United States Educational Foundation in India,
DRTC/Indian Statistical Institute,
DLIS/University of Mysore
Joint Workshop on Digital Libraries
12th – 16th March, 2001

Paper: I

Library Digital Resources: Relational Databases or Catalogues? (A Short Tutorial)

Gurunandan R. Bhat

Department of Computer Science and Technology

Goa University, Goa

Abstract

With the rush to deploy Information Technology in all service areas, there is tremendous pressure on small and medium size (around 1 lakh holdings) libraries in India to digitize their records. In doing so, Librarians wishing to go digital face the risk of opting for platforms and back-ends not truly designed for managing library records. The fact that designers of almost all Library Management Systems currently popular, have opted for the Relational Database model in structuring library data, further confounds the Librarian's choice and almost always leads to the deployment of inappropriate tools. This article (tutorial) argues that Library records cannot be conveniently cast in the conventional RDBMS framework without significant modifications to the structures of Library data as librarians have conventionally structured it, as well as key concepts in conventional RDBMS. It is the premise of this article that these modifications can be made only at the risk of seriously detracting from the effectiveness of the tools as well as the quality of the data. The premise here is that Library data are not relational databases but catalogs in the sense that they cannot be normalized to fit the canonical normal forms. An alternative framework of tools (WWWISIS back-end from the ISIS family) and structured data (ISO 8059) is suggested and explained.

1 Introduction

Designers of Library Management Systems will notice that Library records cannot be forced in the conventional Relational Database Model without either seriously comprising the effectiveness of the data or making significant and usually ugly changes to the Database Model itself. Here are a few examples and typical scenarios:

- Library Records frequently demand multivalued attributes: Books can have more than one author and Authors write many books. In view of the fact that RDBMSs demands atomicity (one field one value) forces the need for three tables where one would have sufficed[1]. While this perfectly legitimate and indeed more robust[2] real world exigencies frequently require modifications in design. This can happen where an author presents himself differently on different titles. Is William Shakespeare the same bloke as W. Shakespeare or Shakespeare William? Should he appear as different records in the author table? Or should one create a table of all known aliases of an author?[3] What if an alias for an individual author happens to be the legitimate name of another

[1] A table of book-titles and their identification tag (id), a table of author details and its id and separate table linking each author-id to a single book-id spread over multiple records spanning all authors and all books

[2] If the details of an Author change, a change at one single point (the table of author details only) reflects everywhere. The data is always in a consistent state no matter when the change is made.

[3] Again since aliases are not atomic, records proliferate!

author?[4] The core problem here is that conventional RDBMSs treat (as they rightly should) each record as an *"entity"* rather than a collection of fields. When two entities frequently have similar (or even identical) attributes, the conventional designer is forced to incorporate other attributes to distinguish between them[5]

- Most OPAC[6] queries are free-text or pattern searches and conventional RDBMSs need to work very hard when asked to process these. While there has been tremendous progress in optimizing pattern searches via the regular expressions syntax, these have yet to be incorporated fully or most efficiently in most RDB back-ends. The key here is the way a conventional RDB indexes fields. Indices are almost always atomic, never for multiple sub-parts. Client side SQL does support the 'LIKE' keyword but has to work extremely hard to support it at the back-end. What is required in OPACs is an intelligent method to index records based on patterns rather than atomic values.
- Most Libraries require an OPAC that is integrated with a Management System to handle lending, acquisitions, payments and reminders. The problem is that OPACs work best in the catalog (pattern indices rather than atomic indices) mode whereas the management system works best in the conventional RDBMS mode and resource starved libraries are reluctant to deploy multiple systems with these disparate capabilities. The need here is for a catalog like application that also allows the rich capabilities of conventional RDBMSs like efficient inserts/updates, transaction control, replicability and rapid deployability.
- Serial (Technical Journals and Magazines) control has always been the bane of Library management Systems in developing countries where periodicals are hardly ever delivered periodically. Many suppliers bundle several issues together to save mailing costs. Publishers frequently bring out non-periodic special issues to commemorate events. To compound the misery of the conventional database designer, each serial has its own method of naming volumes depending on its periodicity. Trying to fit wildly varying serials systems into a structured conventional RDB is extremely difficult to say the least. An unpredictable system such as this is further disturbed by the fact that serials are first separate and then after a period (more than one year) bound together in an independent volume. Since the Library Bindery is probably the most stretched section, it is not unusual for this period to stretch to around five years.
- There are hundreds (or even thousands?) of libraries in India that have the a huge number of common titles[7]. It would be advantageous to distribute the digitised catalog of one library to another so as to leverage the automation

[4] Matrix Diagonalisations" by L. Prokofievich and L.Prokofievich!!

[5] for example L. Prokofievich (the father) and L. Prokofievich (the son) thus necessitating, say an age attribute in the author table

[6] Online Public Access Catalogs

[7] Think of all the College and University Libraries. Most College and University Texts and Reference books are the same. A random intersecting sample of our University Library with that of the neighboring Mangalore University showed up an overlap of more than 80%

process. All that a new Library is required to do is to keep the bulk of the fields the same and only change a few attributes to reflect locale. Most RDBMSs store data in proprietary formats and require tricky inter-format translations making leveraging other initiatives difficult or sometimes impossible. What is required here is robust and active support from RDBMSs for a Common Communication Format for titles and serials.

- Conventional High-end databases (at least those with enough feature list to be useful to Libraries) are expensive and Commercial Library Management Systems built around these even more prohibitive[8]. A commercial solution (not costing upgrades) weighs equal to three years annual subscription of a leading and valuable American Physics Journal.
- In view of these and other problems in using conventional RDBM tools, the Goa University Library is currently experimenting with the ISIS family of Bibliographic tools for its Library OPAC and Management back-end. While the ISIS suite has not yet matured to the extent of providing a complete solution to Library needs, it is definitely a frontline candidate in terms of its potential.

2 The ISIS family of bibliographic tools

2.1 The backend view

The ISIS view of a Database is more akin to a Catalog or a Directory, where data is structured not in terms of related but rigid tables with strong integrity relationships, but more in terms of a loose hierarchy of attributes. In this sense the ISIS view of data is similar to that taken by Network Directory services such as X.25 and LDAP or flat text Unix databases such as *gdbm.* In what follows we shall describe the the structure of an ISIS database

2.1.1 The data definition scheme

Data in the ISIS family of tools is organised in conformance with the *'Field Definition Table'* (FDT). The Field Definition Table describes the structure, type and characteristics of the records in the Database completely. Every field is characterised by a numerical tag used to reference it. A field (unlike those in a RDB) can have multiple sub-fields. To the ISIS backend, a field (or a sub-field for that matter) is just a continuous string of characters. In the event that a field consists of sub-fields, the FDT also defines the *'sub-field delimiter'* that separates one sub-field from the next. To the backend, a sub-field is treated in the same manner as an independent field in the sense that sub-fields can be independently indexed[9]. The address field of a publishing house for example can be structured in terms of four sub-fields (say) *address-line-1*, *address-line-2, city-name* and *country-name*. The sub-field delimiter is usually a caret ^

[8] A leading library management system comes with a total bill of around Rs. 4.00 lakhs and rising!
[9] and therefore can form the parametric basis of a faster search

followed by an alphanumeric character such as *^a*. For example the address field of a Publisher manifests in the database of Publishers may appear as

```
^a66, Samudra Mahal^bDaryaganj^cNew Delhi^dIndia
```

In many cases however, the number of sub-fields of a field cannot be known a-priori. A good example is the author field. Since the number of authors of a title is variable, it is not possible to fix the number of sub-fields in the FDT. In such cases, one uses a *'repeatable'* field to characterise this attribute. While the individual occurrences of a repeatable field cannot be referenced separately (as can be done for sub-fields) one can index them and use them as parameters in a search. In summary then, the FDT characterises the records in a particular database by specifying the following parameters:

The Field Tag is a (arbitrary) numerical tag for each field. This tag is used for extracting the tagged field from the record and to pass it as a label for use by the indexing engine

The Field Name is a descriptive name for the field. It is used as a heading for fields but there is no way of accessing it directly and exists only for informational purposes only. Internally ISIS uses only the tag to label or reference the field.

Field Length The field length can be up 1650.

Field Type Field types can be one of four: **X** for alphanumeric, **A** for exclusively alphabetic, **N** for exclusively numeric and **P** for a field that conforms to a predefined pattern.

Repeatability Whether the field is repeatable **(R)**. Since there is no limit (except total record size) to the number of times that a field may repeat this number is not specified separately[10].

Sub-field Delimiters/Pattern Specifier This attribute of a field specifies the alphanumeric character that will separate each sub-field (for example *abcd* above). If the field happens to be one that is required to conform to a pattern, then that pattern is specified here.

2.1.2 The data reference scheme

Any database requires an identification scheme to refer to the individual fields in a record. In conventional query languages like SQL, the field name is the identifying tag. In ISIS, references to fields (or sub-fields) is through the tag as specified in the Field Definition Table (FDT). For example one specifies the field with tag number **32** as **v32**. The sub-field **c** of this field (if it has one) is referred to as **v32^c**. In addition to

[10] If the number of times a field could repeat is known a-priori, then it might be better formatted as a sub-field

individual fields, sub-fields or patterns, ISIS also provides a referencing scheme for fragments of a field using the **offset.length* to refer to character starting at *offset* (base is taken to be zero) and continuing up to *length*[11] as well as entire records.

In addition to referring to individual fields, ISIS also has a rich scheme for formatting or indenting a field or successive sub-fields within a field. These may be gleaned from the ISIS manuals.

2.1.3 Control, logic structures and inter-database references

In addition to referring to individual fields and sub-fields, we also need control and logic structures to add power to our applications. These are provided through the simple boolean comparison operators and structures such as:

```
IF condition THEN tag-1 ELSE tag-2 FI
```

In addition to these elementary control structures, one can maintain a simple level of referential integrity between databases with the **REF** feature. While this is certainly not at the same degree of power as foreign and primary key mechanism in conventional RDBs, it is adequate for bibliographic applications. There are no triggers (yet!) in the ISIS family.

2.1.4 Database backend file scheme

Apart from the data (records) in a database, an efficient back-end also requires a host of files with pre-cooked indexes to speed up access. All these files are formatted to be accessed speedily without regard to their readability. The ISIS family uses the following files for this purpose

The *iso* **file** contains the records packed using the ISO 2709 format. A simple perl script to unpack and extract the individual fields and records is described later.

The *mst* **file** is the master file containing all records packed in a proprietary format. Each record is divided into 3 sections: the leader to keep track of sequencing (both forward and backward), a directory to speed up the location of individual fields and the data itself. The mst file is stored as series of 512 byte blocks.

The *xrf* **file** contains pointers to individual records in the master mst file. Using this cross-reference file, one can get at individual records speedily.

The inverted files are six files with extensions *cnt*, *ifp*, *n01*, *n02*, *l01* and *l02*. All these files contain indexes of the fields in the master table. You can visualize them as binary files that record the position of each indexed term in the master file. The ISIS tools store two indexes, one of terms with ten or less characters and the other with

[11] For example if we want the fifth, sixth and seventh characters of the sub-field c of field tagged 32, we refer to it as v32^c*4.2. Notice that the offset begins with 0, the fifth character is at offset 4

terms of more than 10 characters. ISIS provides tools to generate these index files (inverted files) from the *iso* file.

3 The application layer

3.1 Introduction

To develop useful applications, one needs extensible tools to query the backend. While the ISIS family does come with CDS-ISIS (for DOS and SCO Unix) and Win-ISIS (for the Microsoft Windows platform) Goa University has chosen to use the WWWISIS-4 (also referred to as *wxis*, to distinguish it from the WWWISIS-3) API layer for all its bibliographic applications including its OPAC, Serial Control and Library Management Suite. There are several reasons for this:

- WWWISIS uses public domain protocols like HTTP and CGI overlaid on the universal base TCP/IP layer for all its transactions. We find this use of open standards both free from dependence on proprietary network protocols as well as free from dependence on proprietary OS platforms. WWWISIS makes no demand on the client side platform as it will (through the web server) happily talk to any IETF compliant browsers (almost all browsers are IETF compliant).
- WWWISIS is blazingly fast. This is due to the superb indexing schemes that are used to generate the indexes and dictionaries.
- WWWISIS is completely extensible since the core API is based on XML[12].
- Core API control structures are much more richer than anything available so far in the ISIS family.
- The WWWISIS API allows for integration of OPAC like query features as well as database like insert, update and delete features allowing us to integrate management function functions seamlessly into our application.
- Since WWWISIS uses the HTTP and CGI protocols to service client side applications, it also allows us to seamlessly integrate modern alert services via e-mail, and home-pages of our Library.

3.2 The front-end

The fact that any IETF compliant browser (Netscape or Microsoft Internet Explorer) can serve as a powerful front-end to the WWWISIS server-side script is the single most attractive feature for choosing WWWISIS. This model requires no special configuration at the client side and makes no special demands on the client side operating system except the fact that it supports TCP/IP as a base protocol.

3.3 A (very) short introduction to WWWISIS-4 scripting

WWWISIS-4 relies on the CGI framework to execute scripts. The CGI framework is simple. A client browser requests a transaction from the remote HTTP Server[13] located

[12] WWWISIS-3 was not based on XML, WWWISIS-4 (or wxis) is

on the Library premises. This request could be a search query from a library user or a request to update or insert a record from a library manager. In either case, the webserver requests the WWWISIS backend to process the query through the CGI framework. The process could involve updating the remote database (in case the request came from an authenticated library manager) or output a set of fields or records (in case the query came from a OPAC client). In any case the output is passed back through the CGI gateway to the client browser (success or failure of update, or the list of matches to the OPAC query).

All queries by the WWWISIS backend are processed via scripts. Application development in this model therefore requires us to write simple scripts to process expected queries. What follows below are a set of WWWISIS-4 scripts.

3.3.1 Example I: Salut Mundi!

Here is the traditional first script that just wishes the World a big Hello!

```
<IsisScript name=HelloWorld>
    <section>
        <display>Hello World!</display>
     </section>
 </IsisScript>
```

As seen above, each WWWISIS script has a name, a section and an operative task. The tags are built in extensible XML so that it should not be difficult to extend the backend functionality by designing your own proprietary tags.

3.3.2 Example II: displaying fields of a record from a database

The second example displays a particular (sub)field (with tag 32, sub-field `c` of records 50 to 100, from a database called *unigoa*

```
<do>
    <parm name=db>unigoa</parm>
    <parm name=from>51</field>
    <parm name=to>100</field>
     <loop>
         <display>
         <pft>newline('<br>'),v32^c</pft>
         </display>
     </loop>
 </do>
```

Notice the control structures like <*do*> and <*loop*> which allow us to run through separate tasks and loop through the results.

[13] Goa University uses Apache -- the most popular Web server running on the Linux Operating System (Red Hat 6.2)

3.3.3 Example III: searching a database

The script to search a database for a particular expression (say *'database'*) anywhere in the record and display the serial numbers of the records as well as the title (tag 56) that match looks like:

```
<do task=search>
    <parm name=db>unigoa</parm>
    <parm name=expression>database</parm>
    <loop>
       <display><pft>mfn/,v56</pft></display>
    </loop>
</do>
```

3.3.4 Example IV: updating a record in a database

The following script enters a new record in the *unigoa* database. The updated fields are the authors name (in this case "Davidson A.W." with tag = 12) and the title (in this case "Relational Databases" with tag = 56).

```
<do task=update>
    <parm name=db>unigoa</parm>
    <parm name=mfn>New</parm>
     <update>
        <field action=append tag=12>Davidson A.W.</field>
        <field action=append tag=56>Relational Databases</field>
        <write>Unlock</write>
     </update>
</do>
```

3.3.5 All the other remaining stuff……

The above scripts give a flavour for the rich structures present in the WWWISIS scripting language but obviously do not exhaust all. We have not included the powerful formatting capabilities that can out put the results of the query in a well-defined structured pattern. This can be seen from the Goa University's Library Web Site. In addition to these formatting abilities are the sophisticated locking rules (transactions so that updates are coordinated with each other and queries touching upon those updates are not affected), task structures and search and query formatting that is best referred to in the manual. What makes WWWISIS particularly powerful is the fact that its underlying structure is XML-based making it possible for the user to add tasks and therefore functionality that is particular to his application.

4 Conclusion

The above discussion shows clearly the power of using Directory and Catalog based tools as opposed to tools designed around the RDBM paradigm as far as Bibliographic applications are concerned. It is clear that RDBMSs have much more power, but with that power also comes the application developer's responsibility to adhere to far

stricter standards of design that eventually make the application inflexible. Catalog and Directory based tools like WWWISIS combine the flexibility with exactly as much power as is required, and no more. But with this stripping comes a flexibility and the ability to extend the application in a short time. Since tools rely on the ISO 2709 format, data exchange is also supported so that other resource starved libraries can leverage automation in a time frame that is acceptable.