Database Management Systems
Course 236363

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
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>number</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>363</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>319</td>
</tr>
</tbody>
</table>

Assembly

```
mov $1, %rax
mov $1, %rdi
mov $message, %rsi
mov $13, %rdx
syscall
```

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)
```

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

SQL

QL

```
QL{⟨x⟩ | ∃y,n,z,l,g [S(y,n,'Avia') ∧ C(z,x,l) ∧ T(y,z,g)]}
```

Logic Programming (Datalog)

```
Logic(RC)
QL{⟨x⟩ | ∃y,n,z,l,g [S(y,n,'Avia') ∧ C(z,x,l) ∧ T(y,z,g)]}
```

Logic (RC)

```
QL{⟨x⟩ | ∃y,n,z,l,g [S(y,n,'Avia') ∧ C(z,x,l) ∧ T(y,z,g)]}
```

The Relational Algebra (RA)

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

RA Example

Names of students who study DB:

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

<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></td>
<td></td>
</tr>
</tbody>
</table>
Why RA?

• Understanding the relational algebra is a key 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

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)

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 $(att_1:dom_1, \ldots, att_k:dom_k)$
- A tuple (over $s$) is a sequence $(v_1, \ldots, v_k)$ of values $v_i$ where each $v_i$ is in $dom_i$.
  - That is, a tuple is an element of $dom_1 \times \ldots \times dom_k$.

<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>

Relations

- A relation $R$ is a pair $(s, r)$
  - $s$ is a relation schema
    - Called the header of $R$
  - $r$ is a finite set 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>

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 $(sid, name, year)$ 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 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 $(t.a_1,\ldots,t.a_m)$

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

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

Databases

• A database schema is finite set of relation names, each mapped into a relation schema
  – Example: $\text{Student}((\text{sid},\text{name},\text{year}))$, $\text{Course}((\text{cid},\text{topic}))$, $\text{Studies}((\text{sid},\text{cid}))$

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

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
</table>
| \begin{array}{ccc}
\text{sid} & \text{name} & \text{year} \\
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2 \\
\end{array} | \begin{array}{ccc}
\text{cid} & \text{topic} & \text{sid} \\
23 & PL & 861 \\
45 & DB & 861 \\
76 & OS & 753 \\
\end{array} | \begin{array}{ccc}
\text{sid} & \text{cid} \\
861 & 23 \\
861 & 45 \\
753 & 45 \\
955 & 76 \\
\end{array} |

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:
  \[
o_1(o_2(o_1(x),o_1(y,o_1(z,x))))
  \]

• Examples:
  – Ring of integers: $(\mathbb{Z},+)$
  – Boolean algebra: $(\{\text{true, false}\},\wedge,\vee,\neg)$
  – 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 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 (π)
2. Selection (σ)
3. Renaming (ρ)
4. Union (∪)
5. Difference (∖)
6. Cartesian Product (×)
Task
(end of this section)

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)

Projection by Example

\[ R = \]

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

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

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)
- \( \pi_{A_2, \ldots, A_k}(R) \) is the relation \( S \) with:
  - Header \( (A_2, \ldots, A_k) \)
  - Tuple set \( \{(A_2, \ldots, A_k) \mid t \in R\} \)

Q: If \( R \) has 1000 tuples, how many tuples can \( \pi_{A_2, \ldots, A_k}(R) \) have?
Selection by Example

\[ R = \begin{align*}
\text{student} & \quad \text{year} & \quad \text{course} & \quad \text{grade} \\
\text{Alma} & \quad 1 & \quad \text{DB} & \quad 80 \\
\text{Alma} & \quad 1 & \quad \text{PL} & \quad 94 \\
\text{Ahava} & \quad 2 & \quad \text{DB} & \quad 72
\end{align*} \]

\[ \sigma_{\text{course} = \text{DB}}(R) = \begin{align*}
\text{student} & \quad \text{year} & \quad \text{course} & \quad \text{grade} \\
\text{Alma} & \quad 1 & \quad \text{DB} & \quad 80 \\
\text{Ahava} & \quad 2 & \quad \text{DB} & \quad 72
\end{align*} \]

\[ \sigma_{\text{year} = 1 \land \text{grade} > 84}(R) = \begin{align*}
\text{student} & \quad \text{year} & \quad \text{course} & \quad \text{grade} \\
\text{Alma} & \quad 1 & \quad \text{PL} & \quad 94
\end{align*} \]

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 (\( \land, \lor, \neg \))
    - \( \text{item}_1 = \text{item}_2 \)
    - \( \text{price} \geq 500 \land \text{price} \leq \text{budget} \)
  - **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 \( \{ t \mid t \in R \text{ and } t \vDash c \} \)

*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., \( c > a^2 + b^2 \); 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

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

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

### 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

\[ R = \begin{array}{c|c}
\text{student} & \text{year} \\
Alma & 1 \\
Anna & 1 \\
Ahuva & 2 \\
\end{array} \]

\[ S = \begin{array}{c|c}
\text{student} & \text{year} \\
Alma & 1 \\
Amir & 3 \\
\end{array} \]

\[ R \cup S = \begin{array}{c|c}
\text{student} & \text{year} \\
Alma & 1 \\
Anna & 1 \\
Ahuva & 2 \\
Amir & 3 \\
\end{array} \]

\[ R \setminus S = \begin{array}{c|c}
\text{student} & \text{year} \\
Anna & 1 \\
Ahuva & 2 \\
Amir & 3 \\
\end{array} \]
**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**

<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>

\[ R = \]

<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>

\[ S = \]

\[ R \times S = \]

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

**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 \( (A_1,...,A_k) \) and \( (B_1,...,B_m) \), respectively; then \( R \times S \) is the relation \( T \) with:
  
  - **Header** \( (A_1,...,A_k,B_1,...,B_m) \)
  
  - **Tuple set** \( \{ r \circ s | r \in R \text{ and } s \in S \} \)
    
    - \( \circ \) 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

\[
R = \begin{array}{ccc}
\text{sid} & \text{name} & \text{year} \\
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2 \\
\end{array} \quad \quad S = \begin{array}{cc}
\text{sid} & \text{cid} \\
861 & 23 \\
753 & 45 \\
\end{array}
\]

\[
R \times S = \begin{array}{cccc}
\text{R.sid} & \text{name} & \text{year} & \text{S.sid} & \text{cid} \\
861 & Alma & 2 & 861 & 23 \\
753 & Amir & 1 & 861 & 23 \\
955 & Ahuva & 2 & 861 & 23 \\
\end{array}
\]

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:
  \[ (\sigma_{\text{course}='DB'} (\text{Course})) \times (\rho_{\text{cid}/\text{sid}} (\text{Studies})) \text{ becomes } \sigma_{\text{course}='DB'} \text{Course} \times \rho_{\text{cid}/\text{sid}} \text{Studies} \]

Composition Example

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

Names of students who study DB:

\[
\begin{array}{ccc}
\text{Student} & \text{Course} & \text{Studies} \\
\text{sid} & \text{name} & \text{year} & \text{cid} & \text{topic} & \text{sid} & \text{cid} \\
861 & Alma & 2 & 23 & PL & 861 & 23 \\
753 & Amir & 1 & 45 & DB & 861 & 45 \\
955 & Ahuva & 2 & 76 & OS & 753 & 45 \\
\end{array}
\]
\begin{align*}
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\pi_{\text{sid}}(\pi_{\text{sid}}(\pi_{\text{topic='DB'}}(\text{Course} \times \pi_{\text{sid}}(\text{Student})))))))
\end{align*}

<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>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>

\begin{align*}
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\pi_{\text{sid}}(\pi_{\text{topic='DB'}}(\text{Course} \times \pi_{\text{sid}}(\text{Student})))))))
\end{align*}

<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>

\begin{align*}
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\pi_{\text{sid}}(\pi_{\text{topic='DB'}}(\text{Course} \times \pi_{\text{sid}}(\text{Student})))))))
\end{align*}

<table>
<thead>
<tr>
<th>cid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>DB</td>
</tr>
</tbody>
</table>

\begin{align*}
\pi_{\text{name}}(\sigma_{\text{sid}=\text{sid1}}(\pi_{\text{sid}}(\pi_{\text{topic='DB'}}(\text{Course} \times \pi_{\text{sid}}(\text{Student})))))))
\end{align*}

<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>
\[ \pi_{\text{name}}(\pi_{\text{cid}}(\pi_{\text{sid}}(\sigma_{\text{topic} = 'DB'}(\text{Course} \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'PL'}(\text{Course}) \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'OS'}(\text{Course})))))))))) \]

<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>

\[ \pi_{\text{name}}(\pi_{\text{cid}}(\pi_{\text{sid}}(\sigma_{\text{topic} = 'DB'}(\text{Course} \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'PL'}(\text{Course}) \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'OS'}(\text{Course}))))))))))) \]

<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>

\[ \pi_{\text{name}}(\pi_{\text{cid}}(\pi_{\text{sid}}(\sigma_{\text{topic} = 'DB'}(\text{Course} \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'PL'}(\text{Course}) \times \pi_{\text{sid}}(\pi_{\text{cid}}(\sigma_{\text{topic} = 'OS'}(\text{Course}))))))))))) \]

<table>
<thead>
<tr>
<th>cid</th>
<th>sid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>861</td>
<td>DB</td>
</tr>
<tr>
<td>45</td>
<td>753</td>
<td>DB</td>
</tr>
</tbody>
</table>
\[ \pi_{\text{name}}(\pi_{\text{sid}}(\text{Student} \times \pi_{\text{topic}}(\pi_{\text{cid}}(\text{Course} \times \pi_{\text{cid}}(\text{Studies})))))) \]

<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>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>
<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>

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

<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>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>
<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>

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

<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>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>
<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>

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

<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>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>
<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>
### 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)

```sql
π_{name}(π_{sid}((π_{name}(Student × π_{sid}(π_{topic}(Course × π_{sid}(Studies))))))
```

### Outline

- Background
- The Primitive Operators
  - Implied Operators
    - Joins
    - Division
- Equivalence & Independence
- Taste of Query Optimization
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
• 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, $v$ a constant value, and $\theta$ 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

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

$T = \begin{array}{cc}
stud & course \\
861 & PL \\
861 & DB \\
762 & OS \\
955 & OS \\
\end{array}$

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

- Cartesian product, equality on all common attributes, projection on unique attributes
- Formally, $R \bowtie S$ is equivalent to:
  $$
  \pi_{B_1, \ldots, B_m, C_1, \ldots, C_l} \sigma_{A_1 = A'_1, \ldots, A_k = A'_k}(R \times \rho_{A'_1/A_1, \ldots, A'_k/A_k}S)
  $$
  where:
  - $(B_1, \ldots, B_m)$ is the header of $R$
  - $A_1, \ldots, A_k$ are the attributes common to $R$ and $S$
  - $(C_1, \ldots, C_l)$ is the header of $S$ with $A_1, \ldots, A_k$ removed
- Should we care about which new names are defined by renaming?

Natural Join Example

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

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

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

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
  $$
  \pi_{A_1, \ldots, A_m}(R \bowtie S)
  $$
  where $(A_1, \ldots, A_m)$ is the header of $R$
Semijoin Example

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

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

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

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
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 $S$

Formal Definition

• 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
    $\{t[X] \in \pi_X R \mid t[X,Y] \in R$ for all $s[Y] \in S\}$

• This is an abuse of notation, since the attributes in $X$ need not necessarily come before those of $Y$
Questions

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

\( Q: \) 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 \( R \div S \)?

---

**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?**

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

---

**R+\text{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 \( S \) s.t. \( X \) in \( R \), \( Y \) in \( S \), but \( (X,Y) \) not in \( R \).

\( R \div S \) in \( R \) where for some \( Y \) in \( S \), \( (X,Y) \) is not in \( R \).
Examples of Inexpressible Queries

Some very useful queries cannot be expressed in RA!

<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>

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?)

Outline

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

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:
  - $\pi_{sid}(\sigma_{sid=stud}(\text{Student} \times \rho_{sid/stud}(\text{Studies})))$
Query Result

• Let $S$ be a database schema
• Let $\phi$ be an RA query over $S$
• Let $I$ be a database over $S$
• The result of evaluating $\phi$ over $I$, denoted $\phi(I)$, is the relation obtained by applying $\phi$ 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 $\phi$ and $\psi$ be two RA queries over $S$
• We say that $\phi$ and $\psi$ are equivalent, denoted $\phi \equiv \psi$, if:

  for every database $I$ 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)
  – $\pi_A(R \cup S) \equiv \pi_A(R) \cup \pi_A(S)$
  – $(R \bowtie S) \bowtie T \equiv (T \bowtie S) \bowtie R$
  – Is it true that $\rho_{R.A/R.A}(R \times S) \equiv \pi_R$?
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?

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

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

**Q:** How do we prove containment? equivalence?

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

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

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

**Q:** How do we prove non-containment? non-equivalence?
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"?

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).
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 \( o \), 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 \( \sigma_c(\varphi_1(I)), \pi_{A_1}(\varphi_1(I)), \rho_{A_2}(\varphi_1(I)), \varphi_1(I) \bowtie \varphi_2(I), \varphi_1(I) \setminus \varphi_2(I) \)
4. Find an expression \( \psi \) such that \( \psi \) uses \( o \) and \( \psi(I) \) violates \( P \)
   - \( \psi=R\cup S \)

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?

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

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

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

\[ \pi_{\text{ssn}} (\pi_{\text{album} \boweq \text{topic}=\text{dog}} (\text{Likes} \bowtie (\text{Person} \bowtie \text{Picture}))) \]

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: \( \text{Person(ssn,country)} \)
- 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:
  \[ \pi_X (\pi_{X_1} R_1 \bowtie \pi_{X_2} R_2) \]
- Which \( Z_1 \) and \( Z_2 \) can work (equivalence preserved)?
Correct Projection Push

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

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

Selection Push

- 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

\[ \sigma_{c \land d}(R_1 \bowtie R_2) \equiv \pi_X(\pi_Y(\sigma_{c \land d}(R_1 \bowtie R_2))) \]

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

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

- Reorder joins

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

- Split selection

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

- Push selection

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

- Push selection (x2)

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

- Push projection

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

- Push projection

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

- Push projection

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

- Remove redundant projection

Example (cont’d)

\[ \pi_{\text{ssn}} (\pi_{\text{ssn}} (\text{Person} \bowtie (\text{Likes} \bowtie \text{Picture}))) \]

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. Astraham, 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, 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