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

How to reference groups of records in relational databases

Response 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 thing, subject, a noun. What we do with it is an activity, a verb.

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)

In order to respond to that properly, a couple of concepts need to be discussed.

Relational Theory

First, a quick tour of the relevant parts of the *Relational Algebra* that apply here.

The *Relational Model* is based on **First Order Logic**. Without providing a full explanation, basically, this is what it means:

- Each proposition is a single sentence, in technically precise English, that evaluates to true or false.
- The propositions are Predicates, the terms are interchangeable.

1. The database is **defined** (modelled, and subsequently implemented) as a set of declarations. These declarations are Predicates.
   - In this context, the database is be said to be the implementation of a single set of Predicates, and nothing else. Of course those Predicates have to be thought out carefully, they have work together.
2. The database is **queried** (read) evaluating Predicates. There are two levels:
   a. The first is the Predicates that have been implemented, that should be obvious.
   b. The second is any Predicate that can be logically derived from those implemented Predicates. That is where the real power of the *Relational* Model is exposed, by ‘power’ users (unfortunately not developers), who understand the data and understand Predicates.
   - These Second Order Predicates are often implemented as Views.
   - The point is, if the data is Relational, not only the planned queries, but the unplanned queries, can be serviced using a single `SELECT` command.
   - Commonly, any query is either a single implemented Predicate, or a series of Predicates chained (`JOINed`) together, or some subset thereof.
3. Originally (1970) it was decreed that the database shall be **written** to using Predicates, only. However, that is obsolete for two important reasons:
   - Through the 1970’s and well into the 1980’s, on both mainframe and mid-range computers and the [pre-relational] DBMS platforms for them, we had and enjoyed ACID Transactions for OLTP systems.
   - As of Sybase in 1984, the first genuine Relational DBMS platform, with true SQL, and true OLTP, we have had ACID Transactions in SQL. Other vendors followed with SQL platforms, but only IBM followed with true SQL and true OLTP. (The recent range of freeware pretend-sqls do not figure in this.)

- That allows implementation of **Open Architecture Standards**. The database is completely independent of the applications, and it is positioned as being central to the organisation, not a slave to a single application. The database hosts any number of applications and report tools, it can be accessed with confidence and security. The converse is a closed system, a non-architecture (not a "closed architecture").
- Transactions that are designed properly (OLTP Standards, ACID properties) and packaged properly, as procedures or functions *within* the database, are declared constraints. The "Database API" is a set of declared Transactions.

a. Therefore the concept of writing to a database via Predicate, does not apply, it is not used. More precisely, it is used, but only by imbeciles who are evidently quite ignorant of post-1980's software components and their deployment.
   - In the notional context of writing to a database via Predicates, a number of complexities develop. This is due to, as per [2.b] above, any logically valid derived Predicate may be used, these need not be limited to the implemented Predicates.
   - This means the data must be updatable via Vews. Since vendors cannot control the correctness of tables, let alone the correctness of views on those tables, the implementation of Updatable Views has been a squirming mess from 1984 to this day, which has never been resolved. For those poor souls who insist on closed systems, sans Transactions, Sybase and IBM have provided an `INSTEAD OF` capability, but technically, that is an Extension, outside SQL.

b. Thus Updatable Views, or writing to a database via Predicate, is a very backward and ignorant thing to do.
   - The "theoreticians" in typical schizophrenic form, reject most of the *Relational Model*, and market their pre-1970's, pre-relational ISAM Record Filing Systems, but retain the obsolete parts and wave it around like a flag, to demonstrate how "relational" they are.

- In any case, given that it is 2015, and that Open Architecture with ACID Transactions should be the Standard for every implementation, Updatable Views are completely unnecessary.

Predicate

If you understand the above, you will appreciate the the entire Relational database is defined by Predicates, and nothing else. But we don't usually speak of it as such. There are many types of Predicates, and most of them are constraints, they are discussed as such: the logical or physical implementation object, eg. keys and indices; that a table exists; all the structures that provide the database container. The Predicates that are commonly discussed as Predicates are those that relate to using the tables and relationships, as opposed to the container, the constraints.

> Most of the Predicates are implemented as Constraints, and these constrain the **domain** of data, in a column or a table. The term *domain* is directly from Codd's *Relational Model*, unfortunately it is not understood by the famous authors and "theoreticians", it is not explained in their books, and thus it is not widely understood.
>
> The *Relational Model* defines three Normal Forms that are suppressed by all post-Codd authors.
>
> While the "theoreticians" have produced mathematical definitions for a number of hysterical things, such as tiny fragments of the Normal Forms, they have not been able to produce a mathematical definition for either a complete (ie. un-fragmented) Normal Form, or the three Normal Forms in the *RM*.
>
> There a ample evidence that this Relational Database industry has not been served by theoreticians since the great Dr E F Codd. I am supported by a small mountain of evidence, which I will not enumerate here. Forty five years have paased, and not a millimeter of advancement. If not for the hign-end vendors, we would not have the full Relational capability that Codd defined in the *RM*.
>
> Thus the ultimate goal of the *Relational Model* is Domain Key Normal Form, where the Integrity and Consistency of all data is maintained by virtue of constraints on Domains and Keys, only. All my databases qualify as such, much to the chagrin of the "theoreticians", who flatly state that that is not possible. It should be noted here that they do not understand or implement Relation Keys, therefore it is not possible for them to theorise about higher order concepts which are built on top of the foundation of Relational Keys.
>
> No surprise, consistent with the reasons given above, the "theoreticians" have not produced a definition for the full DKNF as intended. They have produced one by that name, but it is a schizophrenic, tiny fragment of the intended and possible thing. Due this confusion, and the misinformation marketed by the "theoreticians", I don't declare my database to be "in DKNF", I simply define it in the documentation.

There are many types of Predicates. Please refer to **Predicate** for full details. The Predicates we are most interested in , that we use the most in discussions, are those that relate to Existence (establishment of a Fact) and relationships (how Facts are related to other Facts).

Your Predicate

"An animal named x eat y for meal."

The problem here is, your propositions are starting to get complex.

- Each proposition is a single sentence, that evaluates to true or false.
  - Yours is getting into two propositions in one sentence. Break that up into two sentencee, and understand that they can be chained. Typically SQL queries are a chain of Predicates that are evaluated.
- In the data model, we declare only the Predicates, that we have determined to be relevant, that we have implemented in the database, not the chains or the derived ones, which in any case, can be easily determined.
- The structure of the Predicate is:
  - **Existence**

    `Subject2 is { Independent | dependent on Subject1 }`

    `Subject1` and `Subject2` are tables
  - **Identification**

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

    `Subject Action Cardinality Object`

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

     `Cardinality` as per `Object` end
- Note that the Relationship Predicates or relations can be expressed in converse form, this especially important for users (as in the documentation, for discussions) and for beginners.
  - If I tell you that `Fred is Sally's father` (ala a FK Constraint definition), then you know from that one 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 "theoreticians" and "teachers" don't, they implement two relationships, and then occupy themselves with puzzling over the insanity that they themselves have created. And of course, they blame SQL for it.
- 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 legal 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 Meal
Each Meal is a consumption of 1 Food
Therefore Each Animal consumes 0-to-n Foods
```

Again, 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 concerns are beyond the definition of the data, the modelling exercise.

> 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.

The second point is, you are insinuating a grouping. That was your initial concern, that must be made explicit.

> 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 !

Certainly a large slice of Aptitude is required to build a good database, and it is partly an Art, of doing all tasks well, such that the components are all integrated with each other (otherwise they are fragmented, isolated, 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 uneducated non-scientists, failed accountants, 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 by the occupation forces.

Date, Darwen, Warden, Fagin, Pascal, Zaniolo, Ambler, Fowler, Kimball, are all ignorant of the science involved, of the *Relational Model*. But they write books, market them heavily, and propagate their pre-relational Record Filing Systems and their precious myths. People like you read them, and get confused, subverted. I have no sympathy for the “professors” who use their books as textbooks, without understanding, without knowledge of the *RM*, without determining the veracity of the author’s postulations.

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*, etc. 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 others produce models that are broken, or work badly. Those who read the books written in the post-Codd era produce Record Filing Systems, that are difficult to use and navigate, that have none of the Relational integrity, power or speed.

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

> Are all great systems similar?

Yes. The architecture and principles that define successful systems are clearly visible.

All failed systems are similar, too. The architecture and principles that define successful systems are visibly absent.

> 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, Descriptors of the Key, that can be 'read' from the model quite easily. The formal Predicate, that I don't bother to state is:

```
Each Animal is identified by ( Name )
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.

Data Hierarchy

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, we are in the Relational Data Modelling world here.)

Now please read my Answer, and the linked documents, re Data 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, which should be performed without any regard to the application or objects.

- Possibly relevant, if and when you get to write code, and creating objects. Rather than an object hierarchy, I would have just one object for Animal, containing all its 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. Abstraction is a valuable tool for certain tasks. The problem is, these days, theoreticians abstract themselves to such an extent 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. Your question was *so* abstracted that it did not have any meaning, 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, separate to their abject ignorance of the *Relational Model*, this abstraction to too many levels, where the meaning and context was lost, and then they tried to add *some* of meaning they lost, back in. Whereas I maintained the context, and limited my abstraction to sane levels, thus I never reached the insane point where my abstraction produced the problem. Since my humanity and my application of science prevents all such problems, specifically including their proposed problems, I don't need their proposed solutions.

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

However, they don't teach that any more, they teach only non-relational, pre-1970's ISAM Record Filing Systems. But labelled as "relational". That is how the *RM* is suppressed. Same as the mockumentaries that in fact propagate fiction, but allege that they are "histories", thus suppressing history. We live in cursed times.

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 hysterical restrictions that the imbeciles say it has), however you lack the fundamentals of (a) understanding the data, and (b) the modelling exercise. 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 need as initially expressed, how to implement groups for your example, as well as generically, 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. More than one modelling change is implemented in each increment (otherwise there will be too many steps). 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 from the perspective of 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 only, not for consideration duriing the modelling exercise. Eg. the code required is made hard or easy with a certain progression.

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 to Update 1, 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, flattened views (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 (shown in the model). 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: 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.

Principle

Before we go further into the solution, there are two principles of Database Design (as opposed to modelling, or Normalisation) that we are touching, that require exposure. In fact, you are aware of the first principle intuitively, you know something is seriously wrong with your colleagues' implementation; you are painfully aware of the consequential problems; your question seeks to understand grouping; but you cannot articulate it. Let me do that for you.

1. The principle is, **establish groups at the highest level possible**. This results in substantially fewer rows to establish the binary relation (between the group and the entity that is grouped), which in turn results in much less work for the server on every access to the data related to the groups.
2. The second principle is, **match the real world**. We know that a database is a collection of Facts, about the real world, scoped to the requirement of the enterprise, so we model the data such that it contains all the Facts that we need for the enterprise to operate, and nothing more.

   However when we record those Facts about the real world, we must match it as closely as possible.

   - That is to say, if eg. we are recording Addresses, do not record two address lines plus a State and Country, even if that is all the enterprise needs. Record the Address as closely as possible to the real world: Suite No; Street No; Lot No; PO Box No; Street Name; Street Type; Suburb; Post Code; etc.
   - For data items for which Standards exist, record the data in the Standardised form. Eg. instead of AnimalClass, observe and record Taxonomy.

This isolates the database from change. First, when new requirements come up, the standardised structure of the data is not likely to change, whereas any non-standard structure is. Conversely, if the minimum data were recorded, then as issues arise (eg. two address lines are neither explicit enough to compare against other addresses, nor are they manageable), the database has to be changed to handle that more explicit data. And it will keep changing, until the structure of the data matches the real world, until it is in standardised form.

This is the vehicle for future-proofing, it is not merely a cliche (it is to the uneducated, because they cannot articulate the method).

Step 6

The purpose of drawing your attention to Taxonomy, is to make sure that we implement the classifications that we need, as closely as possible to the real world, using one that is standard and well-understood. This affords us the confidence that the structure will not change with every incremental new requirement in the future.

The Relational Keys form the logical structure of the database, they are migrated to the child tables, they form the Data Hierarchies: we want that to be stable, such that we can build upon it.

Generic 5 Data Model

Therefore we introduce Taxonomy, in full replacement of both AnimalClass and FoodType.

- It provides the full Data Hierarchy (*Update 1, Hierarchy Type 2. Rows within One Table*), just as AnimalClass did, now with standard values.
- In order to produce the full Taxomony 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 Consumption (food) that a Taxonomy (class of animals) eats. Another simple recursive function.
- Such derived columns are placed in Views, as shown.
- The levels of the Taxonomy ("categories" in the link) can be readily determined from the number of commas or slashes, there is no need to store it.

The Alternate Keys provide uniqueness within the parent class.

I have retained Animal, to demonstrate the difference between Taxonomy, which is the classifier, and an instance of it. Gender is an attribute at that level. As such, it forms a group.

The Activity grouping remains at the Taxonomy level.

At this stage I am differentiating a Taxonomy that is a Consumer vs a Consumable. The correct method is to use Subtypes. Here it is a Non-exclusive Subtype: Taxonomy can be either Consumer or Consumable or both.

Step 7

If the Consumer and the Consumable can be the same Taxonomy, let's merge them, and model it, in order to evaluate it, to observe whether we gain or lose anything.

Generic 6 Data Model

We do not lose anything. The Subtypes based on Consumer and Consumable were superfluous. Paper is cheap, implementation is expensive, that is why we model data.

Because Consumer and Consumable are now in the same table, and we need relationships in both directions (ancestor vs descendant), we need the third type of Data Hierarchy (*Update 1, Hierarchy Type 3. Rows within One Table, via an Associative Table*). Each row establishes one Fact, but it identifies two relationships (the `Fred::Sally` Fact identifies that `Fred is Sally's father`, *and* that `Sally is Fred's daughter`).

Note again, check the Predicates for relevance and precision, whether they precisely declare the Facts we need, and whether the Facts match the real world.

Step 8

That iteration [6] looks very good, and it exposes one more thing, as iterations do. I do not agree with these Predicates:

```
Taxonomy[Consumable] nourishes 0-to-n [Taxonomy]Consumptions
Taxonomy[Consumer] consumes 0-to-n [Taxonomy]Consumptions
```

They are simply not precise enough. I do not agree that any level in the Taxonomy consumes, or is consumed. Only a Species, the eighth level in Taxonomy, is a Consumer or Consumable. This exposes the need to differentiate between the intermediate levels vs the eighth level.

Generic 7 Data Model

Here an Exclusive Subtype cluster is used to differentiate intermediate and leaf levels of the Taxonomy. The leaf level is of course, a Species.

The Predicates are now precisely correct.

```
Species[Consumable] nourishes 0-to-n Species[Consumptions]
Species[Consumer] consumes 0-to-n Species[Consumptions]
```

This model also allows another improvement: only the intermediate levels of Taxonomy have children.

Note that Animal remains an occurrence of Taxonomy, with possible grouping at that level.

That model could be final.

Summary

You touched on a number of specific subjects, each of which is an essential part of Relational Database design. I trust that I have explained and demonstrated them adequately:

1. Relational Data Modelling
2. Important aspects of Relational Theory
3. The exercise of modelling; the iterations
4. Predicates; their use; their value
5. The importance of using Predicates to verify the model
6. Data Hierarchies; their ordinariness
7. All three types of Data Hierarchies
8. One specific implementations of each type of Data Hierarchy
9. Relational keys; their use
10. Use Subtypes, both Exclusive and Non-exclusive
11. Determining and implementing the Facts that we need
12. Recording those Facts such that they match the real world
13. Implementing Standard definitions of such Facts, where they exist
14. Proper methods for grouping data
15. Doing so at the highest level in the hierarchies
16. Exemplary use of Naming Convention (without explanation).

In other words, the brilliance of the *Relational Model*, sans subversion, sans perversion, san insanity, sans suppression.

Please study each Step carefully, and comment.

And please feel free to give me further detail re your Specification data, I will provide a progressed model.