This is a response to a question from **dzhu** on StackOverflow:

How to reference groups of records in relational databases

Response 1/2 to Update 3

You have been busy with the original question. Here are my responses.

> The basic Things (Subjects) are:
>   Animal (AnimalName)
>   Food (FoodName)
> ~~The Activities are:~~
>   Meal (AnimalName, FoodName)

Meal is a subject, a noun.

The Predicates are:

> Animal and Food are independent.
>   Meal is dependent on Animal and Food.

and:

```
Animal consumes 0-to-n Meals
Food is consumed in 0-to-n Meals
```

> I wanted to reference a record. What I really want to know is whether the following proposition is true.
> "An animal named x eat y for meal."
> Meal where (AnimalName, FoodName) = (x, y)

The problem here is, your propositions are starting to get complex. The *Relational Model* uses **First Order Logic**. Without explanation, basically, that means:

- each Predicate is a single sentence, that forms a proposition (which is true or false)
- we declare only the true propositions, that we are implemented in the model
- The structure of the proposition is:
- **Existence**

  `*Object* is { Independent | dependent on *Subject* }`

      Subject is a table

      Object is a table
- **Identification**

  `*Subject* is [ primarily | alternately ] identified by ( *Key* … )`
- **Relation**

  `*Subject Action Cardinality Object*`

      Action is a relation between the *Subject* and the *Object*, the **Verb Phrase**

      Cardinality as per *Object* end
- Note that these can be expressed in converse form, especially important for users (as in documentation, for discussions) and beginners. If I tell you that `Fred is Sally's father` (ala a FK Constraint definition), then you know from that fact, that `Sally is Fred's daughter`. It isn't two facts, we don't implement two FK constraints, SQL knows what a relationship is (the "teachers don't).
- It is one fact, like a coin, that has two sides. And both sides can be read, they are Predicates.

  ```
  Each Animal consumes 0-to-n Meals
  Each Meal is consumed by 1 Animal
  Each Food is consumed in 0-to-n Meals
  Each Meal is a consumption of 1 Food
  ```
- In the early and intermediate stages of modelling, usually only one side is expressed (the Verb Phrase), and the reader is expected to be competent and to understand and evaluate the converse side. Once it is stable, the second side is expressed.
- For users, both sides are expressed in the documentation.

Therefore you have to limit yourself to simple propositions. And of course they can be chained. This:

> "An animal named x eat y for meal."

> Meal where (AnimalName, FoodName) = (x, y)

becomes:

```
Each Animal consumes 0-to-n Meals; Each Meal is a consumption of 1 Food
Therefore Each Animal consumes 0-to-n Foods
```

Forget about the `WHERE` and the `(x, y)` That is your business, in your head re how to access the data, or in the object layers, or whatever. Those are outside the definition of the data.

> I also wanted to reference a group of Animals (such as all humans).
> Proposition: "All humans eat y"
>
> Wait... how can we differentiate humans and other animals? the Animal relation only has one attribute AnimalName.
>
> We need to add a new attribute AnimalType.
> Animal(AnimalName, AnimalType)
> Animal join Meal where (AnimalType, FoodName) = ('Human', y)*
>
> I wanted to know whether the following proposition is true.
> "All females eat y".
>
> Need yet another attribute: Gender.
> Animal(AnimalName, AnimalType, Gender)
>
> Animal join Meal where (Gender, FoodName) = ('Female', y)
>
> The approach I take is naive and without organization. I think I just scratch the surface of the Relational Model. I see that attributes can be used to differentiate/classify data.

All that is very good, given that you started this yesterday!.

One remark, you are trying to implement classifying info in Attributes. Generally, that should be in a **Reference Table**, with a FK reference in the child table, so that the values are controlled, not free-form. Further, these Reference Tables may or may not be elevated to Dimensions.

> Classification is crucial to the design. Maybe analysis is even more important.

Absolutely.

The way I see it is this. Analysis [of the data] is the task that you perform; classifications is the result of that task, the classes that you determine during the task. The two are not separate.

> Anyway, designing is not an exact science.

It absolutely is !

There is a large component of Aptitude in the application, and it is partly an Art, of doing all tasks well, in a way that they are all integrated with each other (otherwise they are fragmentsed, isoalted, dis-integrated). But it is a Science, and only a Science, which is competently (or not) and artfully (or not) applied.

The problem is that the "teachers" and the authors who write books, are clueless about the *Relational Model* and about the various scientific tasks involved in database design. They are florists and pot-scrubbers, so they do not recognise a science when they see it, they do not understand it, they cannot execute it, and therefore they cannot teach it. Notice, I can give you exact, precise instructions for each and every step. I am an Engineer, educated in the old system, before the education system was destroyed.

Date, Darwen, Warden, Fagin, Pascal, Zaniolo, Ambler, Fowler, Kimball, are all ignorant of the sciences involved, of the *Relational Model*. But they write books, market them heavily, and propagandise their precious myths. People like you read them, and get confused, subverted. I have no sympathy for the professors who use their books, without understanding, without knowledge of the Relational Model, without determining the authors to be the frauds that they are.

The Result is, OO and ORM types think the *Relational Model* has the capabilities of a pre-1970's Record Filing System, where (as they state) *you can't do this, you can't do that, and oh, it doesn't handle hierarchies ...* Since they do not know the *Relational Model* they cannot know what it does or does not support.

> For a complicated system, different people may produce different designs, with different table names and structures.

Yes. But some will be better than others, some will work, others won't, others still will work but only with extra coding.

So what's the difference ? Those who are properly educated, in both the technology and standards, and experienced, will produce models that work, the other produce models that are broken, or work badly.

The more complicated the system, the more that education, standards and experience come to bear.

> Are all great systems similar?

Yes.

All failed systems are similar, too.

> I see hierarchies in this way: (using propositions)
>   tuple (AnimalName)
>
> Here is an animal whose AnimalName is 'Tom'.
>   tuple (AnimalName, AnimalType)
>
> Here is an animal whose AnimalName is 'Tom', AnimalType is 'Human'.
>   tuple (AnimalName, AnimalType, Gender)
>
> Here is an animal whose AnimalName is 'Tom', AnimalType is 'Human',
>   Gender is 'Male'.*

Not bad for a beginner, and you hadn't yet read my directions in this response. Those are all minor propositions, minor Predicates, that can be read from the model quite easily. The formal Predicate, that I don't bother to state is:

```
Each Animal is described by ( Name, AnimalType, Gender, Description )
```

If AnimalType and Gender are Reference Tables, those two attributes get handled via *important* Predicates (discussed above), and they won't appear as plain attributes in the model, they will appear as Foreign Keys, bold, and with a Relationship.

Those are not hierarchies by any stretch of the imagination. They are a simple compounding of descriptors, attributes. (They might be “hierarchies” in the OO/ORM world, or on the fourth moon of Jupiter, but that is not relevant to us here, we are in the Relational Data Modelling world here.)

Now please read my posts, and linked posts, re Hierarchies, because you do need to understand them, you have them in your data, and we need to model them. Ask questions re the documents that I have given you, rather than posing what hierarchies are (as above).

> This reminds me of the superclass/subclass relationships in OO.
>   tuple (AnimalName)
>
> is more general than
>   tuple (AnimalName, AnimalType)
>
> (A point in an (n-1)-dimensional space represents a line in an n-dimensional space, oh, the projection operator)

Sure. A star that you see is already dead by the time you see it. So what.

That is possibly an inheritance hierarchy, as seen through the tiny lens of the OO perspective. Completely irrelevant when modelling data. Possibly relevant, if and when you get to write code, and creating objects. At this early stage, I would not comment on whether that is right or wrong, or whether the class hierarchy is valid.

I would have just one object for Animal, that has attributes, and I would load it from a View (which is a derived relation, more than one table, joined).

> It seems levels of abstraction is a constant theme in computer science.

Yes and no. These days, theoreticians abstract themselves so much that they are completely isolated from the problem space, and the abstractions have lost their meaning. Just study your question as initially posed. You are good learner, but your "teachers" teach poison. It was *so* abstracted that it did not have any meaning, and it could not be answered.

Second, they "teach" you to analyse and evaluate the "truths" about these fragments that have been abstracted, in an isolated manner, and without regard to everything else in the database. That means **context is lost**, and the result is, **integrity is lost**. And they don't even know that they lost it.

Both the Hidders and Köhler failures were due to this abstraction to too many levels, where the meaning and context was lost, and then they try to add *some* of what they lost, back in. Whereas I maintained the context, limited my abstraction to sane levels, thereby eliminating their problem, and their solution.

After my first response, after you learned some of the basics, in addition to lights going on in your head, you subsequently you re-posted the same question, with much more meaning. Now it can be answered (which is this response).

The point is, abstraction is great, I wouldn't do without it, but the abstraction that they "teach" these days teaches you to be schizophrenic and incompetent, to create massively inefficient Record Filing Systems with none of the Integrity, Power, or Speed of a Relational Database. But to label that heap of junk, a "relational database"

> Set theory and predicate logic give the *Relational Model* great power.

Yes. That is the theoretical underpinning.

And the fundaments are the tasks that comprise modelling:

- Normalisation (elimination of data duplication)
- Relational Keys (Hierarchies); etc,
- according to the requirements in the *Relational Model*.

But they don't teach that any more, they teach pre-1970's ISAM Record Filing Systems.

Solution

Now that you have given me detail, with meaning, I can answer the question much better. The requirements are easy to implement (the *Relational Model* does not have the restrictions that the hysterical imbeciles say it has), however you lack the fundamentals of (a) understanding the data, and (b) the modelling exrcise. So we have to do that first, in order for you to work through some progressions, and thus understand (a) and (b).

- **The purpose of data modelling is primarily to understand the data**. As data, and nothing but data. That includes Identifiers; Relationships; Basetype::Subtype structures; Predicates; etc.
- It is not to design a database. That is easy enough to do, once one understands the data correctly and completely. but that remains a very secondary goal
- the problem here is that you are rushing to make design and implementation decisions, without understanding and modelling the data.

Let me take you on a tour of the Data Modelling exercise. Our starting point is your question, re Animals, as described in your question, as per the details given yesterday. The goal is to implement a model such that the generic need as initially expressed, and *all* your further questions in the comments and the above interaction, are provided for.

I will give it in Steps. There are seven progressions to the model. At each stage, you will need to understand:

- the **Data Model** completely
  read & understand the **IDEF1X Notation** doc)
- the **Predicates**
  I have given all the relevant Predicates, so that you can work those against the model, and vice versa. (As described above, the minor descriptor Predicates are not given, since the attributes can be read quite easily from the model.)
- not only the Model vs Predicates but the difference between them
- the differences between Steps, the progression
- and how that changes re the Predicates (again, a very important feedback loop)
- and certainly, think about any reports that you need from the data, and the SQL code that is required to produce it. That is for appreciation, eg that it is made hard or easy with a certain progression, as opposed to understanding the model.

Step 1

Here is the first Data Model, it supplies the solution to the question, re Animals, Food, etc, as initially posed.

Generic 1 Data Model

Marvellous. Question answered. End of story ? Certainly not.

Step 2

We know that you want groups or grouping, of both Animals and Food. Then there is Gender (although Animals are usually happy with natural Sex). Those are Classifiers, attributes, sure, but as a FK Constraint, as described above. And I have given you Activity, to demonstrate the use of a different sort of Classifier.

Generic 2 Data Model

```
AnimalClass = ( Bird, Fish, Mammal, … )
FoodType = ( Meat, Vegetable, Krill, Carbon Dioxide )
```

Step 3

That isn't final either, because

- more groupings, or better groupings have become apparent.
- if you consider AnimalClass, Birds could be (Flight, Non-Flight, Diving etc).

Therefore we have an Hierarchy of AnimalClasses. Refer to my description *Response 2. Hierarchy Type 2. Rows within One Table*

Generic 3 Data Model

Now you can set up a grand hierarchy of AnimalClasses, as deep and as detailed as required. Note the **Relational Key** is compounded in Animal. And compounded again in AnimalFood. That is a *Hierarchy Type 1. Sequence of Tables*.

- In order to produce the full AnimalClass hierarchy, in a single *result* column, comma or slash or dot-delimited, as in `$PATH`, you need recursion, a simple Function that navigates the tree for ancestors. Similarly, you might need a single *result* column that lists the Food that an Animal eats. Another simple recursive function.
- Note that these are *result* columns, or flattened (de-normalised is the wrong term), not *data* columns, which remain Normalised. Typically these will be supplied in a View, one per "complex" table. Given in the Data Model.
- Note also, that these Views, the derived relations, are what the OO and ORM types focus on, obsess on, the data values, the de-normalised **View** of the data. As in a spreadsheet that contains the *result* set. And they implement that as "tables". Normalisation is not possible.
- If one focuses on the data values, at such a low level, one is prevented from stepping back and evaluating the overall picture, the context of each data occurrence, and thus prevented from modelling.

Step 4

Modelling is an iterative process. Each iteration resolves the set of problems identified in the previous iteration, great. **And** (not but) it exposes the next set of issues, which were not visible earlier. You might need six or ten or fifteen iterations, before the model is stable.

Each iteration establishes new entities, in order to record a new set of resolved facts-to-be-recorded. Each iteration may change or delete entities that were established in the previous iteration.

- This is the reason stamping every entity with an `ID` field, and treating it like a file, cripples the modelling exercise. One cannot progress anything if the model is a bunch of spreadsheets, fixed in one's mind, and all the entities are "known".

Now if you study that model [3] carefully, you should notice two issues.

- First, we know that some Animals are the Food of other Animals, some subjects are the object of some other subject, the issue of subject vs object is unresolved. Food and Animal, or better, AnimalClass have to merge.
- Second, AnimalFood is implemented at a level that is a bit low (in the hierarchy, which is laid out vertically in the model). It would be much better if we could relate AnimalClass to Food or FoodType
- Likewise with AnimalActivity.

Let's resolve that second issue and locate Food and Activity at a higher level.

Generic 4 Data Model

Step 5

At this stage, I would ask you to look at the new set of Predicates carefully, and the delta between [3] and [4]. I don't know about you, but it seems to me that the model has regressed. While relating Food and Activity at a higher level is a Good Thing, we also have exceptions that only an Animal (not AnimalClass) will consume. Likewise there are many Foods that are consumed by Animals (not AnimalClass).

- Resolved: the consumer will consume Food, not FoodType.
- Resolved: the consumer will be Animal, not AnimalClass.
- Resolved: the consumer will be Animal, not AnimalClass.
- Resolved: likewise the Animal, not AnimalClass will occupy itself with an AnimalActivity.
- Revert to model [3] on that issue, three pieces.

This model [4] further exposes the first issue, the closeness of Animal and Food. `Lion consumes gazelle`, and `Gazelle consumes some_bush`, and `Some_bush consumes carbon dioxide`.

Education. The level in the hierarchy, where animal products and plant products are the same Type of Thing, is **Taxonomy**. So we better get a handle on that, before proceeding:

Taxonomy • Quick Tour

- You don't have to read the whole page. Go to "The categories are", understand the 8 levels, and look at the pictures.

Enough for one day, five Steps, four progressions in the model and one research item. We will cover the remaining iterations tomorrow.