Schedule

• Today Oct. 2 (T) Relational Model.
  ◆ Read Sections 3.1-3.4. Assignment 1 due. Personal Letter of Introduction and Background due (2% of course grade).

• Oct. 4 (TH) Functional Dependencies and Normalization.
  ◆ Read Sections 3.5-3.6. Project Part 1 due.

• Oct. 9 (T) Multivalued Dependencies, Relational Algebra
  ◆ Read Sections 3.7, 5.1-5.2. Assignment 2 due.
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>
</tbody>
</table>

- 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.
Relational Data Model

Relation as table
- Rows = tuples
- Columns = components
- Names of columns = attributes
- Set of attribute names = schema
  REL (A1,A2,...,An)

Set theoretic
- Domain — set of values
  like a data type
- Cartesian product (or product)
  $D_1 \times D_2 \times ... \times D_n$
- k-tuples $(V_1,V_2,...,V_n)$
  s.t., $V_1 \in D_1, V_2 \in D_2,...,V_n \in D_n$
- Relation-subset of cartesian product
  of one or more domains
  FINITE only; empty set allowed
- Tuples = members of a relation inst.
- Arity = number of domains
- Components = values in a tuple
- Domains — corresp. with attributes
- Cardinality = number of tuples
Relation: Example

Cardinality of domain

<table>
<thead>
<tr>
<th>Domain of Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>N1</td>
</tr>
<tr>
<td>N1</td>
</tr>
<tr>
<td>N1</td>
</tr>
</tbody>
</table>

Arity 3
Cardinality <=5x3x7

Domain: Name, address, tel#

<table>
<thead>
<tr>
<th>Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>N1</td>
</tr>
<tr>
<td>N2</td>
</tr>
<tr>
<td>N3</td>
</tr>
<tr>
<td>N4</td>
</tr>
<tr>
<td>N5</td>
</tr>
<tr>
<td>T6</td>
</tr>
<tr>
<td>T7</td>
</tr>
</tbody>
</table>

Attribute

Component

Tuple μ

N1 A3 T1
N2 A1 T1
# Relation Instance

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>123 Main St</td>
<td>555-1234</td>
</tr>
<tr>
<td>Bob</td>
<td>128 Main St</td>
<td>555-1235</td>
</tr>
<tr>
<td>Pat</td>
<td>123 Main St</td>
<td>555-1235</td>
</tr>
<tr>
<td>Harry</td>
<td>456 Main St</td>
<td>555-2221</td>
</tr>
<tr>
<td>Sally</td>
<td>456 Main St</td>
<td>555-2221</td>
</tr>
<tr>
<td>Sally</td>
<td>456 Main St</td>
<td>555-2223</td>
</tr>
<tr>
<td>Pat</td>
<td>12 State St</td>
<td>555-1235</td>
</tr>
</tbody>
</table>
About Relational Model

Order of tuples not important
Order of attributes not important (in theory)

Collection of relation schemas  (intension)
   Relational database schema
Corresponding relation instances     (extension)
   Relational database

intension vs. extension
schema vs. data

metadata
includes schema
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`:
  \[
  \text{Beers}(\underline{\text{name}}, \text{manf})
  \]
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 → Relation**

E.S. attributes become relational attributes.

Becomes:

```plaintext
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.
• For one-one relation Married, we can choose either husband or wife as key.
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 $\text{Drinker(name, addr)}$ with $\text{Favorite(drinker, beer)}$ to get $\text{Drinker1(name, addr, favBeer)}$.

- Danger in pushing this idea too far: redundancy.
- e.g., 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>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: $\text{Favorite}$ 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 (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
### OO-Style

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beers</td>
</tr>
<tr>
<td>SummerBrew</td>
<td>Pete's</td>
<td>dark</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ales</td>
</tr>
</tbody>
</table>

### E/R Style

<table>
<thead>
<tr>
<th>name</th>
<th>manf</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SummerBrew</td>
<td>Pete's</td>
<td>dark</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ales</td>
</tr>
</tbody>
</table>

### 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></td>
<td>Pete's</td>
<td>dark</td>
</tr>
<tr>
<td>SummerBrew</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beers</td>
</tr>
</tbody>
</table>

Functional Dependencies

$X \rightarrow A = \text{assertion about a relation } R \text{ that whenever two tuples agree on all the attributes of } X, \text{ then they must also agree on attribute } A.$

- 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

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. name → addr
  2. name → favoriteBeer
  3. beersLiked → manf

- Note: These happen to imply the underlined key, but the FD’s give more detail than the mere assertion of a key.
• Key (in general) functionally determines all attributes. In our example:
Name beersLiked → addr favoriteBeer beerManf
• Shorthand: combine FD's with common left side by concatenating their right sides.
• When FD's are not of the form
  Key → other attribute(s),
then there is typically an attempt to “cram” too much into one relation.
• Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:
  beer bar → price
Formal Notion of Key

• $K$ is a key for relation $R$ if:
  1. $K \rightarrow$ all attributes of $R$. (Uniqueness)
  2. For no proper subset of $K$ is (1) true. (Minimality)
• If $K$ at least satisfies (1), then $K$ is a superkey.

FD Conventions

• $X$, 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 \rightarrow beersLiked\ is false, so name not a superkey.
• beersLiked \rightarrow 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.
Example 2

- Keys are \{Lastname, Firstname\} and \{StudentID\}

Note: There are alternate keys
Who Determines Keys/FD’s?

• We could define a relation schema by simply giving a single 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 $\text{room \ time} \rightarrow \text{course}$.