Relational Model & Algebra
CPS 116
Introduction to Database Systems

Announcements
- Please read news: duke.cs.cps116
- Please sign up for DB2 accounts and indicate your availability to attend discussion sessions (sign-up sheet is circulating)
- Lectures slides on Web
  - "Notes" version available in hardcopy and online an hour before lecture
  - Complete version available online after the lecture
- Homework #1 will be assigned next Tuesday
- Office hours: see course Web page
- Book update: Prentice-Hall sent something out yesterday

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)
  - Duplicate tuples are 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>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

Example

<table>
<thead>
<tr>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student (SID integer, name string, age integer, GPA float)</td>
</tr>
<tr>
<td>Course (CID string, title string)</td>
</tr>
<tr>
<td>Enroll (SID integer, CID integer)</td>
</tr>
</tbody>
</table>

Instance
- { (142, Bart, 10, 2.3), (123, Milhouse, 10, 3.1), ... }
- { (CPS116, Intro. to Database Systems), ... }
- { (142, CPS116), (142, CPS114), ... }

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 type and objects of type in a programming language
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
  $\sigma_{\text{GPA} > 3.0} (\text{Student})$

<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>

<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
    $\sigma_{\text{GPA} \geq 4.0 \land (\text{age} < 18 \lor \text{age} > 21)} (\text{Student})$
  - But you must be able to evaluate the predicate over a single row of the input table
  - Example: student with the highest GPA
    $\sigma_{\text{GPA} = \text{max(GPA in Student table)}} (\text{Student})$

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

- ID's and names of all students
  $\pi_{\text{SID}, \text{name}} (\text{Student})$

<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: student ages

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

**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)

\[ \text{SID} \times \text{CID} = \text{CID} \times \text{SID} \]

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

**Join example**

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

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

Use \text{table}\_name\_\text{column}\_name syntax to disambiguate identically named columns from different input tables
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 \( \pi_{\sigma} ( R \bowtie S ) \)
  - \( L \) is the union of all attributes from \( R \) and \( S \), with duplicate attributes removed
  - \( \sigma \) equates all attributes common to \( R \) and \( S \)

Natural join example

- \( \text{Student} \bowtie \text{Enroll} = \pi_{L} ( \text{Student} \bowtie_{\pi_{\sigma} \text{Student.SID} = \text{Enroll.SID}} \text{Enroll} ) \)

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 \( R - ( R - S ) \)
- Also equivalent to \( S - ( S - R ) \)
- And to \( R \bowtie_{\neq} S \)

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

- SID’s of students who take at least two courses

:\[ \pi_{\text{SID}}(\text{Enroll } \bowtie_2 \text{ Enroll}) \]

:\[ \rho_{\text{Enroll}((\text{SID}, \text{CID}))(1)}(\text{Enroll}) \]

:\[ \rho_{\text{Enroll}((\text{SID}, \text{CID}))(2)}(\text{Enroll}) \]

\[ \rho_{\text{Enroll}((\text{SID}, \text{CID}))(1)}(\text{Enroll}) = \text{SID} \land \text{CID} \neq \text{CID} \]

\[ \text{SID} \]

\[ \text{Enroll} \]

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 processing power

Summary of derived operators

- Join: \( R \bowtie_j S \)
- 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

:\[ \pi_{\text{CID}}(\text{Course}) \]

:\[ \sigma_{\text{name}} = "\text{Lisa}"(\text{Student}) \]

:\[ \text{All CID’s} \]

:\[ \text{CID’s of the courses that Lisa IS taking} \]

Another exercise

- Names of students in Lisa’s classes

:\[ \pi_{\text{name}}(\text{Student}) \]

:\[ \text{Their names} \]

:\[ \pi_{\text{SID}}(\text{Student}) \]

:\[ \text{Their SID} \]

:\[ \text{Student} \]

A trickier exercise

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

:\[ \pi_{\text{SID}}(\text{Student}) \]

:\[ \pi_{\text{Student}1.\text{SID}}(\text{Student}) \]

:\[ \bowtie_1(\text{Student}1.\text{GPA} < \text{Student}2.\text{GPA}) \]

:\[ \text{Student}1 \]

:\[ \text{Student}2 \]

:\[ \text{Student}1 \]

:\[ \text{Student}2 \]

A deeper question:
When (and why) is “…” needed?
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, \( R \subseteq R' \) implies \( \text{RelOp}(R) \subseteq \text{RelOp}(R') \)

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 4.1
  - Add another GPA 4.2
  - 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 argument?
- Selection? Projection?
  - Homework problem 😛

Why is r.a. a good query language?

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

Relational calculus

\[
\{ s.SID \mid s \in \text{Student} \land \neg (\exists s' \in \text{Student} : s.GPA < s'.GPA) \}, \text{ or } \{ s.SID \mid s \in \text{Student} \land (\forall s' \in \text{Student} : s.GPA \geq s'.GPA) \}
\]

- 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
  - \( \{ s.name \mid \neg (s \in \text{Student}) \} \)
  - 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