RELATIONAL ALGEBRA: OUTLINE

Relational Algebra

Basic Operations
- Selection
- Projection
- Union
- Set difference
- Cartesian product

Additional Operations
- Intersection
- Join
- Assignment
- Rename
- Div
Two mathematical query languages form the basis for “real” relational query languages (e.g., SQL) and for implementation.

- **Relational Algebra**
  - Procedural (**step-by-step**).
  - Need to describe how to compute a query result.

- **Relational Calculus**
  - Non-procedural (**declarative**).
  - Only need to describe what query result is wanted, not how to compute it.

Relational algebra is very useful for representing and optimizing query execution plans.

Understanding relational algebra is the **key** to understanding SQL and how it is processed!
The relational algebra is an algebra whose

- **operands** are either **relations** or **variables** that represent relations.
- **operations** perform **common, basic manipulations** of relations.

☞ A relational algebra expression is evaluated from the inside-out.

**Closure Property**

- Relational algebra is **closed** with respect to the relational model.

☞ Each operation manipulates one or more relations and returns a relation as its result.

Due to the closure property, operations can be **composed**!
Relational Algebra

Basic Operations
- Selection
- Projection
- Union
- Set difference
- Rename
- Cartesian product

Additional Operations
- Intersection
- Join
- Assignment
- Div
**INTERSECTION**: \( \cap \)

**Query**: Find tuples that appear in both Plane\(_1\) and Plane\(_2\).

<table>
<thead>
<tr>
<th>Plane(_1)</th>
<th>Plane(_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>company</td>
<td>model</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Airbus</td>
<td>A310</td>
</tr>
<tr>
<td>Airbus</td>
<td>A320</td>
</tr>
<tr>
<td>Airbus</td>
<td>A330</td>
</tr>
<tr>
<td>Airbus</td>
<td>A340</td>
</tr>
<tr>
<td>Boeing</td>
<td>B747</td>
</tr>
<tr>
<td>Boeing</td>
<td>B777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>company</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comac</td>
<td>C929</td>
</tr>
<tr>
<td>Comac</td>
<td>C939</td>
</tr>
<tr>
<td>Boeing</td>
<td>B747</td>
</tr>
<tr>
<td>Boeing</td>
<td>B777</td>
</tr>
</tbody>
</table>

\[
\text{INTERSECTION: } \cap \]

\[
\text{Query: Find tuples that appear in both Plane\(_1\) and Plane\(_2\).}
\]

\[
\begin{align*}
\text{Plane\(_1\)} & \cap \text{Plane\(_2\)} = \\
\text{Airbus} & \text{A310} \\
\text{Airbus} & \text{A320} \\
\text{Airbus} & \text{A330} \\
\text{Airbus} & \text{A340} \\
\text{Boeing} & \text{B747} \\
\text{Boeing} & \text{B777}
\end{align*}
\]

\[
\begin{align*}
\text{company} & \text{ model} \\
\text{Boeing} & \text{B747} \\
\text{Boeing} & \text{B777}
\end{align*}
\]
Intersection is not a primitive operation.

\[ R \cap S = ((R \cup S) - (R - S)) - (S - R) \]

Compute all tuples belonging to \( R \) or \( S \).

Remove the ones that belong only to \( R \).

Remove the ones that belong only to \( S \).

Also: \( R \cap S = R \setminus (R \setminus S) \)
OUTER JOIN

- An extension of the natural join operation that avoids loss of information.

- Computes the natural join and then adds tuples from one relation that do not have matching tuples in the other relation to the result of the join.

- Uses null values to fill in missing information.
  - Recall that null signifies that the value is unknown or does not exist.

  ⚠️ All comparisons involving null are false.
### OUTER JOIN (cont’d)

- Natural join returns only the tuples that match on the join attributes (the “good tuples”).

- The fact that
  - loan L-260 has no borrower is not explicit in the result.
  - customer Ted Hayes holds a non-existent loan L-155 with no amount and no branch is also not explicit.

<table>
<thead>
<tr>
<th>Loan</th>
<th>amount</th>
<th>branch Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>30000</td>
<td>Central</td>
</tr>
<tr>
<td>L-260</td>
<td>170000</td>
<td>Tsimshatsui</td>
</tr>
<tr>
<td>L-230</td>
<td>40000</td>
<td>Central</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Borrower</th>
<th>loan Number</th>
<th>amount</th>
<th>branch Name</th>
<th>client Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pat Lee</td>
<td>L-170</td>
<td>30000</td>
<td>Central</td>
<td>Pat Lee</td>
</tr>
<tr>
<td>Mary Kwan</td>
<td>L-230</td>
<td>40000</td>
<td>Central</td>
<td>Mary Kwan</td>
</tr>
<tr>
<td>Ted Hayes</td>
<td>L-155</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LEFT OUTER JOIN:**

Adds to the natural join all tuples in the left relation (Loan) that did not match with any tuple in the right relation (Borrower) and fills in null for the missing information.

<table>
<thead>
<tr>
<th>Loan</th>
<th>Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>client Name</td>
</tr>
<tr>
<td>L-170</td>
<td>Pat Lee</td>
</tr>
<tr>
<td>L-260</td>
<td>Mary Kwan</td>
</tr>
<tr>
<td>L-230</td>
<td>Ted Hayes</td>
</tr>
</tbody>
</table>

The result now shows that loan L-260 has no borrower.
RIGHT OUTER JOIN:

Adds to the natural join all tuples in the right relation (Borrower) that did not match with any tuple in the left relation (Loan) and fills in null for the missing information.

<table>
<thead>
<tr>
<th>Loan Number</th>
<th>amount</th>
<th>branch Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>30000</td>
<td>Central</td>
</tr>
<tr>
<td>L-260</td>
<td>170000</td>
<td>Tsimshatsui</td>
</tr>
<tr>
<td>L-230</td>
<td>40000</td>
<td>Central</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Borrower</th>
<th>Loan Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>client Name</td>
<td>loan Number</td>
</tr>
<tr>
<td>Pat Lee</td>
<td>L-170</td>
</tr>
<tr>
<td>Mary Kwan</td>
<td>L-230</td>
</tr>
<tr>
<td>Ted Hayes</td>
<td>L-155</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loan Number</th>
<th>amount</th>
<th>branch Name</th>
<th>client Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>30000</td>
<td>Central</td>
<td>Pat Lee</td>
</tr>
<tr>
<td>L-230</td>
<td>40000</td>
<td>Central</td>
<td>Mary Kwan</td>
</tr>
</tbody>
</table>

The result now shows that loan L-155 has no amount and no branch.
FULL OUTER JOIN:

Adds to the natural join all tuples in both relations that did not match with any tuples in the other relation and fills in null for missing information.

The result now shows both that

- loan L-260 has no borrower.
- loan L-155 has no amount and no branch.
ASSIGNMENT:

- Works like assignment in programming languages.
- The relation variable assigned to can be used in subsequent expressions.
- Allows a query to be written as a sequential program consisting of a series of assignments followed by an expression whose value is the result of the query.
- Useful for expressing complex queries.
**RENAMING: ρ**

- Assigns a name to, or renames the attributes in, a relational-algebra expression.

  \[ ρ_x(E) \] assigns name \( x \) to the result of \( E \)

  \[ ρ_{x(A_1, A_2, ..., A_n)}(E) \] assigns name \( x \) to the result of \( E \) and renames the attributes of \( E \) as \( A_1, A_2, ..., A_n \)

Renaming is necessary when taking the Cartesian product of a table with itself.
RENAMING: $\rho$

- If attributes or relations have the same name it may be convenient to rename one
  \[ \rho(R'(N_1 -> N'_1, N_n -> N'_n), R) \]

- The new relation $R'$ has the same instance as $R$, but its schema has attribute $N'_i$ instead of attribute $N_i$.
- **Example:** $\rho(Staff(Name -> Family_Name, Salary -> Gross_salary), Employee)$

- Necessary if we need to perform a cartesian product or join of a table with itself.

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Emp_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark</td>
<td>150000</td>
<td>1006</td>
</tr>
<tr>
<td>Gates</td>
<td>5000000</td>
<td>1005</td>
</tr>
<tr>
<td>Jones</td>
<td>50000</td>
<td>1001</td>
</tr>
<tr>
<td>Peters</td>
<td>45000</td>
<td>1002</td>
</tr>
<tr>
<td>Phillips</td>
<td>25000</td>
<td>1004</td>
</tr>
<tr>
<td>Rowe</td>
<td>35000</td>
<td>1003</td>
</tr>
<tr>
<td>Warnock</td>
<td>500000</td>
<td>1007</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Family_Name</th>
<th>Gross_Salary</th>
<th>Emp_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark</td>
<td>150000</td>
<td>1006</td>
</tr>
<tr>
<td>Gates</td>
<td>5000000</td>
<td>1005</td>
</tr>
<tr>
<td>Jones</td>
<td>50000</td>
<td>1001</td>
</tr>
<tr>
<td>Peters</td>
<td>45000</td>
<td>1002</td>
</tr>
<tr>
<td>Phillips</td>
<td>25000</td>
<td>1004</td>
</tr>
<tr>
<td>Rowe</td>
<td>35000</td>
<td>1003</td>
</tr>
<tr>
<td>Warnock</td>
<td>500000</td>
<td>1007</td>
</tr>
</tbody>
</table>
Let \( A \) have two fields \( x \) and \( y \)

Let \( B \) have one field \( y \)

\( A/B \) contains all \( x \) tuples, such that for every \( y \) tuple in \( B \) there is a \( xy \) tuple in \( A \)

\[
\begin{array}{|c|c|}
\hline
x & y \\
\hline
s1 & p1 \\
\hline
s1 & p2 \\
\hline
s1 & p3 \\
\hline
s1 & p4 \\
\hline
s2 & p1 \\
\hline
s2 & p2 \\
\hline
s3 & p2 \\
\hline
s4 & p2 \\
\hline
s4 & p4 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
y \\
\hline
p2 \\
\hline
p4 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
x \\
\hline
s1 \\
\hline
s4 \\
\hline
\end{array}
\]

\[
A/B = \frac{A}{B}
\]
DIVISION: \( / \) (Cont.)

• Compute all possible combinations of the first column of A and B.
• Then remove those rows that exist in A
• Keep only the first column of the result. These are the disqualified values

\[
\pi_x \left( (\pi_x(A) \times B) - A \right)
\]

• \( A/B \) is the first column of A except the disqualified values

\[
A/B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)
\]
**DIVISION: / (Cont.)**

\[
\pi_x(A) = \pi_x(\quad ) = \pi_x(A) \times B = \pi_x(A) \times B =
\]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s2</td>
<td>p4</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p4</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>
\[
(\pi_x(A) \times B) - A = \\

\begin{array}{|c|c|}
\hline
x & y \\
\hline
s1 & p2 \\
s1 & p4 \\
s2 & p2 \\
s2 & p4 \\
s3 & p2 \\
s3 & p4 \\
s4 & p2 \\
s4 & p4 \\
\hline
\end{array}
\]

\[
= \\

\begin{array}{|c|c|}
\hline
x & y \\
\hline
s1 & p1 \\
s1 & p2 \\
s1 & p3 \\
s1 & p4 \\
s2 & p1 \\
s2 & p2 \\
s2 & p4 \\
s3 & p2 \\
s3 & p4 \\
s4 & p2 \\
s4 & p4 \\
\hline
\end{array}
\]

\[
\pi_x(A) - \pi_x(\pi_x(A) \times B) - A) = \\

\begin{array}{|c|}
\hline
x \\
\hline
s1 \\
s2 \\
s3 \\
s4 \\
\hline
\end{array}
\]

\[
= \\

\begin{array}{|c|}
\hline
x \\
\hline
s2 \\
s3 \\
\hline
\end{array}
\]

\[
= \\

\begin{array}{|c|}
\hline
x \\
\hline
S1 \\
S4 \\
\hline
\end{array}
\]
### DIVISION EXAMPLE

Find the Employment numbers of the pilots who can fly all MD planes

\[ \text{Can}_\text{Fly} / \pi_{\text{Model}_\text{No}}(\sigma_{\text{Maker}=\text{MD} \text{ Plane}}) \]

<table>
<thead>
<tr>
<th>Emp_No</th>
<th>Model_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>B727</td>
</tr>
<tr>
<td>1001</td>
<td>B747</td>
</tr>
<tr>
<td>1001</td>
<td>DC10</td>
</tr>
<tr>
<td>1002</td>
<td>A320</td>
</tr>
<tr>
<td>1002</td>
<td>A340</td>
</tr>
<tr>
<td>1002</td>
<td>B757</td>
</tr>
<tr>
<td>1002</td>
<td>DC9</td>
</tr>
<tr>
<td>1003</td>
<td>A310</td>
</tr>
<tr>
<td>1003</td>
<td>DC9</td>
</tr>
<tr>
<td>1003</td>
<td>DC10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maker</th>
<th>Model_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airbus</td>
<td>A310</td>
</tr>
<tr>
<td>Airbus</td>
<td>A320</td>
</tr>
<tr>
<td>Airbus</td>
<td>A330</td>
</tr>
<tr>
<td>Airbus</td>
<td>A340</td>
</tr>
<tr>
<td>Boeing</td>
<td>B727</td>
</tr>
<tr>
<td>Boeing</td>
<td>B747</td>
</tr>
<tr>
<td>Boeing</td>
<td>B757</td>
</tr>
<tr>
<td>MD</td>
<td>DC10</td>
</tr>
<tr>
<td>MD</td>
<td>DC9</td>
</tr>
</tbody>
</table>

Emp_No
---
1003
RELATIONAL ALGEBRA: SUMMARY

- Defines a set of algebraic operations that operate on relations and output relations as their result.

- The operations can be combined to express queries.

- The operations can be divided into:
  - basic operations.
  - additional operations that either
    - can be expressed in terms of the basic operations or
    - add further expressive power to the relational algebra.
COMP 3311: SYLLABUS

✓ Introduction
✓ Entity-Relationship (E-R) Model and Database Design
✓ Relational Algebra

⇒ Structured Query Language (SQL)
   Relational Database Design
   Storage and File Structure
   Indexing
   Query Processing
   Query Optimization
   Transactions
   Concurrency Control
   Recovery System
   NoSQL Databases