

# **Mastering Data Modeling**

## **a dozen key passages**

<b>12-15</b>	<b>many ‘hows’, one ‘what’</b>
<b>84</b>	<b>skills to master</b>
<b>139-144</b>	<b>chicken feet in, out, across; +</b>
<b>174</b>	<b>valid relationships</b>
<b>178</b>	<b>good names</b>
<b>220</b>	<b>the flow</b>
<b>248</b>	<b>grounded</b>
<b>253</b>	<b>W’s</b>
<b>260</b>	<b>misplaced descriptors</b>
<b>309</b>	<b>meta-model</b>
<b>335-339</b>	<b>mapping to relational schema</b>
<b>347-349</b>	<b>graph recipes</b>

## **Singleton/Plural   a.k.a.   One/Many**

**Here are user-supplied instances.  
What's the difference?**

1. My sister, who plays the violin,  
will be home soon.
2. My sister who plays the violin  
will be home soon.

**How many sisters?  
How many sisters play the violin?**

---

**Here is a user-supplied instance.  
Draw corresponding data model.**

Please go out to the garage and bring me  
the rake, which is broken.

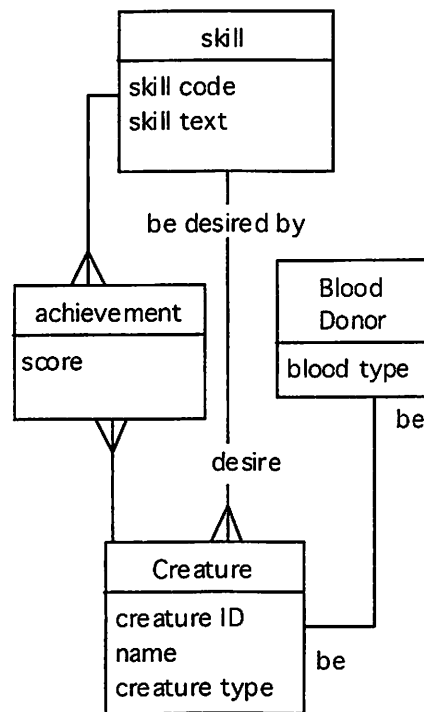
**Which is correct? Why?**

1. Please go out to the garage and bring me  
the rake, which is broken.
2. Please go out to the garage and bring me  
the rake which is broken.
3. Please go out to the garage and bring me  
the rake that is broken.
4. Please go out to the garage and bring me  
the rake, that is broken.

1 & 3 are grammatically correct  
2 & 4 are not

**How many rakes?**

### Ch 3. Reading an LDS



---

### First sentence

About each

*some box name*

we can remember

*set of names.*

---

### Second sentence

*Some box name*

is identified by

*subset of the set of names.*

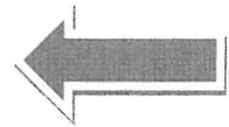
## Ch 4. LDS Notation

*Names of types; populate with instances*

Data Model [set of Entities]

Entity [singular]

*An entity has descriptors*



Descriptor

Attribute [singular]

Link

Relationship [of 2 Links]

Identifier [set of Descriptors]

Link

Single- or multi- valued [max]

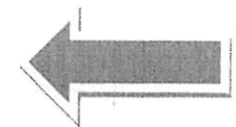
Labeled or unlabeled [mostly unlabeled]

“to-be” or not “to-be” [mostly not]

Identifying Descriptor

Symbol [convention]

“bar” “|” across Link or “\_” under Attribute



# Connecting LDS Reading and Vocabulary

John Carlis

In *Chapter 3: Reading an LDS with Sentences* you learned to read two sentences for each box in these formats :

- Plus
1. About each <box name> we can remember its remembered things.
  2. <box name> is identified by some (or all) of its remembered things - the barred ones.
- Or
2. About each <box name> we must remember its barred remembered things and no two instances of <box name> can have the same value for the entire set of barred remembered things.

Using set notation the formats look like this:

1. About each <box name> we can remember its {remembered thing}.<sup>1</sup>
- Plus
2. <box name> is identified by {barred remembered thing}.

In *Chapter 4: Vocabulary of LDS* you learned a modeler's vocabulary:

About each entity we can remember its descriptors.

An entity is identified by its identifying descriptors.

Or, with set notation:

About each entity we can remember {descriptor}.

An entity is identified by {identifying descriptor}.

Here are several elaborations, that is, successively more specific ways of naming the pieces of an LDS.

- A.
- About each <box name> we can remember its
  - {inside-the-box name}
  - {one-direction box-to-box-line name}.

This becomes, with LDS vocabulary:

About each entity we can remember its

- {attribute},
- {link}.

- B.
- About each entity we can remember its
  - {attribute},
  - {single valued link},
  - {multi-valued link}.

Since attributes and single valued links together form the single valued descriptors, this can be restated as:

About each entity we can remember its

- {single valued descriptor},
- {multi-valued link}.

Notice that {multi valued descriptor} is the same as {multi valued link}.

- C.
- About each entity we can remember its
  - {attribute },
  - {single valued unlabeled link},
  - {single-valued labeled link},
  - {multi- valued unlabeled link},

---

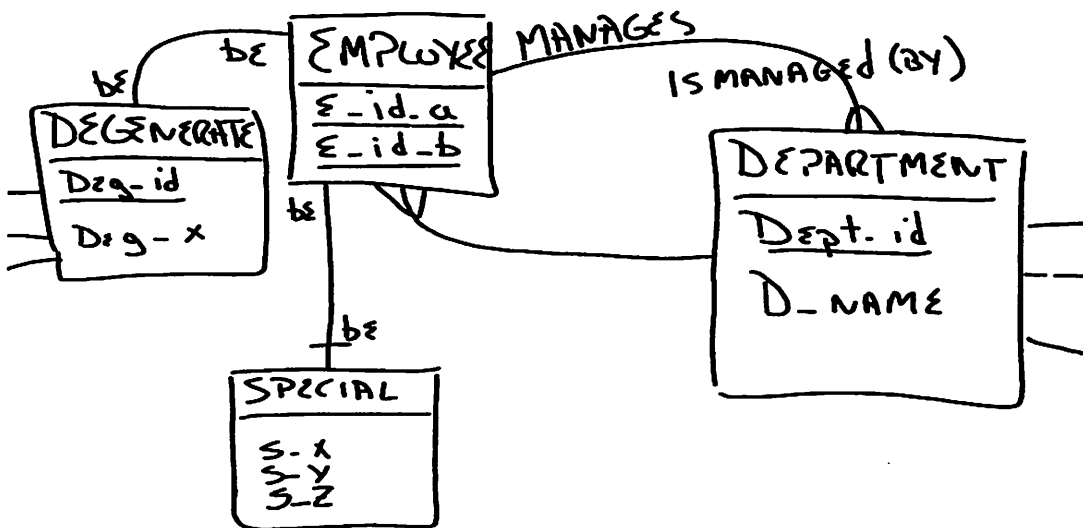
<sup>1</sup> When you see {x}, say "a set of x's".

- {multi-valued labeled link}.

D. About each entity we can remember its

- {non-identifying attribute},
- {identifying attribute},
- {non-identifying single valued unlabeled link},
- {not to-be non-identifying single-valued labeled link},
- {to-be non-identifying single-valued labeled link},
- {identifying single valued unlabeled link},
- {not to-be identifying single-valued labeled link},
- {to-be identifying single-valued labeled link},
- {non-identifying multi- valued unlabeled link},
- {non-identifying multi-valued labeled link}.
- {identifying multi- valued unlabeled link},
- {identifying multi-valued labeled link}.

Study the fragment below. Notice that there are some incomplete lines, which indicate that the fragment is embedded in a larger LDS.



A. Write out the two sentences for each entity

ABOUT EACH Department  
WE CAN REMEMBER ITS  
DEPT-id, D-NAME, EMPLOYEES,  
AND THE EMPLOYEE IT IS MANAGED BY.

Department is identified by its Dept-id.

B. For each level of specificity A-D, state how many of each kind of thing you see.

ABOUT EACH Department  
WE CAN REMEMBER ITS  
<sup>4 descriptors</sup> { DEPT-id, D-NAME, EMPLOYEES,  
AND THE EMPLOYEE IT IS MANAGED BY.

Department is identified by its Dept-id.  
1 identifying descriptor

ABOUT EACH Department  
WE CAN REMEMBER ITS  
<sup>Attributes</sup> { DEPT-id, D-NAME,  
<sup>Link</sup> { EMPLOYEES AND THE EMPLOYEE IT IS MANAGED BY.

Department is identified by its Dept-id.

ABOUT EACH DEPARTMENT

WE CAN REMEMBER ITS

ATTRIBUTE DEPT-ID, D-NAME,

LINK TO EMPLOYEES AND THE EMPLOYEE IT IS MANAGED BY.

DEPARTMENT IS IDENTIFIED BY ITS DEPT-ID.



## Ch. 7 Mastering Shapes

Shape: pattern of boxes, lines, chicken feet, bars.

- not positional
- content-neutral

Shape reading skills

- aloud, in several ways
- visualize sample data, in several ways
- visualize disallowed data, in several ways

Shape recognition skills

- Finding and focusing on shapes within a large LDS
- Recognizing the differences between shapes that are similar
- Recognizing the similarity between seemingly dissimilar shapes
- Distinguishing between legitimate shapes and syntactically invalid LDS fragments

Shape evolution skills

- Knowing how shapes are likely to evolve
- Asking questions that help users choose between two similar shapes
- Knowing when to ask questions of users
- Knowing when and how to modify the LDS to make a shape evolve
- Understanding the relative frequency of the various shapes

Shape studying skills

- Referring to each fundamental shape by its name

## Ch. 8 One Entity Shapes

Entity's Kind based on its identifying descriptors.

Independent.

- Common independent entity.
  - zero links; > 0 attributes; not every attribute.
- Lonely-attribute independent entity. (*rare*)
  - zero links; 1 attribute; every attribute.
- Aggregate independent entity (*rare*)
  - zero links; > 1 attributes; every attribute.

Dependent.

- = 1 link (a degree-one link of a one-many relationship); and >0 attributes.

Intersection.

- > 1 link (each a degree-one links of one-many relationships; and any number of attributes.

Subordinate.

(*rare*)

- = one link (of a one-one relationship); and = 0 attributes.

Collection.

(*very rare*)

- = one descriptor, a degree-many link.
  - One-many collection entity.
  - Many-many collection entity.

## Ch. 9 One Attribute Shapes

An attribute's shape comes from its SCALE.

Nominal.

- names.
- no ordering.
- can test for = or < >.
- can be encoded with numbers  
*but is still nominal.*
- commonly an identifying descriptor.

Numeric.

- real numbers (perhaps restricted).
- can test for difference, and, perhaps, for ratio.
- rarely an identifying descriptor.

Ordinal.

- ranking (perhaps overall, perhaps within a group).
- can test for < or = or >.
- often an identifying descriptor.

Boolean.

- True or False value.
- can test for true or false (obviously).
- seldom an identifying descriptor.

Scale < > Datatype

**Warning!!!!** *If you see numbers, don't assume numeric scale.*

---

## Ch. 10 Two Entity Shapes

### *Two Entities, One Relationship*

#### One-many shapes

- Plain one-many relationship
- One-many relationship making a dependent or intersection entity
- One-many relationship making a collection entity

#### One-one shapes

- Plain one-one relationship
- One-one relationship making a subordinate entity
- To-be one-one relationship
- Not-to-be one-one relationship
- To-be one-one relationship making a subordinate entity
- Plain to-be one-one relationship
- Plain not-to-be one-one relationship
- Not-to-be one-one relationship making a subordinate entity

#### Many-many shapes

- Plain many-many relationship
- Many-many relationship making a collection entity

### *Two Entities, Two Relationships*

- Two One-One Relationships

#### One-One and One-Many Relationship

- Not-to-be One-One Relationship and One-Many Relationship
- To-be Relationship and One-Many Relationship

#### Two One-Many Relationships

- Two Same-Direction One-Many Relationships
  - directed graph *versus* not directed graph
- Two Opposite-Direction One-Many Relationships
- Many-Many Relationship Plus Some Other Relationships

### *Two Entities, n Relationships*

---

## Ch. 11 More Than Two Entity Shapes

### Chicken feet (Ignore bars)

- in
- out
- across

### Subordinates

- out
- across

### Multiple to-be relationships

### Multiple short paths

---

## Ch. 12 Reflexive Relationship Shapes

### Shape: One-One Reflexive Relationship

### Sequence Data and Cyclic Sequence Data

### Ordered Pairs

### One-Many Reflexive Relationship

### Many-Many Reflexive Relationship

## Chapter 13 LDS Syntax Rules

Unique Names

No Reflexive Relationship Is a To-be Relationship

Between Any Pair of Entities, There Is at Most One \_To-be Relationship

---

### *Identifiers*

Each Entity Has > 0 Identifiers

No Cycles of Identification Dependency

No Link of a Reflexive Relationship in an Identifier

Both Links Cannot Contribute to an Identifier

A Single-Descriptor Identifier Cannot Include the Degree-One Link of a One-Many Relationship

A Multiple-Descriptor Identifier Cannot Include a Link of a One-One Relationship

---

## Chapter 14 Getting the Names Right

Entity Names

- Too-Exclusive (missing instances)                      person
- Too-Inclusive (extra instances)                                      achievement event
- Too-Coarse (one instance for many)                                      city
- Too-Fine (many instances for one)                                      city
- Completely Inaccurate                                      soils 5
- Overloaded    theatre

---

## Chapter 16 -Labeling Links

Most relationships do **not** have labeled links.

Either **neither** or **both** links of a relationship have labels.

Each One-One Relationship Has Labels

Each Reflexive Relationships Has Labels

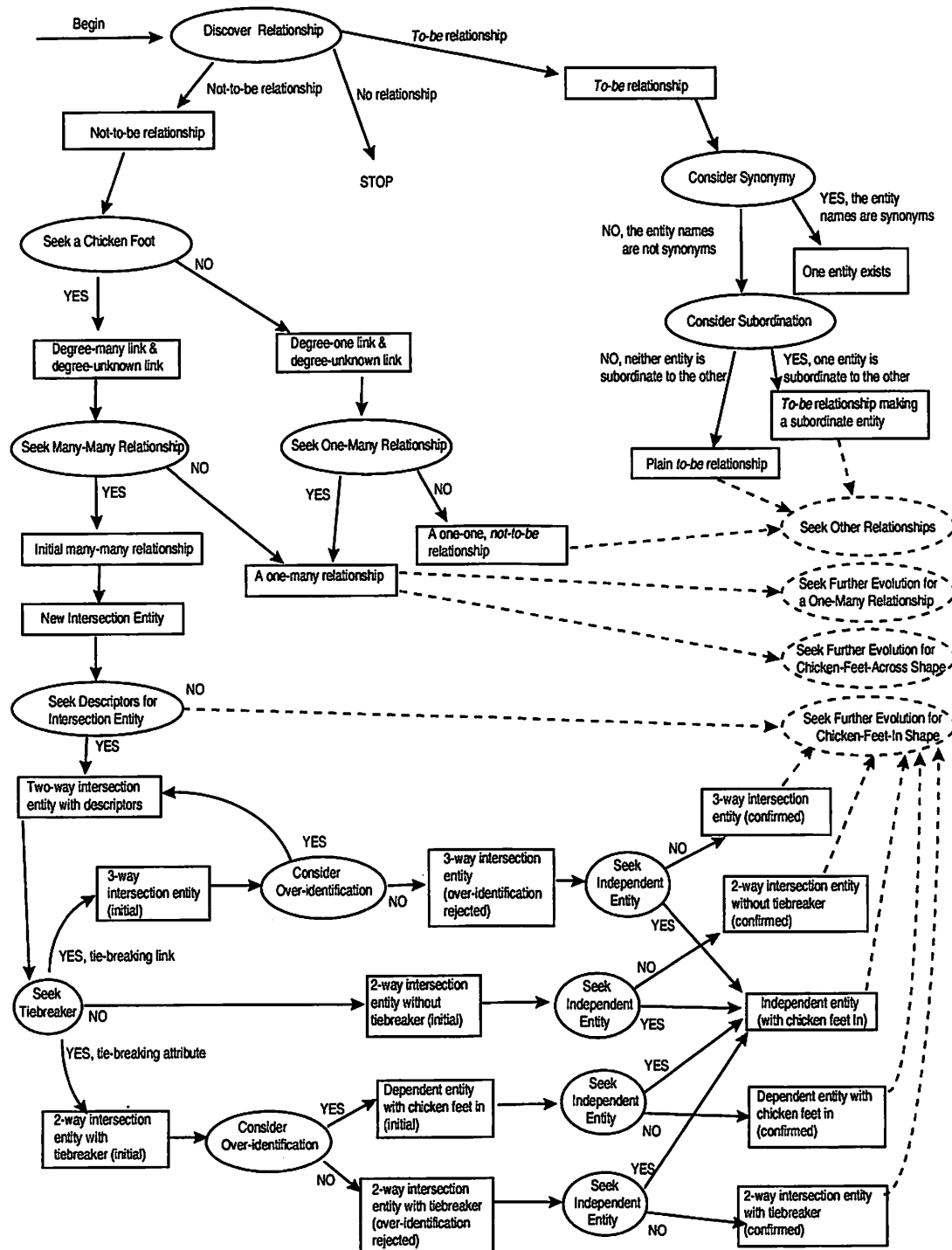
Between Any Pair of Entities, There Is at Most One Unlabeled Relationship

Link labels do **not** include an entity name.

A good label uses a verb.

- The link labels of a relationship use the same verb.
- Prefer a transitive verb (achieve, aspire to) over an intransitive one (sit, die).
  - subject; verb; direct object
  - doctor; treats; patient
  - patient; treated by; doctor
- Prefer powerful verbs
- Use parens as needed
- If stuck use "use"

## Stages and Investigations



## Chapter 19: Local, Anytime Steps of Controlled Evolution

### To Discover Entities

1. notice unnamed data and
2. stay **grounded** by focusing users attention on:
  - a. Nouns before verbs: *creature* and *skill* before *achieve*.
  - b. Nouns and verbs before modifiers: *achieve* before *proficiency*.
  - c. Things with mass before things that occur: *victim* before *crime*, *committee* before *meeting*; *cow* before *milking event*.
  - d. Discrete things before things with more vague or changing boundaries: *cow* before *herd*; *cow* before *milk*
  - e. Things before connections between things: *graph node* before *edge*; *airport* before *flight*.
  - f. Thing and place before thing-in-place: *chimp* and *map coordinate* before *chimp at map coordinate*; *movie* and *theater* before *show (on a date, at a time, perhaps on a screen)*
  - g. Place before connected places before connected places in time: *flight route* before *flight schedule* before *itinerary* (before *reservation* before *special meal*).
  - h. Thing before state or event or interval: *player* before *current event*; *player* before *trade*; *player* before *major-league stint*; *drama*, *theater*, and *show* before *reservation*.

### Fixing Identifiers

Under-identification  
 Over-identification  
 General misidentification  
 Extraneous Identifiers

### Seeking Descriptors

listen, ask, and study

W's: who, where, when, why, how, what, and measurement

### Promoting Attributes

Promoting a plural attribute  
 Promoting a singular attribute

### Relocating Misplaced Descriptors

Factors for Characterizing Descriptor Misplacement  
 Factor: The *Type* of Descriptor Misplacement

- Gross misplacement – The descriptor belongs in a completely unrelated entity.
- Too-fine misplacement – The descriptor belongs in a related but coarser entity. That is, the descriptor describes things that are coarser than the instances of the entity.
- Too-coarse misplacement – The descriptor belongs in a finer entity. That is, the descriptor describes things that are finer than the instances of the entity.
- Too-inclusive misplacement – The descriptor belongs in a more exclusive entity. That is, the descriptor describes only some of the instances of the entity.
- Too-exclusive misplacement – The descriptor belongs in a more inclusive entity. That is, the descriptor applies to every instance of the entity, plus some other instances.

---

## Extra Help: W's

### Who, What, Where, When, Why, How

**Tiny DB captures:**

**Who does What** -- Creature achieve Skill; Plus **How well** -- Proficiency

**Where:** point in space (place)

Creature resides in a Town (general)

Carlis resides in Philly (specific)

...

or change in place (distance)

240 miles between Mpls and Bemidji

or rate of change (speed, acceleration, ...)

**When:** point in time or duration (interval) or offset

Achievement date

Length of Achievement test

Past, current, future [planned versus actual] choices

**Precision for:**

**Who:** Creature or Type (named group with members) achieves a Skill;

Kermit or Frog Float

**Where:** Mpls or Lake&Hennepin; lamnia propria or gut

**When:** date/microsecond/eon; week/microsecond (sic), ...

**How much:** tonne/gram; units, ...

**Other Issues:** Certainty/Context/Horizon/Trace

## W COMBINATIONS

**Who does What plus Where & When**

Achievement: on a date in a place

versus

Achievements: on dates and in places

**Who plus Who**

Boss (teach, judge, befriend,...)

Neff #4 bosses Carlis

Marry: The set {Neff #4, Carlis}

**What's worth remembering**  
**"What do we mean by one of these"**

Patient: mother & fetus (feti?)

Cow's: milk, manure, tongue, ...

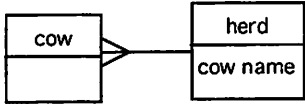
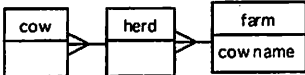
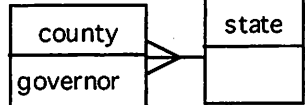
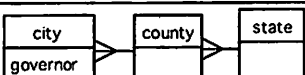
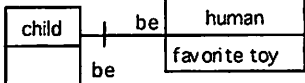
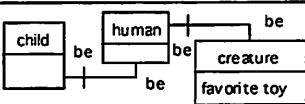
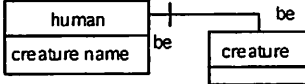
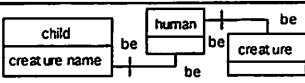
Milking machine failure

Purchasing: backorder [like a change of grade]

Metabolic Pathway(s)

*Advice: ask the users about W's. Doing so will help them decide.*

---

Type of Misplacement	Shape	Relocation	Example of Misplacement
Too coarse	One-many relationship	A descriptor moves toward the chicken foot; stopping at the related entity.	
	Chicken Feet Across	A descriptor moves toward the chicken foot, perhaps along several consecutive relationships	
	Chicken Feet In	A descriptor moves from a <i>Chicken Feet In</i> entity of a few relationships to one of more relationships.	The relocation of a descriptor from <i>Authorship</i> to <i>ReviewOfBookByPerson</i> in Fragment 11-24.
Too fine	One-many relationship	A descriptor moves away from the chicken foot to the related entity.	
	Chicken feet across	A descriptor moves away from the chicken foot, perhaps along several consecutive relationships	
	Chicken Feet In	A descriptor moves from a <i>Chicken Feet In</i> entity of many relationship to one of fewer relationships	The relocation of the <i>PersonalBest</i> attribute. (See Fragments 18-14 through 18-16.)
Too inclusive	<i>To-be</i> relationship	A descriptor moves along a to-be relationship, perhaps toward a subordinate entity.	
	<i>To-be's</i> across	A descriptor moves along one or several consecutive to-be relationships, perhaps toward subordinate entities.	
Too exclusive	<i>To-be</i> relationship	A descriptor moves along a to-be relationship, perhaps away from a subordinate entity.	
	<i>To-be's</i> across	A descriptor moves along one or several consecutive to-be relationships, perhaps away from subordinate entities.	

Factor: The *Existence or Non-Existence* of the Correct Entity

Factor: The *Path Length* from the misplacement entity to the relocation entity

Factor: The *Number of Descriptors* to be relocated

Ways to Detect Descriptor Misplacement

Relying on the names

Studying sample data

## Chapter 20: Global, Anytime Steps Of Controlled Evolution

Redrawing the diagram

Altering the overall style of an LDS

- Too many labeled relationships.  
Remember, if you and the users understand the unlabeled relationship, don't bother labeling its links.
- Too many *many-many* relationships.  
If you detect this, you are probably skipping the step of The Flow called *Flow Investigation: Seek Descriptors for Intersection Entity*.
- Too many *one-one* relationships  
If you detect this, you might be neglecting to investigate *both* links of each relationship.
- Entities with multiple identifiers  
If you find many entities with two (or more) identifiers, check your work. Are there really two (or more) entities, and one identifier applies to each of them? Is there really only one identifier, and the other identifier is merely a set of descriptors each of whose values cannot be null? If you find entities with multiple identifiers, make sure you really understand the concept of identifier. This is a truly rare construction; we have built 1000-entity LDS's in which no entity had multiple identifiers.
- Collection entities  
If you detect the one-many collection entity shape or the many-many collection entity shape, your first instinct should be a deep, abiding suspicion that you have erred. Anchor your understanding with instances. These shapes are so very rare, you can spend an entire career as a commercial data modeler without ever encountering them.

Changing the level of abstraction

Combining Multiple Short Paths

Collapsing a taxonomy

Collapsing Subordinates Out

Guidelines for increasing or decreasing abstractness

When you can and cannot use abstract shapes

## Chapter 22: Constraints

- Constraint definition requires a mature, stabilized data model
- Many candidate constraints turn out to be false
  - The provincial view of data
  - Users fail to appreciate their own flexibility
  -
- Many constraints subject a data model to premature obsolescence

Worthy constraints

Constraints and shifting the burden



## Chapter 24: Decisions: Designing a Data-Modeling Notation

### Overall Decisions

**Question: What is the purpose?**

name the memorable types of data and state identifiers to determine what we mean by *one* of these.

**Question: What concepts get modeled?**

- clutter, training, implementation details.

**Question: What are the names of the modeled concepts?**

Synonymy, Overloading, Near synonymy

**Question: Should a model include behavior?**

no.

**Question: What graphical notations should be used?**

Shift, Dogma, Unrealistic simplicity

### Decisions About Entities

type vs instance

**Question: Should an entity name characterize one instance, or many?**

one

**Question: Should there be different notations for different kinds of entity?**

no

**Question: Should each entity have an identifier?**

yes

### Decisions About Identifiers

**Question: Should all identifiers be arbitrary?**

no

**Question: How should identifiers be annotated?**

some symbol.

**Question: Can identifiers include links?**

yes

**Question: Can an entity have multiple identifiers?**

yes (but it's a rare occurrence).

**Question: Must an entity have multiple identifiers?**

no

**Question: How are multiple identifiers annotated?**

some other symbol

### Decisions About Attributes

**Question: Is there a difference between entities and attributes?**

yes

**Question: Do attributes belong on the diagram?**

yes

**Question: Do data types belong on the diagram?**

no

**Question: Do scales belong on the diagram?**

Yes but not directly

**Question: Are plural attributes allowed?**

no

**Question: Are foreign-key attributes on the model?**

no

**Question: Are type-level attributes allowed?**

no

## Decisions About Relationships

**Question:** Are all relationships binary?

yes

**Question:** Is there a difference between relationships and entities?

yes

**Question:** Can relationship have attributes?

no

**Question:** Can a relationship have links?

no

**Question:** Can a relationship have an ID? Must a relationship have an ID?

no

**Question Revisited:** Is there a difference between relationships and entities?

yes

**Question:** Do relationships have names?

yes (but rarely articulated)

**Question:** Should there be different notations for different kinds of relationships?

no

**Question:** Should relationship names be verbs?

no

**Question:** What does a relationship look like on the diagram?

a line

## Decisions About Links

**Question:** Do links have names?

yes

**Question:** Are link names on the diagram?

no

**Question:** How should a link label be annotated?

a label

**Question:** Which links get labeled?

- when needed.
- neither or both links of a relationship

**Question:** What does maximum degree look like?

chicken foot

## Decisions About Descriptors

**Question:** Should entities and descriptors share a namespace?

yes

## Decisions About Constraints

Many notation systems include the concept of constraint. The words, however, vary. Other notation designers have used these words for constraint: Rule, Business Rule, and Data Constraint.

In designing the LDS notation, we have chosen to exclude constraints. *Chapter 22: Constraints*, explains why. In that chapter, we present the questions we think are important:

- Does a candidate constraint refine the definition of a user-recognized category?
- Is a candidate constraint actually false because the users are overlooking exceptional situations?
- Is a candidate constraint false some of the time, but true at certain moments during data processing?

- Is a legitimately true constraint a function of processing – can it become false when the users change their policies and procedures?

We categorize constraints this way because it helps us distinguish worthy constraints (those deserving aggressive, continuous enforcement by an implemented system) from marginally worthy constraints (those deserving occasional enforcement) and from false constraints (those that should never be enforced).

Designers of other notations ask different questions about constraints. These questions categorize constraints according to their syntactic complexity. For example, designers of other notations can recognize these categories of constraints:

1. Minimum-degree constraints for links.  
For example, "Each FlightLeg must have an arrival airport."
2. Minimum-degree constraints for attributes.  
For example, "Each PlaneType must have a MinimumCrewSize."
3. Intra-instance, intra-attribute constraints.  
For example, "SALARY cannot exceed \$100,000."
4. Intra-instance, inter-attribute constraints.  
For example, "SALARY + BONUS cannot exceed \$130,000."
5. Intra-instance, intra-link constraints.  
For example, "No Flight can have more than five FlightLegs."
6. Intra-instance, inter-link constraints.  
For example, "FlightLeg.DepartureAirport cannot equal FlightLeg.ArrivalAirport."  
(That is, *Each FlightLeg must arrive at a different airport than it departs from.*)
7. Inter-instance, intra-descriptor constraints.  
For example, "Max(SALARY) cannot exceed Min(SALARY) \* 5." (That is, *The highest-paid employee can receive no more than five times the lowest-paid employee.*)
8. Inter-entity constraints.  
For example, "For each FlightLeg, the MinimumRunwayLength of the PlaneType of the Flight must not exceed the RunwayLength of the arrival Airport and must not exceed the RunwayLength of the departure Airport."

Generally, constraints in the low-numbered categories are easy to express on a diagram and those from the high-numbered categories are difficult to express on a diagram. It should not surprise you, then, that some notations accommodate the easily expressed constraints, but no notation accommodates the hard-to-express ones.

That's unfortunate, because there is little or no correlation between a constraint's syntactic complexity and its worthiness. For example, the following worthy constraint is somewhat complex—certainly too complex to be represented graphically on a data model diagram:

*The departure Airport of a Flight's first FlightLeg cannot be the arrival Airport of any of that Flight's FlightLegs.*

Conversely, the following easily expressed constraint is not especially worthy.  
*Each Flight must have a PlaneType.*

The constraint is not worthy because the user who reported it as a constraint had a provincial view of the data – that user had never participated in the early stages of the flight-scheduling process, when flight paths and preliminary schedules are laid out before the PlaneTypes are chosen for each Flight. This constraint is not true at all times, and deserves at best occasional enforcement. More probably, this "rule" should not be considered a constraint at all. Rather,

the system should include a built-in capability to report on all Flights that do not have a PlaneType.

**Question: Should the diagram capture minimum degree?**

no

**Question: Should the diagram capture intra-instance, intra-attribute constraints?**

no

**Question: Should the diagram capture intra-instance, inter-attribute constraints?**

no

**Question: Should the diagram capture intra-instance, intra-link constraints**

no

**Question: Should the diagram capture intra-instance, inter-descriptor constraints?**

no

**Question: Should the diagram capture triangle relationships?**

No (but it is a close call)

**Question: Should the diagram capture inter-instance, intra-descriptor constraints?**

no

**Question: Should the diagram capture inter-instance, inter-descriptor, intra-entity constraints?**

no

**Question: Should the diagram capture inter-instance, inter-entity constraints?**

no

## Summary and Final Thoughts

The choice of whether or not to express a concept on the model comes down to a judgment about whether the concept is valuable enough semantically to tip the balance between content and complexity. A concept must suit a specified audience and its purposes. It must be named reasonably. Its notation must enhance, not impede, your work.

From our point of view, enriching the notation is easy, but we have resisted doing so. Such additions should occur to LDS only if they are likely to yield better data models. A rich notation might theoretically accommodate more than a spare notation, but if the richness of the notation interferes with controlled evolution, it can effectively yield less accurate models. All the decisions we have made about the LDS notation and technique are intended to honor what we know about humans, about data, about software technology, and about how these three things interact. Re-read Chapters 1 and 2 and see if you agree with our decisions.

## Ch 25 Relations

### Mapping an LDS to relations

**Goal: Look at an LDS and see relations and vice versa**

344

SIMPLE, almost no choices

1 entity  $\rightleftharpoons$  1 relation1 attribute descriptor  $\rightleftharpoons$  1 columnMany valued link descriptor  $\rightleftharpoons$  nothingOne valued link descriptor  $\rightleftharpoons$  column or columns corresponding to the describing entity's identifying set of descriptors*[So, a link can map to one or several columns. Perhaps a recursive search]*Column corresponds to an identifying attribute descriptor  $\rightleftharpoons$  primary key columnColumn corresponds to an identifying link descriptor  $\rightleftharpoons$  primary key column-----  
A "1 and 1" relationship [chicken foot –less] mapping requires a choice

Map either or both one valued links

$$\begin{matrix} & n \\ & \uparrow \\ \text{So, 3 choices} \end{matrix}$$

-----

Not so simple

**Split**

Horizontal: all columns with some rows and all column with other rows

Vertical: Id plus some columns and id plus other columns

Partition vs Copy

**Denormalize**

Trade lower retrieval cost for more space and update costs

**Absorb**

Entity of one valued link

Entity of many valued link [COBOL record; object, ...]

**Why?**