Relational Model & Algebra

CPS 196.3
Introduction to Database Systems

Announcements

- Lectures slides on Web
  - “Notes” version available an hour before lecture
  - Complete version available after the lecture
  - No hardcopy handout
- Reading assignment posted on Web (under “Tentative Syllabus”)
- The first homework will be assigned next Tuesday
- Office hours
  - After lectures on Tuesdays and Thursday, in D327
  - Or by appointment

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>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>

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

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS196</td>
<td>Intro to Database Systems</td>
</tr>
<tr>
<td>123</td>
<td>CPS130</td>
<td>Analysis of Algorithms</td>
</tr>
<tr>
<td>857</td>
<td>CPS196</td>
<td>Computer Networks</td>
</tr>
<tr>
<td>456</td>
<td>CPS114</td>
<td>Computer Networks</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
  \( \sigma_{GPA \geq 3.0}(Student) \)
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
    \[ σ_{GPA \geq 4.0 \land age < 18 \lor age > 21} (Student) \]
- But you must be able to evaluate the predicate over a single row
  - Example: student with the highest GPA?
    \[ σ_{GPA = \text{MAX}(GPA)} (Student) \]

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

- ID’s and names of all students
  \[ \pi_{\text{SID}, \text{name}} (Student) \]

<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>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>416</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>
More on projection

- Duplicate output rows must be removed
  - Example: student ages

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

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

- \( \text{Student} \times \text{Enroll} \)
A note on column ordering

- The ordering of columns in a table is considered unimportant (so is the ordering of rows)

<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>CPS196</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>CPS114</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>11</td>
<td>3.1</td>
<td>CPS196</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
<td>CPS114</td>
</tr>
</tbody>
</table>

- That means cross product is commutative, i.e., $R \times S = S \times R$ for any $R$ and $S$

Derived operator: join

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

Join example

- Info about students, plus CID’s of their courses

**Student** $owtie_{\text{Student.SID} = \text{Enroll.SID}}$ **Enroll**
Derived operator: natural join

- **Input:** two tables $R$ and $S$
- **Notation:** $R \bowtie S$
- **Purpose:** relate 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 all attributes from $R$ and $S$, with duplicates removed
  - $p$ equates all attributes common to $R$ and $S$

Natural join example

- $\text{Student} \bowtie \text{Enroll} = \pi_L, (\text{Student} \bowtie_p, \text{Enroll})$

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**
- **And to**

Renaming

- **Input:** a table \( R \)
- **Notation:** \( \rho_{A_1, A_2, \ldots} (R) \)
  - \( A_1, A_2, \ldots \) are column names
- **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
  \[ \text{Student} \bowtie \text{Student} \]
  \[ \text{Student} \bowtie \text{Student} \]
  \[ \text{ρ}_{\text{Student}(\text{SID}_1, \text{Name}_1, \text{Age}_1, \text{GPA}_1)} \]
  \[ \text{ρ}_{\text{Student}(\text{SID}_2, \text{Name}_2, \text{Age}_2, \text{GPA}_2)} \]

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: \( \rho_{A_1, A_2, \ldots}(R) \)
  - Does not really add to expressive power

Summary of derived operators

- Join: \( R \bowtie S \)
- Natural join: \( R \bowtie S \)
- Intersection: \( R \cap S \)

- Many more
  - Semijoin, anti-semijoin, quotient, …
An exercise

- CID's of the courses that Lisa is NOT taking

A trickier exercise

- Who has the highest GPA?

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, $R \subseteq R'$ implies $\text{RelOp}(R) \subseteq \text{RelOp}(R')$
Classification of relational operators

- Selection: $\sigma_p (R)$
- Projection: $\pi_L (R)$
- Cross product: $R \times S$
- Join: $R \bowtie S$
- Natural join: $R \bowtie S$
- Union: $R \cup S$
- Difference: $R - S$
- Intersection: $R \cap S$

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
  - 
- Cross product
  - 
- Union
  - 
  - 
- Selection? Projection?
  -  
  - 
Why is r.a. a good query language?

- Declarative?
  - Yes, compared with older languages like CODASYL.
  - But operators are inherently procedural.
- Simple
  - A small set of core operators who semantics are easy to grasp.
- Complete?
  - With respect to what?

Relational calculus

- \{ s.SID | s \in Student \land \\
  \neg(\exists s' \in Student: s.GPA < s'.GPA) \}, or
- \{ s.SID | s \in Student \land \\
  (\forall s' \in Student: s.GPA \geq s'.GPA) \}

- Relational algebra = “safe” relational calculus
  - Every query expressible as a safe relational calculus query is also expressible as a relational algebra query.
  - And vice versa.
- Example of an unsafe relational calculus query
  - \{ s.name | \neg(\exists s \in Student) \}
  - Cannot 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 nevertheless.