<tr>
<th>DNumber</th>
<th>DName</th>
<th>MgrSSN</th>
<th>MgrStartDate</th>
<th>DNo</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>5</td>
<td>Bellaire</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>5</td>
<td>Sugarland</td>
</tr>
<tr>
<td>5</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>5</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
<td>01-Jan-85</td>
<td>4</td>
<td>Stafford</td>
</tr>
<tr>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
<td>19-Jun-71</td>
<td>1</td>
<td>Houston</td>
</tr>
</tbody>
</table>

- Remember that, if R and S have an attribute in common, then some attributes of R and/or S have to be renamed
- The condition can be more general than a test for equality of 2 attribute values
  - simple condition = ($R$’s attribute) (comparison) ($S$’s attribute)
  - comparison: $\neq$, $\leq$, $>$, $\geq$
  - condition: combination of simple conditions with AND
- Two ways of defining the semantics of joins
  - declarative: create a tuple in the result for each pair of tuples in the relation arguments that satisfy the condition
  - operational (evaluation strategy): the condition is applied to every tuple of the Cartesian product; if it is satisfied, the tuple is kept in the answer
- As seen above, the join can be evaluated as a combination of Cartesian product, selection, and projection but this is not an efficient evaluation strategy; there are various strategies for implementing joins (see later): a basic method to implement the operation definition above is with nested loops:

```sql
R \bowtie_{\text{A}=\text{B}} S =
\begin{align*}
&\text{for each } r \in R \\
&\text{do for each } s \in S \\
&\text{do if } r.A = s.B \\
&\text{then concat}(r, s) \Rightarrow \text{result} \\
&\text{fi} \\
&\text{end} \\
&\text{end}
\end{align*}
```
Kinds of joins

- **theta join**: it is the general join (when all the attributes of the operand relations appear in the result of the join and the join condition is not simply a test of equality of 2 attributes)
- **equijoin**: when the join condition is a simple equality (e.g., A = B)
- **natural Join**: equijoin + only one of the attributes tested for equality is included in the result

\[ *_{R \times S}^* \text{A=B} \]

\[ *_{R \times S}^* \text{A=B} \text{ only one of A and B (say, A) is retained in the result} \]

\[ *_{R \times S}^* \text{A=B} \text{ only one of them is retained in the result (this version of join does not require the attributes of R and S to be all different)} \]

Natural Join

- Add the location information to the information about departments (with the DNumber attribute of DeptLocs renamed as DNo)

\[ \text{DeptLocs} \leftarrow \text{Department} \times \text{DeptLocations} \]

<table>
<thead>
<tr>
<th>DNumber</th>
<th>DName</th>
<th>MgrSSN</th>
<th>MgrStartDate</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Bellaire</td>
</tr>
<tr>
<td>2</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Sugarland</td>
</tr>
<tr>
<td>3</td>
<td>Research</td>
<td>333445555</td>
<td>22-May-78</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
<td>01-Jan-85</td>
<td>Stafford</td>
</tr>
<tr>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
<td>19-Jun-71</td>
<td>Houston</td>
</tr>
</tbody>
</table>

Relative Order of Selection and Join

Find name and address of employees who work for the Research department

- Selection on an operand of the join, or selection “before” or “inside” join

\[ \text{ResDept} \leftarrow \sigma_{\text{DName}='Research'}(\text{Department}) \]
\[ \text{ResDeptEmps} \leftarrow \text{ResDept} \times \sigma_{\text{DNumber}=\text{DNo}}(\text{Employee}) \]
\[ \text{Result} \leftarrow \pi_{\text{LName, Address}}(\text{ResDeptEmps}) \]

- Selection on the result of the join, or selection “after” or “outside” join

\[ \text{DeptsEmps} \leftarrow \text{Department} \times \text{Employee} \]
\[ \text{ResDept} \leftarrow \sigma_{\text{DName}='Research'}(\text{Department}) \times \sigma_{\text{DNumber}=\text{DNo}}(\text{Employee}) \]
\[ \text{ResDeptEmps} \leftarrow \text{ResDept} \times \text{Employee} \]
\[ \text{Result} \leftarrow \pi_{\text{LName, Address}}(\text{ResDeptEmps}) \]
About query optimization

- The two formulations of the query above are equivalent (it should be clear that they produce the same result).
- The first one does the selection before the join (i.e., the result of the selection serves as operand of the join), while the second one evaluates the selection on the result of the join.
- Such equivalences are frequent in the relational algebra.
- If the algebraic formulation was taken as guidance for actual evaluation, then in general there would be differences in performance (in the example above, the first formulation would probably produce a more efficient execution, as the selection on DName in Department produces a small relation before the join is evaluated).
- Query optimizers of current relational technology are able to perform comparative evaluation of performance and select a good strategy.
- Query optimizers for SQL will not beat the best Cobol programmer, but
  - good programmers are scarce and differences in individual productivity are enormous compared to other human activities.
  - it is impossible, in practice and in theory, to similarly optimize the compilation of programs in imperative languages (e.g., Cobol or C).
  - the best strategy selected depends on the database populations; if populations change so much that the best evaluation strategy ceases to be the best.
    - Cobol programs have to be rewritten to adjust to the new situation and remain optimum.
    - SQL optimization is redone dynamically by the DBMS: they can be optimized without being rewritten.
- Demonstrates the clear superiority of nonprocedural approaches over imperative ones: this was a key factor in establishing the relational model.
- For actual evaluation, a useful heuristics is to perform selections before joins, because joins are expensive operations that should be evaluated on operands as small as possible.
- The user has to choose one of the two algebraic formulations (but the query optimizer may decide to evaluate the query with the other one).

An Example with Two Joins

For every project located in 'Brussels', list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

- The result of joining Project and Department is joined with Employee.

  \[
  \text{BrusselsProjs} \leftarrow \sigma_{\text{PLocation} = 'Brussels'}(\text{Project}) \\
  \text{ProjDept} \leftarrow \text{BrusselsProjs} \bowtie \text{Department} \\
  \text{ProjDeptMgr} \leftarrow \text{ProjDept} \bowtie \text{Employee} \\
  \text{Result} \leftarrow \pi_{\text{PNumber}, \text{DNum}, \text{LName}, \text{Address}, \text{BDate}}(\text{ProjDeptMgr})
  \]

- The result of joining Department and Employee is joined with Project.

  \[
  \text{BrusselsProjs} \leftarrow \sigma_{\text{PLocation} = 'Brussels'}(\text{Project}) \\
  \text{DeptMgr} \leftarrow \text{Department} \bowtie \text{Employee} \\
  \text{ProjDeptMgr} \leftarrow \text{BrusselsProjs} \bowtie \text{DeptMgr} \\
  \text{Result} \leftarrow \pi_{\text{PNumber}, \text{DNum}, \text{LName}, \text{Address}, \text{BDate}}(\text{ProjDeptMgr})
  \]

- The query requires two binary joins and formulations differ in the ordering of those joins.
- For actual evaluation, the query optimizer will choose the most efficient order (in this case, most probably the second one).
- A third formulation would express a product of Employee and Project, and would join the result with Department.
Join or Intersection

List the names of managers who have at least one dependent

\[
\text{EmpDep} \leftarrow \text{Employee} \bowtie_{\text{SSN} = \text{ESSN}} \text{Dependent}
\]
\[
\text{MgrDep} \leftarrow \text{EmpDep} \bowtie_{\text{SSN} = \text{MgrSSN}} \text{Department}
\]
\[
\text{Result} \leftarrow \pi_{\text{LName}}(\text{MgrDep})
\]

- Projections can be “moved down” to make join arguments smaller
- Joins of one-attribute relations are intersections
- For actual evaluation, the query optimizer chooses the location of projections

\[
\begin{align*}
\text{Mgs}(\text{SSN}) & \leftarrow \pi_{\text{MgrSSN}}(\text{Department}) \\
\text{EmpsWithDep}(\text{SSN}) & \leftarrow \pi_{\text{ESSN}}(\text{Dependent}) \\
\text{MgrsWithDep} & \leftarrow \text{Mgs} \bowtie \text{EmpsWithDep} \\
\text{Result} & \leftarrow \pi_{\text{LName}}(\text{MgrsWithDep} \bowtie \text{Employee})
\end{align*}
\]

- The join \text{Employee} \bowtie_{\text{SSN} = \text{ESSN}} \text{Dependent} illustrates the loss of information in a join (see later, outer join)
- The join \text{MgrsWithDep} \bowtie \text{Employee} is sometimes called a semi-join, it is similar to a selection: it selects the tuples of \text{Employee} whose SSN appears in the one-attribute relation \text{MgrsWithDeps}.

Difference

List the names of employees who have no dependent

\[
\begin{align*}
\text{AllEmps} & \leftarrow \pi_{\text{SSN}}(\text{Employee}) \\
\text{EmpsWithDeps}(\text{SSN}) & \leftarrow \pi_{\text{ESSN}}(\text{Dependent}) \\
\text{EmpsWithoutDeps} & \leftarrow \text{AllEmps} \setminus \text{EmpsWithDeps} \\
\text{Result} & \leftarrow \pi_{\text{LName}}(\text{EmpsWithoutDeps} \bowtie \text{Employee})
\end{align*}
\]

- The join \text{Employee} \bowtie_{\text{SSN} = \text{ESSN}} \text{Dependent} illustrates the loss of information in a join (outer join)
- The join \text{EmpsWithoutDeps} \bowtie \text{Employee} is sometimes called a semi-join, it is similar to a selection: it selects the tuples of \text{Employee} whose SSN appears in the one-attribute relation \text{EmpsWithoutDeps}.

Union

List of project numbers for projects that involve an employee whose last name is ‘Smith’ as a worker or as a manager of the department that controls the project

\[
\begin{align*}
\text{Smiths}(\text{ESSN}) & \leftarrow \pi_{\text{SSN}}(\pi_{\text{LName} = \text{Smith}}(\text{Employee})) \\
\text{SmithWorkerProjs} & \leftarrow \pi_{\text{PNo}}(\text{WorksOn} \bowtie \text{Smiths}) \\
\text{Mgrs} & \leftarrow \pi_{\text{LName}, \text{DNumber}}(\text{Employee} \bowtie_{\text{ESSN} = \text{SSN}} \text{Department}) \\
\text{SmithMgrs} & \leftarrow \pi_{\text{LName} = \text{Smith}}(\text{Mgrs}) \\
\text{SmithManagedDepts}(\text{DNum}) & \leftarrow \pi_{\text{DNumber}}(\text{SmithMgrs}) \\
\text{SmithMgrProjs}(\text{PNo}) & \leftarrow \pi_{\text{PNumber}}(\text{SmithManagedDepts} \bowtie \text{Project}) \\
\text{Result} & \leftarrow \text{SmithWorkerProjs} \cup \text{SmithMgrProjs}
\end{align*}
\]

This query really is the union of two simpler queries.
An Example with Division

- Retrieve the name of employees who work on all the projects that Smith works on:

  \[ \text{Smith} = \pi_{\text{Name}}(\text{Employee}) \]
  \[ \text{SmithPNos} = \pi_{\text{PNo}}(\text{WorksOn} \bowtie_{\text{ESSN} = \text{SSN}} \text{Smith}) \]
  \[ \text{SSNPNos} = \pi_{\text{PNo}, \text{ESSN}}(\text{WorksOn}) \]
  \[ \text{SSNS(SSN)} = \text{SSNPNos} \div \text{SmithPNos} \]
  \[ \text{Result} = \pi_{\text{FName}, \text{LName}}(\text{SSNS} \bowtie \text{Employee}) \]
### Semantics of Division

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>T = R ÷ S</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>a1</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>a2</td>
</tr>
<tr>
<td>a1</td>
<td>b3</td>
<td>a3</td>
</tr>
<tr>
<td>a1</td>
<td>b4</td>
<td>a4</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>a4</td>
<td>b3</td>
<td></td>
</tr>
</tbody>
</table>

- Partition relation $R$ according to the $A$ values; to each value $a_i$ is attached a set of $B$ values associated with that $a_i$ value in $R$.
- Include in $T$ each $a_i$ such that the set of $B$ values associated with $a_i$ contains $S$.

### Relational Completeness

- A language is “relational complete” if it has at least the power of the relational algebra.
- Relational completeness is the only widely-accepted measure of power (besides computational completeness).
- Domain calculus and tuple calculus have the same power of expression as the relational algebra.

### Equivalences (Theory)

- Not all operations of the algebra are independent.
- \{σ, π, ∪, −, ×\} is a complete set, i.e., it has all the expression power of the algebra.
- ∩, ∖, and ÷ can be derived from them.
- $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$.
- $R \setminus \text{condition} = \pi_{\text{condition}}(R \times S)$.
- $R \setminus \text{condition} \cdot S = \pi_{\text{attr}}(R \text{condition}(R \times S))$.
- $R \div S = T$ can be reexpressed with difference and Cartesian product.
• $T_1$ contains all the candidate $a$-values, the answer is a subset of $T_1$
• $T_1 \times S$ associates each $a$-value with all $b$-values of $S$
• if an $a$-value is in $T_2$, this means that an $a$-value is associated in $R$ with fewer $b$-values than the $b$-values with which it is associated in $T_1 \times S$: that $a$-value should not be part of the result

**Equivalences (Practical)**

1. $\sigma_{A \cap B}(R) = \sigma_A(\sigma_B(R))$
2. $\sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$
3. if $A \subseteq A_1$, then $\pi_A(R) = \pi_A(\pi_{A_1}(R))$
4. $\pi_A(\sigma_B(R)) = \sigma_B(\pi_A(R))$ if attributes in $c \subseteq A$ attributes in $A$
5. $\sigma_A(R \bowtie S) = \sigma_A(R) \bowtie S$ if attributes in $c \subseteq A$ attributes in $R$
6. $\pi_{A,B}(R \bowtie S) = \pi_A(R) \bowtie \pi_B(S)$ if $c$ involves only attributes in $A$ of $R$ and in $B$ of $S$
7. $\pi_{A,B}(R \bowtie S) = \pi_{A,B}(\pi_{A_1}(R) \bowtie \pi_{B_1}(S))$ if $c$ involves attributes in $A$, $A_1$ of $R$ and in $B$, $B_1$ of $S$
8. $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
9. $\sigma_A(R \bowtie S) = \sigma_A(R) \bowtie \sigma_B(S)$
10. $\pi_A(R \bowtie S) = \pi_A(R) \bowtie \pi_A(S)$

(1) Break conjunctive selection $\sigma_{A \cap B}(R) = \sigma_A(\sigma_B(R))$
(2) Commute selections $\sigma_A(\sigma_B(R)) = \sigma_B(\sigma_A(R))$

Used for
• applying the most selective join first for efficiency
• commute a simple selection with another operation (join, projection)

3. Sequence partial projections: if $A \subseteq A_1$, then $\pi_A(R) = \pi_A(\pi_{A_1}(R))$
4. Commute selection and projection:
• $\pi_A(\sigma_B(R)) = \sigma_B(\pi_A(R))$, if attributes in $c \subseteq A$ attributes in $A$
• $\sigma_A(\pi_A(R))$ can be evaluated as $\pi_A(\sigma_A(R))$ (but not the other way around)
5. Enter selection into join: if attributes in $c \subseteq A$ attributes in $R$, then $\sigma_A(R \bowtie S) = \sigma_A(R) \bowtie S$
6. Enter projection into join: if $c$ involves only attributes in $A$ of $R$ and in $B$ of $S$, then $\pi_{A,B}(R \bowtie S) = \pi_A(R) \bowtie \pi_B(S)$
7. Enter projection into join (general case):
• if $c$ involves attributes in $A$, $A_1$ of $R$ and in $B$, $B_1$ of $S$, then $\pi_{A,B}(R \bowtie S) = \pi_{A,B}(\pi_{A_1,A_2}(R) \bowtie \pi_{B_1,B_2}(S))$
• the attributes of $R$ (schema for $S$) comprise $A$, $A_1$, and $A_2$: $A$ is needed in the result, $A_1$ participates in the join but is not needed in the result; $A_2$ does not participate at all
(8) Associate, commute joins (also valid for set-theoretic operations)
\[(R \Join S) \bowtie T = R \Join (S \bowtie T)\]
\[R \Join S = S \Join R\]
Used for choosing the order of joins for efficiency

(9) Enter selection into union (also intersection, difference)
\[\sigma_c(R \cup S) = \sigma_c(R) \cup \sigma_c(S)\]

(10) Enter projection into union (also intersection, difference)
\[\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)\]

Motivation for Outer Joins: Ordinary Joins are often Lossy
- \[\pi_{FName,...,DNo}(\text{Employee } \text{SSN} = \text{MgrSSN} \text{ Department}) \subseteq \text{Employee}\]
- \[\pi_{FName,...,DNo}(\text{Employee } \text{SSN} = \text{MgrSSN} \text{ Department}) = \text{Employee}\]
- \[\pi_{FName,...,DNo}(\text{Department }) = \pi_{FName,...,DNo}(\text{Employee } \text{SSN} = \text{MgrSSN} \text{ Department})\]

- "Lossy" = information is lost in the result of the join (e.g. employees who are not department managers disappear in \(\text{Employee } \text{SSN} = \text{MgrSSN} \text{ Department}\))
- In \(R \Join S\), only tuples satisfying the join condition contribute to the result and information may be lost (projections of \(R \Join S\) on \(R\) or \(S\) may be smaller than \(R\) or \(S\))

Outer Joins
- **Outer Joins** preserve information from the operands (outer joins are “lossless”)
- **Left Outer Join** \(R \bowtie S\): retains all tuples of the left operand relation \(R\); if, for a tuple of \(R\), no matching tuple is found in \(S\), the attribute values corresponding to \(S\) in the result are set to null
- **Right Outer Join** \(S \bowtie R\): retains all tuples of \(S\)
- **Full Outer Join** \(\bowtie\): retains all tuples in both relations

Example of Outer Join
- Retrieve the name of all employees, plus the name of the departments that they manage (if any)

```
Temp ← \text{Employee } \text{SSN} = \text{MgrSSN} \text{ (Department)}
Result ← \pi_{FName,MInit,LName,DName}(Temp)
```

<table>
<thead>
<tr>
<th>FName</th>
<th>MInit</th>
<th>LName</th>
<th>DName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>null</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>Research</td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td>null</td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td>Administration</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>null</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>null</td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jabbar</td>
<td>null</td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Borg</td>
<td>Headquarters</td>
</tr>
</tbody>
</table>

Relational Algebra, October 7, 2008 – 25

Relational Algebra, October 7, 2008 – 26
Outer Union

- Union of tuples from two partially compatible relations (only some of their attributes are union compatible)
- Attributes that are not union compatible from either relation are kept in the result, and tuples with no values for these attributes are padded with null values
- Outer union of 
  \[
  \text{Student}(\text{Name, SSN, Department, Advisor})
  \]
  \[
  \text{Professor}(\text{Name, SSN, Department, Rank})
  \]
  is a relation \( R(\text{Name, SSN, Department, Advisor, Rank}) \) obtained from
  - \( \text{Student} \times \text{Professor} \cup \)
  - Professors that are not students (with null for Advisor) \( \cup \)
  - Students that are not professors (with null for Rank)
- All tuples of the operand relations appear as a subtuple of the result

Outer Union as ER Generalization

- Outer union corresponds to generalization in the ER model

Transitive Closure

- Natural and frequent operation for exploring nested structures (e.g., part-subpart composition)
- Natural in algebra style but not available
- Applies to a recursive relationship between tuples of the same relation, e.g., between employee and supervisor in relation Employee

Example

- Retrieve all employees supervised by James Borg
  - all employees directly supervised by James Borg
  - all employees directly supervised by the previous ones
  - . . .
- Although it is possible to specify each level in relational algebra, the number of levels is not known since it depends on the extension

- Remember that the relational algebra (nor TRC, DRC, SQL - see later) is not computational complete (\( \neq \) does not have the expressive power of algorithmic languages)
- Computational completeness = Church-Turing thesis
  - \( \Diamond \) thesis = any algorithm that you can think of can be formulated with any of the popular programming languages
  - \( \Diamond \) revised with Gödel theorem and undecidable problems
Recursive Formulations of Transitive Closure

- Prolog-style
  \[
  \text{Result}(s) \leftarrow \text{BorgSSN}(s)
  \]
  \[
  \text{Result}(s_1) \leftarrow \text{Supervision}(s_1, s_2) \text{ and } \text{Result}(s_2)
  \]

- Recursive equation
  \[
  \text{Result}([S, S]) \leftarrow \text{BorgSSN}(S) \cup \pi_{S_1}(\text{Supervision} \ast S_{S} = S \text{ Result})
  \]

- Alternative: usual algorithmic program

More about recursion ...

- A dictionary definition:
  - recursive: see “recursive”

- “Anything in computer science that is not recursive is no good” (Jim Gray?)

- Recursion = fundamental linguistic tool (like iteration) for expressing, in a well-defined finite way, patterns of action that repeat an unknown number of times (in natural language: “and so on”, “etc.”, “...”)

Incomplete Algebra Solution for Transitive Closure

BorgSSN \( \leftarrow \pi_{S_{1}}(\text{FName=James \& LName=Borg (Employee)}) \)

Supervision\( (S_{1}, S_{2}) \leftarrow \pi_{S_{1}S_{2}}(\text{Supervision (Employee)}) \)

Result1\( (S_{1}) \leftarrow \pi_{S_{1}S_{2}}(\text{Supervision} \ast S_{S_{2}=S_{1}, \text{BorgSSN}}) \)

Result2\( (S_{1}) \leftarrow \pi_{S_{1}S_{2}}(\text{Supervision} \ast S_{S_{2}=S_{1}, \text{Result1}}) \)

<table>
<thead>
<tr>
<th>BorgSSN</th>
<th>Supervision</th>
</tr>
</thead>
<tbody>
<tr>
<td>888665555</td>
<td>123456789</td>
</tr>
<tr>
<td>888665555</td>
<td>999887777</td>
</tr>
<tr>
<td>666844444</td>
<td>333445555</td>
</tr>
<tr>
<td>666844444</td>
<td>987987987</td>
</tr>
<tr>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

Incomplete Algebra Solution for Transitive Closure

<table>
<thead>
<tr>
<th>Result1</th>
<th>Result2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>SSN</td>
<td>SSN</td>
</tr>
<tr>
<td>333445555</td>
<td>123456789</td>
<td>333445555</td>
</tr>
<tr>
<td>987654321</td>
<td>999887777</td>
<td>897654321</td>
</tr>
<tr>
<td>666844444</td>
<td>333445555</td>
<td></td>
</tr>
<tr>
<td>666844444</td>
<td>987987987</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>supervised by</th>
<th>supervised by Borg’s subordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borg</td>
<td>453453453</td>
</tr>
</tbody>
</table>

Relational Algebra, October 7, 2008 – 29

Relational Algebra, October 7, 2008 – 30