A STRATEGY TO IMPLEMENT A DYNAMIC DENTAL EPIDEMIOLOGICAL RELATIONAL DATABASE

D. Taylor[1], R.N.G. Naguib[1], S. Amin[1], A. James[1], S. Boulton[2]
[1]BIOCORE, School of Mathematical and Information Sciences, Coventry University, Coventry, UK
[2]Dental Public Health, Coventry Health Authority, Coventry, UK

Abstract - **Dental surveys to collect data on the developing dentition of school aged children are routinely performed by Health Authorities in the UK. The information collected varies greatly from survey to survey, so that each survey file has a unique structure and consists of unique data elements. The data itself is currently stored in such a way that analysis of data between different survey sets is not possible. This paper investigates a method of modelling a dynamic data store capable of accepting differing historical survey data sets, and studies the functionality required to support the dynamic data store design.**

Keywords - **Dynamic data structure, dentistry**

I. Introduction

Dental epidemiological data have been collected and stored by Coventry Health Authority in the UK for many years. Survey data dating back to 1989 exist for school aged children from the ages of 5 to 16. At least one survey exists per year, and each survey file on average contains over 3,000 records. Similar numbers of records have been collected by most other Health Authorities in the UK. To date there has been no attempt to collect the data into one central data store. All the historical data have been entered into a DOS based survey program. The program stores the data in two ASCII files as follows. The data file consists of the collected survey data as a series of fixed length records, with no delimiters between the individual data elements. The definition file describes the data elements contained within the survey. The definition file consists of a list of field names, each field name having a field length and field type as illustrated in Table I.

There are two distinct field types. The first type is either a string or numerical field of specified length. The second type is referred to as *Coded*, and consists of a defined length numeric field of one or more integers, together with a description of the possible values and meanings associated with the integers.

Each survey may have a unique data file structure where each individual record within the file consists of many individual data elements. A corresponding definition file describes the sequence, type and length of data elements within the data file. To perform more in-depth analysis of the data it is desirable to store the data sets in a single data store and remove the need for the definition files. Ideally the data should be stored within a well designed database to optimise the use of the data.

TABLE I
Example Data File and associated Definition File.

Taylor.... High St….. 010119851012..Smith..... Low St…...
050519852331..Jones…..Centre Rd…...0606198512….

Field	Length	TYPE	NOTES
Name	10	String	
Address	15	String	
DateofBirth	8	String	
Gender	1	Coded	1 = male 2 = female
Attendance	1	Coded	0 = 6 monthly 1 = yearly 2 = pain 3 = no reply
Sport	4	Coded	1 = Football 2 = Rugby 3 = Boxing 4 = Hockey

II. Designing a Dynamic Data Store

The traditional approach to constructing a relational database application advocates the logical design of the intended relational database in the form of a model followed by the construction of the database schema based on the logical model [1,2]. Clearly in the case where data elements collected for any given survey may be unique to that survey, the structure of any proposed relational database needed to store such data is not known until the information is collected and ready for inclusion into the database. As the structure of the database will change, then some method of reflecting and managing structural changes, and recording the distribution of data within the database for each data set is needed [3].

If we adopt the approach that:

- A table PERSON will be created to hold details of each individual child within the survey,
- All data elements of type *String* and *Number* can be regarded as attributes of PERSON,
- All data elements of type *Coded* with length =1 can be modelled as a One : Many relationship. This involves the creation of a relation *(the detail table)* to hold the information outlined in the notes field of the definition file, and the creation of a foreign key domain in the PERSON table to hold the data from the data file,

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
25 Oct 2001	N/A	-

Title and Subtitle A Strategy to Implement A Dynamic Dental Epidemiological Relational Database	**Contract Number**
	Grant Number
	Program Element Number
Author(s)	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) BIOCORE, School of Mathematical and Information Sciences Coventry University Coventry, UK	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es) US Army Research, Development & Standardization Group (UK) PSC 802 Box 15 FPO AE 09499-1500	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)

Distribution/Availability Statement
Approved for public release, distribution unlimited

Supplementary Notes
Papers from 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society, October 25-26, 2001 held in Istanbul, Turkey. See also ADM001351 for entire conference on cd-rom.

Abstract

Subject Terms

Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU

Number of Pages
4

- All data elements of type *Coded* with length >1 can be modelled as a Many:Many relationship. This involves the creation of a relation *(the detail table)* as above, and the creation of a second relation *(the link table)* to hold the data from the data file,

then we have the basis for mapping the data elements to a relational database structure. We can therefore create a dynamic data store capable of adapting to store the data from any data file. Our definition of a dynamic data store is 'a database structure that can be adapted to store differing data elements by the creation of domains, detail tables and link tables'.

In order to construct and manage a dynamic database structure that will change as new data files are processed, an internal representation of the database schema and a record of domains occupied by individual data files is required.

The Database Definition Structure (DBDS), utilises the concepts found within the data dictionary, extended to describe the tables, domains, and the relationships between the PERSON table and other detail and link tables (see Table II). Each record within the DBDS consists of the base table name, the domain name, the domain type, and a domain Tcode that acts as an indication of the function of the domain. This structure can be referenced and used by a database interface, that accepts and processes data files. Based on the information supplied by the definition file and the DBDS, decisions can be made as to which tables and domains each survey data element from each data file will be stored to.

There needs to be a distinction between domains that are available to be used to store raw data elements from the data sets, and domains used by the database to maintain the data structure by means of the primary and foreign keys.

The DBDS therefore serves several purposes. It defines the domains for each relation in terms of the data type, and also the function of each domain. Tcode values reflect the function and type of each domain within the database.

- Domains of type AT within the PERSON table can be used to hold data of type *String* and *Number* from the data file.

- Domains of type FK within the PERSON table can be used to hold data of type *Coded* (length 1) from the data file. The single value held in this type of domain matches a primary key value from the corresponding link table.

- Domains of type LF can be used to hold data of type *Coded* (length >1) from the data file. There may be many entries in a link table from one data element. The single value held in this type of domain matches a primary key value from the corresponding link table.

TABLE II
An Example Database Definition Structure.

Tablename	Domain	Datatype	Tcode
Person	PersonID	Number	PK
Person	Name	Text	AT
Person	DateofBirth	Date	AT
Person	Gender	Number	FK
Gender	ID	Number	PK
Gender	Txt	Text	AD
Sport	ID	Number	PK
Sport	Txt	Text	AD
SportLink	PersonID	Number	LP
SportLink	ID	Number	LF

Tcode values

PK primary key (in PERSON and *link* tables)
FK foreign key (in PERSON)
AT attribute of PERSON table
AD attribute of *detail* table
LP PersonID foreign key in *link* table
LF Foreign key to *detail* table

The Data Definition Structure (DDS) records which domains the individual data sets have been loaded to. This will distinguish between a NULL value stored in a domain where data may have been expected for a particular survey data set but is missing or unknown, and a NULL value stored in a domain where no data was associated with the data set to begin with.

The DDS consists of three domains. The first two domains together hold details of which table and which particular domain the data is held, the two domains forming the primary key of the DDS.

III. Database Interface Functionality

The information presented within the definition file provides the basis of identifying the data elements, the data element types and the data element lengths held within the corresponding data file. In order to process the data file and add the data elements to the database, an interface is required that accepts the data and maps data elements to the correct domains, or creates domains, detail tables and link tables, whilst maintaining the DBDS and DDS structures.

Consider the logical steps that need to be followed in order to process the example data file shown in Table I to a new database. The database consists initially of the three tables PERSON, DBDS, and DDS. The DBDS initially contains two tuples (PersonID and Name), the other two tables being empty. The DBDS is referenced to list all domains currently available to store data from the data file. A list of all existing domains available for the storage of data elements is indicated

TABLE III
An Example Data Definition Structure.

Table	Domain	Dataset
Person	Name	5yr89
Person	DateofBirth	5yr89
Person	Gender	5yr89
Sportlink	ID	5yr89

by the Tcode domain value of the DBDS, where the value is one of AT, FK or LF. Data elements of type *String* and *Number* are to be mapped to a domain of type AT within the PERSON table. Coded data elements of length 1 are mapped to a domain type FK within the PERSON table where reference is made to the corresponding detail table. Coded data elements of length >1 are mapped to a domain type LF where reference is made to the corresponding detail table.

A. Mapping Data Elements to Existing Domains

The first data element from the example data file contains the name of the person as a character string of length ten. We wish to map the data element to the existing NAME domain. The data element is mapped to the NAME domain of PERSON, the DDS is updated to reflect the addition of data to the database.

B. Creating New Domains in the PERSON Table

The next fifteen characters of the data file contain the address of the child. The DBDS table does not offer a domain suitable to store the data field, and so a database interface function is required to add a domain ADDRESS to the PERSON table of type Text. Domains created by this routine are assigned a Tcode value of AT. The DBDS is updated to reflect the change in the database structure. The data field ADDRESS can now be mapped to the newly created domain. The DDS is updated to reflect the addition of data to the database.

The next data element described by the data definition file is a string of length eight, representing the Date of Birth of the child. This data element is handled in the same manner as the address data element.

C. Creating New Coded Domains (length =1)

Data elements of type *Coded*, where the coded length is 1, need to be stored as a value within a foreign key domain in the PERSON table. The value stored within the foreign key domain must match one of the values in the associated primary key domain of the detail table.

The fourth data element to be processed from the file is a coded domain of length 1, representing the gender of the child. No currently existing suitable domain of Tcode type FK exists within the PERSON table to store the data element, and so a database interface function to create a new foreign key domain and a new table is required. A new table GENDER is created with two domains, a primary key (ID) and a domain (Txt) that holds the description of the value of the primary key. A domain GENDER is also created in the PERSON table. The DBDS is updated to account for the addition of the new GENDER table, and the new GENDER domain in the PERSON table.

The values within the new GENDER table can be entered as described in the notes column of the definition file. Two rows of data are added to show that a male child will be represented by a primary key value of 1, and a female child by the value 2. The gender data element can now be mapped to the available GENDER domain and the DDS updated to reflect the addition of the data to the database. The values added to the GENDER domain in the PERSON table must match one of the values of the primary key of the GENDER table.

D. Creating New Coded Domains (length >1)

Data elements of type *Coded*, where the coded length >1, need to be stored as a value or values within a link table. The values stored within a link table comprise the primary key value of the PERSON table and a primary key value of the associated detail table.

The fifth and final data element to be processed from the data file is a coded domain of length 4, representing the contact sports that the child might partake in. No currently existing suitable domain of Tcode type LF exists to store the data elements, and so a database interface function to create two new tables, a detail SPORT table and a link SPORTLINK table is required. A new table SPORT is created with two domains, a primary key (ID) and a domain (Txt) that holds the description of the value of the primary key. The notes column of the data definition file can now be used to insert the values detailing the sports and associated values. A new table SPORTLINK is created with two numeric domains, PersonID and ID. PersonID holds the value of the primary key from the PERSON table, and ID holds the value from the coded domain, which must match one of the values of the primary key of the SPORT table. The structure of the SPORT and SPORTLINK tables together allow the Many : Many relationship between person and sport to be modelled [4]. The primary key of the SPORTLINK table comprises both domains within the table.

The data file can now be processed and all the data stored within the correct domains. The final DBDS and DDS database structures after the processing of the example data file are illustrated in Tables II and III, respectively.

IV. PROCESSING ADDITIONAL DATA SETS

In addition to the database interface functions already discussed, two additional features are required. Consider a new data set comprising of three data elements: childname,

gender and sport. The first data element is mapped to the name domain within the PERSON table, childname being a synonym for name. In this case the operator managing the processing of the data set selects the appropriate domain to map the data element to.

For the second element, gender, the definition file may define 'Male' as value 0 and 'Female' as value 1. If the values existing within the detail table differ from those within the definition file then we can map the proposed values from the data file to existing values within the detail table. In this case there needs to be a mechanism to allow the translation of values of this data element so that a value of 0 (representing Male) read from the data file is stored as a value of 1 within the database. An additional *MAPPING* function is therefore required.

The third element, sport, may contain additional contact sports not already listed within the SPORT detail table. In this case there needs to be a mechanism to allow additional rows of data to be added to an existing detail table, therefore an *ADDITION* function is required which allows additional values to be inserted into the SPORT detail table but does not allow existing values to be changed. Once any additional sport details have been added, the *MAPPING* function can be used if required to resolve any differences between values stored within the detail table and values to be read from the data file.

V. Discussion

The adoption of a database structure consisting of a PERSON table, with the addition and management of detail tables and link tables, offers a solution to the storage of many data files of differing structures into one data store. The use of the DBDS structure and the DDS structure offer a solution to the management of the database structure and the data sets. A prototype database built using Access2000 and implementing the above structures and database interface functions has been successful to a large degree.

One problem encountered was the naming conventions used in the definition files. As an example, data set A may use the definitions of *name* and *school* to represent the PERSON name, and the school attended. A second data set may use the definitions of *pupilname* and *name* to represent the PERSON name, and name of school attended respectively. Here, the use of the descriptor NAME is misleading as it refers to data that needs to be stored in two different domains. It is this inconsistency factor that prevents the further development of an automatic submission routine to manage the mapping of a data file to the database using the data definition file as a mapping reference.

VI. Conclusions

The management of a relational database where the schema is required to change is possible given the limitations discussed above. It is possible to transfer data sets from the survey ASCII files by using an interface that allows the user to define where to place the data elements, and to create new tables, domains and relationships to accommodate different data structures. The user does not need to fully understand the resolution of One : Many or Many : Many relationships as the functionality of the database interface manages the creation of tables and mapping of values, providing the data type and length values from the data definition file are recorded.

One : Many and Many : Many relationships can be modelled and managed by the use of detail and link tables and the management of foreign key values in domains of Tcode type FK and LF [5]. Management of the mapping and recording of the domains involved in the storage of data elements can be managed by the DBDS and DDS structures.

Automatic interpretation of data sets by a programme is difficult unless a standard naming convention for data elements is adopted [6]. If consistent element names are used from survey to survey, or a table of synonyms is provided to resolve naming conflicts, then in theory a database interface to automatically process data files and definition files may be feasible.

References

[1] C.J. Date, *An Introduction to Database Systems*,7th Edition, Addison Wesley, 1999

[2] K. Silberschatz, H.F. Korth, S. Sudarshan, *Database Systems Concepts*, McGraw-Hill 1997

[3] D. Taylor, R.N.G. Naguib, S. Amin, A. James, S. Boulton, "Design considerations for a relational database schema to hold dental epidemiological data", Proc. IEEE ITAB, Washington DC, USA, 2000

[4] R. Buelow, "The Folklore of Normalization", Journal of Database Management. vol.11, no.3, 2000. pp.37-41

[5] W.Y. Mok, "On Keys and Normal Forms", Information Processing Letters. vol.62, no.5, 1997, pp.255-258

[6] G. Rahmstorf, "Identifying Words, Senses and Concepts", Proc. 5th International Congress on Terminology and Knowledge Engineering, Vienna, Austria, 1999 pp.410-426