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
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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Course
<table>
<thead>
<tr>
<th>CID</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPS196</td>
<td>Intro. to Database Systems</td>
</tr>
<tr>
<td>CPS130</td>
<td>Analysis of Algorithms</td>
</tr>
<tr>
<td>CPS114</td>
<td>Computer Networks</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Enroll
<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS196</td>
</tr>
<tr>
<td>142</td>
<td>CPS114</td>
</tr>
<tr>
<td>123</td>
<td>CPS196</td>
</tr>
<tr>
<td>857</td>
<td>CPS130</td>
</tr>
<tr>
<td>456</td>
<td>CPS114</td>
</tr>
<tr>
<td>...</td>
<td>...</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}, ... }
  - { {CPS196, Intro. to Database Systems}, ... }
  - { {142, CPS196}, {142, CPS114}, ... }
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>1.1</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>

Projection example
- ID’s 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>1.1</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: student ages

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

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)

- 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 \( \sigma_p \left( R \times S \right) \)

Join example

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

\[ \text{Student} \bowtie_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll} \]
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

- $Student \bowtie Enroll = \pi_4(Student \bowtie_3 Enroll)$
  - $\pi_{\text{SID, name, age, GPA, CID}}(Student \bowtie_3 \text{Student.SID} = \text{Enroll.SID}, Enroll)$

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** $R \cap (R - S)$
  - Also equivalent to $S - (S - R)$
  - And to $R \bowtie S$

Renaming

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

"Student" as input

<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>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
<td>CPS114</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

"Enroll" as input

<table>
<thead>
<tr>
<th>SID</th>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
<th>Enroll.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>10</td>
<td>3.1</td>
<td>CPS196</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

"Student.SID = Enroll.SID" as input

Looking at the natural join example, we see that the rows are combined based on the condition that the SID column is equal in both tables. This ensures that only matching rows are included in the result, preserving the integrity of the data.
Renaming example

- All pairs of (different) students
  
  \[
  \text{Student} \bowtie_1 \text{Student}
  \]

\[
\rho_{\text{Student}(\text{SID1}, \text{Name1}, \text{Age1}, \text{GPA1})}
\]

\[
\text{Student}
\]

\[
\rho_{\text{Student}(\text{SID2}, \text{Name2}, \text{Age2}, \text{GPA2})}
\]

\[
\text{Student}
\]

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: \( R \bowtie_p S \)
- 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

\[
\pi_{\text{CID}} 
\]

\[
\sigma_{\text{Name} = \text{Lisa}}
\]

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:** $\sigma_p(R)$  
  - Monotone
- **Projection:** $\pi_L(R)$  
  - Monotone
- **Cross product:** $R \times S$  
  - Monotone
- **Join:** $R \bowtie S$  
  - Monotone
- **Natural join:** $R \bowtie S$  
  - Monotone
- **Union:** $R \cup S$  
  - Monotone
- **Difference:** $R - S$  
  - Non-monotone (not w.r.t. $S$)
- **Intersection:** $R \cap S$  
  - Monotone

## 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 adds 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 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?

## 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

## Relational calculus

- $\{ s \text{.SID} \mid s \in \text{Student} \land \neg(\exists s' \in \text{Student} : s.\text{GPA} < s'.\text{GPA}) \}$, or
  $\{ s \text{.SID} \mid s \in \text{Student} \land (\forall s' \in \text{Student} : s.\text{GPA} \geq s'.\text{GPA}) \}$
- **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**
  - $\{ s.\text{name} \mid \neg(s \in \text{Student}) \}$
  - Cannot evaluate this query just by looking at the database