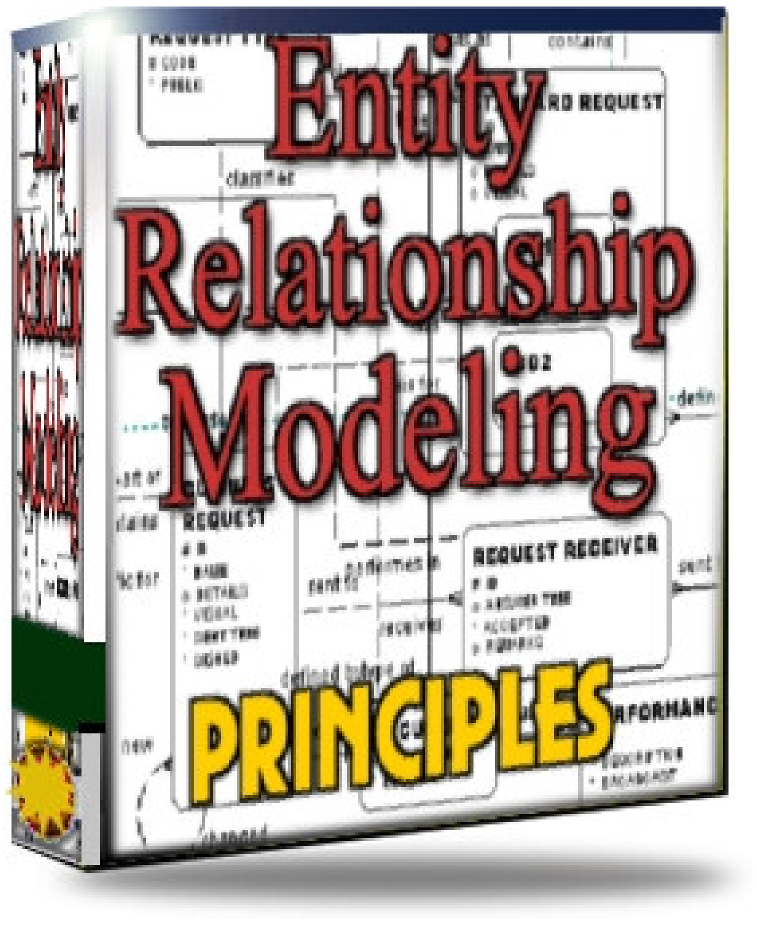
Entity Relationship Modeling – Principles



Author: Alf A. Pedersen

Introduction.

Entity Relationship Modeling (ER modeling) is by far the most common way to express the analytical result of an early stage in the construction of a new database. This eBook describes the principles for ER modeling, as well as the most important terms used in modeling a new database.

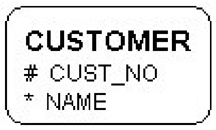
In the analysis phase, we are concerned about finding out about the business. At this early stage, we normally do not talk about performance-specific issues. The main goal of the Analysis phase is to determine what information the business needs to perform a certain task (accounting, invoicing, customer support), how that information should be best possibly organized, and what are the relationships between the different sets of information. We start with a few definitions:

1. Entity.

An entity is a specific object of interest to the business area. We might say that in an accounting system, two things are basically necessary: Accounts and transactions. In a customer support system, we would need to have some information about customers, for a start. Each such unit of essential information is named 'an entity', with a name and attributes.

2. Attributes.

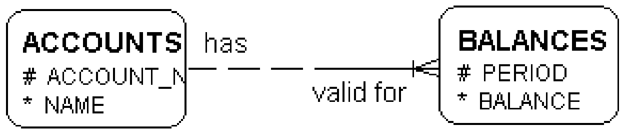
Each entity will normally have one or more attributes. Attributes may be thought of as smaller pieces of information within an entity. Together they describe our entity to the degree we find necessary. An example is a customer entity:



The entity is named CUSTOMER, and has two attributes, that we have found to be important to know about the customer: Customer number and customer name. Most likely, the customer will have several other attributes as well, but that will show up during the analysis phase. The attributes have a # and a * in front of them: A # means that the attribute is (part of) the primary key for that entity. The primary key is a unique value for every new customer we get so that we can uniquely identify him from all the other customers. The attribute NAME has a * in front: This indicates that it is a mandatory attribute; for every new customer, name MUST be filled in. The opposite is also possible; if an attribute has an o in front of it, it is an optional attribute, and a value is not required.

3. Relationships

In a relational database, all entities have bonds between them, expressed as relationships. A relationship is a link between to entities, and it tells us something about which relationships exists between our entities. The following example illustrates this:



This sketch tells us that a given account MAY have zero or more balances, but a given balance MUST BE valid for one and only one account. That sounds reasonable. Reading ER diagrams with relationships, gives us a very quick view of the logical structure of a database in the making, without thinking about the physical implementation of it, at this point.

These are the basic elements you need to know about before starting out on your ER modeling: Entities, their attributes, what uniquely identify an occurrence of an entity, and what the relationships between the different entities are.

The Entity.

1. The single entity

As we define different entities, we find that we are digging deeper and deeper into the problem domain of the business we are modeling. The more we, as system analysts, interact with the business we are modeling, the more information we find belong in different entities. This is how the process works.

You should not be afraid to create new entities, if so just as placeholders for ideas. In the analysis phase we need not be concerned about it; the analysis phase is where we test out ideas, new questions, and eventually evolve towards a solution which is sound, correct, and shows the business as it really is (or as we would like it to be).

It is also a good idea to create attributes as they are coming up. Again, they can be evaluated and thrown away later if we really do not need them. With a good ER modeling tool, your ER diagram may be thought of as a sketchbook, especially in an early phase of analysis.

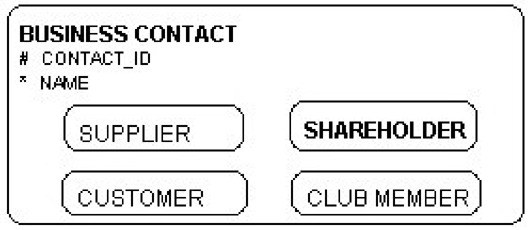
Single entities are relatively easy to spot: An account is something different from transactions or balances, and a customer is something different from an order. However, customers and orders are related to each other through relationships, as well as accounts are related to transactions and balances. Sometimes we may enter a dilemma, when two or more entities actually look very similar: F. inst.: What is really the difference between a customer and a supplier?

Both entities have some form of identification; they both have a name, address, telephones, contact persons etc. We may even have a discount system for our customers, and our suppliers also give us some discount mechanisms. The two are really quite like.

In our analysis work, we have two mechanisms for dealing with such uncertainties:

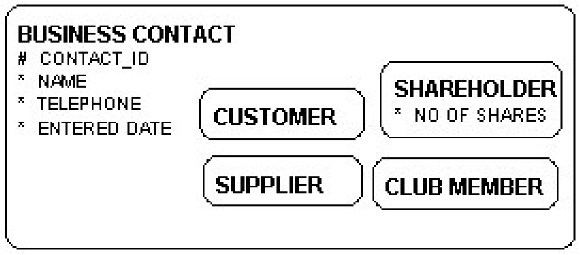
2. The super-entity and the sub-entity.

If we first draw an entity we call BUSINESS CONTACT, and then draw two or more entities inside the BUSINESS CONTACT entity, then BUSINESS CONTACT becomes a super-entity, and the others become sub-entities of BUSINESS CONTACT. The purpose of this is to verify if we really are talking about different kinds of contacts:

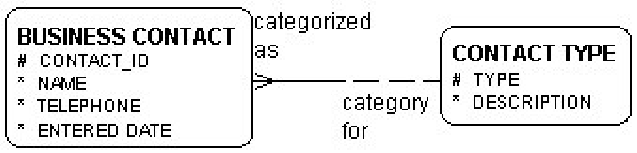


As we add attributes into this model, we must ask ourselves: The attribute I am about to create; is it valid for all the sub-entities, or is it specific to only one (or two...) of the sub-entities? If the attribute is valid for all sub-entities, we place it in the super-entity in stead.

When we work like this, we will (most likely) see that information-wise, there is no difference between a customer and supplier: They are two things of the same kind. However; shareholder may be something different:



We see that our analysis has led us to understand that there are really no difference in the information needs whether it is customers, suppliers or club members (we run a customer club…), But we need to track each shareholder's number of shares. The best thing to do until further, is to remove the three common entities and replace them by a type:



We can read from this that each BUSINESS CONTACT must be categorized as one and only one contact type, while CONTACT TYPE may be a category for zero or more BUSINESS CONTACTs. Contact type will here typically be customers, suppliers, club-members, or whatever business contact we have. We can se that we have built much more flexibility into the system by using super- and sub-entities in an early stage of analysis. We will return to this model after the chapter on allowed relationship types.

By the way, what about the SHAREHOLDER?

Well, the shareholder was a bit different from our other business contacts. If we want to keep the entity, we must give it all the attributes of BUSINSS CONTACT, in addition to the number of shares attribute, and keep it as an own entity. However, we will remove it totally for now, and save it for a later occasion.

Attributes in entities.

1. Most common metadata on attributes.

We take for granted, that doing a major analysis study in a given field, you use tools that are appropriate. As for the choice of a modeling tool, I give no concrete recommendations: I have used Oracle Designer (formerly Oracle CASE*Method) for the last 15 years, and I have found it to be a powerful and rich system, which delivers in many more areas than I have needed to use it for. I am probably a little biased here.

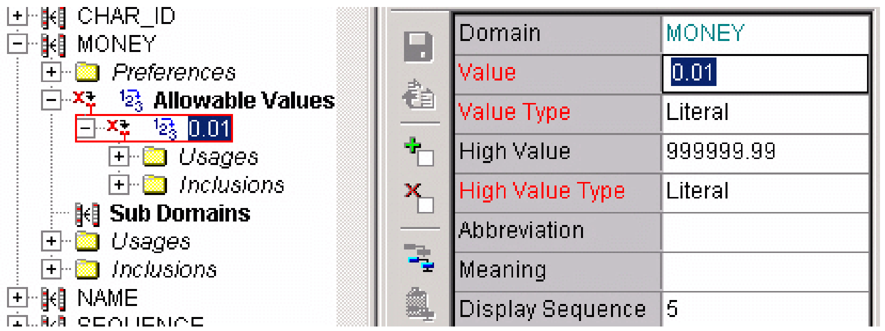
However, a toolset should include reusable objects: The results from the analysis phase should be the basis for generating a logical database model with tables and all other database objects for use in the design phase, as well as functions should be used for generating candidate modules.

Furthermore, the database objects should be used to generate the DDL (Data Definition Language) scripts for physically building all the elements of the database, as well as the candidate modules should be used to generate running program modules. Not that I expect a system to be 100% generated – far from it.

However, with such functionality you could show a prototype, which illustrated the resulting, needed functionality, but without the last finishing touch, or the most advanced business constraints built into it.

As you enter attributes, it is important that you can define their descriptions, optionality, data format and –length restrictions for later use in design. Here is what I would recommend you to: If at all possible, place each and every attribute in a domain.

A domain is a container that holds general descriptions, as well as data types and length, and in certain circumstances also can keep legal values or legal ranges for a domain. By relating every attribute to a domain definition, you ensure that if you change the domain, the change is reflected for all attributes that belong to that domain. A short example, taken from Oracle Designer:



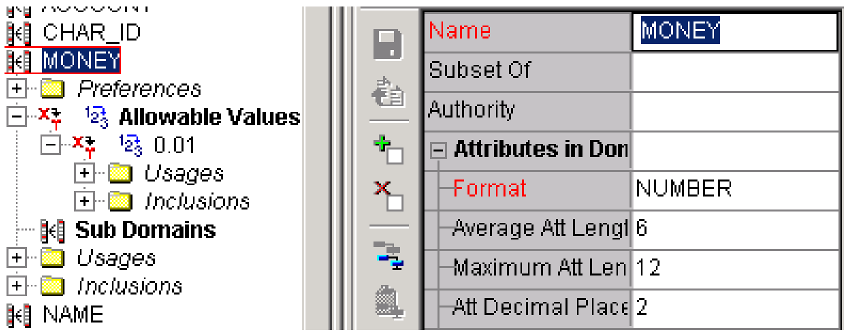
Domain	MONEY
Value	0.01
Value Type	Literal
High Value	999999.99
High Value Type	Literal
Abbreviation	
Meaning	
Display Sequence	5

We se that the domain MONEY has a legal range between 0.01 and 999,999.99. It sounds fair: Money must have some minimum value. How about maximum? This domain was created for use in transactions, and we have a rule in the business saying that no transaction larger than a million is allowed, for security reasons.

Now, we have suddenly gotten the opportunity to define a business rule here! This definition WILL result in a constraint in the complete database, at the end of the design phase, like this:

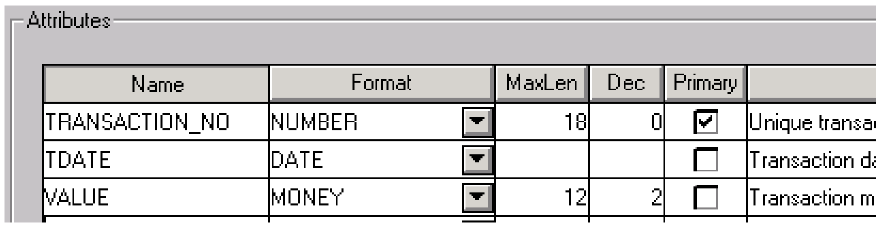
```
PROMPT Creating Check Constraint on 'TRANSACTIONS'
ALTER TABLE TRANSACTIONS
 ADD (CONSTRAINT AVCON_1071502401_VALUE_000 CHECK (VALUE BETWEEN
0.01 AND 999999.99)).
```

But we are too far into design now: Let us step back, and look at how we can define a type-domain as well:



Name	MONEY
Subset Of	
Authority	
Attributes in Don	
Format	NUMBER
Average Att Lengt	6
Maximum Att Len	12
Att Decimal Place	2

Since we have the legal range constraint defined, we should be able to reduce the length of the domain. Since the MONEY domain is attached to the transaction amount attribute in some transaction, changes in the domain have immediate impact. I will stress this: Relate all attributes to its domain, like this:



Name	Format	MaxLen	Dec	Primary	
TRANSACTION_NO	NUMBER	18	0	☑	Unique transa
TDATE	DATE			☐	Transaction d
VALUE	MONEY	12	2	☐	Transaction m

The best advice I can give, is: Do not settle for tools that do not support most of your development cycle.

Relationships.

1. Introduction

Relationships are the bonds between entities. They tell about certain rules that apply if you want to insert, update or delete occurrences (rows) from what later becomes tables. Relationships also tell us much about the logic connections between entities. In the physical design of the database, relationships become constraints that govern how you are allowed to manipulate data in the different tables. The relationships are there to enforce referential integrity: That your system, when it refers to something that is supposed to be in another table; it is actually there.

Relationships, as we use them in analysis, are in some ways in contrast to what is allowed when you physically transform the ER model to a physical, live database. This is due to the fact that in analysis, we model the business, but in design, we have to abide by laws of the database system. This is all, again, to enforce referential integrity; securing that your database does not get corrupt.

2. Three types of relationships

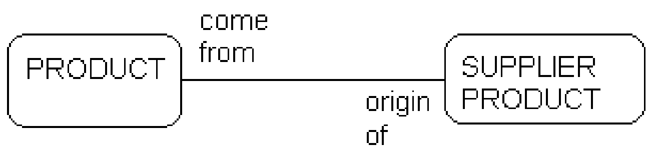
We have three types of relationships:

One-to-one relationships
One-to-many relationships
Many-to-many relationships.

For each relationship type, we may have one or both ends optional or mandatory, which give us a total of 10 different theoretical variations of relationships between entities. However; some of them are actually impossible logically, and a few other are so far-fetched that you probably have some less well-thought constructs in your analysis if you need them.

2.1 One-to-one relationships.

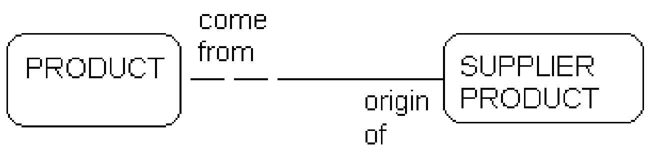
Mandatory-mandatory



This relationship says; a given product MUST come from one and only one supplier product. A given supplier product MUST be the origin for one and only one product.

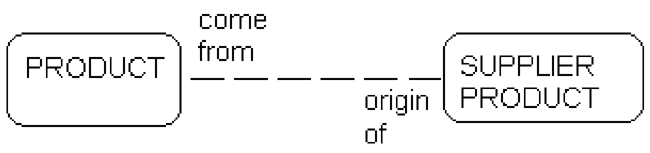
It sounds right, but it is not. If the supplier product is the origin of one single product, then the attributes of the supplier product most likely should be included in our product entity. Furthermore, we can deduct from this model that we cannot enter a NEW product unless a supplier product exists. However, we cannot enter a NEW supplier product before it has a product to relate to. It is a deadlock: We will never be able to enter anything into these two entities (when they become tables). This will do us no good; it is a result of theory, and not practice.

Mandatory-optional



This one is a little more flexible: a product MAY come from one and only one supplier product; however, a given supplier product MUST BE the origin of one and only one product. See the above relationship on how to solve this properly.

Optional-optional



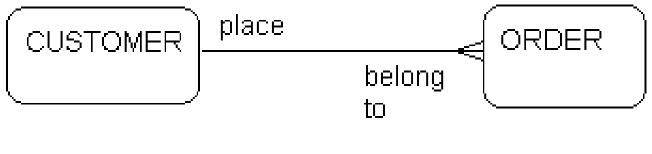
At last some freedom: A given product MAY come from one and only one supplier product, and a supplier product MAY BE the origin of one and only one product.

Do they seem a bit far-fetched? Yes they are. They are a signal of lacking analysis, for all practical purposes. Most likely, as said, the supplier product entity is really nothing more than a set of attributes in the product entity.

One-to-one relationships are a signal that you have different entities that are probably the same entity.

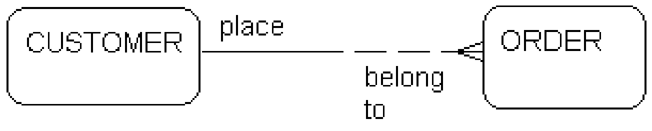
2.2 One-to-many relationships.

Mandatory-mandatory



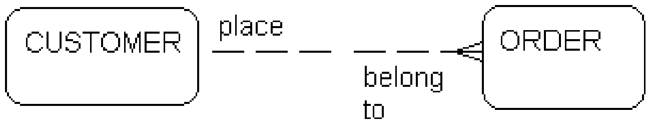
This signals that an order (in this case) MUST belong to ONE AND ONLY ONE customer, and a customer MUST place ONE OR MORE orders. While theoretically possible to construct, this relationship would violate constraints in the database: It will not be possible to create a new customer if there are no orders; however, an order cannot be created without customers.

Mandatory-optional



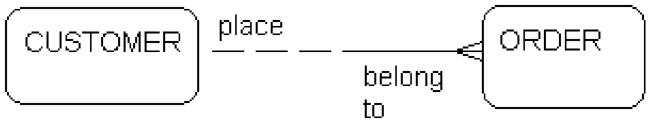
A new customer MUST place one or more orders, while an order MAY belong to a customer. Note that the order can exist without a customer, while you cannot have customers who do not have any orders. Well, there might be such a situation, but I find it hard. I would have looked more deeply into the problem domain. This relationship is possible, but relates to odd business rules.

Optional-optional



This is a weak relationship. A customer MAY place zero or more orders, while an order MAY belong to one and only customer. This is a so-called indecisive mode, normally used when you have two or more entities sharing an arc (more on this later). It may be used for optional attributes; however, an optional attribute is in itself a reason for further investigation: The problem domain may be incompletely analyzed. Be sure that the right questions have been asked – and have been answered.

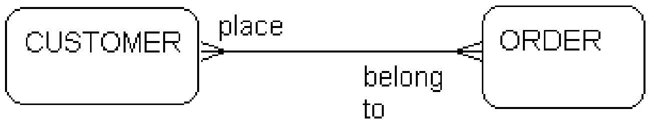
Optional-mandatory



By far the most common and useful relationship you can have: A customer MAY have zero or more orders, while an order MUST belong to one and only one customer. This relationship solves all the most common relationships between entities in the analysis.

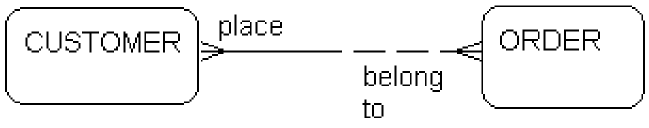
2.3 Many-to-many relationships.

Mandatory-mandatory



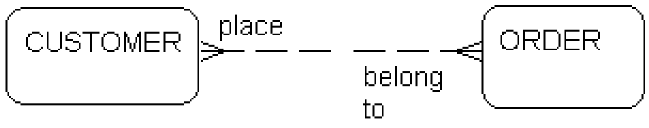
This relationship is impossible, for the same reason as the other mandatory-mandatory relationships: It is a catch-22, or deadlock: Neither entity may have new occurrences (rows) before the other has got its.

Mandatory-optional



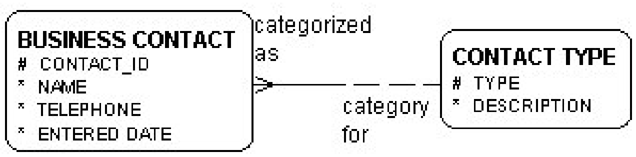
This is a relationship that needs to be resolved; analysis is unclear.

Optional-optional

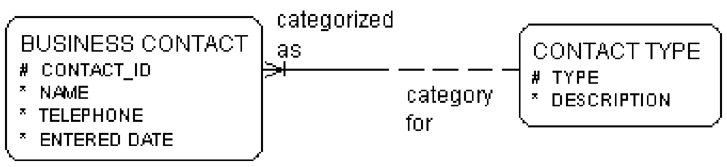


This is a very useful relationship: It implies what we call an 'intersection entity', which will give an entity holding all legal combinations of orders and customers. This many-to-many relationship is much used in an early stage of analysis to indicate just that. This is such a useful relationship that I will evaluate it right here:

We return to the model of business contact and contact type we had earlier:

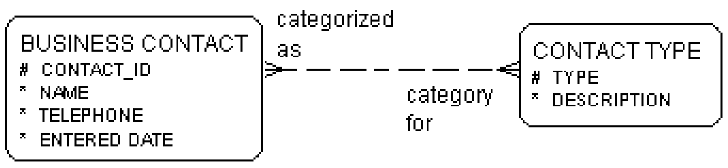


The way this is constructed, it only allows for a business to exist once. If we want a customer also to be a supplier, he must get a new contact id, and all information must be re-entered. We can (partly) solve it by making the primary key a combination of contact id and contact type:

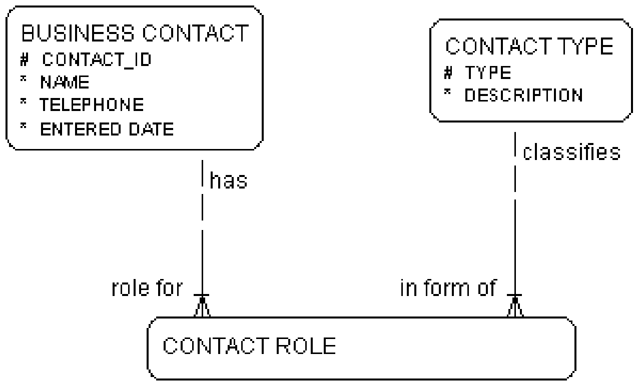


Note the vertical bar on the crow's foot: It indicates that the relationship is part of the primary key in contact id. This is one of the most common errors being made in ER modeling.

This is a violation of the Second Normal Form. Name, telephone and entered date have nothing to do with contact type, only with business contact. We see that we are not able to properly represent the fact that a business contact may be a number of contact types. Here we can take full use of the many-to-many relationship capabilities:



This relationship says that a business contact MAY BE categorized as zero or more contact types, while a given contact type MAY BE a category for zero or more business contacts. This is total freedom. However, as stated earlier, we have no way of representing this in a physical database, and we have no way of taking care of all intersections in the ER model either, so we dissolve the relationship like this:



We may now read all rows in contact role to find out which roles a given business contact has towards our business.

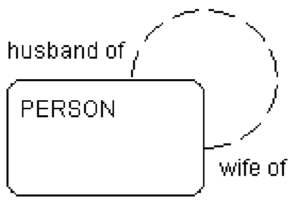
Do not be afraid to use this construction. In my experience, it will always add flexibility into the finished analysis result.

3. Involuted (or recursive) relationships.

Involuted, or recursive, relationships, are self-relationships; relationships from and to the same entity.

This is a special construction normally used to represent a hierarchical tree structure. This kind of relationship alone has just as many possibilities as the ones discussed above have together. However; most of the combinations just theoretically usable: For all practical reasons they are impossible since they relate back to themselves; you cannot relate to something that do not exist yet. Constraints will be broken in the database. We will therefore concentrate on the three forms of involuted relationships that may be of any use to you:

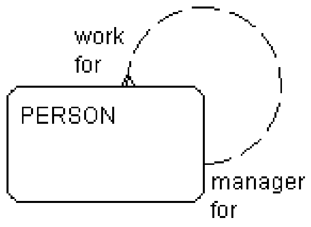
One-to-one relationship, optional-optional



This may be a useful relationship. The relationship makes it possible for you, if you wish, to be married to one person (at the time). However, if you want historical data as to who you were married to earlier (as it happens to often today), you need more analysis.

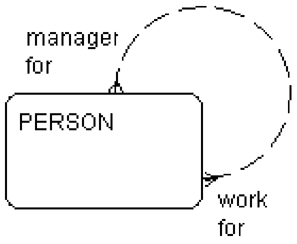
If you substitute PERSON with PRODUCT, this relationship may also be used for suggesting an alternative product, if the original is sold out, or is no longer manufactured.

One-to-many, optional-optional



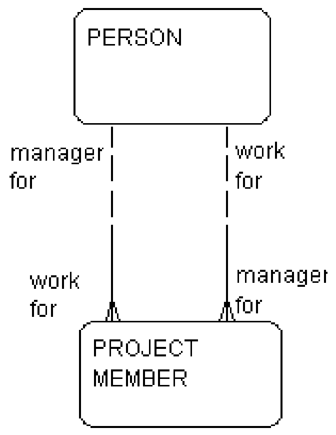
This is a classical hierarchy, or tree structure. Note however that any one person MAY work for ZERO OR ONE AND ONLY ONE manager, while a given person MAY be manager for zero or more persons.

Many-to-many, optional-optional



This is a network structure. In a project-organized environment, any one person MAY have ZERO or more managers, while any person MAY be manager for ZERO OR MORE persons. This construction is also good for decomposition of products: If you swap PERSON with PRODUCT, we may read it as: A product MAY contain zero or more (sub)products, while a given (sub)product MAY be a part of zero or more products.

As with all many-to-many relationships, we must create an intersection entity, and resolve the many-to-many relationship into two one-to-many relationships:

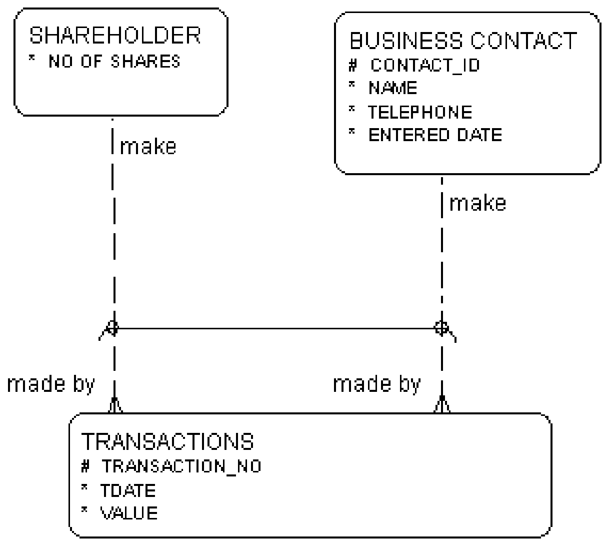


When naming both ends of relationships, try to find meaningful and descriptive phrases so it feels natural to read the relationships back and forth. Note how we have read all the relationships in this chapter. Avoid, if you can, words like ‘is’, ‘has’, ‘in’. Those are quite meaningless. (Although I cannot help myself, either, at times…)

Using Arcs.

What is an arc?

An arc is a constraint that crosses to or more relationships going into an entity, and it indicates that the relationships in the arc are mutually exclusive. I will explain it with graphics:



The arc in this model tells that for any one transaction, it is either valid for a shareholder, or for a business contact, never both at the same time. In my opinion, arcs are a result of insufficient analysis: We must have both relationships optional and that reduces the strength of referential integrity. Furthermore, it is an indecisive signal: ‘We are not sure here, choose for yourself’. They also give a signal about a possible missing business rule, also.

Try to avoid arcs; however, in an early analysis phase, they are a good signal about a problem domain that has to be explored further.

The Database Analysis Team
- A Teamwork

The importance of a professional Database analysis team.

The complexity and degree of computer involvement in the business is constantly growing. No wonder; each 18th month, we can buy hardware with twice the performance at the same price. We are therefore able to put more demands on our software, until we reach some limit. However, it only takes another 18 months; then we can buy new hardware without these limits... and so on.

There is also a good reason for it: We may very well rely on a standard system for our accounting or payroll routines, we can use market standards in word processing and spreadsheets, and so on. What separates a high-performing business from a failure is the way the CUSTOMER is reached for, and how he is treated.

That is customer marketing and customer care. The customer is always the business. If you do not have customers, you do not have a business.

If such a business exists, however, please let me know.

Such systems, systems that give the business an advance compared to its competitors, we may call strategic systems. If two businesses buy the same strategic system from the shelf, then they do not gain any system advantage towards each other.

On the other hand, the business that is fastest to respond, and deliver, and at the same time is competitive, will definitely have an advantage. In a world with rising competition and globalization, this will grow more and more important. Good news for the software industry and the analysis team...

In the analysis stage, we need an analysis team of both business experts as well as experienced system analysts. In addition, we need tools that can help us seeing the overall picture, as well as helping us further forward.

Top business experience is required in the analysis team in order to get in-depth understanding of the business itself.

Remember: Our model, at this stage in development, must reflect the business, not some constraints given by any tool or personal preferences. This happens all too often and usually with not-to-good results.

In the analysis team, the system analysts must have an expert level knowledge of business modeling. By business modeling, I mean exactly that. I do not mean expert knowledge of f.i. Entity Relationship modeling without relating it to the real world. However good a person may be educated, nothing beats the experience earned from several similar tasks at the equivalent level of complexity. How does one gain experience then? Participate under a tutor. I would never hire a consultant without experience and trust her to understand the complexity of my business, all by herself.

I will not go into detail as to the total project staff is composed. This will depend on many outside factors, such as degree of participation from each party, size of the project, formal requirements (public sector tends to require at higher level of project staffing, partly due to rigorous documentation requirements), etc. What we focus on is the tightly performing party that determines the final business model: The experts on the business together with the system analysts, preferably more than one in this phase.

Due to the increasing complexity that tends to be taken into systems development, I cannot imagine a development project of any noticeable size that should not use a development tool as support for the analysis team as well as for each stage other of development. I regard the tool(s) chosen as an integral part of the team. In many cases, the tool is also the communicator between the business and the analyst.

I have often started a project by going through the way we communicate with each other. In my experience, the business soon finds Entity Relationship diagrams familiar, if not as familiar as to the system analysts. However, they are a means of communication that work. The same may be said for function diagrams, or function hierarchies. They are even easier to understand for a non-system person.

You are free to use this eBook as a give-away in your ongoing or upcoming projects.

Analysis progress

With the tools of today's technology (laptops, projectors, computer networks and not least, the Internet), the analysis process can take place literary anywhere.

Actually, most processes are able to. For all practical purposes, we are more mobile: some parts of the analysis progress may take place at the business office, and other parts at the analyst's office. They can communicate on the net at any time, collaborating outside of formal meetings. No doubt, this is increasing performance in the analysis progress, as well as saving time, and thereby costs.

For instance, sometimes I do vital parts of a project at my home office, but my tools repository (Oracle), resides on the server park at the office. I can also get limited access to it for my customers, if I wish.

(Sidebar: The invention of home offices is perfect for employers: We, the employees, work both day and night, and on top of it, we love it…Crazy world).

OK, back to business: While we may have inherited a sketch of an ER model from the strategy phase of the project, now is the time to start asking the difficult questions. We will be probing that sketch, and stressing both the model and the business representatives (yes, I am now talking on behalf of the system analyst; since you are still reading this, most likely you are also one).

The difference between an experienced and an un-experienced analyst can make great differences on the finished work. While an un-experienced analyst will try to fit the business' points of view into the E-R model, the experienced analyst will start to doubt some points of view, remembering her experiences from two years back, in a similar situation.

This phase in the project is not about making nice models; it is about making models that work, and making models that will work also if the business perspective changes, somewhere in time. It will. It always does.

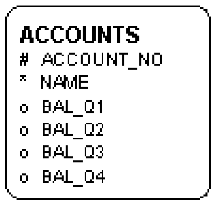
Building flexibility into a business model is really quite simple: Follow the rules for the five Normal Forms (do at least 3NF!), and you have already asked a lot of vital questions, securing flexibility. If you want an in-depth study of the five Normal Forms, go to our eBooks section on our website www.databasedesign-resource.com

The sooner we can ask, and collect answers, to questions about the business, the faster the analysis progress will be. I must emphasize the importance of delivering results to your customer as fast as you can, without compromising quality; of course, it is the best way of making repetitive business too, as well as marketing.

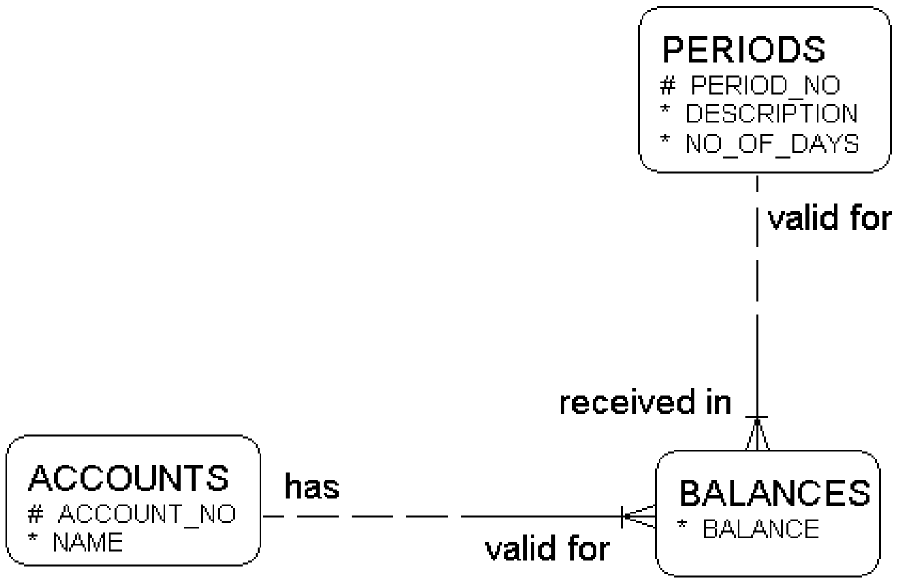
Analysis trap 1
- Modeling with incompleteness

When we work on establishing the conceptual business model through Entity Relationship modeling, one of the most common pitfalls we risk walking into are:

 You may be asked: 'We need a system for reporting quarterly balances'. I have seen models put up like this: (the example is from a real world situation that eventually went wrong):

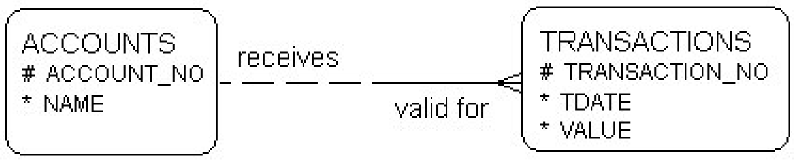


This was the first suggestion, which obviously will not pass the test of even the First Normal form. They ended up with this:



Nice enough.

But, this is based on a complete model of the reality? Where do we get our BALANCE from? It does not come from out of the blue. A full and true picture of the reality would be something like:



Quarterly balances are computed from of transactions. The entities for BALANCES and PERIODS are denormalizations; derived (or computed values) that has to come from transactions. It MAY be correct to use the former model (in a data warehouse), but it violates 3NF if you expand your view to look at the complete business model.

I call this The Analysis Trap 1. This may happen if one tries to limit the scope of the task at hand, and it would lead us into a situation where, whenever a transaction is inserted, updated or deleted, we MUST have a business rule that says 'Update BALANCES whenever something happens in TRANSACTIONS. There may even be a worse cause for this analysis trap 1 to happen: Someone might say: 'Performance will be lacking if we do not sum up here and there'. But those quotes are a certain sign that one is thinking physical implementation of the model; tables, SELECT statements, and so on. However,

A computer's performance ability has NOTHING to do with the analysis of the business!

(We will return to that in an upcoming Design Phase topic).

Remember, it is the business; how we run it (or would like to run it), that determines what information (entities) we need, and how the different types of information interact with each other (relationships). As you work your way through the analysis phase, the business model (and implicitly the E-R model) will become more and more accurate. Applying normalization techniques will ensure low (or no) level of redundancy (repeating the same information), and, as a consequence, also deliver a high level of referential integrity (correct relationship values).

Analysis Trap 2
- Incomplete business understanding

The Analysis Trap 2 is about incomplete business understanding or misinterpretation of the business processes. This is a common source for inadequate data models. There are (at least) two factors that may lead to this; The system analyst is lacking experience either from the specific business area, or in general, and/or the business fails to bring forward enough detailed information about the business needs.

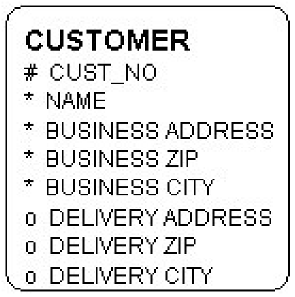
I have come to the conclusion, after more than 15 years of ER modeling, and a total of 28 years in the software industry, that more often than not, the "professional" part in the business analysis task, the system analyst, must take a fair share of the burden here.

Being a system analyst is not the same as being a programmer. These are two distinct different professions, and different skills are necessary. It is not a disadvantage for a system analyst to have a programming background; you may quickly see areas where the model will influence the development phase in a negative way.

Actually, problems with building queries against a relational database are very often an effect of an inadequate (denormalized) database structure. The opposite is also a fact: It is much easier to write optimal queries and updates against a normalized database structure.

The business probably do not know data modeling in detail. That is why they use consultants. However, some basic knowledge is required in order to participate in the analysis phase. You may forward them this free eBook on ‘Entity Relationship Modeling – Principles’ as a primer to understand it. You find it in our eBooks section. If the customer learns the basic principles, he is much better prepared to communicate his business and have a dialogue with the system analyst(s). Here are some common errors in ER modeling:

Failing to understand that the same information is repeated:



The business may say: For each customer, we need his business address and his delivery address. That is two addresses. If you model it like this,

you are violating the First Normal Form: Repeating attributes/group of attributes.

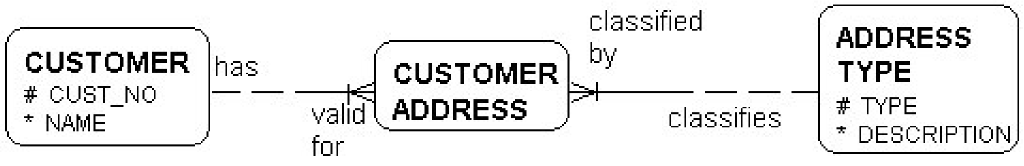
Ask questions such as:

* Does the customer have more than one delivery address after all?

* Does it ever happen that some customer asks for delivery, not to his address, but to his customer's address?

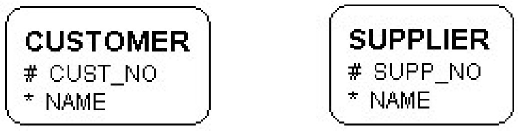
* Do customers sometimes refer to a branch address for business or delivery?

This model will let him have as many choices as he may wish. I do not say this is good enough, though…



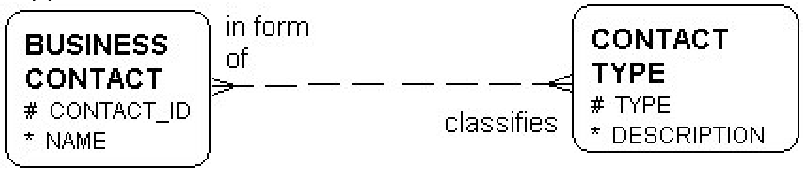
Failing to build flexibility:

The business says; We have customers and suppliers. You might model it like this,

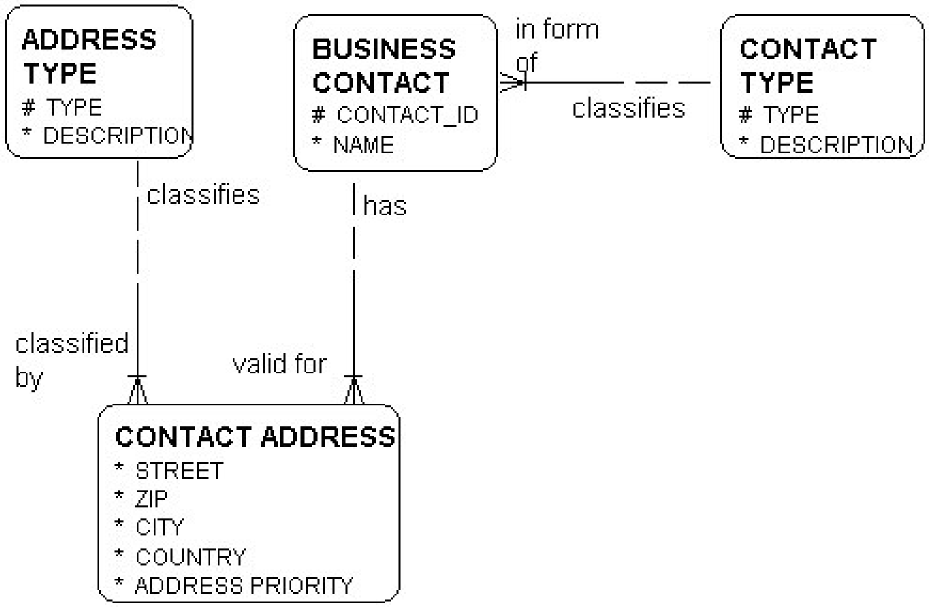


While instead, you could ask:

Does it ever happen that a customer is also a supplier of other goods? It happens more often than one should think. You could solve it like this,



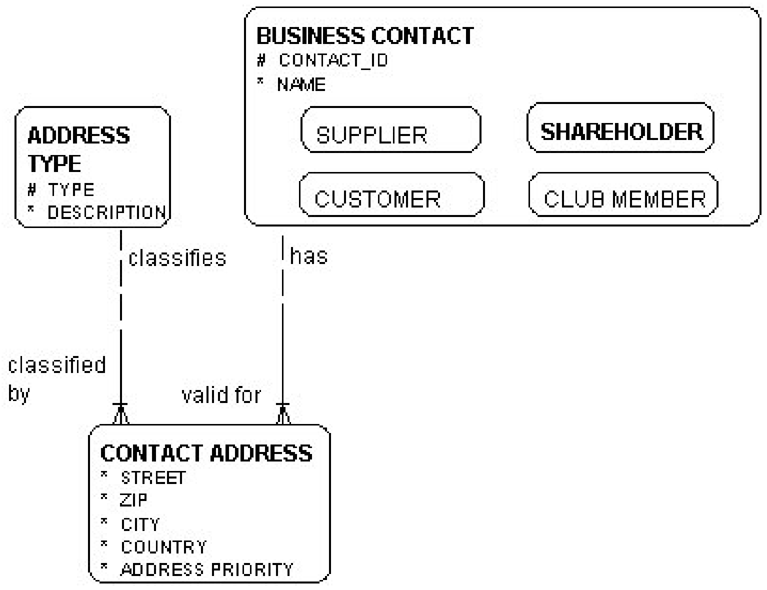
We changed the name of CUSTOMER into BUSINESS CONTACT. Using the many-to-many relationship in an early stage of analysis is a powerful mechanism. It shows that a relationship is more complex than first anticipated, and as long as that many-to-many relationship is there, it is a reminder to resolve it at some point in time.
Even worse; The business might say: We identify our customers and suppliers by a number from separate series to identify them, so we will need the customer type as an additional means to identify them. Truly, this happens. The analyst gives them this,



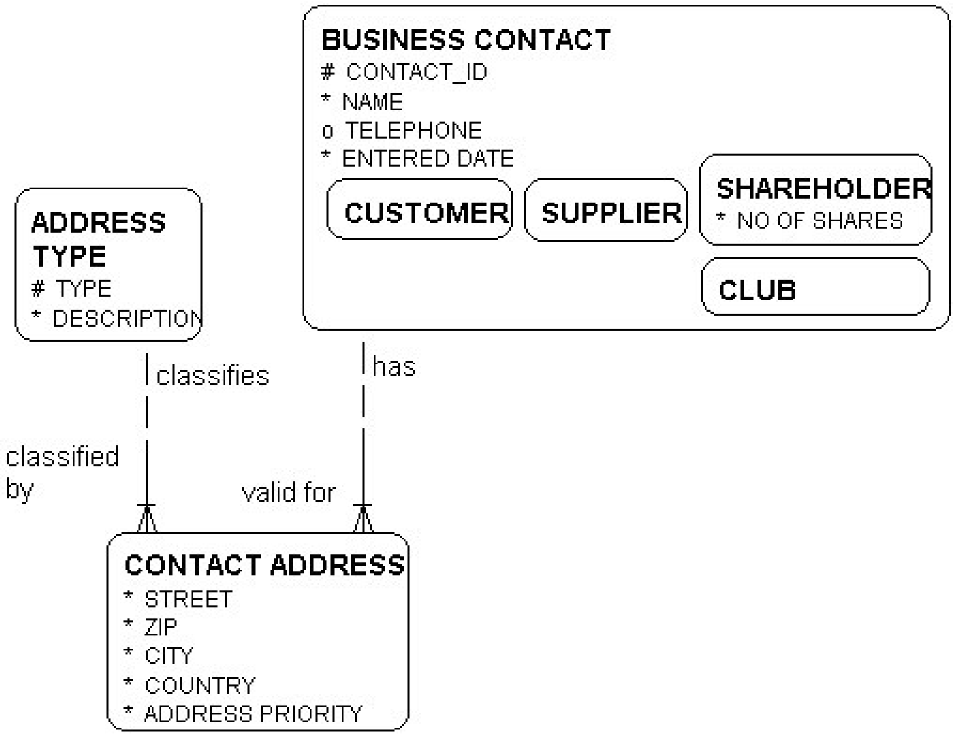
where the unique combination of Business contact no and contact type is the primary key. (The vertical bar at the crow's foot notates part of primary key).

This analysis trap breaks Second Normal Form. Name, address and so on is not dependent on contact type, only on business contact. If a business contact is both a customer and a supplier, all names and address information etc. must be stored twice.

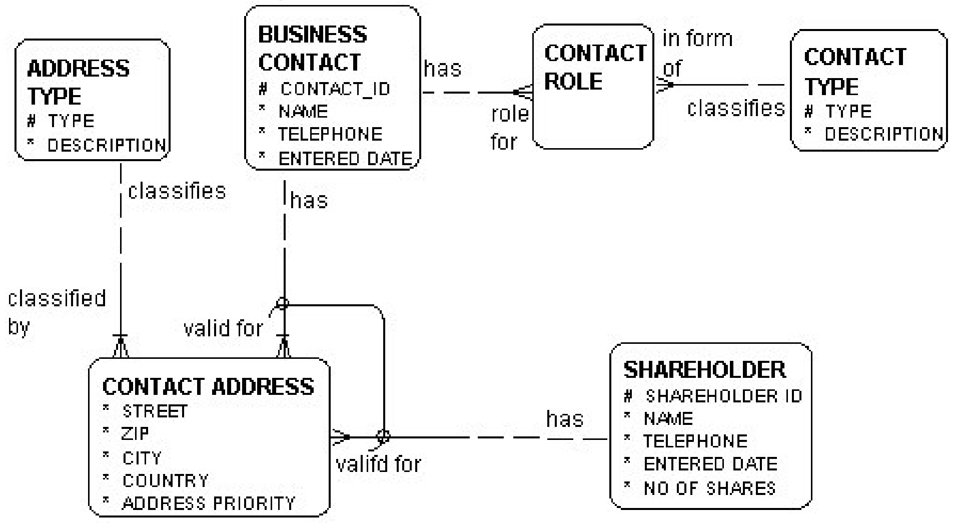
But what about the whole concept of customers and suppliers? They are only two of several groups that are relating to the business. In an early stage, we should rather say: We have several groups of business contacts, such as customers, suppliers, and maybe other interest groups, all of which we may have to deal with. In such cases, one may use the terms super-entity and sub-entity as a very powerful illustration of generic (common) structures, like this,



We are trying to find all attributes for each sub-entity. Since they are obviously closely related to each other, many of the attributes will be common. As the analysis moves forward, we add attributes to each sub-entity. Where an attribute exists in all sub-entities, we move it to the super-entity. We se that actually, some of the sub-entities are different types of the same thing, because they do not have un-common attributes:



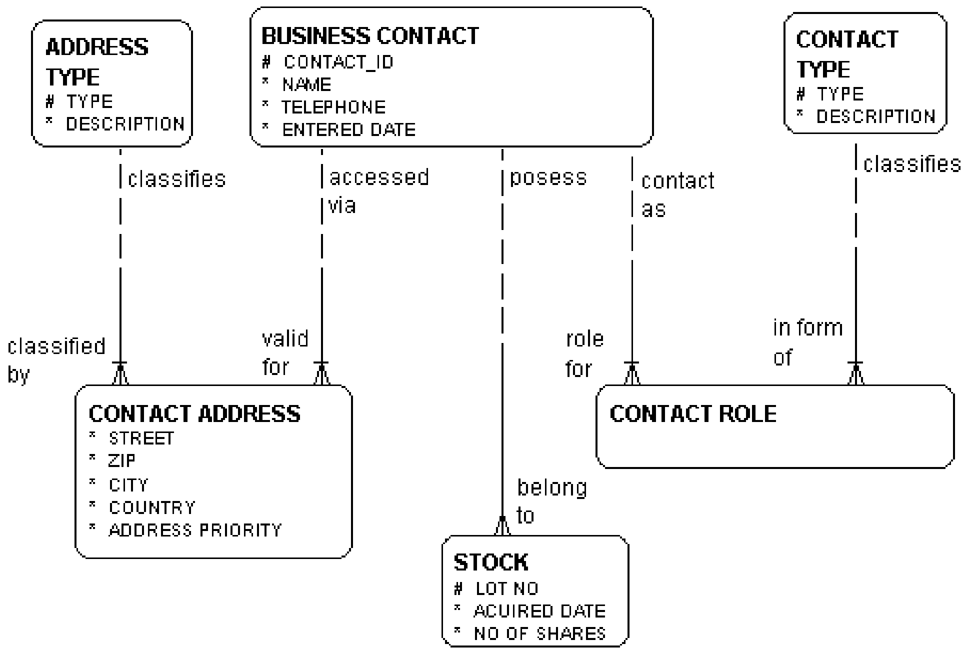
Now we can be more specific:



The model is still not complete - Primary key definitions are insufficient.

Also, the arc between business contact and shareholder has to be studied closer. Arcs are, in my opinion, a signal that the area is insufficiently studied: What we really say with an arc is: It's either this or that. Choose your pick. That is not good enough, in my opinion. Let us discuss this: The only difference between any kind of business contact and a shareholder is that a shareholder owns one or more stocks in our company. Otherwise, he may be a customer, a supplier, or whatever.

If we change the name of shareholder to stock, and we add a couple of attributes, and we define a primary key for contact role, we get:



Generic or specific models ?

We will look into the use of generic or specific database models in this keynote.

There are many analysts who advocate for using generic ER models to build a system, while others favor building it costumer-specific.

Actually, I slightly disagree with both directions.

I have studied many generic models. One common denominator is that generic models try to capture everything imaginable in a certain field of business, like financials, production, etc. If you need to cover everything imaginable, you are probably already a customer of Oracle Financials, Peoplesoft, or some other high-end ERP system.

The reason for starting a project involving ER modeling, design, development and deployment of an application, is that the business wants to build something unique; a new way of treating customers, new supply logistics, or anything else that gives it a competitive edge. A generic model will involve many entities, relationships, and probably business rules that are not relevant to the business. It means developing functions that will never be used, an extended database with tables that are never accessed, triggers that fire and execute nothing; Even this code;

```
begin
  null;
end;
```

Will have some impact on performance if it is executed often enough each day. A model specifically designed for the business sounds good. However, you have a chance of narrowing your scope too much, so that you build inflexible limitations into the system. Refer to the keynote on Analysis Trap 2.

An experienced analyst brings with him into the project a perspective based on earlier experience: He is able to recognize patterns in different areas of the business domain.

The customer is an area for which a pattern can be given. There are many common questions regarding customers, which can be reflected in a recognizable pattern.

How we build a sales order is also a common pattern: At some point there is a customer involved, the order is about one or more products, it should be effectuated and delivered, someone's has to pay, etc.

Patterns are studies of small, specific domains of a larger ER model, with one (or a few) in- and outgoing links to surrounding domains of the ER model. An experienced analyst will immediately recognize a given domain in the model as something she has studied before, and knows about. She can therefore bring added value into the ER model by discussing the domain with the business, and ask 'difficult' questions, based on limitations that are obvious with regards to the patterns of that domain, and which the analyst brings with her, in her experience.

Main point:

If you go through a development cycle for a new application, it is because you want something others do not have. You do it in order to gain some competitive edge, or you are realizing a very new business idea. Either way, copying something old is not too smart. However, keeping patterns, that cover separate smaller domains of the model, at work, may build you a system that is not only something unique, but also contains flexibility based on earlier, hard-earned experience.

Important Analyst Ethics - Reminding Words

The analyst ethics and business.

You have done the job, and a complete (?) Entity Relationship model, with Function Hierarchies, Events, and Business Constraints are all in place.

And, when conditions change, and the customer needs to order a rewrite, is he happy? No.

If he knows he can just create a new customer type in the CUSTOMER_TYPE table and then go on, without calling you, will he be happy then? I think so.

This is the responsibility of the system analyst: Giving the customer flexibility for future growth and change, without those costly rewrites. As a consultant you may disagree, you may just love those rewrites; an income generating tool! But I can assure you: In the long run, you will win by building flexibility and room for change for your customer.

I just read somewhere on the Internet: In the good old days, a dissatisfied customer would tell his frustration to 6 others; in the days of the Internet, he will tell it to 6.000 others.

I would not have liked that.

Summary.

Congratulations on completing your study on Entity Relationship Modeling - Principles !

I hope this lecture has been a learning one, and that you may use it in your future work on database design.

We would highly appreciate it if you took a couple of minutes to give us feedback on this lecture. If there are things you didn't understand, or you disagree with, we would like to hear it from you.
Just send us an email so we can make changes where appropriate.

Search the net for +normal +form to find articles related to this lecture.

http://www.databasedesign-resource.com
er-modeling@databasedesign-resource.com