The Relational Algebra (RA)

- Mathematical query language
- Introduced by Edgar Codd
- Since invention, developed and studied by Codd and many others

RA Example

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>year</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>

Outline

- Background
- The Primitive Operators
- Implied Operators
  - Joins
  - Division
- Equivalence & Independence
- Taste of Query Optimization

The Relational Model

- A conceptual model for representing data, integrity constraints, and queries
  - All based on the notion of a schema
- DBMS is responsible for translating specifications into the physical environment at hand
  - Storage in files, caches, indexes
  - Queries translated to query plans (high-level imperative programs)
  - Query plans translated to low-level execution over stored data
Why RA?

- Understanding the relational algebra is a key understanding of central concepts in databases: SQL, query evaluation, query optimization
- Tool for building theoretical foundations of various query languages (e.g., SQL)
- Tool for developing novel data/query models

RA vs Other QLs

- Some subtle (yet important) differences between RA and other languages
  - Can tables have duplicate records?
    - (RA vs. SQL)
  - Are missing (NULL) values allowed?
    - (RA vs. SQL)
  - Is there any order among records?
    - (RA vs. SQL)
  - Is the answer dependent on the domain from which values are taken (not just the DB)?
    - (RA vs. RC)
- For RA, the answer to all questions is “no”
  - At least in the “textbook” model we study here

Relation Schema

- A relation schema is a finite sequence of distinct attribute names $\text{att}$ with a mapping of each to a domain $\text{dom}$ of legal values
- Notation: $(\text{att}_1: \text{dom}_1, \ldots, \text{att}_k: \text{dom}_k)$
- Example: $(\text{sid}: \text{int}, \text{name}: \text{string}, \text{year}: \text{int})$

Tuples

- Let $s$ be a relation schema $(\text{att}_1: \text{dom}_1, \ldots, \text{att}_k: \text{dom}_k)$
- A tuple (over $s$) is a sequence $(v_1, \ldots, v_k)$ of values $v_i$, where each $v_i$ is in $\text{dom}_i$
  - That is, a tuple is an element of $\text{dom}_1 \times \ldots \times \text{dom}_k$

Ignoring Domains

- In this lecture we ignore the attribute domains, since they play no special role
  - (Well, almost; they make a difference for query equivalence, but we do not get there…)
- For example, we will write $(\text{sid}, \text{name}, \text{year})$ instead of $(\text{sid}: \text{int}, \text{name}: \text{string}, \text{year}: \text{int})$
Notation

- Notation 1:
  - Let $R$ be a relation with the header $(\text{att}_1, \ldots, \text{att}_k)$
  - Let $t=(v_1, \ldots, v_k)$ be a tuple of $R$
  - We refer to $v_i$ by $t.\text{att}_i$

- Notation 2:
  - Let $a_1, \ldots, a_m$ be attributes among $\text{att}_1, \ldots, \text{att}_k$
  - We denote by $t[a_1, \ldots, a_m]$ the tuple $(t.a_1, \ldots, t.a_m)$

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

$\text{t.sid} = 861$
$\text{t.name} = \text{Alma}$
$\text{t[sid, name]} = (861, \text{Alma})$
$\text{t[name, sid, year]} = (\text{Alma}, 861, 2)$

Databases

- A database schema is a finite set of relation names, each mapped to a relation schema
  - Example: Student(sid, name, year), Course(cid, topic), Studies(sid, cid)

- A database (or instance) over a database schema associates with each relation schema a relation over that schema

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>year</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
</tr>
<tr>
<td>cid</td>
<td>topic</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td>861</td>
</tr>
<tr>
<td>76</td>
<td>05</td>
<td>753</td>
</tr>
<tr>
<td>861</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>76</td>
<td></td>
</tr>
</tbody>
</table>

What is “Algebra”?

- An abstract algebra consists of:
  - A class of elements
  - A collection of operators
- Each operator:
  - Has an arity $d$
  - Has a domain of sequences $(e_1, \ldots, e_d)$ of elements
  - Maps every sequence in its domain to an element $e$
- The definition of an operator allows for composition:
  - $o_1(o_2(x), o_3(y))$
  - Examples:
    - Ring of integers: $(a+b)$
    - Boolean algebra: $(\text{true}, \text{false}, \neg, \land, \lor)$
    - Relational algebra

The Relational Algebra

- In the relational algebra (RA) the elements are relations
  - Recall: a relation is a pair $(s, r)$
  - RA has 6 primitive operators:
    - Unary: projection, selection, renaming
    - Binary: union, difference, Cartesian product
- Each of the six is essential (independent)—we cannot define it using the others
  - We will see what exactly this means and how this can be proved
- We commonly allow many more useful operators that can be defined using the primitive ones
  - For example, intersection via union and difference

Outline

- Background
  - The Primitive Operators
  - Implied Operators
    - Joins
    - Division
  - Equivalence & Independence
  - Taste of Query Optimization

6 Primitive (Basic) Operators

1. Projection ($\pi$)
2. Selection ($\sigma$)
3. Renaming ($\rho$)
4. Union ($\cup$)
5. Difference ($\setminus$)
6. Cartesian Product ($\times$)
Phrase a query that finds the names of students who get private lessons (i.e., the student takes a course that no one else takes)

\[
\pi_{\text{sid, name}}(R) = \pi_{\text{year}}(R) = \{ \text{Alma}, \text{Ahuva} \}
\]

**Definition of Projection**

- Projection is a unary operator of the form \( \pi_{A_1, \ldots, A_k} \)
  - where each \( A_i \) is an attribute name
- Legal input: relation \( R \) in with attributes \( A_1, \ldots, A_k \) (and possibly others)
- \( \pi_{A_1, \ldots, A_k}(R) \) is the relation \( S \) with:
  - Header \( (A_1, \ldots, A_k) \)
  - Tuple set \( \{ t[A_1, \ldots, A_k] \mid t \in R \} \)

**Q**: If \( R \) has 1000 tuples, how many tuples can \( \pi_{A_1, \ldots, A_k}(R) \) have?

**Selection by Example**

\[
\sigma_{\text{course}=\text{DB}}(R) = \sigma_{\text{year}=1 \land \text{grade}>84}(R) = \{ \text{Alma}, \text{Ahuva} \}
\]

**Definition of Selection**

- Selection is a unary operator of the form \( \sigma_c \) where \( c \) is a logical condition (selection predicate) on attributes
  - \( c \) consists of comparisons and logical connectors (\( \land, \lor, \neg \))
  - e.g., \( \text{price} \geq 500 \land \text{price} \leq \text{budget} \)
- Legal input: A relation with all the attributes mentioned in the selection predicate
- The condition is applied to each tuple in the input, and each violating tuple is filtered out
- Formally, \( \sigma_c(R) \) is the relation \( S \) with the header of \( R \) and the tuple set \( \{ t \mid t \in R \land t \models c \} \)

**Q**: If \( R \) has 1000 tuples, how many tuples can \( \sigma_c(R) \) have?

**Variants of Selection**

- Various variants of RA may allow different languages for specifying selection predicates
  - e.g., \( c^2 > a^2 + b^2 \); name starts with ‘A’, etc.
- Common to all predicate formalisms: a predicate applies to a single tuple
  - Cannot state cross-tuple conditions, e.g.,
    - “there is another tuple with the same name”
    - “contains at least 100 tuples”
Renaming by Example

\[ R = \{\text{student, year, course, grade}\} \]

\[ \rho_{\text{year/level}}(R) = \{\text{student, level, course, grade}\} \]

Definition of Renaming

- Renaming is a unary operator of the form \( \rho_{A/B} \) where \( A \) and \( B \) are attribute names.
- **Legal input:** A relation with a header that contains \( A \) and does not contain \( B \).
- Renaming changes only the header: attribute \( A \) becomes \( B \).
- Formally, \( \rho_{A/B}(R) \) is the relation \( S \) with:
  - The header of \( R \) having \( A \) replaced by \( B \).
  - The tuple set of \( R \).

Q: If \( R \) has 1000 tuples, how many tuples can \( \rho_{A/B}(R) \) have?

Union and Difference by Example

\[ R = \{\text{student, year}\} \]

\[ S = \{\text{student, year}\} \]

\[ R \cup S = \{\text{student, year}\} \]

\[ R \setminus S = \{\text{student, year}\} \]

Definition of Union and Difference

- Binary operators, interpreted as operations over the tuple sets.
- **Legal input:** a pair of relations \( R \) and \( S \) with the exact same header.
- We then say that \( R \) and \( S \) are union compatible.
- Formally:
  - \( R \cup S \) is the relation with the header of \( R \) (and \( S \)) and the union of the tuple sets.
  - \( R \setminus S \) is the relation with the header of \( R \) (and \( S \)) and the difference between the tuple sets.

Q: If each of \( R \) and \( S \) have 1000 tuples, how many tuples can be in \( R \cup S \)? \( R \setminus S \)?

Cartesian Product by Example

\[ R = \{\text{sid, name, year}\} \]

\[ S = \{\text{cid, topic}\} \]

\[ R \times S = \{\text{sid, name, year, cid, topic}\} \]

Definition of Cartesian Product

- Binary operator, similar to set product, but each output pair is combined into a single tuple.
- **Legal input:** A pair of relations with disjoint sets of attributes.
- So how to cross-product \( \text{Mom(ssn)} \) with \( \text{Dad(ssn)} \)?
- Formally, let \( R \) and \( S \) have the headers \( (A_1, \ldots, A_k) \) and \( (B_1, \ldots, B_m) \), respectively; then \( R \times S \) is the relation \( T \) with:
  - Header \( (A_1, \ldots, A_k, B_1, \ldots, B_m) \).
  - Tuple set \( \{r \circ s \mid r \in R \text{ and } s \in S\} \).

Q: If each of \( R \) and \( S \) has 1000 tuples, how many tuples can be in \( R \times S \)?
Shorthand Notation

For Cartesian product of named relations (e.g., \( R \), \( S \)), we actually allow common attributes, and implicitly assume their renaming to \( \text{name.attribute} \).

\[
\begin{align*}
R &= \begin{array}{lll}
861 & \text{Alma} & 2 \\
753 & \text{Amir} & 1 \\
955 & \text{Ahuva} & 2 \\
\end{array} \\
S &= \begin{array}{lll}
861 & 23 \\
753 & 45 \\
\end{array} \\
R \times S &= \begin{array}{lll}
861 & \text{Alma} & 23 \\
753 & \text{Amir} & 45 \\
955 & \text{Ahuva} & 45 \\
\end{array}
\]

Parentheses Convention

- We have defined 3 unary operators and 3 binary operators.
- It is acceptable to omit the parentheses from \( o(R) \) when \( o \) is unary.
- Then, unary operators take precedence over binary ones.
- Example:

\[
(\sigma_{\text{topic=\text{DB}}}(\text{Course})) \times (\rho_{\text{cid} / \text{sid}}(\text{Studies}))
\]

becomes

\[
\sigma_{\text{topic=\text{DB}}}(\text{Course}) \times \rho_{\text{cid} / \text{sid}}(\text{Studies})
\]

Composition Example

\[
\pi_{\text{name}}(\pi_{\text{topic}}(\text{Student} \times \pi_{\text{topic}}(\text{Course} \times \pi_{\text{topic}}(\text{Studies}))))
\]

Names of students who study DB:

\[
\begin{array}{lll}
\text{Student} & \text{Course} & \text{Studies} \\
\hline
861 & \text{Alma} & 23 \\
753 & \text{Amir} & 45 \\
955 & \text{Ahuva} & 45 \\
\end{array}
\]

\[
\begin{array}{lll}
\text{Student} & \text{Course} & \text{Studies} \\
\hline
861 & \text{PL} & 70 \\
753 & \text{DB} & 70 \\
955 & \text{OS} & 70 \\
\end{array}
\]

\[
\begin{array}{lll}
861 & 23 & 861 \\
753 & 45 & 753 \\
955 & 45 & 955 \\
\end{array}
\]

\[
\begin{array}{lll}
861 & 23 & 23 \\
753 & 45 & 45 \\
955 & 76 & 76 \\
\end{array}
\]

\[
\begin{array}{lll}
861 & 23 & 23 \\
753 & 45 & 45 \\
955 & 76 & 76 \\
\end{array}
\]

\[
\begin{array}{lll}
861 & 23 & 23 \\
753 & 45 & 45 \\
955 & 76 & 76 \\
\end{array}
\]

\[
\begin{array}{lll}
861 & 23 & 23 \\
753 & 45 & 45 \\
955 & 76 & 76 \\
\end{array}
\]
### Student Table

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>861 Alma 2</td>
<td>23 PL</td>
<td>861 23</td>
</tr>
<tr>
<td>753 Amir 1</td>
<td>45 DB</td>
<td>861 45</td>
</tr>
<tr>
<td>955 Ahuva 2</td>
<td>76 OS</td>
<td>753 45</td>
</tr>
</tbody>
</table>

### Course Table

<table>
<thead>
<tr>
<th>Course</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL DB</td>
<td>45</td>
</tr>
</tbody>
</table>

### Studies Table

<table>
<thead>
<tr>
<th>Studies</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL OS</td>
<td>23</td>
</tr>
</tbody>
</table>

### SQL Query

```sql
SELECT * FROM Student
JOIN Course ON Student.sid = Course.cid
JOIN Studies ON Course.id = Studies.cid
```
\[
\pi_{\text{name}}(\pi_{\text{sid}=1}(\pi_{\text{name}}(\text{Student} \times \pi_{\text{topic}='DB'}(\text{Course} \times \pi_{\text{sid}=1}(\text{Studies}))))))
\]

**Outline**

- Background
- The Primitive Operators
  - Implied Operators
    - Joins
    - Division
    - Equivalence & Independence
    - Taste of Query Optimization

**Task**

Phrase a query that finds the names of students who get private lessons (i.e., the student takes a course that no one else takes)

**Implied Operators**

- We now discuss relational operators that are:
  - Not among the 6 basic operators
  - Can be expressed in RA (implied)
  - Very common in practice
- Good to have!
  - Easier to write queries
  - Easier to understand/maintain queries
  - Easier for DBMS to apply specialized optimizations

**Joins**

- Cartesian product is rarely standalone without selection, and is commonly followed by projection
- The combination \( \pi \sigma \times \) is referred to generally as “join”
- There are several common cases that apply specific selections and projections, which we introduce here
Conditional Join

- Binary operator $R \bowtie_c S$ where $c$ is a condition over the header of $R \times S$
- Shorthand notation for: $\sigma_c(R \times S)$
- Example: $R \bowtie_{a=b \land c<d} S$

Theta Join and Equijoin

- Theta join is a special case of conditional join $\bowtie_c$ where $c$ has the form $A \theta B$ or $A \theta v$ where $A$ and $B$ are attributes, $\theta$ a comparison operator
  - Example: $R \bowtie_{a=b} S$
- Equijoin is the special case where $c$ has the form $A = B$ where $A$ and $B$ belong to the left and right operands, respectively
  - Example: Course$\bowtie_{name=course} Studies$

Equijoin Example

$S = \begin{array}{ccc}
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2
\end{array}$

$T = \begin{array}{ccc}
861 & PL \\
762 & OS \\
955 & OS
\end{array}$

$S\bowtie_{sid=stud}T = \begin{array}{ccc}
861 & Alma & 2 & 861 & PL \\
861 & Alma & 2 & 861 & DB \\
955 & Ahuva & 2 & 955 & OS
\end{array}$

Natural Join $\bowtie$

- Cartesian product, equality on all common attributes, projection on unique attributes
- Formally, $R \bowtie S$ is equivalent to:
  \[ \pi_{B_1,\ldots,B_m}(\sigma_{A_1=A_1',\ldots,A_k=A_k'}(R \times S)) \]
  where:
  - $A_1,\ldots,A_k$ are the attributes common to $R$ and $S$
  - $(B_1,\ldots,B_m)$ is the header of $R$
  - $(C_1,\ldots,C_l)$ is the header of $S$ with $A_1,\ldots,A_k$ removed

- Should we care about which new attribute names are used in the renaming?

Natural Join Example

$S = \begin{array}{ccc}
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2
\end{array}$

$T = \begin{array}{ccc}
861 & PL \\
861 & DB \\
955 & OS
\end{array}$

$S\bowtie T = \begin{array}{ccc}
861 & Alma & 2 & 861 & PL \\
861 & Alma & 2 & 861 & DB \\
955 & Ahuva & 2 & 955 & OS
\end{array}$

Semijoin

- Semijoin of $R$ and $S$ is the restriction of $R$ to the tuples that can naturally join with $S$
- Formally: $R \bowtie S$ is the operator equivalent to
  \[ \pi_{A_1,\ldots,A_k}(R \bowtie S) \]
  where $(A_1,\ldots,A_k)$ is the header of $R$
Semijoin Example

\[
S = \begin{bmatrix}
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2
\end{bmatrix}
\quad T = \begin{bmatrix}
861 & PL \\
762 & CB \\
955 & OB
\end{bmatrix}
\]

\[
S \bowtie T = \begin{bmatrix}
861 & Alma & 2 \\
955 & Ahuva & 2
\end{bmatrix}
\]

Intersection

- The usual binary set-theoretic operator \( \cap \)
- **Legal input:** a pair of relations that are union compatible (i.e., same header)
- Special case of natural join and semijoin
  - If \( R \) and \( S \) have the same header, then \( R \bowtie S \) and \( R \bowtie S \) are equal to \( R \cap S \)

Outline

- Background
- The Primitive Operators
- Implied Operators
  - Joins
    - Division
  - Equivalence & Independence
  - Taste of Query Optimization

Division

- Consider relations \( R(\mathbf{X}, \mathbf{Y}) \) and \( S(\mathbf{Y}) \)
  - Here, \( \mathbf{X} \) and \( \mathbf{Y} \) are disjoint tuples of attributes
- \( R \bowtie S \) is the relation \( T(\mathbf{X}) \) that contains all the \( \mathbf{X} \)s of \( R \) that occur in \( R \) with every \( \mathbf{Y} \) in \( S \)

Formal Definition

- **Legal input:** \((R,S)\) such that \( R \) has all the attributes of \( S \)
- \( R \bowtie S \) is the relation \( T(\mathbf{X}) \) with:
  - The header of \( R \), with all attributes of \( S \) removed; let it be \( \mathbf{X} \)
  - Tuple set
    \[
    \{ t \in \pi_\mathbf{X}R \mid (t,s) \in R \text{ for all } s \in S \}
    \]
  - This is an abuse of notation, since the attributes in \( \mathbf{X} \) need not necessarily come before those of \( \mathbf{Y} \)

Who took all core courses?

<table>
<thead>
<tr>
<th>sid</th>
<th>student</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>DB</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>PL</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>DB</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>AI</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>DB</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>AI</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>sid</th>
<th>student</th>
<th>course type</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>DB core</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>PL core</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>AI elective</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>DB elective</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>PL core</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>DB elective</td>
</tr>
</tbody>
</table>
Questions

\[(R \times S) \div S = ?\]

\[(R \times S) \div R = ?\]

Who took all core courses?

\[\text{Studies} + \pi_{\text{course type} = \text{core}} \text{CourseType}\]

Examples of Inexpressible Queries

Some very useful queries cannot be expressed in RA!

- Aggregates: How many followers does Ahuva have? How many persons does one follow on average?
- Transitive closure: Is there a follower path from Anna to Amir? Is there a cycle?

(How can one prove inexpressiveness?)

RA Expressions (Queries)

- Let \(S\) be a relation schema
  - Recall: \(S\) is a finite set of named relation schemas
- An RA expression (RA query) over \(S\) is an expression in RA, applied to the relation names of \(S\)
- For example:
  - \(\pi_{\text{sid} = \text{stud}}(\text{Student} \times \rho_{\text{sid} = \text{stud}} \text{Studies})\)

Outline

- Background
- The Primitive Operators
- Implied Operators
- Joins
- Division
- Equivalence & Independence
- Taste of Query Optimization
Query Result

- Let $S$ be a database schema
- Let $\varphi$ be an RA query over $S$
- Let $I$ be a database over $S$
- The result of evaluating $\varphi$ over $I$, denoted $\varphi(I)$, is the relation obtained by applying $\varphi$ to the relations of $I$
  - That is, every relation name is replaced with the corresponding relation in $I$

Equivalence of RA Expressions

- Let $S$ be a database schema, and let $\varphi$ and $\psi$ be two RA queries over $S$
- We say that $\varphi$ and $\psi$ are equivalent, denoted $\varphi \equiv \psi$, if:
  - for every database $I$ over $S$ it holds that $\varphi(I) = \psi(I)$

Who Cares?

- Query optimization: we wish to allow DBMS to replace a query with an equivalent one that is more efficient to evaluate
- Expressiveness: do different sets of operators “give the same” class of expressible questions?
- Examples on $R(A,B), S(A,B), T(A,B)$
  - $\sigma_{A='a'}(R \bowtie S) \equiv (\sigma_{A='a'} R) \bowtie (\sigma_{A='a'} S)$ (selection push)
  - $\pi_A(R \cup S) \equiv \pi_A(R) \cup \pi_A(S)$
  - $(R \bowtie S) \bowtie T \equiv (T \bowtie S) \bowtie R$
  - Is it true that $\rho_{(A,B)}^T(R \times S) \equiv \pi_A R$?

Containment

- Let $S$ be a database schema, and let $\varphi$ and $\psi$ be two RA queries over $S$
- We say that $\varphi$ is contained in $\psi$, denoted $\varphi \subseteq \psi$, if for every instance $I$ over $S$ we have $\varphi(I) \subseteq \psi(I)$

Q: How does containment relate to equivalence?

- $\pi_{sid}(\text{Student} \bowtie \text{Studies}) \subseteq \pi_{sid}\text{Student} \cap \pi_{sid}\text{Studies}$
- $\pi_{sid}(\text{Student} \bowtie \text{Studies}) \supseteq \pi_{sid}\text{Student} \cap \pi_{sid}\text{Studies}$

Q: How do we prove containment? equivalence?

- $\pi_{sid}(\text{Student} \cap \text{TA}) \supseteq \pi_{sid}\text{Student} \cap \pi_{sid}\text{TA}$
- $\pi_{sid}(\text{Student} \cap \text{TA}) \subseteq \pi_{sid}\text{Student} \cap \pi_{sid}\text{TA}$

Q: How do we prove non-containment? non-equivalence?
6 Primitive Operators

1. Projection (π)
2. Selection (σ)
3. Renaming (ρ)
4. Union (∪)
5. Difference (∖)
6. Cartesian Product (×)

Q: Is this a "good" set of primitives? Could we drop an operator "without losing anything"?

Independence

- Let o be an RA operator, and let A be a set of RA operators
- We say that o is independent of A if o cannot be expressed in A; that is, no expression in A is equivalent to o

Independence among Primitives

THEOREM: Each of the six primitives is independent of the other five

Proof:
- Separate argument for each of the six
- Arguments follow a common pattern (next slide)
- We will do one operator here (union)

Independence of Union

1. Fix a schema S and an instance I over S
   - \( R(A), S(A) \quad I: \{ R(0), S(1) \} \)
2. Find a property P over relations
   - \#tuples < 2
3. Prove that for every expression φ that does not use o, the relation φ(I) satisfies P
   - Induction base: If R and S have \#tuples<2
   - Inductive: If φ₁(I) and φ₂(I) have \#tuples<2, then so do
     \( π(S(0)), π(S(0)), ρ(A)(φ₁(I)), ρ(A)(φ₂(I)), φ₁(I) \times φ₂(I), φ₁(I) \setminus φ₂(I) \)
4. Find an expression ψ such that ψ uses o and ψ(I) violates P
   - ψ = R \∪ S

Recipe for Proving Independence

- Proving that operator o is independent:
  1. Fix a schema S and an instance I over S
  2. Find a property P over relations
  3. Prove that for every expression φ over S that does not use o, the relation φ(I) satisfies P
     - Such proofs are typically by induction on the size of the expression, since operators compose
  4. Find an expression ψ such that ψ uses o and ψ(I) violates P

Task

1. Easy: Prove that Cartesian product (×) is independent of the other five
2. Harder: Prove that selection (σ) is independent of the other five, even if restricted to selections of the type σ"A='a"
3. In TA class: Prove that difference (∖) is independent of the other five
Outline

- Background
- The Primitive Operators
- Implied Operators
  - Joins
  - Division
- Equivalence & Independence
- Taste of Query Optimization

Rules of Thumb for Optimization

- Main computational challenges in RA:
  - Large intermediate results
  - Join is expensive
- Make intermediate results as small as possible before joining (while preserving equivalence)
  - Apply selection and projection as early as possible ("push select/projection")
  - Reorder joins to minimize intermediate relations
- Some optimization decisions are "always beneficial" (e.g., push selection) while others require knowledge on the data (e.g., join order)

Correct Projection Push

\[ \pi_x(R_1 \bowtie R_2) \equiv \pi_x(R_1) \bowtie \pi_x(R_2) \]

\[ \pi_x(R_1 \bowtie R_2) \equiv \pi_{x \times y}(R_1) \bowtie \pi_{x \times y}(R_2) \]

\[ \pi_x(R_1 \bowtie R_2) \equiv \pi_{x \times y}(\pi_x(R_1)) \bowtie \pi_{x \times y}(R_2) \]

Pushing Projection

- Projection reduces the length of each row, and can substantially reduce the number of rows
  - Example: Person(ssn,country)
- Consider the query \( \pi_x(R_1 \bowtie R_2) \); denote:
  - \( Y \) = \( R_1 \bowtie R_2 \) (i.e. the attributes in both \( R_1 \) and \( R_2 \))
  - \( X_1 \) = \( \pi_x R_1 \)
  - \( X_2 \) = \( \pi_x R_2 \)
- (Abuse of notation – we mix attribute sequences with attributes sets)
- We would like to push projections into the join, that is:
  \[ \pi_x(\pi_z R_1 \bowtie \pi_z R_2) \]
  - Which \( z_1 \) and \( z_2 \) can work (equivalence preserved)?

Pushing Down the Expression Tree

\[ \pi_x(\pi_x(R_1 \bowtie R_2)) \]

\[ \pi_x(R_1 \bowtie R_2) \]

\[ \pi_{x \times y}(\pi_x R_1) \bowtie \pi_{x \times y} R_2 \]
Selection Push

- Can we rewrite $\sigma_c(R_1 \bowtie R_2)$ as $(\sigma_a R_1 \bowtie \sigma_b R_2)$?
- If all the attributes of $C$ are in $R_1$, then
  $\sigma_c(R_1 \bowtie R_2) \equiv (\sigma_a R_1 \bowtie \sigma_b R_2)$
- If all the attributes of $C$ are in $R_2$, then
  $\sigma_c(R_1 \bowtie R_2) \equiv (\sigma_a R_1 \bowtie \sigma_b R_2)$
- If all the attributes of $C$ are in both $R_1$ and $R_2$, then
  $\sigma_c(R_1 \bowtie R_2) \equiv (\sigma_a R_1 \bowtie \sigma_b R_2)$
- Pushing selection is generally beneficial: we may need some rewriting to get opportunities...

Examples of Rewriting Operations

- Splitting conjunctions:
  $\sigma_{c \land d}(R) \equiv \sigma_c(\sigma_d(R)) \equiv \sigma_d(\sigma_c(R))$
  - Applies to disjunction as well?
- Pushing through selection:
  $\sigma_e(\sigma_d(R)) \equiv \sigma_d(\sigma_e(R))$
  - Pushing through projection:
    $\sigma_e(\pi_d(R)) \equiv \pi_d(\sigma_e(R))$
    - Assuming that $e$ uses only attributes from $A$

Pushing Down the Expression Tree

Rewriting Joins

- Up to order of attributes, the natural join is commutative and associative
  - Commutative: $R \bowtie S \equiv S \bowtie R$
  - Associative: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- Proof: straightforward
  - So, given an RA query that involves only natural joins, apply the joins in whatever order you want (similarly to addition)
    - We may need to reorder attributes... nonissue

Example (cont'd)

- $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))$
  - Push projection
- $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))$
  - Push projection
- $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))$
  - Push projection
- $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))$
  - Remove redundant projection
- $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))$
  - $\pi_{\text{ssn,country}}(\text{Person}) \bowtie (\pi_{\text{album,topic}}(\text{Picture}))$
**Perspective on Query-Plan Optimization**

- Algorithms for RA query-plan optimization have been the subject of much research.
- One of the first and common algorithms is the “Sellinger algorithm” from IBM Almaden.
  - [Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, Thomas G. Price: Access Path Selection in a Relational Database Management System. SIGMOD Conference 1979: 23-34]
  - Idea: dynamic programming; compute cost & size estimation for every possible subquery, using the costs of smaller subqueries.
- General toolkit and concepts apply to many data/query models: algebra, equivalence, cost, plan optimization.

---

**Note on Alternative Approaches**

- In a recent line of research, several alternative algorithms for RA computation are developed.
- These algorithms do not construct intermediate results from sub-queries.
  - Rather, simultaneously scan all input relations.
- More reading:
  - Stanford’s *Minesweeper* [Ngo, Nguyen, Ra, Rudra: Beyond worst-case analysis for joins with minesweeper. PODS 2014: 234-246]
- Not discussed in this course.

---