Today

- Quiz 5
- Outer Joins
- Normalization (intro)

Assignments

- HW 6 due
- HW 7 out
Outer Joins

The basic join is called an “inner join”

- An inner join is the default ... the plain JOIN keyword
- We can also write an inner join this way:

\[
\text{SELECT *}
\text{FROM Customer c INNER JOIN Salesperson s ON c.salesperson = s.number;}
\]

- The answer includes all “matches”
- The answer **excludes**:
  - Customer rows that do not have a Salesperson
  - Salesperson rows that are not assigned any Customers

An “outer join” includes the “non-matches”

- A **LEFT OUTER JOIN** includes all matches plus all
  - Customers that do not have a Salesperson
- A **RIGHT OUTER JOIN** includes all matches plus all
  - Salespeople that are not assigned to any customers
- A **FULL OUTER JOIN** includes all of these

The “missing” attribute values in result are assigned NULL
INNER vs. OUTER JOIN

- **INNER JOIN** on c.salesperson = s.number gives:
  
  1, Smith, 55, 55, Miller  
  2, Jones, 65, 65, Adams

- **LEFT OUTER JOIN** on c.salesperson = s.number gives:
  
  1, Smith, 55, 55, Miller  
  2, Jones, 65, 65, Adams  
  3, Wei, NULL, NULL, NULL

- **RIGHT OUTER JOIN** on c.salesperson = s.number gives:
  
  1, Smith, 55, 55, Miller  
  2, Jones, 65, 65, Adams  
  NULL, NULL, NULL, 75, Martin

- **FULL OUTER JOIN** on c.salesperson = s.number gives:
  
  1, Smith, 55, 55, Miller  
  2, Jones, 65, 65, Adams  
  3, Wei, NULL, NULL, NULL  
  NULL, NULL, NULL, 75, Martin
In MySQL

- LEFT JOIN is same as LEFT OUTER JOIN
- RIGHT JOIN is same as RIGHT OUTER JOIN
- FULL JOIN is not supported – have to UNION
Normalization

“Normalization” involves replacing 1 table with 2 (or more) tables

- For example, we might split this table:

<table>
<thead>
<tr>
<th>eid</th>
<th>name</th>
<th>dept</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Alice</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

- Into these:

**Emp**

<table>
<thead>
<tr>
<th>eid</th>
<th>name</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Alice</td>
<td>12</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
</tr>
</tbody>
</table>

**Dept**

<table>
<thead>
<tr>
<th>did</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

Q: Which is better? Why?
Normalization issues

The **EmpDept schema combines two different concepts**

- Employee and department information into one table

What about this?

- If we **separate**, can save space
  - but some queries would run slower due to joins
- If we **combine**, we add redundancy
  - but some queries would run faster (no joins)
- So we have a tradeoff (space vs. efficiency)

Redundancy has a side effect though: “**anomalies**”