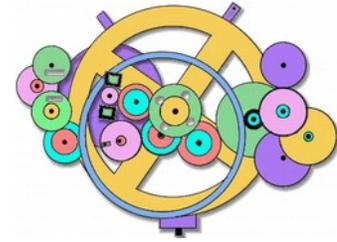


# Antikythera Publications



---

## DATABASE DESIGN NOTE SERIES

---

Relational Database Design  
<http://www.AntikytheraPubs.com>  
jnhummel @ AntikytheraPubs.com

### The Maltese Runway - Part 1 of 2: Triage of a Data Model Segment

Prepared by: J. N. Hummel and F. Oberle; with apologies to Dashiell Hammett

The aim of Antikythera Publications' training material is to translate the teachings of historic data organization experts to modern database design and counter common assumptions that such practices became irrelevant with the arrival of technology.

Of course, we also recommend locating and studying existing modeling solutions for similar situations, with the caveat that each "discovered solution" must be understood and carefully evaluated before accepting it at face value.

We were recently presented with such a discovered solution that closely paralleled what an inexperienced modeler was attempting to do. The modeler, however, was unable to understand the logic of the published solution, and requested some hints.

Since several logical errors in the solution are ones that we commonly encounter, it seemed that this might provide a useful example of how classical processes could expose these errors and suggest a better approach to designing the data model.

Revised for public distribution: 15 March 2015

See page 16 for information on other material from Antikythera Publications.

PART 1

Copyright © 2015 by the Authors

Permission is granted to distribute unaltered copies of this document, so long as this is not done for recompense or similar commercial purposes. Be aware that the views expressed in this document, while congruent with mathematics and logic, do not reflect typical IT wisdom. While we aren't disturbed by this, you may be. We're willing to let the arguments presented here stand or fall on their own merits.

---

**THIS PAGE IS INTENTIONALLY BLANK**

# Database Design Note Series – The Maltese Runway

## INTRODUCTION TO PART 1

The block below contains a verbatim copy of the “discovered” solution, found on page 26 of a freely available pdf.<sup>1</sup> The author had introduced Exclusivity Arcs on the previous page, then presented this example to illustrate their use.

**Consider:**

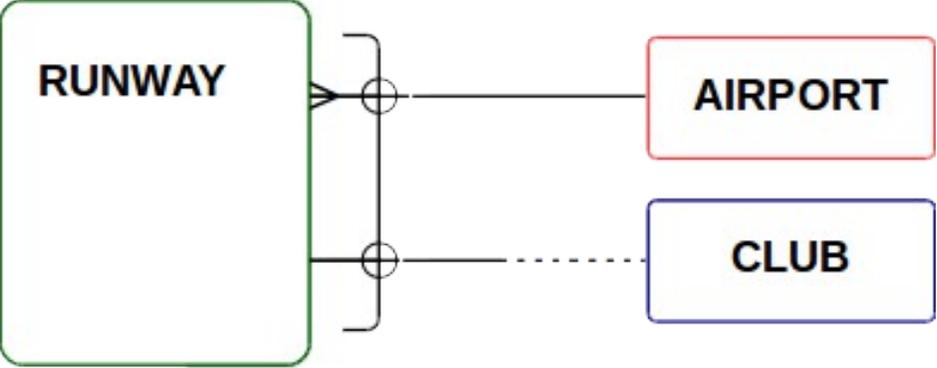
A Database is required to store a list of runways. A runway might be owned by an airport, such as Malta International Airport. An airport would own at least one runway but may own more. A runway may instead be owned by a club, such as a Motor Racing Club that owns the runway in Hal-Far, or the Remote Control club that owns the runway in Ta' Qali. Lets say that a law in our country imposes that a club is allowed to own only one runway. Lets also say that no runway may be abandoned. [<sup>2</sup> – our footnote, not the original author's ]

**N.B.** We cannot use subtypes since a RUNWAY is not an AIRPORT, nor is it a CLUB. Also an airport may have more than one runway whilst a club may have only one runway. Therefore the following is wrong.



This scenario therefore cannot be resolved by using subtypes. But we need the mutually exclusive rule. Therefore we use the exclusive relationship arc.

**Solution:**



Frankie Inguanez – 2012 Page 26 of 31

A few things should be clearly stated before commenting on this scenario, however:

- The primary purpose of the document from which the above example was extracted was evidently to describe

1 *Entity Relationship Modelling*, by Frankie Inguanez, is available for download at the time of this writing from <http://frankieinguanez.wordpress.com/2012/01/16/erd-modelling-using-barkers-notation/>.

2 Hal-Far is not a fictitious location, but was the first airfield built on Malta by the British Royal Air Force during World War II. It now serves as a drag racing strip. Ta' Qali, also quite real, is the location of Malta's national stadium.

what the author referred to as the “Barker Notation” for Entity-Relationship Diagrams and, in our view, he succeeded quite well in that regard. In particular, his explanation of how the relationships are graphically represented is arguably more clear than what Barker himself provided in his original CASE\*Method book.<sup>3</sup> So, admittedly, we’re addressing what may be considered to be only incidental aspects of the document. Nonetheless, we feel that these comments can help other analysts, or at least provoke some thought.

- Exclusivity Arcs are important and even necessary in modeling the “real world,” and the particular use of the Exclusivity Arc illustrated by the author is commonly encountered in database design and modeling literature. Many such examples<sup>4</sup>, however, are demonstrably illogical, and therefore incorrect. Furthermore, such misuse almost always leads to system constipation<sup>5</sup> down the road. Coupled with a misuse of the term “subtype,” the presence of the exclusivity arc reinforced our decision to use the author’s scenario as a case study of sorts.
- Finally, since the original version of this document was prepared for a modeler known to be familiar with the key lessons presented in *Business Database Triage*, extended definitions and explanations of the terms Class, Set, Type, Predicate, Proposition, Normalization, Normalized Proposition, and so forth<sup>6</sup> are not provided, nor do we discuss why government regulations and business requirements (to give common examples) are inherently Arbitrary and should only be of secondary concern when developing the *structures* of database tables.

However important those concepts may be to logical database design though, a thorough understanding of these topics is not critical to following along with the present analysis. We have attempted to accommodate those who are unfamiliar with such concepts by doing the following:

- Commenting briefly on some terms where this may be warranted for a wider audience.
- Utilizing the original publicly available “discovered” scenario rather than the proprietary one for which the modeler was attempting to find a solution. We have, though, taken the liberty of incorporating an interesting additional requirement the modeler had, but that will be introduced at the appropriate point.

We’ll begin by pointing out the most significant problems that will result from the “solution” presented. Following that, we’ll examine the “requirements” the author provided, identify the most obvious of the “missing” information that should be gathered, then proceed to restate these statements in a more formal way that that has been proven more suitable for the organization of data over the centuries. Finally, we’ll see where such a more formal approach can lead us.

## DEFICIENCIES IN THE “REQUIREMENTS”

“Requirements” is a term that normally covers a much broader universe than is appropriate here. When designing a database, however, the best interpretation of the term is “a collection of mostly Normalized Propositions that define the Entities of interest to our business and the possible legitimate Associations between any pairs of those Entities.” Because of our focus on designing the data structures, we will not be considering any process requirements.

The author presents us with some statements that come quite close to such a limited definition of requirements. We’ll begin taking each of them in turn, taking care to highlight each **Noun** and Verb (always recommended) we encounter:

- ❶ “A **Runway** might be owned by an **Airport**.”

This looks like a legitimate (although certainly not normalized) Proposition, but there are several red flags:

- 3 Richard Barker; “CASE\*Method – Entity Relationship Modelling”; Addison-Wesley Publishing Company; Copyright © 1990 Oracle Corporation UK Limited; ISBN 0-201-41696-4. This notation is also referred to as the “Oracle” notation.
- 4 ... including this one. Another particularly interesting one will be mentioned in footnote 14.
- 5 ... not, of course, an accepted term in I/T communities (yet). Generally “system constipation” refers to the inability of any combination of software and database designs to be easily extensible to support changing business needs or new opportunities.
- 6 In other words, this paper supplements several other publications we provide that the modeler asking the questions was known to be familiar with. But it is incumbent on anyone with aspirations of designing corporate data models responsibly to become fairly intimate with these concepts. As with much of the Science of data organization (and yes, it is a Science, although if your only exposure was a typical business database or application, you might never suspect that), the works of Aristotle, Lewis Carroll, George Boole and others on these subjects are quite helpful if you are unfamiliar with our own publications.

- An **Airport** is not a legal entity and therefore cannot “own” anything. Before arguing, please read the next four comments about this statement. Therefore, a **Runway** cannot be owned by an **Airport**.
- Because the statement isn’t True, it isn’t suitable for use in a database design, and the association it describes needs to be unraveled and restated more precisely before proceeding to model it correctly.
- The statement is in the passive voice. Any association between two entities represents and is described by at least two propositions. The most common (and likely only) situation where the passive voice is acceptable is as a corollary to certain types of partner proposition – where the corollary simply cannot be stated in the active voice. Although this limitation can itself be a useful clue to the association’s type (e.g. is it a relationship or something else) when analyzing a model, we don’t want to get ahead of ourselves here. Active voice should always be the first choice for stating a Proposition.<sup>7</sup>
- The modal word “might” is generally not used in formal propositions, since it suggests a suspicion or guess rather than an acknowledgment of a possibility or permission. Instead, the word “may” is prescribed.<sup>8</sup>
- For the moment, we’ll ignore the fact that we haven’t defined either **Airport** or **Runway**. The failure to define **Airport** (are we talking here about the buildings and infrastructure, or about legal entities such as the management company or some other entity?) is quite likely why the subtle imprecision of this statement slips by unnoticed.
- ❷ “An **Airport** would own at least one **Runway** but may own more.”
  - Here we have what appears to be the active voice corollary to the previous statement. With experience, you’ll learn that stating the active voice version of any pair of propositions first is a useful habit that will keep you more firmly grounded in reality.
  - Of course, as stated above, an **Airport**, not being a legal entity, cannot own a **Runway**. It can – and most likely must – *have* at least one **Runway**, but that is an altogether different proposition.
  - The modal word “would” is also never used in formal propositions; see footnote 8.
  - Another issue with statement ❷ is that when normalizing a Proposition, all quantifiers for an object should be consolidated. Therefore, the separate clause “but may own more” must be combined with “at least one” so the whole statement reads: An **Airport** may own one or more **Runways**. Nonetheless, the statement, reflecting an impossibility, is still false, and therefore still requires some reworking.
- ❸ “A **Runway** may *instead be owned* by a **Club**.”
  - Once again, the Proposition is in the passive voice. When comparing this with the next proposition, which appears to be its active voice corollary, it is easy to spot the benefits of stating the active voice version first, because the passive version gives no hint of the ownership restriction on a **Club**.

Observant readers and experienced modelers will perhaps chime in with a suggestion that the prevalence of the passive voice is because **Runway** is the central entity of interest in the author’s scenario. This may be, of course, but the fact that no active voice Proposition can be formed with **Runway** as the subject lets us know that – in reality – it is not suitable as the primary focus of a logical design.

The author’s use of the word “instead” implies an arbitrary limitation that a **Runway** cannot be owned by both an **Airport** and a **Club**. By extension, we could legitimately infer that a **Runway** can have only one owner. Inferences must, of course, be recognized, validated and made specific in any good design.

---

7 As your old grammar teacher might have said, passive voice is rather wimpy and causes us to begin our analysis looking “backwards.” The Subject of interest and its Object have been exchanged – not an auspicious beginning for a logical analysis.

8 The difference between these two modals is, as with much English grammar, rather subtle; if you wish to pursue these differences further, that’s certainly encouraged, but it is considered way beyond the scope of this document – particularly because such distinctions are handled differently depending on which language you are speaking. Quit quibbling and pay attention.

- ④ “A **Club** is allowed to own only one **Runway**.”
  - It is critically important when creating database *structures* that the designer distinguish between those things (and associations) that are Natural (always and inherently true) and those that are Arbitrary (dictated by some human entity and therefore subject to change). Arbitrary things and associations include laws, government and corporate policies as well as industry standards and conventions. As designers it is, of course, our responsibility to accommodate all of these in our model, but as described in *Business Database Triage*, the proper techniques for handling the Natural and the Arbitrary need to be quite different.

The word “allowed” is an obvious clue that this is not a statement of some natural fact, but rather a statement of an arbitrary restriction created by some human entity. This is reinforced by the lead-in of the sentence: “Let’s say that a law in our country imposes that ... ”
- ⑥ “No **Runway** may be abandoned.”
  - The statement is given in the negative. Such requirements need to be accommodated, but negative propositions are unsuitable for initial database designs. The many tactics for removing negation from a proposition are out of scope here, but we’ll show how this particular one can be handled in a later section.
  - While the word “abandoned” seems to be an adjective in this statement, suggesting perhaps that it may be a status or condition of some sort, the initial negation suggests that such a status would be impossible, making this statement a paradox of sorts. The word “abandoned” cannot therefore be an adjective.
  - If the word “abandoned” was *intended* as an adjective, however, the statement is obviously untrue; it isn’t difficult to locate weed infested, crumbling runways virtually anywhere, and all we need to do is to locate just one to prove that the statement isn’t true. When such an obviously untrue statement is encountered, it is quite easy to infer that it may be a colloquialism, so it’s equally likely that the real meaning of the statement is evident *to a human*. The danger comes when such statements are untrue, but not obviously so.

This is one of the reasons why, for database designs, the propositions must always be converted to ones that are categorically true – or at least not false. This statement, while providing a clue (and one that certainly shouldn’t be ignored), needs to be restated more precisely as a Normalized Proposition<sup>9</sup> or you will fall into the trap of literally interpreting and modeling a colloquialism – never a good practice.

  - Any attempt to formulate an active voice corollary to this statement – something that should be done – leads to something like “An **Owner** must not abandon a **Runway**.” The word “abandoned” therefore clearly carries the sense of a verb. More importantly, the concept of an **Owner** has been introduced.

The final deficiency in this set of “requirements” is the lack of definitions for the various key terms – both nouns and verbs – that are used. In “Transforming Statements into Propositions” on page 8, we’ll attempt to begin addressing these deficiencies.

These observations about requirements deficiencies, while possibly tedious, are an important part of performing triage on any actual or proposed database designs. At the risk of losing the remaining half of readers who began on page one, however, we’ll stop the critique of the requirements and move on to the two solutions the author presented.

### AUTHOR’S REJECTED SOLUTION ONE

The author correctly rejects this, saying that the “scenario ... cannot be resolved by using subtypes.” For simplicity, we could simply ignore this solution, but there are benefits to exploring it further in order to see *why* it is inappropriate.

The primary concern we have is that the use of “subtypes” was considered a common enough potential approach in the first place that the author chose to point it out – and attempting to understand why. The author after all clearly states that **Airport** and **Club** are not types or sub-types of **Runway**. A “type” (“sub” or not) is based on shared Attribute

---

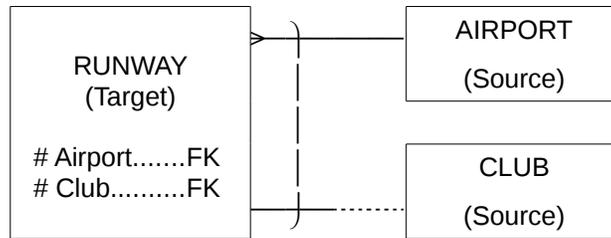
<sup>9</sup> You will recall from other lessons that, among other requirements, a Normalized Proposition used for the design of database structures must always be categorically True.

Values, not on shared Attributes after all, and recognizing this distinction is quite crucial to achieving a logically defensible data model.<sup>10</sup>

Once again, such a discussion is a distraction here, but in simple terms – is an **Airport** even a type of **Runway**, much less a subtype? Of course it's not! It is even more far-fetched to suggest that a **Club** is a type or subtype of **Runway**.

### AUTHOR'S PROPOSED SOLUTION TWO

So, what's wrong with the author's presumably final solution to the modeling quandary? On page 27 of his document, he establishes **Runway** as the target (by a parallel example), with **Airport** and **Club** as sources. The target table **Runway** includes one column for each source (**Airport** and **Club**), with each column being constrained as a foreign key to the primary key of its respective source. The solution he presents is illustrated in the following diagram.



The amazing polymath Lewis Carroll wrote extensively about logical structures. Were he still around, my suspicion is that he would be both gleefully amused and uncomfortably appalled by how easy we can be sucked down the old rabbit hole (his literary invention) by this sort of diagram. Ignoring for the moment that the Solution Two is simply a slightly different notation for the rejected Solution One, consider what the model implies:

- We have a **Runway** table with a column for each possible “type.”<sup>11</sup> Even if we consider these to be “ownership types” rather than “runway types,” the problem remains, as we shall see below.
- Each of these columns is constrained by a foreign key – except when it's not convenient for the developer!

Having Foreign Keys that permit NULL values is an extremely questionable practice, although often permitted by modeling tools and RDBMS implementations that need to cater to both ends of the bell curve.

- We have a not-all-that-subtle First Normal Form violation here (an owner, another owner, and so forth), and having mutually exclusive Foreign Keys doesn't really disguise that. Obfuscation isn't correction.

And if the counter-argument is made that – really – only one of these is an active foreign key *at any given time*, then we are forced to accept the idea that the structure of our database is effectively different on a row-by-row basis. Even without understanding the logical and theoretical problems that *will cause*, such a conclusion should certainly engender some intuitive level of discomfort even in those with limited experience.<sup>12</sup>

A mutually exclusive type, by the way, is typically implemented as a single column, constrained by a foreign key to a table of acceptable types. This seems off-topic, since it doesn't address the ownership question we're attempting to model, but we'll get to that later. A “type” of anything is, by definition, an example of the thing in question – i.e. a “type” of Widget is always itself a Widget.

- It's difficult to make the case that some other class of entity (a City, for example) might never own a runway, or that there might never be multiple ownership of the same **Runway**. This design, however, has arbitrarily re-

<sup>10</sup> Those with no previous exposure to Logic or formal database design using Predicate Logic need to be aware that, unlike in procedural circles (such as OO programming), there are important differences in the definitions of Class, Set, and Type. Aristotle's *Organon* or *Business Database Triage* each give more detailed explanation of these distinctions and their importance.

<sup>11</sup> Type, remember, is based on attribute values, not attributes themselves. Adding “types” by adding attributes is not logical!

<sup>12</sup> For those with a theoretical bent, the identical Predicate should be applicable to every row in a Relational Database; otherwise it cannot be said to be always True – and if it is not always True, it is not a normalized Proposition (or Table).

stricted ownership to a single entity at a time.<sup>13</sup> Even with no knowledge of formal logical analysis, most folks might suspect that such possibilities can eventually exist. Somewhere. Sometime. It's true enough that some group ownership might be handled through a single partnership entity created just for purchasing the runway, but that certainly isn't a given. The table structure(s) of any *valid* design must not preclude the possibility of sharing ownership, even if the implementation temporarily precludes it through some easily reversible means.

- Following the existing pattern, we would need to add a new column to the **Runway** table for each category, type, class, or set (and it makes no difference what term you accept as correct here) who might own a runway. For those familiar with Logic, this is an example of how *reductio-ad-absurdum* applies to database design.

While this is not the whole story of the design's shortcomings, what we've covered is certainly sufficient to reject this model out of hand. The fact that there are unnecessary impediments to extension makes it demonstrably incorrect from a logical standpoint – and therefore inextensible from a development standpoint (and therefore also from a business perspective and so on).

Unfortunately it is startling – and not a little depressing – how easy it is to locate virtually identical models that confuse classes, sets, and types in a variety of publications, and often utilize exclusivity arcs in a totally inappropriate fashion.<sup>14</sup> So, once again, we'd like to stress that criticism of this model isn't aimed at the particular author whose example we are using, and whose essay just happened to show up in our novice modeler's search.

I/T folks are insular, and have proven themselves quite capable of attempting to reinvent the wheel with no knowledge whatever of the earlier lessons learned about wheels outside their little world. So let's proceed, and see where a logical evaluation leads us, and what other questions can be identified and, of course, answered or deliberately ignored.<sup>15</sup>

## TRANSFORMING STATEMENTS INTO PROPOSITIONS

Recall that the original statements provided by the author were:

- ❶ “A **Runway** might be owned by an **Airport**.”
- ❷ “An **Airport** would own at least one **Runway** but may own more.”
- ❸ “A **Runway** may instead be owned by a **Club**.”
- ❹ “A **Club** is allowed to own only one **Runway**.”
- ❺ “No **Runway** may be abandoned.”

As mentioned earlier, we are adding one more requirement to meet the needs of the modeler who inquired about the “discovered solution” we are discussing. Restating the modeler's requirement in terms of this example would result in something like:

- ❻ “Proportional sharing of **Ownership** of any **Asset** must be supported.”

Generally, regardless of specific requirements, any well-designed data structures must never *preclude* such ownership sharing but, in this case, we will need to explicitly include it in our model.

## SIMPLIFIED ANALYSIS OF THE STATEMENTS

The use of Normalized Propositions is key to the design of a logically defensible data model. Although generally always an iterative process, the steps for assembling such Propositions for a logical database design, stated simply, are:

1. Identify the various “Things of Interest” in our business. Determine if each “Thing of Interest” qualifies as a

---

13 A cardinal rule when designing database *structures* is that while they need not *completely* reflect reality, they must never *conflict* with reality. This is similar to Hippocrates' admonition to medical folks “to do good or to do no harm.” (the actual translation)

14 A typical example is at [http://sd271.k12.id.us/lchs/faculty/bkeylon/Oracle/database\\_design/section12/dd\\_s12\\_104.pdf](http://sd271.k12.id.us/lchs/faculty/bkeylon/Oracle/database_design/section12/dd_s12_104.pdf), dissection of which can be a useful exercise. This seems to have the imprimatur of Oracle, but other vendors offer similar examples.

15 Ignoring or choosing not to follow through on certain questions can be a legitimate response depending on circumstances. Not recognizing that there are unanswered questions, however, must be considered a poor design practice.

Class or is merely a Set.<sup>16</sup> As Analysts, we must establish the validity of each Entity-Set possibly related to our model, which involves examining every Noun in any discussions of our “universe of discourse.”

2. Confirm that each Class or Set we identify is distinguishable from each other Class or Set, and that each member of any Class is itself uniquely identifiable (or can be made to be so) from other members of that Class.
3. Discover any Associations between each pair of “Things” we’ve identified. Each Association should be stated as a pair of Propositions in order to confirm that it is precisely understood. Eventually, we will Normalize the Propositions – at least those defining Associations between “Things” we have identified as Classes.
4. For each Association, determine whether it will likely be implemented as a Relationship between two “Things” (or between the “Thing” and itself) or defined as an Attribute or even an Attribute Value.
5. (Optional but recommended) Create a graphical model of the Things and Associations to more effectively clarify the broader context of the associations. This may take the form of Euler or Venn diagrams or, as is more frequently done, some form of Entity-Relationship Diagram (ERD).

The following rather free-form comments will follow *one* example of a path this set of steps might take:

1. Identify the various “Things of Interest,” and determine if they qualify as classes or merely sets. For a Normalized Proposition (and a normalized database), any “Thing of Interest” should be identified by a Common Noun.

From the original model on page 3, we identified the following Nouns: **Runway**, **Airport**, and **Club**. The noun **Owner** was added during the cursory analysis given above. From the additional requirement (see 6 above), **Ownership** and **Asset** were added. All six of these are Common Nouns.

Additionally, we saw three Proper Nouns (**Malta**, **Hal-Far**, and **Ta’ Qali**) in the original model, from which we can predicate (infer) the existence of at least one more common noun that we’ll call **Location** for the moment.

Adjectives may expose Nouns as well, but none have been specified, so we can ignore that for the moment.

Whether any or all of these common nouns are named appropriately, or are actually of interest to us for our model will need to be determined as the analysis proceeds. For simplicity, we won’t discuss this process.

**Airport:** Obviously, we need to define what is meant by the word “Airport.” Does an **Airport** include the property or land on which it’s located? In this example, we’ve seen that the term **Airport** must certainly have been used to refer to a corporate entity rather than a physical facility. Does the word refer to the facility itself as well, thereby introducing a lack of clarity in the problem statement? Does an “**Airport**” even require a facility? Context is important, but sometimes difficult to pin down.

We’ve already seen that, while **Airport** is an actual entity (proven by a peculiar adjunct – the unique set of attributes that defines it), it isn’t actually relevant to the discussion of **Runway** ownership.

Even this isn’t the whole truth, however, as it is conceivable that a Government body (e.g. a City) might actually own the Runway while a separate entity owns and/or operates the Airport facility.

**Asset:** A generalization used by the modeler who asked for this commentary – note that this refers to physical assets for which ownership is an issue. Unfortunately, the term **Asset** is also commonly used for things such as “personnel” and “financial holdings,” suggesting we might spend a bit more time to ensure the naming is unambiguous within a particular environment. Also observe that **Runway** is a specific example of such an **Asset** definition.

**Club:** We need to note here that “A Club is allowed to own only one Runway” is an entirely arbitrary restriction (as are most man-made laws, business requirements, and so forth). There are other obvious questions that must be asked: For example is this arbitrary restriction valid everywhere? Or is it only valid on Malta? And how is a “Club” defined from the perspective of the database users or the lawmakers?

---

<sup>16</sup> Distinctions among Class and Set are out of scope here, but are discussed in Aristotle’s *Categories* and summarized in *Business Database Triage* (see page 16). Such distinctions, generally not relevant in Programming, are important in Database Design.

**Location:** We can implement no specific solution without answering: Is the ownership restriction only applicable in a single area (e.g. within a town, zone, state) and the related “are we (or might we ever be) covering a broader geographic or geopolitical area than just Malta with the data model?”

**Owner:** This is a more general term for what the original author called **Airport**. It would seem that some type of **Organization** is referred to but, in the real world, certainly any **Party** (of which **Organization** is a sub-class) can own an Asset. The data structures we produce must never preclude that possibility.

**Runway:** A **Runway**, or indeed any **Asset**, can be owned by one or more Organizations since Club and Company have that in common (we will ignore cardinality for the moment – identification of the entities and fundamental associations between them must be settled before we can reliably address cardinality issues), but a good design must not *preclude* other classes of Party from being Owners, even if it doesn’t explicitly support such ownership initially.

And, of course, we need to settle on a definition of **Runway** irrespective of any super-class to which we believe it belongs. Does the word refer to a specific isolated **Runway**, or to an interconnected system of runways and taxiways used with them? Does it include helicopter landing pads? Or, as suggested earlier, does it refer more to the land (real estate) on which the **Runway** has been constructed.

2. Confirm that each Class or Set we identify is distinguishable from each other Class or Set and how (i.e. what are the essential attributes of each and/or what is the peculiar adjunct of each?) We will skip this step for the sake of brevity in this paper, although some obvious attributes are listed in the diagrams that appear later.
3. Discover any associations between each pair of Things (so far, these are **Runway**, **Airport**, **Club**, **Owner**, and **Location**) and state these associations as pairs of complementary Normalized Propositions. Just as identification and classification of any nouns we encounter is important to forming the subjects and possibly objects of Propositions, identification of the verbs helps us form the appropriate predicates, which in turn define the associations.

The Verbs we’ve encountered are: **Own**, **Allow**, and **Abandon**. Remember that, although the author used the term “abandoned,” which seemed to imply an adjective, we determined that, logically, it actually represented a verb.<sup>17</sup> A little consideration – assumed for simplicity – suggests that the verbs **Have** and **Use** should be added.

**Abandon:** Upon further analysis, the original statement “No Runway may be abandoned” more likely means that no owner entity may abandon a property. Any **Asset**, particularly real property – such as a **Runway** – must have at least one **Owner** – owner in the author’s example being a **Party** that has legal liability for the Runway: the responsibility for paying any applicable taxes and providing some minimal level of maintenance and upkeep, and anything else an imaginative lawyer or politician might dream up.

**Allow:** As mentioned earlier, use of this verb clearly implies that any suggested requirement is arbitrary and, while enforcement in our system is required, human impositions must not be implemented in the form of table structures that would need to be changed if the “rule” itself was eventually changed.

**Own:** Is “own” the precise word intended? Does this cover exclusive usage rights (a form of “ownership” in a colloquial sense) for example? See comments under the word **Have** below.

Is “ownership” exclusive? Can a Club, for instance, own 50% of one **Runway** and 50% of another **Runway** without violating the arbitrary ownership restriction of “one **Runway**” (i.e. 100%)?

**Have:** An Airport must certainly “have” (although not necessarily own) one or more Runways. Otherwise, what right does it have to call itself an Airport? See the earlier comments for **Own** above.

**Use:** Use of or access to at least one Runway is an essential attribute of an Airport. Even exclusive use of a Runway does not, however, imply ownership of a Runway.

When normalizing Propositions related to the previous three verbs (**Own**, **Have**, and **Use**), we run into a problem. The verbs “own” and “use” are not acceptable verbs for a Normalized Predicate.<sup>18</sup> Therefore, we need to restate

---

<sup>17</sup> See the last bullet point under Deficiencies in the “Requirements” on page 6.

this – attempting to see if there is a noun that can be used will help. With the verb “own,” for instance, the noun “ownership” comes to mind, as does the more solid noun “contract” - the “thing” that defines ownership.

Darned inconvenient, that limitation on “approved” verbs! After all, what did Aristotle, Boole, Carroll and those old fogies know about computers anyway? But just on a whim, let’s see where that leads us.

Ownership (**Own**), Possession (**Have**), and Use (or use of) are different concepts as mentioned in the next few entries. When preparing a suitable model for the data, a modeler must understand the semantic differences to determine which are required.

4. Given the number and significance of the unanswered questions exposed in the simplified analysis described
5. above, it is obviously impossible to discuss the appropriate Associations and Relationships in any but very general terms, resulting of course in very general and quite simplified model diagrams. Nonetheless, we can get a general idea how a logic-based model might look by proceeding just a bit further.

### MOVING TO (MOSTLY) NORMALIZED PROPOSITIONS AND DIAGRAMS

Acknowledging that, because of the unanswered questions exposed in the cursory examination outlined above, we cannot produce a logically defensible database design from the information at hand, we can still proceed with a simplified overview to illustrate the process. We begin by examining the statements we do have – along with a few inferences that seem likely to be true – and convert as many as possible into Propositions. For the Propositions that can be Normalized, we’ll also illustrate each as a very simple Entity-Relationship Diagram in the right column.

Propositions (Normalized pairs are in <b>SMALL CAPS</b> )	Proposition Pairs in ERD Form
<p>† A Runway is an Asset<sup>19</sup></p> <p>α † An Airport is an Asset</p> <p>† An Airport must have (use of) one or more Runways</p>	(Not modeled in this exercise)
<p>β † An Asset has (must have)<sup>20</sup> one or more Owners</p>	... will be addressed below
<p>† A Club may own (one) Runway</p> <p>γ ... so, generally:</p> <p>† A Club may own one or more Assets</p>	<p>Arbitrary Restriction !</p> <p>Structure must reflect this !</p>
<p>F An Airport may own a Runway</p> <p>... not possible (a non-Party can’t own anything), but:</p> <p>δ † An Airport Corporation may own many Runways</p> <p>... so, generally:</p> <p>† An Airport Corporation may own one or more Assets</p>	<p>False (see earlier discussion)</p> <p>(a Possessive Proposition)<sup>21</sup></p> <p>(another Possessive Proposition)</p>
<p>† A Club is a Party</p> <p>† An Airport Corporation is a Party</p> <p>ε ... therefore, when considered with δ, suggest:</p> <p>{ † A Party may own one or more Assets</p> <p>{ † An Asset must be owned by one or more Parties</p>	<p>(a Taxonomic Proposition)</p> <p>(another Taxonomic Proposition)</p> <p>(another Possessive Proposition)</p> <p>A revision of Proposition β</p>

18 Normalization, including an introduction to normalization of Propositions, is introduced in *Business Database Triage* and covered in more detail in *Business Database Design*. Both books are discussed on page 16.

19 This notation for Propositions is described in *Business Database Triage* although it should be reasonably self-explanatory. In simple terms, “†” indicates a true proposition, “F” a false proposition, and “?” an incomplete proposition (i.e. insufficient to draw any conclusion about its truth or falsehood).

20 Interpretation of logically formed Propositions is discussed in *Business Database Triage*, and will not be repeated in this paper.

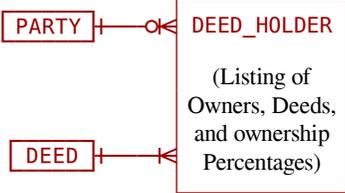
21 Various categories of Proposition used in logical data organization are discussed in *Business Database Design*. Briefly, for those mentioned here, a Possessive Proposition proposes some form of ownership or simply the possession of a particular characteristic, and a Taxonomic Proposition proposes the position of a Class in an inheritance (or similar) hierarchy.

Propositions (Normalized pairs are in <b>SMALL CAPS</b> )	Proposition Pairs in ERD Form
<p>ζ Reviewing Propositions related to the Party class<sup>22</sup>,</p> <ul style="list-style-type: none"> <li>{ T A PARTY IS AN ORGANIZATION OR A PERSON</li> <li>{ T AN ORGANIZATION IS (ALWAYS) A PARTY</li> <li>{ T A PARTY IS AN ORGANIZATION OR A PERSON</li> <li>{ T A PERSON IS (ALWAYS) A PARTY</li> </ul>	<p>A Party must be one or the other!</p>
<p>So, generally, it will be true that:</p> <ul style="list-style-type: none"> <li>T A Party may own one or more Assets</li> </ul> <p>η “Own” is not an acceptable verb for use as a Predicate in a Normalized Proposition. Two logical conversions, Taxonomic (θ) and Possessive (ι) are possible:</p>	<p>A Deed Holder must be a Party (or any sub-Class) unless either the Class (e.g. Club) or an individual member is limited by some arbitrary rules. See text.</p>
<ul style="list-style-type: none"> <li>θ T A Party may be an Owner of one or more Assets</li> <li>ι T A Party may have Ownership of one or more Assets</li> </ul> <p>... from these two, we need to choose the best option:</p>	<p>(another Taxonomic Proposition) (another Possessive Proposition)</p>
<p>λ Evaluation of “Owner” as a taxonomic Class fails, since “Owner” has no Essential Attributes not present in any potential intersecting class such as Club, Person, etc.</p>	
<p>μ Evaluation of “Ownership” as a Class succeeds, since it has Essential Attributes beyond those of any Class having “Ownership,” such as “Ownership Percent,” etc.</p>	<p>Note that Accidental “Ownership” Attributes like “Purchase Price,” etc. are possible as well.</p>
<p>ν Although “Ownership” qualifies as a Class, it is always more desirable – if possible – to have a tangible element as the object of a Possessive Proposition.</p>	<p>Data, representing “facts” can coexist with intangibles, but only somewhat uncomfortably.</p>
<p>ξ T Ownership is represented by a Deed ... so we can model a Deed-Asset relationship with the following pair of simple Normalized Propositions:</p>	<p>This might also be a Contract, Title, or some other entity depending on the situation.</p>
<ul style="list-style-type: none"> <li>{ T A DEED MUST REFERENCE ONE OR MORE ASSETS</li> <li>{ T AN ASSET HAS (MUST BE ON) ONLY ONE DEED</li> </ul> <p>ο ... but while the Proposition/diagram itself is normalized, it doesn’t account for temporal concerns<sup>23</sup>. Therefore, we would limit the Propositions as follows:</p>	<p>... or, for some Asset types, the following may be True:</p>
<p>π ... At any given time:</p> <ul style="list-style-type: none"> <li>{ T A DEED MUST REFERENCE ONE OR MORE ASSETS</li> <li>{ T A DEED’S COMPOSITION HAS ONLY ONE DEED</li> </ul> <p>ρ ... and, at any given time:</p> <ul style="list-style-type: none"> <li>{ T AN ASSET HAS (APPEARS ON) ONLY ONE DEED*</li> <li>{ T A DEED’S COMPOSITION HAS ONE OR MORE ASSETS</li> </ul> <p>*... assuming it requires a Deed at all (see Part 2)</p>	<p>Similar to a BOM<sup>24</sup> or Order-Line Item structure.</p>
<p>σ Now we can revisit and rework the proposition pair first proposed in step ε above by restating the Party-Deed relationship with the following refined Proposition pair:</p>	

22 The Party-Organization-Person Taxonomic hierarchy is discussed in *Business Database Triage*, and will not be repeated in this paper. See footnote 27 as well.

23 We could easily implement this by having a foreign key reference to a specific deed on each row in the Asset table, but that limits us from tracking the asset’s appearance on multiple deeds over the course of its life (as it’s bought and sold). For this exercise we will assume implementation of a more flexible model, with attributes such as Deed id, Asset id, and Deed Date.

24 Bill of Materials. Most good data modeling texts describe composition structures like BOMs and Order-Line Item Pairs.

Propositions (Normalized pairs are in <b>SMALL CAPS</b> )	Proposition Pairs in ERD Form
<p><b>τ</b> Since there may be shared ownership of multiple items,</p> <ul style="list-style-type: none"> <li><b>τ</b> Multiple Parties may (together) have multiple Deeds</li> <li><b>τ</b> Any Deed must be held by one or multiple Parties</li> </ul> <p>... this has the further implication that:</p> <p><b>υ τ</b> A Party may hold all or part of one or many Deeds</p> <p>... since neither of the subjects in the <b>τ</b> Proposition pair (or its graphical representation) is singular – a requirement for Normalization – we reduce the “many-to-many” relationship to a pair of “one-to-many” relationships using an Associative Entity as an intermediate:</p>	 <p>The word “multiple” is, in this case, a more natural way to say “one or more.”</p> <p>A Many-to-Many Relationship, while valid, cannot be used for logical inference, since inferences require singular subjects.</p>
<p><b>φ</b> The Proposition pair from <b>υ</b> ...</p> <ul style="list-style-type: none"> <li><b>τ</b> A PARTY MAY HOLD ALL OR PART OF MANY DEEDS</li> <li><b>τ</b> EACH DEED HOLDER IS (MUST ALWAYS BE) A PARTY</li> </ul> <p><b>χ</b> ... is then joined with the Proposition pair from <b>τ</b>:</p> <ul style="list-style-type: none"> <li><b>τ</b> A DEED MAY APPEAR ON MANY OWNER RECORDS</li> <li><b>τ</b> EACH OWNER RECORD RELATES TO ONE DEED</li> </ul>	
<p><b>ψ</b> Traditional predicate logic tends to acknowledge only the integer-like “zero, one, or more” – hence the common (but unfortunate) restrictions in most diagramming tools. These of course merely reflect the declarative RDBMS/SQL DDL syntax limitations for cardinality. The phrase “or part of” in the first Proposition of pair <b>φ</b> is not therefore prohibited as a Quantifier by Logicians; they simply don’t consider percentages in syllogisms.</p>	<p>In this example, any Party may own some percentage of a given Deed. Specifically the desired constraint would (in simple<sup>25</sup> terms) be that a Party’s percentage of ownership of a specific Deed must be greater than 0% and less than or equal to 100%.</p>

The result of this admittedly superficial analysis of the Natural Facts we are dealing with – as opposed to any Arbitrary Constraints imposed on us (e.g. the ownership limitations on a “Club”) – can be simplified as the series of Proposition Pairs **ζ**, **φ**, **χ**, **π**, and **ρ** discussed above, summarized here as a diagram with its extended Proposition Pairs:



- τ** Any subclass of Party may hold all or part of many Deeds, each of which establishes ownership of one or more Assets.
- τ** (The Corollary) Any Asset appears on only one Deed, which is owned by one or more Party subclass instances.

We now have very precise statements regarding the Natural Rules for this mini-project, and a good idea of what details remain to be clarified. This will form the basis of our table structures, onto which we will graft the constraints required to enforce the remaining Natural Rules as well as – temporarily – whatever Arbitrary Rules are required.

By now, the few remaining readers are likely wondering whether this level of analysis or this elaborate a model is justified – or far too complicated for such a simple scenario.<sup>26</sup> But “complex” and “complicated” are not at all the same, and while this approach to design may take longer, the first time a system change is required, you will get home in time for dinner, while your agile friends will likely be working through the night or even the next several weeks. Moreover, the time it takes to perform such analysis will decline significantly as you gain practice, experience, and intuition, while the time it takes for others to cobble together an illogical (being random) model will not decline.

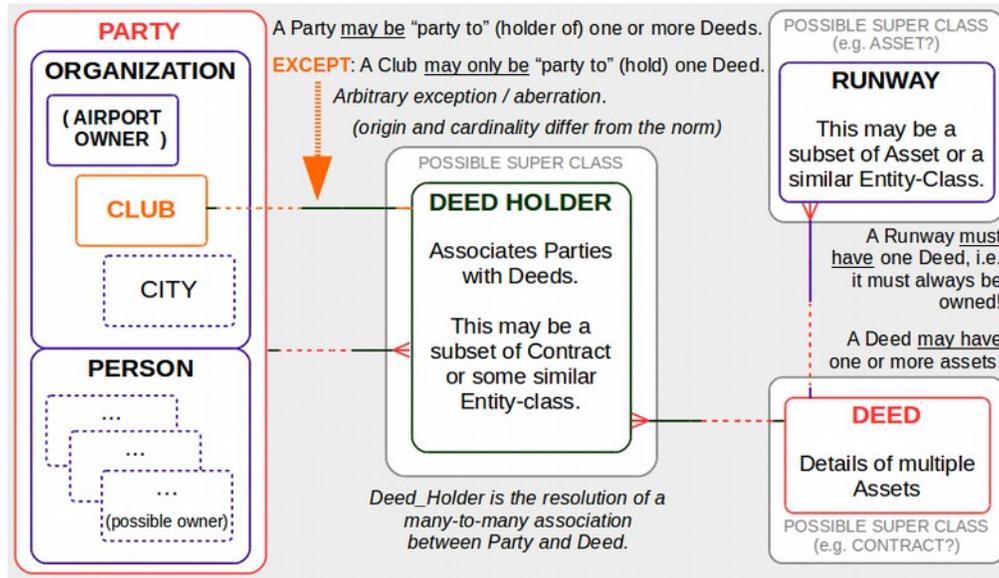
We’ll conclude Part 1 by comparing the generalization we derived with the original, then add a few attributes. In Part 2, we will continue with discussing approaches to enforcing the Requirements – even the Arbitrary ones.

<sup>25</sup> A little thought will show that, while necessary, such a constraint is insufficient to enforce data integrity. Be patient!

<sup>26</sup> It could be argued that this scenario probably doesn’t require a database at all, so we are justified in assuming it must actually be part of a larger system! The author’s statement could be interpreted as requiring a new database or simply new structures.

## GENERALIZED LOGICAL SOLUTION

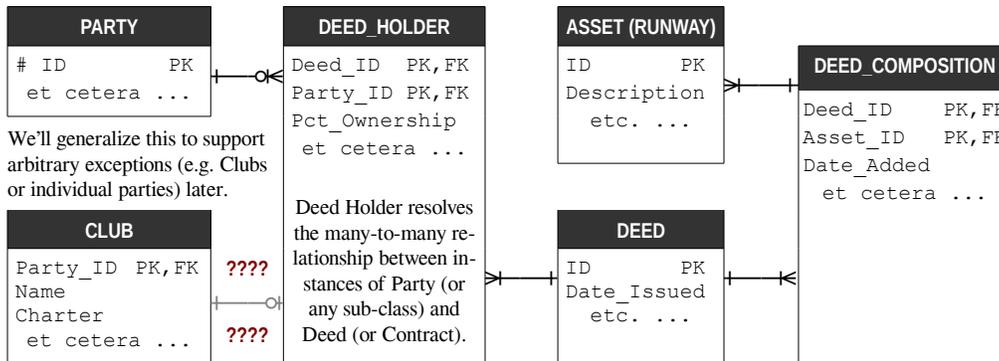
This revised approach to modeling the situation is shown using the CASE\*Method/Oracle/Barker notation to permit a direct comparison to the original model introduced on page 3 of this document.



Although a great deal more needs to be known about the particular situation to recommend a specific physical implementation, the eventual table structures would most likely resemble something like those shown below. Club, of course, is actually a subclass of Organization – itself a subclass of Party, but that relationship is not shown here in order to maintain clarity in this explanation<sup>27</sup>.

## GENERALIZED PHYSICAL STRUCTURES (AS AN ERD)

Once again, it is important to recognize that the requirement that Club be handled differently than what reality might suggest is the only unusual issue that needs to be addressed in the design. Shown below in a more common modeling notation is what the entity-classes shown above might look like as tables with their important attributes.



A Club is a Party, although that relationship isn't shown here to keep things simple. An Asset could, of course, be an Airport; some Runways might be included in an Airport Deed, but obviously not all would need to be. A Deed is held by one or more Deed Holders, and any Deed Holder may own a percentage so long as the total percentage for any deed always equals (or at least never exceeds) 100%.

<sup>27</sup> If we assume that these are new structures being added to an existing well-designed business database, the Party-Organization-Person structure – a classic use of an exclusivity arc, but out of scope for this discussion – will already be in place. The fundamental definition of Business, after all, is interactions between and among various Parties.

## INTERMISSION

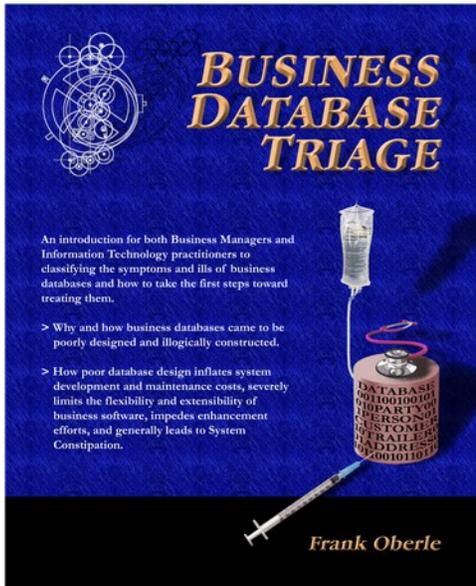
From some myopic perspectives, it might seem that we've spent twelve pages critiquing a seemingly simple data model, and have been overly pedantic about what may appear to the inexperienced developer as some rather trivial logical concerns.

Unfortunately, experience has shown that inattention to such details inevitably results in database designs that support a business for only a very short time, but result in making subsequent system enhancements – particularly to software applications – increasingly difficult and time-consuming to implement: in short, the system constipation referred to in footnote 5 on page 4.

A concert pianist doesn't seem to give much thought to the actual fingering used to play some complex Bach fugue, nor does a basketball player give much thought to the angle of incidence when using the backboard to bank a shot. As with most professionals, all of these "picky details" have become intuitive through extensive study, observation, and practice. This is also the case with logical data analysis and modeling and, for experienced modelers, most of what has been described so far takes place intuitively once the process is understood.

But – if we wish to protect our business data assets in a responsible and professional manner, we haven't yet reached the point where we can convert our "Generalized Logical Solution" into an actual database definition. For that, we need to go considerably further.

In *The Maltese Runway – Part 2*, therefore, we will explore the various constraints required to insure data integrity for this small group of tables, and discuss in general terms some techniques that can be used to implement them.



More information and sample pages at:  
[www.AntikytheraPubs.com](http://www.AntikytheraPubs.com)

In addition to an ongoing series of Database Design Notes, Antikythera Publications recently released the book “*Business Database Triage*” (ISBN-10: 0615916937) that demonstrates how commonly encountered business database designs often cause significant, although largely unrecognized, difficulties with the development and maintenance of application software. Examples in the book illustrate how some typical database designs impede the ability of software developers to respond to new business opportunities – a key requirement of most businesses.

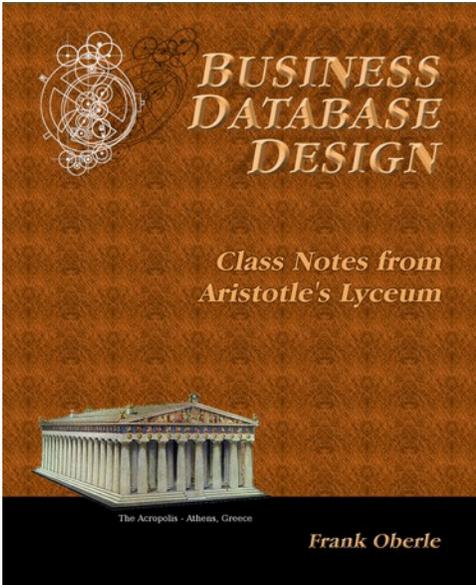
A number of examples of solutions to curing business system constipation are presented. Urban legends, such as the so-called object-relational impedance mismatch, are debunked – shown to be based mostly on illogical database (and sometimes object) designs.

“*Business Database Triage*” is available through major book retailers in most countries, or from the following on-line vendors, each of which has a full description of the book on their site:

CreateSpace: <https://www.createspace.com/4513537>

Amazon:

[www.amazon.com/Business-Database-Triage-Frank-Oberle/dp/0615916937](http://www.amazon.com/Business-Database-Triage-Frank-Oberle/dp/0615916937)



A follow-up book, “*Business Database Design – Class Notes from Aristotle's Lyceum*” is due to be available in 2015.

“*Business Database Design*” leads the reader through the logical design and analysis techniques of data organization in more detail than the earlier work – which concentrated more on understanding and identifying problems caused by illogical database design rather than their solutions.

These logical approaches to data organization, espoused by Aristotle and an “A-List” of his successors, have formed the basis for scientific discovery over more than 2,400 years, and directly led to the technology we deal with today, notably including both relational and object theory.

“*Business Database Triage*” explained the reasons why these principles were virtually impossible to apply during the early years of our transition to the use of computers in business, but since the technology is now sufficiently mature that such compromises can no longer be justified, the time has come to relearn logical data organization techniques and apply them to our businesses.



To download the ERD\_A TrueType Font used in this document, along with a tutorial and keyboard map, visit [www.AntikytheraPubs.com](http://www.AntikytheraPubs.com), where a variety of other Database Design Notes are also available for download.