Modeling the Hierarchical Nature of Data

An Entity Relationship Approach
Introductions

- Richard Broersma
  - LAPUG Leader & PostgreSQL Enthusiast
  - Systems Integrator / Industrial Applications Developer
    Mangan Incorporated

- Mangan is a nationwide engineering and automation firm that serves multiple industries:
  - Petrochemical, Oil & Nat. Gas Pipelines, Oil & Nat. Gas Production
  - Chemical Production
  - Bio-Pharmaceutical Production
  - Solar Energy Production
1) Controversy
   - Hierarchical data in Relational Databases

2) Perceptions
   - Reality and Data modeling

3) Conceptual Designs
   - Entity Relationship Diagrams (ERD)
   - Modeling the Concept of Hierarchical Data

4) Physical Designs
   - Implementations of conceptual ERD Designs
In RDBMS, R is for **Relational**!

What's all this Hierarchical Nonsense?
Network and Hierarchical database are "things of the past."

Relational databases should be implemented using entities and relationships described in relational theory.

Should Hierarchical modeling be avoided?
Basics of ERD Modeling

- Thinking in terms of data modeling
  - Entities
  - Relationships
  - Entity types
- Implementation using 3 Normal forms
Perceptions of Reality

- Our Perceptions
  - Entities & Relationships
- Classifications
- Taxonomy
  - How do the attributes of similar type of entities differ
Short-fall of Entity Types

- Employee Types table
  - Can't express attribute similarities or differences of similar types.
  - Can't define relationships between people of related types.
Conclusion

- If we need to know about the:
  - Extended Attributes of Entity Types
  - Relationships between Entity Types
- Then hierarchical data modeling must be implemented
Conceptual Designs

- **ERD Provides Generalization Hierarchy Model**
  - Defines hierarchical constraints for hierarchical mapping.
  - Grouping of similar entity types.
  - Similarities and differences are defined.
  - Relationships can be created between entities of any (sub)type.
ERD - Hierarchical Constraints

- **$C_1$ Property**
  - \{T\} Total Coverage
  - \{P\} Partial Coverage

- **$C_2$ Property**
  - \{E\} Exclusive Coverage
  - \{O\} Overlapping Coverage

Coverage Properties

Animals

Carnivores

Herbivores
Entity types having equal attributes are grouped together.

Similarities and differences are defined.

Animals

<table>
<thead>
<tr>
<th>id</th>
<th>animalcode</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bear-1</td>
<td>bear</td>
<td>500</td>
</tr>
<tr>
<td>Sheep-2</td>
<td>sheep</td>
<td>100</td>
</tr>
<tr>
<td>Wolf-3</td>
<td>wolf</td>
<td>120</td>
</tr>
<tr>
<td>Deer-4</td>
<td>deer</td>
<td>240</td>
</tr>
<tr>
<td>Puma-5</td>
<td>puma</td>
<td>200</td>
</tr>
</tbody>
</table>

Carnivores

<table>
<thead>
<tr>
<th>id</th>
<th>animalcode</th>
<th>weight</th>
<th>favoritePrey</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bear-1</td>
<td>bear</td>
<td>500</td>
<td>salmon</td>
</tr>
<tr>
<td>Wolf-3</td>
<td>wolf</td>
<td>120</td>
<td>sheep</td>
</tr>
<tr>
<td>Puma-5</td>
<td>puma</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

Herbivores

<table>
<thead>
<tr>
<th>id</th>
<th>animalcode</th>
<th>weight</th>
<th>favoriteVegi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bear-1</td>
<td>bear</td>
<td>500</td>
<td>berries</td>
</tr>
<tr>
<td>Sheep-2</td>
<td>sheep</td>
<td>100</td>
<td>grass</td>
</tr>
<tr>
<td>Deer-5</td>
<td>deer</td>
<td>240</td>
<td>grass</td>
</tr>
</tbody>
</table>

Omnivores (implied by Overlapping)

<table>
<thead>
<tr>
<th>id</th>
<th>animalcode</th>
<th>weight</th>
<th>favoritePrey</th>
<th>favoriteVegi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bear-1</td>
<td>bear</td>
<td>500</td>
<td>salmon</td>
<td>berries</td>
</tr>
</tbody>
</table>
Beware of the “platypus”!

Valid criticisms of G/H exist.

Some entities will map to most of the hierarchical attributes but not all.

Careful consideration required to minimize platypus affect.*

*If practical, G/H redesign can eliminate most “Platypuses”.

Animals

Avians  Mammals  Reptiles

( T, O )

Newly Classified Monotremata
Complex Relationships are possible between sub types.
There are 5 physical designs (that I know of) for implementing conceptual Generalization Hierarchies.

Each physical design varies in the features that the conceptual Generalization Hierarchy Model defines.
Physical Designs

- **Entity-Attribute-Value Table (EAV)**
  ~Relational purists favorite

- **Nullable Attributes Table (NA)**
  ~Happens overtime

- **Vertical Disjunctive Table Partitioning (VDP)**
  ~My favorite

- **Horizontal Disjunctive Table Partitioning (HDP)**
  ~PostgreSQL Table inheritance

- **(NA–EAV) Hybrid Table**
  ~Worst Design Ever – know it to avoid it
# Implementation Features List

<table>
<thead>
<tr>
<th>Model VS. Features</th>
<th>EAV</th>
<th>NA</th>
<th>VDP</th>
<th>HDP</th>
<th>EAV-NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Coverage</td>
<td>A</td>
<td>E</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Partial Coverage</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Exclusive Coverage</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Overlapping Coverage</td>
<td>A</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Supports Grouping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Supports Relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

**Legend**

- **E** = Enforced (DDL or DRI)
- **A** = Allows (Client Enforced)
- **S** = Supports
Good Design Guidelines

- Always include an entity type column associated with the primary key throughout the hierarchy
  - This is still the best way to identify the type of hierarchical entity or hierarchical relationship
  - CHECK Constraints can then be implemented based on the entity types (Hierarchical Foreign Keys can be used also.)
  - This will prevent data corruption at the RDBMS tier that could be caused by application bugs or lazy users.
Entity Attribute Value (EAV)

- Physical Implementation:

Guard this table with your life!
CREATE TABLE Animaltypes(
    animalcode VARCHAR( 20 ) PRIMARY KEY,
    description TEXT NOT NULL DEFAULT ' ' );

CREATE TABLE Animals ( 
    animal_id VARCHAR( 20 ) PRIMARY KEY,
    animalcode VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )
        ON UPDATE CASCADE,
    weight NUMERIC( 7, 2 ) CHECK( weight > 0 ));

CREATE TABLE Attributetypes(
    attributecode VARCHAR( 20 ) PRIMARY KEY,
    description TEXT NOT NULL DEFAULT ' ' );

CREATE TABLE Animalattributes(
    animal_id VARCHAR( 20 ) REFERENCES Animals( animal_id )
        ON UPDATE CASCADE ON DELETE CASCADE,
    attribute VARCHAR( 20 ) REFERENCES Attributetypes( attributecode )
        ON UPDATE CASCADE,
    att_value TEXT NOT NULL,

PRIMARY KEY ( animal_id, attribute ));
Advantages:

- Provides a flexible mechanism to record the attributes associated with any entity.
- The flexibility eliminates the possibility of “platypuses”.
- This EAV design requires almost no consideration of the nature of the applicable hierarchical data and requires very little time to implement (cookie cutter).
Disadvantages:

- Users or Application logic becomes responsible to ensuring that all entities of a specific type will have the required associated attributes. (no DDL or DRI server constraints will work).
- The EAV table uses a TEXT (or VARCHAR) column for all attribute values regardless if Dates, Timestamps, Integers, Numerics or Booleans would be more appropriate.
- No Foreign Keys on Attribute Values: There isn't a way to prevent bad data-entry. For example nothing would prevent a user from entering 'I like peanut butter.' for the attribute value for Birth Date.
Null-able Attributes (NA) Table

- Physical Implementation:
CREATE TABLE Animaltypes(
    animalcode VARCHAR(20) PRIMARY KEY,
    description TEXT NOT NULL DEFAULT ''
);

CREATE TABLE Animals(
    animal_id VARCHAR(20) PRIMARY KEY,
    animalcode VARCHAR(20) REFERENCES Animaltypes(animalcode)
        ON UPDATE CASCADE,
    weight NUMERIC(7,2) CHECK(weight > 0),
    favoriteprey VARCHAR(20) REFERENCES Animaltypes(animalcode)
        ON UPDATE CASCADE
        CHECK(CASE WHEN animalcode IN ('Bear', 'Wolf', 'Puma')
            THEN favoriteprey IS NOT NULL
            WHEN animalcode IN ('Deer', 'Sheep')
            THEN favoriteprey IS NULL END ),
    favoritevegi VARCHAR(20) REFERENCES Vegitypes(Vegicode)
        ON UPDATE CASCADE
        CHECK(CASE WHEN animalcode IN ('Bear', 'Deer', 'Sheep')
            THEN favoritevegi IS NOT NULL
            WHEN animalcode IN ('Wolf', 'Pump')
            THEN favoritevegi IS NULL END )
);
Advantages:

- Provides a flexible mechanism to record the attributes associated with any entity.
- All attributes values can be constrained with foreign keys.
- Requires almost no consideration of the nature of the applicable hierarchical data. Hierarchical attributes are added via DDL as they are encountered during the life time of the application.
Disadvantages:

- Validating Hierarchical data integrity requires too many checks constraints. This can really hurt INSERT and UPDATE performance.
- Tuples in the table can get to be too big with many-many unused nulled columns.
- The concept of null can get obscured. Does Null mean “Don't Know” or “Doesn't Apply”.
(VDP) Table

Physical Implementation:

- Animals
  - Carnivores
  - Herbivores

- is a (0, 1)
- is a (1, 1)

- Animal Types
  - (0, n)

Diagram:

- Animals
  - Carnivores (1, 1)
  - Herbivores (1, 1)

- Animal Types
  - (0, n)
CREATE TABLE Animals (  
animal_id       VARCHAR( 20 ) UNIQUE NOT NULL,  
amimalcode      VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )  
ON UPDATE CASCADE,  
weight          NUMERIC( 7, 2) CHECK( weight > 0 ),  

   PRIMARY KEY ( animal_id, animalcode ),  
   --RI to handle denormalization of sub-tables);

CREATE TABLE Carnivores (  
animal_id       VARCHAR( 20 ) UNIQUE NOT NULL,  
amimalcode      VARCHAR( 20 ) NOT NULL  
CHECK( animalcode IN ( 'Bear', 'Wolf', 'Puma' )),  
favoriteprey    VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )  
ON UPDATE CASCADE,  

   PRIMARY KEY ( animal_id, animalcode ),  

   FOREIGN KEY ( animal_id, aminalcode )  
   REFERENCES Animals( animal_id, animalcode )  
   ON UPDATE CASCADE ON DELETE CASCADE,  

   --RI to handle denormalization of animalcode );
CREATE TABLE Herbivores (  
animal_id VARCHAR( 20 ) UNIQUE NOT NULL,  
animalcode VARCHAR( 20 ) NOT NULL  
CHECK( animalcode IN ( 'Deer', 'Sheep', 'Bear' )),  
favoriteprey VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )  
ON UPDATE CASCADE,  

PRIMARY KEY ( animal_id, animalcode ),  

FOREIGN KEY ( animal_id, animalcode )  
REFERENCES Animals( animal_id, animalcode )  
ON UPDATE CASCADE ON DELETE CASCADE,  

--RI to handle denormalization of animalcode );
Advantages:

- All attributes values can be constrained with foreign keys.
- Requires few Checks Constraints than the NA Table.
**Disadvantages:**

- Checks only required for Entity type field, but too many check constraints can still hurt INSERT performance.
- Additional Application logic required to handle multiple INSERTs and UPDATEs to various (sub)type tables.
- Requires some denormalization to enforce data integrity. Referential Integrity handles this problem.
- This design requires the designer to be well versed in the domain that is being modeled.
Physical Implementation:

- Animals
  - Carnivores
  - Herbivores

- Animal Types
  - Carnivores
  - Herbivores
  - Omnivores

(1, 1) is a (0, n)

T, O
CREATE TABLE Animals (  
    animal_id       VARCHAR( 20 ) PRIMARY KEY,  
    animalcode      VARCHAR( 20 ) REFERENCES AnimalTypes( animalcode )  
        ON UPDATE CASCADE,  
    weight          NUMERIC( 7, 2 ) CHECK( weight > 0 ));  

CREATE TABLE Carnivores (  
    favoriteprey    VARCHAR( 20 ))  
INHERITS( Animals );  

ALTER TABLE Carnivores  
ADD CONSTRAINT Carnivores_animalcode_check_is_carnivore  
CHECK( animalcode IN ( 'Bear', 'Wolf', 'Puma' ));
CREATE TABLE Herbivores (  
    favoritevegi VARCHAR(20))  
INHERITS( Animals );

ALTER TABLE Herbivores  
ADD CONSTRAINT Herbivores_animalcode_check_isHerbivore  
CHECK( animalcode IN ( 'Bear', 'Deer', 'Sheep' ));

CREATE TABLE Omnivores ()  
INHERITS( Carnivores, Herbivores ); -- PostgreSQL also inherits Check Constraint  
-- The Overlapping checks will algebraically  
-- reduce to CHECK( animalcode = 'Bear' )

-- CarnivoreCodes ∩ HerbivoreCodes = OmnivoreCodes
Advantages:

- All attributes values can be constrained with foreign keys. But you have to re-implement these Inherited foreign keys yourself.
- Possible to allow for relationships only between hierarchical leaf entities.
- The application logic is simplified since all accesses to sub-entities are to a single table.
HDP Table - Consideration

- Disadvantages:
  - SLOW Sequential Scans are the only way to search the Root or Branch nodes of the hierarchy since scans on these tables are based on UNION ALL queries.
  - Uniqueness cannot be enforced across the hierarchy.
  - This design requires the designer to be well versed in the domain that is being modeled.
(NA – EAV) Hybrid Table

- Physical Implementation:

```
<table>
<thead>
<tr>
<th>Animals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carnivores</td>
</tr>
<tr>
<td>Herbivores</td>
</tr>
</tbody>
</table>

Animals (1, 1) is a (0, n) Animal Types

(T, O)
```
CREATE TABLE Animaltypes(
    animalcode      VARCHAR( 20 ) PRIMARY KEY,
    description     TEXT NOT NULL DEFAULT ''
);

CREATE TABLE Animals (
    animal_id       VARCHAR( 20 ) PRIMARY KEY,
    animalcode      VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )
    ON UPDATE CASCADE,
    column1         VARCHAR( 255 ), --The application maps the attributes of each
    column2         VARCHAR( 255 ), --entity type to these intentionally vague
    column3         VARCHAR( 255 ), --columns. Each entity type will have a unique
    column4         VARCHAR( 255 ), --mapping for column1 thru column100.
    column5         VARCHAR( 255 ),
    column6         VARCHAR( 255 ), --Unmapped columns not needed by an entity type
    column7         VARCHAR( 255 ), --may be treated as custom fields that the users
    column8         VARCHAR( 255 ), --may use any way they see fit.
    -- ... 
    column100       VARCHAR( 255 )
);
Advantages:

- Provides a flexible mechanism to record the attributes associated with any entity.
- The flexible mechanism eliminates the possibility of “platypuses”.
Disadvantages:

- These VARCHAR columns have no meaning. Each entity can map a column for a completely unrelated attribute.

- The Application mapping becomes a major source of data corruption bugs if mapping isn't cleanly implemented or if entity type changes are required overtime.

- If unmapped columns are exposed to the users as custom column, there is not way to ensure that various users will be consistent when implementing these columns.

- Users or Application logic becomes responsible to ensuring that all entities of a specific type will have the required associated attributes. (no DDL server constraints will work)

- The NAEAV table uses a VARCHAR column for all attribute values regardless if Dates, Timestamps, Integers, Numerics or Booleans would be more appropriate

- No Foreign Keys on Attribute Columns: The isn't a way to prevent bad data-entry. For example nothing would prevent a user from entering 'I like peanut butter.' for the attribute value for Birthday

- Table design concept is badly de-normalized.
Works Cited


I have a question! What's all this Hierarchical Nonsense?