Announcements

- Assignment 1:
  - Due next Tuesday, Oct 11.
  - Submit it in class!
- Talk on online dating by Ali Hortaçsu
  - Economics department ("Freakonomics")

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.
- Functional dependencies.

Relational Model

- Table = relation.
- Column headers = attributes.
- Row = tuple.
- Beers example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>manf</td>
</tr>
<tr>
<td>Honkers Ale</td>
<td>Goose Island</td>
</tr>
<tr>
<td>BudLite</td>
<td>A.B.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Relational Model

- Relation schema:
  - name (attributes)
  - other structure info., e.g., keys, other constraints.
- Example: Beers(name, manf).
- Order of attributes is arbitrary.
  - In practice we need to assume the order given in the relation schema.
- Relation instance is current set of rows for a relation schema.
- Database schema is collection of relation schemas.
- "A Relational Model of Data for Large Shared Data Banks" by E. F. Codd in Communications of ACM, Vol 13. No. 6, June 1970
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.

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

  \[
  \text{name} \rightarrow \text{Beers} \rightarrow \text{manf}
  \]

  \[
  \text{Becomes:}
  \]

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

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 name is a key for 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.

Example

- Diagram showing relationships between entities such as Beers, Drinkers, Buddies, and Married.

  - \text{Beers} \leftarrow \text{name}, \text{manf}
  - \text{Drinkers} \leftarrow \text{name}, \text{manf}
  - \text{Buddies} \leftarrow \text{husband}, \text{wife}
  - \text{Married} \leftarrow \text{husband}, \text{wife}
  - Relationships include likes, favorites, and marriage statuses.
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 $\text{Drinkers}(\text{name}, \text{addr})$ with $\text{Favorite}(\text{drinker}, \text{beer})$.
  - Resulting in: $\text{Drinkers1}(\text{name}, \text{addr}, \text{favBeer})$.
- Danger in pushing this idea too far: redundancy.
- Example:
  - Combining $\text{Drinker}$ with $\text{Likes}$ causes the drinker’s address to be repeated, viz.:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike</td>
<td>111 E Ohio</td>
<td>Guinness</td>
</tr>
<tr>
<td>Mike</td>
<td>111 E Ohio</td>
<td>Newcastle</td>
</tr>
</tbody>
</table>
- 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

```
$\text{Hosts(name)}$
$\text{Users(name, hostName)}$
$\text{At(userName, hostName, hostName2)}$
- In $\text{At}$, $\text{hostName}$ and $\text{hostName2}$ must be the same host, so delete one of them.
- Then, $\text{Users}$ and $\text{At}$ become the same relation; delete one of them.
- In this case, $\text{Hosts}$’ schema is a subset of $\text{Users}$’ schema. Delete $\text{Hosts}$?
```

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.
E/R Style

- An entity is in a network of classes related by \textit{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>Honkers Ale</td>
<td>Goose Island</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 \textit{NULL} in attributes not relevant to a given entity.

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

Functional Dependencies

- $X \rightarrow A$
  - assertion about a relation \( R \) that whenever two tuples agree on all the attributes of \( X \), then they must also agree on attribute \( A \).
  - Important as a constraint on the data that may appear within a relation.
  - Schema-level control of data.
  - Mathematical tool for explaining the process of "normalization" – vital for redesigning database schemas when original design has certain flaws.

Example

\texttt{Drinkers(name, addr, beersLiked, manf, favoriteBeer)}

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favoriteBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike</td>
<td>111 E Ohio</td>
<td>Bud</td>
<td>A.B.</td>
<td>Blonde Ale</td>
</tr>
<tr>
<td>Mike</td>
<td>111 E Ohio</td>
<td>Blonde Ale</td>
<td>G.I.</td>
<td>Blonde Ale</td>
</tr>
<tr>
<td>Anna</td>
<td>123 W Grand</td>
<td>BudLite</td>
<td>A.B.</td>
<td>BudLite</td>
</tr>
</tbody>
</table>

- Reasonable FD's to assert:
  1. ...
  2. ...
  3. ...
- Note: FD's can give more detail than just assertion of a key.