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>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>manf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuples (rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winterbrew</td>
</tr>
<tr>
<td>Bud Lite</td>
</tr>
<tr>
<td>Pete's</td>
</tr>
<tr>
<td>Anheuser-Busch</td>
</tr>
<tr>
<td>Beers</td>
</tr>
</tbody>
</table>

Schemas

- Relation schema = relation name + attributes, in order (+ 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.
  - But SQL uses bags, while the relational model is a set-based model.

From E/R Diagrams to Relations

- Entity sets become relations with the same set of attributes.
- Relationships become 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

- It is OK to combine the relation for an entity-set $E$ with the relation $R$ for a many-one relationship from $E$ to another entity set.
- 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:

```
+-------+  +------+
| name  |  | beer  |
| Sally  |  | Bud   |
| Sally  |  | Miller|
```

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 (double-diamond) relationship is redundant and yields no relation.

Example

```
+-----+  +-----+
| time|  | name |
| loginName |  | hostName |
| Logins |  | Hosts |
```

Entity Sets With Subclasses

- Three approaches:
  1. **Object-oriented**: each entity belongs to exactly one class; create a relation for each class, with all its attributes.
  2. **Use nulls**: create one relation; entities have null in attributes that don’t belong to them.
  3. **E/R style**: create one relation for each subclass, with only the key attribute(s) and attributes attached to that E.S.; entity represented in all relations to whose subclass E.S. it belongs.
Example

- Beers
- Ale
- Summerbrew Pete’s dark

Object-Oriented

- Name: Bud Anheuser-Busch
- Name: Summerbrew Pete’s dark
- Name: Ales

E/R Style

<table>
<thead>
<tr>
<th>Name</th>
<th>Manufacturer</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>

Using Nulls

<table>
<thead>
<tr>
<th>Name</th>
<th>Manufacturer</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>

Comparisons

- O-O approach good for queries like "find the color of ales made by Pete’s."
  - Just look in Ales relation.
- E/R approach good for queries like "find all beers (including ales) made by Pete’s."
  - Just look in Beers relation.
- Using nulls saves space unless there are lots of attributes that are usually null.