Relational Model & Algebra

CPS 216
Advanced Database Systems

Announcements (January 18)

- Homework #1 will be assigned on Thursday
- Reading assignment for this week
  - Posted on course Web page
  - 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

<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>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>

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
  $\sigma_{\text{GPA} > 3.0} (\text{Student})$

<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>

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 must be removed
  - Example: student ages

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

\begin{tabular}{l|l|l}
SID & name & age \\
--- & --- & --- \\
142 & Bart & 10 \\
123 & Milhouse & 10 \\
857 & Lisa & 8 \\
456 & Ralph & 8 \\
\end{tabular}

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 \( s \))

\[ \text{SID name age GPA} \]
\[ \hline \]
\[ 142 \text{ Bart 10 2.3} \]
\[ 123 \text{ Milhouse 10 3.1} \]
\[ 857 \text{ Lisa 8 4.3} \]
\[ 456 \text{ Ralph 8 2.3} \]

A note on column ordering

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

\[ \text{SID name GPA SID CID} \]
\[ \hline \]
\[ 142 \text{ Bart 10 2.3 CPS216 CPS216} \]
\[ 142 \text{ Bart 10 2.3 CPS214 CPS214} \]
\[ 857 \text{ Lisa 8 4.3 CPS216 CPS216} \]
\[ 857 \text{ Lisa 8 4.3 CPS214 CPS214} \]
\[ 456 \text{ Ralph 8 2.3 CPS216 CPS216} \]
\[ 456 \text{ Ralph 8 2.3 CPS214 CPS214} \]

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

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

\[ \text{SID name age GPA SID CID} \]
\[ \hline \]
\[ 142 \text{ Bart 10 2.3 CPS216 CPS216} \]
\[ 142 \text{ Bart 10 2.3 CPS214 CPS214} \]
\[ 123 \text{ Milhouse 10 1.1 CPS214 CPS214} \]
\[ 123 \text{ Milhouse 10 1.1 CPS216 CPS216} \]
\[ 123 \text{ Milhouse 10 1.1 CPS214 CPS214} \]
\[ 123 \text{ Milhouse 10 1.1 CPS216 CPS216} \]
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
  - \( \rho \) equates all attributes common to \( R \) and \( S \)

### Natural join example

Student \( \bowtie \) Enroll = \( \pi_L( \text{Student} \bowtie \text{Enroll} ) = \pi_{\text{Student.IID, name, age, GPA, CID}}( \text{Student} \bowtie \text{Enroll} ) \)

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>age</th>
<th>GPA</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>CPS216</td>
</tr>
<tr>
<td>2</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
<td>CPS214</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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: \( \rho_{\rho_{S}}( R ) \), or \( \rho_{S}( S_{1}, S_{2}, \ldots)( R ) \)
- 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
  \[ \pi_{\text{SID}}(\text{Enroll} \triangleright\triangleright_{2} \text{Enroll}) \]
  \[ \rho_{\text{Enroll}}(\text{SID}_1, \text{CID}_1) \]
  \[ \rho_{\text{Enroll}}(\text{SID}_2, \text{CID}_2) \]

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: \( R \bowtie S \)
- 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
  \[ \pi_{\text{CID}}(\text{Course}) \]
  \[ \sigma_{\text{name} = \text{"Lisa"}}(\text{Student}) \]

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?
  \[ \pi_{\text{SID}}(\text{Enroll}) \]
  \[ \rho_{\text{Enroll}}(\text{SID}_1, \text{CID}_1) \]
  \[ \rho_{\text{Enroll}}(\text{SID}_2, \text{CID}_2) \]

Monotone operators

- If some old output rows may 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
- Difference: $R - S$ 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
- Projection
  - The only operator that removes columns
- Cross product
  - The only operator that adds columns
- Union
  - The only operator that allows you to add rows?
  - A more rigorous proof?
- Selection? 🤔

Why is r.a. a good query language?

- Declarative?
  - Yes, compared with older languages like CODASYL
  - Though operators still feel “procedural”
- Simple
  - A small set of core operators who semantics are easy to grasp
- Complete?
  - With respect to what?

Relational calculus

- $\{ e.SID \mid e \in Enroll \land \neg \exists e' \in Enroll: e'.SID = e.SID \land e'.CID \neq e.CID \}$ or
  - $\{ e.SID \mid e \in Enroll \land (\forall e' \in Enroll: e'.SID \neq e.SID \lor e'.CID = e.CID) \}$
- 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
  - $\{ e.name \mid \neg \exists e \in Student \}$
  - 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 Barr’s ancestors?
- Why not recursion?
  - Optimization becomes undecidable
  - You can always implement it at the application level
  - Recursion is added to SQL nevertheless