Announcements

- Lab 2 on RA on Thursday
  - Do not forget your laptop!

- Homework 1 (Part 1 and 2) have been posted on sakai
  - First deadline next Tuesday: 09/17
  - Parsing XML will take time!

- Next week:
  - Revisit Relational Calculus!
  - New topic: Database internals and indexes!

Today’s topic

- Relational Algebra
- Normalization

A Quick Recap

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]

Relational Algebra (RA)

Relational Algebra (RA)

- A language for querying relational data based on "operators"
- Takes one or more relations as input, and produces a relation as output
  - operator
  - operand
  - semantic
  - so an algebra!

- Since each operation returns a relation, operations can be composed
  - Algebra is "closed"
Relational Algebra

- Basic operations:
  - Selection (σ) Selects a subset of rows from relation
  - Projection (π) Deletes unwanted columns from relation.
  - Cross-product (×) Allows us to combine two relations.
  - Set-difference (\( - \)) Tuples in reln. 1, but not in reln. 2.
  - Union (\( \cup \)) Tuples in reln. 1 or in reln. 2.

- Additional operations:
  - Intersection (\( \cap \))
  - Join (⨝)
  - Division (/)
  - Renaming (\( \rho \))

- Not essential, but (very) useful, especially join!

Example Schema and Instances

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Boats</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

- Projection

- Deletes attributes that are not in projection list.

- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (single) input relation.

- Projection operator has to eliminate duplicates (Set semantic!)

- Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it (performance)

Selection

- Selects rows that satisfy selection condition

- No duplicates in result
  - Because input is a set!

- Schema of result identical to schema of (single) input relation

Composition of Operators

- Result relation can be the input for another relational algebra operation
  - Operator composition

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - “Corresponding” fields must have the same type and same schema as the inputs

- You would lose points if your relations in \( S1 \cup S2 \) are not union compatible!
### Union, Intersection, Set-Difference

**$S_1$**

<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>

**$S_2$**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<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>

**$S_1 - S_2$**

What do we get here?

**$S_1 \cap S_2$**

What do we get here?

---

### Cross-Product

- Each row of $S_1$ is paired with each row of $R$.
- Result schema has one field per field of $S_1$ and $R$, with field names "inherited" if possible.
  - Conflict: Both $S_1$ and $R$ have a field called `sid`.

**$S_1 \times R$**

What do we get here?

---

### Renaming Operator $\rho$

$(\rho_{sid \rightarrow sid_1} S_1) \times (\rho_{sid \rightarrow sid_1} R_1)$

or

$\rho(C(1 \rightarrow sid_1, 5 \rightarrow sid_2), S_1 \times R_1)$

C is the new relation name

- In general, can use $\rho(<Temp>, <RA-expression>)$

---

### Joins

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

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

$S_1 \bowtie_R S_1\bowtie_R R_1$

- Result schema same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
  - We will do join algorithms later

---

### Find names of sailors who've reserved boat #103

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- **Solution 1:**
- **Solution 2:**

---

### Expressing an RA expression as a Tree

You can use either RA expression or a logical query plan in exams.

But in the lab on Thursday, you will use RA expressions in RADB and RATest tools.
Find sailors who’ve reserved a red or a green boat

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

Use of rename operation

• Can identify all red or green boats, then find sailors who’ve reserved one of these boats:

What about aggregates?

• Supported by extended relational algebra
  • \( \text{\textit{agg}}(\text{\textit{Exp}}) \rightarrow \text{\textit{Exp}} \text{\textit{Sailors}} \)
  • Also extended to "bag semantic": allow duplicates
    – Take into account cardinality
    – \( R \cup S \) has tuple \( t \) resp. \( m \) and \( n \) times
    – \( R \cap S \) has \( \min(m,n) \) times
    – \( R - S \) has \( \max(0,n-m) \) times
    – \( \sigma \text{\textit{Exp}} \), duplicate removal (\( \delta \) operators)

Example: Aggregates in RA

\[
\text{\textit{Sailors}}(\text{\textit{sid}}, \text{\textit{sname}}, \text{\textit{rating}}, \text{\textit{age}}) \\
\text{\textit{Boats}}(\text{\textit{bid}}, \text{\textit{bname}}, \text{\textit{color}}) \\
\text{\textit{Reserves}}(\text{\textit{sid}}, \text{\textit{bid}}, \text{\textit{day}})
\]

\[ \text{\textit{bnames}} \text{\textit{that have been reserved by at least 5 sailors}} \]

\[ \text{\textit{bnames}} \text{\textit{that have been reserved by at least 5 sailors}} \]

Output bnames that have been reserved by at least 5 sailors

Database Normalization
What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies — a new kind of integrity constraints (IC)
- Normal Forms
- How to obtain those normal forms

Why is redundancy bad?

1. Redundant storage:
   - Some information is stored repeatedly
   - The rating value 8 corresponds to hourly_wage 10, which is stored three times

2. Update anomalies
   - If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
   - Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency

3. Insertion anomalies:
   - It may not be possible to store certain information unless some other, unrelated info is stored as well
   - We cannot insert a tuple for an employee unless we know the hourly wage for the employee’s rating value

Example

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Suppose for a given rating, there is only one hourly_wage value. Why is redundancy bad?

1. Why is redundancy bad? 1/4

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

2. Why is redundancy bad? 2/4

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

3. Why is redundancy bad? 3/4

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

4. Why is redundancy bad? 4/4

<table>
<thead>
<tr>
<th>ssn ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
Nulls may or may not help

- Does not help redundant storage or update anomalies
- May help insertion and deletion anomalies
  - can insert a tuple with null value in the hourly_wage field
  - but cannot record hourly_wage for a rating unless there is such an employee (SSN cannot be null) – same for deletion

Summary: Redundancy

- Redundancy arises when the schema forces an association between attributes that is “not natural”
- We want schemas that do not permit redundancy
  - at least identify schemas that allow redundancy to make an informed decision (e.g. for performance reasons)
- Null value may or may not help

Solution?
  - Decomposition of schema!

Decomposition: Example-1

<table>
<thead>
<tr>
<th>uid ($)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly_wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Decomposition – Example-2

(ommitted)

- User id
- User name
- Twitter id
- Group id
- Joining date (to a group)
- Both uid and user_id are keys

Unnecessary decomposition

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>twitterid</th>
<th>wage (W)</th>
<th>worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>Bart</td>
<td>@BartSimpson</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>@MilhouseVan</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>142</td>
<td>Ralph</td>
<td>@ralphwiggum</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>353</td>
<td>Lisa</td>
<td>@lisa_simpson</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

Why?

Still correct: join returns the original relation

Unnecessary: no redundancy is removed; schema is more complicated (and uid is stored twice!)
Lossless join decomposition

- Decompose relation $R$ into relations $S$ and $T$
  - $\text{attrs}(R) = \text{attrs}(S) \cup \text{attrs}(T)$
  - $S = \pi_{\text{attrs}(S)}(R)$
  - $T = \pi_{\text{attrs}(T)}(R)$
- The decomposition is a lossless join decomposition if, given known constraints such as FD's, we can guarantee that $R = S \bowtie T$
- $R \subseteq S \bowtie T$ or $R \supseteq S \bowtie T$
- Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  - A lossy decomposition is one with $R \subset S \bowtie T$