The Relational Data Model

Tables
Schemas
Conversion from E/R to Relations

A Relation is a Table

<table>
<thead>
<tr>
<th>Attributes (column headers)</th>
<th>Tuples (rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>manf</td>
</tr>
<tr>
<td>Winterbrew</td>
<td>Pete's</td>
</tr>
<tr>
<td>Bud Lite</td>
<td>Anheuser-Busch</td>
</tr>
</tbody>
</table>

Beers

Schemas

- **Relation schema** = relation name and attribute list.
- Optionally: types of attributes.
- Example: **Beers**(name, manf) or Beers(name: string, manf: string).
- **Database** = collection of relations.
- **Database schema** = set of all relation schemas in the database.
Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.

From E/R Diagrams to Relations

- Entity set -> relation.
  - Attributes -> attributes.
- Relationships -> relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

Entity Set -> Relation

Relation: Beers(name, manf)
Combining Relations

- OK to combine into one relation:
  - The relation for an entity-set $E$
  - The relations for many-one relationships of which $E$ is the “many.”
- Example: Drinkers(name, addr) and Favorite(drinker, beer) combine to make Drinker1(name, addr, favBeer).

Risk with Many-Many Relationships

- Combining Drinkers with Likes would be a mistake. It leads to redundancy, as:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>123 Maple</td>
<td>Bud</td>
</tr>
<tr>
<td>Sally</td>
<td>123 Maple</td>
<td>Miller</td>
</tr>
</tbody>
</table>

Redundancy
Handling Weak Entity Sets

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes.
- A supporting relationship is redundant and yields no relation (unless it has attributes).

Example

<table>
<thead>
<tr>
<th>Logins</th>
<th>Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>billTo</td>
<td></td>
</tr>
</tbody>
</table>

Hosts(hostName, location)
Logins(loginName, hostName, billTo)
At(location)

At becomes part of Logins
Must be the same

Subclasses: Three Approaches

- **Object-oriented**: One relation per subset of subclasses, with all relevant attributes.
- **Use nulls**: One relation; entities have NULL in attributes that don't belong to them.
- **E/R style**: One relation for each subclass:
  - Key attribute(s).
  - Attributes of that subclass.
Example

Object-Oriented

E/R Style

Good for queries like "find the color of ales made by Pete's."

Good for queries like "find all beers (including ales) made by Pete's."
Using Nulls

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>Anheuser-Busch</td>
<td>NULL</td>
</tr>
<tr>
<td>Summerbrew</td>
<td>Pete's</td>
<td>Dark</td>
</tr>
</tbody>
</table>

Saves space unless there are lots of attributes that are usually NULL.