The Data Organization

1251 Yosemite Way
Hayward, CA 94545
(510) 303-8868
info@thedataorg.com

Common Data Modeling and Database Design Mistakes

By Rainer Schoenrank

Data Warehouse Consultant

April 2013

Biography

Rainer Schoenrank is the senior data warehouse consultant for The Data Organization. He has degrees in physics from the University of Victoria and computer science from the University of Victoria and California State University Hayward. He has built data warehouses for clients such as Pacific Bell, Genentech, GE Leasing, SGI, PPFA, Brobeck, BofA, Clorox, Leapfrog and Intuitive Surgical. He can be reached at rschoenrank@computer.org.

Table of Contents

1. INTRODUCTION

The other day, a colleague asked me, “How do I know that the database I’m trying to program from is a good start to the database?” What first came to mind was C. A. Hoare’s comment about design:

> “You can either make it so simple that there are obviously no mistakes, or you can make it so complicated that there are no obvious mistakes.”

However, that would not have solved my colleague’s problem, although simplicity is a good sign.

Whether a particular database is an appropriate solution to the business problem depends on two things: the data model’s value and fit. The value depends on the data model’s interpretation by the business, and the fit depends on the data model making the same assumptions as the assumptions under which the business operates.

While there are always alternative data modeling solutions, it is very easy to have errors in a data model or database. The errors are caused by a complicated database design process that has three steps. The first step is the creation of the conceptual data model. This step has the goals of:

- Determining the scope of the database. What part of the business’s view of the world is to be included in the database?
- Naming and describing the entities that the business deals with. An entity is a thing about which the business wants to keep records such as customer, vendor, and employee where we describe customer as someone who sends us money.
- Identifying the relationships between the entities. The relationships are the connections between entities. For example, an employee received a customer’s order for a product in our inventory; this statement shows relationships between customers and employees, customers and products, products and inventory.

During the conceptual database design, there is the possibility of mistakes in scope, entity identification and relationship creation.

The second step is the creation of the logical data model. The goals of this step are:

- Identifying all the attributes that comprise each of the entities of the conceptual data model. An attribute describes one aspect of an entity such as its color, height, weight, label or name.
- Defining the properties of each of the attributes. The attribute properties are name, description, logical data type and sample values.
- Creating the structure of the entity and all of its attributes.

During the logical database design, there is the possibility of error in not finding all the attributes, using the wrong logical data types and creating inflexible entity structures.

The final step is the implementation of the logical data model in a database management system (DBMS). The goal is to translate the logical data model with its entities, attributes and logical data types into a physical database of tables, columns and data types where the relationships are enforced by the DBMS.

In the sections below, the most common database design errors are grouped according to the database design step in which they occur.

2. CONCEPTUAL DATA MODELING

At the highest level of abstraction, the database design process starts by collecting information on the scope and context of the database. The set of this information, data elements and element relationships, is known as the database universe. To create the conceptual data model, the database universe is partitioned into non-overlapping entities according to the Principle of Orthogonal Design (Date).

These entities consist of lists and measurements. The lists are things that the business tracks such as customer, employee, vendor, etc. The measurements are what the business records to know its financial states such as sales, purchases and time cards. The data universe and its partitioning are shown in Figure 1.

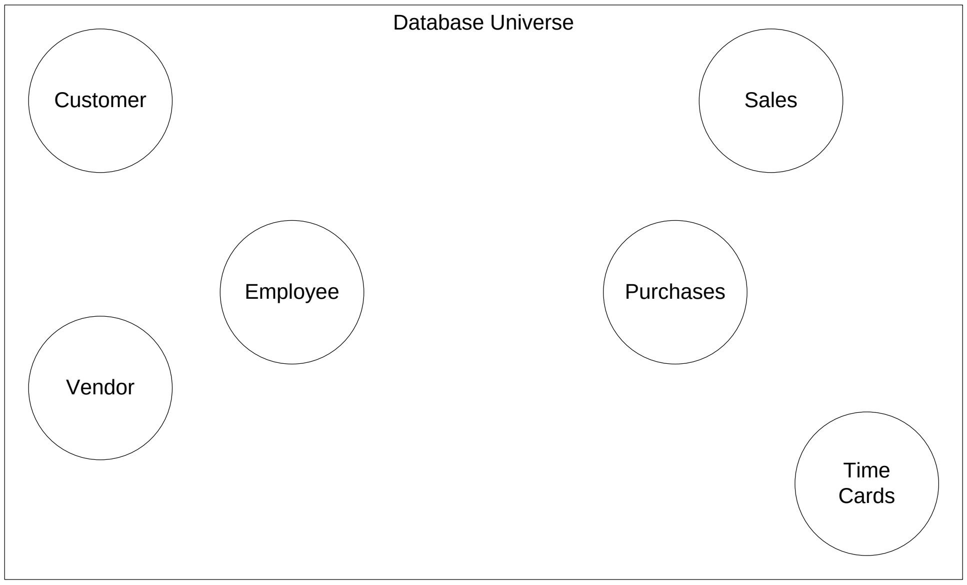

Figure 1 - The Database Universe partitioned by non-overlapping entities

It is necessary to ensure that the entities in the conceptual data model represent the most general logical data model that is appropriate to the business context and scope. Changes in the data model will have very expensive repercussions as the changes cascade into the application programs that use the database. Changes in the business scope will require changes in the database, which will require a number of changes in each of the applications that use the data model. A more general data model will be able to absorb the changes in the business scope without changing the model representation.

Since the partitioning of the database universe is arbitrary (Kent), there are lots of opportunities for error.

2.1 Modeling Company Policy

As you investigate the scope and relationships of the database, business users will tell you that it is company policy for one employee to be assigned to one position within the company organization. What they mean is “At any one time …” They forget the circumstances when someone is acting as interim CEO and that person is still the company’s CFO. The question of whether a history of promotions is required is not addressed by the company policy.

2.2 Process States versus Entities

When discussing business measurements with business users, it is common for them to describe process states as entities in business applications. Business users are quite used to filling out a new form when the sales process enters a new state. Business users expect these forms to become entities in the application database as in the left side of Figure 2. These forms (“price inquiry,” “quote,” “sales opportunity,” “proposal,” “order,” “delivery,” “invoice,” “payment”, “return,” and “refund”) are all states of the measured sales process. A better solution is to have an entity for measuring the sales process that is related to the customer and to the sales process states. The valid values of the sales process state are “price inquiry,” “quote,” “sales opportunity,” “proposal,” etc.

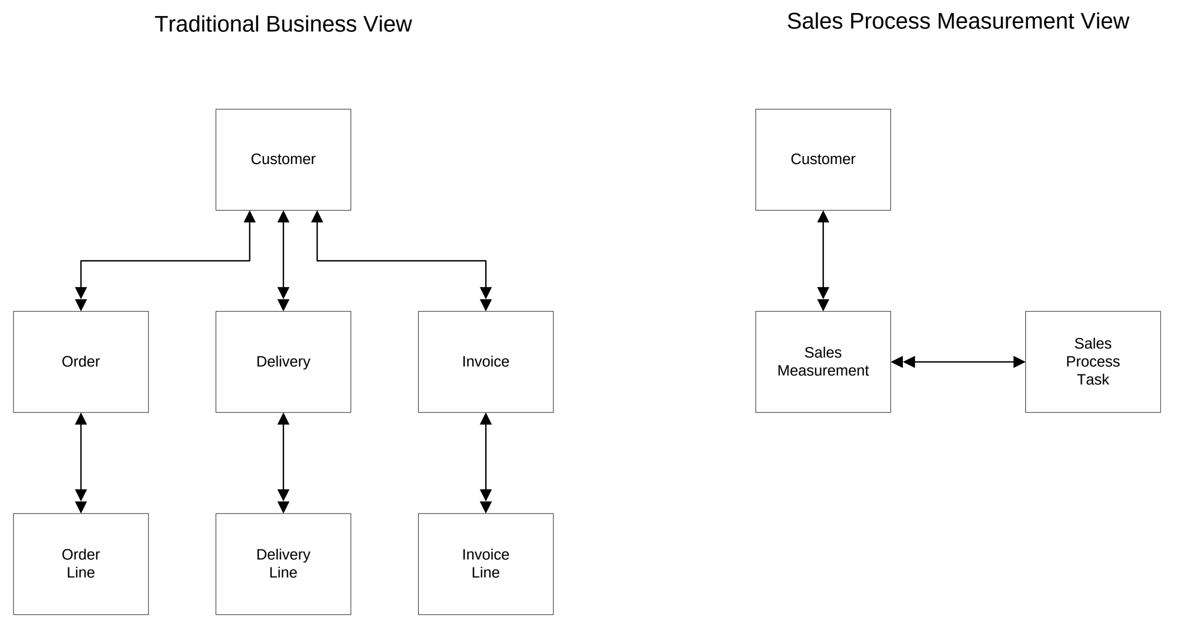

Figure 2 - Comparing Process Measurement Conceptual Models

The conceptual data model shown on the right side of Figure 2 with fewer entities is simpler and more robust. If the sales process changes, the traditional view would have to add or remove entities while the process measurement view would add or remove values in the Sales Process Task list.

2.3 Confusing Entity State Changes with Business Measurements

When examining business transactions, not all the transactions are business measurements. Business measurements involve money changing hands or business process events occurring. A customer address change transaction in the customer maintenance application is not a business measurement; it is a change of state of the customer.

2.4 Unconnected Entities

At the conceptual level, all the identified entities are related to each of the business measurements. The actual operation of the business may make assumptions about the required data that will be collected, but there should be no entities in the conceptual data model that are not related to some other entity.

2.5 Relationship Loops

The conceptual data model should have no relationship loops. You should not be able to follow a relationship path through the data model and return to the same entity without retracing the path. Relationship loops in the conceptual model diagram means that one or more of the relationships are redundant (Simsion, section 3.2.6).

Relationship loops are a symptom of over-normalization of the data model. Over-normalization is the result of using object-oriented analysis methodologies that identify each new noun as an entity. The relationships between these extra entities and the general data model entities always produce relationship loops. The problem can be alleviated by asking whether the new noun is

- a synonym of an existing entity
- a subtype of an existing entity
- a property (attribute) of an existing entity and a logical data type.

The answers to these questions can eliminate the problem of relationship loops.

A conceptual (or logical) model with redundant relationships means that the model is not a member of a relational algebra. Also, leaving the redundant relationships in the database means that the DBMS query optimizer will not produce valid results since the query optimizer assumes that the database is a relational algebra.

3. LOGICAL DATA MODELING

During the second phase of the database design process, the conceptual data model is transformed into a logical data model. This transformation is achieved by adding details to each of the entities in the conceptual data model. In the conceptual model, the information known about the entity is a paragraph or two describing the typical contents of an entity occurrence.

During the logical data modeling phase, each of the entities' unstructured description is parsed into structured attributes. Each attribute describes one aspect of the entity such as its color, height, weight, label, name or some other aspect of the entity. Again, the attributes do not overlap and the aspect described by one attribute is not described by any other attribute. For business measurements, the attributes describe who, what, when, where, and how much.

Since the not all businesses describe their entities and measure their performance the same way, there are many opportunities for error.

3.1 Knowing When to Stop the Attribute Creation

Many data modelers continue the parsing of the conceptual entity to ever finer levels of detail until they arrive at attributes defined by a label and the DBMS data types. Using the DBMS implemented data types for the logical data model is wrong.

The DBMS data types tend to make the logical data model unnecessarily complex and therefore difficult to validate. The appropriate level of abstraction for closing the logical data model (i.e., how much detail) must be decided before the data analysis begins. A good rule to follow for a logical data model is that data analysis is finished when the entities have been decomposed into their attributes and each attribute has been assigned to a logical data type, such as monetary amount, measured quantity, name, address, or phone number, as opposed to DBMS data types.

3.2 Using Consistent Logical Data Types

When you are attempting to learn about the attributes of an entity, users will tell you how they have been using this aspect of the entity on their forms and data capture applications. For example, they will say that the area of a rental unit is always measured in square feet. The data modeler then defines area as both an attribute and a logical data type. The logical data type of area implies that square feet is the unit of measure.

Using area as a logical data type violates the Information Principle (Date). The Information Principle states that everything in the data model must be stated explicitly. Because the area data type implies the units of measure and the implementation excludes the units of measure, the Information Principle is violated.

As the business continues to grow policy changes, the implied units of measure for area change, and all the processing for area must be modified. Recognizing that area has a data type of measured quantity means that the resolution of area is more complicated than a single numeric column but the changes required when the units of measure change are much simpler.

3.3 Logical Data Types Not Generalized Enough

As business users describe their data, they take many short cuts with their processing. In employee payroll application, there is both a salary table for supervisory employees and a wages table for hourly employees. By recognizing that wages and salary both have a logical data type of monetary amount, you can generalize a table “pay” that includes both “wages” and “salary” as in Figure 3.

WAGE TABLE			SALARY TABLE			PAY TABLE		
							monetary amount	
emp #	wages		emp#	salary		emp #	amount	rate
618-13	13.5					618-13	13.5	per hour
			418-63	26000		418-63	26000	per year

Figure 3 - Using Generalized Logical Data Types

This generalization of wages and salary enables hourly payroll and salaried payroll applications to be implemented using a single database.

3.4 Confusing Entity Organization with Entity Attributes

Businesses attempt to organize their customers, employees and such, into categories and hierarchies. The most familiar example is the company organization chart. This chart leads to the employee attribute “works in department” and the department is part of a district which is part of a region which reports to the company head office. This results in a data model as shown in the left side of Figure 4.

Every time the company is reorganized, this data model has problems. As the levels of the hierarchy are shuffled and renamed, major database changes are required.

This problem can be partially solved by recognizing that the company organization is not an attribute of employee, but a separate organizational entity. The other part of the solution is to recognize that the organization chart is an arbitrary creation and no particular organization chart is “correct”. There can be many organization charts and an employee can belong in each one as depicted on the right side of Figure 4.

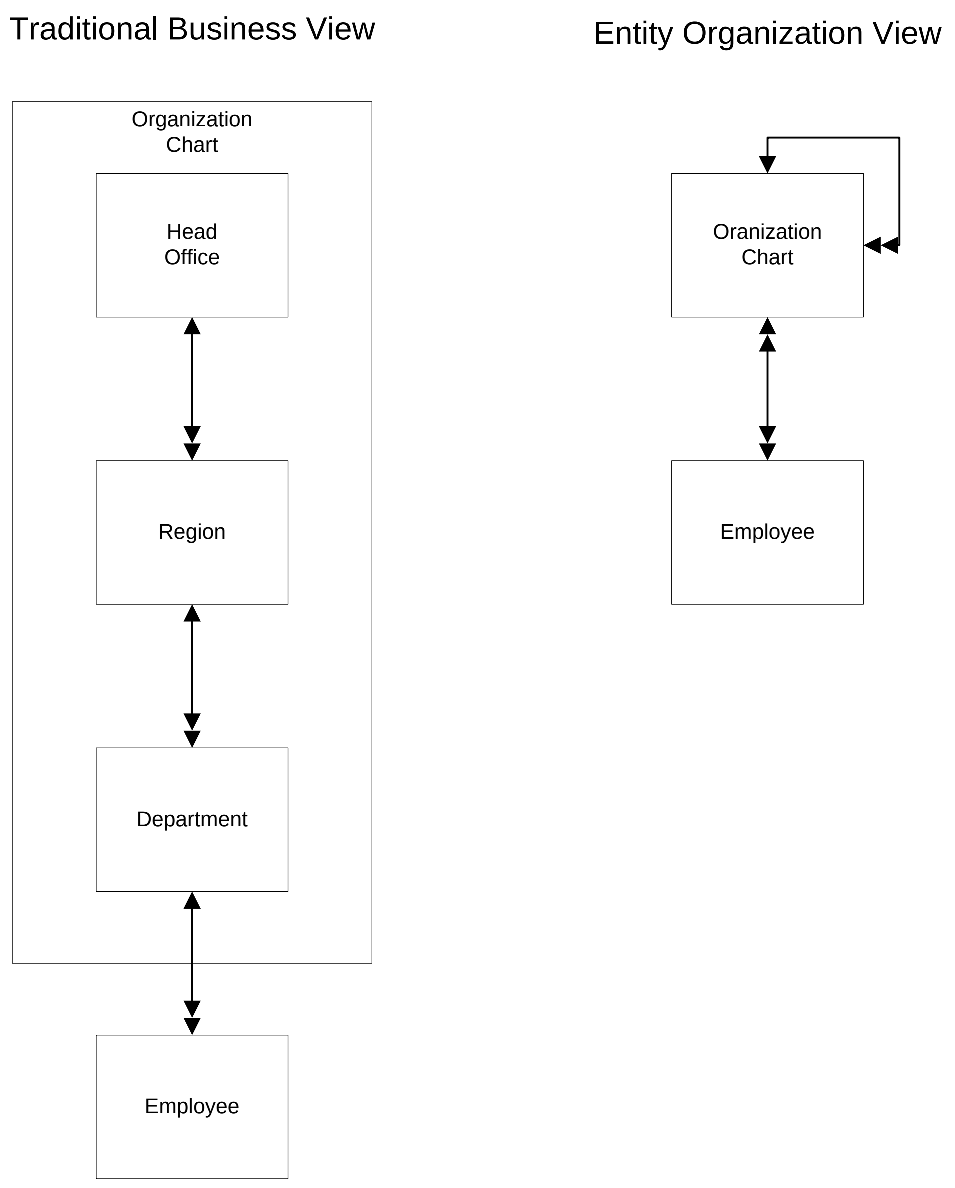

Figure 4 - Separating the Organization from the Entity

By separating the organization of employees from the employee entity, we have avoided the problem of representing each new organization chart and the database changes that entails. Now we can have each employee working at the correct organizational level without special processing to recognize the region instead of department. The names of the hierarchy levels are in the record rather than in the entity label, as required by the Information Principle (Date).

3.5 Confusing Entity Relationships with Entity Attributes

As the entity is parsed into structured attributes, some attributes are very specific, such as person height and weight, while other attributes are ambiguous, such as spouse. Is "spouse" of an employee an attribute of the employee entity, an entity in its own right, or a relationship between the employee entity and a person entity? In general, spouse is a relationship between one person and another person. The relationship should be implemented explicitly in the database design according to the Information Principle (Date) as shown in Figure 5.

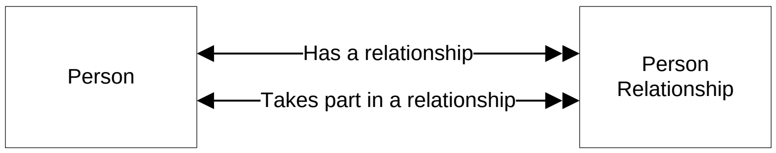

Figure 5 - Person Entity with explicit Relationship Entity

By making the relationship explicit, you can see that "spouse" is not the only type relationship between persons. The person relationship can also handle other relationships such as father, mother, beneficiary and cousin. With appropriate attributes in the relationship entity; the database can handle previous and future spouses as well as the current spouse without any changes in the logical design.

3.6 Inappropriate Resolution of Attribute by Their Properties

When decomposing entities into attributes, the decision as to whether the attribute is part of the entity or a separate attributive entity depends on the properties of the attribute.

To create the logical data model, we decompose the entities of the conceptual model by describing the properties of the entity. For example, the customer is described by the attributes:

- Name
- Address (billing, shipping, office, etc.)

- Credit limit
- Gross Revenue

The traditional approach creates one entity with 6 or more attributes, but there are questions to answer about the customer entity and each of its attributes. The questions to answer about the entity are:

1. Is the list of attributes complete?
2. What other attributes need to be tracked?

The questions to answer about each attribute are:

3. What is the logical data type of the attribute?
4. How many occurrences of the attribute are there? The answer will be either one or many.
5. Is it important to track the changes in the attribute value over time? For example, is it important to know when the credit limit has changed?

The answers to Questions 4 and 5 will determine the structure of the customer entity. An example of the allowed answers and the required entity type is shown in Table 1.

Customer Property	Logical Data Type	Number of Occurrences	Change Tracking	Required Entity Type	Example Entity Name
Customer Name	Name	one	no	entity	Customer
Credit Limit	Monetary Amount	one	yes	entity history	Customer Credit History
Customer Address	Address	many	no	entity attribute	Customer Address
Gross Revenue	Monetary Amount	many	yes	entity attribute history	Customer Performance History

Table 1 - Properties of Customer Attributes

Table 1 shows four different entity types in the second last column and names the entities in the last column. This means the customer entity should be decomposed into multiple entity types as shown in Figure 6.

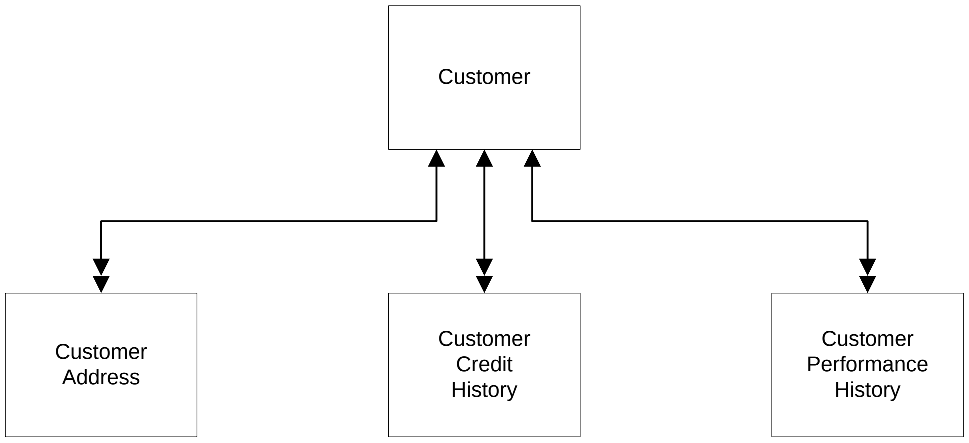

Figure 6 - Decomposing Customer Entity according to Attribute Properties

3.7 Using Inappropriate Values for an Attribute

When attributes are created, they are given a label (name) and a logical data type. Also, to complete the definition of the attribute, a list of typical values is given. These values should be describing the same aspect of the entity. For example, in a public safety application, the value for a person's hair includes black, brown, blonde, white, and bald. Bald is not a color. This indicates that the attribute of hair is more complex than only color.

4. PHYSICAL DATA MODELING

After the logical data model is complete, it must be implemented as a database in a Database Management System (DBMS). To implement the database, the logical data types must be translated into the data types supported by the DBMS, the entities translated into tables, and the attributes translated into table columns. To support the assumptions of the Relational Data Model, keys need to be created for each table (an entity is a set) and the relationships between tables need to be implemented with foreign keys in the appropriate tables since there is no constraint construct in the DBMS.

Since the DBMS does not support user defined data types and there are many new items to create, the physical data modeling process provides opportunities for error.

4.1 Inappropriate Logical Data Type Resolution

Representing logical data types in the DBMS database is a fundamental difficulty. Only some of the logical data types map directly into an implemented data type. For example, the logical data types of ordinal and date will map directly in DBMS data types. Other logical data types will require multiple columns to implement, such as the logical data types of measured quantity and monetary amount.

The enumerated data type is the most complex implementation. The enumerated data type allows only a limited set of values for the attribute or column. For example, currency is one of US dollars, CDN dollars, pesos, pounds, yen, Yuan, etc. The enumerated data type is implemented as a lookup table with the DBMS providing referential integrity as shown in Figure 7.

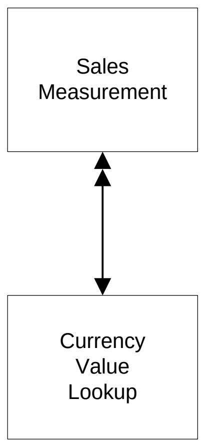

Figure 7 - Using Lookup Table as a restriction on a Logical Data Type

This implementation approach is very confusing in the physical database. The lookup tables add a lot of complexity to the physical database diagram for very little information about the structure of the database.

4.2 Resolving Logical Data Types as Tables

The representation of the complex logical data types such as monetary amount, address or name can be problematic. The entities customer, vendor, and employee all use the data type of address and address becomes several separate columns in the implemented database. This is because the DBMS does not support user defined data types.

By applying the rules of data normalization to the implemented database, the address data type becomes a table as shown in Figure 8.

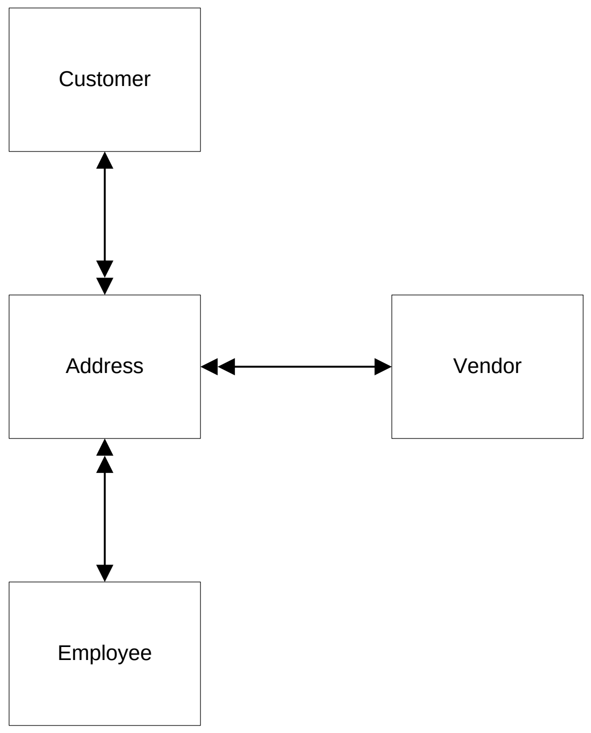

Figure 8 - Implementing the Address Logical Data Type as an Entity

The data model shown in Figure 8 is wrong for three reasons. First, a logical data type should not be represented as an entity in the database and address is a logical data type. Secondly, extra processing is required to determine whether the new address already exists when adding a new customer, vendor or employee. Also, extra processing is required to check that an address is not used by the other entities and the existing address can be deleted. And finally, the attribute of customer address is no longer explicit in the database in violation of the Information Principle (Date).

Another example of this error is the use of party by ORACLE. Party can be a customer, employee or a vendor in the business model, therefore party is a data type. The statement in the meta data is customer is of type party where party is a user defined data type even more complex the data type of address.

4.3 Normalizing a Physical Data Model

Normalization is a process that validates that a logical database adheres to the requirements of the relational data model. Codd devised The Relational Data Model as a logical algebra which showed that a database could always be manipulated correctly using simple set operations.

Codd defined a database as a set of relations and constraints where the meanings of the relations do not overlap. A relation is defined as a set of tuples. The design of a database is equivalent to the problem of partitioning of a set of sets into non-overlapping three sets. The set partitioning problem is an NP complete problem and so has no algorithmic solution in polynomial time. But a given partitioning can be quickly tested to see whether it is a valid partitioning, this is the normalization process.

Normalization is used to validate a logical database design. Using normalization to modify the logical database design as the database is implemented in a DBMS implies misapplication of the relational data model and the database implementation process.

5. SUMMARY

It is important to get the database design correct because mistakes are very expensive to correct. In general, a database is used by many applications and each change in the database design will cascade into many changes required in each application. A robust database design will be able to absorb the changes to the data model without changing the existing table definitions.

Database design mistakes are caused by misconceptions about the Relational Data Model and misunderstanding the database design process

In the Relational Data Model, Codd describes a mathematical algebra of relations and operations that provides the specification for building a consistent DBMS. The Relational Data Model is a mathematical construct, like Linear Algebra or Number Theory, not a database design methodology. The Relational Data Model provides a validation technique for the logical data model not for the physical database implementation. The implementation of the database requires many approximations and engineering choices that invalidate the axioms of the Relational Data Model.

The database design process is a hierarchical decomposition of the database universe using the Principle of Orthogonal Design (Date) and the Information Principle (Date) to ensure the entities and attributes of the database do not overlap and are explicitly described in the data model. The confusion is caused by misunderstanding the process of database design, that is, what are the levels of abstraction, what are the components of each level of abstraction and what do you need to know about each of the components.

6. REFERENCES

C.A.R. Hoare, IEEE Computer, February 1995

Codd, Edgar, A relational model for large shared databanks, Communications of the ACM, vol 13, number 6, June 1970

Date, C.J., *The Relational Data Dictionary*, O'Reilly Media, Inc., Sebastapol, CA, 2006

Kent, William, *Data and Reality*, North Holland Publishing, New York, 1978

Simsion, Graeme C. and Graham C. Witt, *Data Modeling Essentials, 3rd Ed.*, Morgan Kaufman Publishers, San Francisco, CA, 2005