CS 133: Databases

Fall 2016
Lec 7 – 9/21
Prof. Beth Trushkowsky

Goals for Today

- Learn about how relational algebra operates on sets of tuples
- Compose the basic relational algebra operators to form queries on relations
- Understand the goals for Lab 2

Example: Logical Query Plan

- Example: college database
  Students(SID, name, gpa)
  Enrolled(SID, CID, grade)

  SELECT S.name, E.CID
  FROM Students S, Enrolled E
  WHERE S.SID=E.SID;

  Get tuples from Students
  Get tuples from Enrolled
  Combine
  Pull out name and CID fields
  When query optimizer forms physical query plan, it will consider available implementations and/or choices for the logical operators!

Operations on Sets of Tuples

- Relational model: data represented as sets of tuples (i.e., relations)
- Relational algebra: an algebra on sets of tuples
  - Used to express queries about those relations
  - I.e., a query language

- Note: Sets \( \neq \) Bags
  - Sets: relations have no duplicate tuples
  - Bags, aka multi-sets: duplicate tuples possible

Example: Edgar F. Codd

Turing award, 1981
Formal Relational Query Languages

- **Query languages** allow manipulation and retrieval of data from a database
  - Query languages ≠ programming languages!

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:
  - **Relational Algebra**: More operational, very useful for representing query execution plans.
  - **Relational Calculus**: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

We’ll see some differences between SQL and relational algebra

Preliminaries

A query is applied to *relation instances*, and the result of a query is also a relation instance.

![Diagram showing query execution]

Depending on the query, output relation schema may be the same or different than input schema

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>login</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Alice</td>
<td>alicious</td>
<td>3.4</td>
</tr>
<tr>
<td>67</td>
<td>Bob</td>
<td>bobtastic</td>
<td>3.9</td>
</tr>
<tr>
<td>78</td>
<td>Carl</td>
<td>carl</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Example Instances

**Sailing Database:**
*Boats, Sailors, Reserves*

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Boats**

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

What is “an Algebra”??

Mathematical system consisting of:

- Variables or constants
- Operands
- Operators

![Diagram showing components of an algebra]

Example:

Arithmetic: \( x, y, 15 \) \( +, -, \ast, / \)

Relational: Relations *Let’s see…*

An algebra allows us to build expressions by applying operators to operands and/or other expressions

Example Instances

**Sailing Database:**
*Boats, Sailors, Reserves*

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**Boats**

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Relational Algebra: 5 Basic Operations

- **Selection** (\(\sigma\)) Selects a subset of rows from relation (horizontal).
- **Projection** (\(\pi\)) Retains only wanted columns from relation (vertical).
- **Cross-product** (\(\times\)) Allows us to combine two relations.
- **Set-difference** (\(-\)) Tuples in relation1, but not in relation2.
- **Union** (\(\cup\)) Tuples in relation1 and/or in relation2.

Since each operation returns a relation, operations can be composed!

Selection (\(\sigma\)) – Horizontal Restriction

- Selects rows that satisfy selection condition.
  - Note: not the same thing as SELECT in SQL
  - Can have several conditions, combined with \(\lor\) (or), \(\land\) (and)

- **Schema** of result is same as that of the input relation.

Examples: Projection

- **Examples**: \(\pi_{\text{age}}(S2)\), \(\pi_{\text{name}, \text{rating}}(S2)\)
- Retains only attributes that are in the “projection list”.

- **Schema** of result:
  - exactly the fields in the projection list, with the same names that they had in the input relation

- Projection operator has to eliminate duplicates
  - How would duplicates arise?
  - Note: real systems typically don’t do duplicate elimination in SQL unless the user explicitly asks for it (why not?)
Composing Operators

- Output of a Relational Algebra operator is a relation, so...
  - Can use result as input to another Relational Algebra operator

\[
\pi \text{name, rating} (\sigma \text{rating} > 8 (S2))
\]

Union and Set-Difference

- Take two input relations, which must be *union-compatible*:
  - Same number of fields
  - “Corresponding” fields have the same type
  
  For which, if any, will the operator have to do duplicate elimination?

### Union

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[S1 \cup S2\]

### Set-difference

\[S1 - S2\]

\[S2 - S1\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{name} & \text{rating} \\
\hline
22 & dustin & 7 \\
31 & lubber & 8 \\
58 & rusty & 10 \\
44 & guppy & 5 \\
28 & puppy & 9 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{name} & \text{rating} \\
\hline
28 & puppy & 9 \\
31 & lubber & 8 \\
44 & guppy & 5 \\
58 & rusty & 10 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{name} & \text{rating} \\
\hline
22 & dustin & 7 \\
31 & lubber & 8 \\
58 & rusty & 10 \\
44 & guppy & 5 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{name} & \text{rating} \\
\hline
28 & puppy & 9 \\
44 & guppy & 5 \\
\hline
\end{array}
\]
Cross-Product

• **S1 x R1:** Each tuple of S1 paired with each tuple of R1

How many tuples in the result?

• **Result schema** has one field per field of S1 and R1, with field names ‘inherited’ if possible.
  – *May have a naming conflict:* Both S1 and R1 have a field with the same name.
  – In this case, can use the renaming operator:

\[ \rho (C(1 \rightarrow \text{sid}1, 5 \rightarrow \text{sid}2), \text{S1} \times \text{R1}) \]

Rename the first and fifth fields in the relation that results from S1 x R1

Compound Operators

• In addition to the five basic operators, there are several additional “Compound Operators”
  – These add no computational power to the language, but are useful short-hands
  – Can be expressed solely with the basic operators

• Intersection, Join, Division

Cross Product Example

<table>
<thead>
<tr>
<th>S1</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R1</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td></td>
</tr>
</tbody>
</table>

\[ \rho (C(1 \rightarrow \text{sid}1, 5 \rightarrow \text{sid}2), \text{S1} \times \text{R1}) = \]

<table>
<thead>
<tr>
<th>sid1</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Intersection

**Intersection** takes two input relations, which must be union-compatible.

• *How to express it using only basic operators?*

  e.g., \( R \cap S = R - (R - S) \)

<table>
<thead>
<tr>
<th>S1</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
</tbody>
</table>

\( S1 \cap S2 \)
Join (⋈)

- Joins are compound operators involving
  - cross product,
  - selection,
  - and (sometimes) projection.

- Most common type of join is a natural join (often just called “join”). R⋈S conceptually is:
  - Compute the cross product R × S
  - Select rows where attributes that appear in both relations have equal values
  - Project all unique attributes and one copy of each of the common ones.

Usually done much more efficiently than this

Other Types of Joins

- **Condition Join (or “theta-join”):**
  \[ R \bowtie_c S = \sigma_c (R \times S) \]

  - Result schema same as that of cross-product.
  - (May have fewer tuples than cross-product)

- **Equi-Join:** Special case: condition c contains only conjunction of equalities.

Natural Join Example

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

“Theta” Join Example

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

What if R1 also contained the tuple: (22, 105, 12/13/96)
Compound Operator: Division

- Useful for expressing “for all” queries like: 
  *Find names of sailors who have reserved all boats.*

- For A/B, attributes of B are subset of attributes of A
  - May need to use “project” operator first
  - E.g., let A have 2 fields, x and y; B has only field y:
    
    \[ A/B = \{ (x) | \forall (y) \in B (\exists (x,y) \in A) \} \]
    
    *A/B* contains all x tuples such that for every y tuple in B, there exists a tuple x,y in A

Examples of Division A/B

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</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>sno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td></td>
</tr>
</tbody>
</table>

A/B1 A/B2 A/B3

Exercise 8: Division

- Write relational algebra expression using division to find names of sailors who have reserved all boats.

- Hints
  - What could you assign to A, B, x, y?
  - Try just getting sids of sailors who have reserved all boats first

Expressing A/B Using Basic Operators

- Division is not an essential operator; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)

- *Idea:* For A/B, compute all x values that are not ‘disqualified’ by some y value in B.
  - x value is disqualified if by attaching y value from B, we obtain an x,y tuple that is not in A.

Disqualified x values:

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

\[ A/B: \pi_x (A) - \text{Disqualified x values} \]
Lab 2: SimpleDb Operators

- **Goal**: building on Lab 1, be able to perform simple queries over **multiple relations**

- Three parts! **Be sure to submit each on Sakai**
  - Filter and Join: **out tonight, due next Wednesday**
  - Aggregate, HeapFile Mutability
  - Insert and Delete, Buffer Pool Eviction

- Operators will implement the interface `DbIterator`

---

Operators are `DbIterators`

```java
public abstract class Operator implements DbIterator {
    public boolean hasNext() throws DBException, TransactionAborted
    public Tuple next() throws DBException, TransactionAbortedException
    if (next == null) {
        next = fetchNext();
        if (next == null) throw new NoSuchElementException();
    }
    Tuple result = next;
    return result;
}
protected abstract Tuple fetchNext();
```

Your operators will have to implement this

Example: Filter

```java
public class Filter extends Operator {
    public Filter(Predicate p, DbIterator child) {
        super.open();
        A child operator from which to read tuples!
        super.close();
        You will need to call
        Hint: check out Field.compare() See Project.java as an example
    }
    Each tuple will be filtered by p.filter()
}
```

- **Goal**: support queries like
  ```sql
  SELECT *
  FROM table1, table2
  WHERE table1.field1 = table2.field2
  AND table1.id > 5;
  ```
**Join**

Takes a `JoinPredicate`:

```java
public JoinPredicate(int field1, Predicate.Op op, int field2) {
    Which field from the tuple from child1
    Which field from the tuple from child2
}
```

Recall nested-loop algorithm...

*What will the TupleDesc look like for the tuples output by JOIN?*

**Example: Logical Query Plan**

- Example: college database
  - `Students(SID, name, gpa)`
  - `Enrolled(SID, CID, grade)`

```sql
SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.SID=E.SID;
```

- Relational algebra expression?
- How do changes to the R.A. expression change the tree?

Sets of tuples flow upward

- Get tuples from `Students`
- Get tuples from `Enrolled`
- Pull out name and CID fields
- Combine