DEVELOPMENT OF AN ELECTRONIC RESOURCES WEB/DATABASE SYSTEM FOR THE UNIVERSITY OF NORTH CAROLINA AT CHAPEL HILL LIBRARIES

by
Janet A. McLaughlin

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in Information Science.

Chapel Hill, North Carolina

April, 2000

Approved by:

_____

Advisor

Table of Contents

Introduction

The Electronic Resources Collection at the University of North Carolina, Chapel Hill Libraries consists of electronic indexes and database (EIDs) and electronic journals (ejournals). No central repository exists through which information pertaining to these resources is maintained. Presentation of these resources to the public is accomplished through various tools including simple static html pages and a flat file database. This thesis project, begun in the summer of 1999 and continuing through the spring of 2000, was developed in response to the needs of the Public Access Services Committee Task Force on Access to the Libraries' Electronic Resources (the "Task Force"). The Task Force was charged with improving access to this collection, streamlining its administration, and introducing the ability to search the collection alphabetically and by subject category.

The objectives of this project are:

1. To design and develop a relational database for all electronic resources.
2. To design and implement a public access web interface that allows both alphabetic and subject searching.
3. To design a web-based administrative module for use in maintaining the database, adding new resources, and improving work flow for those responsible for maintenance of the electronic resources collection.

Project Description

Background

The University of North Carolina at Chapel Hill Libraries have an extensive collection of electronic resources, consisting of over 215 electronic indexes and databases and over 500 electronic journals. The EID pages are produced using the bibliographic database tool "ProCite"[1]. ProCite can store information about each resource and also has the ability to generate its stored data in html format. It is, however, a "flat" database: no relationships between fields exist and information collected for each record is unrelated to other records or fields.

While the ProCite solution is a significant improvement over the previous system of manually created html pages, it is limited in its ability to meet staff needs and is difficult to maintain and administer. The library holds a limited number of site licenses for ProCite, meaning a limited number of users can add, edit or delete records as required. A lack of staff expertise in using the ProCite software compounds the problem; the burden of maintaining this collection falls to the two staff members with adequate database training who have the skill required to manipulate the database when needed.

At the time this project was initiated, ejournals were manually maintained via separate alphabetical html pages. When ejournals are added or deleted, the ejournal is added or removed from the html page along with the corresponding html code. This is a time consuming process with a great capacity for error, and is especially inadequate given the increasing size of the ejournal collection.

---

[1] ProCite for Windows, Version 4.0.3; Research Information Systems

Ejournal records were to be added to the ProCite database in the Spring of 2000 as an interim step until the solution described in this paper is implemented. The drawbacks cited above for EIDs will continue to apply.

Statement of Need

To improve access to and maintenance of the electronic resources collection, the following is needed:

1. A relational database designed to minimize the administrative burden on library staff and to act as the backend through which information about electronic resources is stored and accessed for public display.
2. A public interface through which users may search the collection either alphabetically or by subject.
3. An administrative interface through which records may be easily added, edited and deleted by appropriate library personnel.

A relational database is the most efficient means to store, manipulate and process the resources in the collection. While requiring significant effort during the design phase, these efforts will ultimately minimize the effort needed to add and maintain the records in the collection. Relational databases prevent or limit the redundancy of data. In the ProCite database, redundancy of data and work effort is an issue. For example, many resources are provided by the "NCLive" service. Data about "NCLive" (name, URL, description) is repeated in every NCLive resource entry. In a relational database, information about providers will be maintained in one table. Basic information about the resource is maintained in another table. These two tables will be "related" by a single

value or "foreign key" through which the two are linked. Specific details about the database design are discussed below.

The second need is to improve upon the public interface through which users access the resources by adding subject searching. The ability to search the collection by subject is critical for users, especially given the size of the collection. Alphabetic searching is helpful for patrons who are familiar with the resources within a particular discipline, however those who are unfamiliar with the collection need a means to discover what resources are available to assist them.

Finally, an internal interface must be created so as to allow library staff to maintain the database via the web—and limit the need to manipulate the actual database. A web-based interface will eliminate the problem of site licensing and can be created so users with minimal web and database knowledge can edit collection records.

Prototype Development

Early in the project, it was decided the initial phase of implementation—specifically my involvement—would be deemed a prototype or pilot project. It was understood that a project of this nature impacts many different library departments, all of whom would require input into the design and structure of the database. The need to make decisions quickly to keep the project moving forward necessitated limiting such involvement. As a student who intended to graduate within a year, I did not wish to get mired in the committee decision-making process that is known to slow the progress of numerous large-scale library projects. It was therefore decided that as a pilot project, the prototype could be created with input from a minimal team of library staff ("the project team"), made up of people from within the Task Force.

Prior to the project being introduced to the public, the project team will solicit comments about the proposed system from other library personnel. Suggestions will be appropriately addressed through modification, where necessary. The ability of staff members to comment on a working prototype is expected to promote understanding of the functionality of the proposed system and facilitate effective feedback.

Project Scope

This project will involve designing a relational database to house electronic resources and creating the web pages through which the public will access the data stored within it. The final element of the project is the creation of an administrative web module through which library staff can maintain the database, without touching the database system. This element was added to allow ease of maintenance by library staff unfamiliar with database theory or technology.

Resources

I developed the prototype using the tools available at the School of Information and Library Science. I selected Microsoft Access as the initial database product for its relatively straightforward user interface and the ease with which tables, fields and data could be added or deleted. The project team acknowledged that such a "backend database" would be insufficient once the system was implemented due to speed and processing limitations, but for purposes of producing a working prototype, its flexibility would be beneficial. Later efforts outside the scope of this project would involve the migration from Access to the Library's MySQL database server.

I selected Cold Fusion as the middleware used to bridge the web and database servers. The public web pages and the internal administration site were created using the Cold Fusion markup language, or "cfml". An important advantage to Cold Fusion is its portability. The pages created for the prototype can be used with other ODBC compatible database systems. Assuming the MySQL system is based on the same database schema, only minimal changes should be required to access the data via the web interfaces.

Assumptions

The library will purchase the software needed to implement the prototype developed here. Software will include the Cold Fusion application server, Cold Fusion Studio to maintain the Cold Fusion files, a web server and a database server.

During prototype development, the project team will initially meet weekly to discuss the database design and confirm its required functionality. Once the database schema is finalized, minimal or no changes will be made to the design of this prototype. Any such changes would endanger the ability to complete the project prior to the April 2000 deadline.

The team will continue to meet biweekly or on an as-needed basis to discuss the design and functionality of the web interfaces, and to address developer questions as they arise.

Critical Success Factors

The Associate University Librarians (upper management in library administration) must support the development of the prototype and ultimate

implementation of the Electronic Resources Database System. They must be willing to commit the resources necessary to implement and maintain the new system. Staffing issues will undoubtedly arise given the expected redistribution of collection maintenance duties. The AULs must be prepared to support the Task Force during the politically sensitive introduction of this new system. Adequate training must be approved for all affected staff.

This project can only succeed if the Task Force and project team fully cooperate during the design phase, and support the ultimate goals described above. Members must be willing to make themselves available for questions concerning design issues, and must acknowledge the need to respond quickly when questions arise.

The developer must create an intuitive interface for both the administrative and public sites. The administrative interface must meet the needs of those responsible for data entry and should provide the searching and editing features needed to easily maintain resource information; minimal training of library staff should be required. The public interface must provide alphabetical and subject searching in a concise format requiring little or no instruction.

Deliverables

Upon completion of this project, deliverables will include the following: a relational database using Microsoft Access; documentation including an entity relationship diagram, database schema and data dictionary for use in migrating to the MySQL database option; and Cold Fusion files for both the public pages and the administrative interface.

Data is not included as a deliverable; electronic resources data will be regenerated from the ProCite database prior to migration to MySQL. The project team determined it was more efficient to generate a new data set than attempt to update the prototype data during the development process. Also, evaluation of the administrative interface during development and staff testing will be easier if users are not concerned with maintaining the integrity of the prototype data set.

Design Development

Database Design

I developed the database using Microsoft Access97. The database contains all pertinent information about electronic resources. Information about resource providers, vendors, subjects, access restrictions, and library staff selector are stored in separate tables with appropriate relationships built into the schema.

11 tables were needed to include the various different types of data, and insure normalization of data wherever possible. Because much of the data recorded for ejournals and EIDs is the same, one of the more difficult choices the team had to make was whether to create two or three tables with data for each type. One design suggested creation of one table ("RESOURCE") with both EID and ejournal data included; the table would hold the fields both types of resources had in common (name, URL, provider, vendor, etc.). Separate EID and EJOURNAL tables were to contain the additional data exclusive to that type of resource. After much discussion, the team decided that it would be a disadvantage to have to link 2 tables to retrieve the basic data about a particular resource—especially given that the ultimate design of the database necessitated linking many tables to display all relevant information about a single source. I created two

distinct tables: one for EIDs and one for ejournals from which most of the other tables are related through foreign keys. Each table contains many of the same field names, reflecting the common data.

The resulting entity relationship diagram and data dictionary are included in Appendices A & B. A brief description of each table, the data included, and the relationship to the two resource tables follows.

The ACCESS table contains information about the various types of access restrictions placed on each resource, as determined by the licensing agreement between the library and the resource vendor or provider. For example, certain resources can only be accessed from locations within the UNC-Chapel Hill campus: "On campus only (any location)". Other resources are unrestricted on campus, but require a UNC personal identification number for off campus access: "Unrestricted access on-campus; Off campus w/ 9 digit PID."

There are presently 26 different types of access restrictions. Information about each type includes a description, an instruction URL, a URL for an access icon (for future use), a staff note, and a field to indicate if the resource requires use of the library proxy server. It has a one-to-many relationship with the EID and EJOURNAL tables; the primary key (accessID) is a foreign key within the two resource tables.

The FORMAT table contains data about the various types of formats in which an ejournal may be produced. Examples are html, ASCII, PostScript and SGML. This table has a many-to-many relationship to the EJOURNAL table, and as such an additional table EJOURNAL_FORMAT is required to link the two together. The primary key from the

EJOURNAL table (EJID) and the primary key from the FORMAT table (formatID) make up its compound primary key, and are the only two fields within this table.

The PROVIDER table contains relevant data about the service that provides the resources. Currently, this table is only applicable to EIDs, although the database was designed so that similar information could be recorded for ejournals, if it becomes necessary to do so. The PROVIDER table has a one-to-many relationship to the EID table. Its primary key (providerID) is a foreign key within the EID table. The two providers associated with the library at present are NCLive and UNC Literature Exchange (UNCLE).

The VENDOR table contains information about the company that produces and/or sells the database or ejournal. Examples are OCLC FirstSearch, Academic Universe, and EBSCO for EIDs, and JSTOR, WILEY, and Project Muse for ejournals. The VENDOR table has a one-to-many relationship with both resource tables, hence its primary key (vendorID) is a foreign key within each.

The SELECTOR table contains information about the library staff member who chose the resource. Name and library department affiliation are recorded. The SELECTOR table has a one-to-many relationship with both resource tables, and again, its primary key (selectorID) is a foreign key within each.

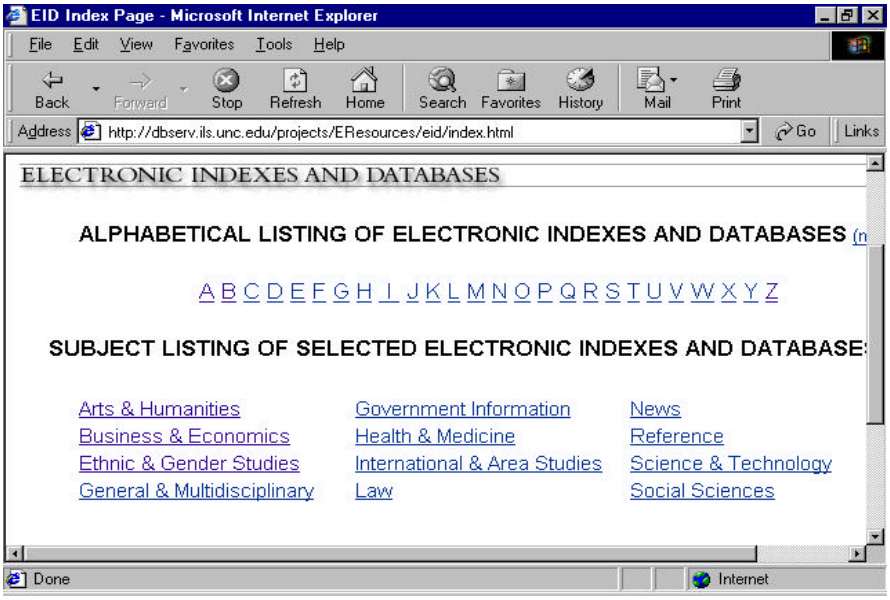
The SUBJECT table contains a list of both broad subject headings and narrow discipline subject headings. The expectation is that at present, EIDs will only be assigned broad categories, while ejournals will be assigned only narrow headings. The *isBroad* field indicates if the subject is broad or narrow (through a yes/no option). A description of each subject is also available to assist users understand the subject's scope.

The mechanism behind retrieving the subject data is the same for both resources. Since SUBJECT has a many-to-many relationship with the EID and EJOURNAL tables, an additional table was created to represent the relationship RESOURCE_SUBJECT. A compound primary key is used, whereby the primary key from the resource table(either an EIDID or an EJID) and the primary key from SUBJECT (subjectID) create a unique identifier. The field *core* was added to specify if this resource is a principal index within this subject heading, as specified by library personnel.

I added an additional table to the database to represent the relationship between a narrow subject heading and a broad subject heading. SUBJECT_HEADINGS takes as its primary key the combination of two distinct subjectIDs from the SUBJECT table. *subjectID* holds the ID number of a narrow subject from the SUBJECT table and *broadID* holds the ID number of one of 12 possible broad subjects. To date, this table is not being used within the pages created, but the ability to produce a report listing what "narrow" subjects are considered to be within the scope of a "broad" subject might be beneficial for future web page designs featuring different search parameters or for addressing departmental library subject requests. For example, certain departments would like to nest some subjects within other subject pages. While no plans exist to add this feature, this table provides the means to address such requests in the future.

Interface Designs

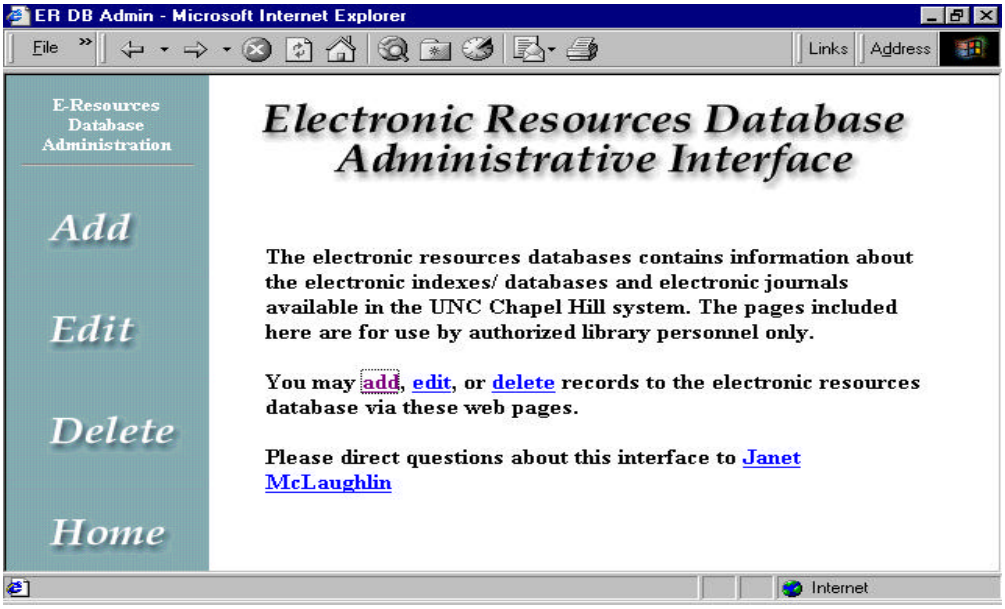
I designed a simple public interface based on the look and format of current library web pages. Library users have the option to view a list of resources either alphabetically or via subject. The index page for EIDs is reproduced below.



The index page for ejournals differs only in its subject heading selections (Appendix C). A drop down box is provided listing the 75 narrow subjects by which ejournals may be searched.

The administrative interface is built with frames to allow quick linking to its different functions. These functions include adding, editing and deleting records within each table in the electronic resources database. Web accessibility is a significant advantage to this interface design; with a web-based interface, library personnel in various departments and campus locations can maintain the records for which they are responsible. This will become increasingly important when the Academic Affairs Library begins to incorporate the Health Sciences Library collection into the database. Current plans are for the Health Sciences Library staff to maintain their own records, despite their physical location across campus—something that would not be possible under the current ProCite system.

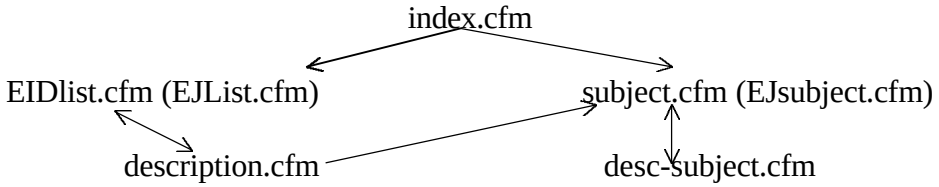
Few members of the library staff have database administration experience. Another advantage of the web interface is the ability of staff to manipulate the data without touching the actual database. This will become especially significant when the database is transferred to MySQL from Access as MySQL uses a command line interface which requires detailed database knowledge.



System Description

Public Interface

The file structure for both EIDs and ejournals is the same. An index page allows library patrons to choose to see an alphabetical list or a subject list of resources. From there a user can choose to view a more detailed description page (for EIDs), launch the URL, or select another letter to search. The basic three tier file structure is diagrammed below. Arrows indicate the presence of links within each page the user may elect to follow.



Ejournals do not have descriptions, so the ejournal structure does not include the third tier (*description.cfm* and *desc-subject.cfm*). The public interface consists of a total of five pages for EIDs and 3 for ejournals. This compares to the present system which consists of hundreds of html files; one for each letter of the alphabet (for ejournals and EIDs) and one for each EID resource description. I included a complete inventory of the Cold Fusion files created for this project as Appendix D.

Alphabetical Searching: The URL for each letter of the alphabet points to "*EIDlist.cfm*" or "*EJlist.cfm*". The letter is passed as a variable within the URL. For example, if a user wishes to view the list of databases beginning with the letter "Z", the URL passed when he clicks on the letter Z is

http://dbserv.ils.unc.edu/projects/EResources/eid/EIDlist.cfm?letter=Z

The *#letter#* variable is used within the WHERE clause of the SQL statement to retrieve only those resources beginning with the letter requested.[2] The query logic is exhibited in the following sample SQL statement.

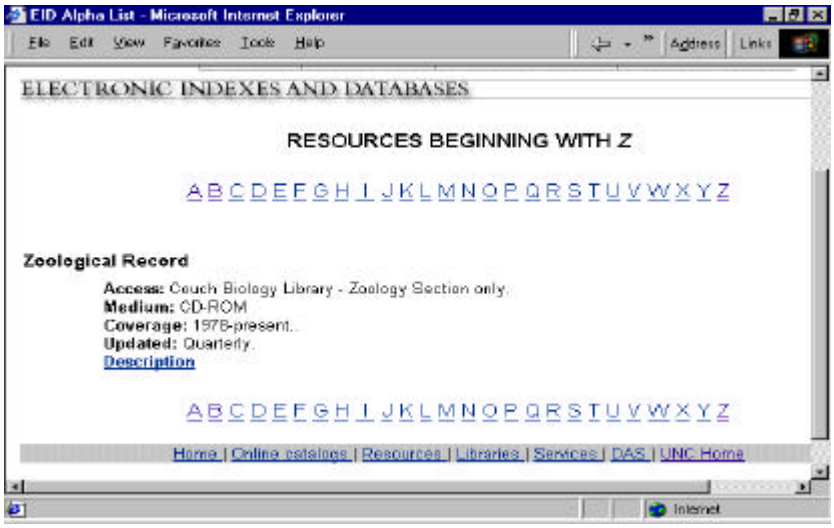
```
SELECT *
FROM EID
WHERE EIDName like '#letter#%'
```

The percentage sign acts as a wild card. The above query would retrieve all information from the EID table where the *EIDName* begins with the letter Z.

---

[2] Cold Fusion variables are enclosed within pound signs (#).

The actual query used is substantially more complicated, due to the complex relationships between the tables within the database. Four tables (EID, PROVIDER, VENDOR, ACCESS) are linked to display complete user-friendly information, such as provider name, vendor name, access information and appropriate URLs. *Views* were created to facilitate processing; further information about views is provided during the discussion of cross referencing. The actual query used can be found in Appendix E. The query result is displayed below.



Subject Searching: Retrieving EIDs by subject category employs the same method of passing variables within the URL. EIDs are presently categorized by the 12 broad subject categories created by the task force: Arts & Humanities, Business & Economics, Ethnic & Gender Studies, General & Multidisciplinary, Government Information, Health & Medicine, International & Area Studies, Law, News, Reference, Science & Technology, Social Sciences. Each broad subject is displayed as a hot link. The URL for each subject points to "*subject.cfm*" and the subject is passed as variable *subjectName* within the URL. For example, the URL for Law resources is

http://dbserv.ils.unc.edu/projects/EResources/eid/subject.cfm?subjectName=Law

EID resources are further delineated as either "principal indexes" or "other indexes" within certain subject headings, as determined by library staff. As mentioned in the database schema, the RESOURCE_SUBJECT table field "core" is used to indicate those resources that are considered to be "principal" within some subjects (not all subjects require this distinction).

```
SELECT *
FROM RESOURCE_SUBJECT, SUBJECT, vAllInfo
WHERE RESOURCE_SUBJECT.subjectID=SUBJECT.subjectID
AND vAllInfo.EIDID=RESOURCE_SUBJECT.resourceID
AND subjectName='#subjectName#'  AND RESOURCE_SUBJECT.core='yes'
ORDER BY EIDName
```

A similar query is created for "other" or "non-core" resources adjusting the "where" clause to "RESOURCE_SUBJECT.core='no'". The *subject.cfm* page employs a series of <cfif> statements to properly format the page based on the results of the "core" and non-core" queries. Sample code from "*subject.cfm*" is included in Appendix F.

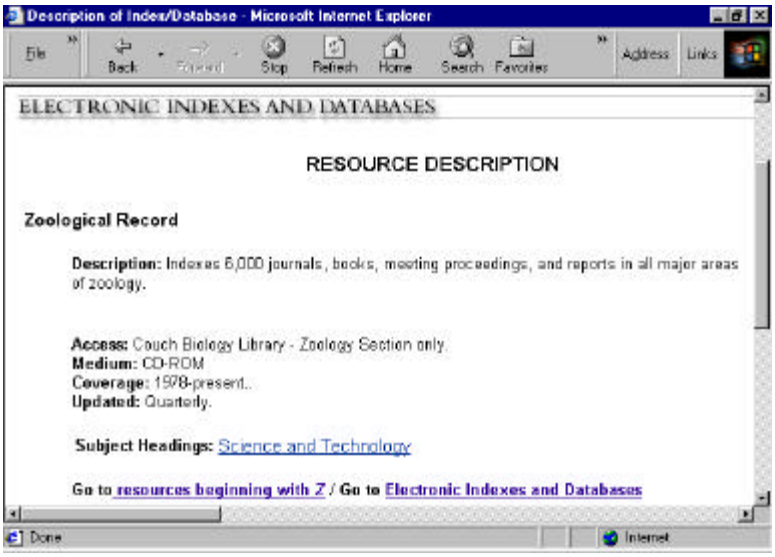
Ejournal Subjects: Ejournals are searched by 75 narrow subjects, adapted from UNC-Chapel Hill curriculum disciplines. Narrow subjects are necessary as the collection has over 500 ejournals and the broad categories do little to refine the search. A drop down box is provided listing all the narrow subjects in the database. The box is populated by an SQL query to the database selecting resources that are not "broad".

```
SELECT *
FROM SUBJECT
WHERE isBroad='no' OR isBroad IS NULL
ORDER BY subjectName
```

Using the html form "post" method, the subject name is passed to the "*EJsubject.cfm*" page when the user clicks the "submit" button. From here, the logic follows as with the EID "*subject.cfm*" page, returning all resources corresponding to that subject name.

```
SELECT *
FROM RESOURCE_SUBJECT, SUBJECT, ejAllInfo
WHERE RESOURCE_SUBJECT.subjectID=SUBJECT.subjectID
AND ejAllInfo.EJID=RESOURCE_SUBJECT.resourceID
AND subjectName='#subjectName#'
ORDER BY EJName
```

EIDs also have two detailed description pages available -- *description.cfm* and *desc-subject.cfm*. Both are formatted like the *EIDlist.cfm* page, but also include the description and a hot linked list of subject headings which lead the user directly to the appropriate *subject.cfm* page of related resources. The only difference between *description.cfm* and *desc-subject.cfm* is the referring page (either an alphabetical listing or a subject listing of resources) and the corresponding links at the bottom of the page that direct the user back to the previous (or referring) page. An example of the *description.cfm* page for the EID *Zoological Record* follows.



Display Variables: The *trialCheck* field, subscription end date, and display flags were created to control the display in the public interface pages described above.

It is common for the library to own a database on a trial basis. The "Trial Check" feature was added to allow easy monitoring of trial or temporary databases. The administrative interface module allows the staff member to indicate that the database is owned on a temporary basis by checking a box on the input form. This field acts as an indicator within the *EIDlist.cfm* file. When the field *trialCheck* is "yes", red text saying "This is a trial database" will be displayed next to the name of the resource.

```
<cfif #trialCheck# eq "yes">
<font color="red">This is a test database</font>
</cfif>
```

Presently, ejournals are not obtained on a trial basis, but I included the functionality for future use when necessary.

The *trialCheck* field works in tandem with the *subscriptEndDate* field. The EID is displayed as long as the subscription has not expired. The entire resource entry is within a conditional statement which insures only data with NULL or valid subscription dates are displayed (most resources have no such "end date" specified so NULL values must be accepted).

In response to the collection's volatility--especially ejournals--the team asked for a means to control whether or not a record was displayed to the public. Staff did not want to delete a resource from the collection database if it was likely to be renewed at a later date. I created a variable called *displayFlag* to meet this requirement. The EID and EJOURNAL tables have the *displayFlag* field. The default is "yes" but can be changed

to "no" through the administration pages. A defining WHERE clause was added to the pertinent SQL to insure output of only those resources where "*displayflag* = 'yes'" (Appendix E).

Complications/Challenges

While the database functionality was largely the same for both ejournals and EIDs, as evidenced in the schema described above, each had a peculiarity which necessitated customization of certain features within the SQL needed to display the public pages. EIDs required cross references, a means by which one EID refers to another. Ejournals had many different format types, which needed to be displayed within a single field. Solutions are discussed in detail below.

EIDs - Cross References

Electronic indexes and database titles are somewhat volatile. An index is often known by many names, and names may be changed when ownership changes. It was necessary to include a means to cross reference a previous title to the current titles under which all information is recorded so users could easily locate the resource even if they were aware of the former name only. I added a field to the EID tables called *crossRefID*. This field contains the ID number (from the EID table) of the resource to which the user should be referred. The vast majority of the records within the database have NULL values in this field, although the number of cross references will increase as resource names, product ownership, and vendor relationships change over time.

The display format in place on the library pages necessitated linking four tables (EID, PROVIDER, VENDOR, ACCESS) to retrieve all the needed data. The difficulty

arose when trying to make a copy of the EID table through which we could appropriately implement the crossRefID field so as to have an active link to the appropriate resource.

Upon consultation with various advisors and the project team, I decided to use the "create view" option. A "view" is a SELECT statement that becomes part of the database. It acts as a virtual table and can be used like a normal table for retrieving data. I created the view "vAllInfo" to retrieve all relevant data for each record, including appropriate cross references.

```
CREATE VIEW vAllInfo AS
SELECT EID.EIDID, EID.EID Name, ACCESS.accessDesc,
PROVIDER.providerName, VENDOR.vendorName, EID.EID_URL,
ACCESS.instructURL, EID.medium, EID.coverage, EID.updated,
EID.instructions, PROVIDER.providerURL, EID.description,
PROVIDER.providerIcon, EID.target, EID.crossRefID, EID.trialCheck,
EID.subscriptEndDate

FROM ((ACCESS RIGHT JOIN EID ON ACCESS.accessID = EID.accessID)
LEFT JOIN PROVIDER ON EID.providerID = PROVIDER.providerID) LEFT
JOIN VENDOR ON EID.vendorID = VENDOR.vendorID
WHERE EID.displayFlag='yes';
```

This view, which is the product of four tables, was then called upon in the SQL within the Cold Fusion file "*EIDlist.cfm*" in which it was linked with the main EID table.

```
<cfquery datasource="EResources" name="list" dbtype="ODBC">
SELECT vAllInfo.EIDID, vAllInfo.EIDName,vAllInfo.accessDesc,
vAllInfo.providerName, vAllInfo.vendorName, vAllInfo.EID_URL,
vAllInfo.instructURL, vAllInfo.medium, vAllInfo.coverage, vAllInfo.updated,
vAllInfo.instructions, vAllInfo.providerURL, vAllInfo.providerIcon,
vAllInfo.description, vAllInfo.target, vAllInfo.trialCheck,
vAllInfo.subscriptEndDate, EID.EIDName as RefName, EID.EID_URL as
RefURL, EID.target as RefTarget
FROM EID RIGHT JOIN vAllInfo ON EID.EIDID=vAllInfo.crossRefID
<cfif #letter# neq "ALL">
WHERE vAllInfo.EIDName like '#letter#%'</cfif>
ORDER BY vAllInfo.EIDName, vAllInfo.vendorName, vAllInfo.providerName;
</cfquery>
```

The cross reference information (reference name, URL and target) is retrieved from the EID table; the tables are linked via a right join where the EIDID in the EID table equals the crossRefID in the vAllinfo virtual table. I used a right join to insure all the information is returned from the vAllInfo table - even if the crossRefID field is NULL (which is the case for most resources).

A conditional statement based on *RefName* determines what is displayed. If the value for R*efName* is not NULL, the record will include an appropriate "see" reference.

```
<cfif #RefName# neq "">
<cfset newletter = Left(#RefName#, 1)>
<B>See </B><A
HREF="EIDlist.cfm?letter=#newletter####RefTarget#">#RefName#</A></cfif>
```

The variable *letter* is the means by which we retrieve a resource list of only the appropriate alphabetical letter (EIDS are presented one page per letter). This code reassigns the variable #letter# to the first letter of the resource name to which we are referring. The *RefTarget* assures that the resource being requested is displayed at the top of the page.

Ejournals - Formats

A "format" describes the medium through which the ejournal is provided. The most common types are html, PostScript, and PDF (portable document format). Many ejournals are provided in more than one format, creating a many to many relationship between the EJOURNAL and FORMAT tables. This relationship is specified within the EJOURNAL_FORMAT table. Initially, I thought simply linking the EJOURNAL_FORMAT table to the main SQL statement on the *EJList.cfm* page would suffice, however it simply created multiple entries for each ejournal with multiple formats. I realized I needed to find a way to concatenate the ejournals format data into

one field. Loops proved to be the best solution, specifically the <cfloop> function in Cold Fusion.

One of the drawbacks of Cold Fusion is the inability to nest query outputs. Only certain Cold Fusion specialty functions will allow you to output the result of multiple queries at the same time. CFLOOP is one such function. By splitting my large SQL query into three separate queries, I was able to use CFLoops to nest them and produce the desired output.

I began by getting a list of all the ejournals that should be returned (based on the letter or subject selected).

```
<cfquery datasource="EResources" name="list" dbtype="ODBC">
SELECT EJOURNAL.EJID, EJOURNAL.EJName
FROM EJOURNAL
WHERE EJOURNAL.EJName like '#letter#%'
ORDER BY EJOURNAL.EJName
</cfquery>
```

I then set up my cfloop function and set the EJID number equal to a new variable "idNum" which would act as the loop control.

```
<cfloop query="list">
<cfset idNum = #EJID#>
```

From here I created another query (within the loop structure) calling the main SQL statement that retrieves the majority of data.

```
<cfquery datasource="EResources" name="fullList" dbtype="ODBC">
SELECT ejAllInfo.EJID, ejAllInfo.EJName,ejAllInfo.accessDesc,
ejAllInfo.providerName, ejAllInfo.vendorName, ejAllInfo.EJ_URL,
ejAllInfo.instructURL, ejAllInfo.medium, ejAllInfo.holdings, ejAllInfo.ISSN,
ejAllInfo.language, ejAllInfo.frequency, ejAllInfo.userInstruct, ejAllInfo.providerURL,
ejAllInfo.providerIcon, ejAllInfo.target, ejAllInfo.trialCheck
FROM ejAllInfo
WHERE ejAllInfo.EJID=#idNum#
ORDER BY ejAllInfo.EJName, ejAllInfo.vendorName, ejAllInfo.providerName ;
</cfquery>
```

Note the WHERE clause refers to the new variable created. This query will execute for each resource returned from the initial "list" query.

I needed to get the format data next. I started by retrieving the format types for each resource:

```
<cfquery datasource="EResources" name="format" dbtype="ODBC">
SELECT *
FROM EJOURNAL_FORMAT, FORMAT
where EJOURNAL_FORMAT.EJID=#idNum# and
(EJOURNAL_FORMAT.formatID=FORMAT.formatID)
</cfquery>
```

Note again the WHERE clause refers to the new variable created, and the query will execute for each record in the initial "list" query. These two queries will execute consecutively within the loop structure.

I then created a variable called "formats" into which I concatenate the results from the "format" query above, using the Cold Fusion function "ListAppend".

```
<cfset formats="">
<cfoutput query="format2">
<cfset formats=ListAppend(formats, #formatName#)>
</cfoutput>
```

From here it was a simple matter of outputting the results of the "fullList" query with the added variable "formats" included to provide the format type information. Sample code from this process is included in Appendix G.

To reiterate the loop functionality, each of the two queries within the loop runs independently for each id number returned from the initial "list" query. For example, as there are two ejournals beginning with the letter "Z", the "fullList" query and the "format" query shown above run two times; the loop executes two times as there are two records returned by our initial list query.

Sample output produced by the Cold Fusion server is included as Appendix H.

Administrative Module

The administrative module was created to allow library staff without database expertise to maintain the collection. Through this interface, the user can add new records, modify existing records, or delete records from the database.

Adding Records: Authorized users may add records to all tables or may assign subject headings to existing records. The interface guides the user through the data entry process through a series of forms which pass data on to "action" pages—which in turn pass data back to form pages in a somewhat circular fashion.

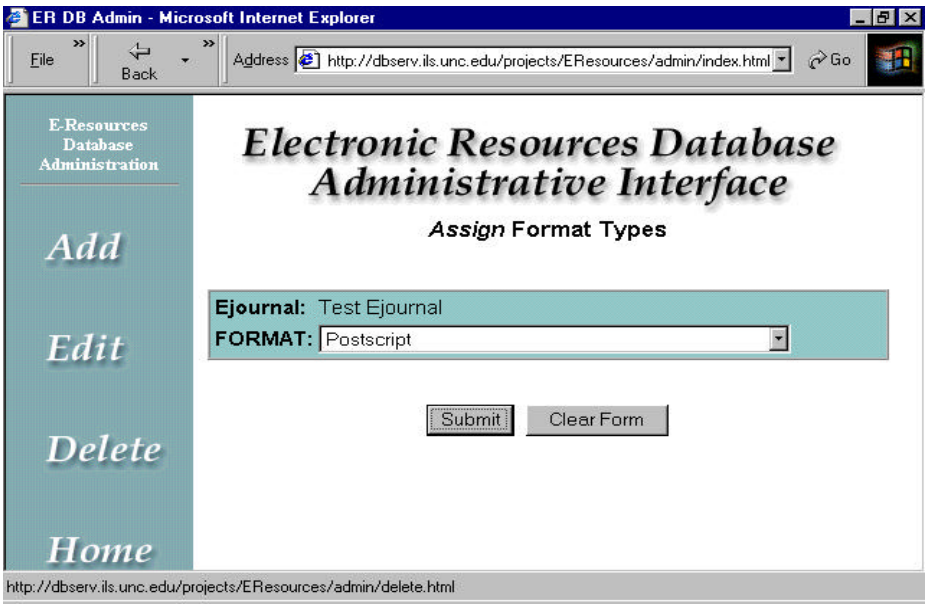
Two Cold Fusion pages are needed to add records to tables: a form page and an "action" page. The form page contains various types of form fields (text, textarea, radio buttons and drop down boxes) for entering data. Drop down or "select" boxes are populated by fixed choices (such as "Web" or "CD_ROM" for the EID field *medium*) or

by SQL queries to the database (for fields such as access restriction, provider, vendor, and selector ID keys). For example, the following query populates the access restriction drop down menu:

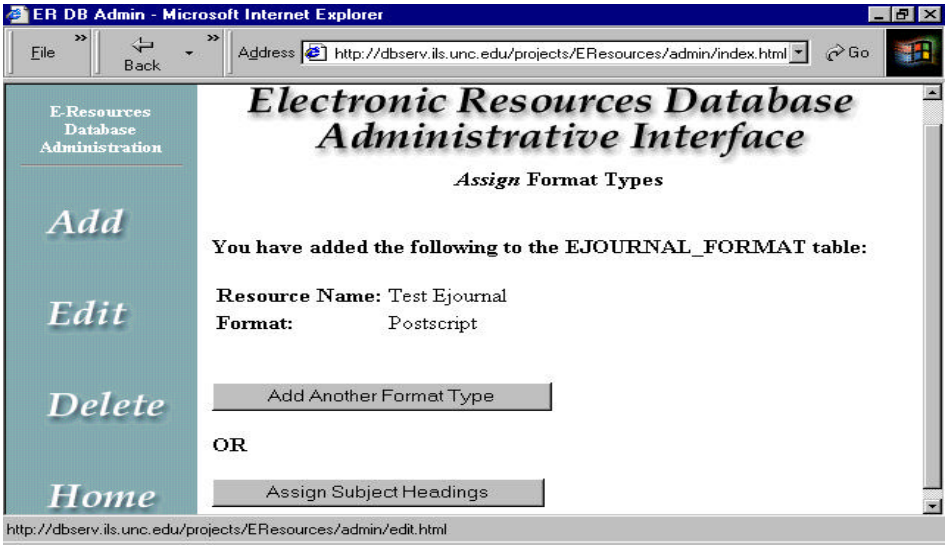
```
<CFQUERY name="accessquery" datasource="EResources" dbtype="ODBC">
SELECT accessID, briefDesc
FROM ACCESS
ORDER By briefDesc
</CFQUERY>
```

The name of the "action" page is specified within the <form> tag. This page inserts the passed variables into the database and displays confirmation of the input via html output. If a new EID or Ejournal is added, the confirmation page will include two buttons directing the user to enter subject headings or format information (ejournals only). In this instance, the confirmation page becomes a form page passing a variable (either EIDID or EJID depending on the resource) to yet another form page which will then permit the user to select either the subject or format.

This action page (below) will insert the *EJID* for the journal "Test Journal" and the *formatID* for the format type "Postscript" into the EJOURNAL_FORMAT table.



The confirmation page, shown below, gives the user the option to add another format or add subject headings.



Editing Files: To edit existing records, the interface employs a series of forms in a three file progression. The first file is a simple form with a drop down menu through which the user chooses the record to be edited. That record's ID key is passed to the second file which queries the database (through a simple SELECT query) and populates a form with the returned data. Authorized users then make changes to individual fields which are passed to the third file—an action page which processes the SQL "UPDATE" and displays confirmation in html. For consistency, the Cold Fusion editing files were numbered to reflect this three step process. For example, three files used for editing EID records are *updateEID1.cfm, updateEID2.cfm* and *updateEID3.cfm* (see the administration section of the file inventory Appendix D.)

Deleting Files: The delete function files mirror the editing files in structure. The first file allows authorized users to select which record to delete. The second file displays

the data about that record and asks for confirmation that this record should be deleted. When the user presses the "Confirm Delete" button, the third file is called which deletes the record from the database and displays confirmation to the user.

Referential integrity: Because the ultimate home for this database will be MySQL, and MySQL does not support referential integrity updates or deletes, it was necessary to build integrity constraints into the interface files. To enforce integrity when adding or editing data, I used input control methods such as select boxes and mandatory fields. As discussed above, the select boxes require the user to select from valid data options. By deeming certain fields as "required", we ensure valid relationships exist between tables when necessary. Required fields are enforced via Cold Fusion's JavaScript validation functions (see error checking below).

When deleting data, I entered multiple DELETE queries to the third delete file where appropriate. For example, when an ejournal is selected, the record is deleted from the EJOURNAL, RESOURCE_FORMAT and RESOURCE_SUBJECT tables. The third file executes the following three queries to delete a single ejournal:

```
DELETE    *
FROM      RESOURCE_SUBJECT
WHERE     ResourceID=#EJID#

DELETE    *
FROM      EJOURNAL_FORMAT
WHERE     EJID=#EJID#

DELETE    *
FROM      EJOURNAL
WHERE     EJID=#EJID#
```

#*EJID*# is the variable passed from the second delete file (*deleteEJ2.cfm*).

Error checking: I used both server-side and client-side error checking throughout the administrative interface. Client-side error checking verifies that required fields are completed and that values entered are of valid data types. For example, when a user adds a new ejournal record to the database, an *EJName* must be provided—the field is "required". If the user submits the form without a name, a message will appear informing her that the field is required prior to form submission. Date field validation is another example. In my interface, dates must be in the form mm/dd/yyyy. If a user attempts to enter "May 5, 2002" as a subscription end date, a message will appear asking for the appropriate date format. The Cold Fusion <cfform> tag is very useful here as its input field options will generate the JavaScript code needed when certain attributes are included. For example, the form field which creates the validation code for an ejournal name is

```
<CFINPUT type="text" name="EJName" size="25" required="yes"
message="Please enter the name of the journal.">
```

The "required" and "message" attributes trigger the JavaScript function.

I used server-side validation to prevent errors in data entry involving primary keys and to check for input errors on fields which should be unique. Primary keys are generated by a JavaScript code each time a person begins to enter a record . The code queries the database, returns the highest value, and then adds one. An error can only occur if the user uses the browser's "Back" button and fails to reload the entry form page after submitting a new record. The following query checks the database to see if the number already exists.

```
<CFquery name="EJIDcheck" datasource="EResources">
SELECT *
FROM EJOURNAL
```

```
WHERE EJID=#EJID#
</cfquery>
```

If the query returns a record (<cfif EJIDcheck.RecordCount gt 0>) an error message is displayed and the page is aborted using the <CFABORT> function. I included the code for this process as Appendix J.

The EJOURNAL and EID table have two fields which should contain unique values: *target* and *DRADBCN*. As described above, the code queries the database and, if the value exists, terminates processing and returns a message to the user.

```
<cfquery name="controlnumbercheck" datasource="EResources">
SELECT *
FROM EJOURNAL
WHERE DRADBCN='#DRADBCN#'
</cfquery>

<cfif controlnumbercheck.RecordCount gt 0>
<b>Error:</b> the DRADBCN number you entered already exists, please try
again.
<CFABORT>
</cfif>
```

Internal Review and Testing

The project team presented the prototype to the Associate University Librarians in March, 2000. The AULs agreed to support its implementation and approved its introduction to library staff as soon as possible. The project team plans to begin demonstrating the new system in mid April to the staff most affected by its introduction—specifically those staff members in the Technical Services and Serials departments who are to assume a large part of the responsibility for collection maintenance after implementation. Staff will be asked to test and comment on both the public and administrative portions of the prototype. As the alphabetical searching within

the public interface section mirrors the current library web page format, comments are expected to focus on the introduction of subject searching and the selection of subject categories. Functionality of the public system is not expected to generate much discussion.

An important aspect of this initial testing phase is the staff's reaction to the administrative interface. During the development process, comments about interface design and functionality were solicited from members of the project team. During the staff testing phase, it will be important to determine if user needs were adequately addressed. It is hoped staff members will address issues of data entry process flow, ease of use of the web pages, and completeness and value of the data collected.

The final phase of testing will be to introduce the prototype to the entire library staff. Only the public interface will be presented, as the administrative interface will be accessible by only a few authorized users. Comments about the intuitiveness of the design and value of the alphabetical and subject searching structure will be considered; reference department staff will be asked to specifically assess the patron's ability to successfully navigate the new interface. Modifications based on comments received will be addressed as needed prior to inplementation of the final MySQL product.

Implementation and Maintenance

This prototype will not be implemented in its current form, but will be used as the basis from which the Library MySQL database system is built. Implementation of this system began in March, 2000. The initial step involves the migration of the database from Access to MySQL. To date, the database schema has been imported to MySQL and the data types of all fields have been reviewed and adjusted as needed.

The next step will be to import the data from the ProCite database. As discussed above, the data within the prototype is out of date. The project team decided early in the development process that it would be difficult and time consuming to attempt to maintain information about the collection in both the ProCite database and the prototype Access database. Therefore to insure a complete and accurate data set, data files will be generated from the ProCite database and imported into the MySQL database.

Testing the interaction between the Cold Fusion files and the MySQL database is the last step prior to implementation. It is expected that some of the Cold Fusion code will need to be adjusted to work with the new database system. That MySQL does not presently support *views* is one known problem. As views are used to support cross referencing, an alternate solution is needed. At the time of this writing, I have identified Cold Fusion's <CFLoop> function as a potential alternative to views. I am also concerned about case sensitivity. The prototype uses Microsoft Access and Cold Fusion on an NT server; NT is case insensitive, meaning variable, table and field names may be written in Cold Fusion without concern for case. MySQL on a Unix machine is case sensitive. For example, error messages will result if the SQL in a Cold Fusion file refers to table "Ejournal" when the actual name is "EJOURNAL". I attempted to be sensitive to this issue during development, however it is certainly possible that such errors may be found, as no error messages would have been generated during development under an NT system. The project team plans to test the functionality of all the Cold Fusion files as soon as the data is imported into MySQL and the Cold Fusion files are transferred to the library's web server.

The Assistant Head of Library Systems has accepted responsibility for the maintenance of the Cold Fusion files and the MySQL database once the system is implemented. Maintenance efforts should be limited in scope; the collection records will largely be updated through use of the administrative interface, thus requiring minimal interaction with the database. However, periodic evaluation of query processing speeds and verification that the system continues to meet user needs will be required as the collection size increases.

The MySQL database is of adequate size and functionality to handle the anticipated growth of the collection, but it is expected that a new online catalog system capable of providing similar access to electronic resources will remove the need for this separate system within the next 10 years. Short term enhancements under consideration include introduction of keyword searching and interaction with the Health Sciences Library's UNCLE database. Such plans are in early development and are not expected to be addressed until the new system is operational.

Final Comments

Lessons learned during this process include the need to insure the project team had the required expertise to complete the project, to insure full commitment of those selected, to understand the political environment under which the project is directed, and to schedule regular meetings between the project team and other parties (including the Task Force) with a strong interest in the ultimate success of the project and who might be able to provide needed insight. I ran into several problems during development of this project which I may have avoided had I understood certain competing obligations of

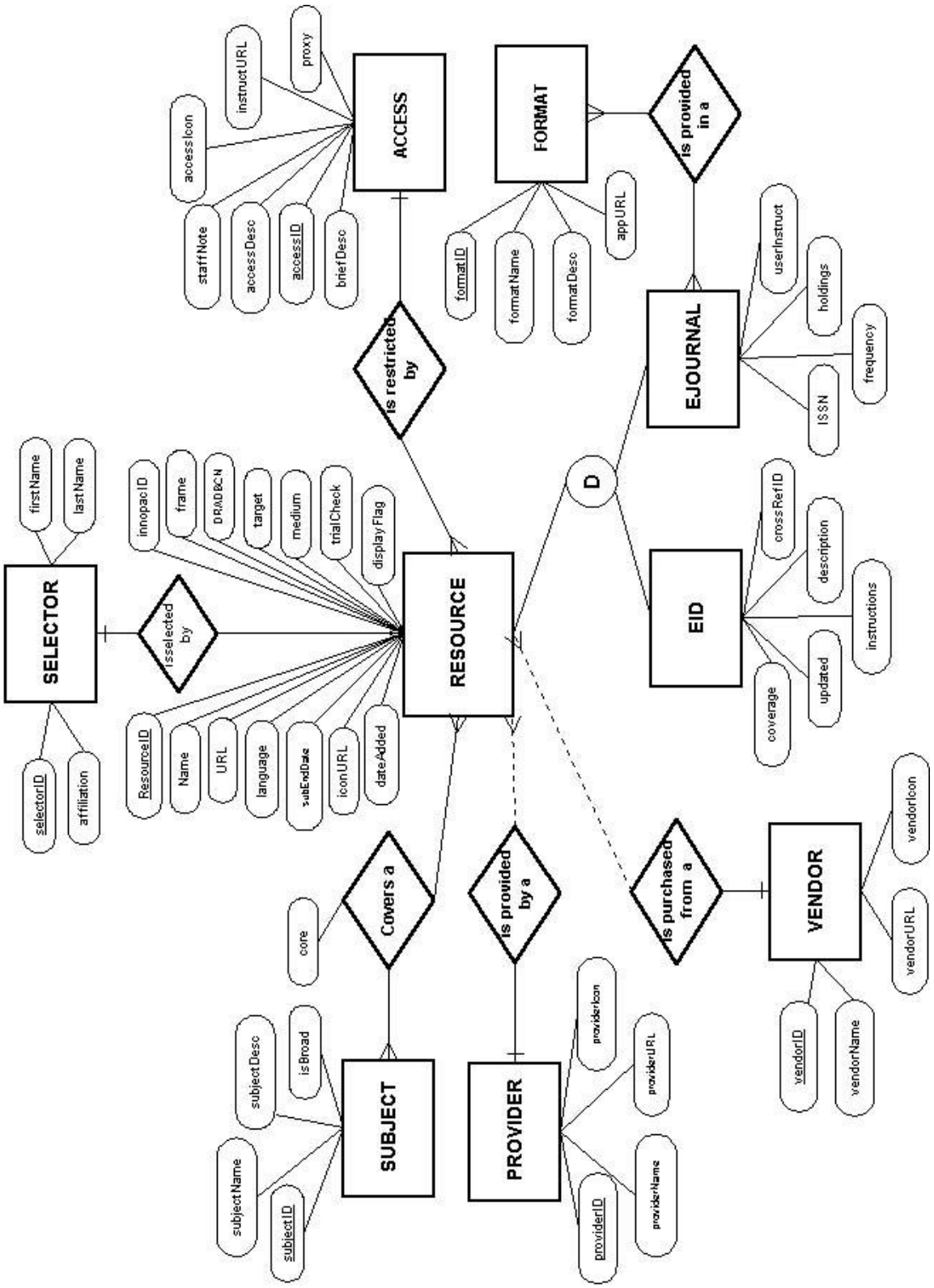
those involved and insisted on greater participation of all concerned before the project began.

A persistent problem was an inability to finalize the database design. Despite my requests that all parties review the database schema and verify that all issues were addressed, project team members requested database table and field modifications months into the interface development phase. Similarly, certain parties who were not on the project team provided crucial information about the database needs *after* we had finished the design work. For example, one month prior to completion of the prototype, the "displayFlag" issue was mentioned by a Task Force member. No one had mentioned the need to control whether or not a record was displayed until this point. Luckily, it was not a significant change to add the *displayFlag* field and add an additional "where" conditional to certain SQL statements to meet this need, but the fact that this important function was not discovered until the virtual completion of the prototype indicates the absence of consistent participation and review by critical decision makers.

Complicating the process was the fact that the Task Force had many varied and sometimes competing responsibilities. Task Force members are generally overextended and could not allocate sufficient time to review the prototype design. During the development of this project, I met with the entire Task Force only once. I received only minimal feedback about the functionality of the database and the design of the interfaces. The project team had no choice but to move forward without significant Task Force input. To date, the Task Force has been satisfied with the prototype design and functionality, but the project may have been completed sooner and had fewer last minute alterations had feedback been provided throughout the process.

Despite some frustrations, this proved to be a very rewarding project. I was allowed to take a project from conception to prototype completion, and am continuing to work on its ultimate implementation. I employed database design skills and relational database theory to create an electronic resources database, practiced interface design skills to create public and administrative web pages, and used web-database programming skills to create the required functionality between the interfaces and the collection database. When implementation is complete, this project will enhance library services provided to students and staff while significantly reducing the effort needed to maintain the electronic resources collection.

Appendix A Entity Relationship Diagram of EResources Database



Appendix B Data Dictionary

Table: EID			
Field name	**Data Type**	**Size**	**Description**
EIDID	Number	-	Primary key; foreign key to RESOURCE_SUBJECT table
EIDName	Text	150	Name of the resource
EID_URL	Text	250	URL of the resource
language	Text	50	Language in which the index/database is published
subscriptEndDate	Text	10	Date when the subscription ends (if any, may be null)
iconURL	Text	250	URL of the resource icon (if any)
dateAdded	Text	10	Date the resource was added to the database
innopacID	Text	50	ID assigned for the innopac system
frame	Text	3	Yes/No; does this resource require frame setting?
crossRefID	Number	-	Reference to the EIDID of another resource within this table (EID). Used to refer patron to the current resource.
DRADBCN	Text	50	DRA Database Control Number - assigned by cataloging department.
coverage	Text	200	Dates covered by this resource.
updated	Memo	-	Description of how often the information within the resource is updated.
instructions	Text	250	Description of user instructions relating specifically to this resource.
description	Memo	-	Description of the resource.
medium	Text	20	Medium of the resource; ex. web, cd-rom, email.
accessID	Number	-	Foreign key; primary key from ACCESS table; provides link to type of access restriction.
providerID	Number	-	Foreign Key; primary key from PROVIDER table; provides link to provider data
selectorID	Number	-	Foreign key; primary key from SELECTOR table; provides link to selector information
vendorID	Number	-	Foreign key; primary key from VENDOR table; provides link to vendor information
target	Text	50	Manually created shorthand for resource, used for html anchor tag notation.
trialCheck	Text	3	Yes/No field; is this resource provided on a trial basis?
displayFlag	Text	3	Yes/No field; should this resource be displayed? Included to provide means to maintain resource data without deleting it from the database.

Table: EJOURNAL			
Field name	**Data Type**	**Size**	**Description**
EJID	Number	-	Primary key; foreign key to RESOURCE_SUBJECT and EJOURNAL_FORMAT table
EJName	Text	150	Name of the resource
EJ_URL	Text	250	URL of the resource
language	Text	50	Language in which the ejournal is published
subscriptEndDate	Text	10	Date when the subscription ends (if any, may be null)
iconURL	Text	250	URL of the resource icon (if any)
dateAdded	Text	10	Date the resource was added to the database
innopacID	Text	50	ID assigned for the innopac system
frame	Text	3	Yes/No; does this resource require a frame setting
DRADBCN	Text	50	DRA Database Control Number - assigned by cataloging department.
ISSN	Text	50	ISSN number of the ejournal
frequency	Text	200	How often the ejournal is published
holdings	Memo	-	Which volumes or dates of coverage are included in the collection
userInstruct	Memo	-	Description of user instructions relating specifically to this resource.
medium	Text	20	Medium of the resource; ex. web, cd-rom, email.
accessID	Number	-	Foreign key; primary key from ACCESS table; provides link to type of access restriction.
providerID	Number	-	Foreign Key; primary key from PROVIDER table; provides link to provider data (CURRENTLY NONE)
selectorID	Number	-	Foreign key; primary key from SELECTOR table; provides link to selector information
vendorID	Number	-	Foreign key; primary key from VENDOR table; provides link to vendor information
target	Text	50	Manually created shorthand for resource, used for html anchor tag notation. (DRADBCN to be used instead)
trialCheck	Text	3	Yes/No field; is this resource provided on a trial basis?
displayFlag	Text	3	Yes/No field; should this resource be displayed? Included to provide means to maintain resource data without deleting it from the database.

Table: ACCESS			
Field name	**Data Type**	**Size**	**Description**
accessID	Number	-	Primary key; foreign key to EID and EJOURNAL tables.
accessDesc	Memo	-	Long description of the access restriction; displayed to the user through public html pages.
briefDesc	Text	100	Short description of the restriction; for use only in internal administration pages.
staffNote	Memo	-	Internal note for staff with information about this restriction (not currently used).
accessIcon	Text	100	URL of an icon to be used as shorthand for the type of restriction; not currently used.
instructURL	Text	50	URL of the instructions for user as to how access is provided (internal URL).
proxy	Text	3	Yes/No; to be used to identify proxy server resources (not currently used).

Table: EJOURNAL_FORMAT			
Field name	**Data Type**	**Size**	**Description**
EJID	Number	-	Primary key; ejournal ID (from EJOURNAL table)
formatID	Number	-	Primary key; format ID (from FORMAT table)

Table: FORMAT			
Field name	**Data Type**	**Size**	**Description**
formatID	Number	-	Primary key; foreign key to EJOURNAL_FORMAT table; number assigned to each type of format
formatName	Text	50	Name of the format type (ex. postscript, html, ASCII)
formatDesc	Text	100	Description of the format type
appURL	Text	200	URL where user can download needed application

Table: PROVIDER			
Field name	**Data Type**	**Size**	**Description**
providerID	Number	-	Primary key; foreign key to EID and EJOURNAL tables; number assigned to each provider
providerName	Text	50	Name of the provider (ex. NCLive, UNCLE)
providerURL	Text	100	Provider's web site
providerIcon	Text	50	URL of provider icon (for display)

Table: RESOURCE_SUBJECT			
Field name	**Data Type**	**Size**	**Description**
resourceID	Number	-	Primary key; either EIDID from the EID table, or EJID from the ejournal table
subjectID	Number	-	Primary key; from the SUBJECT table
core	Text	3	Yes/No field; is this resource to be considered a "principle" resource

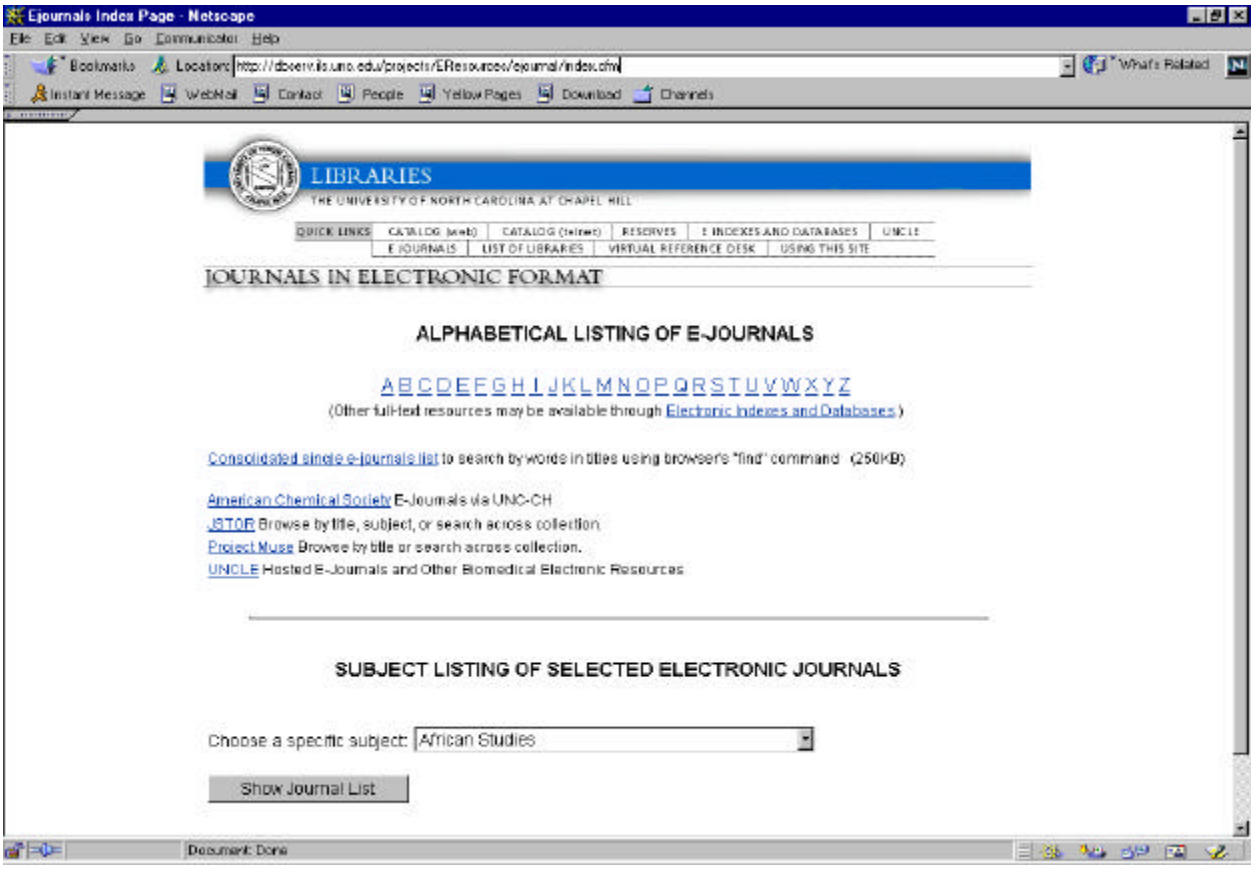
Table: SELECTOR			
Field name	**Data Type**	**Size**	**Description**
selectorID	Number	-	Primary key; foreign key to EID and EJOURNAL tables.
firstName	Text	50	First name of the person who selected the resource.
lastName	Text	50	Last name of the person who selected the resource.
affiliation	Text	50	Library or department with which the selector is affiliated.

Table: SUBJECT			
Field name	**Data Type**	**Size**	**Description**
subjectID	Number	-	Primary key; foreign key to RESOURCE_SUBJECT table.
subjectName	Text	50	Name of the subject
subDesc	Memo	-	Description of the subject matter included under this heading.
isBroad	Text	3	Yes/No field; is this subject a broad (not narrow) subject heading?

Table: SUBJECT_HEADINGS			
Field name	**Data Type**	**Size**	**Description**
subjectID	Number	-	Primary key; from SUBJECT table.
broadID	Number	-	Primary key; subject ID from SUBJECT table; links all narrow subjects with appropriate broad subjects.

Table: VENDOR			
Field name	**Data Type**	**Size**	**Description**
vendorID	Number	-	Primary key; foreign key to EID and EJOURNAL tables; number assigned to each vendor
vendorName	Text	100	Name of the vendor (ex. OCLC FirstSearch, OVID, JSTOR).
vendorURL	Text	100	Vendor's web site
vendorIcon	Text	50	URL of vendor icon (for display)

Appendix C Ejournal Home Page



Appendix D File Inventory

EID

- desc_subject.cfm
- description.cfm
- EIDlist.cfm
- index.html
- subject.cfm

Ejournal

- EJlist.cfm
- EJSubject.cfm
- index.cfm

Admin

- add.html
- addEID.cfm
- addEIDform.cfm
- addEIDsub.cfm
- addEJ.cfm
- addEJform.cfm
- addEJformat.cfm
- addEJsub.cfm
- addprovider.cfm
- addproviderform.cfm
- addselector.cfm
- addselectorform.cfm
- addsubject.cfm
- addsubjectform.cfm
- addvendor.cfm
- addvendorform.cfm
- adminhome.html
- delete.html
- deleteEID1.cfm
- deleteEID2.cfm
- deleteEID3.cfm
- deleteEIDsubject1.cfm
- deleteEIDsubject2.cfm
- deleteEIDsubjectform.cfm
- deleteEJ1.cfm
- deleteEJ2.cfm
- deleteEJ3.cfm
- deleteEJformat1.cfm
- deleteEIDformat2.cfm
- deleteEIDformatform.cfm
- deleteEJsubject1.cfm
- deleteEJsubject2.cfm
- deleteEJsubjectform.cfm
- deleteProvider1.cfm
- deleteProvider2.cfm
- deleteProvider3.cfm
- deleteSelector1.cfm
- deleteSelector2.cfm
- deleteSelector3.cfm
- deleteSubject1.cfm
- deleteSubject2.cfm
- deleteSubject3.cfm
- deleteVendor1.cfm
- deleteVendor2.cfm
- deleteVendor3.cfm
- edit.html
- EIDbroadform.cfm
- EIDnarrowform.cfm
- EIDsubsform.cfm
- EJbroadform.cfm
- EJsubsform.cfm
- EJnarrowform.cfm
- EJsubsform.cfm
- index.html
- toc.html
- updateEID1.cfm
- updateEID2.cfm
- updateEID3.cfm
- updateEJ1.cfm
- updateEJ2.cfm
- updateEJ3.cfm
- updateProvider1.cfm
- updateProvider2.cfm
- updateProvider3.cfm
- updateSelector1.cfm
- updateSelector2.cfm
- updateSelector3.cfm
- updateSubject1.cfm
- updateSubject2.cfm
- updateSubject3.cfm
- updateVendor1.cfm
- updateVendor2.cfm
- updateVendor3.cfm

Appendix E SQL Statements

1) *Views* created within MS Access:

a) EID View *vAllInfo:*

SELECT EID.EIDID, EID.EIDName, ACCESS.accessDesc, PROVIDER.providerName, VENDOR.vendorName, EID.EID_URL, ACCESS.instructURL, EID.medium, EID.coverage, EID.updated, EID.instructions, PROVIDER.providerURL, EID.description, PROVIDER.providerIcon, EID.target, EID.crossRefID, EID.trialCheck, EID.subscriptEndDate

FROM ((ACCESS RIGHT JOIN EID ON ACCESS.accessID = EID.accessID) LEFT JOIN PROVIDER ON EID.providerID = PROVIDER.providerID) LEFT JOIN VENDOR ON EID.vendorID = VENDOR.vendorID

WHERE EID.displayFlag='yes';

b) Ejournal View *ajAllInfo:*

SELECT EJOURNAL.EJID, EJOURNAL.EJName, ACCESS.accessDesc,PROVIDER.providerName, VENDOR.vendorName, EJOURNAL.EJ_URL, ACCESS.instructURL, EJOURNAL.holdings, EJOURNAL.medium, EJOURNAL.ISSN, EJOURNAL.language, EJOURNAL.frequency, EJOURNAL.userInstruct, PROVIDER.providerURL, PROVIDER.providerIcon, EJOURNAL.target, EJOURNAL.DRADBCN, EJOURNAL.trialCheck

FROM ((ACCESS RIGHT JOIN EJOURNAL ON ACCESS.accessID = EJOURNAL.accessID) LEFT JOIN PROVIDER ON EJOURNAL.providerID = PROVIDER.providerID) LEFT JOIN VENDOR ON EJOURNAL.vendorID = VENDOR.vendorID

WHERE EJOURNAL.displayFlag='yes';

2) Main SQL Query within *EIDList.cfm:*

<cfquery datasource="EResources" name="list" dbtype="ODBC">

SELECT vAllInfo.EIDID, vAllInfo.EIDName,vAllInfo.accessDesc, vAllInfo.providerName, vAllInfo.vendorName, vAllInfo.EID_URL, vAllInfo.instructURL, vAllInfo.medium, vAllInfo.coverage, vAllInfo.updated, vAllInfo.instructions, vAllInfo.providerURL, vAllInfo.providerIcon, vAllInfo.description, vAllInfo.target, vAllInfo.trialCheck, vAllInfo.subscriptEndDate, EID.EIDName as RefName, EID.EID_URL as RefURL, EID.target as RefTarget

FROM EID RIGHT JOIN vAllInfo ON EID.EIDID=vAllInfo.crossRefID
<cfif #letter# neq "ALL">

WHERE vAllInfo.EIDName like '#letter#%'</cfif>

ORDER BY vAllInfo.EIDName, vAllInfo.vendorName, vAllInfo.providerName ;
</cfquery>

Appendix F Code Sample from *Subject.cfm*

```
<!-- new row, with table, holds Principal Indexes gray bar/header ** if there are principal indexes-->
<cfif corequery.recordcount gt 0>
<BR><TR><td>
      <TABLE cellspacing="0" BORDER="0" WIDTH="680"  BGCOLOR="#cccccc">
        <TR><TD><A NAME="principal"><B><SPAN CLASS="subhead">Principal Indexes </SPAN>
</B></A></TD><TD ALIGN="RIGHT">
<cfif noncorequery.recordcount gt 0>
<SPAN CLASS="smaller"><A HREF="#other">Other Resources</A></SPAN><cfelse> </cfif>
        </TD></TR></TABLE>
<br></TD></TR>                        </cfif>

<!-- new row, another table, holds principal indexes data - if any --><TR><TD>
<cfif corequery.recordcount gt 0>
<cfoutput query="corequery">
<table width="680">
<tr><td><a name='#target#'></a>
<cfif #EID_URL# eq "">
<b>#EIDName#</b>
<cfelse>
<a href="#EID_URL#"><b>#EIDName#</b></a></cfif>

.....code for other EID fields here.....

</td></tr>
</table></cfoutput> <!-- close table and output with principal data -->
</cfif><br>
</TD></TR><!-- end row containing data and end if statement regarding principal indexes -->

<!--start new row, with if statement regarding other indexes, this statement prevents gray header bars from
showing for certain subjects, like News and reference where there are no principal vs. other indices -->
<TR><TD><cfif noncorequery.recordcount gt 0 and corequery.recordcount gt 0>

<!-- table with Other Indexes gray bar/header -->
 <TABLE cellspacing="0" BORDER="0" WIDTH="680" BGCOLOR="#cccccc">
        <TR><TD><A NAME="other"><B><SPAN CLASS="subhead">Other
<cfoutput>#subjectName#</cfoutput> Resources</SPAN></B></A></TD>
        <TD ALIGN="RIGHT"><SPAN CLASS="smaller"><A HREF="#principal">Principal
Indexes</A></SPAN></TD></TR></TABLE>
</cfif>
</TD></TR>

<TR><TD><!-- new row, another table, holds other indexes data - if any -->
<br><cfoutput query="noncorequery">
<!-- table with resource name, provider, vendor info -->
<table width="680">
<tr><td><a name='#target#'></a>
<cfif #EID_URL# eq ""><b>#EIDName#</b>
<cfelse><a href="#EID_URL#"><b>#EIDName#</b></a></cfif>

.....code for other EID fields here.....

</td></tr></table></cfoutput> <!-- close table & output with "other indices" data -->
```

Appendix G Sample Code from <CFLoop> function (*EJlist.cfm)*

```
<cfquery datasource="EResources" name="list" dbtype="ODBC">
SELECT ejAllInfo.EJID, ejAllInfo.EJName
FROM ejAllInfo
<cfif #letter# neq "ALL">
WHERE ejAllInfo.EJName like '#letter#%'</cfif>
ORDER BY ejAllInfo.EJName
</cfquery>

<cfloop query="list">
<cfset idNum = #EJID#>

<cfquery datasource="EResources" name="fullList" dbtype="ODBC">
SELECT ejAllInfo.EJID, ejAllInfo.EJName,ejAllInfo.accessDesc, ejAllInfo.providerName,
ejAllInfo.vendorName, ejAllInfo.EJ_URL, ejAllInfo.instructURL, ejAllInfo.medium, ejAllInfo.holdings,
ejAllInfo.ISSN, ejAllInfo.language, ejAllInfo.frequency, ejAllInfo.userInstruct, ejAllInfo.providerURL,
ejAllInfo.providerIcon, ejAllInfo.target, ejAllInfo.trialCheck
FROM ejAllInfo
WHERE ejAllInfo.EJID=#idNum#
ORDER BY ejAllInfo.EJName, ejAllInfo.vendorName, ejAllInfo.providerName ;
</cfquery>

<cfquery datasource="EResources" name="format2" dbtype="ODBC">
SELECT *
FROM EJOURNAL_FORMAT, FORMAT
where EJOURNAL_FORMAT.EJID=#idNum# and
(EJOURNAL_FORMAT.formatID=FORMAT.formatID)
</cfquery>

<cfset formats="">
<cfoutput query="format2">
<cfset formats=ListAppend(formats, #formatName#)>
</cfoutput>

<cfoutput query="fullList">
<tr><td><a name='#target#'></a>
<cfif #EJ_URL# eq "">
<b>#EJName#</b>
<cfelse>
<a href="#EJ_URL#"><b>#EJName#</b></a></cfif>

.....code for other ejournal fields here.....


<cfif #formats# neq ""><br><b>Format(s):</b> #formats#</cfif>

<br></td></tr></table>
</cfoutput>
</cfloop>
```

Appendix H Server Output Code - CFLoop Function

http://dbserv.ils.unc.edu/projects/EResources/ejournal/EJlist.cfm?letter=Z

Queries

```
list (Records=2, Time=1368ms)
SQL =
SELECT ejAllInfo.EJID, ejAllInfo.EJName
FROM ejAllInfo

WHERE ejAllInfo.EJName like 'Z%'
ORDER BY ejAllInfo.EJName

fullList (Records=1, Time=442ms)
SQL =
SELECT ejAllInfo.EJID, ejAllInfo.EJName,ejAllInfo.accessDesc,
ejAllInfo.providerName, ejAllInfo.vendorName, ejAllInfo.EJ_URL,
ejAllInfo.instructURL, ejAllInfo.medium, ejAllInfo.holdings,
ejAllInfo.ISSN, ejAllInfo.language, ejAllInfo.frequency,
ejAllInfo.userInstruct, ejAllInfo.providerURL, ejAllInfo.providerIcon,
ejAllInfo.target, ejAllInfo.trialCheck
FROM ejAllInfo
WHERE ejAllInfo.EJID=10497
ORDER BY ejAllInfo.EJName, ejAllInfo.vendorName, ejAllInfo.providerName
;

format2 (Records=1, Time=245ms)
SQL =
SELECT *
FROM EJOURNAL_FORMAT, FORMAT
where EJOURNAL_FORMAT.EJID=10497 and
(EJOURNAL_FORMAT.formatID=FORMAT.formatID)

fullList (Records=1, Time=74ms)
SQL =
SELECT ejAllInfo.EJID, ejAllInfo.EJName,ejAllInfo.accessDesc,
ejAllInfo.providerName, ejAllInfo.vendorName, ejAllInfo.EJ_URL,
ejAllInfo.instructURL, ejAllInfo.medium, ejAllInfo.holdings,
ejAllInfo.ISSN, ejAllInfo.language, ejAllInfo.frequency,
ejAllInfo.userInstruct, ejAllInfo.providerURL, ejAllInfo.providerIcon,
ejAllInfo.target, ejAllInfo.trialCheck
FROM ejAllInfo
WHERE ejAllInfo.EJID=10498
ORDER BY ejAllInfo.EJName, ejAllInfo.vendorName, ejAllInfo.providerName
;

format2 (Records=1, Time=19ms)
SQL =
SELECT *
FROM EJOURNAL_FORMAT, FORMAT
where EJOURNAL_FORMAT.EJID=10498 and
(EJOURNAL_FORMAT.formatID=FORMAT.formatID)
```

Appendix I Server Side Error Checking (<CFAbort> Function)

```
<CFquery name="EIDIDcheck" datasource="EResources">
SELECT *
FROM EID
WHERE EIDID=#EIDID#
</cfquery>

<cfif EIDIDcheck.RecordCount gt 0>
<br><br>
<h3>Administrative Error!</h3><HR color="blue">
<br>
<b>Oops.</b> The ID number calculated already exists. This occurs if you did not reload the
page before entering another record. Please return to the form and reload the page or click on the
button below.  A new EIDID will be automatically generated.
<P>
<form action="addEIDform.cfm" method="post">
<input type="submit" value="Add an Index or Database">
</form>
<P>
Thank you.
<CFABORT>
</cfif>
```