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

How to reference groups of records in relational databases

Response to Update 1

Questions in the comments re Hierarchies.

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 a Record ID mindset, an ID stamped on every file, cripples the Data Modelling exercise.

Please read **this Answer**.

Data Hierarchy

Hierarchies Exist in the Data.

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 them, so they suppress it. Just like with the *Relational Model*, they do not understand it, so they teach pre-1970s Record Filing Systems badged as "relational". As a result of that, the hierarchies commonly implemented are grossly incorrect and massively inefficient.

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

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

   The hierarchy is most visible in the form of the Relational Key, which progresses in compounding. The Relational Key is essential for ordinary data integrity, the kind that Hidders and 95% of the database implementations do not have. This requirement, the need for Relational Keys, **occurs in every database**, and conversely, the lack of it cripples the database.

   The **Hidders Response** has a great example of Hierarchies:

   a. that exist naturally in the data

   c. that OO/ORM types are blind to them (as Hidders is)

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

   Whereas I implemented the hierarchy in a straight-forward Relational form, and the problem disappeared entirely, eliminating the proposed “solution”. Relational-isation eliminates Theory.

   The two hierarchies in those four tables are:

   ```
   Domain::Animal::Harvest
   Domain::Activity::Harvest
   ```

   Here's **another example**, although the modelling is not yet complete. Make sure you 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, thus the Key does not compound:

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

   This hierarchy is an ancestor/descendant structure (without the OO connotations and limitations), for a single parent (ancestor). It is a tree structure of some sort, there are millions of them (eg. the unix file system: a file belongs to just one folder). Done properly, there is no limit to the number of levels, the height of the tree. You do need recursion in the server, in order to traverse the tree structure vertically, so that you can write simple procs and functions that are recursive.

   Here is one for **Messages** (archived version of article deleted from host).

   - Please read both the Question and the Answer, , and visit the linked Data Model.
   - 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, and it suited him perfectly.
   - The hierarchy is:

     ```
     Message::Message[Reply] …
     ```

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

   This hierarchy too, is an ancestor/descendant structure, but allows multiple parents (ancestors). It requires two relationships, so 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 HDBMS, that we overcame partially in Network DBMS. It was a burning issue at the time, and one of IBM's problems that 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 according to the *Relational Model*.

- Please excuse the preamble, it is for another article. Skip the top two rows, look at the bottom row.
- 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

Due to the suppression of hierarchies in the 'education' systems, and in Modern 'science', the hierarchies that do exist in nature are not recognised as such, and therefore they are neither modelled nor implemented as relational hierarchies. Separately, when they *are* recognised, they are implemented in the most ridiculous, ham-fisted ways. Note that both of these examples, and anything else they come up with in the future (other than the Relational Hierarchy) focus on the *rendition* of the data (which is irrelevant an multiple), than on the data itself (single).

1. **Adjacency List**

   The suppressors hilariously declare that *the Relational Model doesn't support hierarchies*, in ignorance or denial of the fact that it is founded on the *Hierarchical Model*. So they can't use the name. This is the stupid name they use, they cannot name it for what it is because that would expose precisely what they are suppressing. Generally, the implementation will be cognisant of the hierarchy in the data, but it will be very poor, limited by physical Record `IDs`, etc.

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

2. **Nested Set**

   An abortion, straight from hell. This **fixes** the records in the filing system, in concrete. Moving a single node requires the entire tree to be re-written. Maintenance is prohibitive. Beloved of the Date; Darwen; Fagin; Fowler; Ambler; and Celko types.

   As you can imagine, the code that is required to maintain and navigate these monstrosities, is horrendous. The web is full of them.