Relational Model

- Table = relation.
- Column headers = attributes.
- Row = tuple

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinterBrew</td>
<td>Pete’s</td>
</tr>
<tr>
<td>BudLite</td>
<td>A.B.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Beers

- Relation schema = name(attributes) + other structure info., e.g., keys, other constraints. Example: Beers(name, manf).
  - Order of attributes is arbitrary, but 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 = collection of relation schemas.
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 and each relationship to a relation.

Entity Set $\rightarrow$ Relation

E.S. attributes become relational attributes.

Becomes:

\[
\text{Beers(name, manf)}
\]
E/R Relationships → Relations

Relation has attribute 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.
Likes(drinker, beer)
Favorite(drinker, beer)
Buddies(name1, name2)
Married(husband, wife)
Combining Relations

Sometimes it makes sense to combine 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 Drinker$(\text{name, addr})$ with Favorite$(\text{drinker, beer})$ to get Drinker1$(\text{name, addr, favBeer})$.

- Danger in pushing this idea too far: redundancy.
- e.g., combining Drinker with 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>Sally</td>
<td>123 Maple</td>
<td>Bud</td>
</tr>
<tr>
<td>Sally</td>
<td>123 Maple</td>
<td>Miller</td>
</tr>
</tbody>
</table>

- Notice the difference: Favorite is many-one; 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(hostName)
Logins(loginName, hostName)
At(loginName, hostName, hostName2)

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

Three approaches:

1. Object-oriented: each entity is in one class. Create a relation for each class, with all the attributes for that class.
   ✦ Don’t forget inherited attributes.

2. E/R style: an entity is in a network of classes related by isa. Create one relation for each E.S.
   ✦ An entity is represented in the relation for each subclass to which it belongs.
   ✦ Relation has only the attributes attached to that E.S. + key.

3. Use 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.
Example

```
Example

name

Beers

isa

Ales

manf

color
```
### OO-Style

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
</tr>
</tbody>
</table>

**Beers**

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>SummerBrew</td>
<td>Pete’s</td>
<td>dark</td>
</tr>
</tbody>
</table>

**Ales**

### E/R Style

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>SummerBrew</td>
<td>Pete’s</td>
</tr>
</tbody>
</table>

**Beers**

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>SummerBrew</td>
<td>dark</td>
</tr>
</tbody>
</table>

**Ales**
Using Nulls

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
<td>NULL</td>
</tr>
<tr>
<td>SummerBrew</td>
<td>Pete’s</td>
<td>dark</td>
</tr>
</tbody>
</table>

Beers
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 \( A \).

Example

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>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete’s</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

- Reasonable FD’s to assert:

1. \( \text{name} \rightarrow \text{addr} \)
2. \( \text{name} \rightarrow \text{favoriteBeer} \)
3. \( \text{beersLiked} \rightarrow \text{manf} \)
• Shorthand: combine FD’s with common left side by concatenating their right sides.

• Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:

  beer bar → price
Keys of Relations

$K$ is a key for relation $R$ if:

1. $K \rightarrow$ all attributes of $R$.
2. For no proper subset of $K$ is (1) true.

- If $K$ at least satisfies (1), then $K$ is a superkey.

Conventions

- Pick one key; underline key attributes in the relation schema.
- $X$, etc., represent sets of attributes; $A$ etc., represent single attributes.
- No set formers in FD’s, e.g., $ABC$ instead of $\{A, B, C\}$. 
Example

Drinkers(name, addr, beersLiked, manf, favoriteBeer)

- \{name, beersLiked\} FD’s all attributes, as seen.
  - Shows \{name, beersLiked\} is a superkey.
- name → beersLiked is false, so name not a superkey.
- beersLiked → name also false, so beersLiked not a superkey.
- Thus, \{name, beersLiked\} is a key.
- No other keys in this example.
  - Neither name nor beersLiked is on the right of any observed FD, so they must be part of any superkey.
- Important point: “key” in a relation refers to tuples, not the entities they represent. If an entity is represented by several tuples, then entity-key will not be the same as relation-key.
Who Determines Keys/FD’s?

- We could assert a key $K$.
  - Then the only FD’s asserted are that $K \rightarrow A$ for every attribute $A$.
  - No surprise: $K$ is then the only key for those FD’s, according to the formal definition of “key.”

- Or, we could assert some FD’s and deduce one or more keys by the formal definition.
  - E/R diagram implies FD’s by key declarations and many-one relationship declarations.

- Rule of thumb: FD’s either come from keyness, many-1 relationship, or from physics.
  - E.g., “no two courses can meet in the same room at the same time” yields $\text{room time} \rightarrow \text{course.}$
Inferring FD’s

And this is important because . . .

- When we talk about improving relational designs, we often need to ask “does this FD hold in this relation?”

Given FD’s $X_1 \rightarrow A_1, X_2 \rightarrow A_2 \cdots X_n \rightarrow A_n$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- Start by assuming two tuples agree in $Y$. Use given FD’s to infer other attributes on which they must agree. If $B$ is among them, then yes, else no.
**Algorithm**

Define $Y^+ = \text{closure of } Y = \text{set of attributes functionally determined by } Y$:

- **Basis:** $Y^+ := Y$.

- **Induction:** If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add $A$ to $Y^+$.

- **End when $Y^+$ cannot be changed.**
Example

$A \rightarrow B, \ BC \rightarrow D$.

- $A^+ = AB$.
- $C^+ = C$.
- $(AC)^+ = ABCD$. 
Finding All Implied FD’s

Motivation: Suppose we have a relation $ABCD$ with some FD’s $F$. If we decide to decompose $ABCD$ into $ABC$ and $AD$, what are the FD’s for $ABC$, $AD$?

- Example: $F = AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. It looks like just $AB \rightarrow C$ holds in $ABC$, but in fact $C \rightarrow A$ follows from $F$ and applies to relation $ABC$.

- Problem is exponential in worst case.
Algorithm

• For each set of attributes $X$ compute $X^+$.
  ♦ But skip $X = \emptyset$, $X =$ all attributes.
  ♦ Add $X \rightarrow A$ for each $A$ in $X^+ - X$.

• Drop $XY \rightarrow A$ if $X \rightarrow A$ holds.

• Finally, project the FD’s by selecting only those FD’s that involve only the attributes of the projection.
  ♦ Notice that after we project the discovered FD’s onto some relation, the eliminated FD’s can be inferred in the projected relation.
Example

In $ABC$ with FD’s $A \rightarrow B$, $B \rightarrow C$, project onto $AC$.

1. $A^+ = ABC$; yields $A \rightarrow B$, $A \rightarrow C$.

2. $B^+ = BC$; yields $B \rightarrow C$.

3. $AB^+ = ABC$; yields $AB \rightarrow C$; drop in favor of $A \rightarrow C$.

4. $AC^+ = ABC$ yields $AC \rightarrow B$; drop in favor of $A \rightarrow B$.

5. $C^+ = C$ and $BC^+ = BC$; adds nothing.

- Resulting FD’s: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$.
- Projection onto $AC$: $A \rightarrow C$. 