The Big Picture

- Stages of building a database application:
  - Real-world domain.
    - understand client needs.
  - Design data model:
    - using entity-relationship (E/R) model
  - Database data model:
    - using relational model
  - Create schema in DBMS, load data.
  - Open for business!

Outline

- Relational model.
- From E/R diagrams to relations.

Relational Model

- Table = relation.
- Column headers = attributes.
- Row = tuple.
- **Beers example:**
<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honkers Ale</td>
<td>Goose Island</td>
</tr>
<tr>
<td>BudLite</td>
<td>A.B.</td>
</tr>
</tbody>
</table>

Why Relations?

- Very simple model.
- *Often* a good match for the way we think about our data.
- Abstract model that underlies SQL, the most important language in DBMS’s today.
  - But SQL uses *bags* while the abstract relational model is set-oriented.
Relational Design

• Simplest approach (not always best):
  – convert each E.S. to a relation
  – convert each relationship to a relation.

Entity Set → Relation

• E.S. attributes become relational attributes.
  
  \[
  \text{name} \quad \text{Beers} \quad \text{manf}
  \]

  • Becomes:
  \[
  \text{Beers(name, manf)}
  \]

Keys in Relations

• An attribute or set of attributes \( K \) is a key for a relation \( R \) if we expect that in no instance of \( R \) will two different tuples agree on all the attributes of \( K \).
• Indicate a key by underlining the key attributes.
• Example: If \( \text{name} \) is a key for \( \text{Beers} \):
  \[
  \text{Beers(name, manf)}
  \]

E/R Relationships → Relations

• Relation has attribute(s) for key attributes of each E.S. that participates in the relationship.
• Add any attributes that belong to the relationship itself.
• Renaming attributes OK.
  – Essential if multiple roles for an E.S.

Combining Relations

• Common case: Relation for an E.S. \( E \) plus the relation for some many-one relationship from \( E \) to another E.S
• Example:
  – Combine \( \text{Drinkers(name, addr)} \) with \( \text{Favorite(drinker, beer)} \).
  – Resulting in: \( \text{Drinkers(name, addr, favBeer)} \).
• Danger in pushing this idea too far: redundancy.
• Example:
  – Combining \( \text{Drinker} \) with \( \text{Likes} \) causes the drinker's address to be repeated, viz.:
    
    | name | addr  | beer   |
    |------|-------|--------|
    | Mike | 111 E Ohio | Guinness |
    | Mike | 111 E Ohio | Newcastle |
  
• The difference: \( \text{Favorite} \) is many-one; \( \text{Likes} \) is many-many.
Weak Entity Sets, Relationships → Relations

- Relation for a weak E.S. must include its full key (i.e., attributes of related entity sets) as well as its own attributes.
- A supporting (double-diamond) relationship yields a relation that is actually redundant and should be deleted from the database schema.

Example

```
Hosts(name)
Users(name, hostName)
At(userName, hostName, hostName2)
```

- In At, hostName and hostName2 must be the same host, so delete one of them.
- Then, Users and At become the same relation; delete one of them.
- In this case, Hosts’ schema is a subset of Users’ schema. Delete Hosts?

Subclasses → Relations

- Three approaches:
  - Object-oriented
  - E/R style
  - Using nulls

Object-oriented Style

- Each entity is in one class.
- Create a relation for each class, with all the attributes for that class.
  - Don’t forget inherited attributes.

```
<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BudLite</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td>Beers</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td>Honkers Ale</td>
<td>Goose Island</td>
<td></td>
</tr>
</tbody>
</table>
```

E/R Style

- An entity is in a network of classes related by isa.
- Create one relation for each E.S.
  - Relation has only the attributes attached to that E.S. + key.

```
<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BudLite</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td>Beers</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td>Honkers Ale</td>
<td>Goose Island</td>
<td></td>
</tr>
<tr>
<td>Ales</td>
<td>Honkers Ale</td>
<td>dark</td>
</tr>
</tbody>
</table>
```
Using NULLs

- Create one relation for the root class or root E.S., with all attributes found anywhere in its network of subclasses.
  - Put NULL in attributes not relevant to a given entity.

<table>
<thead>
<tr>
<th>Beers</th>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BudLite</td>
<td>A.B.</td>
<td>NULL</td>
</tr>
<tr>
<td>Honkers Ale</td>
<td>Goose Island</td>
<td>dark</td>
<td></td>
</tr>
</tbody>
</table>