Subtypes & Supertypes

The Data Modeler's most Valuable Construct

©Gordon C. Everest
Professor Emeritus, Carlson School of Management
University of Minnesota
Community Faculty, MetroState

7. Sub/Super Types

• “Abstractions” & Collections
• Attribute and Relationship Generalization
• Subtypes and Supertypes (Entity Generalization)
  – Underlying assumptions; conditions
  – Generalization vs. Specialization
  – Graphical representations
• Constraints
• Subtype Definition - distinguishing attribute
• Sub/SuperType Hierarchy/Lattice
  – The Universal Relation
• Inheritance (single; multiple) & Reuse
• Mapping to Tables

Gordon C. Everest
Carlson School of Management
University of Minnesota
"Abstractions" & Collections

Focusing on selected properties of objects

“Abstractions”: (used in a different sense here)

• CLASSIFICATION (“Member-of”)
  – Forming Types - entity sets/populations, domains

• AGGREGATION (“Part-of”)
  – Building an entity record with descriptors (clustering attributes)
  – COMPOSITION (stronger "Part of" - no independent existence)

• GENERALIZATION/SPECIALIZATION (“Is-a”)
  – Forming subtypes/supertypes, population subsets

Collections: (assumes homogeneous members)

• SET – no duplicates and no order
• BAG – counting duplicates
• SEQUENCE – order matters
• …

Omitting vs. Hiding Detail

OMIT unimportant detail

REALITY

MODEL = an Abstract Re.presentation of Reality

HIDE detail / parts

Abstraction: Presentation

“There is no abstract art. You must always start with something. Afterward you can remove all traces of reality.” -- Pablo Picasso
Generalization

- Recognizing commonalities (+valued, -cost)
- Moving "up" to a higher, more inclusive, more generic, more "abstract" view

**TYPES:**

- **Attribute**
  - constrained by Entity Generalization
  - often a prelude to Entity Generalization

- **Entity**
  - represented using subtypes/supertypes
  - implications for placement and naming of attributes and relationships

- **Relationship**

**Attribute Generalization Examples**

- For a Tuxedo Rental shop, store Customer attributes:
  - Waist size
  - Leg length
  - Neck size
  - Arm length
  - Shoulder width

=> Later add Shoe size.

*What does that do to your database schema?*
*How might you solve the problem?*
*How does referential integrity become important here?*
*What is the down side of this schema redesign?*

Similarly for Phone numbers:

**Problems:**

- Handling international numbers
- Handling other contact information, e.g. email
Attribute Generalization Example

Given the following three entity type populations:

What do you observe?

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>SUPPLIER</th>
<th>ASSOCIATE (Employee)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td>Company name</td>
<td>First name</td>
</tr>
<tr>
<td>Last name</td>
<td>Contact first name</td>
<td>Last name</td>
</tr>
<tr>
<td>Address line</td>
<td>Address line</td>
<td>Address line</td>
</tr>
<tr>
<td>City</td>
<td>City</td>
<td>City</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>Zip code</td>
<td>Zip code</td>
<td>Zip code</td>
</tr>
<tr>
<td>Phone number</td>
<td>Phone number</td>
<td>Phone number</td>
</tr>
<tr>
<td>Fax number</td>
<td>Fax number</td>
<td>Fax number</td>
</tr>
<tr>
<td>Tax id</td>
<td>Cell Phone Number</td>
<td>Social Security #</td>
</tr>
<tr>
<td>First order date</td>
<td>Credit Rating</td>
<td>Email address</td>
</tr>
<tr>
<td>DUNS #</td>
<td>First PO Date</td>
<td>Hire date</td>
</tr>
</tbody>
</table>

An ASSOCIATE can be assigned to several CUSTOMERs, and manage the relationship with many SUPPLIERs. A CUSTOMER or SUPPLIER can contact multiple ASSOCIATES.

Attribute Generalization - Financial

Suppose you saw a table defined like this:

How many rows would it have?

What would YOU want to do?

FINANCIAL DATA:

<table>
<thead>
<tr>
<th>Dept</th>
<th>Year</th>
<th>Qtr</th>
<th>Bud/Act</th>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
</table>

Classic Fact Table for a Dimensional Model!

How many rows would this table have?

FINANCIAL DATA:

<table>
<thead>
<tr>
<th>Dept</th>
<th>Year</th>
<th>Qtr</th>
<th>Bud/Act</th>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Extreme Attribute Generalization**

What is lost here?

What is hard?

What else might be of interest about an attribute?

What is easy?

The Power of Generalization Thinking!

If find you are mixing value domains, you may have generalized too much.

**Relationship Generalization**

Reducing multiple relationships:

owner

beneficiary

insured

INSURANCE POLICY

to one:

role

relationship name?

PERSON

INSURANCE POLICY

NOTE: We have already generalized the individual roles into a Person entity.

Where would you store the ‘role’ attribute?

How many foreign keys are stored? Where?
Fathers, Mothers, their Marriages, and Children:

*What do these data models assume?*

Together with subtypes and supertypes:

... leaving open the question of where the Tables will be built.

---

**Generalization: Pros & Cons**

- Fewer entities and relationships (simpler?)
- Greater flexibility to incorporate extensions
- Greater long-term stability of the model
- Looking for commonalities → greater understanding
- Handle special treatment of subsets
  - Hides business vocabulary
    - Business terms not in schema names but in attribute values
  - Harder to express and enforce business rules or constraints. Most become conditional on the specialized subtype.
  - Problem defining the identifiers (Ref. modes)
  - Queries are more difficult to formulate and less efficient to execute (more JOINs).
    - E.g., "FIND Customers WHERE Waist = 42" - no longer works
Discerning Inter-Entity Relationships

Construct a high-level, conceptual data model.

Problems?
Observations?

A Fundamental Assumption in a Data Model Diagram

- The main construct is an Entity.
- Each labeled box/circle represents an Entity Type
  - a Defined Structure (a schema template)
  - a Population of Instances
- Grouping Instances into Types is essentially Arbitrary.
  - The world isn't naturally that way; the designer imposes a view
- All Entity Type Populations are strictly Disjoint (mutually exclusive; or non-overlapping).
  - At least that is the system's assumption, thus each file/table has its own set of records/tuples.

Is this always true?

What about: EMPLOYEE SHAREHOLDER
Using Subtypes and Supertypes

- Subtypes and Supertypes allow us to formally represent overlapping populations
  - Every member of a subtype population is-a member of its supertype population(s).

so we can model:
- different roles played by members of a common population, e.g.:
  - role determines the attributes
- different states of an entity (over time), e.g.:

Subtype-Supertype "Relationship"

- Tempting to call it a "Relationship"

- pop(subtype) ⊆ pop(supertype)
- AND the related members in the two sets are the same instance
  - that is what makes it different from relationships in a traditional ER/Relational data model, where the entity type populations are (assumed) disjoint!
Generalization / Specialization

Forming Entity Types

- Fundamentally an “arbitrary” choice made by the DB Designer
- Recognizing when to use Subtypes and Supertypes
- Think about the entity populations you are modeling

Two basic and distinct situations:

• **Generalization**: (bottom-up - from several to a common supertype)
  - When you observe commonalities (e.g., common attributes*) across multiple entity populations.
  - the members may actually be from the same population, the same type of ‘thing’, so define a common supertype.

• **Specialization**: (top-down - from one to subtypes)
  - When there is something special about a subset of a population
    - They have some unique attributes*
    - You want to treat them differently
      - e.g., Apply a constraint, or have some attributes mandatory

*NOTE: speaking of attributes in ORM, means roles in relationships with other objects.

Subtypes and Supertypes

TWO CONDITIONS MUST ALWAYS BE TRUE:

• each subtype population must be a **subset** (potentially) of each of its supertype populations
  i.e., each instance of the subtype is in every supertype population

• each subtype inherits all the roles of its supertypes and must have **additional roles/relationships**

If either condition is NOT true, no reason to call out the subtype in a separate definition.
Diagramming Subtypes and Supertypes

Two Basic Representations:

1. **NESTED** (Euler Diagram)
   - Intuitive - visually shows inclusion
   - Clean and Compact
   - Only good for simple cases
   - Generally assumed disjoint
   - Not good for complex cases - difficult to represent both exclusive and overlapping subtypes (like a Venn Diagram):

```
A
B C D
E
```

2. **SEPARATED**
   - More common
   - Easier to show constraints and multiple supertypes for more complex cases.
   - Not visually intuitive
   - Confusion with "relationship"
   - Adds more "clutter"
Diagramming Exercise

- Convert the following Nested diagram into a Separated S/Stype diagram:

```
A
B C D
E
```

How to model 'E'?  
Any constraints required?

Subtype / Supertype Constraints

IN GENERAL, WITHOUT CONSTRAINTS, ASSUME:
- overlapping subtype populations  
  - else Disjoint, so apply Exclusion constraint:
- non-exhaustive (Partial) on the supertype,  
  i.e., a supertype instance need not be in any subtype  
  - else Mandatory/Totality/Dependency constraint

=> Declare constraints on the more restrictive cases
- Some systems allow only Disjoint and Total  
  - it is possible to model Overlapping, even if the system only allows Disjoint subtypes.
- Some systems make Disjoint and Total the defaults
Diagramming S/Stype Constraints

CONSTRAINTS (3 cases):
1. Exclusive or disjoint subtypes – X
2. Exhaustive or Total on the supertype – T
3. Exhaustive or Total on the subtype – T_b

S

M W C

T

S_r

O M B F

T_b

e.g., MAN, WOMAN, CHILD
e.g., OVIPAROUS, MAMMAL, BIRD, FISH

S/Stype Constraints - Other Notations

<table>
<thead>
<tr>
<th></th>
<th>Exclusive or disjoint</th>
<th>Overlapping</th>
<th>Exhaustive or Total</th>
<th>Partial</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIAM</td>
<td>X</td>
<td></td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>ORM*</td>
<td>X</td>
<td>(default)</td>
<td>*</td>
<td>(default)</td>
</tr>
<tr>
<td>EER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDEF1X*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*combined

NOTE: No modeling schemes or systems recognize totality on the subtype.
"Well-Defined" Subtypes

- based on an attribute of the supertype
  - called the "distinguishing" attribute
- characteristics of the relationship determine the constraints on the subtypes
  - mandatory attribute => exhaustive/totality constraint
  - single-valued attribute => exclusive subtypes constraint

What if an optional attribute?
What if a multi-valued attribute (M:N relationship)?

Subtype Definition

Attribute-Defined Subtype (Intentional Set)
- a rule for including a Supertype instance in the Subtype
- Defined in terms of the values of a supertype attribute
- in general, a Boolean expression on attribute(s) of the supertype
- can be considered a constraint rule on subset membership
- there are many possible subgroupings (specializations) of an entity type based on the values of its attributes, so find those that matter.
- it is not always possible to define the rules for membership in a subtype, hence:

User-Defined Subtype (Extensional Set)
- inclusion determined by “existence” in the set; membership is manual, the system cannot automate or validate membership.
- Some systems require subtype definitions such as VisioEA (but... as free-form text!)
- Can always come up with an artificial distinguishing attribute
What commonalities do you observe?

Who ‘owns’ the Subscription? Faculty or the Dept/School that pays?
Gift Subscription? Who gets the renewal notice?
“UNIVERSAL RELATION”
- A single type population
  => ONE TYPE
  (the "root" class in O-O)

Generalization

Specialization

The population of all instances.
- Each instance is its own Type
  => NO TYPES (or N types!)

AT THE EXTREMES:
• a single supertype at the top is called the
  UNIVERSAL RELATION. If you built a single
  table for all the data in your organization:
  – what would be the entity?
  – what would be the identifier?
  – what would be the attributes?
  – would all the attributes be relevant for each row?
• at the bottom would be individual instances, each instance being its own type!
  – But sharing many attributes with other entities

The real art of database design
is picking the appropriate entity types
within the levels of the hierarchy.
  – Allows the designer to defer choosing what tables
to build
Single vs. Multiple Inheritance

- SINGLE - every subtype has only one supertype
  => a strict Hierarchy of types
- MULTIPLE supertypes for a (Shared) Subtype
  - If multiple supertypes, they must converge on one population higher up = the root type
  => a lattice.

so what is wrong/incomplete with:

- no lattice if no overlapping subtypes
- it is possible to transform multiple inheritance to a generalization hierarchy by defining all possible combinations of subtypes
- each mini hierarchy or lattice has a root, all root objects are disjoint.

Extreme Entity Generalization

“What is the ‘THING’?”

“The level of generalization is critical.” - Graeme Simsion
Inheritance and Reuse

• Separate but related notions - often confused (C. Date got it right)

DESIGN NOTION based on characteristics of populations:
• Multiple populations with some different characteristics sharing some common characteristics, … so define a supertype (generalization).
• Need for special treatment of a subset of a population … so define a subtype (specialization)
  – Subtype inherits common characteristics from its supertypes plus has some additional characteristics of interest

CONSTRUCTION - implementation efficiency => REUSE
• Inheritance of definitions of data and procedures
  – for efficiency of implementation
  – overriding and blocking
  – static (copy at creation time only),
    vs. dynamic (maintain linkage to automatically inherit changes)
  – multiple inheritance => conflict, priority order

Comparing Modeling Schemes

<table>
<thead>
<tr>
<th>Class Hierarchy</th>
<th>ER/Rel (Chen)</th>
<th>EER (Teorey++)</th>
<th>ORM (Halpin)</th>
<th>UML (OMG)</th>
<th>ODL (ODMG)</th>
<th>SQL 1999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disjoint*</td>
<td>Y</td>
<td>Y</td>
<td>default</td>
<td>only</td>
<td>only</td>
<td></td>
</tr>
<tr>
<td>Overlapping</td>
<td>default</td>
<td>default</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Total covering*</td>
<td>Y</td>
<td>Y</td>
<td>user-defined on abstr.class</td>
<td>partly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Attribute-defined discriminator</td>
<td>Y</td>
<td>‘must’ limited but... (pseudo attrib)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared Subclasses (multiple inheritance)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

*the more restrictive case, calling for a constraint declaration.
Mapping to (Relational) Tables

THREE BASIC CHOICES:

- **Supertype only:**
  (Absorption - ‘flatten’ up)
  \[
  K_P \{ P_i \} \ldots \{ A_i \} \ldots \{ B_i \} \ldots
  \]

- **Subtypes only:** (not possible in VisioEA)
  (Separation - ‘flatten’ down)
  \[
  K_A \{ P_i \} \ldots \{ A_i \} \ldots
  \]
  \[
  K_B \{ P_i \} \ldots \{ B_i \} \ldots
  \]

- **Both:**
  (Partition)
  \[
  K_P \{ P_i \} \ldots
  \]
  \[
  K_A \{ A_i \} \ldots
  \]
  \[
  K_B \{ B_i \} \ldots
  \]

CONSIDER:
- **D =** Distinguishing Attribute on P
  (optional)
  (single-valued?)
- **Exclusive:** on A & B
  - A & B overlapping
- **Exhaustive (Total):**
  - P in neither A or B

PROBLEMS:
- Redundancy
- Incomplete
- Querying

Inheritance - Three Kinds:

Common:
- METHODS
- VARIABLES

SUPER CLASS
(object type)

SUB CLASS
(object type)

Additional:
- METHODS
- VARIABLES

OBJECT
(instance)

MESSAGES

Super/Sub Classes
creates a Class Hierarchy
- selective?
- overrides?

© Gordon C. Everest, All rights reserved.
Benefits of Subtype/Supertype

- Consciously and Creatively think...
  - commonalities => generalization to supertype
  - special cases => specialization to subtypes
- Generalization can reveal common patterns for reuse.
- Abstraction for presentation, collapse the subtypes into their supertypes
  – equivalent of "leveling" in process/DFD models
- Use subtyping to aid human understanding with no intention of implementing as separate tables.
- Can approach design top-down, bottom-up, or middle-out
- Explicit representation of multiple table designs, thus deferring the choice for later implementation.

References

- Terry HALPIN, Information Modeling and Relational Databases, Morgan Kaufmann, 2001, in ORM context.
- Graeme C. SIMSSON and Graham C. WITT, Data Modeling Essentials, 3e, Morgan Kaufmann, 2005, chap. 4.
- Ramez ELMASRI and Shamkant NAVATHE, Fundamentals of Database Systems, 3e, Addison-Wesley, 2000, chapter 4 - academic, theoretical.