Relational Query Languages

- Procedural vs non-procedural (declarative)
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Expressive power of a query language
  - What queries can be expressed in this language?
- Relational algebra:
  - Algebra of relations: set of operators that take relations as input and produce relations as output
  - Closed: the output of evaluating an expression in relational algebra can be used as input to another relational algebra expression
- Now: First introduction to operators of the relational algebra

Relational Algebra

- Procedural language
- Six basic operators:
  - select: s
  - project: π
  - union: ∪
  - set difference: −
  - Cartesian product: ×
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.
  - composable

Select Operation – Example

- Relation r

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- σ_{AB + D > 5}(r)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Select Operation

- Notation: σ_p(r)
- p is called the selection predicate
- Defined as:
  \[ σ_p(r) = \{ t \mid t \in r \land p(t) \} \]

Where p is a formula in propositional calculus consisting of terms connected by: ∨ (and), ∧ (or), ¬ (not)

Each term is one of:
- <attribute> op <attribute> or <constant>
- where op is one of: =, ≠, >, <, ≥, ≤

Example of selection:

\[ σ_{\text{dept name} = \text{Physics}}(\text{instructor}) \]
Project Operation – Example

- Relation \( r \):
  
  \[
  \begin{array}{ccc}
  A & B & C \\
  \alpha & 10 & 1 \\
  \alpha & 20 & 1 \\
  \beta & 30 & 1 \\
  \beta & 40 & 2 \\
  \end{array}
  \]

- \( \Pi_{A \cup C}(r) \):
  
  \[
  \begin{array}{cc}
  A & C \\
  \alpha & 1 \\
  \beta & 2 \\
  \end{array} = \begin{array}{cc}
  A & C \\
  \alpha & 1 \\
  \beta & 1 \\
  \end{array}
  \]

Project Operation

- Notation:
  
  \[ \Pi_{A_1, A_2, \ldots, A_k}(r) \]

  where \( A_1, A_2, \ldots, A_k \) are attribute names and \( r \) is a relation name.

- The result is defined as the relation of \( k \) columns obtained by erasing the columns that are not listed

- Duplicate rows removed from result, since relations are sets

- Let \( A \) be a subset of the attributes of relation \( r \) then:
  
  Example: To eliminate the dept_name attribute of instructor

  \[ \Pi_{ID, name, salary}(instructor) \]

Union Operation – Example

- Relations \( r, s \):
  
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \beta & 3 \\
  \end{array}, \quad \begin{array}{ccc}
  A & B \\
  \alpha & 2 \\
  \beta & 3 \\
  \end{array}
  \]

- \( r \cup s \):
  
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \beta & 3 \\
  \end{array}
  \]

Union Operation

- Notation:
  
  \( r \cup s \)

- Defined as:
  
  \[ r \cup s = \{ t \mid t \in r \lor t \in s \} \]

- For \( r \cup s \) to be valid.
  1. \( r, s \) must have the same arity (same number of attributes)
  2. The attribute domains must be union compatible (example: 2nd column of \( r \) deals with the same type of values as does the 2nd column of \( s \))

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

  \[
  \Pi_{course_id}(\sigma \text{semester="Fall" \land year=2009}(section)) \cup \\
  \Pi_{course_id}(\sigma \text{semester="Spring" \land year=2010}(section))
  \]

Set difference of two relations

- Relations \( r, s \):
  
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \end{array}, \quad \begin{array}{ccc}
  A & B \\
  \alpha & 2 \\
  \beta & 3 \\
  \end{array}
  \]

- \( r \setminus s \):
  
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 1 \\
  \beta & 1 \\
  \end{array}
  \]

Set Difference Operation

- Notation \( r \setminus s \)

- Defined as:
  
  \[ r \setminus s = \{ t \mid t \in r \land t \notin s \} \]

- Set differences must be taken between compatible relations.
  - \( r \) and \( s \) must have the same arity
  - Attribute domains of \( r \) and \( s \) must be compatible

- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

  \[
  \Pi_{course_id}(\sigma \text{semester="Fall" \land year=2009}(section)) \setminus \\
  \Pi_{course_id}(\sigma \text{semester="Spring" \land year=2010}(section))
  \]
Cartesian-Product Operation – Example

- Relations r, s:
  - r: A B C D E
    - a 1 α 10 a
    - b 2 α 10 a
    - α 1 β 10 a
    - β 2 α 10 b
    - β 2 β 20 b
  - s: A B C D E
    - a 1 α 10 a
    - b 2 β 20 b

- r x s:
  - A B C D E
    - a 1 α 10 a
    - b 2 β 20 b
    - α 1 β 10 a
    - β 2 α 10 b
    - β 2 β 20 b

Cartesian-Product Operation

- Notation r x s
- Defined as:
  \[ r \times s = \{ t, t' \mid t \in r \land t' \in s \} \]
  
- Assume that attributes of r(R) and s(S) are disjoint. (That is, \( R \cap S = \emptyset \)).
  
- If attributes of r(R) and s(S) are not disjoint, then renaming must be used.

Composition of Operations

- Can build expressions using multiple operations
- Example: \( a_{in}(r \times s) \)

- \( r \times s \): A B C D E
  - a 1 α 10 a
  - b 2 β 20 b
  - α 1 β 10 a
  - β 2 α 10 b
  - β 2 β 20 b

- \( a_{in}(r \times s) \): A B C D E
  - a 1 α 10 a
  - b 2 β 20 b
  - α 1 β 10 a
  - β 2 α 10 b
  - β 2 β 20 b

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:
  \[ \rho_{x}(r) \]
  returns the expression \( X \) under the name \( X \)
- If a relational algebra expression \( E \) has arity \( n \), then
  \[ \rho_{X}(r) = \{ t(X) \mid t \in r \} \]
  \[ \rho_{X}(A) = \{ t(X).A \mid t \in r \} \]

Example Query

- Find the largest salary in the university
  - Step 1: Find instructor salaries that are less than some other instructor salary (i.e. not maximum)
    - using a copy of instructor under a new name \( d \)
      \[ \pi_{\text{instructor.salary}, \text{course.id}}(\sigma_{\text{instructor.salary} < \text{d.salary}}(\text{instructor} \times \rho_{d}(\text{instructor}))) \]
  - Step 2: Find the largest salary
    \[ \pi_{\text{salary}}(\text{instructor}) \]
A basic expression in the relational algebra consists of either one of the following:

- A relation in the database
- A constant relation: e.g., \{(1), (2)\}

Let \(E_1\) and \(E_2\) be relational-algebra expressions; the following are all relational-algebra expressions:

- \(E_1 \cup E_2\)
- \(E_1 - E_2\)
- \(E_1 \times E_2\)
- \(\sigma_P(E_1)\), \(P\) is a predicate on attributes in \(E_1\)
- \(\Pi_{x}(E_1)\), \(x\) is the new name for the result of \(E_1\)

Let \(E\) be a relational algebra expression. We use \([E](I)\) to denote the evaluation of \(E\) over a database instance \(I\):

- For simplicity we will often drop \(I\) and \([\]\)
- The result of evaluating a simple relational algebra expression \(E\) over a database is defined as
  - Simple relation: \([R](I) = R(I)\)
  - Constant relation: \([C](I) = C\)

It is possible for tuples to have a null value, denoted by null, for some of their attributes:

- null signifies an unknown value or that a value does not exist.

Examples:

- We register a new employee Peter, but the salary for this employee has not yet been determined
  - Unknown value
- A government agency collects data about residents including their SSN. Some residents are not allowed to work and, thus, do not have an SSN
  - The attribute SSN is not applicable for such residents

Comparisons with null values return the special truth value: unknown

- If false was used instead of unknown, then not \((A < 5)\) would not be equivalent to \(A \geq 5\)

Three-valued logic using the truth value unknown:

- OR: (unknown or true) = true, (unknown or false) = unknown
- AND: (true and unknown) = unknown, (false and unknown) = false, (unknown and unknown) = unknown
- NOT: (not unknown) = unknown
- In SQL "\(P\) is unknown" evaluates to true if predicate \(P\) evaluates to unknown

Result of selection predicate is treated as false if it evaluates to unknown

The result of any arithmetic expression involving null is null.

Aggregate functions simply ignore null values (as in SQL)

For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)
Additional Operations

We define additional operations that do not add any expressive power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection Operation

- Notation: \( r \cap s \)
- Defined as:
  \[ r \cap s = \{ t \mid t \in r \land t \in s \} \]
- Assume:
  - \( r, s \) have the same arity
  - attributes of \( r \) and \( s \) are compatible
- Note: \( r \cap s = r - (r - s) \)
  - That is adding intersection to the language does not make it more expressive

Set-Intersection Operation – Example

Relation \( r, s \):

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline 1 & 2 \\
2 & 3 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
A & B \\
\hline 2 & 1 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
\hline
\end{array}
\]

\( r \cap s \):

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline 2 & 1 \\
\hline
\end{array}
\]

Natural-Join Operation

- Notation: \( r \bowtie s \)
- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively. Then, \( r \bowtie s \) is a relation on schema \( R \land S \) obtained as follows:
  - Consider each pair of tuples \( t_r \) from \( r \) and \( t_s \) from \( s \).
  - If \( t_r \) and \( t_s \) have the same value on each of the attributes in \( R \cap S \), add a tuple \( t \) to the result, where
    - \( t \) has the same value as \( t_r \) on \( r \)
    - \( t \) has the same value as \( t_s \) on \( s \)
- Example:
  - \( R = (A, B, C, D) \)
  - \( S = (E, B, D) \)
  - Result schema = \( (A, B, C, D, E) \)
  - \( r \bowtie s \) is defined as:
    \[
    \Pi_{A,B,C,D,E}(\sigma_{B=B \land D=D}(r \times s))
    \]

Natural-Join Operation (cont.)

- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively. Then, \( r \bowtie s \) is defined as:
  \[
  X = R \cap S \\
  S' = S - R \\
  r \bowtie s = \pi_{R,S'}(\sigma_{X=s.X}(r \times s))
  \]

Natural Join Example

Relations \( r, s \):

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
A & B & C & D \\
\hline \alpha & \beta & \gamma & \delta \\
\hline \alpha & \beta & \gamma & \delta \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|}
\hline
B & D & E \\
\hline \alpha & \beta & \gamma \\
\hline \alpha & \beta & \gamma \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline \alpha & \beta & \gamma & \delta \\
\hline \alpha & \beta & \gamma & \delta \\
\hline
\end{array}
\]

\( r \bowtie s \):

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline \alpha & \beta & \gamma & \delta \\
\hline \alpha & \beta & \gamma & \delta \\
\hline
\end{array}
\]

\( r \bowtie s \):

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline \alpha & \beta & \gamma & \delta \\
\hline \alpha & \beta & \gamma & \delta \\
\hline
\end{array}
\]

\( r \bowtie s \):

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline \alpha & \beta & \gamma & \delta \\
\hline \alpha & \beta & \gamma & \delta \\
\hline
\end{array}
\]
Natural Join and Theta Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  \[ \pi_{\text{name, title}} (\sigma_{\text{dept_name = "Comp. Sci."}} (\text{instructor} \bowtie \text{teaches})) \]
- Natural join is associative
  \[ (\text{instructor} \bowtie \text{teaches}) \bowtie \text{course} \]
  is equivalent to
  \[ \text{instructor } \bowtie (\text{teaches} \bowtie \text{course}) \]
- Natural join is commutative (we ignore attribute order)
  \[ \text{instructor } \bowtie \text{teaches} \]
  is equivalent to
  \[ \text{teaches } \bowtie \text{instructor} \]
- The theta join operation \( r \bowtie_{\theta} s \) is defined as
  \[ r \bowtie_{\theta} s = \sigma_{\theta} (r \times s) \]

Assignment Operation

- The assignment operation \( (\rightarrow) \) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
  1. a series of assignments
  2. an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
  \[ E_1 \leftarrow \sigma_{\text{salary} > 40000} (\text{instructor}) \]
  \[ E_2 \leftarrow \sigma_{\text{salary} < 10000} (\text{instructor}) \]
  \[ E_3 \leftarrow E_1 \cup E_2 \]

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
- Uses null values:
  - null signifies that the value is unknown or does not exist
  - All comparisons involving null are roughly speaking false by definition.
  - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation instructor1

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
</tr>
</tbody>
</table>

- Relation teaches1

<table>
<thead>
<tr>
<th>ID</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>FIN-201</td>
</tr>
<tr>
<td>76766</td>
<td>BIO-101</td>
</tr>
</tbody>
</table>

Outer Join – Example

- Join

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
</tbody>
</table>

- Left Outer Join

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>null</td>
</tr>
</tbody>
</table>

Right Outer Join – Example

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
<tr>
<td>76766</td>
<td>null</td>
<td>null</td>
<td>BIO-101</td>
</tr>
</tbody>
</table>

Full Outer Join – Example

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>null</td>
</tr>
<tr>
<td>76766</td>
<td>null</td>
<td>null</td>
<td>BIO-101</td>
</tr>
</tbody>
</table>
Defining Outer Join using Join

- Outer join can be expressed using basic operations

\[ r \bowtie s = (r \bowtie s) \cup ((r - \Pi_R (r \bowtie s)) \times \{null, \ldots, null\}) \]
\[ r \bowtie s = (r \bowtie s) \cup ((null, \ldots, null) \times (s - \Pi_S (r \bowtie s))) \]
\[ r \bowtie s = (r \bowtie s) \cup ((null, \ldots, null) \times (s - \Pi_S (r \bowtie s))) \]

Division Operator

- Given relations \( r(R) \) and \( s(S) \), such that \( S \subset R \) and \( r \bowtie s \) is the largest relation \( r \bowtie (R - S) \) such that
- Alternatively, all tuples from \( r - (R - S) \) such that all their extensions on \( R \setminus S \) with tuples from \( s \) exist in \( R \)
- E.g., let \( r(ID, course\_id, \Pi_{dept\_name = \text{Biology}}(course) \) and \( s(course\_id) = \Pi_{dept\_name = \text{Biology}}(course) \) then \( r \bowtie s \) gives us students who have taken all courses in the Biology department
- Can write \( r \bowtie s \) as

\[ E_1 = \Pi_{R - S}(r) \]
\[ E_2 = \Pi_{R - S}(E_1 \times s - \Pi_{R - S}(r \bowtie s)) \]
\[ r \bowtie s = E_1 - E_2 \]

Division Operator Example

- Return the name of all persons that read all newspapers

<table>
<thead>
<tr>
<th>reads</th>
<th>newspaper</th>
<th>newspaper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>Wall Street</td>
<td>Wall Street</td>
</tr>
<tr>
<td>Bob</td>
<td>Times</td>
<td>Times</td>
</tr>
<tr>
<td>Alice</td>
<td>Times</td>
<td>Times</td>
</tr>
</tbody>
</table>

\[ B_1 = \Pi_{name}(\text{reads}) \]
\[ B_2 = \Pi_{name}(\Pi_{name \bowtie \text{newspaper}}\text{reads \bowtie \text{newspaper}}) \]
\[ \text{reads \bowtie \text{newspaper}} = B_1 - B_2 \]
\[ \text{reads \bowtie \text{newspaper}} = \{\{\text{Alice}\}\} \]

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

\[ \pi_{F_1, \ldots, F_n}(E) \]

- \( E \) is any relational-algebra expression
- Each of \( F_1, F_2, \ldots, F_n \) are arithmetic expressions and function calls involving constants and attributes in the schema of \( E \)
- Given relation \( \text{instructor}(ID, \text{name}, \text{dept\_name}, \text{salary}) \) where salary is annual salary, get the same information but with monthly salary

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>Biology</td>
<td>10000</td>
</tr>
</tbody>
</table>

- Adding functions increases expressive power
- In standard relational algebra there is no way to change attribute values

Aggregate Functions and Operations

- Aggregation function takes a set of values and returns a single value as a result.

- Aggregation operation in relational algebra

\[ G_1, G_2, \ldots, G_n \cdot G(F_1(A_1), F_2(A_2), \ldots, F_n(A_n))(E) \]

- \( E \) is any relational-algebra expression
- \( G_1, G_2, \ldots, G_n \) is a list of attributes on which to group (can be empty)
- Each \( F_i \) is an aggregate function
- Each \( A_i \) is an attribute name
- Note: Some books/articles use \( \Gamma \) instead of \( G \) (Calligraphic G)
Aggregate Operation – Example

- Relation \( r \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>7</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>10</td>
</tr>
</tbody>
</table>

- \( G_{\text{sum}(c)}(r) \):

\[
\text{sum}(c) = 27
\]

Aggregate Operation – Example

- Find the average salary in each department

\[
\text{dept\_name} \ G_{\text{avg(salary)}} \text{(instructor)}
\]

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7676</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>40583</td>
<td>Katz</td>
<td>Comp. Sci</td>
<td>75000</td>
</tr>
<tr>
<td>89101</td>
<td>Venkatesan</td>
<td>Comp. Sci</td>
<td>63000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci</td>
<td>92000</td>
</tr>
<tr>
<td>98343</td>
<td>Kim</td>
<td>Elec. Eng</td>
<td>80000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>76343</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>32343</td>
<td>PT Sold</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>38593</td>
<td>Califiri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>13133</td>
<td>Mozart</td>
<td>Music</td>
<td>48000</td>
</tr>
<tr>
<td>34099</td>
<td>Gage</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
</tbody>
</table>

Aggregate Functions (Cont.)

- What are the names for attributes in aggregation results?
  - Need some convention!
  - E.g., use the expression as a name \( \text{avg(salary)} \)
  - For convenience, we permit renaming as part of aggregate operation
    \[
    \text{dept\_name} \ G_{\text{avg(salary)}} \text{as avg\_sal (instructor)}
    \]

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating

- All these operations can be expressed using the assignment operator

- Example: Delete instructors with salary over $1,000,000

\[
R \leftarrow R - (\sigma_{\text{salary} > 1000000}(R))
\]

Restrictions for Modification

- Consider a modification where \( R=(A,B) \) and \( S=(C) \)

\[
R \leftarrow \sigma_{C > 5}(S)
\]

- This would change the schema of \( R \):
  - Should not be allowed

- Requirements for modifications:
  - The name \( R \) on the left-hand side of the assignment operator refers to an existing relation in the database schema
  - The expression on the right-hand side of the assignment operator should be union-compatible with \( R \)

Tuple Relational Calculus
**Predicate Calculus Formula**

1. Set of attributes and constants
2. Set of comparison operators: \( (\leq, \geq, <, >, \leq, \geq, \neq) \)
3. Set of logical connectives: and (\&), or (\lor), not (\neg)
4. Implication (\Rightarrow): if \( x \) is true, then \( y \) is true
   \[ x \Rightarrow y \]
5. Set of quantifiers:
   - \( \exists \) “there exists” a tuple \( r \) in relation \( r \) such that predicate \( Q(t) \) is true
   - \( \forall \) “for all” tuples \( t \) in relation \( r \)

**Example Queries**

- Find the names of all instructors whose department is in the Watson building
  \[ \{ t | t \in instructor \land t [dept_name] = \text{"Watson"} \} \]
- Find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both
  \[ \{ t | t \in section \land t [course_id] = s [course_id] \lor t [semester] = \text{"Fall"} \lor t [year] = 2009 \} \]

**Safety of Expressions**

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, \( \{ t \} \) results in an infinite relation if the domain of any attribute of relation \( r \) is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression \( \{ t | P(t) \} \) in the tuple relational calculus is safe if every component of \( t \) appears in one of the relations, tuples, or constants that appear in \( P \).
  - Note: this is more than just a syntax condition.
  - e.g., \( \{ t | t [A] = 5 \land \text{true} \} \) is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in \( P \).
Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - \( \forall s \in \text{student} \left( \exists i, d \left( i \in \text{course} \land \exists t \left( t \in \text{section} \land s \in \text{section} \land i \in \text{course} \land t \in \text{section} \land d \in \text{dept} \land d = \text{Biology} \right) \right) \) 
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:
  \[ \langle i, n, d, s \rangle \]
  - \( \langle i, n, d, s \rangle \) represents domain variables
  - \( s \) variables that range of attribute values
  - \( P \) represents a formula similar to that of the predicate calculus
  - Tuples can be formed using \(<>\)
    - E.g., \(<\text{Einstein}, \text{Physics}>\)

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both
  \( \langle x_1, x_2, ..., x_n \rangle \land (P_a(x_1, x_2, ..., x_n)) \)
  - \( x_1, x_2, ..., x_n \) represent domain variables
  - \( P_a \) variables that range of attribute values
  - \( \langle i, n, d, s \rangle \) represents domain variables
  - \( P \) represents a formula similar to that of the predicate calculus
  - Tuples can be formed using \(<>\)
    - E.g., \(<\text{Einstein}, \text{Physics}>\)

Safety of Expressions

The expression:
\[ \langle x_1, x_2, ..., x_n \rangle \land (P_a(x_1, x_2, ..., x_n)) \]

is safe if all of the following hold:
1. All values that appear in tuples of the expression are values from \( \text{dom}(P) \) (that is, the values appear either as constants in \( P \) or in a tuple of a relation mentioned in \( P \)).
2. For every “there exists” subformula of the form \( \exists x (P_i(x)) \), the subformula is true if and only if there is a value of \( x \) in \( \text{dom}(P_i) \) such that \( P_i(x) \) is true.
3. For every “for all” subformula of the form \( \forall x (P_i(x)) \), the subformula is true if and only if \( P_i(x) \) is true for all values \( x \) from \( \text{dom}(P_i) \).
Universal Quantification

Find all students who have taken all courses offered in the Biology department

\[
\{i \mid \exists n, d, tc (\langle i, n, d, tc \rangle \in \text{student} \land \\
(\forall cl, li, dl, cr (\langle cl, li, dl, cr \rangle \in \text{course} \land \\
dl = \text{"Biology"} \Rightarrow \exists si, se, y, g (\langle i, cl, si, se, y, g \rangle \in \text{takes})))\}
\]

Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

* Above query fixes bug in page 246, last query

Relationship between Relational Algebra and Tuple (Domain) Calculus

Codd’s theorem

- Relational algebra and tuple calculus are equivalent in terms of expressiveness
- That means that every query expressible in relational algebra can also be expressed in tuple calculus and vice versa
- Since domain calculus is as expressive as tuple calculus the same holds for the domain calculus
- Note: Here relational algebra refers to the standard version (no aggregation and projection with functions)

Recap

- Query language
  - Declarative
  - Retrieve, combine, and analyze data from a database instance
- Relational algebra
  - Standard relational algebra:
    - Selection, projection, renaming, cross product, union, set difference
    - Null values
    - Semantic sugar operators:
    - Intersection, joins, division,
    - Extensions:
    - Aggregation, extended projection
- Tuple Calculus
  - safety
- Domain Calculus

Outline

- Introduction
- Relational Data Model
- Formal Relational Languages (relational algebra)
- SQL - Introduction
- Database Design
- Transaction Processing, Recovery, and Concurrency Control
- Storage and File Structures
- Indexing and Hashing
- Query Processing and Optimization

End of Chapter 3

Modified from:
Database System Concepts, 6th Ed.
See also the book for the exercises on page
<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.14

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.15

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.16

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.17

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.18

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept</th>
<th>major</th>
<th>course</th>
<th>sec</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Boris</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1002</td>
<td>Silbersch</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-105</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1003</td>
<td>Korth</td>
<td>CS</td>
<td>Comp Sci</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1011</td>
<td>Silbersch</td>
<td>EN</td>
<td>Finance</td>
<td>EN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1013</td>
<td>Silbersch</td>
<td>ME</td>
<td>Mech Eng</td>
<td>ME-399</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>1016</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>1021</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1022</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1023</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>1024</td>
<td>Lohr</td>
<td>EN</td>
<td>Physics</td>
<td>EN-100</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>

Figure 6.19
Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes.
- A deletion is expressed in relational algebra by:
  \[ r \setminus E \]
  where \( r \) is a relation and \( E \) is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.
  \[ r_1 \setminus \text{branch\_city = "Perryridge" (account)} \]
- Delete all loan records with amount in the range of 0 to 50
  \[ r_2 \setminus \text{amount \geq 0 and amount \leq 50 (loan)} \]
- Delete all accounts at branches located in Needham.
  \[ r_3 \setminus \text{branch\_city = "Needham" (account)} \]

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- In relational algebra, an insertion is expressed by:
  \[ r \uplus E \]
  where \( r \) is a relation and \( E \) is a relational algebra expression.
- The insertion of a single tuple is expressed by letting \( E \) be a constant relation containing one tuple.

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.
  \[ r_1 \uplus \{("A-973", "Perryridge", 1200)\} \]
- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.
  \[ r_2 \uplus \{("Smith", "A-973")\} \]
Updating

- A mechanism to change a value in a tuple without charging all values in the tuple.
- Use the generalized projection operator to do this task.

\[ r \leftarrow \Pi_{F_i \in F} \ A_i \ (r) \]

- Each \( F_i \) is either
  - the \( i \)th attribute of \( r \), if the \( i \)th attribute is not updated, or,
  - if the attribute is to be updated \( F_i \) is an expression, involving only constants and the attributes of \( r \), which gives the new value for the attribute.

Example Queries

- Find the names of all customers who have a loan and an account at bank.
  \( \Pi_{\text{customer_name}} (\text{borrower}) \cap \Pi_{\text{customer_name}} (\text{depositor}) \)

- Find the name of all customers who have a loan at the bank and the loan amount
  \( \Pi_{\text{customer_name}, \text{loan_number}, \text{amount}} (\text{borrower} \times \text{loan}) \)

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.
  \( \Pi_{\text{customer_name}, \text{branch_name}} (\text{depositor} \times \text{account}) \cap \Pi_{\text{branch_name}} (\sigma_{\text{branch_city} = \text{"Brooklyn"}} (\text{branch})) \)