Relational Algebra

Relational algebra is a set of 6 operators that act on tables to produce tables. Just as we operate on numbers with arithmetic, we operate on tables with relational algebra.

Key to understanding SQL and query processing, optimization

- SQL is, roughly speaking, a generalization of relational algebra
- Internal languages: an SQL query is rewritten as a relational-algebra expression, which can in turn be rewritten into a more efficient form and evaluated using a bunch of well-developed algorithms
Queries and query languages

- **Query**: a question about the data in a database.
  A statement requesting the retrieval of information from a database.
  Example: find the names of students who are taking CS2.

- **Query language**: language in which queries are expressed.
  Query languages <-> programming languages!
  - QLs are not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Preliminaries

- Review: a relational database is a collection of tables.
- A query is applied to relation instances, and the result of a query is also a relation instance.
  - Schemas of input relations for a query are fixed (the query will run regardless of instances!)
  - The schema for the result of a given query is also fixed!
  - Determined by the query.

Example schemas:
- **Students** *(sid: string, sname: string, gpa: real)*
- **Courses** *(cid: string, cname: string, credit: integer, teacher: string)*
- **Enroll** *(sid: string, cid: string, grade: string)*
Example instances

S:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>2.8</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

E:

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
<tr>
<td>003</td>
<td>166</td>
<td>A</td>
</tr>
<tr>
<td>003</td>
<td>CS2</td>
<td>A</td>
</tr>
</tbody>
</table>

C:

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>166</td>
<td>math</td>
<td>3</td>
<td>poe</td>
</tr>
<tr>
<td>CS2</td>
<td>db</td>
<td>4</td>
<td>fan</td>
</tr>
</tbody>
</table>

Question: both John and Joe have a GPA of 3.0. Is S legal?

Relational Algebra

A set of operations (functions), each of which takes a relation (or relations) as input and produces a relation as output.

Basic operations: using these we can build up sophisticated database queries.

- Projection
- Selection
- Union
- Difference
- Product
- Renaming

Additional operations: intersection, join, division.
Projection

Given a list of column names $A$ and a relation $R$, $\pi_A(R)$ extracts the columns in $A$ from the relation.

Example: given $S$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
</tbody>
</table>

$\pi_{\text{sid}, \text{gpa}}(S)$ is

<table>
<thead>
<tr>
<th>sid</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Question:
- What is the schema of the result? Recall $S$ has schema $\text{Students (sid: string, sname: string, gpa: real)}$.
- What is the query (in English)?

Projection - continued

Suppose the result of $\pi_A(R)$ has duplicate values.

Example: given $S1$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

$\pi_{\text{gpa}}(S1)$ is

<table>
<thead>
<tr>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
</tr>
<tr>
<td>2.8</td>
</tr>
<tr>
<td>4.0</td>
</tr>
</tbody>
</table>

- In relational algebra, the answer is always a set (has to eliminate duplicates).
- However, SQL and some other languages return, by default, a bag (don’t eliminate duplicates).
### Selection

Given a condition $C$ and a relation $R$, $\sigma_C(R)$ extracts those rows from the relation $R$ that satisfy $C$.

**Example:** given $S$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**Question:**
- What is the schema of the result? Recall $S$ has schema $\text{Students}$ (sid: string, sname: string, gpa: real)
- What is the query to find (in English)?

$\sigma_{\text{gpa} \geq 3.0}(S)$ is

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**What can go into a condition?**

Condition $C$ in $\sigma_C(R)$ is built up from

- Boolean operations on the field names: $<$, $<=$, $=$, $\neq$, $>$, $\leq$, $\geq$.
  
  Example: $\text{gpa} \geq 3.0$, $\text{sname} = \text{"john"}$.
- Predicates constructed from these using $\land$ (and), $\lor$ (or), $\neg$ (not).

**Question:** what is the result of $\sigma_{\text{gpa} \geq 3.0 \land \text{sname} = \text{"john"}}(S)$?

given $S_1$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>john</td>
<td>2.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Set operations

- Set operations: $S_1 \cup S_2$, $S_1 - S_2$, $S_1 \cap S_2$.
- What are these?
- **Condition**: all these operations must be union-compatible:
  - Same number of fields (same arity)
  - ‘Corresponding’ fields have the same domain (same type).

Union-compatibility is a database terminology.

What is the corresponding programming language terminology?
- Having the same type!

Question: recall $S$ and $C$ given above. Can we write $S \cup C$?
- Why?

- What is the schema of the result of a set operation?

---

Set operations – union

**Given $S_1$:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**$S_1 \cup S_2$ is:**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
<tr>
<td>005</td>
<td>peter</td>
<td>4.0</td>
</tr>
</tbody>
</table>
## Set operations – set difference

Given S1:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

S1 – S2 is:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Question: what is S2 – S1?

## Set operations – intersection

Given S1:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
<tr>
<td>004</td>
<td>john</td>
<td>3.0</td>
</tr>
</tbody>
</table>

S1 ∩ S2 is:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Set operations – intersection

In relational algebra, basic set operations are union and set difference only. It turns out that we can implement the other set operations using those basic operations. For example, for any relations (sets) $S_1$ and $S_2$, we can already express $S_1 \cap S_2$.

How?

$$S_1 \cap S_2 = S_1 - (S_1 - S_2)$$

However, we have to be careful. Although it is mathematically nice to have fewer operators, operations like set differences may be less efficient than intersection.

Product

Product $R \times S$ connects two relations $R$ and $S$ that are not necessarily union compatible.

Example: given $S$:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Example: given $C$:

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>166</td>
<td>math</td>
<td>3</td>
<td>poe</td>
</tr>
<tr>
<td>CS2</td>
<td>db</td>
<td>4</td>
<td>fan</td>
</tr>
</tbody>
</table>
**S × C** is:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
<th>cid</th>
<th>cname</th>
<th>credits</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
<td>166</td>
<td>math</td>
<td>3</td>
<td>poe</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>166</td>
<td>math</td>
<td>3</td>
<td>poe</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
<td>166</td>
<td>math</td>
<td>3</td>
<td>poe</td>
</tr>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
<td>CS2</td>
<td>db</td>
<td>4</td>
<td>fan</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>CS2</td>
<td>db</td>
<td>4</td>
<td>fan</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
<td>CS2</td>
<td>db</td>
<td>4</td>
<td>fan</td>
</tr>
</tbody>
</table>

---

**Cartesian product**

- Each row of S is paired with each row of R.
- Schema of the result has one field per field of S and R.
  - Example: the schema of S × C is
    
    $(\text{sid}: \text{string}, \  \text{sname}: \text{string}, \  \text{gpa}: \text{real}, \  \\
    \text{cid}: \text{string}, \  \text{cname}: \text{string}, \  \text{credits}: \text{integer}, \  \\
    \text{instructor}: \text{string})$

  - Question: what is the primary key of R × S in general?
  - Cardinality. Suppose that S has n rows and R has m rows. What is the cardinality of R × S?
Product – continued

What happens when we form a product of two relations with columns having the same name?

Example: given S

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
</tr>
</tbody>
</table>

E:

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
</tbody>
</table>

Details may vary among systems/formalisms but a common answer is to suffix the attribute names with 1 and 2.

Example: S × E is

<table>
<thead>
<tr>
<th>sid:1</th>
<th>sname</th>
<th>gpa</th>
<th>sid:2</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
<tr>
<td>003</td>
<td>grace</td>
<td>4.0</td>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
</tbody>
</table>
Product – continued

Products are hardly used alone; they are typically used in conjunction with a selection.

Example: \( \sigma_{\text{sid:1} = \text{sid:2} \land \text{cid} = \text{CS2}} (S \times E) \)

<table>
<thead>
<tr>
<th>sid:1</th>
<th>sname</th>
<th>gpa</th>
<th>sid:2</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
</tbody>
</table>

Question: what does this query do (in English)?

Suppose we want to find the names and grades of students who are taking CS2. How to write the query?

\( \pi_{\text{sname, grades}} (\sigma_{\text{sid:1} = \text{sid:2} \land \text{cid} = \text{CS2}} (S \times E)) \)

<table>
<thead>
<tr>
<th>sname</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>mary</td>
<td>C</td>
</tr>
</tbody>
</table>

Question: why is this possible?

Joins – conditional join

The combination of a selection and a join is so common that we give it a special symbol (and name).

\( R \bowtie_{c} S \) is defined to be \( \sigma_{c} (R \times S) \)

Example: \( S \bowtie_{\text{sid:1} = \text{sid:2}} E \)

<table>
<thead>
<tr>
<th>sid:1</th>
<th>sname</th>
<th>gpa</th>
<th>sid:2</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>joe</td>
<td>3.0</td>
<td>001</td>
<td>166</td>
<td>B</td>
</tr>
<tr>
<td>002</td>
<td>mary</td>
<td>2.8</td>
<td>002</td>
<td>CS2</td>
<td>C</td>
</tr>
</tbody>
</table>

Questions:
- What is the result schema?
- Conditional join is in general more efficient than cross product. Why?
- The condition \( C \) in a conditional join is usually an equality or conjunction of equalities.
natural join

\( R \bowtie S \): a ‘special case’ of conditional join: equality on common fields of \( R \) and \( S \)
- Equality condition only
- On all common fields
- Leave only one copy of these fields in the resulting relation.

\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
1 & 2 & a \\
1 & 2 & b \\
1 & 3 & c \\
2 & 1 & g \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|}
\hline
A & B & D \\
\hline
1 & 2 & d \\
1 & 2 & e \\
1 & 4 & d \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
1 & 2 & a & d \\
1 & 2 & a & e \\
1 & 2 & b & d \\
1 & 2 & b & e \\
\hline
\end{array}
\]

Question: what if \( R \) and \( S \) have no fields in common? -- Cartesian product!

Example – natural join

Example: given \( S \):

\[
\begin{array}{|c|c|c|}
\hline
sid & sname & gpa \\
\hline
001 & joe & 3.0 \\
002 & mary & 2.8 \\
003 & grace & 4.0 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|}
\hline
sid & cid & grade \\
\hline
001 & 166 & B \\
002 & CS2 & C \\
\hline
\end{array}
\]

Question: what is \( S \bowtie E \)?

Answer:

\[
\begin{array}{|c|c|c|c|c|}
\hline
sid & sname & gpa & cid & grade \\
\hline
001 & joe & 3.0 & 166 & B \\
002 & mary & 2.8 & 002 & CS2 & C \\
\hline
\end{array}
\]

Question: what is \( S \bowtie C \) where \( C \) is an instance of

\( Courses (cid: \text{string}, \ cname: \text{string}, \ credits: \text{integer}, \ instructor: \text{string}) \)?
Example – queries (1)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

1. Find the names of Students.
   \( \pi_{\text{sname}}(\text{Students}) \)
2. Find the courses taught by Fan.
   \( \sigma_{\text{instructor} = \text{“fan”}}(\text{Courses}) \)
3. Find the titles of courses taught by Fan.
   \( \pi_{\text{cname}}(\sigma_{\text{instructor} = \text{“fan”}}(\text{Courses})) \)
✓ These queries involve a single relation
✓ The result of a query is also a relation and therefore, can be used as input of another query

Example – queries (2)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the names of Students who are taking CS2.
✓ Two relations. Therefore, use (natural) join or product
✓ Fields: projection
✓ Condition: selection

Solutions:
1. \( \pi_{\text{sname}}(\sigma_{\text{cid} = \text{“CS2”}}(\text{Students} \bowtie \text{Enroll})) \)
2. \( \pi_{\text{sname}}(\text{Students} \bowtie \sigma_{\text{cid} = \text{“CS2”}}(\text{Enroll})) \)
renaming – another operator (1)

renaming operator:

- \( \rho (R, E) \): define temporary relation \( R \) to hold the result of \( E \)
- \( \rho (R (A \rightarrow B), E) \): in addition, rename \( A \) attribute in \( E \) as \( B \).

Example. Find the names of Students who are taking CS2.

\[ \pi_{sname} (\ Students \ \bowtie\ \sigma_{cid = "CS2"} (Enroll)) \]

is equivalent to:

\[ \rho (\ temp1, \sigma_{cid = "CS2"} (Enroll)) \]
\[ \rho (\ temp2, \ Students \ \bowtie\ temp1) \]
\[ \pi_{sname} (\ temp2) \]

renaming – another operator (2)

Consider Students \( \times \) Enroll. Note we have name conflict.

\( (sid1 \ string, \ sname: string, \ gpa: real, \ sid2 \ string, \ cid: string, \ grade: string) \)

To eliminate name conflict, we can do the following:

\[ \rho (\ temp1 \ (sid \rightarrow sid1), \ Students) \]
\[ \rho (\ temp2 \ (sid \rightarrow sid2), \ Enroll) \]
\[ temp1 \times temp2 \]
Example – queries (3)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the names of Students who are taking a course taught by Fan.

1. Information about instructor only available in Courses; so we need an extra join. Solutions:
   1. $\pi_{\text{sname}}(\text{Students} \bowtie \text{Enroll} \sigma_{\text{instructor} = \text{fan}}(\text{Courses}))$
      Remark. Associative!
      \[
      R \bowtie S \bowtie T = (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)
      \]
   2. A more efficient solution:
      $\pi_{\text{sname}}(\text{Students} \bowtie \text{Enroll}(\pi_{\text{sid}}(\text{Enroll} \sigma_{\text{instructor} = \text{fan}}(\text{Courses}))))$
      Remark. A query optimizer can find this given the first solution.

Exercises – queries

1. Find the GPAs of Grace.
2. Find the ids of the courses being taken by Grace.
3. Find the instructors of the courses being taken by Grace.
4. Find the names of students who are taking at least one course.

Solutions:
1. $\pi_{\text{gpa}}(\sigma_{\text{sname} = \text{grace}}(\text{Students}))$
2. $\pi_{\text{cid}}((\sigma_{\text{sname} = \text{grace}}(\text{Students}) \bowtie \text{Enroll}))$
3. $\pi_{\text{instructor}}(\text{Courses} \bowtie \text{Enroll}(\sigma_{\text{sname} = \text{grace}}(\text{Students})))$
4. $\pi_{\text{sname}}(\text{Students} \bowtie \text{Enroll(Enroll(\sigma_{\text{sname} = \text{grace}}(\text{Courses}))))}$
Example – queries (4)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the names of Students who are taking a course taught by Poe or Fan.

1. $\pi_{sname}(\text{Students} \bowtie \text{Enroll}\bowtie (\sigma_{\text{instructor} = \text{"poe"}} \lor \text{instructor} = \text{"fan"}} (\text{Courses}))$

How to do this using union?

- $\rho(\text{temp}, \sigma_{\text{instructor} = \text{"poe"}} (\text{Courses}) \cup \sigma_{\text{instructor} = \text{"fan"}} (\text{Courses}))$
- $\pi_{sname}(\text{Students} \bowtie \text{Enroll} \bowtie \text{temp})$

Example – queries (5)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the ids of Students who are taking a course taught by Poe and a course taught by Fan.

1. $\pi_{sid}(\text{Enroll} \bowtie \sigma_{\text{instructor} = \text{"poe"}} (\text{Courses})) \cap \pi_{sid}(\text{Enroll} \bowtie \sigma_{\text{instructor} = \text{"fan"}} (\text{Courses}))$

The following is incorrect!

- $\pi_{sid}(\text{Enroll} \bowtie (\sigma_{\text{instructor} = \text{"poe"}} \land \text{instructor} = \text{"fan"}} (\text{Courses}))$

Question: Find the names of Students who are taking a course taught by Poe and a course taught by Fan.

$\pi_{sname}(\text{Students} \bowtie \pi_{sid}(\text{Enroll} \bowtie \sigma_{\text{instructor} = \text{"poe"}} (\text{Courses})) \cap \pi_{sid}(\text{Enroll} \bowtie \sigma_{\text{instructor} = \text{"fan"}} (\text{Courses})))$
Example – queries (6)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the sids of Students who are not taking CS2.
Solution:

\[ \pi_{\text{sid}}(\text{Students}) - \pi_{\text{sid}}(\sigma_{\text{cid} = \text{"CS2"}}(\text{Enroll})) \]

Exercise: Find the names of Students who are not taking CS2.
Solution:

\[ \rho(\text{temp}, \pi_{\text{sid}}(\text{Students}) - \pi_{\text{sid}}(\sigma_{\text{cid} = \text{"CS2"}}(\text{Enroll}))) \]
\[ \pi_{\text{sname}}(\text{temp} \bowtie \text{Students}) \]

Division – introduction

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

Find the sids of students who are taking all courses.

1. Build a relation with all possible pairs of sids and cids.
   Let Allpairs be \( \pi_{\text{sid}}(\text{Enroll}) \times \pi_{\text{cid}}(\text{Courses}) \)
2. Find the set of (sid, cid) pairs for which sid is not taking cid.
   Let temp1 be Allpairs \( - \pi_{\text{sid}, \text{cid}}(\text{Enroll}) \)
3. Find the set of sids of students who are not taking some course.
   Let temp2 be \( \pi_{\text{sid}}(\text{temp1}) \)
4. Find the set of sids of students who are taking all courses.
   \( \pi_{\text{sid}}(\text{Enroll}) - \text{temp2} \)
Division – another operator

Find the sids of students who are taking all courses.

\[ \pi_{\text{sid, cid}}(\text{Enroll}) \div \pi_{\text{cid}}(\text{Courses}) \]

In general, \( A / B \).

- The schema of B must be a proper subset of the schema of A, i.e., \( B \subset A \) (check: \( \{\text{cid}\} \subset \{\text{cid, sid}\} \))
- The schema of the result is \( A - B \), i.e., the set difference of the schema of A and the schema of B (check: result is \( \{\text{sid}\} \))
- For every tuple \( t \) in the result and every tuple \( s \) in B, \( ts \) (\( t \) appended onto \( s \)) is in the first relation A.

Division – examples of \( A/B \)
Example – queries (7)

Students (sid: string, sname: string, gpa: real)
Courses (cid: string, cname: string, credits: integer, instructor: string)
Enroll (sid: string, cid: string, grade: string)

1. Find the names of students who are taking all courses.

\[ \rho (\text{tempid}, \pi_{\text{sid, cid}}(\text{Enroll}) \sqcap \pi_{\text{cid}}(\text{Courses})) \]

\[ \pi_{\text{sname}}(\text{Students} \bowtie \text{tempid}) \]

1. Find the names of students who are taking all courses taught by Fan.

\[ \rho (\text{tempid}, \pi_{\text{sid, cid}}(\text{Enroll}) \sqcap \pi_{\text{cid}}(\sigma_{\text{instructor = “fan”}}(\text{Courses}))) \]

\[ \pi_{\text{sname}}(\text{Students} \bowtie \text{tempid}) \]

What we can’t compute with relational algebra?

- **Arithmetic expressions**, e.g., 4 + 3.
- **Aggregate operations**, e.g., “the number of students who are taking CS2”, or “the average GPA of students”.
  
  In SQL, these are **possible** – SQL has numerous extensions to relational algebra.
- **Recursive** queries, e.g., given a relation Parent(parent, child), compute the ancestor relation. This appears to call for an arbitrary number of joins.
  
  These are **not possible** in SQL either.
- **Complex structures**, e.g., lists, arrays, bags, or nested relations.
  
  SQL can’t handle complex structures either, but they are possible in object-oriented data models and query languages.
Summary

What are query languages?

Relational algebra. A set of operations (functions), each of which takes a relation (or relations) as input and produces a relation as output.

Basic operations: using these we build up queries.

- Projection
- Selection
- Union
- Difference
- Product
- Renaming.

Additional operations: intersection, join, Very useful.

What we cannot do with relational algebra?

SKILL: “programming” in relational algebra!