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

How to reference groups of records in relational databases

Response to Update 4

My answers have triggered new questions for you. I can't give a full discourse re the Right and Wrong ways of doing things, that requires formal education. However, I will respond in point form.

> Thanks, Derek Asirvadem.
>
> After reading Response to Update 3, the previous answers you provided make more sense to me now. I really appreciate the power and intuitiveness of data modeling using the Relational Model.
>
> I have some ideas after reading the **Taxonomy • Quick Tour**. Please correct me if there is some misunderstanding.
>
> **1. Classifier**
>
> We, humans, have been using taxonomy all the time although most of us don't realize that.
> To understand the world, we create, differentiate things/concepts and group them.
> Now I know relational data modeling is doing the same thing.

I wouldn't put it in those words, but yes. The *Relational Model* allows us to classify and define data in the highest possible manner, to be useful in the highest possible way … and we have a need to classify data, thus we use the *RM*.

> When defining a table (relation, entity, subject), we are actually grouping things.
>
> For example, an Animal table is a set of things that are animals as opposed to other things.

Yes.

> In this sense, the "classifier" (group name) is the table name (or entity type, subject type).
>
> However, the classifier is not reflected in the data.

Now if you are concerned about the label I have given for the Taxonomy table, consider this. The Standard naming convention is that tables are named for the row that it contains, notice that is true for all the other tables. But in the case of Taxonomy, it contains rows for `Kingdom; Phylum; Class; Subclass; Order; Family; Genus; Species`. So I have to go beyond the convention, and I have named it for its collective content.

However, I will follow your line of thinking. The Animal table does not have any content "`animal`". But is contains nothing but Animals. So what. The petrol tank does not have a label "Petrol".

> In order to represent information about Animal as a group (type, classifier), we need to add a new subject for Taxonomy (or AnimalType, Species) which is a true classifier in ordinary use.

No.

Following your previous logic, and the data model I have supplied, no. We have already classified Animals, using Taxonomy, It gives us eight levels of an Hierarchy of classifications for Animals. We now have the structure required, such that we can state that all Animals of a certain `Order` live in water, without lungs, or that all Animals of a certain `Family` have an anal fin, or that a certain `Genus` has horns.

So thus far, given the "use" that has been identified, the "true classifiers for ordinary use" are the eight levels of Taxonomy.

One example of use of those classifiers is, that we have separated the leaf level, `Species`, because only that can be instantiated (hate the word) as a physical Animal, the other seven levels are purely for grouping, they do not exist in the real world , as Animals do.

> **2. Root of EVERYTHING**
>
> Kingdom, the top level of the Taxonomy, may not be on the top level.
> It is the root of the "Tree of Life", but not the root of everything.

`Kingdom` was the highest level for the data previously given. With the new data requirement, sure, there may be more "things", and some of them may be "higher" than Taxonomy.

> Everything is a thing.
> The ideal root may be "thing" (or "entity", "object").
>
> Is it necessary to model the root of everything?
> e.g.
>
> Thing (ThingNo)

Taxonomy (ThingNo, TaxonomyNo)

Animal (ThingNo, TaxonomyNo, Name)

Whoa.

One principle you mentioned for data modeling is to match the real world.
Given that the database stores only a small part of facts about the real world rather than the whole world, we don't need a Thing to represent everything.

The conclusion is correct, but that is not the reason.

Also, ThingNo is almost useless in real world, so it is unnecessary to do so.

Yes.

But when it is useful (just assume it is) in some scenario, shall we do it this way?
It just makes me comfortable to have a concept such as Thing so that we can understand/model the world from top down.

Three answers, from quite different perspectives:

1. Single Hierarchy.
   a. It is quite necessary to understand that there is a root of everything, but it is not necessary to model it. We do have a single hierarchy (starting from the root of everything), but we do not need to model that specific Fact.
      - Eg. take the Taxonomy table. The root row is `Kingdom`. By virtue of the defined model, every row identifies a parent row (`Taxonomy is a constituent of 1 TaxonomyIntermediate`). `Kingdom` is the one exception, it does not have a parent, the FK is zero.
   b. We need to model only the Things that we need to record Facts about. Yes, each of those Things stands as a Group.
   c. The entire database taken together is **the** "everything", so one might say the database container itself is the root.
   d. Codd named it the **Domain of Discourse**.
   e. You can therefore take it that the database, the Domain of Discourse, is the root of everything, the uppermost grouping, and that each of the tables (containing different Things) are lower-order groupings, that we need to store Facts about.

   Thus we know there is a root of everything, but it does not have to be modelled as a single hierarchy, the `ThingNo` does not have to be an Identifying or Non-identifying Foreign Key in every table. (Ie. separate to it being useless.)

   Consider **Generic 8 Data Model**, obviously as a progression from Generic 7.

2. There is nothing to prevent anyone from performing any task related to the database, once it exists. One is free to perceive whatever is in the database, the Domain of Discourse, as a bunch of Things. One can obtain the precise enumeration of the Things are in the database. This is an understanding (as detailed above), a perception, therefore it is a View or derived relation, as shown below the data model. This does not have to be implemented (ie. in physical form) in the database (which was your attempt). Every DBA has some form of this, it is an essential part of administering a database.

```
SELECT "Taxonomy", COUNT(*) FROM Taxonomy
UNION
SELECT "Person",   COUNT(*) FROM Person
UNION
SELECT "Animal",   COUNT(*) FROM Animal
etc
```

3. If you take the Root of Everything concept literally, then yes, you can reduce the structure (the Facts, Normalised, modelled) of all data to a set of simple Facts: its Identifier; its Attributes; and its Relationships.

d. Note that this is not the ultimate in Normalisation.

e. This is the ultimate in Reduction.

   Maslow[1] wrote an excellent paper on the dangers of too much reduction. It is the same as too much abstraction. While reduction and abstraction are valid and useful methods in science, too much of it, the point where the meaning is lost, is dangerous, and has many negative effects.

   I am saying that a scientist must be aware of this boundary, and not cross it. The "theoreticians" of today are *Unskilled and Unaware*[2], they only have a Hammer, they operate entirely within that danger zone. Intelligent imbeciles create monsters, just look at the fact that 95% of the databases created by people following these imbeciles, are Record Filing Systems, not Relational database..

f. If you do reduce the model to such a level, you will not be the first, nor the last. It has been done many times.

   Consider **Ultimate Reduction**.

g. When it is done properly, a set of views is provided, one for each of the discrete Things that exist in the Domain of Discourse, the database, in this case, in the one table.

h. The reason it is not famous is that it is always a failure, because meaning is lost at that level of reduction, and it is very difficult to use or code against. For sanity reasons, people need, for both ordinary reporting and for coding apps, a set of Things that are organised and modelled as separate Things.

However, adding a ThingNo column (or more) to every table seems awkward.

Unnecessary, if you understand the above.

Even if you had a Thing table, you need a `ThingNo` Identifier (as an Identifying or Non-identifying Foreign Key) only in those tables that require it, not in every table.

3. Meta-data

In most databases, there are tables that describes tables (catalog, meta-data).

In my opinion, meta-data is just ordinary data that serves as a description of other data.
There may be data describes "meta-data" too. The meta-data, meta-meta-data, or meta-...-meta-data form a hierarchy similar to the Taxonomy.
Can we say "meta" is just a kind of classifier that is used to describes the purpose/meaning of the data?

No, we can't say that. That is (a) too reductionist, past the point where meaning is lost, and (b) not the full difference between data and meta-data. While it is a truth, it is only one truth amongst many truths.

One cannot design anything or form reasonable conclusions about anything when one is hanging on to one truth, and denying all the other truths. That is precisely the schizophrenia (obsession with one fact , while denying other facts) that is the "normal" state for the "theoreticians" of today. And note gravely, they teach it. Which is why you have ended up thinking that that is somehow a valid reasoning.

The Thing seems like a kind of meta-data too.
But Thing is beyond meta-data. It is the root of everything.

Anyway, meta-data table has a lot in common with Thing.

So what. People have a lot in common with apes. But we are not created as animals. People have a lot that Animals do not have. Meta-data is very different to data.

In Oracle, for example, information about the tables is stored in user_tables whose role is similar to Thing, where table_name is like the ThingNo.

By the way, that is not something that Oracle alone does, it is a requirement of the ISO/IEC/ANSI SQL Standard, that the objects (etc) in the database are visible to the user, and that that visibility is in the form of tables.

No table includes the table_name column because it is itself a table identified by its name.
This eliminates a table_name column for all tables, which saves space.

---

1 Abraham Maslow is famous for the ***Hierarchy of Needs***, and the ***Law of the Instrument*** (more commonly known as the *Maslow's Hammer* or *the Hammer Principle* "if all you have is a hammer, every problem looks like a nail"). He contributed much to psychology, in the early years when it was a science, before it was hijacked and perverted, same as Codd and the *Relational Model*, before it was hijacked and perverted.

2 In 1999, Justin Kruger and David Dunning wrote a famous paper ***Unskilled and Unaware of It***, that should be required reading for any professional. The "theoreticians" attacked some of the content, using their usual pharisaic "logic", so in 2008 Ehrlinger, Johnson, Banner, Dunning, and Kruger wrote a second paper ***Why the Unskilled are Unaware***, to close those "holes" and to provide supporting material.

(The space saving is a minor issue, not a goal that was sought. We no longer think is kilobytes or megabytes, we think in gigabytes. SQL platforms are heavily optimised for Relational data. Although Key width used to be a problem, it is no longer.)

In the Ultimate Reduction model, which has just one Primary Key, one could prefix that with table_name, .

> The downside(?) is that this info is not reflected in the data. The person who writes SQL should know which table means which thing.

Well, in the normal case, the table is named for the rows it contains, so that is not an issue. In the Thing case, or the Ultimate Reduction case, that is precisely one of the problems that causes it to fail. Even though this can be mitigated by supplying views for every Thing; group; collection; would-be-table.

But there is an overall problem in your thinking, your approach. I have noted two separate instances of it here, and one in the other question.

Principle

There is a principle that you may not be aware of, that differentiates data vs meta-data; Data vs Process. Certainly, by the evidence in that paper and elsewhere, the theoreticians are unskilled and unaware of this principle, they teach the mixture of Data vs Process in an object; they cannot keep data vs meta-data separate, and the consequence is guaranteed: failure. But they teach it anyway, they write books, and market and propagate the insanity. You are a victim of that "education" system.

- Why are the nerves separate from the muscle tissue, that the nerves control ?
- In an oil pipeline, why are the control mechanisms (wiring, etc) separate from the pipe containing the oil ? (It would be much cheaper to provide the wiring within the pipe.)
- In Wifi or RFID, why is the control (program) information always on a separate frequency to that of data ?

We have observed this principle, that exists throughout nature, for millennia. We have articulated it, and made use of it, since the Middle Ages (ok, not in primitive cultures). It can be observed readily in any man-made object that does not fail, and its absence can be observed in that which fails. And notably, these days, with more IT projects failing than succeeding, it absence has become "normal". We have regressed two thousand years, or to the state of primitive tribesmen.

The principle is **separation of control from that which is controlled**. In the IT space that translates to control vs data; in the database space, to program vs data.

- We simply cannot afford to allow the data to contaminate the mechanism that controls the data, even by accident. One can imagine the effect on all users, the consequences, if the corruption of one user's data (accidental or otherwise) caused the machine to crash. Pipeline fires are prevented (in part) by physically separating the electrical controls from the oil.
- Further, the access and security placed on the control mechanism is quite different to that placed on the data. Eg. the catalogue and meta-data is updated only by qualified administrators, and read (indirectly) by all users, whereas the data is read and written according to the access control that is contained in the catalogue, the meta-data.
- Everything in the world that is transmitted, that does not break, uses two channels, one for control, the other for data.
- The systems that break use one channel. The OO/ORM model, makes no separation between the process object and the data contained in the object.
- The denial of this principle is partly based on the denial of authority, and the hilarious notion of "equality". Data is not equal to the program that controls it. The catalogue is the authority than controls the content, they are not equal. DDL is different to DML. They cannot be mixed up and served up together.

Ignorance of this scientific principle (it is science), or worse, subversion of it, is an universal problem in these dark days of IT. I have detailed (a) the abject ignorance of this principle in the OO/ORM world, (b) the expensive consequences, the total failure of such systems, as well as (c) the solution, in my **Response to Hidders**, please read that document again, carefully. (I have updated and clarified the document 06 Jun 15.)

Dead Letter Office

There is one item that may appear to be an exception, but it is not, it is a different class. PDFs, good XML documents, etc, can safely contain both meta-data (that controls the display) and data (that is displayed). The distinction is, these are fixed, dead, products, of an active, live, process that produces them. The principle applies to systems, not to the products, documents, message packets.

- Throwing the electrical controls into a barrel of oil (ie. not flowing in the live pipeline); grinding up nerves and muscle (from a carcass) together to make hamburger; drawing the control elements and data elements on the same diagram, causes no problem, they are static, products, not the live system that produces them.
- The implication re objects that control the data that they contain, should be noted.

Application

Therefore the control mechanism must be separated from things that it controls. The catalogue and meta-data are in one class with one set of security, and the data that it controls is in another class with a different set of security. even though they reside in the same database container.

While it is true that "meta-data is a kind of data", that is not the only truth, or the governing truth. The governing truth is higher-order principles, science: meta-data is a different class of data because it controls data. At the cellular level nerve cells are very similar to muscle cells, but at the visible and electrically active level, nerves are clearly and cleanly separated from the muscle tissue that it controls.

Even in the Ultimate Reduction case, the catalogue and meta-data are separated from Things; their attributes and their relationships. The catalogue and meta-data control the structure of the database, and the production of the views[3], the container. The data, Things, provide the content in the database, and in the views.

Summary

The Root of Everything is a great concept:

- Foremost for the purpose of understanding, both of the world in general, and of the Domain of Discourse
  - Such an understanding will allow you to form groups at the highest level, as per the design principle.
- It need not be implemented
- If it is implemented:
  - First it must be separated into Control vs Data, catalogue plus meta-data vs tables
  - Second, in each of Control vs Data, either a single root (Ultimate Reduction) or multiple first level roots, with the single root taken as the Domain of Discourse, the database (Generic 8)
  - Third, the content, the data can then be poured in.

3 In my systems that are similar, the production of views is automated, directly from the catalogue and meta-data.