Hierarchical data models in Relational Databases

In RDBMS, R is for **Relational**. What's all this hierarchical nonsense?

Richard Broersma Jr.
Mangan Inc. (Systems Integrator)
PostgreSQL Enthusiast
Controversy of Hierarchical data in Relational Databases
Perceptions of Reality and Data modeling
Logical ERDs for Generalization Hierarchies
Exploring Physical implementations of logical ERDs
Controversy

- 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 RDB Modeling

- Thinking in terms of data modeling
  - Entities
  - Relationships
  - Entity types
- Implement using 3 Normal forms.
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 or the relationships between entity types, Hierarchical data modeling must be implemented.
Generalization Hierarchy (logical modeling):

- 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
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”.
Complex Relationships are possible between sub types

<table>
<thead>
<tr>
<th>Fears</th>
<th>carnivoreid</th>
<th>animalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Deer-4</td>
<td>Wolf-3</td>
</tr>
<tr>
<td>1</td>
<td>Deer-4</td>
<td>Puma-5</td>
</tr>
<tr>
<td>2</td>
<td>Deer-4</td>
<td>Bear-1</td>
</tr>
<tr>
<td>3</td>
<td>Sheep-2</td>
<td>Wolf-3</td>
</tr>
<tr>
<td>4</td>
<td>Sheep-2</td>
<td>Puma-5</td>
</tr>
<tr>
<td>5</td>
<td>Bear-1</td>
<td>Bear-6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maulings</th>
<th>carnivoreid</th>
<th>animalid</th>
<th>Mauling-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bear-1</td>
<td>Wolf-3</td>
<td>01/15/08</td>
</tr>
<tr>
<td>1</td>
<td>Bear-1</td>
<td>Deer-4</td>
<td>07/12/07</td>
</tr>
<tr>
<td>2</td>
<td>Wolf-3</td>
<td>Sheep-2</td>
<td>09/22/07</td>
</tr>
</tbody>
</table>
Physical Implementations

- There are 5 physical designs for implementing logical Generalization Hierarchies.
- Each physical design varies in the G/H features that it is able to implement.
  - Entity-Attribute-Value table (EAV) (Relational purists favorite)
  - Null-able Attributes (NA) table (Happens overtime)
  - Vertical Disjunctive Partitioning (VDP) table partitioning (My favorite)
  - Horizontal Disjunctive Partitioning (HDP) table (i.e. PostgreSQL Table inheritance)
  - Null-able attributes – EAV hybrid Table (Worst Design Ever – know it to avoid it)
Good Design Guidelines

- Regardless of the physical implementation:
  - Always include a type column associated with the primary key.
    - This is still the best way to identify an entities or relationships type.
      - CHECK( ... ) Constraints can then be implemented based on the entity type *
      - This will prevent data corruption at the server level that could be caused by application bugs or lazy users.

* A tree that mirrors the structure of the Generalization Hierarchy can be used in coordination with a custom lookup function can replace lengthy check constraints.
Physical Implementation:

- Animals
  - Carnivores
  - Herbivores

- Animal Attributes
  - is a
  - has
  - is a

- Animal Types
  - (0, n)

- Attribute Types
  - (0, n)
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 VARCHAR(1000) NOT NULL,
  PRIMARY KEY ( animal_id, attribute ));
EAV Table - Consideration

- **Good**
  - Provides a flexible mechanism to record the attributes associated with any entity.
  - The flexible mechanism 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)

- **Bad**
  - 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 EAV table doesn't provide a mechanism to create relationships between entities of different sub-types.
  - The EAV table does nothing to provide a grouping of related entity types.
  - The EAV table uses a VARCHAR column for all attribute values regardless if Dates, timestamps, integers, numerics or booleans would be more appropriate
  - 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
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=ANY('Bear', 'Wolf', 'Puma') 
            THEN favoriteprey IS NOT NULL 
            ELSE favoriteprey IS NULL END )

favoritevegi      VARCHAR( 20 ) REFERENCES Vegitypes ( Vegicode ) 
        ON UPDATE CASCADE 
        CHECK( CASE WHEN animalcode=ANY('Bear', 'Deer', 'Sheep') 
            THEN favoritevegi IS NOT NULL 
            ELSE favoritevegi IS NULL END )
);
NA Table - Consideration

- **Good**
  - The most Common Hierarchical Table I've seen. Is that a good thing?
  - Provides a flexible mechanism to record the attributes associated with any entity.
  - All attributes values can be constrained with foreign keys.
  - This NA design requires almost not consideration of the nature of the applicable hierarchical data. Hierarchical attributes are added via DDL as they are encounter during runtime.

- **Bad**
  - The NA table doesn't provide a mechanism to create relationships between entities of different sub-types.
  - The NA table does nothing to provide a grouping of related entity types.
  - Fewer Checks required, but too many check constraints can still hurt INSERT performance.
  - Tuples in the table can get to be too big with many-many unused nulled columns.
  - The concept of null gets obscured.
Physical Implementation:

- Animals
  - Carnivores
  - Herbivores

- Animal Types
  - is a
    - Animals
      - is a
        - Carnivores
        - Herbivores
  - is a
    - Carnivores
    - Herbivores
CREATE TABLE Animals ( 
  animal_id       VARCHAR( 20 ) UNIQUE NOT NULL, 
  animalcode      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 Carnivore ( 
  animal_id       VARCHAR( 20 ) UNIQUE NOT NULL, 
  animalcode      VARCHAR( 20 ) NOT NULL 
    CHECK( animalcode = ANY( 'Bear', 'Wolf', 'Puma' ) ), 
  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 
);
### Good

- All attributes values can be constrained with foreign keys.
- Almost all logical ERD concepts of Generalizations Hierarchies can be implemented with this design.
- Allow for relationships between all levels of subtype entities
- Allows for entity type grouping of related entities

### Bad

- Checks only required for Entity type field, but too many check constraints can still hurt INSERT performance
- VDP cannot enforce overlapping when required by entity type.
- 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
  - Animals
  - Carnivores
  - Herbivores

The diagram illustrates the relationship between Animals, Carnivores, Herbivores, and Animal Types with cardinality constraints: (T, O), (1, 1), (1, n), (0, n).
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 Carnivore (  
  favoriteprey VARCHAR( 20 ))  
INHERITS( Animals ); 

ALTER TABLE Carnivore  
ADD CONSTRAINT Cornivore_animalcode_check_iscarnivore  
  CHECK( animalcode = ANY( 'Bear', 'Wolf', 'Puma' )); 

CREATE TABLE Herbivore (  
  favoritevegi VARCHAR( 20 ))  
INHERITS( Animals ); 

ALTER TABLE Herbivore  
ADD CONSTRAINT Herbivore_animalcode_check_isHerbivore  
  CHECK( animalcode = ANY( 'Bear', 'Deer', 'Sheep' ));
<table>
<thead>
<tr>
<th><strong>Good</strong></th>
<th><strong>Bad</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- All attributes values can be constrained with foreign keys.</td>
<td>- Checks only required for Entity type field, but too many check constraints can still hurt INSERT performance</td>
</tr>
<tr>
<td>- Allow for relationships between hierarchical leaf entities</td>
<td>- HDP correctly implement overlapping when required by entity type.</td>
</tr>
<tr>
<td>- Allows for entity type grouping of related entities</td>
<td>- No relationships can be drawn between various levels of the G/H.</td>
</tr>
<tr>
<td>- The application logic is simplified since all accesses to sub-entities are to a single table.</td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- Uniqueness cannot be enforced across the hierarchy.</td>
</tr>
<tr>
<td></td>
<td>- This design requires the designer to be well versed in the domain that is being modeled</td>
</tr>
</tbody>
</table>
(NA – EAV) Hybrid Table

- Physical Implementation:

- Animals
  - Carnivores
  - Herbivores

- Animals
  - is a
  - Animal Types

- Constraints:
  - $(T, O)$
  - $(1, 1)$
  - $(0, n)$
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 )
);
NA – EAV Table - Consideration

### Good

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

### Bad

- 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 doesn't provide a mechanism to create relationships between entities of different sub-types.
- The NAEAV table does nothing to provide a grouping of related entity types.
- The NAEAV table uses a VARCHAR column for all attribute values regardless if Dates, timestamps, integers, numerics or booleans would be more appropriate
- 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.

Questions?

Ya, what's all this hierarchal nonsense?