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)
  - Duplicate tuples are 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)

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), ... }
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>

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 of the input table
  - Example: student with the highest GPA
    $\sigma_{\text{GPA} \geq \text{all GPA in Student}} (\text{Student})$

Projection

- Input: a table $R$
- Notation: $\pi_L (\text{R})$
  - $L$ 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>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>
More on projection

- Duplicate output rows must be removed
  - Example: student ages
    \[ \pi_{\text{name}} (\text{Student} ) \]

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 (so 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
  - \( \text{Student} \bowtie_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll} \)
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 duplicate attributes removed
  - \( \pi \) equates all attributes common to \( R \) and \( S \)

**Natural join example**

**Student** \( \bowtie \) **Enroll** = \( \pi_L( \text{Student} \bowtie \text{Enroll} ) \)

\[ = \pi_{\text{SID}, \text{name}, \text{age}, \text{GPA}, \text{CID}}( \text{Student} \bowtie \text{Enroll} ) \]

**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:** \( \rho_S( R ) \), 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

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

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

\[ \rho_{\text{Enroll} \bowtie (\text{SID}_1, \text{CID}_1)} \]

\[ \rho_{\text{Enroll} \bowtie (\text{SID}_2, \text{CID}_2)} \]

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 expressive power

Summary of derived operators

- Join: \( R \bowtie S \)
- Natural join: \( R \bowtie \bowtie S \)
- Intersection: \( R \cap S \)

- Many more
  - Semijoin, anti-semijoin, quotient, …

An exercise

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

\[ \sigma_{\text{name} = "Lisa"}(\text{Enroll}) \]

\[ \pi_{\text{CID}}(\text{Course}) \]

\[ \rho_{\text{Student}_1, \text{SID}} \]

Another exercise

- Names of students in Lisa’s classes

\[ \sigma_{\text{name} = "Lisa"}(\text{Enroll}) \]

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

A trickier exercise

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

\[ \rho_{\text{Student}_1, \text{GPA} < \text{Student}_2, \text{GPA}} \]

\[ \rho_{\text{Student}_1} \]

\[ \rho_{\text{Student}_2} \]
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: $\sigma_f(R)$ 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: $R - S$ Non-monotone (not 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 4.3
  - Add another GPA 4.5
  - 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 proof?
- 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
  - But operators are "procedural"
- Complete?
  - With respect to what?

Relational calculus

- $\{ s.SID \mid s \in \text{Student} \land 
  \neg(\exists' \in \text{Student} : s.GPA < s'.GPA) \}$, or
- $\{ s.SID \mid s \in \text{Student} \land 
  (\forall' \in \text{Student} : s.GPA \geq s'.GPA) \}$
- 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
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