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

Logical Query Plan Example

• Example: college database

  Students\((\text{SID, name, gpa})\)
  Enrolled\((\text{SID, CID, grade})\)

  \[
  \text{SELECT } S\text{.name, E\text{.CID}} \\
  \text{FROM Students} S, \text{Enrolled} E \\
  \text{WHERE} S\text{.SID}=E\text{.SID};
  \]

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 != Bags**
  
  – Sets: relations have no duplicate tuples
  
  – Bags, aka multi-sets: duplicate tuples possible
Formal Relational Query Languages

- **Query languages** allow manipulation and retrieval of data from a database
  - Query languages \(\neq\) 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

What is “an Algebra” ??

Mathematical system consisting of:

- Variables or constants
- Operands
- Operators
- Procedures that construct new values from given values

Example:

Arithmetic \(x, y, 15\) \(+, -, *, /\)

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

**Boats**

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

**Sailors**

<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>

**Reserves**

<table>
<thead>
<tr>
<th>sid</th>
<th>bname</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>

Preliminaries

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

Relation instance(s) \(\rightarrow\) Query (e.g., algebra or SQL) \(\rightarrow\) Relation instance

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>alias</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>

Query \(\rightarrow\) Relation instance \(\rightarrow\) name Bob
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 relation 1, but not in relation 2.
- **Union** ($\cup$): Tuples in relation 1 and/or in relation 2.

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:**

\[
\begin{array}{cccc}
\text{sid} & \text{name} & \text{rating} & \text{age} \\
28 & \text{yuppy} & 9 & 35.0 \\
31 & \text{lubber} & 8 & 55.5 \\
44 & \text{guppy} & 5 & 35.0 \\
58 & \text{rusty} & 10 & 35.0 \\
\end{array}
\]

\[
\sigma_{\text{rating} > 8}(S2)
\]

Examples: Projection

- **Examples:**

\[
\begin{array}{cccc}
\text{sid} & \text{name} & \text{rating} & \text{age} \\
28 & \text{yuppy} & 9 & 35.0 \\
31 & \text{lubber} & 8 & 55.5 \\
44 & \text{guppy} & 5 & 35.0 \\
58 & \text{rusty} & 10 & 35.0 \\
\end{array}
\]

\[
\pi_{\text{age}}(S2)
\]

\[
\pi_{\text{name}, \text{rating}}(S2)
\]

- **Examples:**

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

- In relational algebra, projection operator **eliminates 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

\begin{align*}
\pi_{\textit{name}, \textit{rating}}(\sigma_{\textit{rating} > 8}(S))
\end{align*}

Union and Set-Difference

- Take two input relations, which must be \textit{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

\begin{align*}
S_1 \cup S_2
\end{align*}

Set-difference

\begin{align*}
S_1 - S_2 \\
S_2 - S_1
\end{align*}
Cross-Product

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

How many tuples will be in the result S1 x R1?

• 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:

\[ \rho \left(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1\right) \]

Cross Product Example

<table>
<thead>
<tr>
<th>S1 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>
</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>

<table>
<thead>
<tr>
<th>R1 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>

\[ \rho \left(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1\right) \]

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

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 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>
</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>

<table>
<thead>
<tr>
<th>S2 sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Join ($\bowtie$)

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

• Most common type of join is a natural join. R $\bowtie$ S conceptually is:
  – Compute the cross product R X 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.
Exercise 5-7: Relational Algebra using Joins

Find names of sailors who’ve reserved boat #103
\[ \pi_{\text{name}}((\sigma_{\text{bid}=103} \text{Reserves}) \bowtie \text{Sailors}) \]

Find names of sailors who’ve reserved a red boat
\[ \pi_{\text{name}}((\sigma_{\text{color}='red'} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

Find sailors who’ve reserved a red and a green boat
\[ \rho (\text{Tempred}, \pi_{\text{sid}}((\sigma_{\text{color}='red'} \text{Boats}) \bowtie \text{Reserves})) \]
\[ \rho (\text{Tempgreen}, \pi_{\text{sid}}((\sigma_{\text{color}='green'} \text{Boats}) \bowtie \text{Reserves})) \]
\[ \pi_{\text{name}}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors}) \]

Relational Calculus

• Tuple Relational Calculus:
  – Variables range over (i.e., get bound to) tuples
  – Answer tuples: an assignment of constants to variables that make an expression evaluate to true
  \[ \{S \mid S \in \text{Sailors} \land S.\text{rating} > 7\} \]
  \[ \{P \mid \exists S \in \text{Sailors}(S.\text{rating} > 7 \land P.\text{name} = S.\text{name} \land P.\text{age} = S.\text{age})\} \]
  Effectively the projected attributes

• Every relational algebra query can be expressed as a safe calculus query, and vice versa

Check out Section 4.3 in the book for more!

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:
  \[ A/B = \left\{ \langle x \rangle \mid \forall \langle y \rangle \in B(\exists \langle x, y \rangle \in A) \right\} \]

A/B contains all x tuples such that for every y tuple in B, there exists a tuple x,y in A

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 tomorrow, due next Wednesday
  – Aggregate, HeapFile Mutability
  – Insert and Delete, Buffer Pool Eviction

• Operators will implement the interface

Dbiterator
Lab 2: Overview

- Goal: support queries like

```
SELECT *
FROM table1, table2
WHERE table1.field1 = table2.field2
AND table1.id > 5;
```

Operators are DbIterators

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

Example: Filter

```java
/** *
 * Filter is an operator that implements a relational select.
 * *
 * public class Filter extends Operator {
 *    /** *
 *     * Constructor accepts a predicate to apply
 *     * and a child operator to read tuples to filter from.
 *     * *
 *     * @param p The predicate to filter tuples with
 *     * @param child The child operator
 *     */
 *    public Filter(Predicate p, DbIterator child) {
 *        ...
 *    }
 */
```

Jump

Join

```
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?
**Logical Query Plan Example**

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

  \[
  \text{SELECT S.name, E.CID} \\
  \text{FROM Students S, Enrolled E} \\
  \text{WHERE S.SID=E.SID;} \\
  \]

Sets of tuples flow upward

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

Relational algebra expression?