How to reference groups of records in relational databases?

[+2] [2] dzhu

[2015-04-24 02:50:08]

[relational-database database-normalization referential-integrity]

[https://stackoverflow.com/questions/29837901/how-to-reference-groups-of-records-in-relational-databases]

Suppose we have the following table structures:

Humans

```
| HumanID | FirstName | LastName | Gender |
|---------+-----------+----------+--------|
|       1 | Issac     | Newton   | M      |
|       2 | Marie     | Curie    | F      |
|       3 | Tim       | Duncan   | M      |
```

Animals

```
| AmimalID | Species | NickName |
|----------+---------+----------|
|        4 | Tiger   | Ronnie   |
|        5 | Dog     | Snoopy   |
|        6 | Dog     | Bear     |
|        7 | Cat     | Sleepy   |
```

I wonder how to reference a group of records in other tables.

For example, I have a Foods table and an EatenBy column.

Foods

```
| FoodID | FoodName | EatenBy |
|--------+----------+---------|
|      8 | Rice     | ???     |
```

What I want to store in EatenBy may be

1. **a single record** in the Humans, Animals tables (e.g., Tim Ducan)
2. **a group of records** in a table (e.g. all dogs, all males, all females)
3. **a whole table** (all humans, for example)

A simple solution is to use a **concatenated string**, which includes primary keys from different tables and special strings such as 'Humans', 'M'. The application could parse the concatenated string and do things accordingly.

Foods

```
| FoodID | FoodName | EatenBy       |
|--------+----------+---------------|
|      8 | Rice     | Humans, 6, 7 |
```

I know using a concatenated string is a bad idea from the perspective of relational database design.

Another option is to add another table and use a **foreign key**.

Foods

```
| FoodID | FoodName |
|--------+----------|
|      8 | Rice     |
```

EatenBy

```
| FoodID | EatenBy |
|--------+---------|
|      8 |  Humans |
|      8 |       6 |
|      8 |       7 |
```

I think it's better than the first solution. The problem is that the EatenBy field stores values of different meanings. Is that a problem?

What is the best way to model this requirement? How to achieve 3NF in this case?

The example tables here are a bit contrived, but I do run into situations like this at work. I have seen quite a few tables just use a concatenated string. I think it is bad but can't think of a more *relational* way to deal with it.

Quick Answer

For those who want to know the answer for this problem quickly. Here're the basic ideas:

1. Concatenated strings, CSVs, repeating groups should not be used in the *Relational Model*. So we need to **normalize** the table, eliminating repeating groups.
2. In this problem, the concatenated string contains values of different meanings.
   Relational databases store facts about the real world. Each table should store facts about ONE subject and each column should store **ONE fact**.
   Thus, we need one **associative table** for each fact to fix "EatenBy".
   e.g.
   Food_Human (FoodID, HumanID)
   Food_Animal (FoodID, AnimalID)
   Food_Species (FoodID, SpeciesID)

Caveat

The ideas to solve the problem is good. However, the data model here is awful.

The obvious problems:

1. Humans are Animals
2. Using surrogates (IDs) during data modeling is bad practice

(1) Are your *Things* honestly only humans; animals; and food ? - **PerformanceDBA**
(1) Let's try a different technique. Perhaps you could (a) name all your *Things* (b) name all activities, such as *EatenBy* (c) list all your classifiers eg. *Gender, AnimalType, etc.* - **PerformanceDBA**
Similar to the "Catch Fish" example, humans and animals can be both represented as animals. Just as fishers catch fish, animals eat foods. How about creating a new entity like this: Eat (AnimalType Animal Food)? If we create a table for Eat, will the table contain too many records when all humans eat a food? - **dzhu**

(1) (a) *Too Many recds* No, not in a RDb, we have unduplicated **rows**. (Yes, in a RFS that has 1 record per whatever, and masses of duplication). (b) So it is looking like you want a "map" between *Things* and *Activities*, where the *Thing* could be the subject or the object. (c) *Eat* is a verb, why would we need verbs as entities ? Verbs are joins between entities. Refer to the linked post re **real** Predicates, the ones you were taught are violently not Predicates. That's why I asked for a list of *Activities*. - **PerformanceDBA**

(1) That is no problem at all, but it sounds too simple. Eg. (a) How are the Specs grouped, by Project ? (b) Then are Projects and UserGroups the same, or different ? (c) Does the User who creates/edit a Spec get access to it (even if they are not in the Group that the spec belongs to) ? (d) Does an User belong to one Group or many ? (e) I take it **AccessibleTo** is either an User xor a Group ... in which case, *A super user specifies whether a spec can be accessed by a user or users of a specific group* is false or confused ? - **PerformanceDBA**

I have responded to the update in my Answer. Next time you update your question, ping me, via a comment, starting with @PerformanceDBA. - **PerformanceDBA**

What I wanted to know at first was exactly a way to create a "map" between Things and Activities, where the Thing could be the subject or the object. Now I realized that it is not possible in the relational model. A relation can't be the value of an attribute. - **dzhu**

(1) a) That *xxxx is not possible in the RM* is OO/ORM nonsense, just like all the other pathetic nonsense eg. *the RM doesn't support hierarchies*, etc, that they write. There is nothing that cannot be modelled in the RM. But it will take a few progressions to get to the point where you can understand it. - **PerformanceDBA**

(1) Rather than add *Update* headers, can you please consolidate the 'updates' into one coherent whole? Your question needs to be readable and usable for future visitors too, and the edit history doesn't matter to them. For those interested how the final version came about there is always the edit history, visible to everyone. - **Martijn Pieters**

@MartijnPieters. Thanks, Martijn. I will find a way to make the question more readable. - **dzhu**

@PerformanceDBA. Any idea how to *normalize* the **updates**? To clear up the bloated question, we can : a) consolidate the updates into one coherent whole, as MartijnPieters suggests; b) move them to separate answers c) use other methods. - **dzhu**

This is unclear. (That is why the reams of comments on this page.) What does "reference a group of records in other tables" mean? What is your question? It's not even clear whether you are addressing database design or displaying results. PS Please ask one question per question. Please clarify via edits, not comments. PS There is nothing wrong per-se with relation-valued attributes. - **philipxy**

[+4] [2015-04-24 16:08:50] PerformanceDBA [✓ACCEPTED]

1. This Answer is laid out in chronological order. The Question progressed in terms of detail, noted as **Updates**. There is a series of matching **Responses**.

2. The progression from the initial question to the final answer stands as a learning experience, especially for OO/ORM types. Major headings mark Responses, minor headings mark subjects.

3. The Answer exceeds the maximum length exceeded. I provide them as links in order to overcome that.

Response to Initial Question

You might have seen something like that at work, but that doesn't mean it was right, or acceptable. CSVs break 2NF. You can't search that field easily. You can't update that field easily. You have to manage the content (eg. avoid duplicates; ordering) manually, via code. You don't have a database or anything resembling one, you have a grand Record Filing System that you have to write mountains of code to "process". Just like the bad old days of the 1970's ISAM data processing.

1. The problem is, that you seem to want a relational database. Perhaps you have heard of the data integrity, the relational power (Join power for you, at this stage), and speed. A Record Filing System has none of that.

   If you want a Relational database, then you are going to have to:

   - think about the data relationally, and apply Relational Database Methods, such as modelling the data, as data, and nothing but data (not as data values).

   - Then classifying the data (no relation whatever to the OO class or classifier concept).

    - Then relating the classified data.

2. The second problem is, and this is typical of OO types, they concentrate on, obsess on, the data *values*, rather than on the meaning of the data; how it is classified; how it relates to other data; etc.

   No question, you did not think that concept up yourself, your "teachers" fed it to you, I see it all the time. And they *love* the Record Filing Systems. Notice, instead of giving table definitions, you state that you give "structure", but instead you list data values.

    - In case you don't appreciate what I am saying, let me assure you that this is a classic problem in the OO world, and the solution is easy, if you apply the principles. Otherwise it is an endless mess in the OO stack. Recently I completely eliminated an OO proposal + solution that a very well known mathematician, who supports the OO monolith, proposed. It is a famous paper.

    - I *relationalised* the data (ie. I simply placed the data in the Relational context: modelled and Normalised it, which took a grand total of ten minutes), and the problem disappeared, the proposal + solution was not required. Read the **Hidders Response** [1]. Note, I was *not* attempting to destroy the paper, I was trying to understand the data, which was presented in schizophrenic form, and the easiest way to do that is to erect a Relational data model. That simple act destroyed the paper.

    - Please note that the link is an extract of a formal report of a paid assignment for a customer, a large Australian bank, who has kindly given me permission to publish the extract with a view to educating the public about the dangers of ignoring Relational database principles, especially by OO proponents.

    - The exact same process happened with a second, more famous paper **Kohler Response** [2]. This response is much smaller, less formal, it was not paid work for a customer. That author was theorising about yet another abnormal "normal form".

Therefore, I would ask you to:

- forget about "table structures" or definitions
- forget about what you want
- forget about implementation options
- forget `ID` columns, completely and totally
- forget `EatenBy`
- think about what you *have* in terms of data, the meaning of the data, *not* as data values or example data, *not* as what you want to do with it
- think about how that data is classified, and how it can be classified.
- how the data relates to other data. (You may think that your `EatenBy` is that but it isn't, because the data has no organisation yet, to form relationships upon.)

If I look at my crystal ball, most of it is dark, but from the little flecks of light that I can see, it looks like you want:

- Things
- Groups of Things
- Relationships between Things and ThingGroups

The Things are nouns, subjects. Eventually we will be doing something between those subjects, that will be verbs or action statements. That will form Predicates (First Order Logic). But not now, for now, we want the only the Things.

Now if you can modify your question and tell me more about your Things, and what they mean, I can give you a complete data model.

Response to Update 1 re Hierarchy

Record IDs are Physical, Non-relational

If you want a Relational Database, you need Relational Keys, not Record IDs. Additionally, starting the Data Modelling exercise with an ID stamped on every file cripples the exercise.

Please read **this Answer** [3].

Hierarchies Exist in the Data

If you want a full discourse, please ask a new question. Here is a quick summary.

Hierarchies occur naturally in the world, they are everywhere. That results in hierarchies being implemented in many databases. The *Relational Model* was founded on, and is a progression of, the *Hierarchical Model*. It supports hierarchies brilliantly. Unfortunately the famous writers do not understand the *RM*, they teach only pre-1970s Record Filing Systems badged as "relational". Likewise, they do not understand hierarchies, let alone hierarchies as supported in the *RM*, so they suppress it.

The result of that is, the hierarchies that are everywhere, that have to be implemented, are not recognised as such, and thus they are implemented in a grossly incorrect and massively inefficient manner.

Conversely, if the hierarchy that occurs in the data that is being modelled, is modelled correctly, and implemented using genuine Relational constructs (Relational Keys, Normalisation, etc) the result is an easy-to-use and easy-to-code database, as well as being devoid of data duplication (in *any* form) and extremely fast. It is quite literally Relational at its best.

There are three types of Hierarchies that occur in data.

1. **Hierarchy Formed in Sequence of Tables**

   This requirement, the need for **Relational Keys**, occurs in **every database**, and conversely, the lack of it cripples the database ad produces a Record Filing System, with none of the integrity, power or speed of a Relational Database.

   The hierarchy is plainly visible in the form of the Relational Key, which progresses in compounding, in any sequence of tables: father, son, grandson, etc. This is essential for ordinary Relational data integrity, the kind that Hidders and 95% of the database implementations do not have.

   The **Hidders Response** [4] has a great example of Hierarchies:

   a. that exist naturally in the data

   b. that OO types are blind to [as Hidders evidently is]

   c. they implement RFS with no integrity, and then they try to "fix" the problem in the object layers, adding even *more* complexity.

Whereas I implemented the hierarchy in a classic Relational form, and the problem disappeared entirely, eliminating the proposed "solution", the paper. Relational-isation eliminates theory.

The two hierarchies in those four tables are:

```
Domain::Animal::Harvest

Domain::Activity::Harvest
```

Note that Hidders is ignorant of the fact that the data is an hierarchy; that his RFS doesn't have integrity precisely because it is not Relational; that placing the data in the Relational context provides the very integrity he is seeking outside it; that the Relational Model eliminates all such "problems", and makes all such "solutions" laughable.

Here's **another example** [5], although the modelling is not yet complete. Please make sure to examine the Predicates, and page 2 for the actual Keys. The hierarchies are:

```
Subject::CategorySubject::ExaminationResult

Category::CategorySubject::ExaminationResult

Person::Registrant::Candidate::ExaminationResult
```

Note that last one is a progression of state of the business instrument, thus the Key does not compound.

2. **Hierarchy of Rows within One Table**

   Typically a tree structure of some sort, there are literally millions of them. For any given Node, this supports a single ancestor or parent, and unlimited children. Done properly, there is no limit to the number of levels, or the height of the tree (ie. unlimited ancestor and progeny generations).

   - The terms *ancestor* and *descendant* use here are plain technical terms, they do not have the OO connotations and limitations.

   You do need recursion in the server, in order to traverse the tree structure, so that you can write simple procs and functions that are recursive.

   Here is one for **Messages** [6]. Please read both the question and the Answer, and visit the linked **Message Data Model** [7]. Note that the seeker did not mention *Hierarchy* or *tree*, *because the knowledge of Hierarchies in Relational Databases is suppressed*, but (from the comments) once he saw the Answer and the Data Model, he recognised it for the hierarchy that it is, and that it suited him perfectly. The hierarchy is:

   ```
   Message::Message[Message]::Message[::Message[Message]] ...
   ```

3. **Hierarchy of Rows within One Table, Via an Associative Table**

   This hierarchy provides an ancestor/descendant structure for multiple ancestors or parents. It requires two relationships, therefore an additional Associative Table is required. This is commonly known as the Bill of Materials structure. Unlimited height, recursively traversed.

   The Bill of Materials *Problem* was a limitation of Hierarchical DBMS, that we overcame partially in Network DBMS. It was a burning issue at the time, and one of IBM's specific problems that Dr E F Codd was explicitly charged to overcome. Of course he met those goals, and exceeded them spectacularly.

   Here is the Bill of Materials hierarchy, **modelled and implemented correctly** [8].

- Please excuse the preamble, it is from an article, skip the top two rows, look at the bottom row.
- Person::Progeny is also given.
- The hierarchies are:

```
Part[Assembly]::Part[Component] ...

Part[Component]::Part[Assembly] ...

Person[Parent]::Person[Child] ...

Person[Child]::Person[Parent] ...
```

Ignorance Of Hierarchy

Separate to the fact that hierarchies commonly exist in the data, that they are not recognised as such, due to the suppression, and that therefore they are not implemented as hierarchies, when they *are* recognised, they are implemented in the most ridiculous, ham-fisted ways.

- Adjacency List

  The suppressors hilariously state that "the *Relational Model* doesn't support hierarchies", in denial that it is founded on the *Hierarchical Model* (each of which provides plain evidence that they are ignorant of the basic concepts in the *RM*, which they allege to be postulating about). So they can't use the name. This is the stupid name they use.

  Generally, the implementation will have recognised that there is an hierarchy in the data, but the implementation will be very poor, limited by physical Record IDs, etc, absent of Relational Integrity, etc.

  And they are clueless as to how to traverse the tree, that one needs recursion.

- Nested Sets

  An abortion, straight from hell. A Record Filing System within a Record Filing system. Not only does this generate masses of duplication and break Normalisation rules, this **fixes** the records in the filing system in concrete.

  Moving a single node requires the entire affected part of the tree to be re-written. Beloved of the Date, Darwen and Celko types.

  The MS `HIERARCHYID` Datatype does the same thing. Gives you a mass of concrete that has to be jack-hammered and poured again, every time a node changes.

Ok, it wasn't so short.

Response to Update 2

- **Response to Update 2** [9]

Response to Update 3

- **Response to Update 3** [10]

Response to Update 4

- **Response to Update 4** [11]

[1] http://www.softwaregems.com.au/Documents/Article/Application%20Architecture/UTOOS%20Response.pdf
[2] http://www.softwaregems.com.au/Documents/Article/Normalisation/DNF%20Data%20Model%20B.pdf
[3] https://stackoverflow.com/a/29726132/484814
[4] http://www.softwaregems.com.au/Documents/Article/Application%20Architecture/UTOOS%20Response.pdf
[5] http://www.softwaregems.com.au/Documents/Student%20Resolutions/lee%20TRD%206.pdf
[6] http://www.softwaregems.com.au/Documents/Student%20Resolutions/Ali%20-%20How%20to%20build%20a%20table%20for%20a%20private%20messaging%20system%20that%20supports%20replies.pdf
[7] http://www.softwaregems.com.au/Documents/Student%20Resolutions/Ali%20DM.pdf
[8] http://www.softwaregems.com.au/Documents/Student%20Resolutions/Bill%20of%20Materials.pdf
[9] http://www.softwaregems.com.au/Documents/Student%20Resolutions/dzhu%20Groups%20Update%202%20Response.pdf
[10] http://www.softwaregems.com.au/Documents/Student%20Resolutions/dzhu%20Groups%20Update%203%20Response.pdf
[11] http://www.softwaregems.com.au/Documents/Student%20Resolutions/dzhu%20Groups%20Update%204%20Response.pdf

Thanks for the direction and the links. I am enlightened by what you said. Could you expand on "hierarchies exist in the data"? - **dzhu**
@dzhu. (a) You are welcome. (b) Are you sure that you have read the linked docs, and understood them ? (c) *Hierarchies* I have responded in my Answer. - **PerformanceDBA**
@dzhu. PS. If you found **this Answer** re *Hierarchy* to be of value, please vote, and vote to undelete the Question. The system here is "democratic". - **PerformanceDBA**
(1) Thanks for your answers and great patience! It's a pity that the question has been deleted already. I am not sure whether I really understand the linked docs. I added my understanding to the question. - **dzhu**
Thanks for the Specification Data Model. I finally find my peace of mind. SpecificationUser and SpecificationGroup are two different relations and should be represented in two entities, rather than put in an AccessibleTo table together. I realized that many of the normalization problems I met are caused by focusing on record values. - **dzhu**
The real application is more complicated. Groups are hierarchical and implemented using Adjacency List, i.e. Groups (pkid, fkGroup, ...). A group can be assigned roles, a user can be assigned roles too. Roles are involved when checking security. A child group can inherit roles from its parent group (`Inherit' can be turned on or off). - **dzhu**
Specs are grouped by SpecType (material, formulation, trade, menu, ...). Each spec type corresponds to a table, there is also a SpecSummary table which all specs share. The primary key looks like '214733f941e2-0639-4ed9-99e3-ef96a5a4a157', where '2147' identifies SpecType. It's a surrogate key, but also has business meaning. The app is designed using the ORM approach (home made ORM tool) ten years ago. Almost all business rules are handled in code. The database serves merely as a data storage. No foreign key constraints. - **dzhu**
I am not asking you to provide a complete Data Model for the application. It's not my intention to ask this question in SO. Also, I can't specify all the requirements clearly in a short time. I just feel the database is a mess. The code decays over time. Does all complex systems look like this? - **dzhu**
Different Specs are related to each other in various ways. When customers want to see the relationships hierarchically, they create a DENORM table to extrat the relationships using Nested Sets, i.e. Hierarchy(fkParent, fkAncestor, BoxLeft, BoxRight,...). This solves some problems, but when I creating reports, the performance still suffers because security checking and hierarchy are tangled up. - **dzhu**
I think maybe the app can perform better if we have a sound database structure. Clearly, the app is created to help customers manage data and the database should be carefully designed. Some teammates told me that relational databases are outdated, new systems use ORM, NoSQL, etc. I am relatively new to the industry and sometimes I am confused and not sure what I should learn to be a good developer. - **dzhu**
What do you think about these opinions. (a) Databases should deal with data only, not business logic (b) PL/SQL, T-SQL are less powerful than Java/C#, we have more flexibility when business logic is dealt with in Code. (c) Since we are using ORM, we code business logic in code, so the application can deal with different kinds of databases without changing code. - **dzhu**

(d) Requirements always change, there is no true primary key. Changing primary key has great impact, and changing code is easy, so a surrogate key should be used. [I interpret it as this: if we want to deal with a book, we don't create an entity to model the book. Instead, we create a class (or table) first, it can represent anything, we give it a name "Book" and only put books in it (the code checks whether it is a book, the code prevents duplicates)]. - **dzhu**
(e) Database design is time consuming, code first and Agile development is more productive. - **dzhu**
All arguments sounds true in a sense, I don't have enough experience and wisdom to tell the right from wrong. Could you share your ideas with me? - **dzhu**
(2) I have gained more than an answer by asking my first question in SO. I am sure I will learn more. Thanks again. - **dzhu**
(2) @dzhu. *Deleted Question* Please go to that **Question**, and (a) vote to undelete it (b) vote to re-open it. Thanks for choosing my Answer. If you like my Answers, please vote for it, both here and **there**. - **PerformanceDBA**
(1) @dzhu. That Answer exceeded the limits, so I have reformatted tit, and provided the further Responses as linked docs. I expect the finish the second part of the DM prgression tonight. I will not be able to attend again until Thurs. - **PerformanceDBA**
(1) Thanks. The responses and links you provide are very helpful. I will take time to digest them. - **dzhu**
Looking forward to the second part! - **dzhu**
(2) @dzhu. First, thanks for the choice and the vote. Second, I will definitely take this answer all the way to completion. Your question is a very important one, and the fact that you want a generic answer gives me the opportunity to provide the principles and considerations, instead of just a specific answer. I have modified five slides from my courses, making seven for this question, that was completed last week. I intended to finish the words to go with the slides this week, but humble apologies, I was just too busy. Next week for sure. Thanks for your patience, please stand by. - **PerformanceDBA**
(2) @dzhu. (a) Once again, I must beg for forgiveness re the delay in completing the words. (b) There are minor corrections to **DM 1** through **4** and **Response to Update 2**, nothing substantial, but it needs to be noted if you saved copies. (c) **Response to Update 3** is now complete, in a single document. First & second parts amalgamated. Please read from the top. - **PerformanceDBA**
(2) That's a lot of information! Thanks for your effort! I have updated my question. See Update 4. Sorry for the delay too. - **dzhu**
I have asked another question in SO regarding subtypes. See stackoverflow.com/questions/30568689/... - **dzhu**
@dzhu. Good question in your Update 4. It begs an Answer. I will attend as time permits. - **PerformanceDBA**
Thanks a lot, take your time. By the way, do you have the following papers by Codd: "Further Normalization of the Data Base Relational Model", "Normalized Data Base Structure: A Brief Tutorial."? - **dzhu**
@dzhu. Please go to my profile & email me for the Codd papers. - **PerformanceDBA**
@dzhu. I have submitted **Response to Update 4**. Please review. - **PerformanceDBA**
Thank you. I have some feedback in Update 5. - **dzhu**

1

[0] [2015-05-11 05:01:08] Ormoz

For each category who eats the food, you should add one table. for example, if one food may be eaten by some specific gender, you would have:

```
Food_Gender(FoodID,GenderID)
```

for humans you would have:

```
Food_Human(FoodID,HumanID)
```

for animals species:

```
Food_AnimalSpc(FoodID,Species)
```

for an entire table:

```
Food_Table(FoodID,TableID)
```

and so on for other categories

Thanks for your answer. I agree that extra tables are needed for the the "group" concept. - **dzhu**

2