Relational Model & Algebra

CPS 216
Advanced Database Systems

Announcements

- Homework #1 and course project will be assigned next Wednesday (January 22)
- Reading assignments in the “red book” will be photocopied and available outside my office (D327)
  - Codd paper is already there!
- There IS a recitation session this Friday (January 17) on E/R database design and review of relational algebra and design theory
- Instructor and TA office hours will be finalized this Wednesday

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>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>

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

Why did Codd call them “relations”?

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 types and instances of types 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
  $\sigma_{GPA > 3.0}(\text{Student})$

<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>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
</tbody>
</table>

$\sigma_{GPA > 3.0}$ (Student)
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
  - Example: student with the highest GPA
    \[ \sigma_{\text{GPA} = \max \text{GPA in Student}} \ (\text{Student}) \]

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

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>214</td>
<td>Milhouse</td>
<td>12</td>
<td>3.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>146</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

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

<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>1.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>CPS216</td>
</tr>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
<td>CPS214</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>1.1</td>
<td>CPS216</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>1.1</td>
<td>CPS214</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>1.1</td>
<td>CPS216</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>1.1</td>
<td>CPS214</td>
</tr>
</tbody>
</table>
A note on column ordering

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

<table>
<thead>
<tr>
<th>ID</th>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>142</td>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>ID</th>
<th>Student.SID</th>
<th>Enroll.SID</th>
<th>Enroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10 2.3</td>
<td>142 CPS216</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10 3.1</td>
<td>123 CPS216</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Student \( \bowtie_p \) Student.SID = Enroll.SID Enroll
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 $\pi_L \left( R \bowtie S \right)$**
  - $L$ is the union of all attributes from $R$ and $S$, with duplicates removed
  - $\pi$ equates all attributes common to $R$ and $S$

Natural join example

- $\text{Student} \bowtie \text{Enroll} = \pi_{\text{SID}, \text{name}, \text{age}, \text{GPA}, \text{CID}} \left( \text{Student} \bowtie \text{Enroll} \right)$

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** $S - (S - R)$

### 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 column names for natural joins
Renaming example

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

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 expressive 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

A trickier exercise

- SID's of students who take exactly one course

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, \( R \subseteq R' \) implies \( \text{RelOp}(R) \subseteq \text{RelOp}(R') \)
Classification of relational operators

- Selection: $\sigma_p(R)$
- Projection: $\pi_L(R)$
- Cross product: $R \times S$
- Join: $R \bowtie S$
- Natural join: $R \bowtie S$
- Union: $R \cup S$
- Difference: $R - S$
- Intersection: $R \cap S$

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
- 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
- Cross product
- Union
- 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 whose semantics are easy to grasp.

- **Complete?**
  - With respect to what?

---

Relational calculus

- \{ s.SID | s ∈ Student ∧ ¬(∃s' ∈ Student: s.GPA < s'.GPA) \}, or
- \{ s.SID | s ∈ Student ∧ (∀s' ∈ Student: s.GPA ≥ s'.GPA) \}

- Relational algebra = “safe” relational calculus
  - Every query expressible as a safe relational calculus query is also expressible as a relational algebra.
  - And vice versa.

- Example of an unsafe relational calculus query
  - \{ s.name | ¬(s ∈ 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.