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>SID</td>
<td>name</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
</tr>
</tbody>
</table>

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

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

$$\sigma_{GPA > 3.0}(\text{Student})$$

<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>857</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 $=, \leq$, etc., and Boolean connectives $\land, \lor$, and $\neg$
  - Example: straight A students under 18 or over 21

$$\sigma_{GPA \geq 4.0 \land (\text{age} < 18 \lor \text{age} > 21)}(\text{Student})$$

- But you must be able to evaluate the predicate over a single row
  - Example: student with the highest GPA?

$$\sigma_{GPA = \text{max}(\text{GPA in Student table})}(\text{Student})$$

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

$$\pi_{\text{SID, name}}(\text{Student})$$

<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>857</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 projection

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

\[ \pi_{\text{age}} (\text{Student}) \]

- 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 \( s \))

\[ \pi_{\text{age}} \]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
\text{SID} & \text{name} & \text{age} & \text{GPA} & \text{SID} & \text{name} & \text{age} \\
\hline
142 & Bart & 10 & 2.3 & & & \\
123 & Milhouse & 10 & 3.1 & & & \\
857 & Lisa & 8 & 4.3 & & & \\
456 & Ralph & 8 & 2.3 & & & \\
\hline
\end{array}
\]

Cross product example

\( \text{Student} \times \text{Enroll} \)

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\hline
\text{SID} & \text{name} & \text{age} & \text{GPA} & \text{SID} & \text{CID} & \text{SID} & \text{name} & \text{age} & \text{GPA} & \text{SID} & \text{CID} \\
\hline
142 & Bart & 10 & 2.3 & & & & & & & & \\
123 & Milhouse & 10 & 3.1 & & & & & & & & \\
142 & Bart & 10 & 2.3 & 142 & CPS 216 & 142 & CPS 216 & & & & \\
123 & Milhouse & 10 & 3.1 & 142 & CPS 214 & 142 & CPS 214 & & & & \\
123 & Milhouse & 10 & 3.1 & 123 & CPS 214 & 123 & CPS 214 & & & & \\
\hline
\end{array}
\]

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 \( \sigma_{p} (R \times S) \)

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 \( \pi_{L} (R \bowtie_{p} S) \)
  - \( L \) is the union of the attributes from \( R \) and \( S \), with duplicates removed
  - \( p \) matches all attributes common to \( R \) and \( S \)
Natural join example

\[ \text{Student} \bowtie \text{Enroll} = \pi_{\text{SID, name, age, GPA}} (\text{Student} \bowtie \text{Enroll}) \]

\[ = \pi_{\text{SID, name, age, GPA}} (\text{Student} \bowtie \text{Enroll}) \]

\[
\begin{array}{cccc}
\text{SID} & \text{name} & \text{age} & \text{GPA} \\
142 & Bart & 10 & 2.3 \\
123 & Milhouse & 10 & 3.1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{SID} & \text{name} & \text{age} & \text{GPA} & \text{CID} \\
142 & Bart & 10 & 2.3 & CPS 216 \\
142 & Bart & 10 & 2.3 & CPS 214 \\
123 & Milhouse & 10 & 3.1 & CPS 216 \\
\end{array}
\]

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 \( S \)

Renaming

- Input: a table \( R \)
- Notation: \( \rho_S (R) \), or \( \rho_S (A_1, A_2, \ldots) (R) \)
- 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

\[
\begin{array}{cccc}
\text{SID1} & \text{name1, age1, GPA1} \\
\text{SID2} & \text{name2, age2, GPA2} \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{SID1} & \text{name2, age2, GPA2} \\
\text{SID2} & \text{name1, age1, GPA1} \\
\end{array}
\]
Summary of core operators

- Selection: $\sigma_p(R)$
- Projection: $\pi_L(R)$
- 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: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Intersection: $R \cap S$
- Union: $R \cup S$
- Difference: $R - S$
- Many more
  - Semi-join, anti-semi-join, quotient, ...

An exercise

- CIDs of the courses that Lisa isn’t taking

A trickier exercise

- Who has the highest GPA?
- Who does not have the highest GPA?
- Whose GPA is lower than somebody else’s?

Monotone operators

- 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, $R \subseteq R' \Rightarrow RelOp(R) \subseteq RelOp(R')$

Classification of relational operators

Monotone ✓ Non-monotone ✗

- Selection: $\sigma_R(R)$ ✓
- Projection: $\pi_L(R)$ ✓
- Cross product: $R \times S$ ✓
- Union: $R \cup S$ ✓
- Difference: $R - S$ ✗ (Not with respect to $S$)
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 non-monotone
  - Current highest GPA is 4.3
  - Add another GPA 4.5
  - Old answer is invalidated
- So it must use difference!

Why do we need core operator X?

- Difference
  - The only non-monotone operator
- Cross product
  - The only operator that allows you to add columns
- Union
  - The only operator that allows you to add rows?
  - A more rigorous proof?
- Selection? Projection?
  - Homework problem :-)

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 | Student (s) ∧ ¬(∃s': Student (s') ∧ 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
  \{ s.name | ¬Student (s) \}
  - Can’t evaluate this query just by looking at the database

Turing machine?

- Relational algebra has no recursion
  - Example of something not expressible in relational algebra: Given relation Parent (parent, child), who are Bart’s ancestors?
- 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