Relational Model & Algebra

CPS 116
Introduction to Database Systems

Announcements (Thurs. Aug. 30)

- Check out what some former 116ers (Anthony Bishopric and Tomas Barreto) have been up to: http://www.shoeboxed.com/
- Homework #1 will be assigned next Thursday
- Office hours: see also course Web page
  - Jun: Tuesday before/after class; Thursday before class
  - Yi: Wednesday and Friday afternoons 12:30pm-2:00pm
- Note on lecture notes
  - The “complete” version will be posted after lecture

Relational data model

- A database is a collection of relations (or tables)
- Each relation has a list of attributes (or columns)
- Each attribute has a domain (or type)
  - Set-valued attributes not allowed
- Each relation contains a set of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed
    - Two tuples are identical if they agree on all attributes
- 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>

<table>
<thead>
<tr>
<th>Course</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPS116</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>
</tbody>
</table>

Enroll

<table>
<thead>
<tr>
<th>SID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>CPS116</td>
</tr>
<tr>
<td>142</td>
<td>CPS114</td>
</tr>
<tr>
<td>123</td>
<td>CPS118</td>
</tr>
<tr>
<td>333</td>
<td>CPS115</td>
</tr>
<tr>
<td>456</td>
<td>CPS113</td>
</tr>
<tr>
<td>857</td>
<td>CPS119</td>
</tr>
<tr>
<td>456</td>
<td>CPS114</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 type and objects of type 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), … }
  - { (CPS116, Intro. to Database Systems), … }
  - { (142, CPS116), (142, CPS114), … }
Relational algebra
A language for querying relational databases based on operators:

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

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
  - $\sigma_{\text{GPA} > 3.0} \text{Student}$

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
  - $\pi_{\text{SID, name}} \text{Student}$
More on projection

- Duplicate output rows are removed (by definition)
  - 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 (as 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 \( \sigma_p ( R \times S ) \)

Join example

- Info about students, plus CID’s of their courses
  \( \text{Student} \bowtie_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll} \)

Use table_name.column_name syntax to disambiguate identically named columns from different input tables
Derived operator: natural join

- Input: two tables $R$ and $S$
- Notation: $R owtie S$
- Purpose: relate rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes
- Shorthand for $\pi_p (R \bowtie_p S)$, where
  - $p$ equates all attributes common to $R$ and $S$
  - $L$ is the union of all attributes from $R$ and $S$, with duplicate attributes removed

Natural join example

$$\text{Student} \bowtie \text{Enroll} = \pi_j (\text{Student} \bowtie_{j=1} \text{Enroll})$$

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
</tbody>
</table>

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 $R \cap S$
- Also equivalent to $S \cap R$
- And to $R \bowtie S$

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 duplicate rows 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: $\rho_A R$, $\rho_{(A_1, A_2, \ldots)} R$ or $\rho_{(A_1, A_2, \ldots)} R$
- 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

Enroll \(\bowtie\) Enroll

\[ \pi_{\text{SID}} (\text{Enroll} \bowtie \text{Enroll}) \]

Expression tree syntax:

\[
\pi_{\text{SID}}
\]

Enroll

\[
\rho_{\text{Enroll}}(\text{SID1}, \text{CID1})
\]

Enroll

\[
\rho_{\text{Enroll}}(\text{SID2}, \text{CID2})
\]

\[
\text{Enroll} \bowtie \text{Enroll}
\]

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 “processing” 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

- Names of students in Lisa’s classes

Writing a query bottom-up:

Who’s Lisa?

\[
\sigma_{\text{name} = “Lisa”} \text{Student} \rightarrow
\]

\[
\pi_{\text{SID}} \text{Student} \rightarrow
\]

Liza’s classes

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

Students in Lisa’s classes

\[
\pi_{\text{name}} \rightarrow
\]

Their names

Another exercise

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

Writing a query top-down:

Who has the highest GPA?

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

Another trickier exercise

A deeper question:

- When (and why) is “−” needed?
Monotone operators

If some old output rows may need to be removed

- Then the operator is non-monotone
- Otherwise the operator is monotone

Formally, for a monotone operator \( op \):

\[ R \subseteq R' \implies op(R) \subseteq op(R') \]

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: \( R - S \)  — Monotone w.r.t. \( R \); non-monotone 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 4.1
  - 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
  - Though operators do look somewhat “procedural”
- Complete?
  - With respect to what?

Relational calculus

- \{ \( s.SID \mid s \in \text{Student} \land \neg(\exists i \in \text{Student}; s.GPA < i'.GPA) \} \), or
- \{ \( s.SID \mid s \in \text{Student} \land (\forall i' \in \text{Student}; s.GPA \geq i'.GPA) \} \)
- 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
  - \{ \( s.name \mid s \in \text{Student} \) \}
  - 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 Turing machine?
  - Optimization becomes undecidable
  - You can always implement it at the application level

- Recursion is added to SQL nevertheless!