The Relational Data Model

Tables
Schemas
Conversion from E/R to Relations

A Relation is a Table

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winterbrew</td>
<td>Pete's</td>
</tr>
<tr>
<td>Bud Lite</td>
<td>Anheuser-Busch</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:

```
<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>
```

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

```
<table>
<thead>
<tr>
<th>name</th>
<th>loginName</th>
<th>hostName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>loginName</td>
<td>hostName</td>
</tr>
</tbody>
</table>

At becomes part of Logins
Must be the same
```

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

Object-Oriented

E/R Style

Using Nulls

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.