Relational Model & Algebra

CPS 216
Advanced Database Systems

Announcements

• Lecture notes
  – “Notes” version (incomplete) available in the morning on the day of lecture
  – “Slides” version (complete) available after the lecture
• We are working on installing IBM DB2!
  – Help needed
  – Good learning experience
• Reminder: check CourseInfo for announcements!

Relational data model

• A database is a collection of relations (or tables)
• Each relation has a list of attributes (or columns)
  – Set-valued attributes not allowed
• Each attribute has a domain (or type)
• Each relation contains a set of tuples (or rows)
  – Duplicates not allowed

• Simplicity is a virtue!
### Example

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart, 10, 2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse, 10, 3.1</td>
</tr>
<tr>
<td>857</td>
<td>Lisa, 8, 4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph, 8, 2.3</td>
</tr>
</tbody>
</table>

Ordering of rows doesn’t matter (even though the output is always in some order)

<table>
<thead>
<tr>
<th>Enroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>857</td>
</tr>
<tr>
<td>456</td>
</tr>
</tbody>
</table>

### Schema versus instance

- **Schema** (metadata)
  - Specification of how data is to be structured logically
  - Defined at set-up
  - Rarely changes
- **Instance**
  - Content
  - Changes rapidly, but always conforms to the schema
- Compare to types and variables in a programming language

**Example**

- **Schema**
  - `Student (SID integer, name string, age integer, GPA float)`
  - `Course (CID string, title string)`
  - `Enroll (SID integer, CID integer)`
- **Instance**
  - `{ [142, Bart, 10, 2.3], [123, Milhouse, 10, 3.1], ... }
  - `{ [CPS 216, Advanced Database Systems], ... }
  - `{ [142, CPS 216], [142, CPS 214], ... }`
Relational algebra operators

- Core set of operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.

Selection

- Input: a table $R$
- Notation: $\sigma_p(R)$
  - $p$ is called a selection condition/predicate
- Purpose: filter rows according to some criteria
- Output: same columns as $R$, but only rows of $R$ that satisfy $p$

Selection example

- Students with GPA higher than 3.0

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>657</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>
More on selection

- Selection predicate in general can include any column of R, constants, comparisons such as =, ≤, etc., and Boolean connectives ∧, ∨, and ¬.
  - Example: straight A students under 18 or over 21
- But you must be able to evaluate the predicate over a single row
  - Example: student with the highest GPA?

Projection

- Input: a table R
- Notation: \[\pi_L(R)\]
  - \(L\) is a list of columns in \(R\)
- Purpose: select columns to output
- Output: same rows, but only the columns in \(L\)

Projection example

- IDs and names of all students

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>671</td>
<td>Lisa</td>
<td>8</td>
<td>4.0</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
</tr>
<tr>
<td>671</td>
<td>Lisa</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
</tr>
</tbody>
</table>
More on projection

• Duplicate output rows must be removed
  – Example: age distribution of students

Cross product

• Input: two tables $R$ and $S$
• Notation: $R \times S$
• Purpose: pairs rows from two tables
• Output: for each row $r$ in $R$ and each row $s$ in $S$,
  output a row $rs$ (concatenation of $r$ and $s$)

Cross product example
Derived operator: join

- Input: two tables $R$ and $S$
- Notation: $R \bowtie p S$
  - $p$ is called a join condition/predicate
- Purpose: related rows from two tables according to some criteria
- Output: for each row $r$ in $R$ and each row $s$ in $S$, output a row $rs$ (concatenation of $r$ and $s$) if $r$ and $s$ satisfy $p$
- Shorthand for

Join example

- Info about students, plus CIDs of their courses

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td>142 CPS 216</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>142 CPS 214</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>142 CPS 216</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>142 CPS 214</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
<td>123 CPS 216</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>142 CPS 216</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>142 CPS 214</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
<td>123 CPS 216</td>
</tr>
</tbody>
</table>

Derived operator: natural join

- Input: two tables $R$ and $S$
- Notation: $R \bowtie S$
- Purpose: related rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes
- Shorthand for
Natural join example

\[ \text{Student} \bowtie \text{Enroll} = \pi_{\cdot} (\text{Student} \bowtie \text{Enroll}) \]

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Union

- Input: two tables \( R \) and \( S \)
- Notation: \( R \cup S \)
- \( R \) and \( S \) must have identical schema
- Output:
  - Has the same schema as \( R \) and \( S \)
  - Contains all rows in \( R \) and all rows in \( S \), with duplicates eliminated

Difference

- Input: two tables \( R \) and \( S \)
- Notation: \( R - S \)
- \( R \) and \( S \) must have identical schema
- Output:
  - Has the same schema as \( R \) and \( S \)
  - Contains all rows in \( R \) that are not found in \( S \)
Derived operator: intersection

- Input: two tables $R$ and $S$
- Notation: $R \cap S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows that are in both $R$ and $S$
- Shorthand for
- Also equivalent to

Renaming

- Input: a table $R$
- Notation: $ρ_{S}(R)$, or $ρ_{S}(A_1, A_2, \ldots)(R)$
- Purpose: rename a table and/or its columns
- Output: a renamed table with the same rows as $R$
- Used to
  - Avoid confusion caused by identical column names
  - Create identical columns names for natural joins

Renaming example

- All pairs of (different) students
Summary of core operators

- Selection: $\sigma_p(R)$
- Projection: $\pi_p(R)$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_{S}(A_1, A_2, \ldots) (R)$
  - Doesn’t really add to expressive power

Summary of derived operators

- Join: $R \bowtie S$
- Natural join: $R \bowtie S$
- Intersection: $R \cap S$

  - Many more
    - Semi-join, anti-semi-join, quotient, ...

An exercise

- CID of the courses that Lisa isn’t taking
A trickier exercise

- Who has the highest GPA?

Monotone operators

Add more rows to the input...

- If some old output rows must be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows remain “correct” when more rows are added to the input
  - Formally, $R \subseteq R' \Rightarrow \text{RelOp}(R) \subseteq \text{RelOp}(R')$

Classification of relational operators

<table>
<thead>
<tr>
<th>Monotone</th>
<th>Non-monotone</th>
<th>Not with respect to S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection: $\sigma_{P}(R)$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Projection: $\pi_{L}(R)$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cross product: $R \times S$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Union: $R \cup S$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Difference: $R - S$</td>
<td>✗</td>
<td>(Not with respect to S)</td>
</tr>
</tbody>
</table>
Why is “−” needed for highest GPA?

• Composition of monotone operators produces a monotone query
  – Old output rows remain “correct” when more rows are added to the input
• Highest-GPA query is?

Why do we need core operator X?

• Difference
• Cross product
• Union
• Selection? Projection?

Why is r.a. a good query language?

• Declarative?
  – Yes, compared to older languages like CODASYL
  – But operators are inherently procedural
• Simple
  – A small set of core operators whose semantics are easy to grasp
• Complete?
  – With respect to what?
Relational calculus

- \{ s.SID \mid \text{Student}(s) \land 
\neg (\exists s' \colon \text{Student}(s') \land s.GPA < s'.GPA) \}

- Relational algebra = “safe” relational calculus
  - Every query expressible in relational algebra is also expressive as a safe relational calculus formula
  - And vice versa

- Example of an unsafe relational calculus query

Turing machine?

- Relational algebra has no recursion

- Why not recursion?
  - Optimization becomes undecidable
  - You can always implement it at the application level
  - Recursion is added to SQL nonetheless

Next time

- How to design a relational database (and the theory behind it)

- No required reading, but for new comers to the field, reading related sections in a textbook is recommended
  - See Tentative Syllabus on course Web page