A Primer on Network and Relational Data Modeling

Abstract

Data modeling is the process of representing real-world data and their relationships with each other in a form that is ultimately useful within computer programs. Selection of a method for effectively modeling data and relationships is one of the key decisions a developer will make when developing a data-driven system.

Contents

1 INTRODUCTION

Data modeling is the process of representing real-world data and their relationships with each other in a form that is ultimately useful within computer programs. Selection of a method for effectively modeling data and relationships is one of the key decisions a developer will make when developing a data-driven system. There are many different data modeling methods available today, with two common ones being Network and Relational models. This paper is a brief comparison of the intuitive process of data modeling by comparing these two modeling methods.

The following issues will be explored:

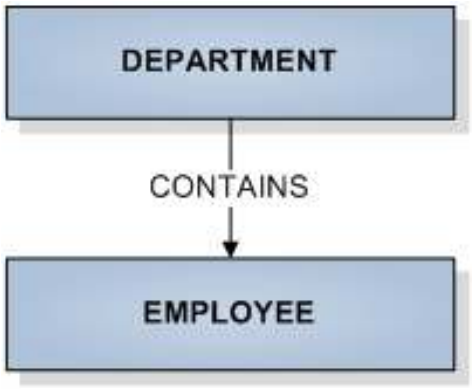
- The process of data modeling using different data models.
- The process of writing programs and how it is affected by the data model.
- Procedural vs. non-procedural access to data.

2 A QUICK DEFINITION OF THE DATA MODELS

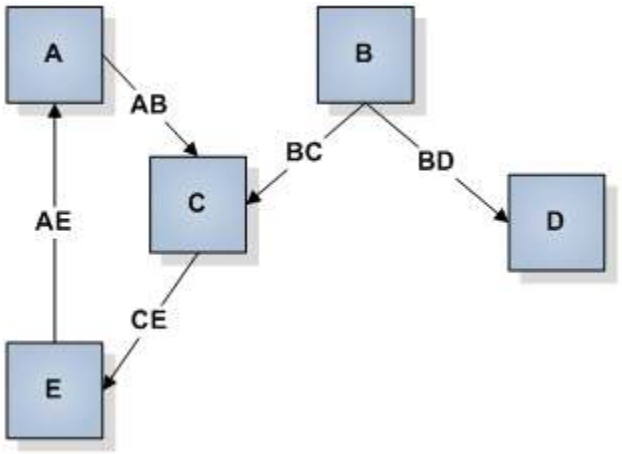
First, it should be stated that one could represent similar data relationships in both the Network and Relational models. The comparison between the two is a question of how naturally one can make their database structure model the real-world data that is being stored, and how naturally one can find and change data stored in the database.

The *Network model* commonly represents data and relationships through diagrams containing boxes and arrows. Each box represents a *record type* and each arrow represents a *relationship type*. A record type will contain *fields*, which are used to store individual values, which together represent defining information about the real-world entity the record represents.

For example, if there is a need to store information about employees in a database, it is natural to define an *employee* record type containing fields such as *last name, first name, SSN, Date of Birth*, etc. If employees are grouped by departments in a certain organization, then a *department* should also be defined to store information about one or more departments. The relationship between departments and employees could be called *contains*. It is a one-to-many relationship, where one department may contain many employees, and each employee (at least for this modeling exercise) can by contained by only one department. A diagram is shown below:



The name "Network model" is used because multiple boxes can be connected to other boxes through multiple arrows:



The *Relational model* represents data in a tabular format, with each *row* representing an entity of some type, and each *column* representing a defining attribute of the entity. The columns for *employee* and *department* tables in a Relational model might look as follows:

EMPLOYEE				
Last name	First Name	SSN	Date of birth	Dept name

DEPARTMENT	
Dept name	Dept desc

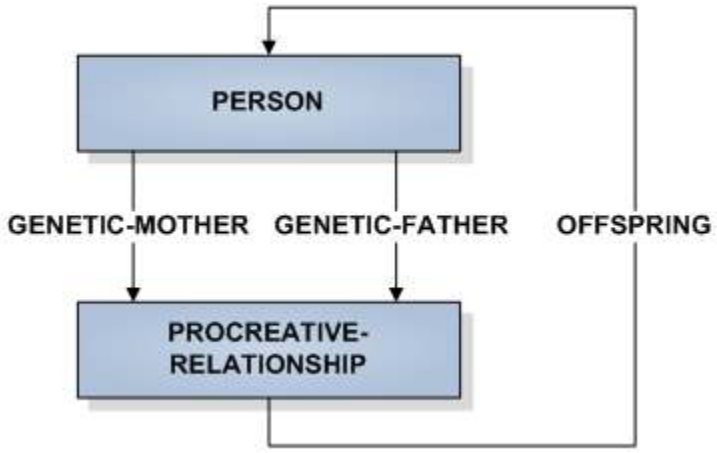
Roughly, a row is the same as a record, and a column is the same as a field. A relational model does not show explicit relationships between rows, but allows relationships to be derived through comparison of column values. In the above example, the *employee* table contains a *department name* (dept name) column, which should identify the department that contains the employee. In this case, where the employee's department name matches the department's department name, we can derive the relationship between an employee and department.

3 AN EXAMPLE FOR COMPARISON

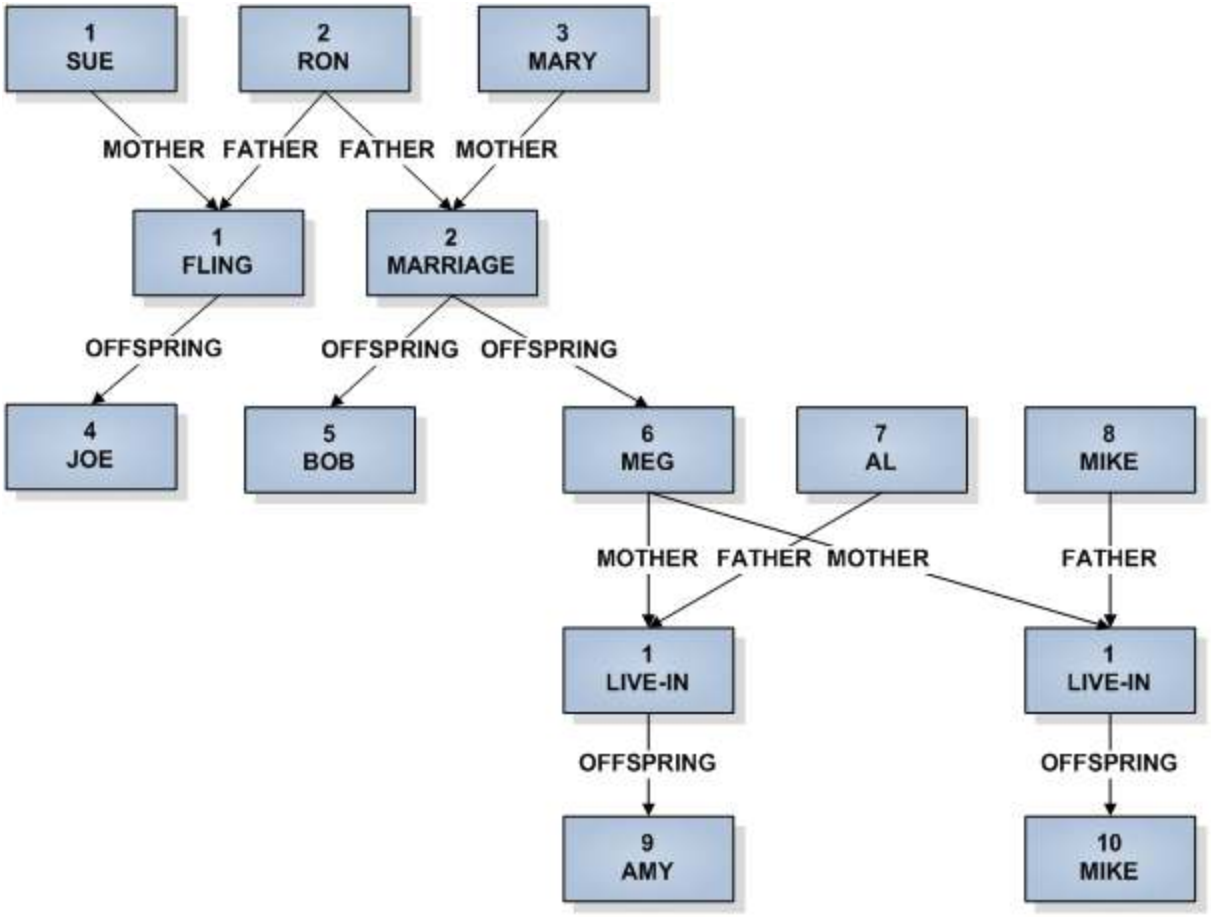
In this section, a fairly simple data model is developed in both Network and Relational models. The same data is shown as it would commonly be shown for both models. In the subsequent section, pseudo-code is included to demonstrate how a programmer might use the models from a software application.

3.1 The Network Model view

A "network" of data relationships that can naturally be visualized is the "family tree." The diagram below shows a network model design in a simple schema diagram.



This example shows that a PERSON may be either the GENETIC-MOTHER or GENETIC-FATHER within zero or more PROCREATIVE-RELATIONSHIPs. A PROCREATIVE-RELATIONSHIP can yield zero or more PERSONs that have an OFFSPRING relationship. A schema diagram shows the *form* of the data, but not actual data. Below shows a visualization of sample data that adheres to the form of the schema. In this sample, a partial family tree is shown for three generations.



With the two record types and three set types, any and every genetic relationship can be represented in a database, including children of parents from multiple relationships.

The simple schema diagram promotes visualization of data records and their relationships to each other, as demonstrated in the example above.

3.2 The Relational Model View

To contrast Network to Relational views, two relational tables are shown below—one containing PERSON data and another containing PROCREATIVE-RELATIONSHIP data. These tables, together with the appropriate primary/foreign-key relationships, represent the very same family tree as shown above.

PERSON			
Person-id	Name	Gender	Offspring
1	SUE	F	
2	RON	M	
3	MARY	F	
4	JOE	M	1
5	BOB	M	2
6	MEG	F	2
7	AL	M	
8	MIKE	M	
9	AMY	F	3
10	CAROL	F	4

PROCREATIVE-RELATIONSHIP			
Person-id	Description	Genetic-father	Genetic-mother
1	FLING	2	1
2	MARRIAGE	2	3
3	LIVE-IN	7	6
4	LIVE-IN	8	6

Relationships between the rows of the tables are through matching column values, often referred to as PRIMARY/FOREIGN KEY relationships. In this example, PERSON-ID and PR-ID are primary keys. OFFSPRING is a foreign key of PR-ID, and both GENETIC-FATHER and GENETIC-MOTHER are foreign keys of PERSON-ID.

This view, while very "computer friendly" is not extremely useful to a programmer who works through designs and algorithms with mental images that are spatial. A family tree is generally viewed, in real life, as a network of people and their relationships with one another. To tabularize this view in order to represent it in a database makes it more difficult for a programmer to work with.

4 APPLICATION CODING COMPARISON

This section assumes the reader has some familiarity with SQL.

Consider how to answer the question "*Who are the known ancestors of Amy?*" in an application program.

A Network model database will have a programming API that allows a programmer to "navigate" or "move around" through the records and sets in a database.

A Relational model database is normally accessed by specifying the data that is desired through a language like SQL. It has been claimed that it is better to state *what is wanted* from the database, rather than *how to find it* in the database. In this case, the DBMS makes the decisions about how to locate the data, and presents the complete answer to the programmer. This is called *non-procedural* database access. However, this paper has deliberately selected an example for which SQL does not have a standard way to specify the results non-Procedurally. The example's recursive hierarchy is not an uncommon database structure.

Navigation of the hierarchies in these tables is possible through a progression of SELECT statements, or through extensions to SQL such as the "START WITH" and "CONNECT BY" clauses supported by Oracle.

The Network model solution to the question is presented below. This pseudo-code mimics the actions of a Network model programming API, but has been simplified. The API allows movement (navigation) from one record to another through their *set* connection with a single call.

```
main() {
    find "Amy" record; /* "Amy" becomes "current" record */
    if found
        find_ancestors(0);
}
find_ancestors(level) {
    print NAME from current record, indent level*3 spaces;
    find parent of OFFSPRING set;
    if found {
        find parent of GENETIC_FATHER set;
        if found {
            find_ancestors(level+1);
        }
        find parent of GENETIC_MOTHER set;
        if found {
            find_ancestors(level+1);
        }
    }
}
```

How about the Relational solution to question? A solution is presented below. Note that this solution is *procedural* even though many of the arguments in favor of the Relational model are about its *nonprocedural* qualities. However, in this example, SQL has no way to express the question in a single statement (that is unless you use vendor-specific extensions which can be found in Oracle). The solution mimics the Network model by *navigating* through the database, but since it must parse and optimize each SELECT statement, it cannot possibly be any faster. And the actual implementation of this, assuming that ODBC is used as an API, results in many more lines of code than the actual Network model implementation.

```
main() {
   /* find "Amy" record */
   SELECT name, offspring FROM person
      WHERE person-id = "Amy";
   if found
      if offspring not null
         find_ancestors(name, offspring, 0);
}
find_ancestors(name, var_offspring, level) {
   print name, indent level*3 spaces;
   SELECT genetic-father, genetic-mother FROM procreative-rel
      WHERE procreative-rel.pr-id = var_offspring;
   if found {
      /* find genetic-father */
      if generic-father not null {
         SELECT name, offspring FROM person
            WHERE person-id = genetic-father;
         if found
            if offspring not null
               find_ancestors(name, offspring, level + 1);
      }
      /* find genetic-mother */
      if generic-mother not null {
         SELECT name, offspring FROM person
            WHERE person-id = genetic-mother;
         if found
            if offspring not null
               find_ancestors(name, offspring, level + 1);
      }
   }
}
```

Please note that the above solution is naturally similar to the Network model solution. But if the Network model view was not available prior to coding this and only the two tables were in front of the programmer, this solution would have been extremely difficult and unnatural to discover.

Another question might be "What are the names of the descendants of Ron?"

The Network model solution to this question will appear as follows:

```
main() {
   find "Ron" record;
   find_descendents(0);
}
find_descendents(level) {
   print NAME from current record, indent level*3 spaces;
   for ( find first PROCREATIVE-RELATIONSHIP member;
      status is FOUND;
      find next member of PROCREATIVE-RELATSIONSHIP )
   {
      for ( find first PERSON member through OFFSPRING set;
      status is FOUND;
      find next PERSON member through OFFSPRING set )
      {
         find_descendents(level+1);
      }
   }
}
```

How about the Relational solution to this question? Again, it is the same algorithm, but contains more complexity because of the need to use the SELECT statement for navigation of the hierarchies.

5 SUMMARY

To summarize the points demonstrated in this paper:

- Data relationships may be modeled in different ways to capture the same relationships.
- The visualization of the data relationships affects the programmer's intuitive ability to create code that navigates the data.
- SQL, in spite of its purpose to allow non-procedural specification of database queries, is not able to do so in a common class of database structures. In this situation, the use of SQL is less intuitive, more difficult to program, and less efficient.

6 CONCLUSION

In examples such as the one shown, where data relationships naturally contain one or more hierarchies, the Network model provides superior visual and navigational approaches for representing and manipulating the data. This results in intuitive programming solutions that are small and efficient.

Want to know more?

Please call us to discuss your database needs or email us at info@raima.com. You may also visit our website for the latest news, product downloads and documentation: www.raima.com.

Headquarter: 720 Third Avenue Suite 1100, Seattle, WA 98104, USA T: +1 206 748 5300
Europe: Stubbings House, Henley Road, Maidenhead, UK SL6 6QL T: +44 1628 826 800