Relational Model & Algebra

CPS 216
Advanced Database Systems

Announcements (January 13)
- Homework #1 will be assigned on Thursday
- Reading assignment for this week
  - Posted on course Web page
  - Remember to register on H2O and join Duke CPS216
  - Review due on Thursday night

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

Students
<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>

Courses
<table>
<thead>
<tr>
<th>CID</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPS216</td>
<td>Advanced Database Systems</td>
</tr>
<tr>
<td>CPS230</td>
<td>Analysis of Algorithms</td>
</tr>
<tr>
<td>CPS214</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>CPS216</td>
</tr>
<tr>
<td>142</td>
<td>CPS214</td>
</tr>
<tr>
<td>123</td>
<td>CPS216</td>
</tr>
<tr>
<td>857</td>
<td>CPS230</td>
</tr>
<tr>
<td>456</td>
<td>CPS214</td>
</tr>
</tbody>
</table>

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

Why did Codd call them “relations”?
Each n-tuple relates n elements from n domains, precisely in the mathematical sense of a “relation”

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 object 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), ... }
  - { (CPS216, Advanced Database Systems), ... }
  - { (142, CPS216), (142, CPS214), ... }
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
  $\sigma_{GPA > 3.0} (\text{Student})$

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 must be removed
  - Example: student ages

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

\[ \begin{array}{ccc}
142 & Bart & 10 \\
123 & Milhouse & 10 \\
857 & Lisa & 8 \\
456 & Ralph & 8 \\
\end{array} \]

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 \( s \))

\[ \text{SID name age GPA} \]
\[ \begin{array}{ccc}
142 & Bart & 10 \\
123 & Milhouse & 10 \\
857 & Lisa & 8 \\
456 & Ralph & 8 \\
\end{array} \]

Cross product example

- \( \text{Student} \times \text{Enroll} \)

\[ \begin{array}{ccc}
\text{SID} & \text{name} & \text{age} \\
142 & Bart & 10 \\
123 & Milhouse & 10 \\
\end{array} \]

\[ \begin{array}{ccc}
\text{SID} & \text{CID} \\
142 & CPS216 \\
142 & CPS214 \\
123 & CPS216 \\
\end{array} \]

A note on column ordering

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

\[ \begin{array}{ccc}
\text{SID} & \text{name} & \text{age} \\
142 & Bart & 10 \\
123 & Milhouse & 10 \\
\end{array} \]

\[ \begin{array}{ccc}
\text{SID} & \text{CID} \\
142 & CPS216 \\
142 & CPS214 \\
123 & CPS216 \\
\end{array} \]

- 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: \( \delta_p R \times S \)
  - \( p \) 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

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

\[ \begin{array}{ccc}
\text{SID} & \text{name} & \text{age} \\
142 & Bart & 10 \\
142 & Bart & 10 \\
123 & Milhouse & 10 \\
\end{array} \]

\[ \begin{array}{ccc}
\text{CID} & \text{SID} & \text{name} \\
CPS216 & 142 & Bart \\
CPS214 & 142 & Bart \\
CPS216 & 123 & Milhouse \\
\end{array} \]
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** $\pi_L(R \bowtie S)$
  - $L$ is the union of all attributes from $R$ and $S$, with duplicates removed
  - $\bowtie$ equates all attributes common to $R$ and $S$

Natural join example
- **Student** $\bowtie$ **Enroll** $= \pi_{\text{ID}, \text{name}, \text{age}, \text{GPA}, \text{CID}} (\pi_{\text{Student}}(\text{Student} \bowtie \text{Enroll}))$

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** $R \bowtie S$

Renaming
- **Input:** a table $R$
- **Notation:** $\rho_S(\pi_{\text{Student}}(\text{Student} \bowtie \text{Enroll}))$
- **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 column names for natural joins
Renaming example

- SID’s of students who take at least two courses
  $\text{Enroll} \bowtie_2 \text{Enroll}$

  $\pi_{\text{SID}}(\text{Enroll} \bowtie_2 \text{Enroll})$

  $\rho_{\text{Enroll}((\text{SID}, \text{CID})_1)}$

  $\rho_{\text{Enroll}((\text{SID}, \text{CID})_2)}$

  $\text{Enroll}$

Summary of core operators

- Selection: $\sigma_p(\text{R})$
- Projection: $\pi_L(\text{R})$
- Cross product: $\text{R} \times \text{S}$
- Union: $\text{R} \cup \text{S}$
- Difference: $\text{R} - \text{S}$
- Renaming: $\rho_{A_1, A_2, \ldots}(\text{R})$
  - Does not really add to processing power

Summary of derived operators

- Join: $\text{R} \bowtie \text{S}$
- Natural join: $\text{R} \bowtie S$
- Intersection: $\text{R} \cap S$
- Many more
  - Semijoin, anti-semijoin, quotient, …

An exercise

- CID’s of the courses that Lisa is NOT taking
  $\text{Student} \bowtie \text{Course}$
  $\text{Enroll} \bowtie \sigma_{\text{name} = "Lisa"}$

- CID’s of the courses that Lisa IS taking

A trickier exercise

- SID’s of students who take exactly one course
  - Those who take at least one course
  - Those who take at least two courses
  - Take the difference!

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, $\text{R} \subseteq R' \implies \text{RelOp}(\text{R}) \subseteq \text{RelOp}(\text{R'})$
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
- Difference: \( R \setminus S \) Non-monotone (not w.r.t. \( S \))
- Intersection: \( R \cap S \) Monotone

Why is “−” needed for “exactly one”?  
- Composition of monotone operators produces a monotone query
  - Old output rows remain “correct” when more rows are added to the input
- Exactly-one query is non-monotone  
  - Say Nelson is currently taking only CPS216
  - Add another record to Enroll: Nelson takes CPS214 too
  - Nelson is no longer in the answer
  - 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?

Relational calculus

- \( \{ e.\text{SID} \mid e \in \text{Enroll} \land \neg \exists e' \in \text{Enroll} : e.\text{SID} = e'.\text{SID} \land e'.\text{CID} \neq e.\text{CID} \} \) or \( \{ e.\text{SID} \mid e \in \text{Enroll} \land (\forall e' \in \text{Enroll} : e.\text{SID} \neq e'.\text{SID} \lor e'.\text{CID} = e.\text{CID} ) \} \)

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