Today

- Quiz 6
- Outer joins
- Normalization (intro)

Assignments

- HW6 due
- HW7 out (start early!)
Example Tables

Customer(number, name, address, c.rating, c.amount, c.balance, salesperson)

Salesperson(number, name, address, office)

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
<th>...</th>
<th>salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>mary</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>102</td>
<td>john</td>
<td>...</td>
<td>8</td>
</tr>
<tr>
<td>103</td>
<td>dave</td>
<td>...</td>
<td>NULL</td>
</tr>
<tr>
<td>106</td>
<td>sam</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>107</td>
<td>oliver</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>109</td>
<td>susan</td>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>luis</td>
<td>...</td>
<td>8</td>
</tr>
</tbody>
</table>

- where Customer.salesperson is a FK to Salesperson.number
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:

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

- 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
Customer (simplified) | Salesperson (simplified)
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Smith</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
</tr>
</tbody>
</table>

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:

EmpDept

<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”