Referential Integrity in VA FileMan

William J. Harvey, Ph.D.
Institute for Information Management and
Department of Computer and Information Systems
Robert Morris College
Pittsburgh and Coraopolis, Pennsylvania

Abstract

The VA File Manager (FileMan) database management system provides automated support for referential integrity. FileMan is examined here with regard to keys, uniqueness and null constraints, domains, file relationships, and rules for insertion, deletion, and updating of records affected by inclusion dependencies as a special case of referential integrity. FileMan referential integrity provisions are compared to those for standard relational (SQL) database systems.

Introduction

The VA File Manager (FileMan) is the most widely used database management system based on M technology. Various versions are in use, for medical and non-medical applications, around the world. FileMan is menu- and dialog-driven and supports nested file structures in the form of "subfiles." Originally developed in the 1970s, FileMan also offers a word processing data type to permit recording of physicians' and nurses' free-text entries in the health care environment. Letters and various documents can be prepared automatically using data in the FileMan database. Unlike form letter facilities in some word processing systems which permit merging a document only with a single prepared file, the FileMan form letter system makes it possible for a form letter to access an entire database in the merge process. Winn and Hoye have described FileMan support for relational database operations, showing FileMan equivalents for relational data manipulation operations.[1]

Report generator facilities in FileMan can handle the nested subfiles conveniently. Code generation is used extensively in FileMan, and the standard data dictionaries provided by FileMan systems document source code for triggered database actions, data validation, and output transforms. A number of database systems and products are derived from the FileMan model or use FileMan file structures.

Many M applications are integrated with FileMan. As an example of the sophistication of current uses of FileMan, a FileMan-based expert system at the University of Texas M. D. Anderson Cancer Center carries out automatic validation of research data in large-scale projects.[2] FileMan stores the required knowledge bases and the expert system routines are written in standard M, supporting slots and fuzzy logic. Rules can be entered which use FileMan field names to which logical names are assigned.

This paper explores the automated provisions of FileMan technology which support relationships between files and endeavors to formulate the rules used by FileMan in maintaining referential integrity. Certain provisions for referential integrity support, such as the FileMan pointer-to-a-file data type, have been present since early versions. To that extent some level of referential integrity support was introduced in FileMan far in advance of the introduction of referential integrity provisions in SQL in the 1989 standard. Referential integrity in the relational model, and referential integrity support specified by the SQL standards and SQL implementations, will be used as a basis for examining FileMan referential integrity. The paper will focus on direct use of the database through its menus and dialog with limited treatment of embedded use of FileMan in M programming through calls to FileMan routines (under which condition certain restrictions may be overridden). References in this paper are to Version 19.0 of FileMan.[3]

Referential Integrity in Relational Databases

Referential integrity applies to database intertable relationships where there is a common domain from which a primary key (a set of one or more columns) in one table (e.g. TABLE_A) and a foreign key (a set of one of more columns in another table (TABLE_B) both draw values. The set of one or more column(s) comprising the foreign key (in TABLE_B) has the same definition(s) as the primary key column(s) in the first table (TABLE_A). Every non-null value in the foreign key column(s) of TABLE_B must exist in the primary key column(s) of TABLE_A. In giving a definition of referential

integrity based on domains and keys, Date uses the qualifier "unmarked" for non-null foreign key values and specifies that "one or more primary keys draw their values" from the common domain."[4]

For example, imagine a hospital database (Database 1) with a table of PHYSICIANS in which the primary key of PHYSICIANS draws on a domain of physician identifier values and a PATIENTS table including a column for an ATTENDING_PHYSICIAN foreign key drawing physician identifier values from the same domain as the primary key of PHYSICIANS. Here are two additional examples (PK=primary key, FK=foreign key):

Database 2
DEPARTMENT(PK DEPARTMENT_ID)
and
EMPLOYEE(PK EMPLOYEE_ID and FK DEPARTMENT_ID)
Database 3
PATIENT(PK PATIENT_ID) and
TEST(PK TEST_ID and FK PATIENT_ID)

Referential integrity support consists of making sure that at any point in time there are no non-null foreign key values that do not also occur in the set of values in corresponding primary key. The constraint requiring that the set of foreign key values in one table (such as TABLE_B above) must be a subset of the set of primary key values in another table (such as TABLE_A) is called an *inclusion dependency.*[5] The table containing the foreign key (TABLE_B) is called the *referencing table* and the table containing the primary key referred to by the foreign key is called the *referenced table.* In database design there is typically a one-to-many relationship involved in referential integrity, as for example:

Database 1
Each PHYSICIAN has zero or more (many) PATIENTs
Each PATIENT has one ATTENDING_PHYSICIAN
Database 2
Each DEPARTMENT has zero or more (many) EMPLOYEEs
Each EMPLOYEE is assigned to one DEPARTMENT
Database 3
Each PATIENT has zero or more (many) TESTs
Each TEST is associated with one PATIENT

Since this is sometimes seen as a hierarchical structure the terms *parent table* (for the referenced table) and *child or dependent* table (for the referencing table) are also used.

For relational databases, referential integrity support requires that the following can be declared:

Definition of the primary key as a means of uniquely identifying a row in a table, including a rule that primary keys may not contain null values.
Definition of the foreign key and its relationship to a given primary key.
Provisions for handling null value requirements.
A rule specifying that no values may be inserted into the foreign key that do not already exist in the primary key of the referenced table (insert rule).
A rule specifying that deletions of values in a referenced table primary key must not violate the inclusion dependency by resulting in "orphaned" foreign key values in the referencing table for which there is no matching value in the corresponding primary key (delete rule).
A rule specifying that updates to a referenced table primary key must not violate the inclusion dependency by deleting referenced values and resulting in "orphaned" foreign key values in the referencing table for which there is no matching value in the corresponding primary key *and* no values may be inserted into the foreign key through an update operation that do not already exist in the primary key of the referenced table (update rule).

There must be a provision to declare the columns as ineligible to receive null values so that any primary key column (set of primary key columns) can be required to have only non-null values. Database design will dictate whether the no-null-values provision should be applied to a foreign key definition. In Database 2, if the organization using the database permits an EMPLOYEE to be, perhaps temporarily, unassigned to any DEPARTMENT, then null values would be permitted in the foreign key.

The ANSI and ISO SQL standards and various SQL implementations provide syntax such as in the following examples using the CREATE TABLE and ALTER TABLE specifications:

Declaring columns may not have nulls:

```
CREATE TABLE DEPARTMENT
(DEPARTMENT_ID CHAR(4) NOT
NULL,
... )
```

Declaring a primary key:

```
ADD PRIMARY KEY
(DEPARTMENT_ID)
```

Declaring a foreign key with a delete rule:

```
ADD FOREIGN KEY DEPT_KEY
(DEPARTMENT_NO)
REFERENCING DEPARTMENT
ON DELETE SET NULL
```

Each column is specified individually as to any null constraint. The primary key declaration results in a uniqueness constraint for the primary key column(s). According to the uniqueness constraint, there will be no duplicate values in a primary key. A uniqueness constraint may be applied to a candidate key in addition to the primary key, through an ALTERNATE KEY declaration. Note that the default declaration of a foreign key, as given here, automatically targets the primary key of the referenced table, so that it is not necessary to specify the primary key column(s) in the foreign key definition. However, relational database systems implementing referential integrity support will check to make sure that the definitions of the corresponding primary and foreign key columns match exactly, assuming a common underlying domain of values for both keys. A foreign key may reference the table in which it is declared. A primary key may also be a foreign key.

The following delete rules (rules for deleting currently referenced rows and implicitly their primary keys) are accounted for in the SQL standard or in implementations:

RESTRICT or NO ACTION
SET NULL
SET DEFAULT
CASCADE

More than one foreign key can be declared for a single table (possibly referencing different tables), but all foreign keys in the same table must be subject to the same delete rule category. RESTRICT, SET NULL, SET DEFAULT, and CASCADE options can exist for declaration of update rules.

The RESTRICT rule inhibits any deletion of a table row with a primary key value referenced by one or more foreign key values in a referencing table. Organizational policy might dictate, for example for Database 1, that a PHYSICIAN not be deleted from the database until *after* all of his/her PATIENTs have been assigned a new PHYSICIAN. Horowitz distinguishes between the RESTRICT and NO ACTION rules, considering the NO ACTION rule as a "soft or implicit restriction."[6]

The SET NULL rule permits the deletion of the referenced row and causes the foreign key value(s) in the corresponding rows in the referencing table to be replaced with null values. This could apply to Database 2 where an EMPLOYEE in a DEPARTMENT being dissolved could temporarily be unassigned (represented by the null value). A SET NULL declaration should be rejected if the foreign key column(s) have been declared NOT NULL or ineligible to receive null values.

The SET DEFAULT rule permits the deletion of the referenced row and causes the foreign key value(s) in the corresponding rows in the referencing table to be replaced with previously declared default value(s). This requires a provision for declaration of defaults.

The CASCADE rule permits the deletion of the referenced row and causes the rows containing the corresponding foreign key values to be deleted as well. Because it is possible that an unintended chain of cascade deletes could be initiated, implementations place restrictions on the declaration of cascade delete rules to inhibit such events. For a general treatment of "safety" with regard to cascade deletes, see Markowitz' article on the topic.[7]

There is an order to defining and entering values into tables under the referential integrity constraint. The referenced table must be declared and its primary key must be declared before the foreign key may be declared in the referencing table. Any value to be inserted into the foreign key of the referencing table must have been inserted into the primary key of the referenced table first.

Files and File Relationships in FileMan

The term referential integrity arose in relational database theory but will be applied here to FileMan, which is not considered to conform to the relational database model. Davis characterizes the design of FileMan as "fundamentally" hierarchical, although it transcends strict hierarchical architecture.[8] There is precedent for description of referential integrity support in non-relational databases in Date's treatment of the topic for hierarchical IMS databases

and for the IDMS database as an example of CODASYL (network model) databases.[9] Date cites the rule that "no child is allowed to exist without its parent," as found in IMS and IDMS, as a form of referential integrity specifying an insert rule requiring prior existence of the parent record (segment in IMS). For IMS, Date mentions cascade update (replace in IMS) and delete rules and a provision that nulls are not permitted (as keys). According to Date, the IDMS rules approximate foreign key rules on allowing nulls, delete rule options ("CASCADES or RESTRICTED or NULLIFIES"), and an update rule of cascade. The role and properties of referential integrity constraints in the "relational representation of object-oriented structures" are treated by Markowitz.[10]

FileMan is examined here with regard to keys, domains, uniqueness and null constraints, file relationships (interfile and intrafile), and rules for insertion, deletion, and updating of records affected by inclusion dependencies as a special case of referential integrity. In addition to forms of insert, delete, and update rules, FileMan provides a LAYGO (Learn-As-You-Go) access option for the "pointer-to-a-file" data type. LAYGO is an automated referential integrity action for insertion of records into a referenced file. The LAYGO action is not integral to SQL implementations. When LAYGO access is enabled, the user editing a referencing file will be alerted to the fact that a new value is not currently displayed in a certain referenced file column and may then decide to have that value entered in the referenced (pointed-to) table. LAYGO access can be specified when creating and defining a field. Nulls are a specific value in FileMan and in the underlying M programming language. Nulls are distinct from spaces but are not marked so that one null is distinguishable from another.

FileMan supports declaration and maintenance of domains for fields. Dialogs for basic data types receive specifications such as upper and lower bounds (for numeric values), maximum and minimum lengths (for free text strings), earliest and latest dates, and pattern match requirements. Certain defaults are provided to assist in the specification process. Value checking is implemented through *input transforms*. The syntax for input transforms is displayed in data dictionary listings. Programmers can design input validation routines.

FileMan databases utilize files, rather than tables. In relational databases, such as SQL, the foreign key references the primary key of the other table. There is no concept of user-defined "primary key" as commonly understood for relational database technology -- a file cannot be created without creating a distinguished field (called the ".01" field) of the type called "mandatory identifier" in FileMan; the .01 field represents an entity; additional mandatory or non-mandatory identifiers may be declared for a file from the fields established; a record cannot be added to a file without providing a value for every mandatory identifier declared for the file; the intent is that the designer determine which properties of an entity *must* be used to identify an instance uniquely and declare each such property to be an identifier; each record has an Internal Entry Number (IEN) which the system uses for unique identification and which is normally not visible to the user. As Winn and Hoye and also Davis emphasize, the user of FileMan is not expected to declare a primary key; this is done automatically by the system. Follingstad has described the need for an easily declarable uniqueness constraint for user-defined FileMan fields and has provided technical information on implementing such constraints.[11] FileMan has a automatic support available only for unique numeric identifiers. Relationships between files in FileMan are implemented using a record pointer system where record pointers (IENs), in combination with evaluation to extract values from the referenced file, carry out the function delivered by the foreign key system in relational databases. Date provides a description of the distinctions between the use of foreign keys and pointers in "Why foreign keys are not pointers."[12]

The philosophy of identification in the FileMan database system derives from the perception that individuals should be identified by examining the minimal set of uniquely identifying properties of individuals. Such properties are chosen because they are known not to change easily or often. If the .01 field contains a person's name, in the form of "Last, First", as would occur in a patient file, then it is taken for granted that duplicate names may show up in the field. Identifiers, such as date of birth, would need to be declared to be consulted in combination with the .01 field. The process of unique identification of records in FileMan using identifiers is analogous to that used by people in asking information from one another and is sometimes called "intelligent pursuit." This approach to

identification encourages using names and other identifying information from the "real world" rather than identifiers such as assigned numbers convenient for internal computer operations. The designers of FileMan obviously hoped users would ask "What is your name?" rather than "What is your account (or patient or staff) number?"

Good FileMan database design requires appropriate use of declarations of fields as mandatory or as identifiers or both. Fields may be declared *mandatory* through interactive dialog when the file is created or edited, which means that users must enter a non-null value when prompted (can't use hit return to enter a null value) as long as no action is taken to override dialog restrictions. Branch-out and branch-around operations may be inhibited. Mandatory fields which are not also identifiers may contain null values. Fields may be declared *identifiers* by using the Utilities File Edit option. An identifier may be automatically displayed on lookup. Identifiers enable additional attributes to be considered in selecting records. Identifiers are used internally by some FileMan functions (i.e. merge) to uniquely identify a record.

A field declared as mandatory and as an identifier (*mandatory identifier*) may not be deleted and must receive a non-null value when new records are added, including through the LAYGO process. Mandatory identifiers may not contain a null value. The combination of .01 field together with mandatory identifiers (where present) may be considered a kind of composite record key. The .01 field is always a mandatory (required) field in FileMan and added fields may be specified as mandatory. The prompt asks whether the field is mandatory, while the List Attributes data dictionary display uses the term *required*: hence mandatory and required are equivalent. Although uses of relational database technology expect (and receive) support for composite primary and foreign keys through the SQL standards and the most widely used SQL implementations, the use of composite keys has been questioned, as by Date in "Why noncomposite keys are a good idea."[13] Kilov and Ross, in discussing multi-attribute keys, emphasize that the policies of a particular database technology and its technical requirements for good database design should not distract those persons modeling the requirements of a business or organization from designing an appropriate information model.[14]

The *cross-reference* entries and utility provide enforcement of integrity rules through SET and KILL commands in M which insert or delete records to maintain the consistent status of cross references. Cross references between internal record numbers and the standard name field are maintained automatically by the system. Other cross references must be established by a user invoking the cross-reference utility. Referential integrity in File Manager systems can also be augmented by custom M programming.

FileMan provides the following relationships between files:

Pointer to a File data type

- The pointer values in a field defined as a pointer to a file type constitute a subset of the values in the IEN field of the pointed-to (referenced) file and the values obtained by evaluating pointer values constitute a subset of the values in the .01 field of the pointed-to (referenced) file.
- If LAYGO is permitted, a new value not in the pointed-to file is added to the pointed-to file in attempting to add a pointer for that value to the pointer field.

Variable Pointer data type

- A sequence of evaluation of multiple pointed-to (referenced) files is declared (as well as prefixes to aid in data manipulation).
- The values obtained in evaluating the pointers in a field defined as a variable pointer type constitute a subset of the values in the union of the sets of values in the .01 fields of the pointed-to files; any pointer value in the variable pointer field is only meaningful in the context of the particular file pointer associated with it, since the pointer values may have been drawn from the IENs of more than one file.
- If adding a new value is permitted (LAYGO) for a given variable pointer file, a new value not in the pointed-to file is added to the pointed-to file in attempting to add a pointer for that value to the pointer field; LAYGO is accepted (or declined) separately for each file pointed to.

Subfile

- A field is declared as a multiple, with a .01 field automatically created; additional fields of subfile records may be defined.

A subfile record is existence dependent on the parent record: no subfile record may be inserted without a parent record and a deletion of the parent record causes cascade deletion of dependent subfile records.

Subfiles may be nested to an arbitrary depth; i.e. subfiles may have subfiles.

When a FileMan user declares a pointer-to-a-file data type in defining (modifying) a field of a file, the user is required through dialog to specify an existing file. FileMan files can be used as lists that are pointed to by references in one or more other files so that data is entered only in a single file and not reentered. Data occurrences can be selected from what is in the pointed-to file or new entries can be inserted as needed under the LAYGO facility described below. The data dictionary for the pointed-to file has reflects of its pointed-to status. Referential integrity is maintained under the pointer option if users always choose to have updates made to files containing the pointer references when an entry is deleted from the *pointed-to* file. The FileMan system does not permit defining pointers to files that do not exist. The pointer values in FileMan fields declared with pointer-to-a-file and variable pointer data types serve as foreign keys and are presented as evaluated in displays of file contents (i.e. the user sees the .01 field value from the referenced file and values from any fields declared as identifiers) but are not true value-oriented foreign key values as in relational databases. This is consistent with the structure of FileMan where unique record identifiers are internal values.

FileMan provides extensive automated support for domains. Users encounter a dialog specific to each basic data type which permits domain definition (such as bounds, date/time ranges, string properties). The rule that foreign key and primary key (in the form of pointer to a file fields and corresponding referenced .01 fields) must draw values from the same domain is enforced by FileMan for the pointer-to-a-file type.

Assuming FILE 2 has a FIELD A which is a pointer to a file type which points to FILE 1, then let us say that FIELD A in FILE 2 is dependent on FILE 1 and that there is a parent/child relationship in which FILE 1 is the parent (referenced file in relational terms) and FILE 2 is the child (referencing file).

FileMan's referential integrity rules should provide:

Regarding entering new values into FIELD A of FILE 2 (Dependent, Child): if LAYGO is not permitted, no FIELD A value may be entered (either as a new value in a new record or as an update of a value in an existing record) which is not already entered as a .01 value in FILE 1; if LAYGO is permitted then a new value may be entered into FILE 1 after which that same value may be entered into FIELD A of FILE 2.

Regarding deleting values from FILE 1 (parent or referenced file): no FILE 1 record (i.e., .01 value) may be deleted from FILE 1 if there are dependent records (records with the same FIELD A values) in FILE 2 without choosing either (1) to delete the value in any dependent records in which it exists (delete record if pointer field is .01 field, otherwise reset field value to null) or (2) to assign an existing different value in any different records in which it exists (set field value to new allowable value); a user may decline the null/update restriction at the risk of leaving a dangling pointer in the pointer field.

These policies assure that the value in FIELD A in FILE 2 remain a subset of the values in the .01 field of FILE 1 as long as one of the two options for dealing with dependent records is accepted by the user. The presentation of options for handling the impact of parent record deletion on dependent records reflects a philosophy that (1) the decision as to whether to set null or update should be made at the time of parent record deletion rather than at the time of defining the file and (2) that selection of any update value should be made based on understanding conditions at the time of the parent record deletion rather than on establishing a default value at the time the file is created. Here is another example:

Database 4

Assume CITY in EMPLOYEE points to US CITY

Enter/Edit rule for EMPLOYEE: if LAYGO is not permitted for this field, a CITY must already be in the US CITY file

If LAYGO is permitted, a new US CITY may be entered and the same CITY may then be entered in EMPLOYEE

As mentioned above, FileMan has a provision for a *Learn-As-You-Go* (LAYGO) option which can be selected to assure referential integrity where the

entire set of object occurrences in the referenced table is not known initially and it is important not to inhibit artificially the entry of new records into the file containing the pointer reference. Davis terms such sets "open sets" as opposed to "closed sets" which cannot receive new data. The user is permitted to add new object occurrences to the referenced table as required in order to insert new objects into the database.[15] The creator of a FileMan table may allow or prohibit LAYGO.[16] This feature helps deliver the needed flexibility to keep data consistent but at the same time not restrict urgent entry of data. LAYGO actions are managed by dialog with the user. Insertion of a new record under LAYGO is limited to the following:

the IEN (internal entry number; done by the system)
the .01 field value
the values for any fields declared as identifiers

LAYGO entry requires that a value be entered for any mandatory identifier. For non-mandatory identifiers entry is prompted under LAYGO but can be overridden by pressing the enter key and entering a null value.

The following should be noted with regard to pointer field insert operations (affecting inserts and updates):

Pointers reference records and a pointer can only reference a single record of a file; the internal pointer value is actually the IEN of the pointed-to record; the visible (evaluated) pointer value consists of the contents of the .01 field of the record pointed to.

A pointer to a subfile .01 field is not allowed.

The .01 field of parent table is not subject to a uniqueness constraint, even if identifiers are used.

The .01 field of a parent file or subfile cannot contain null values.

A matching value of the same domain (subject to the same input restrictions) is assured by a pointer-to-a-file data type.

Entry dialog uses the FileMan partial match but the entry selected for insertion must exactly match a current entry.

The value of the .01 field in a parent file can be changed (updated) when that field has dependent records in the same or another file since only the pointer value (IEN of pointed-to record) is actually stored in the appropriate fields of the referencing file and evaluation of the updated value is immediately effective for users of the referencing (dependent) file with respect to all dependent records.

More than a single insertion (adding of a new entry) can be required where pointer chain relationships are defined involving multiple files with .01 field pointer data types; assuming Files 1, 2, and 3 with the .01 field of File 2 pointing to File 1 and the .01 field of File 3 pointing to File 2, a user of File 3 adding a value in File 1 but not in Files 2 and 3 will be prompted in sequence to add records to Files 2 and 3.

The following should be noted with regard to pointer field delete operations on the referenced (pointed-to) file of a pointer relationship:

Deletion of a referenced (parent) record affects all current referencing (dependent) records.

If the pointer field is a .01 field, deletion of a referenced (parent) record with a current referencing (dependent) record can cause deletion of that record rather than modification of a pointer field value in the record;

Cascade delete chains can occur in transitive pointer relationships involving multiple files with .01 field pointer data types; a deletion attempt will cause the user to be notified in dialog, as deletion-handling options are presented, only about the impact on the immediately dependent (adjacent) file.

A relational theory model was presented at one time for having multiple target files as is the case for the FileMan variable pointer data type.[17] The relational concept of a multiple target foreign key declaration required that all target keys be from the same domain although the dialog enforces specifying which file is affected in using the LAYGO option with a variable pointer. In FileMan a single domain is not required in the variable pointer relationship. Another related referential integrity variant was presented by Rennhackkamp as the "fan-out" referential integrity constraint where "a single foreign key in a table can refer to the key of one of many tables" and "typically" involving a discriminating condition.[18]

The FileMan database system pointer option supports queries that involve more than one table. *Extended pointers* are used in queries, report generation, and in defining derived fields. Three categories of extended pointers are available under

the *computed field* option when files are being created: simple, join, and backwards. Pointer relationships implicit in computed fields through use of extended pointers are supported through referential integrity provisions. Since File Manager multiple-valued fields and multi-line word-processing files may be referenced by extended pointers, sets of values may be returned on use of a single query. [19] The provision for backward extended pointer references to files and fields includes a check of whether a corresponding cross reference (on the pointed-to or referenced field) has been established. The join extended pointer does not require a pointer (data type) relationship between files.

Summary or Conclusion

Although the FileMan referential actions do not conform directly to those of the SQL standard, FileMan possesses a sophisticated system of support for referential integrity. These referential actions are carried out automatically by the system, involving dialog with the user, to maintain referential integrity:

- Insert rule with LAYGO not allowed which prevents insertion of values into pointer fields of referencing files which do not currently exist in the .01 fields of corresponding referenced files (for pointer and variable pointer data types)
- Insert rule with LAYGO allowed which permits a user of a referencing file editing a field with pointer or variable pointer data type to insert new values into .01 fields of corresponding referenced files prior to insertion into the field of the referencing file; automated LAYGO insertion is limited to entry of a new record, including only the IEN and .01 field values and any additional identifier (can be overridden) and mandatory identifier fields (a value must be entered)
- Implicit insert rule for subfiles requiring a parent record
- Delete rule for referenced file (parent) records which permits a user to set referencing (dependent) file record pointer values to null or to another currently existing value in the corresponding referenced file prior to deletion of a record in the referenced file having dependent records in the referencing file (subject to override through declining)
- Delete rule for referenced file (parent) records that rejects attempted deletion for files that are referenced through pointer or variable pointer relationships
- Implicit cascade delete rule for subfiles

Additional relevant properties of the system can be characterized as follows:

- Uniqueness of records for the system is provided by the IEN record identifier
- Records are identified as unique by users through inspection of the .01 field, in some cases together with declared identifier fields as a form of composite key; uniqueness for fields other than the IEN can be implemented only through custom programming
- A not-null constraint can be implemented by use of mandatory identifiers (if programmer override options are not used)
- The pointer-to-a-file data type assures that only values currently existing (before the attempted insert edit if LAYGO is not allowed and immediately after a successful LAYGO action if LAYGO is allowed) in the pointed-to file .01 field can be inserted and thus provides that all values in the pointer and pointed-to fields are drawn from the same domain (subject to the same input transform)
- In use of the variable pointer (multiple target) data type, an inserted value must be drawn from the domain of the .01 field of a single pointed-to (referenced) file

Comparison with the referential integrity provisions of the SQL standards and SQL implementations may assist in FileMan database design and use and also in planning for FileMan development.

Acknowledgements

The research for this paper was supported in part by a contract from the U. S. Department of Veterans Affairs Information Systems Center, San Francisco, California. Appreciation is expressed for assistance from Tami Winn, Maureen Hoye, Greg Shorr, Richard Davis, and others.

Notes

1 Tami K. Winn and Maureen L. Hoye, "Relational Features of VA FileMan," *MUG Quarterly* 21, 3 (1991): 46-51. See also Catherine Pfeil and Maureen Hoye, "Today's

FileMan," *M Computing* 1, 1 (February 1993): 44-45.

2 Kyle A. Lindner, David J. Whitten, Linda S. Elling, and Gerald P. Bodey, "Ntelligence: A FileMan-Based Expert System," *MUG Quarterly* 20, 1 (1990): 64.

3 *VA FileMan User's Manual*, Version 19.0 (San Francisco, CA: Information Systems Center, Department of Veterans Affairs, 1992).

4 C. J. Date with contribution by Andrew Warden, *Relational Database Writings 1985-1989* (Reading, MA: Addison-Wesley, 1990), p. 23.

5 T.-W. Ling, and C. H. Goh, "Logical Database Design with Inclusion Dependencies," *Proceedings of the Eighth International Conference on Data Engineering*, Tempe, Arizona (February 1992): 642-649.

6 Bruce M. Horowitz, "A Run-Time Execution Model for Referential Integrity Maintenance," *Proceedings of the Eighth International Conference on Data Engineering*, Tempe, Arizona (February 1992): 548-556.

7 Victor M. Markowitz, "Safe Referential Integrity Structures in Relational Databases," *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain (September 1991): 123-132.

8 Richard G. Davis, *FileMan: A Database Manager User Manual* (Washington, DC: National Association of VA Physicians and Computec, 1987), p. 1-6.

9 C. J. Date, *An Introduction to Database Systems*, 1:758, 777, 797.

10 Victor M. Markowitz, "Referential Integrity Revisited: An Object-Oriented Perspective," *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia (1990): 578-589.

11 Marianne Follingstad, "Controlling Data Entry in FileMan-Based System Development or Expansion," *MUG Quarterly* 20, 1 (1990): 7-13; and "Controlling Data Integrity in FileMan-Based System Development," *MUG Quarterly* 21, 3 (June 1991): 101-105.

12 C. J. Date with Andrew Warden, *Relational Database Writings 1985-1989*, pp. 140-141.

13 C. J. Date with Andrew Warden, *Relational Database Writings 1985-1989*, pp. 159-163.

14 Haim Kilov and James Ross, *Information Modeling: An Object-Oriented Approach* (Englewood Cliffs, NJ: Prentice Hall, 1994), pp. 209-212.

15 Richard G. Davis, vol. 1, p. 9-22.

16 Richard G. Davis, vol. 1, pp. 3.8-3.9.

17 C. J. Date with Andrew Warden, *Relational Database Writings 1985-1989*, pp. 163-165; see also Codd, E. F., "More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information)," *SIGMOD Record* 16, 1 (March 1987): 42-50.

18 Martin Rennhackkamp, "Unconventional Referential Constraints," *DBMS* 6, 9 (August 1993): 55-58, 60-61.

19 Richard G. Davis, *FileMan: A Database Manager User Manual*, Vol. 2 (1990), pp. 53-85; see also Beza, Fil Y., Jr., *Intermediate VA FileManager* (Martinez, CA: VA Medical Center, 1991).