Hierarchical data models in Relational Databases

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

Richard Broersma Jr.
Mangan Inc. (Systems Integrator)
PostgreSQL Enthusiast

Preview

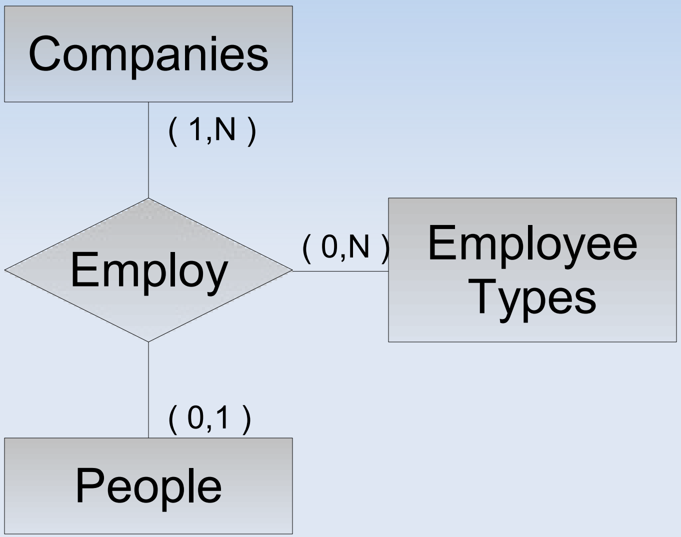
- Controversy of Hierarchical data in Relational Databases
- Perceptions of Reality and Data modeling
- Logical ERDs for Generalization Hierarchies
- Exploring Physical implementations of logical ERDs

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?

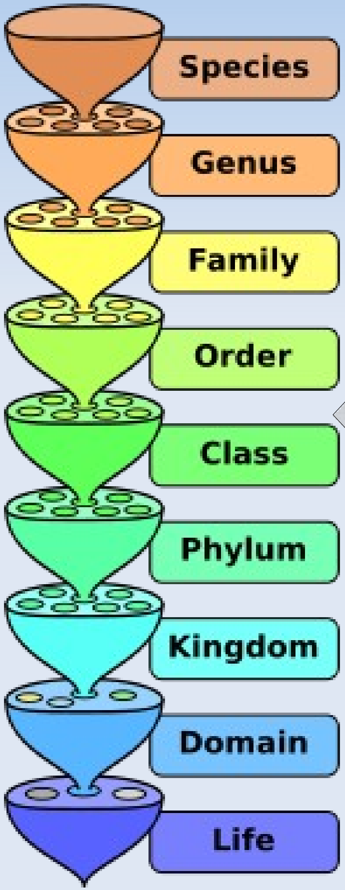
Basics of RDB Modeling

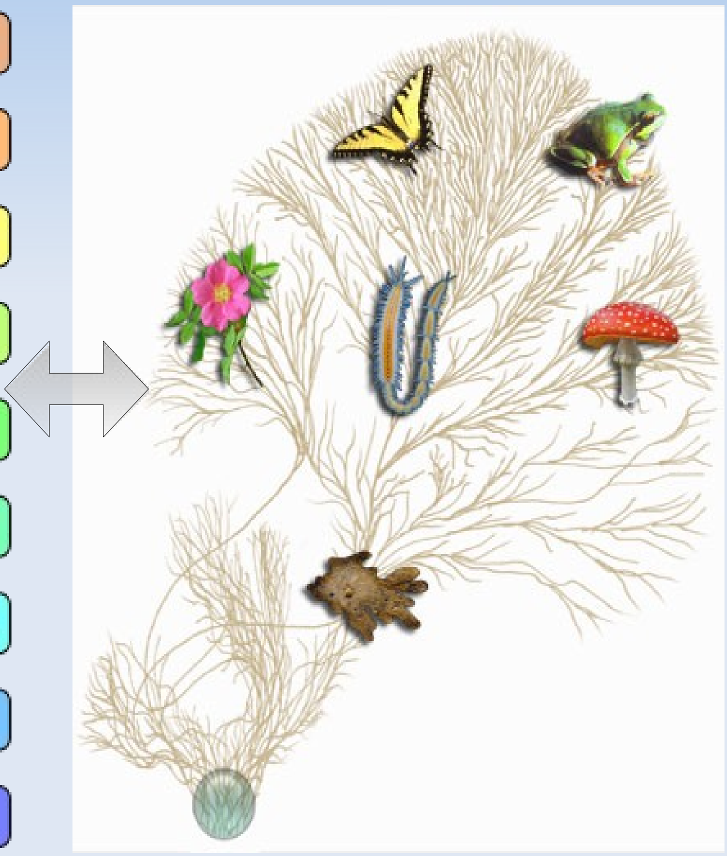
- Thinking in terms of data modeling
  - Entities
  - Relationships
  - Entity types
- Implement using 3 Normal forms.



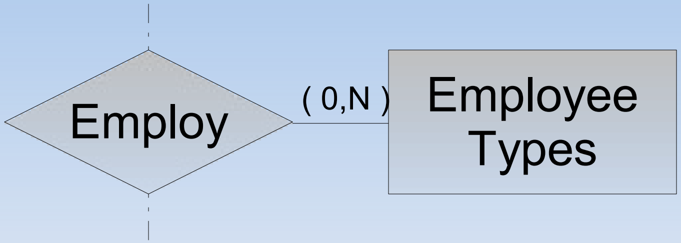
Perceptions of Reality

- Our Perceptions
  - Entities & Relationships
  - **Classifications**
- Taxonomy
  - How do the attributes of similar type of entities differ





Short-fall of Entity Types



Employee Types	
Electrician	Does Electrical Work
Engineer	Does Engineering
Manager	Manages others
President	Presides over a company
Welder	Welds

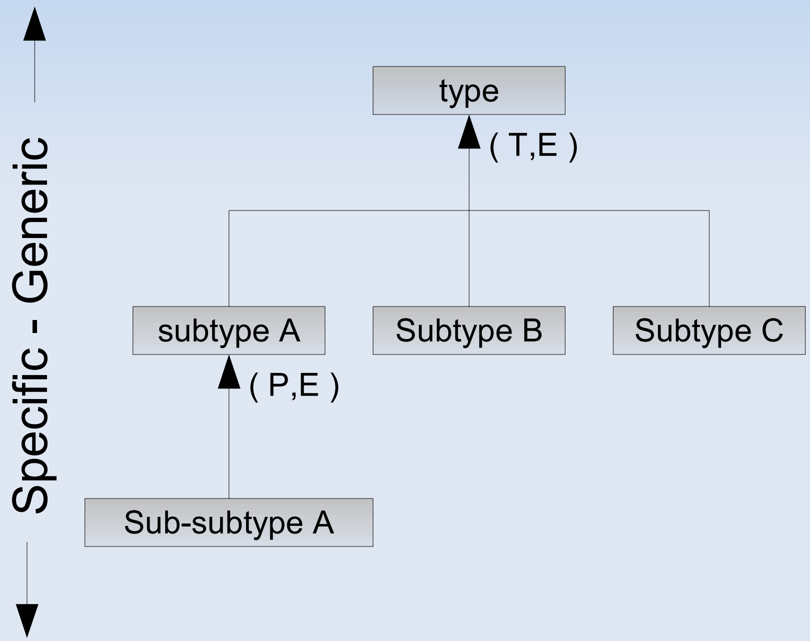
Employ		
ABC	TED	Welder
ABC	SANDY	President
ACME	RON	Electrician
ACME	JILL	Engineer
BP	DAVE	Manager

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

<u>Conclusion:</u> If we need to know about the extended attributes of entity types or the relationships between entity types, Hierarchical data modeling must be implemented.

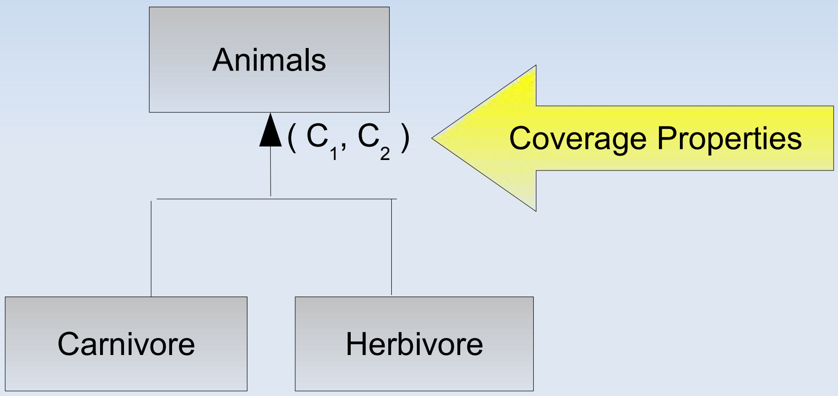
Enter - ERD for Hierarchical Data

- 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.



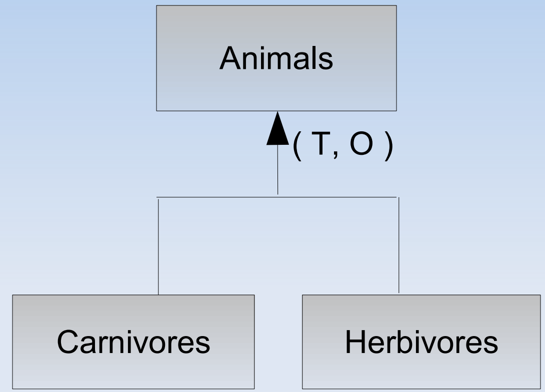
ERD - Hierarchical Constraints

- C_1 Property
  - {T} Total Coverage
  - {P} Partial Coverage
- C_2 Property
  - {E} Exclusive Coverage
  - {O} Overlapping Coverage



ERD - Entity type Groupings

- Entity types having equal attributes are grouped together.
- Similarities and differences are defined.



Animals		
id	**animalcode**	**weight**
Bear-1	bear	500
Sheep-2	sheep	100
Wolf-3	wolf	120
Deer-4	deer	240
Puma-5	puma	200

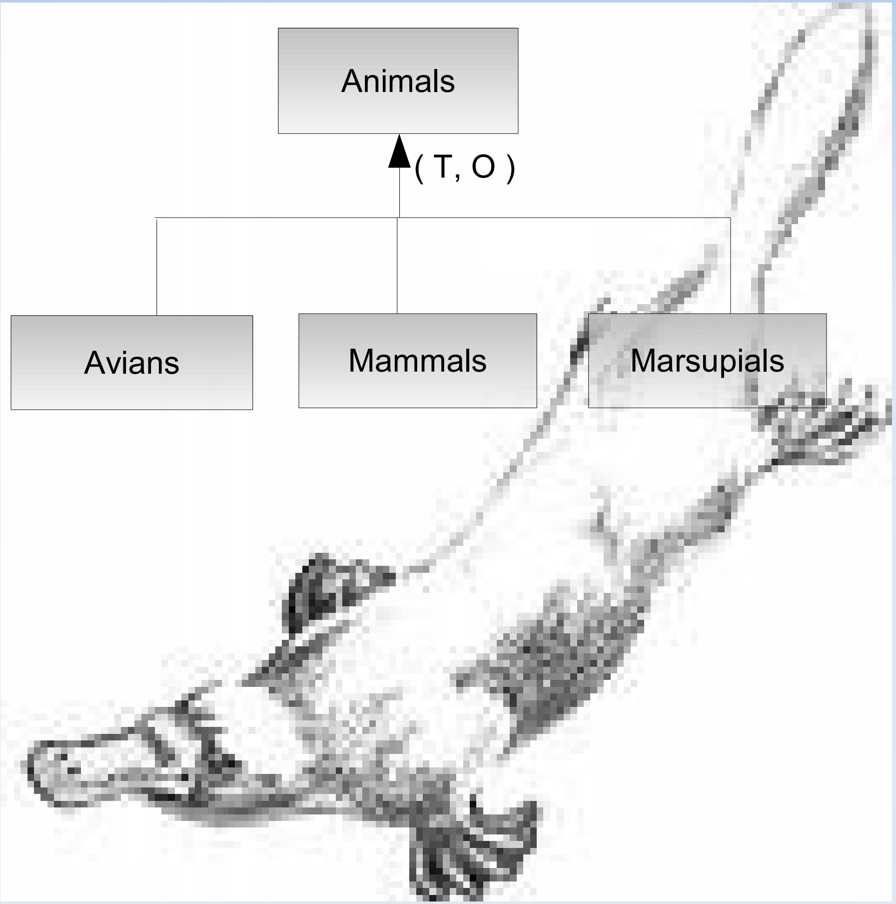
Carnivores			
id	**animalcode**	**weight**	**favoritePrey**
Bear-1	bear	500	salmon
Wolf-3	wolf	120	sheep
Puma-5	puma	200	deer

Herbivores			
id	**animalcode**	**weight**	**favoriteVegi**
Bear-1	bear	500	berries
Sheep-2	sheep	100	grass
Deer-5	deer	240	grass

Omnivores (implied by Overlapping)				
id	**animalcode**	**weight**	**favoritePrey**	**favoriteVegi**
Bear-1	bear	500	salmon	berries

ERD - Entity type Groupings

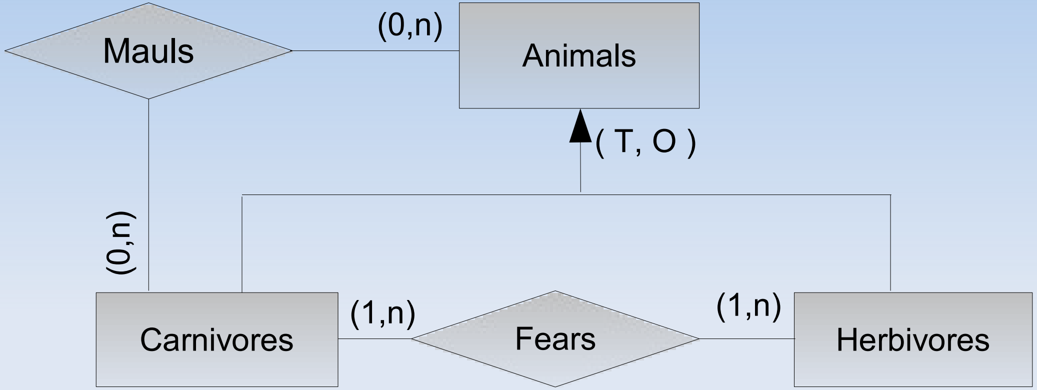
- Beware of the “platypus”!
  - Valid criticisms of G/H exist.
  - Some entities will map to most of the hierarchical attributes but not all.
  - Careful consideration required to minimize platypus affect.*



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

ERD – Entity type Relationships

- Complex Relationships are possible between sub types



Fears	
carnivoreid	**animalid**
Deer-4	Wolf-3
Deer-4	Puma-5
Deer-4	Bear-1
Sheep-2	Wolf-3
Sheep-2	Puma-5
Bear-1	Bear-6

Maulings		
carnivoreid	**animalid**	**Mauling-date**
Bear-1	Wolf-3	01/15/08
Bear-1	Deer-4	07/12/07
Wolf-3	Sheep-2	09/22/07

Physical Implementations

- There are 5 physical designs for implementing logical Generalization Hierarchies
- Each physical design varies in the G/H features that its 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)

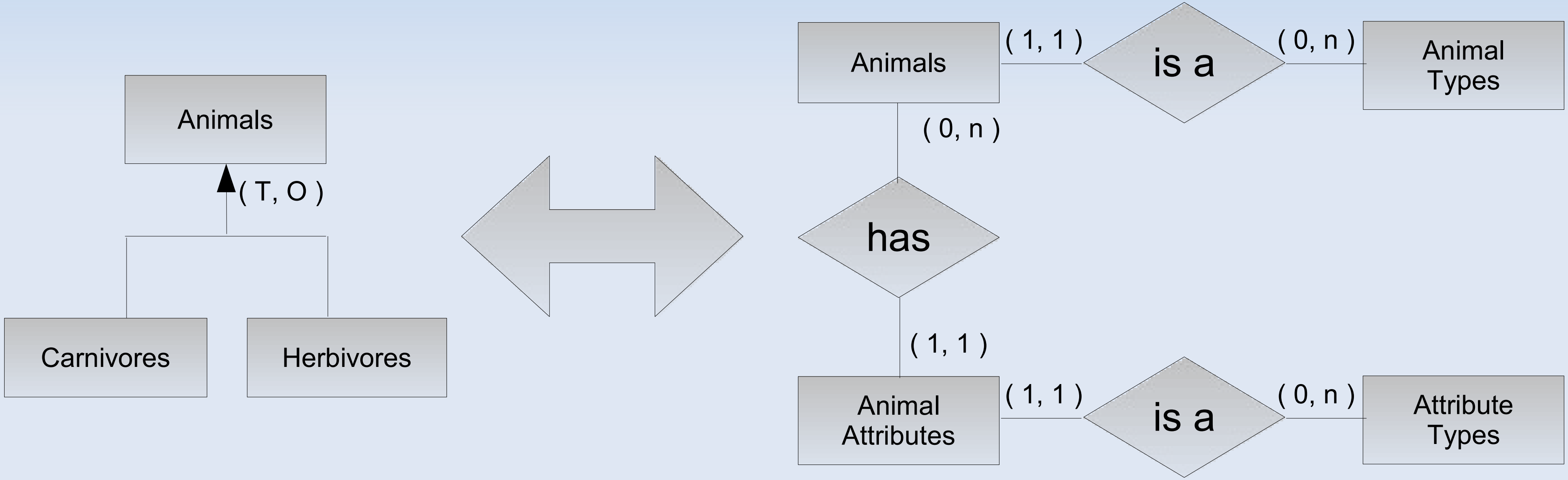
Good Design Guidelines

- 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.

Entity Attribute Value (EAV)

- Physical Implementation:



EAV Table - DDL

```
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 ));
```

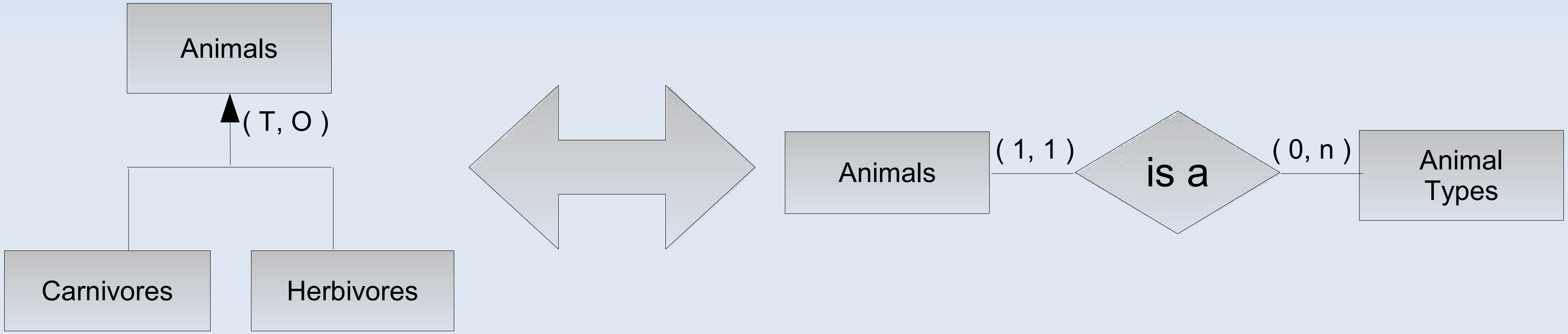
EAV Table - Consideration

- 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:



(NA) Table - DDL

```
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 )
);
```

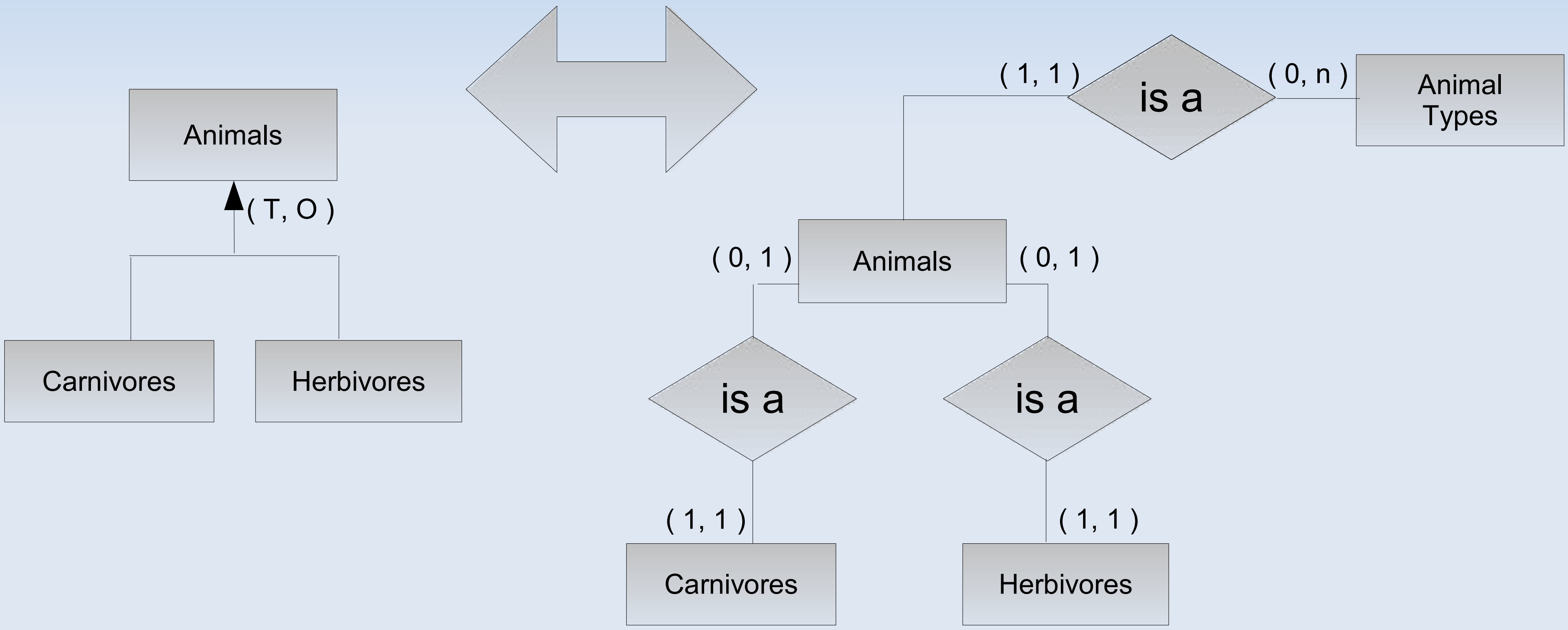
NA Table - Consideration

- ## 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.

(VDP) Table

- Physical Implementation:



(VDP) Table - DDL

```
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 );
```

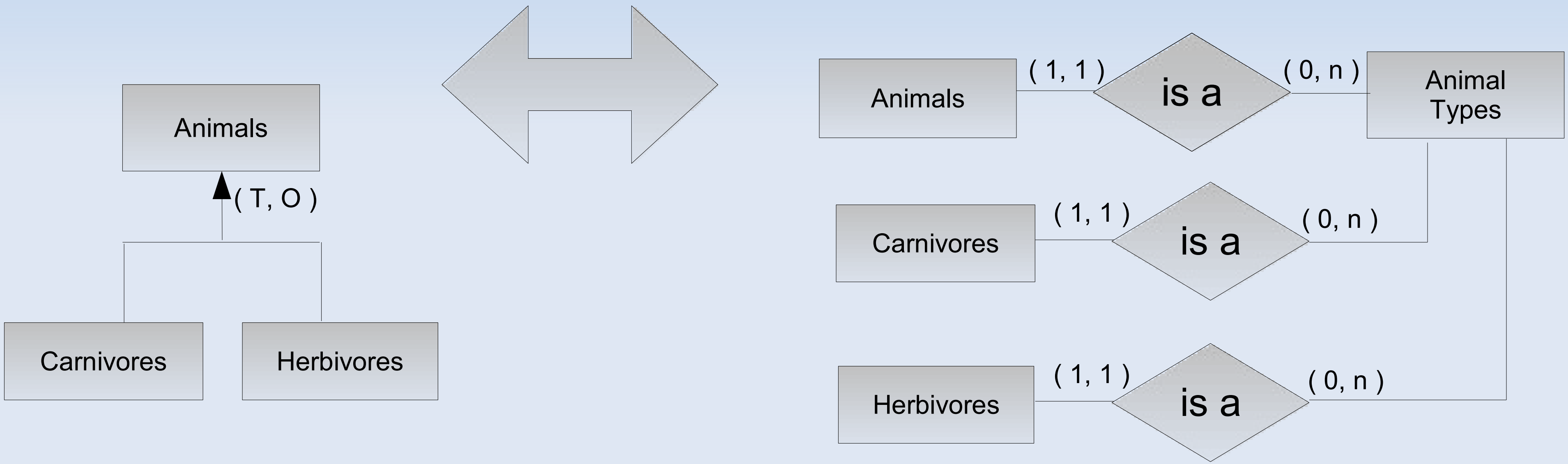
VDP Table - Consideration

- 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 entitles
  - 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

(HDP) Table

- Physical Implementation:



(HDP) Table - DDL

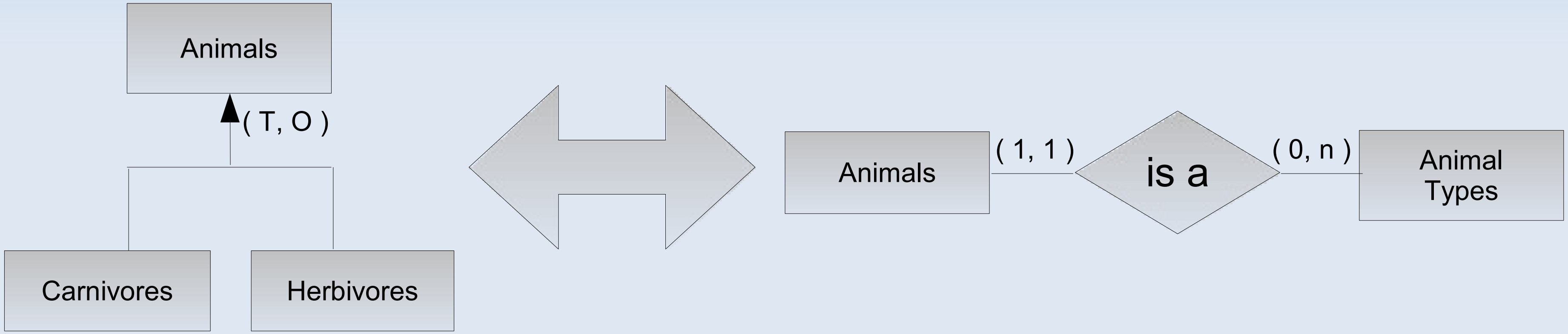
```
CREATE TABLE Animals (
   animal_id        VARCHAR( 20 ) PRIMARY KEY,
   animalcode       VARCHAR( 20 ) REFERENCES Animaltypes( animalcode )
                    ON UPDATE CASCADE,
   weight           NUMERIC( 7, 2) CHECK( weight > 0 ));

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' ));
```

HDP Table - Consideration

- Good
  - All attributes values can be constrained with foreign keys.
  - Allow for relationships between hierarchical leaf entitles
  - Allows for entity type grouping of related entities
  - The application logic is simplified since all accesses to sub-entities are to a single table.
- Bad
  - Checks only required for Entity type field, but too many check constraints can still hurt INSERT performance
  - HDP correctly implement overlapping when required by entity type.
  - No relationships can be drawn between various levels of the G/H.
  - 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:



(NA – EAV) Table - DDL

```
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.

Bibliography

Works Cited

Batini, Carol, Stefano Ceri, and Shamkant B. Navathe. Conceptual Database Design: an Entity-Relationship Approach. Redwood City, California 94065: The Benjamin/Cummings Company; Inc., 1992. 1-470.

Celko, Joe. SQL FOR SMARTIES: Advanced SQL Programming. 3rd ed. San Francisco California 94111: Morgan Kaufmann Publications, 2005. 1-838.

Questions?

Ya, what's
all this
hierarchal
nonsense?