Relational Model & Algebra

CompSci 316
Introduction to Database Systems

Announcements (Tue. Sep. 3)
- Homework #1 has been posted
  - Sign up for Gradiance now!
  - Windows Azure passcode will be emailed soon—sign up as soon as you can!
- Office hours: see course website
- Lecture notes
  - The "notes" version can be (printed out and) used for note-taking; the "complete" version will be posted after lecture; be selective in what you copy down
- Readings: see Tentative Syllabus on course website
- Working on the enrollment issue

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>142</td>
<td>CPS316</td>
</tr>
<tr>
<td>142</td>
<td>CPS310</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPS310</td>
<td>Intro. to Database Systems</td>
</tr>
<tr>
<td>CPS330</td>
<td>Analysis of Algorithms</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Schema (metadata)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of how data is to be structured logically</td>
</tr>
<tr>
<td>Defined at set-up</td>
</tr>
<tr>
<td>Rarely changes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
</tr>
<tr>
<td>Changes rapidly, but always conforms to the schema</td>
</tr>
<tr>
<td>Compare to type and objects of type in a programming language</td>
</tr>
</tbody>
</table>

Example

- 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), ...}
  - {(CPS316, Intro. to Database Systems), ...}
  - {(142, CPS316), (142, CPS310), ...}
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
  \( \pi_{\text{SID}, \text{name}} \text{Student} \)
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 \( s \))

A note on column ordering

- The ordering of columns in a table is unimportant as far as the contents are concerned

Derived operator: join

(A.k.a. “theta-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

\( \text{Student} \bowtie_{\text{Student.SID=Enroll.SID}} \text{Enroll} \)

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: $R \bowtie S$
- Purpose: relate rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes
- Shorthand for $\pi_L(R \bowtie_p S)$, 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

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 $S$

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 $R \bowtie S$

Renaming

- Input: a table $R$
- Notation: $\rho_{S,R}(A_1, A_2, \ldots) R$ 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
  \[ Enroll \bowtie_2 Enroll \]

Expression tree syntax:

\[
\begin{align*}
\pi_{\text{SID}}(Enroll) & \bowtie_2 \pi_{\text{SID}}(Enroll) \\
\rho_{\text{SID}_1 = \text{SID}_2 \wedge \text{CID}_1 = \text{CID}_2} (Enroll) & \bowtie_2 \rho_{\text{SID}_2 = \text{CID}_2} (Enroll)
\end{align*}
\]

Summary of core operators

- Selection: \( \sigma_R \)
- Projection: \( \pi_R \)
- 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: \( R \bowtie_p S \)
- 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:

\[
\begin{align*}
\pi_{\text{Name}} & (\text{Students in Lisa’s classes}) \\
\sigma_{\text{Name} = \text{Lisa}} & (\text{Student}) \\
\pi_{\text{SID}} & (\text{Enroll}) \\
\end{align*}
\]

Another exercise

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

Writing a query top-down:

\[
\begin{align*}
\pi_{\text{CID}} & (\text{Course}) \\
\pi_{\text{SID}} & (\text{Enroll}) \\
\sigma_{\text{Name} = \text{Lisa}} & (\text{Student})
\end{align*}
\]

A trickier exercise

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

A deeper question:

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

Formally, for a monotone operator \( op \): 
\[ R \subseteq R' \implies op(R) \subseteq op(R') \text{ for any } R, R' \]

Why is “−” needed for highest GPA?

Composition of monotone operators produces a monotone query

1. Old output rows remain “correct” when more rows are added to the input
2. Is highest-GPA query monotone?
   - No!
   - 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 \( X \)?

1. Difference
2. The only non-monotone operator
3. Cross product
4. The only operator that adds columns
5. Union
6. The only operator that allows you to add rows?
7. A more rigorous argument?
8. Selection? Projection?
9. Homework problem 😊

Extensions to relational algebra

1. Duplicate handling ("bag algebra")
2. Grouping and aggregation
3. Extension (or extended projection) to allow new attribute values to be computed

- All these will come up when we talk about SQL
- But for now we will stick to standard relational algebra without these extensions

Why is r.a. a good query language?

1. Simple
   - A small set of core operators whose semantics are easy to grasp
2. Declarative?
   - Yes, compared with older languages like CODASYL
   - Though operators do look somewhat "procedural"
3. 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)\}, 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(\exists 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!