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

The Relational Algebra (RA)

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

Outline

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

Querying: Which Courses Avia Took?

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>number</th>
<th>topic</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Halls</td>
<td>363</td>
<td>DB</td>
<td>95</td>
</tr>
<tr>
<td>1395</td>
<td>Boris</td>
<td>Venue</td>
<td>319</td>
<td>PL</td>
<td>82</td>
</tr>
</tbody>
</table>

Assembly

```assembly
SELECT c.name FROM S,c,t WHERE c.name = 'Avia' AND S.ID = C.ID AND t.chi = C.number
```

Logic (RC)

- Logic Programming (Datalog)
  - `∀x χ(x,x,Avia),(x,y,Avia)`
  - `∃x χ(x,y,Avia)`
  - `∀x χ(x,y,Avia) ∧ t.chi = t.number`

Logic Programming (Datalog)

- `χ(x,y,Avia)`
  - `∀x χ(x,x,Avia)`
  - `∃x χ(x,y,Avia)`

RA Example

Names of students who study DB:

- `∀x χ(x,y,Avia)`
  - `∀x χ(x,x,Avia)`
  - `∃x χ(x,y,Avia)`

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Avia</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>alma</td>
<td>2</td>
</tr>
<tr>
<td>861</td>
<td>PL</td>
<td>23</td>
</tr>
<tr>
<td>753</td>
<td>DB</td>
<td>45</td>
</tr>
<tr>
<td>753</td>
<td>CS</td>
<td>76</td>
</tr>
<tr>
<td>955</td>
<td>CS</td>
<td>76</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 att with a mapping of each to a domain dom of legal values
- Notation: (att1:dom1, ..., attk:domk)
  - Example: (sid:int, name:string, year:int)

Tuples

- Let s be a relation schema (att1:dom1, ..., attk:domk)  
- A tuple (over s) is a sequence (v1, ..., vk) of values vᵢ, where each vᵢ is in domᵢ  
  - That is, a tuple is an element of dom₁ × ... × domₖ

Relations

- 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

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 $(a_1, \ldots, a_k)$
  - Let $t = (v_1, \ldots, v_k)$ be a tuple in $R$
  - We refer to $v_i$ by $t.a_i$

- **Notation 2:**
  - Let $a_1, \ldots, a_m$ be attributes in $(a_1, \ldots, a_k)$
  - 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>

Databases

- 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

<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>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>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>sid</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>861</td>
<td>23</td>
<td>861</td>
<td>PL</td>
</tr>
<tr>
<td>753</td>
<td>45</td>
<td>753</td>
<td>DB</td>
</tr>
<tr>
<td>955</td>
<td>76</td>
<td>955</td>
<td>OS</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: $o(e_1, e_2, \ldots, e_d) = e$
- Examples:
  - Ring of integers: $(\mathbb{Z}, +, \cdot)$
  - Boolean algebra: $(\{true, false\}, \&, \|, \neg)$
- Relational algebra

The Relational Algebra

- In the relational algebra (RA) the elements are relations
  - Recall: pairs $(u, v)$
- 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

Task (for the end of this part)

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)

Pairs / groups allowed. Email solution to me:
  * bennyK@cs.technion.ac.il
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**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>915</td>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

$\pi_{\text{sid}, \text{name}}(R) = \begin{cases} 
\text{sid} & \text{Alma} \\
\text{sid} & \text{Amir} \\
\text{sid} & \text{Ahuva} 
\end{cases}$

Fewer tuples (why?)

**Selection by Example**

<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>

$\sigma_{\text{course} = \text{DB}}(R) = \begin{cases} 
\text{student} & \text{Alma} \\
\text{student} & \text{Ahuva} 
\end{cases}$

$\sigma_{\text{year} = 1 \land \text{grade} > 84}(R) = \begin{cases} 
\text{student} & \text{Alma} 
\end{cases}$

**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 many different projection operators.

• Legal input: a 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 $\{t[A_1, \ldots, A_k] \ | \ t \in R\}$

**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 ($\land, \lor, \neg$).
  – $\text{price} \geq 500 \land \text{price} \leq \text{budget}$

• 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 $\{t \ | \ t \in R \land t \models c\}$

**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”
**Cartesian Product by Example**

\[ R \times S = \]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{id} & \text{name} & \text{year} & \text{topic} \\
\hline
861 & Alma & 2 & PL \\
753 & Amir & 1 & DB \\
955 & Ahuva & 2 & OS \\
\hline
\end{array}
\]

**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 \( \{ (r,s) \mid r \in R \text{ and } s \in S \} \)
    - denotes concatenation

**Definition of Renaming**

- Renaming is a unary operator of the form \( \rho_{A/B} \)
- **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 \( R \)

**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

**Renaming by Example**

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

\[
\begin{array}{|c|c|c|}
\hline
\text{student} & \text{year} & \text{level} \\
\hline
Ahuva & 955 & 861 \\
Anna & 753 & 76 \\
Alma & 955 & 23 \\
Amir & 753 & 45 \\
Ahuva & 955 & 76 \\
Alma & 861 & 23 \\
Amir & 753 & 45 \\
Alma & 861 & 76 \\
Ahuva & 955 & 76 \\
\hline
\end{array}
\]

**Union and Difference by Example**

\[ R = \]

\[
\begin{array}{|c|c|c|}
\hline
\text{student} & \text{year} & \text{course} \\
\hline
Ahuva & 1 & DB \\
Anna & 1 & PL \\
Alma & 2 & DB \\
Amir & 1 & PL \\
\hline
\end{array}
\]

\[ S = \]

\[
\begin{array}{|c|c|c|}
\hline
\text{student} & \text{year} & \text{course} \\
\hline
Ahuva & 1 & DB \\
Anna & 1 & PL \\
Alma & 2 & DB \\
Amir & 1 & PL \\
\hline
\end{array}
\]

\[ R \cup S = \]

\[
\begin{array}{|c|c|c|}
\hline
\text{student} & \text{year} & \text{course} \\
\hline
Ahuva & 1 & DB \\
Anna & 1 & PL \\
Alma & 1 & PL \\
Amir & 1 & PL \\
Ahuva & 2 & DB \\
Alma & 1 & DB \\
Amir & 1 & DB \\
Alma & 2 & DB \\
Ahuva & 2 & DB \\
\hline
\end{array}
\]

\[ R \setminus S = \]

\[
\begin{array}{|c|c|c|}
\hline
\text{student} & \text{year} & \text{course} \\
\hline
Ahuva & 2 & DB \\
Alma & 2 & DB \\
\hline
\end{array}
\]

**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 \( R \)

**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

**Answer:**

- \( \rho_{\text{year/level}}(R) \) contains 1000 tuples, how many tuples can be in \( R \cup S ? \)
- \( R \setminus S \) contains 0 tuples, how many tuples can be in \( R \times S ? \)
- If each of \( R \) and \( S \) have 1000 tuples, how many tuples can be in \( R \times S ? \)
- If \( R \) has 1000 tuples, how many tuples can \( \rho_{\text{year/level}}(R) \) have?
Shorthand Notation

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

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

\[
S = \begin{array}{c|c}
\text{id} & \text{cid} \\
861 & 23 \\
753 & 45 \\
955 & 76 \\
\end{array} 
\]

\[
R \times S = \begin{array}{c|c|c|c|c}
\text{id} & \text{name} & \text{year} & \text{id} & \text{cid} \\
861 & Alma & 2 & 861 & 23 \\
753 & Amir & 1 & 753 & 45 \\
955 & Ahuva & 2 & 955 & 76 \\
\end{array} 
\]

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 presence over binary ones.

Example:

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

becomes

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

Composition Example

Names of students who study DB:

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

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

\[
\begin{array}{c|c|c}
\text{sid} & \text{cid} & \text{topic} \\
861 & 23 & PL \\
753 & 45 & DB \\
955 & 76 & OS \\
\end{array} 
\]

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

<table>
<thead>
<tr>
<th>Course</th>
<th>id</th>
<th>topic</th>
<th>cid</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>753</td>
<td>1</td>
<td>DL</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>3</td>
<td>OS</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>2</td>
<td>DB</td>
<td>753</td>
<td>861</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studies</th>
<th>id</th>
<th>topic</th>
<th>cid</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>753</td>
<td>1</td>
<td>DL</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>3</td>
<td>OS</td>
<td>45</td>
<td>861</td>
</tr>
<tr>
<td>753</td>
<td>2</td>
<td>DB</td>
<td>753</td>
<td>861</td>
</tr>
</tbody>
</table>
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)

Pairs / groups allowed. Email solution to me:

bennyk@cs.technion.ac.il

• Write p[A] for π_s,cid=α(β)

\[ a = \pi_{cid}(\sigma_{cid=\alpha}(\text{Studies} \times \rho_{sid/cid/\text{Studies}})) \]

• Courses with >2 students:
  \[ a = \pi_{cid}(\sigma_{cid=\alpha}(\text{Studies} \times \rho_{sid/cid/\text{Studies}})) \]

• Courses with precisely one student:
  \[ \beta = (\pi_{cid}\text{Studies} \setminus a) \]

• ID of students who get a private lesson:
  \[ \gamma = \pi_{sid}(\sigma_{cid=\alpha}(\text{Studies} \times \rho_{cid/\beta})) \]

• Final answer (join w/ names):
  \[ \pi_{name,\text{id}=s}(\rho_{sid/\text{Student}} \times \gamma) \]
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: $\alpha_c(R \times S)$.
- 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: $R \bowtie_{a=b \land c<d} S$.
- **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$_{\text{name} = \text{course}}$ Studies.

Natural Join $\bowtie$ 

- Cartesian product, equality on all common attributes, projection on unique attributes.
- Formally, $R \bowtie S$ is equivalent to:
  $\pi_B\sigma_{A_1=A'_1,\ldots,A_k=A'_k}(R \times \rho_{A_1/A'_1}(R) \times \rho_{A_2/A'_2}(R) \times \cdots \times \rho_{A_k/A'_k}(R))$

  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? (No)
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

\[
\pi_{A_1, \ldots, A_m}(R \times S)
\]

where \((A_1, \ldots, A_m)\) is the header of R.

Semijoin Example

\[
S = \begin{array}{ccc}
861 & Alma & 2 \\
753 & Amir & 1 \\
955 & Ahuva & 2 \\
\end{array} \\
T = \begin{array}{ccc}
861 & PL &  \\
861 & DB &  \\
762 & OS &  \\
955 & OS &  \\
\end{array} \\
S \bowtie T = \begin{array}{ccc}
861 & Alma & 2 \\
955 & Ahuva & 2 \\
\end{array}
\]

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 \( R \bowtie S = R \cap S \).

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 \( S \).
Formal Definition

- **Legal input**: \((R,S)\) such that \(R\) has all the attributes of \(S\)
- \(R÷S\) 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

1. Course: If \(R\) has 1000 tuples and \(S\) has 100 tuples, how many tuples can be in \(R÷S\)?
2. Course: If \(R\) has 1000 tuples and \(S\) has 1001 tuples, how many tuples can be in \(R÷S\)?

Examples of Inexpressible Queries

- 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?

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:
  – $\pi_{\text{sid}}(\sigma_{\text{sid} = \text{stud}}(\text{Student} \times \rho_{\text{sid}}/	ext{stud} \text{Studies}))$

Query Result

• 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 $I$

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 instance $I$ 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)
  – $\pi_{A}(R \cup S) \equiv \pi_{A}(R) \cup \pi_{A}(S)$
  – $(R \times S) \bowtie T \equiv (T \times S) \bowtie R$
  – Is $\rho_{A/B}(R \times S) \equiv \pi_{A}R$?

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?

$\varphi \equiv \psi$ is the same as $(\varphi \subseteq \psi$ and $\psi \subseteq \varphi)$
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"?

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

Answer: show a counterexample

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 $o$

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 $P$
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)

Pushing Projection

- Projection reduces the length of each row, and can substantially reduce the number of rows
  - Example: Person(ssn,country)
- Consider the query \( n_k(R_1 \bowtie R_2) \), denote:
  - \( Y = R_1 \bowtie R_2 \) (i.e. the attributes in both \( R_1 \) and \( R_2 \))
  - \( X = X_1 \bowtie R_2 \)
  - \( X = X_1 \bowtie R_2 \)
  - (Note the abuse of notation – we mix attribute sequences with attributes sets)
- We would like to push projections into the join, that is:
  \[ n_k \left( n_{k_2}(R_1) \bowtie n_{k_2}(R_2) \right) \]
  - Which \( Z_1 \) and \( Z_2 \) can work (equivalence preserved)?

Correct Projection Push

\[
\begin{align*}
n_k(R_1 \bowtie R_2) &= n_k(R_1) \bowtie n_k(R_2) \quad ? \\
n_k(R_1 \bowtie R_2) &= n_k(R_1) \bowtie n_k(R_2) \quad ? \\
n_k(R_1 \bowtie R_2) &= n_k(n_k(R_1) \bowtie n_k(R_2)) \quad ?
\end{align*}
\]

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

\[
\begin{align*}
n_k(R_1 \bowtie R_2) &= n_k(n_k(R_1) \bowtie n_k(R_2)) \\
n_k(R_1 \bowtie R_2) &= n_k(R_1) \bowtie n_k(R_2)
\end{align*}
\]
Pushing Down the Expression Tree

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

Examples of Rewriting Operations

- Splitting conjunctions:
  \[ \sigma_{c_1}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_1}(\sigma_{c_2}(R)) \]
  - Applies to disjunction as well?
- Pushing through selection:
  \[ \sigma_c(\pi_d(R)) \equiv \pi_d(\sigma_c(R)) \]
- Pushing through projection:
  \[ \pi_c(\sigma_c(R)) \equiv \pi_c(\sigma_c(R)) \]
  - Assuming that $c$ uses only attributes from $A$!

Rewriting Joins

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

- Person[ssn,country]
- Picture[pid,topic,album]
- Likes[ssn,album]

Example (cont’d)

- Person[ssn,country]
- Picture[pid,topic,album]
- Likes[ssn,album]

Example (cont’d)

- Person[ssn,country]
- Picture[pid,topic,album]
- Likes[ssn,album]
**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. Loras, Thorne 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:
  - LogicBox’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.