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>
</tbody>
</table>

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

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:

SQL

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

Logic (PC)

```
(s(y,n,'Avia') ∧ C(z,x,l) ∧ T(y,z,g))
```

Logic Programming (Datalog)

```
Q(s) ← S(y,n,'Avia'), C(z,x,l), T(y,z,g).
```

Algebra (RA)

```
πC.name (σS.name = 'Avia', S.IID = T.sID, T.cnum = C.number)
```

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 (enrichment)
**Why RA?**

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

**RA vs Other QLs**

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

**Relation Schema**

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

**Tuples**

- Let \( s \) be a relation schema \((\text{att}_1:\text{dom}_1,...,\text{att}_k:\text{dom}_k)\)
- A tuple (over \( s \)) is a sequence \((v_1,...,v_k)\) of values \( v_i \), where each \( v_i \) is in \( \text{dom}_i \)
  - That is, a tuple is an element of \( \text{dom}_1 \times ... \times \text{dom}_k \)
  - Example: \((861,\text{Alma},2)\)

**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 \( s \)

**Ignoring Domains**

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

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

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

\[
\begin{align*}
\text{sid} & \quad \text{name} & \quad \text{year} \\
861 & \quad \text{Alma} & \quad 2 \\
753 & \quad \text{Amir} & \quad 1 \\
955 & \quad \text{Ahuva} & \quad 2 \\
\end{align*}
\]

\[
\begin{align*}
\text{cid} & \quad \text{topic} \\
23 & \quad \text{PL} \\
45 & \quad \text{DB} \\
76 & \quad \text{OS} \\
\end{align*}
\]

\[
\begin{align*}
\text{sid} & \quad \text{cid} \\
861 & \quad 23 \\
753 & \quad 45 \\
955 & \quad 76 \\
\end{align*}
\]

Databases

- A database schema is a finite set of relation names, each mapped to a relation schema
  - Example: \( \text{Student}(\text{sid, name, year}) \), \( \text{Course}(\text{cid, topic}) \), \( \text{Studies}(\text{sid, cid}) \)
  - (Constraints ignored for now)

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

\[
\begin{array}{ccc}
\text{Student} & \text{Course} & \text{Studies} \\
\hline
\text{sid} & \text{name} & \text{year} \\
861 & \text{Alma} & 2 \\
753 & \text{Amir} & 1 \\
955 & \text{Ahuva} & 2 \\
\text{cid} & \text{topic} \\
23 & \text{PL} \\
45 & \text{DB} \\
76 & \text{OS} \\
\text{sid} & \text{cid} \\
861 & 23 \\
753 & 45 \\
955 & 76 \\
\end{array}
\]

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:
  \[
  o_1 \left( o_2 \left( x \right), o_1 \left( y, o_4 \left( x, z \right) \right) \right)
  \]
- Examples:
  - Ring of integers: \( (\mathbb{Z}, +, \cdot) \)
  - Boolean algebra: \( (\{\text{true, false}\}, \&, \lor, \neg) \)
  - 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 using the primitive ones
  - For example, intersection via difference

Outline

- Background
  - The Primitive Operators
  - Implied Operators
    - Joins
    - Division
    - Equivalence & Independence
  - Taste of Query Optimization (enrichment)
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>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>

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

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

More tuples (includes duplicates)

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>80</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{course}=\text{DB}}(R) = \]

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

Q2: If \(R\) has 1000 tuples, how many tuples can \(\sigma_{\text{course}=\text{DB}}(R)\) 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”

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 (\(\land, \lor, \neg\))
    • item1 = item2
    • price \(\geq\) 500 \(\land\) price \(\leq\) budget
• Legal input: A relation \(R\) 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 \(\{t | t \in R \text{ and } t \vDash c\}\)

Q2: If \(R\) has 1000 tuples, how many tuples can \(\sigma_c(R)\) have?
Renaming by Example

\[ R = \{(\text{Alma}, 1, \text{DB}, 80), (\text{Alma}, 1, \text{PL}, 94), (\text{Ahuva}, 2, \text{DB}, 72)\} \]

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

\[ \{(\text{Alma}, 1, \text{DB}, 80), (\text{Alma}, 1, \text{PL}, 94), (\text{Ahuva}, 2, \text{DB}, 72)\} \]

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 having A replaced by B
  - The tuple set of R

Q3: If R has 1000 tuples, how many tuples can \( \rho_{A/B}(R) \) have?

Union and Difference by Example

\[ R = \{(\text{Alma}, 1, \text{DB}, 80), (\text{Alma}, 1, \text{PL}, 94), (\text{Ahuva}, 2, \text{DB}, 72)\} \]

\[ S = \{(\text{Amir}, 3)\} \]

\[ R \cup S = \{(\text{Alma}, 1, \text{DB}, 80), (\text{Alma}, 1, \text{PL}, 94), (\text{Ahuva}, 2, \text{DB}, 72), (\text{Amir}, 3)\} \]

\[ R \setminus S = \{(\text{Alma}, 1, \text{DB}, 80), (\text{Alma}, 1, \text{PL}, 94), (\text{Ahuva}, 2, \text{DB}, 72)\} \]

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

Q4: If each of R and S have 1000 tuples, how many tuples can be in \( R \cup S \)? \( R \setminus S \)?

Cartesian Product by Example

\[ R = \{(\text{861}, \text{Alma}, 2), (\text{753}, \text{Amir}, 1), (\text{955}, \text{Ahuva}, 2)\} \]

\[ S = \{(\text{23}, \text{PL}), (\text{45}, \text{DB}), (\text{76}, \text{OS})\} \]

\[ R \times S = \]

\[ \{(\text{861}, \text{Alma}, 2, 23, \text{PL}), (\text{861}, \text{Alma}, 2, 45, \text{DB}), (\text{861}, \text{Alma}, 2, 76, \text{OS}), (\text{753}, \text{Amir}, 1, 23, \text{PL}), (\text{753}, \text{Amir}, 1, 45, \text{DB}), (\text{753}, \text{Amir}, 1, 76, \text{OS}), (\text{955}, \text{Ahuva}, 2, 23, \text{PL}), (\text{955}, \text{Ahuva}, 2, 45, \text{DB}), (\text{955}, \text{Ahuva}, 2, 76, \text{OS})\} \]

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

Q5: If each of R and S has 1000 tuples, how many tuples can be in \( R \times S \)?
For Cartesian product of named relations (e.g., R, S), we actually allow common attributes, and implicitly assume their renaming to same attribute.

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

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

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

Example:

\[
\sigma_{\text{topic} = \text{DB}}(\text{Course}) \times \sigma_{\text{cid} = \text{cid1}}(\text{Studies})
\]

becomes

\[
\sigma_{\text{topic} = \text{DB}}(\text{Course}) \times \sigma_{\text{cid} = \text{cid1}}(\text{Studies})
\]

\( \sigma_{\text{topic} = \text{DB}}(\text{Course}) \times \sigma_{\text{cid} = \text{cid1}}(\text{Studies}) \)

Names of students who study DB:

\[
\pi_{\text{name} = \text{name}}(\text{Student} \times \pi_{\text{cid} = \text{cid1}}(\sigma_{\text{topic} = \text{DB}}(\text{Course} \times \rho_{\text{cid}/\text{cid}}(\text{Studies}))))
\]

\[
\pi_{\text{name} = \text{name}}(\text{Student} \times \pi_{\text{cid} = \text{cid1}}(\sigma_{\text{topic} = \text{DB}}(\text{Course} \times \rho_{\text{cid}/\text{cid}}(\text{Studies}))))
\]
Outline

• Background
• The Primitive Operators
  ▶ Implied Operators
    • Joins
    • Division
  • Equivalence & Independence
• Taste of Query Optimization (enrichment)

Outline

• Background
• The Primitive Operators
  ▶ Implied Operators
    • Joins
    • Division
  • Equivalence & Independence
• Taste of Query Optimization (enrichment)

Implied Operators

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

Joins

• Cartesian product is rarely standalone without selection, and is commonly followed by projection
• The combination $\pi \sigma \times$ is referred to generally as "join"
• There are several common cases that apply specific selections and projections, which we introduce here

Conditional Join

• Binary operator $R \bowtie_c S$ where $c$ is a condition over the header of $R \times S$
• Shorthand notation for:

  $$\sigma_c(R \times S)$$

• Example: $R \bowtie_{a=b \land c<d} S$
Theta Join and Equijoin

- **Theta join** is a special case of conditional join $\bowtie_c$ where $c$ has the form $A \theta B$ or $A \theta v$ where $A$ and $B$ are attributes, $v$ 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

Natural Join $\bowtie$

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

- Should we care about which new attribute names are used in the renaming?
  - No! They disappear anyway...

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
  \[\pi_{A_1,...,A_m}(R \bowtie S)\]
  where $(A_1,...,A_m)$ is the header of $R$.
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 $R \gg S$ 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 (enrichment)

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

Formal Definition

- **Legal input:** $(R,S)$ such that $R$ has all the attributes of $S$
- $R \bowtie S$ is the relation $T$ with:
  - The header of $R$, with all attributes of $S$ removed; let it be $X$
  - Tuple set $\{t \in \pi_X R \mid (t,s) \in R \text{ for all } s \in S\}$
- Note: "$(t,s) \in R$" 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>AI</td>
</tr>
</tbody>
</table>

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

**Q6**: If \( R \) has 1000 tuples and \( S \) has 100 tuples, how many tuples can be in \( R \div S \)?

**Q7**: 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>AI</td>
</tr>
</tbody>
</table>

**Who took all core courses?**

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

### Examples of Inexpressible Queries

Some very useful queries **cannot** be expressed in RA!

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

(How can one prove inexpressiveness?? Later in the course...)

### R+S in Primitive RA

\[ R(X,Y) \]
\[ S(Y) \]

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

Each \( X \) of \( R \) w/ each \( Y \) of \( S \)

\( (X,Y) \) s.t. \( X \) in \( R \), \( Y \) in \( S \), but \( (X,Y) \) not in \( R \)

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

### Outline

- Background
- The Primitive Operators
- Implied Operators
  - Joins
  - Division
- Equivalence & Independence
- Taste of Query Optimization (enrichment)
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:
  - $\pi_{\text{sid}}(\sigma_{\text{sid} = \text{stud}}(\text{Student} \times \rho_{\text{sid}/\text{stud}}(\text{Studies})))$

<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>topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td></td>
</tr>
</tbody>
</table>

Query Result

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

<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>topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td></td>
</tr>
</tbody>
</table>

Equivalence of RA Expressions

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

<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>topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td></td>
</tr>
</tbody>
</table>

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 $R(A,B), S(A,B), T(A,B)$
  - $\sigma_{A \leftarrow \varphi}(R \bowtie S) \equiv \sigma_{A \leftarrow \varphi}(R) \bowtie \sigma_{A \leftarrow \varphi}(S)$ (selection push)
  - $\pi_{A}(R \cup S) \equiv \pi_{A}(R) \cup \pi_{A}(S)$
  - $(R \bowtie S) \bowtie T \equiv (T \bowtie S) \bowtie R$

Q8: Is it true that $\rho_{R \bowtie A} \pi_{R}(R \times S) \equiv \pi_{\text{R}}$?

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 $\varphi(I) \subseteq \psi(I)$

<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>topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td></td>
</tr>
</tbody>
</table>

Q9: How does containment relate to equivalence?

Q10: How do we prove containment? equivalence?
6 Primitive Operators

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

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

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

\[ \pi \sigma \rho \times \cup \setminus \]

**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 $P$

We prove independence of the other operators in class

Independence of Union

1. Fix a schema $S$ and an instance $I$ over $S$
   $- S = R(A), S(A) \land I = \{R(0), S(1)\}$
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 $P$
   $- \text{Induction base: } R$ and $S$ have $\#tuples < 2$
   $- \text{Inductive: if } \varphi_1(I)$ and $\varphi_2(I)$ have $\#tuples < 2$, then so do
     $\sigma_c(\varphi_1(I)), \pi_c(\varphi_1(I)), \rho_{A\rightarrow B}(\varphi_1(I)), \varphi_1(I) \times \varphi_2(I), \varphi_1(I) \setminus \varphi_2(I)$
4. Find an expression $\psi$ such that $\psi$ uses $o$ and $\psi(I)$ violates $P$
   $- \psi = R \cup S$

Outline

• Background
• The Primitive Operators
• Implied Operators
  • Joins
  • Division
• Equivalence & Independence
  ▶ Taste of Query Optimization (enrichment)

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)

Task: find Israelis who like albums with dog pictures

Which of the equivalent expressions is more efficient to apply?

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

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

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

$\pi_{\text{ssn}}((\text{Likes} \bowtie \sigma_{\text{country}=\text{Israel}}(\text{Person}) \bowtie \sigma_{\text{album}=\text{dog}}(\text{Picture})))$

$\pi_{\text{ssn}}(\sigma_{\text{country}=\text{Israel}}(\text{Person}) \bowtie (\pi_{\text{album}}(\text{Likes} \bowtie \pi_{\text{album}}(\sigma_{\text{topic}=\text{dog}}(\text{Picture}))))$
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 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 \( R_2 \))
  - \( X_i = X \setminus R_1 \)
  - \( X_i = X \setminus R_2 \)
- (Abuse of notation – we mix attribute sequences with attributes sets)
- We would like to push projections into the join, that is:
  \[ \pi_X(\pi_{X_1}(R_1 \bowtie R_2)) \]
  - Which \( Z_i \) and \( Z_c \) can work (equivalence preserved)?

Correct Projection Push

- \( \pi_X(R_1 \bowtie R_2) \equiv \pi_X(R_1) \bowtie \pi_X(R_2) \)?
- \( \pi_X(R_1, R_2) \equiv \pi_X(R_1) \bowtie \pi_X(R_2) \)?
- \( \pi_X(R_1 \bowtie R_2) \equiv \pi_X(\pi_Y(R_1) \bowtie \pi_Y(R_2)) \)?

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

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...

Pushing Down the Expression Tree

- 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...

Pushing Down the Expression Tree

- \( \sigma_{c \land d}(R_1 \bowtie R_2) \)
- \( \pi_X(R_1 \bowtie R_2) \)
- \( \pi_X(\pi_{X_1}(R_1 \bowtie R_2)) \)
- \( \pi_X(\pi_{X_1}(R_1) \bowtie \pi_{X_1}(R_2)) \)

Y = R_1 \cap R_2
X_i = X \setminus R_1
X_i = X \setminus 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

\[
\begin{align*}
\pi_{\text{ssn}}( \pi_{\text{country}=\text{Israel}}(\text{Person} \bowtie (\text{Likes} \bowtie (\text{Picture})))) \\
\pi_{\text{ssn}}( \pi_{\text{country}=\text{Israel}}(\text{Person} \bowtie (\text{Likes} \bowtie (\text{Picture})))) \\
\pi_{\text{ssn}}( \pi_{\text{country}=\text{Israel}}(\text{Person} \bowtie \pi_{\text{album}}(\text{Likes} \bowtie (\pi_{\text{topic}=\text{dog}}(\text{Picture})))))) \\
\pi_{\text{ssn}}( \pi_{\text{country}=\text{Israel}}(\text{Person} \bowtie \pi_{\text{album}}(\text{Likes} \bowtie (\pi_{\text{topic}=\text{dog}}(\text{Picture})))))) \\
\end{align*}
\]

Perspective on Query-Plan Optimization

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

Note on Alternative Approaches

- In a recent line of research, several alternative algorithms for RA computation are developed
- These algorithm do not construct intermediate results from sub-queries
  - Rather, simultaneously scan all input relations
- More reading:
  - Stanford’s Minesweeper [Ngo, Nguyen, Re, Rudra: Beyond worst-case analysis for joins with minesweeper. PODS 2014: 234-245]
- Not discussed in this course

Answers to Questions

- Q1: between 1 and 1000
- Q2: between 0 and 1000
- Q3: 1000
- Q4: between 1000 and 2000; 0 and 1000
- Q5: 1,000,000
- Q6: between 0 and 10
- Q7: 0
- Q8: No, since S can be empty
- Q9: Equivalence is containment in both directions
- Q10: Take a DB $D$ and a tuple $t$ in $Q_d(D)$, and prove that $t$ is also in $Q_d(D)$
- Q11: Show a counterexample, consisting of a database $D$ with the results $Q_d(D)$ and $Q_e(D)$