

Parsing a SAS Database for Correctness: A Conceptual Introduction

By

Edward F. Brown, PhD
Brenda M. Bishop, MBA
Bruce Morrill, MS, MA
EMB Statistical Solutions, LLC.

Abstract

It is known that all well-formed relational databases (schemas and sub-schemas) can be described by a context-free grammar. This fact allows for a host of possibilities, one being that relational databases can be parsed for syntactical correctness. While this concept is commonly accepted and observed in software compilers, the parsing of relational databases is novel and unexplored.

In a clinical database environment, it is common to have 20-50 SAS datasets all linked by a common identifier, as well as secondary keys. One can view the inter-record relationships as a data structure having a tree-shaped geometry. After the initial entry of data, records in each dataset are in constant flux and update operations can add, delete and modify records from multiple sources at multiple times during a day. Each transaction has the potential to change overall geometry of an identifier. At the conclusion of a study, or during any interim period, one must ensure that all records for a particular identifier have the correct geometry across all SAS datasets.

This paper introduces a novel approach that greatly simplifies and automates the process of validating the correctness of inter-record relationships for a particular identifier across multiple SAS datasets. This allows for frequent and repeated validation checks, and serves to authenticate the correctness of a database at a given point in time. Although clinical database structures for the pharmaceutical and biotechnology industries are principally featured, the principles can be adapted to other fields.

Introduction

In a database environment where a large number of transactions occur, ensuring that all inter-record relationships are correct is a challenge. This endeavor can become daunting if an institution maintains many large and complex databases. This is the case in the pharmaceutical industry, where each study has its own database schema consisting of 20-50 unique SAS datasets. One can view the inter-record relationships as a tree structure having a well-defined geometry. However, each primary key in the database can spawn an individualized tree structure, which conforms to the rules, but is unique in appearance.

This paper addresses how one determines whether all records in the database are part of a legitimate tree structure for a given identifier. In an environment where a large number of transactions have occurred from staff with little knowledge in relational databases, this is a legitimate concern that remains difficult to certify. At issue is whether records have been inadvertently dropped, duplicated or orphaned in a given database.

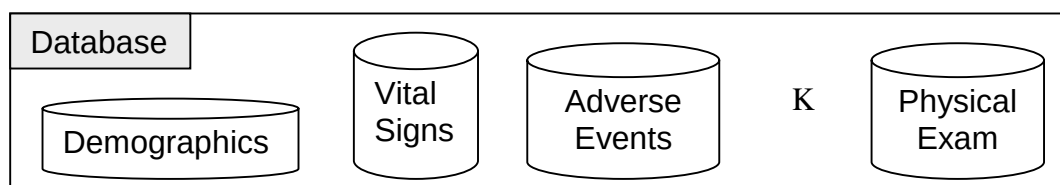
This problem is analogous to a large number of programmers working on a piece of software. On a given day, one may ask whether their language code is syntactically correct. In this instance the answer is simple to determine; one submits the code through a compiler, which parses and identifies all syntactical errors. If the code passes without error, then one knows that the software was legally constructed. (NOTE: It is important to understand that a syntactically correct program does not imply a logically correct program. To determine whether a program will run correctly, or even terminate within finite time, remains an intractable problem.)

In this paper, we are proposing a “database compiler” that parses the inter-record structure of a database to determine if it is syntactically correct for each specified key. It is limited to this aspect, and will not verify the logical integrity of values within or across individual fields.

Terminology

Let a database be a collection of related datasets. In pharmaceutical studies, it is common to have 20-50 individual SAS datasets comprising a single database, all linked by a primary key and having an arbitrary number of secondary keys. Figure 1 serves to illustrate this concept.

Figure 1.
Conceptual View of Database:
A Collection of many Individual Datasets



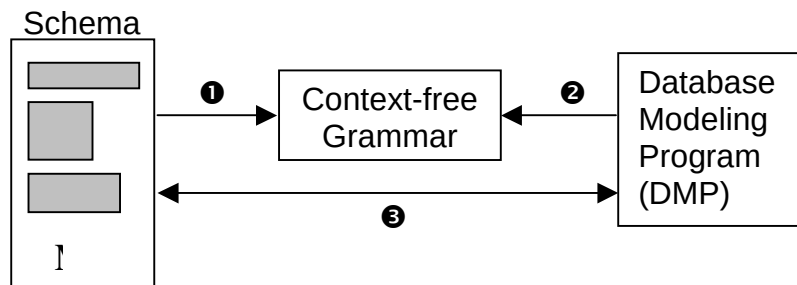
Let schema be the definition of an entire database. It specifies all database keys, variable attributes, and relationships within and across datasets.

Let Database Modeling Program (DMP) be a simple SAS program to capture the inter-record relationships found in a particular database. Although the authors have been working on a more advanced formalism, it is outside the scope of this paper to present this topic.

Methodology

It is known that all well-formed relational databases (schemas and sub-schemas) can be described by a context-free grammar (see ❶ in Figure 2.). Because programming languages are typically based on context-free grammars, let a new program (DMP) be used to describe a database structure (see ❷ in Figure 2.). Because the Schema and DMP are both based on context-free grammars, they are equally expressive, and one can be used to describe the other (see ❸ in Figure 2.).

Figure 2.
Illustration Showing Equality Between the Database Schema and DMP



The above statement guarantees the DMP can be developed to precisely describe any database schema. Once developed, the DMP can be executed to check the validity of the Schema. In essence, the DMP drives a sequence of database queries, where each query must return one-and-only-one record. Otherwise, the database is considered corrupt for one of the following reasons.

- 1) If no records are returned, then an essential record is missing in the database.
- 2) If multiple records are returned, then a record duplicate has been encountered.

To ensure that each and every record is accessed only once, a counter should be associated with each database record. If a database query returns a record whose counter is greater than 0, then this record has been accessed before and generates an appropriate error message. If, after the DMP has been fully executed for all identifiers, records exist whose counter is zero, then these are orphan records that have no linkage and their inclusion in the database is at question.

Example Database

Consider a database of siblings, and their children. A subject is prompted for their name, age and the number of siblings. For each sibling, they are prompted for their sibling's name, age and the number of children they have had. For each child of a sibling, they are again prompted for their name and age.

To illustrate how differently each subject may respond, and form their own unique tree structure, consider the following three subjects:

Subject #1: [Name=Abe, age=30, # of siblings=2]
 Sibling #1: [Name=Barb, age=28, # of children=0]
 Sibling #2: [Name=Cole, age=26, # of children=1]
 Child #1: [Name=Dave, age=3]

Subject #2: [Name=Albert, age=62, # of siblings=0]

Subject #3: [Name=April, age=38, # of siblings=3]
 Sibling #1: [Name=Bob, age=43, # of children=1]
 Child #1: [Name=Carol, age=16]
 Sibling #2: [Name=Drew, age=35, # of children=2]
 Child #1: [Name=Edith, age=6]
 Child #2: [Name=Frank, age=2]
 Sibling #3: [Name=Grace, age=30, # of children=0]

For purposes of illustration, let the following three relational datasets (***Subject***, ***Sibling***, and ***Child***) be used to store the example database for the three subjects given above. Let italicized variable names having an underscore in their names be keys for a particular dataset.

<i>Subject</i>	<i>Id_Num</i>	Name	Age	Siblings
	1	Abe	30	2
	2	Albert	62	0
	3	April	38	3

<i>Sibling</i>	<i>Id_Num</i>	<i>Sib_Num</i>	Name	Age	Children
	1	1	Barb	28	0
	1	2	Cole	26	1
	3	1	Bob	43	1
	3	2	Drew	35	2
	3	3	Grace	30	0

<i>Child</i>	<i>Id_Num</i>	<i>Sib_Num</i>	<i>Ch_Num</i>	Name	Age
	1	2	1	Dave	3
	3	1	1	Carol	16
	3	2	1	Edith	6
	3	2	2	Frank	2

Example DMP

In parsing a database, one is able to detect errors in the inter-record relationships that violate the database description. The following illustrates several examples of

inconsistencies within a given structure that would be detected, many of which may be introduced during update procedures:

Subject #4: [Name=Ally, age=33, # of siblings=1] Sibling #1: [Name=Bruce, age=23, # of children=0] Sibling #2: [Name=Cathy, age=21, # of children=0]	<i>1 sibling specified, but 2 siblings are recorded.</i>
Subject #5: [Name=Adrian, age=28, # of siblings=1] Sibling #2: [Name=Carter, age=1, # of children=0]	<i>Sibling #2 should be labeled as #1.</i>
Subject #6: [Name=Arthur, age=55, # of siblings=1] Sibling #1: [Name=Bill, age=50, # of children=0] Sibling #1: [Name=Bill, age=50, # of children=2] Child #1: [Name=Cameron, age=27] Child #2: [Name=Drake, age=25]	<i>New sibling record was inserted, but old record was never deleted.</i>

Let the following Database Modeling Program (DMP) be used to capture the database description, and test whether the example database is compliant.

```

1)  DATA _NULL_;
2)    DO UNTIL (done);
3)      SET Subject END=done;
4)      %Query_Count (ds=Subject, key=Id_Num);
5)      %Get_Val (ds=Subject, key=Id_Num, var=Siblings, Out=Sib_Val);
6)      DO Sib_Num = 1 TO &Sib_Val;
7)        %Query_Count (ds=Sibling, key=Id_Num Sib_Num);
8)        %Get_Val (ds=Sibling, key=Id_Num Sib_Num, var=Children, Out=Ch_Val);
9)        DO Ch_Num = 1 TO &Ch_Val;
10)         %Query_Count (ds=Child, key=Id_Num Sib_Num Ch_Num);
11)       END;
12)     END;
13)   END;
14)   %Flag_Zero_Counts (ds=Subject Sibling Child);

```

The above DMP consists of the following 3 macro calls:

- %Query_Count (ds=, key=); For a given dataset (ds=) and list of variable keys (key=), let the function issue a query to the specified dataset using the current values stored in the variables that comprise the key. If one-and-only-one record is returned, then the database is conforming to expectations. Otherwise, an error is issued.

In addition, assume the function maintains an array to store the number of times each record has been accessed for all datasets. If the count for the queried record is zero, then increment the counter to 1. Otherwise, this record has been accessed before and an error is issued.

- %Get_Val (ds=, key=, var=, out=); For a given dataset (ds=) and list of variable keys (key=), let the function issue a query to the specified dataset using the current values stored in the variables that comprise the key and return the value

of a specified variable (var=). The result is returned in the macro variable specified in out=.

- %Flag_ Zero_Counts (ds=); For each dataset in the list (ds=), report all orphaned records whose record count equals zero.

The above commands and functions are expressive enough to capture, and validate, all possible configurations for a particular subject. Any deviation in the database model results in an error message. In brief, the above DMP processes Subject #1 in the following manner.

Statement 3): The first record in dataset= **Subject** is accessed, and 1 is stored for the key variable, Id_Num.

Statement 4): %Query_Count() queries the dataset **Subject** where Id_Num=1. One-and-only-one record is returned, and the current count for this record is 0, indicating that this record conforms to the database model. Before returning, the macro increments the record count to 1.

Statement 5): %Get_Val() reads a record from the dataset **Subject** where Id_Num=1. The value for the variable Siblings is read, and stored in the macro variable Sib_Val. Afterwards, &Sib_Val=2.

Statement 6): DO Sib_Num=1 to 2 is resolved. Sib_Num is initialized to 1.

Statement 7): %Query_Count() queries the dataset **Subject** where Id_Num=1 and Sib_Num=1. One-and-only-one record is returned, and the current count for this record is 0, indicating that this record conforms to the database model. Before returning, the macro increments the record count to 1.

Statement 8): %Get_Val() reads a record from the dataset **Sibling** where Id_Num=1 and Sib_Num=1. The value for the variable Children is read, and stored in the macro variable Ch_Val. Afterwards, &Ch_Val=0.

Statement 9): DO Ch_Num=1 to 0 is resolved, so control transfers to Statement 12).

Statement 12): Sib_Num is incremented to 2. The comparison of Sib_Num to the maximum increment value succeeds, so control returns to Statement 7).

Statement 7): %Query_Count() queries the dataset **Subject** where Id_Num=1 and Sib_Num=2. One-and-only-one record is returned, and the current count for this record is 0, indicating that this record conforms to the database model. Before returning, the macro increments the record count to 1.

Statement 8): %Get_Val() reads a record from the dataset **Sibling** where Id_Num=1 and Sib_Num=2. The value for the variable Children is read, and stored in the macro variable Ch_Val. Afterwards, &Ch_Val=2.

Statement 9): DO Ch_Num=1 to 1 is resolved. Ch_Num is initialized to 1.

Statement 10): %Query_Count() queries the dataset **Subject** where Id_Num=1, Sib_Num=1 and Ch_Num=1. One-and-only-one record is returned, and the current count for this record is 0, indicating that this record conforms to the database model. Before returning, the macro increments the record count to 1.

Statement 11): Ch_Num is incremented to 2. The comparison of Ch_Num to the maximum increment value fails, so control returns to Statement 12).

Statement 12): Sib_Num is incremented to 3. The comparison of Sib_Num to the maximum increment value succeeds, so control returns to Statement 13).

Statement 13): Begins iteration of a subject

After processing all subject IDs in the dataset **Subject**, the macro %Flag_Zero_Counts is called last. It's purpose is to report all orphan records found in the parameter list of datasets.

Limitations

The example DMP provided above is simplistic, and used only to illustrate the concept of database parsing. Even in its simplistic form, the example DMP is powerful enough to represent a large number of database structures. With a few added functions, the language can easily handle amendments in database structure, incomplete subject records, and complex key manipulations.

In theory, no limitations should exist. The DMP is equally expressive as a given relational database structure. Only in poorly formed databases could the DMP fail to appropriately capture the database correctly.

Conclusion

In a database environment where a large number of transactions occur, maintaining the structural integrity of a database is difficult. When a large number of databases are created and maintained, writing custom programs for each database is inefficient. This paper presented the concept of a database parser, which validates the consistency within the structure of a database. A simplistic program called Database Modeling Program, was presented to illustrate the parser's operation.

In return for a small initial investment (including DMP design and development of functions), the resulting DMP operates at a higher level of abstraction. This allows for databases to be quickly modeled, and for the operation to be performed repeatedly as a production run. In fact, the DMP could be integrated into any "transaction run", and allow only those transactions that result in a structurally correct database to pass through to the base dataset.

Although a concept paper, the authors have made good progress towards a final product. The authors have found that even a simple DMP is quite powerful in its representational power, and easily models complex databases.

Future work will focus on using the DMP to create test databases. Because the DMP represents all possible database permutations, it is ideal for this task and would simplify the process immensely.