Lecture 3:
Relational Algebra
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

<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></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Querying: Which Courses Avia Took?

<table>
<thead>
<tr>
<th>S</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>name</td>
<td>addr</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
</tr>
</tbody>
</table>

Assembly

```assembly
...  mov $1, %ax
    mov $1, %rdi
    mov $message, %rsi
    mov $13, %rdx
    syscall
    mov $60, %rax
    xor %rdi, %rdi
...```

Python

```python
for s in S:
    for c in C:
        for t in T:
            if s.sName == 'Avia' and s.ID == t.sID and t.cNum == c.number:
                print c.name
```

**SQL**

```sql
SELECT C.name
FROM S, C, T
WHERE S.name = 'Avia' AND S.ID = T.sID
    AND T.cNum = C.number
```

**Logic Programming (Datalog)**

```prolog
\{ (x) | \exists y, n, z, l, g
[ S(y, n, 'Avia a') \land C(z, x, l) \land T(y, z, g) ] \}
```

**Logic (RC)**

```prolog
\{ (x) | \exists y, n, z, l, g
[ S(y, n, 'Avia a') \land C(z, x, l) \land T(y, z, g) ] \}
```

**Algebra (RA)**

```prolog
\pi_{C.name} ( \sigma_{S.name='Avia', number=cNum} ( ID=sID ) ( S \times C \times T ) )
```
The Relational Algebra (RA)

• Mathematical query language

• Introduced by Edgar Codd

• Since invention, developed and studied by Codd and many others
Names of students who study DB:

\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid}_1}(\rho_{\text{sid}/\text{sid}_1}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid}_1}(\sigma_{\text{topic}='DB'}\text{Course} \times \rho_{\text{cid}/\text{cid}_1}\text{Studies})))))} \]
Why RA?

• Understanding the relational algebra is a key to understanding 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
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)
A relation schema is a finite sequence of distinct attribute names att with a mapping of each to a domain dom of legal values.

Notation: \((att_1:dom_1,\ldots,att_k:dom_k)\)

Example: \((sid:int, name:string, year:int)\)
• Let $s$ be a relation schema $\langle \text{att}_1: \text{dom}_1, \ldots, \text{att}_k: \text{dom}_k \rangle$

• 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 $\text{dom}_1 \times \cdots \times \text{dom}_k$

<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>
</tbody>
</table>


• A relation $R$ is a pair $(s, r)$
  – $s$ is a relation schema
  • Called the header of $R$
  – $r$ is a finite set $r$ of tuples over $s$

<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>
• 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 *(sid, name, year)* instead of *(sid:int, name:string, year: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 in $R$
  – We refer to $v_i$ by $t.\text{att}_i$

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

<table>
<thead>
<tr>
<th>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>

$t.\text{sid} = 861$
$t.\text{name} = \text{Alma}$
$t[\text{sid},\text{name}] = (861,\text{Alma})$
$t[\text{name},\text{sid},\text{year}] = (\text{Alma},861,2)$
• A *database schema* is a finite set of *relation names*, each mapped into a relation schema.
  – Example: Student(sid, name, year), Course(cid, topic), Studies(sid, cid)

• A *(database) instance* over a schema consists of a relation for each relation schema.

### Example Relations

#### Student

<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>

#### Course

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>PL</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
</tr>
<tr>
<td>76</td>
<td>OS</td>
</tr>
</tbody>
</table>

#### Studies

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</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 $(e_1,...,e_d)$ of elements
  – Maps every sequence in its domain to an element $e$

• The definition of an operator allows for composition:

$$o_1(o_2(x),o_1(y,o_4(x,z)))$$

• Examples:
  – Ring of integers: $(\mathbb{Z},\{+\cdot\})$
  – Boolean algebra: $(\{\text{true, false}\},\{\land, \lor, \neg\})$
  – Relational algebra
The Relational Algebra

- In the relational algebra (RA) the elements are relations
  - Recall: pairs \((s,r)\)

- 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

- In practice, we allow many more useful operators that can be defined by 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 ($\times$)
Projection by Example

\[ R = \]

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

<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>

\[ \pi_{\text{year}}(R) = \]

<table>
<thead>
<tr>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Fewer tuples (why?)
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
  – A projection is parameterized by attributes, so we actually have infinitely many different projection operators

• **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

\[
R = \sigma_{\text{course} = 'DB'}(R) = \begin{array}{ccc}
\text{student} & \text{year} & \text{course} & \text{grade} \\
Alma & 1 & DB & 80 \\
Alma & 1 & PL & 94 \\
Ahuva & 2 & DB & 72 \\
\end{array}
\]

\[
\sigma_{\text{year} = 1 \land \text{grade} > 84}(R) = \begin{array}{ccc}
\text{student} & \text{year} & \text{course} & \text{grade} \\
Alma & 1 & PL & 94 \\
\end{array}
\]
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 ($\land, \lor, \neg$)
    • $\text{item1} = \text{item2}$
    • $\text{price} \geq 500 \land \text{price} \leq \text{budget}$

• **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 $\{t \mid t \in R \text{ and } t \models c\}$

**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

\[ R = \]

\[ \rho_{\text{year/level}}(R) = \]

<table>
<thead>
<tr>
<th>student</th>
<th>year</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>1</td>
<td>DB</td>
<td>80</td>
</tr>
<tr>
<td>Alma</td>
<td>1</td>
<td>PL</td>
<td>94</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>DB</td>
<td>72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>student</th>
<th>level</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>1</td>
<td>DB</td>
<td>80</td>
</tr>
<tr>
<td>Alma</td>
<td>1</td>
<td>PL</td>
<td>94</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>DB</td>
<td>72</td>
</tr>
</tbody>
</table>
**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 $R$ with $A$ 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**

\[ R = \]

<table>
<thead>
<tr>
<th>student</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>1</td>
</tr>
<tr>
<td>Anna</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ S = \]

<table>
<thead>
<tr>
<th>student</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>1</td>
</tr>
<tr>
<td>Amir</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ R \cup S = \]

<table>
<thead>
<tr>
<th>student</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>1</td>
</tr>
<tr>
<td>Anna</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
<tr>
<td>Amir</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ R \setminus S = \]

<table>
<thead>
<tr>
<th>student</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>
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∪S is the relation with the header of R (and S) and the union of the tuple sets
  – R\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∪S? R\S?
# Cartesian Product by Example

\[ R = \begin{array}{ccc}
\text{sid} & \text{name} & \text{year} \\
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2 \\
\end{array}\]

\[ S = \begin{array}{ccc}
\text{cid} & \text{topic} \\
23 & PL \\
45 & DB \\
76 & OS \\
\end{array}\]

\[ R \times S = \begin{array}{cccccc}
\text{sid} & \text{name} & \text{year} & \text{cid} & \text{topic} \\
861 & Alma & 2 & 23 & PL \\
753 & Amir & 1 & 23 & PL \\
955 & Ahuva & 2 & 23 & PL \\
861 & Alma & 2 & 45 & DB \\
753 & Amir & 1 & 45 & DB \\
955 & Ahuva & 2 & 45 & DB \\
861 & Alma & 2 & 76 & OS \\
753 & Amir & 1 & 76 & OS \\
955 & Ahuva & 2 & 76 & OS \\
\end{array}\]
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 Mom(ssn) with 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 $\{r \circ s \mid r \in R \text{ and } s \in S\}$
    - $\circ$ denotes concatenation

**Q:** If each of $R$ and $S$ have 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 `name.attribute`.

$$R = \begin{array}{|c|c|c|} \hline sid & name & year \\ \hline 861 & Alma & 2 \\ 753 & Amir & 1 \\ 955 & Ahuva & 2 \\ \hline \end{array}$$

$$S = \begin{array}{|c|c|} \hline sid & cid \\ \hline 861 & 23 \\ 753 & 45 \\ \hline \end{array}$$

$$R \times S = \begin{array}{|c|c|c|c|c|} \hline R.sid & name & year & S.sid & cid \\ \hline 861 & Alma & 2 & 861 & 23 \\ 753 & Amir & 1 & 861 & 23 \\ 955 & Ahuva & 2 & 861 & 23 \\ 861 & Alma & 2 & 753 & 45 \\ 753 & Amir & 1 & 753 & 45 \\ 955 & Ahuva & 2 & 753 & 45 \\ \hline \end{array}$$
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:

\[
\left( \sigma_{\text{course}=\text{DB}}(\text{Course}) \right) \times \left( \rho_{\text{cid/cid1}}(\text{Studies}) \right)
\]

becomes

\[
\sigma_{\text{course}=\text{DB}} \text{Course} \times \rho_{\text{cid/cid1}} \text{Studies}
\]
π_{name}(σ_{sid=sid1}(ρ_{sid/sid1}Student × π_{sid,cid}(σ_{cid=cid1}(σ_{topic='DB'}Course × ρ_{cid/cid1}Studies))))

Names of students who study DB:

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

<table>
<thead>
<tr>
<th>Course</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>topic</td>
</tr>
<tr>
<td>23</td>
<td>PL</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
</tr>
<tr>
<td>76</td>
<td>OS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studies</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>
\[
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}='DB'}\text{Course} \times \rho_{\text{cid}/\text{cid1}}\text{Studies}))))
\]

<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></td>
<td>cid</td>
<td>topic</td>
</tr>
<tr>
<td>23</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sid</td>
<td>cid</td>
</tr>
<tr>
<td>861</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>861</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>753</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>76</td>
<td></td>
</tr>
</tbody>
</table>
\[ \pi_{\text{name}} (\sigma_{\text{sid} = \text{sid1}} (\rho_{\text{sid}/\text{sid1}} \text{Student} \times \pi_{\text{sid}, \text{cid}} (\sigma_{\text{cid} = \text{cid1}} (\sigma_{\text{topic} = 'DB'} \text{Course} \times \rho_{\text{cid}/\text{cid1}} \text{Studies})))) ) \]
\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}='DB'}\text{Course} \times \rho_{\text{cid}/\text{cid1}}\text{Studies})))) \]
\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}} \text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}='DB'} \text{Course} \times \rho_{\text{cid}/\text{cid1}} \text{Studies}))))) \]

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
<th>sid</th>
<th>cid1</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>DB</td>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>

**Student**

<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>

**Course**

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>PL</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
</tr>
<tr>
<td>76</td>
<td>OS</td>
</tr>
</tbody>
</table>

**Studies**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>
\[
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}=\text{DB}}\text{Course} \times \rho_{\text{cid}/\text{cid1}}\text{Studies}))))
\]

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
<th>sid</th>
<th>cid1</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>DB</td>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td>753</td>
<td>45</td>
</tr>
</tbody>
</table>

**Student**

<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>

**Course**

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>PL</td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
</tr>
<tr>
<td>76</td>
<td>OS</td>
</tr>
</tbody>
</table>

**Studies**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
</tr>
<tr>
<td>861</td>
<td>45</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
</tr>
</tbody>
</table>
\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid}_1}(\rho_{\text{sid}/\text{sid}_1}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid}_1}(\sigma_{\text{topic}='DB'}\text{Course} \times \rho_{\text{cid}/\text{cid}_1}\text{Studies})))))) \]
\[
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}} \text{Student} \times \pi_{\text{sid}, \text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}='\text{DB}'} \text{Course} \times \rho_{\text{cid}/\text{cid1}} \text{Studies}))))
\]
\[ \pi_{\text{name}}(\sigma_{\text{sid} = \text{sid1}}(\rho_{\text{sid}/\text{sid1}} \text{Student} \times \pi_{\text{sid}, \text{cid}}(\sigma_{\text{cid} = \text{cid1}}(\sigma_{\text{topic} = \text{DB}} \text{Course} \times \rho_{\text{cid}/\text{cid1}} \text{Studies})))))) \]

<table>
<thead>
<tr>
<th>sid1</th>
<th>name</th>
<th>year</th>
<th>cid</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
<td>45</td>
<td>753</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
<td>45</td>
<td>753</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
<td>45</td>
<td>753</td>
</tr>
</tbody>
</table>
\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid}_1}(\rho_{\text{sid}/\text{sid}_1}\text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid}_1}(\sigma_{\text{topic}='DB'}\text{Course} \times \rho_{\text{cid}/\text{cid}_1}\text{Studies}))))) \]

<table>
<thead>
<tr>
<th>sid1</th>
<th>name</th>
<th>year</th>
<th>cid</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
<td>45</td>
<td>753</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|}
\hline
\text{sid} & \text{name} \\
\hline
861 & Alma \\
753 & Amir \\
955 & Ahuva \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{cid} & \text{topic} \\
\hline
23 & PL \\
45 & DB \\
76 & OS \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{sid} & \text{cid} \\
\hline
861 & 23 \\
861 & 45 \\
753 & 45 \\
955 & 76 \\
\hline
\end{array}
\]
\[ \pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\rho_{\text{sid}/\text{sid1}} \text{Student} \times \pi_{\text{sid},\text{cid}}(\sigma_{\text{cid}=\text{cid1}}(\sigma_{\text{topic}='\text{DB}'} \text{Course} \times \rho_{\text{cid}/\text{cid1}} \text{Studies}))))) \]
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)
Normal Form for RA Expressions

Claim: A RA expression of the form $\sigma_F(E)$ is equivalent to a RA expression of the form $\sigma_{F'}(E)$ with no negation in $F'$.

Proof:

$\neg(F \land G) = (\neg F) \lor (\neg G)$

$\neg(G \lor F) = (\neg G) \land (\neg F)$

$\neg\neg F = F$

Repeatedly applying, get

$(X \theta Y) \equiv \neg(X \alpha Y)$

where $\alpha$ is the “complement” of $\theta$. 
Normal Form for RA Expressions

Claim: A RA expression is equivalent to a RA expression in which all selections are of the form \( \sigma_{(X \theta Y)} \) where \( X \) and \( Y \) are constants or variables.

Proof: Apply the previous claim and eliminate negations, only \( \land \) and \( \lor \) connectives. Now,

\[
\sigma_{F_1 \lor F_2}(E) = \sigma_{F_1}(E) \cup \sigma_{F_2}(E)
\]

\[
\sigma_{F_1 \land F_2}(E) = \sigma_{F_1}\left(\sigma_{F_2}(E)\right)
\]
Example: Normal Form for RA Expressions

\[ \sigma_{(\neg (B<2) \land \neg ((A<7) \lor \neg (B<8)))}^{(R)} \]

\[ \sigma_{((B\geq 2) \land (\neg (A<7) \land \neg (B<8)))}^{(R)} \]

\[ \sigma_{((B\geq 2) \land ((A\geq 7) \land (B<8)))}^{(R)} \]

\[ \sigma_{B\geq 2 \left( \sigma_{A\geq 7 \left( \sigma_{B<8}^{(R)} \right)} \right)} \]
Outline

• Background
• The Primitive Operators
  ▪ Joins
  ▪ Division
• Equivalence & Independence
• Taste of Query Optimization
We now discuss relational operators that are:
   – Not among the 6 basic operators
   – Can be expressed in RA (implied)
   – Very common in practice

Enhancing the available operator set with the implied operators is a good idea!
   – Easier to write queries
   – Easier to understand/maintain queries
   – Easier for DBMS to apply specialized optimizations
Outline

• Background
• The Primitive Operators
• Implied Operators
  - Joins
    - Division
• Equivalence & Independence
• Taste of Query Optimization
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 and $\theta$ is a comparison operator
  – Example: $R \bowtie_{c<d} S$

• *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

\[ S = \begin{array}{|c|c|c|} 
\hline
\text{sid} & \text{name} & \text{year} \\
\hline
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2 \\
\hline
\end{array} \]

\[ T = \begin{array}{|c|c|} 
\hline
\text{stud} & \text{course} \\
\hline
861 & PL \\
861 & DB \\
762 & OS \\
955 & OS \\
\hline
\end{array} \]

\[ S \bowtie_{\text{sid} = \text{stud}} T = \begin{array}{|c|c|c|c|} 
\hline
\text{sid} & \text{name} & \text{year} & \text{stud} & \text{course} \\
\hline
861 & Alma & 2 & 861 & PL \\
861 & Alma & 2 & 861 & DB \\
955 & Ahuva & 2 & 955 & OS \\
\hline
\end{array} \]
Natural Join

- Cartesian product, equality on all common attributes, projection on unique attributes
- Formally, \( R \bowtie S \) is equivalent to:

\[
\pi_{B_1,...,B_m,...,C_1...,C_l} \sigma_{A_1=A'_1,...,A_k=A'_k} (R \times \rho_{A_1/A'_1,...,A_k/A'_k} S)
\]

where:

- \((B_1,...,B_m)\) is the header of \( R \)
- \((A_1,...,A_k)\) are the attributes common to \( R \) and \( S \)
- \((C_1,...,C_l)\) is the header of \( S \) with \( A_1,...,A_k \) removed

- Should we care about which new names are defined by renaming?
### Natural Join Example

**S** =

<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>

**T** =

<table>
<thead>
<tr>
<th>sid</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>PL</td>
</tr>
<tr>
<td>861</td>
<td>DB</td>
</tr>
<tr>
<td>762</td>
<td>OS</td>
</tr>
<tr>
<td>955</td>
<td>OS</td>
</tr>
</tbody>
</table>

**S ⋈ T** =

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>year</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
<td>PL</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>955</td>
<td>Ahuva</td>
<td>2</td>
<td>OS</td>
</tr>
</tbody>
</table>
• 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
  \[ \pi_{A_1, \ldots, A_m}(R \bowtie S) \]
  where $(A_1, \ldots, A_m)$ is the header of R
Semijoin Example

\[ S = \]

<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>

\[ T = \]

<table>
<thead>
<tr>
<th>sid</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>PL</td>
</tr>
<tr>
<td>861</td>
<td>DB</td>
</tr>
<tr>
<td>762</td>
<td>OS</td>
</tr>
<tr>
<td>955</td>
<td>OS</td>
</tr>
</tbody>
</table>

\[ S \triangleleft T = \]

<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>955</td>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>
What are Semijoins good for?

Compute $\pi_A (R1 \Join R2 \Join R3)$

Program

$R2' = R2 \Join R3$

$R1' = R1 \Join R2'$

Return $\pi_A R1'$

Can we always write such an equivalent program?

Column shipping
\[ \pi_A (R_1 \bowtie R_2 \bowtie R_2 \bowtie R_4) = \pi_A (((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4) \]
Definition of Attribute-connected Tree

• A tree is attribute-connected if the sub-tree induced by each attribute is connected.
• Semijoin can replace join when we can draw an attribute-connected tree in which the target attributes are contained in the root.

How difficult is it to find such a tree?
Reduction Algorithm

**Problem:** Compute a projection on A of \((R_1 \bowtie ... \bowtie R_n)\) where \(R_1, ..., R_n\) is acyclic, that is there is an attribute connected tree.

**Solution:** Root includes A. Perform \(\bowtie\)'s upwards then project on A

**Example:** \(R_1=AB, R_2=BC, R_3=CD, R_4=AM, R_5=MN, R_6=NK, R_7=MT\)
Algorithm: Is there an attribute-connected tree

- **Input**: \( J = S_1, \ldots, S_n \)
- **While** changes to \( J \) **do**:
  - If there is an attribute \( A \) that appears in one \( S_i \) only, delete \( A \).
  - If some \( S_i \) is a subset of some other \( S_j \), record the undirected edge \((S_i, S_j)\) and delete \( S_i \).
- If \( J \) has only a single \( S_j \), a tree exists with the recorded edges.
Example execution

- J = S1 = AB, S2 = BC, S3 = CD, S4 = AM, S5 = MN, S6 = NK, S7 = MT
  - Iteration 1: throw K
  - Iteration 2: throw S6, edge (S6, S5)
  - Iteration 3: throw T
  - Iteration 4: throw S7, edge (S7, S4)
  - Iteration 5: throw N
  - Iteration 6: throw S5, edge (S5, S4)
  - Iteration 7: throw M
  - Iteration 8: throw S4, edge (S4, S1)
  - Iteration 9: throw A
  - Iteration 10: throw S1, edge (S1, S2)
  - Iteration 11: throw B
  - Iteration 12: throw S2, record (S2, S3)
  - Iteration 13: throw C
  - Iteration 14: throw D
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 $R \bowtie S$ equal to $R \cap S$
Outline

• Background
• The Primitive Operators
• Implied Operators
  ▪ Joins
  ▪ Division
• Equivalence & Independence
• Taste of Query Optimization
### Studies

<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>

### CourseType

<table>
<thead>
<tr>
<th>course</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>core</td>
</tr>
<tr>
<td>PL</td>
<td>core</td>
</tr>
<tr>
<td>AI</td>
<td>elective</td>
</tr>
<tr>
<td>DC</td>
<td>elective</td>
</tr>
</tbody>
</table>

**Who took all core courses?**
Division

• Consider two relations $R(X,Y)$ and $S(Y)$
  – Here, $X$ and $Y$ are tuples of attributes

• $R \div S$ is the relation $T(X)$ that contains all the $X$s that occur with every $Y$ in $S$
• **Legal input:** \((R, S)\) such that \(R\) has all the attributes of \(S\)

• \(R \div S\) is the relation \(T\) with:
  – The header of \(R\), with all attributes of \(S\) removed
  – Tuple set \(\{ t[X] \mid t[X,Y] \in R \text{ for every } s[Y] \in S \}\)

  *This is an abuse of notation, since the attributes in \(X\) need not necessarily come before those of \(Y\)*
Questions

<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>

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

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

**Q:** If \( R \) has 1000 tuples and \( S \) has 100 tuples, how many tuples can be in \( R \div S \)?

**Q:** If \( R \) has 1000 tuples and \( S \) has 1001 tuples, how many tuples can be in \( R \div S \)?
Who took all core courses?

Studies \div \pi_{course} \sigma_{type='core'} CourseType
R▷S in Primitive RA

\[ \pi_X R \setminus \pi_X \left( \left( \pi_X R \times S \right) \ \setminus \ R \right) \]

- Each \( X \) of \( R \) w/ each \( Y \) of \( S \)
  - \((X,Y)\) s.t. \( X \) in \( R \), \( Y \) in \( S \), but \((X,Y)\) not in \( R \)

Xs in \( R \) where for some \( Y \) in \( S \),
  - \((X,Y)\) is not in \( R \)

\[ R \triangleright S \]
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?

<table>
<thead>
<tr>
<th>follower</th>
<th>followed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amir</td>
<td>Alma</td>
</tr>
<tr>
<td>Ahuva</td>
<td>Alma</td>
</tr>
<tr>
<td>Alma</td>
<td>Amir</td>
</tr>
<tr>
<td>Anna</td>
<td>Ahuva</td>
</tr>
</tbody>
</table>

Transitive closure: Is there a follower path from Anna to Amir? Is there a cycle?

(How can one prove inexpressiveness?)
Outline

• Background
• The Primitive Operators
• Implied Operators
  ▪ Joins
  ▪ Division
• Equivalence & Independence
• Taste of Query Optimization
• Let $S$ be a database 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:
  \[ \pi_{\text{sid}}(\sigma_{\text{sid}=\text{stud}}(\text{Student} \times \rho_{\text{sid}=\text{stud}}\text{Studies})) \]
Let \( S \) be a database schema

Let \( \varphi \) be an RA query over \( S \)

Let \( I \) be a database instance 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 \( I \)
Let $S$ be a database schema, and let $\phi$ and $\psi$ be two RA queries over $S$. We say that $\phi$ and $\psi$ are equivalent, denoted $\phi \equiv \psi$, if:

for every instance $I$ over $S$ it holds that $\phi(I) = \psi(I)$
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)
  
  - $\pi_A (R \cup S) \equiv \pi_A (R) \cup \pi_A (S)$
  
  - $(R \bowtie S) \bowtie T \equiv (T \bowtie S) \bowtie R$
  
  - $\pi_{R.A/A} \rho_{R.A/A} (R \times S) \equiv \pi_A R$ ?
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?
Q: How do we prove containment? equivalence?

\[ \pi_{\text{sid}}(\text{Student} \bowtie \text{Studies}) = \pi_{\text{sid}}\text{Student} \cap \pi_{\text{sid}}\text{Studies} \]

\[ \pi_{\text{sid}}(\text{Student} \bowtie \text{Studies}) \subseteq \pi_{\text{sid}}\text{Student} \cap \pi_{\text{sid}}\text{Studies} \]
**Student**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>year</th>
</tr>
</thead>
</table>

**TA**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>year</th>
</tr>
</thead>
</table>

\[ \pi_{\text{sid}}(\text{Student } \cap \text{ TA}) \equiv \pi_{\text{sid}}\text{Student } \cap \pi_{\text{sid}}\text{TA} \]

\[ \pi_{\text{sid}}(\text{Student } \cap \text{ TA}) \subseteq \pi_{\text{sid}}\text{Student } \cap \pi_{\text{sid}}\text{TA} \]

\[ \pi_{\text{sid}}(\text{Student } \cap \text{ TA}) \supseteq \pi_{\text{sid}}\text{Student } \cap \pi_{\text{sid}}\text{TA} \]

**Q:** How do we prove non-containment? non-equivalence?
Class Problem on Semijoin ($\bowtie$)

Consider:

R(A,B,C) $\bowtie$ S(C,D) T(D,E) U(E,F)

Are the following always equivalent? Contained?

(R $\bowtie$ S) $\bowtie$ (T $\bowtie$ U) and R $\bowtie$ (S $\bowtie$ (T $\bowtie$ U))

Consider \{R(a,b,c), S(c,d1), T(d2,e), U(e,f)\}

R $\bowtie$ (S $\bowtie$ (T $\bowtie$ U)) and R $\bowtie$ ((S $\bowtie$ T) $\bowtie$ U)

Consider R(a,b,c), S(c,d), T(d,e1), U(e2,f)
6 Primitive Operators

1. Projection ($\pi$)
2. Selection ($\sigma$)
3. Renaming ($\rho$)
4. Union ($\cup$)
5. Difference ($\setminus$)
6. Cartesian Product ($\times$)

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

82
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 $o$. 
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)
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 $\varphi$ over $S$ that does not use $o$, the relation $\varphi(I)$ satisfies $P$
    - Such proofs are typically by induction on the size of the expression, since operators compose
  4. Find an expression $\psi$ such that $\psi$ uses $o$ and $\psi(I)$ violates $P$
Independence of Union

1. Fix a schema $S$ and an instance $I$ over $S$
   - $S$: $R(A)$, $S(A)$  $I$: \{R(0), S(1)\}

2. Find a property $P$ over relations
   - #tuples < 2

3. Prove that for every expression $\varphi$ that does not use $\circ$, the relation $\varphi(I)$ satisfies $P$
   - Induction base: $R$ and $S$ have #tuples<2
   - Inductive: If $\varphi_1(I)$ and $\varphi_2(I)$ have #tuples<2, then so do $\sigma_c(\varphi_1(I))$, $\pi_A(\varphi_1(I))$, $\rho_{A/B}(\varphi_1(I))$, $\varphi_1(I) \times \varphi_2(I)$, $\varphi_1(I) \setminus \varphi_2(I)$

4. Find an expression $\psi$ such that $\psi$ uses $\circ$ and $\psi(I)$ violates $P$
   - $\psi=\text{RUS}$
Outline

• Background
• The Primitive Operators
• Implied Operators
  ▪ Joins
  ▪ Division
• Equivalence & Independence
• Taste of Query Optimization
Task: find Israelis who like albums with dog pictures

Which of the equivalent expressions is more efficient to apply?

\[
\pi_{ssn} \sigma_{\text{topic}='dog' \land \text{country}='Israel'} (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))
\]

\[
\pi_{ssn} \sigma_{\text{topic}='dog' \land \text{country}='Israel'} ((\text{Likes} \bowtie \text{Person}) \bowtie \text{Picture})
\]

\[
\pi_{ssn} ((\text{Likes} \bowtie \sigma_{\text{country}='Israel'} \text{Person}) \bowtie \sigma_{\text{topic}='dog'} \text{Picture})
\]

\[
\pi_{ssn} ((\text{Likes} \bowtie \sigma_{\text{country}='Israel'} \text{Person}) \bowtie \pi_{\text{album}} \sigma_{\text{topic}='dog'} \text{Picture})
\]

\[
\pi_{ssn} (\sigma_{\text{country}='Israel'} \text{Person} \bowtie (\pi_{ssn} (\text{Likes} \bowtie \pi_{\text{album}} \sigma_{\text{topic}='dog'} \text{Picture})))
\]
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)
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:
  – $Y = R_1 \cap R_2$ (i.e. the attributes in both $R_1$ and $R_2$)
  – $X_1 = X \cap R_1$
  – $X_2 = X \cap R_2$
• (Note the abuse of notation – we mix attribute sequences with attributes sets)
• We would like to push projections into the join, that is:
  $$\pi_X(\pi_{Z_1}R_1 \bowtie \pi_{Z_2}R_2)$$
• Which $Z_1$ and $Z_2$ can work (equivalence preserved)?
Correct Projection Push

\[ \pi_X(R_1 \bowtie R_2) \equiv \pi_{X_1}(R_1) \bowtie \pi_{X_2}(R_2) \]

Where:
- \( Y = R_1 \cap R_2 \)
- \( X_1 = X \cap R_1 \)
- \( X_2 = X \cap R_2 \)

When we push projection, we need to retain all the attributes that are used for (1) joining, and (2) operations outside the join.
Pushing Down the Expression Tree

\[ Y = R_1 \cap R_2 \]

\[ X_1 = X \cap R_1 \]

\[ X_2 = X \cap R_2 \]
• Can we rewrite \( \sigma_c(R_1 \bowtie R_2) \) as \( (\sigma_c R_1 \bowtie \sigma_c R_2) \)?
• If all the attributes of \( C \) are in \( R_1 \), then
  \[ \sigma_c(R_1 \bowtie R_2) \equiv (\sigma_c R_1 \bowtie R_2) \]
• If all the attributes of \( C \) are in \( R_2 \), then
  \[ \sigma_c(R_1 \bowtie R_2) \equiv (R_1 \bowtie \sigma_c R_2) \]
• If all the attributes of \( C \) in both \( R_1 \) and \( R_2 \), then
  \[ \sigma_c(R_1 \bowtie R_2) \equiv (\sigma_c R_1 \bowtie \sigma_c R_2) \]
• 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:
  \[ \sigma_c(\sigma_d(R)) \equiv \sigma_d(\sigma_c(R)) \]

• Pushing through projection:
  \[ \sigma_c(\pi_A(R)) \equiv \pi_A(\sigma_c(R)) \]
  – Assuming that \( c \) uses only attributes from \( A \)!
Pushing Down the Expression Tree

\[ \sigma_{c \wedge d} \pi_x (R_1 \bowtie R_2) \quad \pi_x \sigma_{c \wedge d} (R_1 \bowtie R_2) \quad \pi_x \sigma_c \sigma_d (R_1 \bowtie R_2) \quad \pi_x \sigma_c (R_1 \bowtie \sigma_d R_2) \]
Rewriting Joins

• Up to order of attributes, the natural join is *commutative* and *associative*
  – Commutative: $R \Join S \equiv S \Join R$
  – Associative: $(R \Join S) \Join T = R \Join (S \Join T)$

• 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

\[ \pi_{ssn} \sigma_{\text{topic}=\text{dog} \land \text{country}=\text{Israel}} (\text{Likes} \Join (\text{Person} \Join \text{Picture})) \]

- Reorder joins

\[ \pi_{ssn} \sigma_{\text{topic}=\text{dog} \land \text{country}=\text{Israel}} (\text{Person} \Join (\text{Likes} \Join \text{Picture})) \]

- Split selection

\[ \pi_{ssn} \sigma_{\text{topic}=\text{dog}} \sigma_{\text{country}=\text{Israel}} (\text{Person} \Join (\text{Likes} \Join \text{Picture})) \]

- Push selection

\[ \pi_{ssn} \sigma_{\text{topic}=\text{dog}} (\sigma_{\text{country}=\text{Israel}} \text{Person}) \Join (\text{Likes} \Join \text{Picture})) \]

- Push selection (x2)

\[ \pi_{ssn} ((\sigma_{\text{country}=\text{Israel}} \text{Person}) \Join (\text{Likes} \Join (\sigma_{\text{topic}=\text{dog}} \text{Picture}))) \]
Example (cont’d)

\[ \pi_{\text{ssn}}((\sigma_{\text{country}='Israel'}\text{Person}) \bowtie (\text{Likes} \bowtie (\sigma_{\text{topic}='dog'}\text{Picture}))) \]

Push projection

\[ \pi_{\text{ssn}}((\sigma_{\text{country}='Israel'}\text{Person}) \bowtie \pi_{\text{ssn}}(\text{Likes} \bowtie (\sigma_{\text{topic}='dog'}\text{Picture}))) \]

Push projection

\[ \pi_{\text{ssn}}((\sigma_{\text{country}='Israel'}\text{Person}) \bowtie \pi_{\text{ssn}}(\text{Likes} \bowtie (\pi_{\text{album}}\sigma_{\text{topic}='dog'}\text{Picture}))) \]

Push projection

\[ \pi_{\text{ssn}}((\pi_{\text{ssn}}\sigma_{\text{country}='Israel'}\text{Person}) \bowtie \pi_{\text{ssn}}(\text{Likes} \bowtie (\pi_{\text{album}}\sigma_{\text{topic}='dog'}\text{Picture}))) \]

Remove redundant projection

\[ (\pi_{\text{ssn}}\sigma_{\text{country}='Israel'}\text{Person}) \bowtie \pi_{\text{ssn}}(\text{Likes} \bowtie (\pi_{\text{album}}\sigma_{\text{topic}='dog'}\text{Picture})) \]
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
  – 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, compute answers by simultaneously scanning all input relations.

• More reading:
  – LogicBlox’s Leapfrog Trie Join
  – Stanford’s Minesweeper
    • [Hung Q. Ngo, Dung T. Nguyen, Christopher Re, Atri Rudra: Beyond worst-case analysis for joins with minesweeper. PODS 2014: 234-245]

• Not discussed in this course.