DATA CONNECTION AND MANIPULATION OF ARCHAEOLOGICAL DATABASE CREATED IN VISUAL ENVIRONMENT

by
Manuella Kadar

Abstract: The paper discusses aspects as regards specific issues of storing, manipulating and connecting archaeological data by using visual applications. Key areas include optimization of the index configuration, data placement, storage allocation and connectivity between several applications.
Key words : archaeological finds, SQL server, database design

Archaeological projects provides large quantities of written documents, notes, and forms, as well as drawings (plans, sections, and sketches) and photographic images of the ancient sites, architecture, and artifacts that are recovered during survey and excavation. Effectively using all this material in the pursuit of research goals has always been a major challenge. The sheer amount of data which must be processed and evaluated by project members quite often necessitates the adoption of new tools and strategies for interpretation and analysis.

An archaeological site comprises a complex three dimensional matrix of deposits, cuts and interfaces which can combine great physical and chronological depth, continuity, discontinuity and stasis. Huge variations in deposit formation and deformation processes, deposit make-up and deposit visibility make the recording and interpretation of these deposits and their interrelationship both with one another and with the materials they contain, a complex and complicate process.

The emerging technologies of computer-aided drafting (CAD) and surveying instrumentation, remote sensing/satellite imaging, digital scanning, Global Positioning Systems (GPS), photogrammetric mapping, and digital/video (multimedia) imaging are of critical importance for modeling archaeological data, and Geographic Information Systems (GIS) and Exploratory Data Analysis (EDA) software are of critical importance for archaeological analysis.

Data from archaeological excavation is suitable for computerization although they bring challenges typical of working in non-scientific subjective areas. Meaning and significance within data are established on-site and afterwards by a heuristic process of discussion and contestation, a process at odds with the rigorous demands of database design. As concerns artefacts, for example pottery, a possible model of data organization is given in Figure 1.

A common and powerful method for organizing data for computerization is the relational data model. Relational databases have a very well-known and proven underlying mathematical theory, a simple one (the set theory) that makes possible automatic query optimization, schema generation from high-level models and many other features that are now vital for mission-critical Information Systems development and operations.

Key areas include optimization of the index configuration, data placement, and storage allocation (Kadar e.a. 2003).

One of the greatest impacts on the application's performance is the selection of indexes. For operational databases an index that improves data retrieval performance may degrade performance for all kinds of updates, because the maintenance cost for the index has to be paid for each update. Since there is not a lot of update activity in a data warehouse, the number of indexes is not as much of a concern. Selecting the optimal indexes for a given set of tables (an index configuration) is a non-trivial problem that requires trade-offs between the different kinds of database operations (retrieval operations, update transactions, and utilities). Given an index, one must determine its properties, such as the column ordering for multi-column indexes, and whether or not the index should be clustering, partitioning, or ordering.

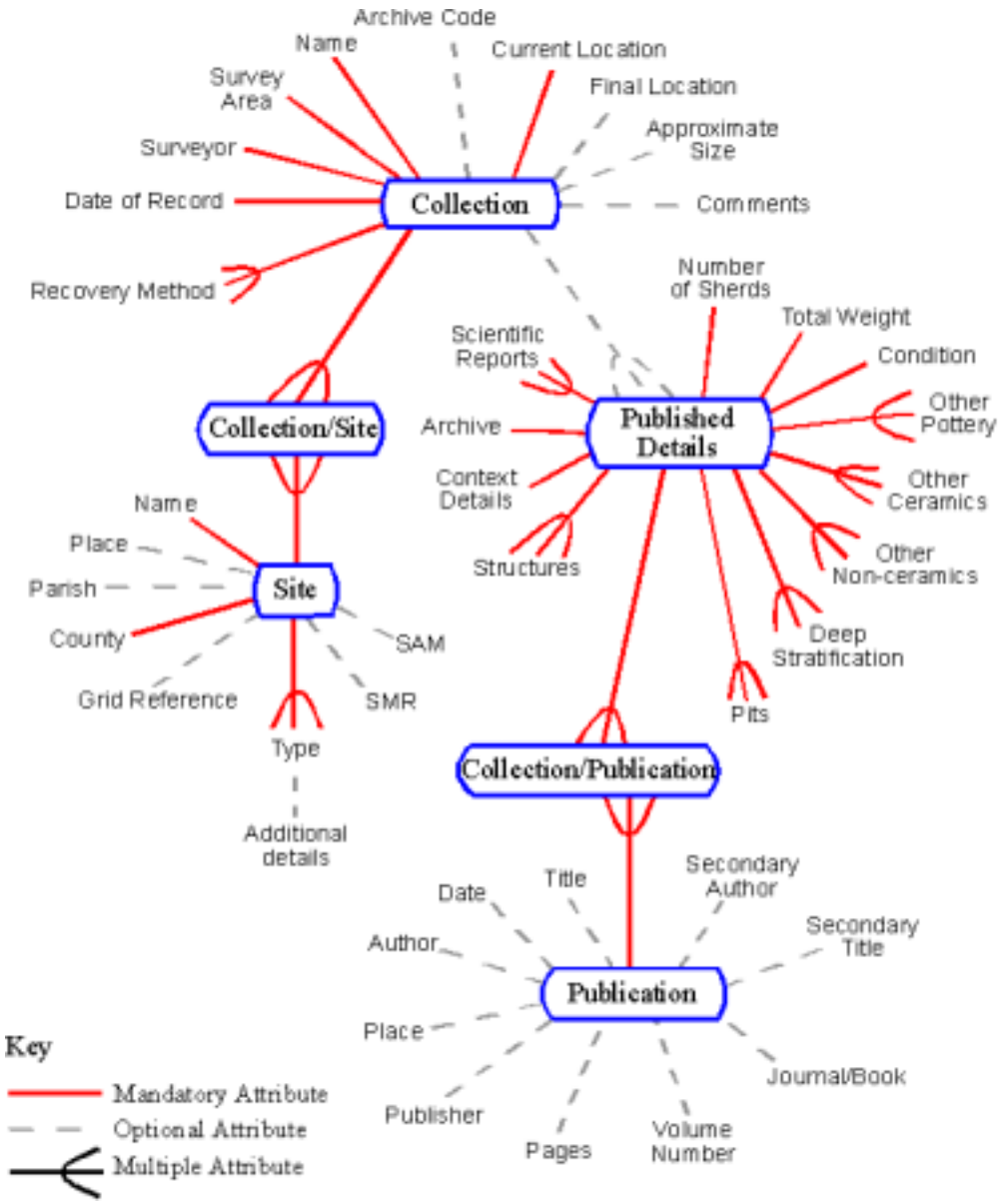

Figure 1. Data model for pottery database (after *English Heritage 1991*)

While index selection is the most important and complex problem in physical database design, data placement and space optimization are important as well. One must allocate the optimal table space to a given table, and choose whether a simple, segmented, or partitioned table space is most appropriate.

Allocate enough space to maintain the clustering properties, without wasting too much space. Index selection, data placement, and space optimization assume that the structure of the relational schema is stable. However, physical database design can also change the relational schema; one may need to split tables into different partitions to take advantage of concurrent operations, or merge tables in cases where different tables are frequently joined, so that performance improves if one merged table represents the stored join. Decisions can also be made on the tuning of the DBMS run-time parameters.

Most applications in archaeology use traditional Entity-Relation modeling and physical database design to create and maintain operational databases. These databases are typically very specialized and are designed to support very specific application requirements. Recent trend is to built data warehouses. These new mega-databases enable end users to access information based on data that was previously unavailable to them in a single place. For the data professional, the newest challenge is to design an optimized relational database that satisfies a much different set of requirements (Kadar 2002).

Visual FoxPro 6.0 offers a full-featured application development system, geared to developing data-oriented applications. Visual FoxPro is both wide and deep, is and a powerful database engine with a rich set of data connection and manipulation functions. Notable among its work spaces are object-oriented visual form and class designers, a tightly integrated report writer, and a database manager. In association with its native database engine, Visual FoxPro blends extended Xbase data management syntax with SQL and a powerful cursor management architecture. Visual FoxPro 6.0. has the ability to create COM components that can be deployed on the server, and has added the ability to create forms as Active Documents.

The Visual FoxPro form design system delivers object-oriented programming capabilities. The visual Class Designer looks very much like the Form Designer, but is oriented to designing objects stored as classes. Visual FoxPro presents the controls that will populate a form as classes so that individual controls on a specific form become instances and inherit the basic characteristics of their class. This approach enables the creation of libraries of reusable custom controls.

The visual form design surface is complemented by the visual class design surface. Very low-level components include form objects (spinners,

combo-boxes and entry fields for other common data elements) and toolbar icons. Mid-level components like entry panels for addresses, line items are used in a variety of places in the application. High-level components like form templates have been used to serve a variety of needs. There are two ways to add objects to a visual class library.

If the class is created explicitly, New Class from the File menu is chosen, the wanted built-in class is selected as the base class, and the visual class library in which the class would be stored is identified. Visual FoxPro then presents a design surface that consists of a layout area and a property sheet where properties for the new class can be set (Figure 2). The tool palette changes to show the created controls in place of the stock controls that were there when Visual FoxPro has been started. Now the custom class can be selected and drop on the current form.

Once an instance of the class has been created this way, further refine of its appearance and behavior can be done for use on a certain category of forms.

The Visual FoxPro local database engine and metadata repository are other key components. Visual FoxPro stores application data in freestanding files (using a .dbf extension), each representing a single logical table. In addition to data files, Visual FoxPro maintains indexes for each table in separate files that have a .cdx extension. As a result of this technology, queries against large databases in Visual FoxPro can perform as fast or faster than against similar-sized SQL Server databases on the same platform.

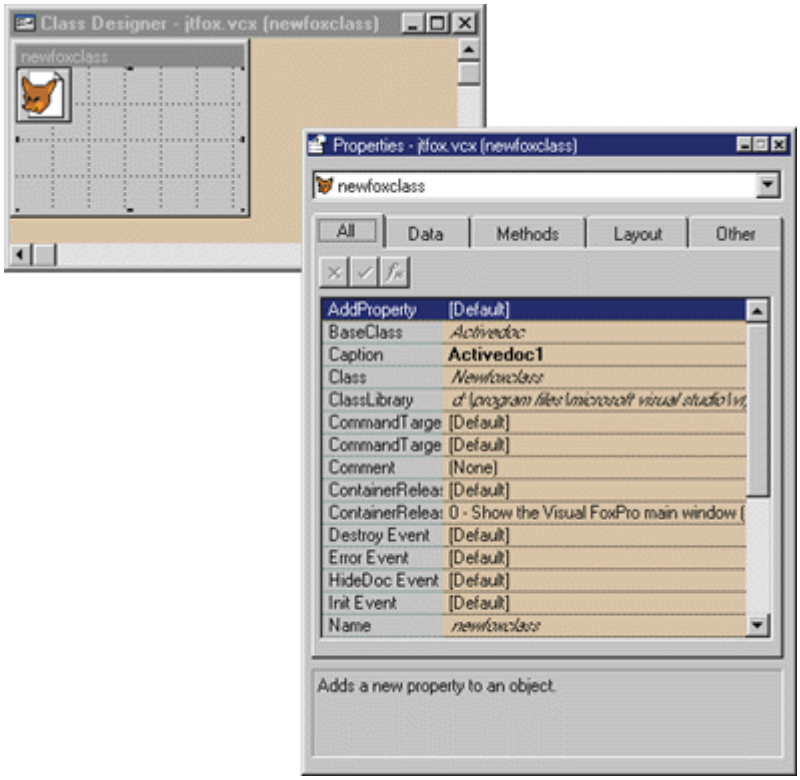

Figure 2. Class designer

Visual FoxPro maintains information about database structures in a database container file (with a .dbc extension), which is really just a specially designed Visual FoxPro table file. You can maintain and specify database rules through language functions such as DBSetProp (which sets the properties of fields, views, and connections) and through the SQL-based CREATE/ALTER TABLE syntax for creating and modifying table structures and indexes.

In Visual FoxPro, there is a visual equivalent for most of the design-oriented aspects of the command language, and the database is no exception. The Database Designer initially presents a blank canvas to which you can add existing tables or create new tables. When one create or modify a table, Visual FoxPro presents the Table Designer (Figure 3), where one can specify field characteristics, including basic data type and size, display format, default caption, validation rule, validation text, default value, and the visual class that should be used to display the field on forms and reports.

Figure 3. Database Designer

Clicking on the Table tab brings up a page where one can specify a record validation rule, which is tested when a record is added to a table. Rules, triggers, and index expressions can call stored procedures that are defined and stored in the database container as well.

Clicking on the Indexes tab in the Table Designer dialog reveals a form where one can define indexes for the table. Indexes can be set as Primary, Candidate, Unique, or Regular. While Primary and Candidate indexes are RDBMS-standard, in Visual FoxPro the Unique and Regular indexes are a little different. Where an index contains multiple records with a common index value, a Unique index stores a pointer to only the first physical record that contains the index value; the index simply ignores the second or later occurrence of a non-unique value. A Regular index is simply an index that is not Unique, Primary, or Candidate. It can be used for ordering and seeking records, and as the target index for the many side of a one-to-many persistent relationship.

To support programmatic data administration, Visual FoxPro supports standard SQL data definition language (DDL), providing a command set that includes the following for establishing a default value and validation rule for a field, and for establishing candidate and foreign keys, table-level validation rules, and relationships.

With Visual FoxPro, one can access server-based data via either persistent application-level connections, which are stored as remote views in

the FoxPro database container, or through transient connections established at execution time. Configurable defaults for persistent remote views include whether to use the current connection for new views, the maximum number or rows to be returned to a view, how many rows to return in a single fetch, whether to fetch memo fields with their records or wait until display of the memo data is requested, whether to compare key fields or key fields plus updateable fields to determine whether server data needs to be updated, and whether to perform updates using UPDATE or DELETE and INSERT. Configurable defaults for connections include whether to enable asynchronous or batch processing and the amount of time to wait before timing out in several connection scenarios. In the interactive View Designer, one first select the tables to include in the view and their relationships, then specify selection criteria, fields, ordering, grouping, and update criteria. The update criteria lets to specify which columns can be updated, which columns should be treated as keys for matching buffered rows back to rows on the server, and whether one want to override the defaults for selection and update of rows whose values have been changed. The mechanics and syntax of establishing a data view and the capabilities the view provides are identical whether the underlying tables are FoxPro tables or server-based tables. Once a view is defined, one can manipulate it via either SQL or Xbase commands.

Another issue addressed in this paper is the connection between an EXCEL sheet and the table of the database in Visual Fox Pro. To execute an SQL SELECT against a Microsoft Excel version 7.0 workbook, one may need to execute the SQLTABLES() function to get the names of the worksheets that reside in the workbook.

The general syntax is:

```
handle  = SQLCONNECT(<data source>,<username>,<password>)

success = SQLTABLES(handle)
```

This will build a cursor that one can then browse to see the actual names of the worksheets to use. The cursor created by the code has five fields, the third of which contains the table names. In the case of Microsoft Excel version 7.0, these table names are actually the names of the worksheets within the workbook to which one have connected.

They all have a dollar sign ($) as the last character, and one must be sure to include the dollar sign when accessing the worksheet.

Here is an example of an SQL SELECT to a Microsoft Excel version 7.0 file:

```
handle  = SQLCONNECT('Excel 7.0 data source','','')

success = SQLEXEC(handle,'select * from "sheet1$"')
```

Note that the sheet name is surrounded by double quotation marks, with the entire select statement inside single quotation marks. This is the required syntax.

References:

[English Heritage 1991] - English Heritage, Management of Archaeological Projects, ISBN 1-85074-359-2, London, 1991.

[Bazian e.a. 2001] - M.Bazian, J.Booth, J.Long, V.Miller, C. Silver, R. Byers, *Totul despre Visual FoxPro™ 6,* Ed. Teora, Bucureşti, 2001.

[Kadar 2002] - M. Kadar, *Data modeling and relational database design in archaeology*, în *ActaUA*, 3, 2002, p. 73-80.

[Kadar e.a. 2003] - M. Kadar, V. Bucur, E. Ceuca, *Relational Database Management System for the Early Metallurgy of Copper and Bronze in Transylvania,* in *Proceedings of the 30th Conference of Computer Applications in Archaeology CAA2002*, (ed.) Martin Doerr and Apostolos Sarris, Atena, 2003, p. 335-339.

Author:

Manuella Kadar - “1 Decembrie 1918” University of Alba Iulia, Romania