# Designing Web Database Applications for Ecological Research

**Dan J. SMITH**
**Barbara J. BENSON***
**and David F. BALSIGER**

**Center for Limnology, University of Wisconsin-Madison**
**Madison, Wisconsin 53706, USA**
* corresponding author

**ABSTRACT**

Many sites conducting ecological research must routinely manage a diverse suite of datasets and make them accessible to researchers. This paper presents an approach to creating an ecological data query system that dynamically creates predefined dataset query interfaces for managed datasets. The query interfaces are created from stored dataset metadata and query creation metadata and include only those field selections and filtering options identified as relevant for the specified dataset. Using only stored metadata for query interface creation provides an extensible and scalable metadata framework, creating a standardized, yet robust, system that can handle diverse datasets. This metadata framework also provides opportunities for developing centralized querying of remote datasets and multi-site data exchange through the use of predefined dataset query interfaces.

**Keywords:** Ecoinformatics, Metadata, Dynamic Database Access, Ecological Metadata Language (EML), XML, Query System, Web Database Interface.

## 1. INTRODUCTION

The World Wide Web has created a ubiquitous environment for access to scientific information. At the North Temperate Lakes Long Term Ecological Research (NTL-LTER) site [1][2], the information management staff has developed a Java application [3] that provides a web interface to a relational database housing a diverse collection of long-term physical, chemical and biological limnology datasets. A query engine driven by metadata tables stored in the same database generates the user interface.

Researchers value the ability to easily extract only the data relevant to a particular research question and have that data in a format readily transported to their favorite analysis packages. This dynamic query application permits the selection of dataset attributes of interest and allows the construction of conditional filters on dataset attributes to accurately specify the data that are relevant to the researcher. The user can select from a variety of output formats as well as request sorting of the data by designated fields.

The Long Term Ecological Research (LTER) [4] program is a network of 24 sites funded by the National Science Foundation to study long-term phenomena in ecosystems. The LTER sites are committed to making data publicly available. Many sites meet this commitment by serving text data files from their websites. This approach limits the ability of a user to select only the data of interest. Other sites create a static template for each data set to permit dynamic database queries. Our approach was to generate a web form for dynamic queries "on-the-fly" for all data in the database using a standard description of the data that was also stored in the database. This design not only provides dynamic access to the database but also is very extensible because of the ease of adding the description of a new dataset to the database.

The standard description of the data was based on an emerging metadata standard for ecological data, Ecological Metadata Language (EML) [5][6]. In this paper we describe some extensions that were made to the metadata standard to support query features such as filtering. We also describe the architecture of the software and discuss its features and opportunities for future development that would support multi-site interoperability and data exchange. The extension to multi-site data sharing will be made possible by the acceptance of a common metadata standard.

## 2. QUERY SYSTEM DESIGN

Several desired features guided the creation of this query system. The system needed to meet the following criteria:

- deliver consistent and current data on demand
- handle diverse table layouts for different datasets
- export EML metadata for each managed dataset
- allow retrieval of only data relevant to the user
- provide the data in various output formats including XML
- maintain data usage and access statistics
- include access control on datasets and users
- scale well to multiple sites to allow data interchanges

The prior system at NTL-LTER, publishing datasets as static data files on the Web, did not meet any of these goals. A system capable of meeting these goals must have access to a stored description of the dataset and present the user with various options for data retrieval based on that description.

The role of the query program (Figure 1) is to act as a conduit between the user and the datasets stored in the database. It guides the user to a particular dataset, presents information about that dataset, translates user choices into a query, and returns a recordset adhering to the conditions of the user request.

The heart of our query prototype is the interface that is dynamically generated for each dataset. This interface has four main tasks:

- providing the acceptable query specification choices to the user (the search framework) for each dataset
- accepting and verifying user input into an input framework
- creating a query to be executed by the query engine
- receiving and formatting the results of an executed query

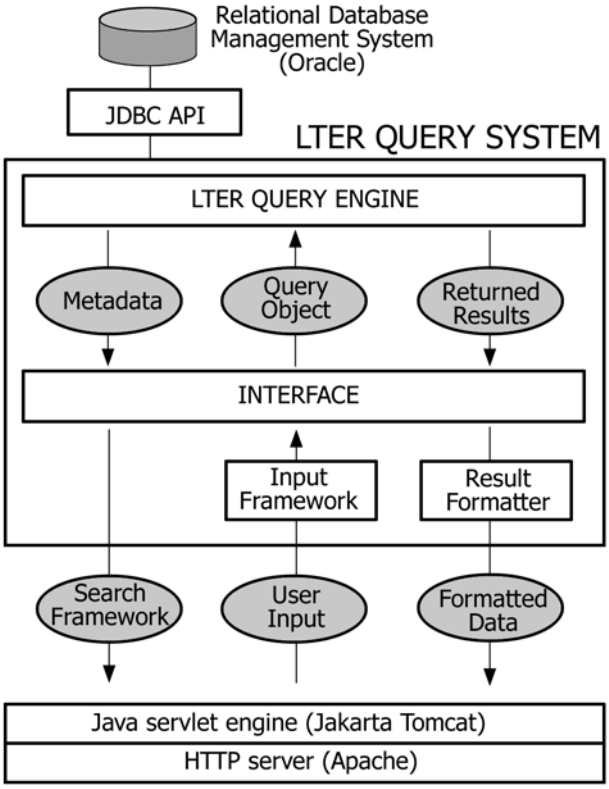


**Figure 1.** Schematic diagram showing the objects and components of the dynamic query application.

This system was designed to be fully extensible and to easily accommodate new datasets with no additional programming or software engineering. In other words, the same software must be able to handle any type of dataset and its associated filters, conditions, and sorting based on stored, known information. In order to accomplish this functionality, information about the datasets must conform to a standard representation. Thus, the first task in implementing the query prototype was to create a standard description of the various datasets contained in the database.

## 3. METADATA

The datasets managed by the NTL-LTER are diverse in the number of fields and field types. In order for the query system to handle these different datasets, a common descriptive language is required. This language allows the query system to understand the table structures of all the datasets in order to query them appropriately and successfully. The language effectively encapsulates the diverse datasets into uniformly accessible entities.

**Metadata Content Standard**

To perform this encapsulation, our implementation uses a beta release of the Ecological Metadata Language (EML) version 2 as our data definition language. EML, an emerging standard in ecological metadata, is being developed by the National Center for Ecological Analysis and Synthesis in collaboration with the Long Term Ecological Research program and the San Diego Supercomputer Center. The content standards in EML are based on standards for ecological metadata developed by the ecological community (Michener et al. 1997 [7]) and are implemented in Extensible Markup Language (XML) [8]. EML consists of a set of modules, each of which specifies one logical part of the complete metadata.

**Table 1.** Description of EML modules used by the query system.

| EML module | Description |
|---|---|
| EML-project | describes the research context in which the dataset was created |
| EML-dataset | contains general information about the dataset; includes abstract, contacts, coverage, etc. |
| EML-attribute | describes attributes (variables) in a dataset; includes type, range, definitions of coded values, etc. |
| EML-party | describes a responsible party (person or organization); contains detailed contract information for the party |

We used the project, dataset, attribute, and party modules (Table 1) from EML as part of the content standard for our metadata. Some additional information that has traditionally been part of the NTL-LTER metadata is not included in EML. These fields were retained in the NTL-LTER metadata content.

The dataset module of EML was reproduced and extended to form the center of the entity-relationship model. Entities in our model reference this dataset object through the use of a unique dataset ID. This linkage provides easy access to all the information the program uses to perform queries on specific datasets.

To describe the data in each dataset, we included the attribute module of EML. Linked to specific datasets, the attribute entity describes the type of data stored in the dataset as well as information about measurement units, precision, enumerated domains, and data quality for each dataset attribute. To store information about people involved with each dataset, we constructed a party entity based on the EML party module. This entity is linked to a role entity that associates individual investigators and contact people with a given dataset. This type of information is required to export our traditional metadata included with our datasets in EML form.

**Extended Metadata**

The EML-based dataset descriptions do not include instructions to the query system as to how the metadata and the datasets described therein should be handled. Filtering and display information is still needed in order to present the user with appropriate selection criteria. Accommodating this additional handling information required an extension to the stored metadata. Including this additional information in the extended metadata enabled the development of a deterministic query

system, fully database driven, which is capable of interpreting and creating the query interface in a standard way for all datasets. This well-defined behavior is needed in multi-site scenarios.

The query interface provides the user with a selection of fields to be included from a dataset, and in addition, offers the user a selection of data filters appropriate to that dataset. Fields within a dataset that might be useful and appropriate as data filters for that dataset are specified within the extended metadata. In addition, the extended metadata stores information as to which of the five available types of filters will be used in each case. Storing this information as extended metadata allows the system to accommodate new filters on additional attributes easily and facilitates the quick integration of new datasets and their associated query interfaces into the program.

**Table 2.** Description of metadata extensions used by the query system to enable dynamic query interfaces.

| Metadata Extensions | Description |
|---|---|
| Dataset Extensions | data access restrictions based on specified dates |
| Attribute Extensions | restrictions on availability of fields for querying and suppression of null records for sparse datasets |
| Filtering Extensions | module created to store filter definitions (Figure 3) |

To accommodate the extended metadata, we expanded our previous entities along with creating several new entities (Table 2), which are not part of the current version of EML. One important new entity contains the query display information that controls the data filtering available in a specific query interface. To accommodate a variety of filter types, this entity matches a specific type of filter with a particular dataset attribute, allowing for fine-tuned filtering descriptions that are specific to individual attributes and datasets. There are currently five types of filters possible for a query:

- enumerated codes displayed as a list (e.g. selecting a subset of lakes)
- date range
- year range
- range on attribute values
- multiple selections from a list of values; displayed as two boxes: the original list of codes and the selected list (e.g., selecting species names)

Additional information was needed to provide access restrictions on certain data by date as well as information about handling sparse datasets. The access restrictions were included in the previously created dataset entity since they apply to all records in the dataset. Information about display and result suppression for individual attributes was placed in the attribute entity and is applicable within each dataset that includes that attribute.

## 4. PROGRAM COMPONENTS

The query program creates a search framework that provides the user with query choices, an input framework that accepts and verifies the input from the user, a database query based on user input, and formatted results returned to the user.

**Search Framework and Filter Creation**

The task of the search framework (Figure 2) generated by the query interface is to provide the possible appropriate query condition choices to the user. The search framework creation begins with the gathering of information about the data contained in the dataset of interest. This information is stored in the standardized metadata in the database. The data description for the chosen dataset is queried from the database backend and delivered to the interface.

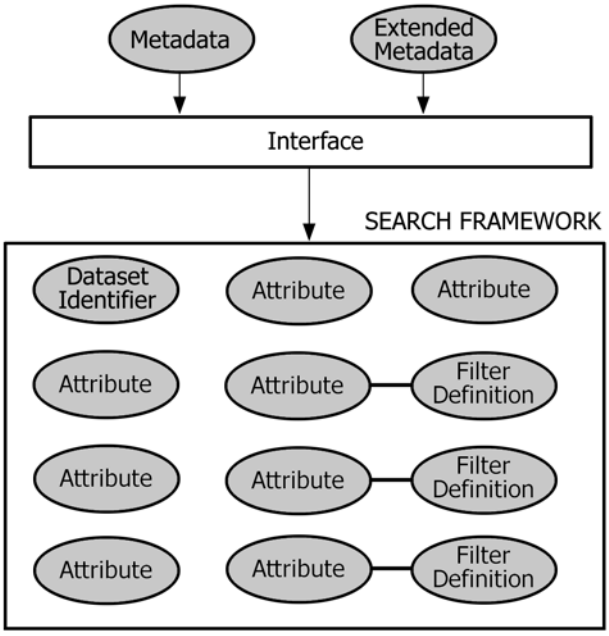


**Figure 2.** Schematic diagram of the search framework and its contained objects. The program interface gathers metadata for a particular dataset to create a search framework with the appropriate attribute definitions and associated filters.

Queries on datasets include pre-specified filters (Figure 3) to allow the user to limit retrieval to only relevant data. The "filter type" specifies which type of filter has been defined and how the filter will be rendered to the user. It also specifies if the defined values object or dates object will be used for this filter. The defined values are enumerated codes derived from EML attribute metadata, database tables, or stored queries. Defined values objects are used in enumerated code or multiple selection queries. The dates object specifies the acceptable range of dates for temporal queries. Filter title and filtering phrases are used to describe to the user the function of the filter and the necessary input from the user.

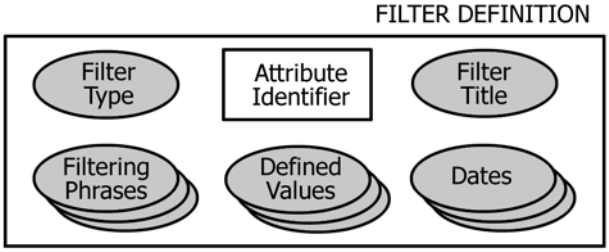


**Figure 3.** Diagram showing the filter object created by the interface and placed into the search framework.

After obtaining the dataset and attribute descriptions, the interface has structural knowledge of the specific fields present in the dataset, along with the description of the dataset attributes. The interface does not yet contain information as to which types of filtering criteria should be attached to those fields. The interface again queries the database backend and requests the extended metadata that contains the filtering conditions that will be presented to the user for this dataset.

These filter definitions are extracted from the extended metadata and attached to a specific attribute. After filtering information has been added, the search framework is completed and the user is presented with a choice of field selections and filters for the dataset. In the current implementation, this framework is transformed into a query form that is presented to the user as a web browser form. In a multi-site scenario, this search framework would be encoded and presented to a remote site in an XML representation, or alternatively could be derived by the remote site from downloaded metadata retrieved in XML.

**User Input**

The user input to the interface contains various types of information ranging from the fields selected for output to filters on ranges and enumerated codes. To manage all this incoming information, the interface must create an input framework for the given dataset. This user-input framework tells the interface what type of input data to expect in order to match the incoming information about the user-specified query conditions to the specified dataset.

The user-input framework is constructed in much the same way as the previous search framework using metadata and extended metadata from the database backend. In addition, the system retrieves any information about restrictions placed on access to the data in this dataset and incorporates this into the framework. This user-input framework can be thought of as an inverse of the earlier constructed search framework. Whereas the search framework specifies to the user what type of input is acceptable for a given dataset, the user-input framework specifies to the query interface what type of data it might be receiving. The user input is then matched against this framework to create an object that represents the user's query.

The user input in a multi-site system would be encoded by the central query system into an XML encapsulated query object. This encapsulation of the query object is possible because of the self-descriptive nature of XML data. The query object would be transported directly to the appropriate remote query engine using the SOAP [9] protocol for object messaging.

**Query Creation and Execution**

After the user input has been parsed and matched against the input framework, a query object exists in the interface. The query object specifies which data fields are to be retrieved, along with various selection criteria for the fields. In addition, any system-imposed selection criteria such as access restrictions or null suppression on sparse datasets are added to the query object as well.

The query object is systematically parsed and translated into a standard SQL command. This newly created SQL query is passed to the query engine where the query is run against the data source containing the dataset. The tuples returned by the query are then handed back to the interface for output.

**Output of Query Results**

The formatting of the output is the final task of the interface. The result output routine takes the set of tuples returned by the query and transforms them into a file or stream in the user-specified format. The choice of output format was presented earlier to the user in the search framework, collected as user input, and stored in the query object.

Because the data are returned from the database simply as a set of tuples, any output format is easily assembled. The interface processes the returned recordset and converts it into the chosen format, adding data description and formatting to the file. The output formats included in the current implementation are XML, web browser display, Microsoft Excel, and comma-delimited text. The web browser display conveniently breaks the result set into smaller screens (pages) and includes easy navigation between pages. The metadata for the dataset can optionally be included in the result file, and due to the design of the system, the metadata included in the XML output format has the added benefit of conforming to the EML standard.

## 5. EXTENSIBILITY

This data query system is driven entirely by tables residing in a database backend. This design provides extensibility for the creation of new interfaces, as can be seen in the ease of adding additional datasets, modifying query interfaces, and in the organization of datasets into hierarchies.

One approach for implementing dynamic database access is to create a unique or customized data query program for each individual dataset based on it's own distinctive layout. Our prototype, instead, utilizes a single program that can access any number of datasets as long as the system has access to their metadata descriptions. Extending the query interface for a new dataset table simply requires placing the metadata for the dataset into the database. This universal program approach allows for any number of additional datasets to be added to the query engine without any additional program development. In addition, when datasets evolve or are modified, the associated metadata can be easily updated and immediately reflected by the search program.

This system also allows updating of handling and filtering options without any changes to the program. The advantage here is that it eliminates the need for creating or updating multiple programs when datasets or query interfaces change.

With a large number of datasets, organization is essential for allowing researchers quick and straightforward access to the relevant data. We group the datasets by category. By adding a category entity to the extended metadata, an extensible system was created for classifying individual datasets. The entry document for the data catalog on the web is generated dynamically from the information stored in the category entity in the database.

A project level entity was also added to encapsulate the various categories. Unrelated projects can share the same query system while their data are kept separate and consistent. Different user interfaces are also possible using templates and cascading style sheets on a per project basis. Our prototype implementation is currently querying and serving NTL-LTER data, Biocomplexity Project data, and Microbial Observatory data.

## 6. MULTI-SITE INTEROPERABILITY

An important goal is to design information systems that allow users to find relevant data quickly. When a researcher is looking for data that are hosted at a separate site, the data acquisition process can become slow. Allowing researchers to run queries across multiple sites in search of data from a central location would streamline this process.

Based on our evaluation of this prototype, we believe there are three main requirements for achieving consistent query interfaces for datasets hosted across multiple LTER sites. This prototype fulfills several of these initial requirements for multi-site searches. The first requirement is a standard descriptive language for the data. For ecological research, EML appears to be the emerging data definition language of choice. When EML is finalized, data descriptions for datasets from multiple sites will be available to any system that conforms to this standard.

The second requirement is a standard format for data representation. In our prototype, we have implemented four distinct data representations for data output. XML is the most extensible and allows for the inclusion of EML metadata within the actual dataset data. For these two reasons, our preferred approach, which is implemented in this prototype, is to serialize the returned records in XML and include the EML description of the data. One problem with XML data retrieval is that it is not as widely accepted as comma delimited text files. For this reason, our prototype allows researchers to choose from several formats for the results of their queries. We expect that XML will gain more widespread usage as more applications become XML compliant.

The third requirement is a descriptive language that allows the query systems to define filters for datasets. Filters could be created dynamically from the EML data representations, but we feel that this approach is not optimal. It is difficult to add filters for a dataset that would be relevant to the researcher without knowing more information than is presented by EML descriptions of attributes. For complex datasets, a large number of different filters could be possible, leading to the user being overwhelmed with unneeded criteria selections. For these reasons, a standardized language to describe relevant filters is needed. Our prototype uses the extended metadata to implement the description of possible filters. Multiple sites will be able to interpret the extended metadata in a deterministic way, enabling the creating of a standard query interface. While this is a start toward meeting this third requirement, the extended metadata that we have created is not a standard nor is it implemented in any other system.

Upon the development of a standardized language for implementing selection and filtering descriptions, a central site will be able to generate query interfaces, as defined by the data managers at the remote sites, for remote datasets. With the ability to create a query interface for a remote dataset, coupled with the ability to serialize query results, multi-site or centralized querying of remote LTER datasets will be attainable.

## 7. CONCLUSIONS

The NTL-LTER query system resulted from a need to create a system that was more robust than the previous system of downloadable static data files. This was accomplished by shifting from static data files to data delivered directly from the backend data warehouse. This query system has achieved many of our intended goals and has created opportunities for future expansion and development. One significant area that this system does not address is the querying and retrieving of spatial data.

At NTL-LTER, spatial data are not stored in the relational database. The current dynamic database prototype provides a link to a separate system for accessing spatial data which are stored in a variety of vector and raster formats. The type of query interface optimal for spatial data would extend beyond that implemented in our prototype to include a flexible array of spatial query and mapping tools. This approach might incorporate web-based geographic information system (GIS) software, such as ArcIMS from Environmental Systems Research Institute (ESRI). The goal of such a system would be to permit the user to query and retrieve spatial data using both spatial and non-spatial search criteria. In addition, the results of spatial data queries should be available both as map graphics in the user's web browser and in a standard vector or raster format suitable for later analysis. The beta version of EML 2.0, upon which the NTL-LTER query system is based, does not contain a module for spatial data although standards exist for spatial metadata [10][11].

The challenge in the creation of this system was defining how query interfaces for datasets would be created. Our solution to this problem uses a metadata framework to describe the datasets and query interfaces. All query-related information needed in the query creation process is based on and interpreted from stored metadata. A key feature of the design was to extend EML to include metadata that describe display and filtering information. This extended information enables the creation of unique predetermined query interfaces for each dataset. The reliance on stored metadata allows for changes to and additions of datasets and query interfaces without additional software engineering costs. The metadata definitions used in the program also provide the needed information to export EML upon request.

XML documents conforming to standard Document Type Descriptors, such as those for EML, will soon be a widely used exchange medium for data and metadata for ecological research. These common data definition formats will allow for rapid and standardized information exchange and inter-site operability within ecological research networks. Since the query program described here is based on these standards, it is staged to deal with these future multi-site scenarios.

## 8. ACKNOWLEDGEMENTS

The authors gratefully acknowledge Matt Jones for comments on our EML implementation and the reviewers of this paper for their insightful critiques. This paper is based on research supported by a grant from the National Science Foundation (DEB-9632853).

## 9. REFERENCES


[1] Benson, B. J. 1996. The North Temperate Lakes LTER research information management system, Proceedings of Eco-Informa '96: Global Networks for Environmental Information, volume 11, Environmental Research Institute of Michigan, Lake Buena Vista, Florida, 4-7 November 1996, pp. 719-724 .

[2] North Temperate Lakes LTER http://lter.limnology.wisc.edu/

[3] NTL-LTER query engine http://lterquery.limnology.wisc.edu/

[4] Long-Term Ecological Research program http://lternet.edu

[5] Nottrott, R., M. B. Jones, and M. Schildhauer. 1999. "Using XML-structured metadata to automate quality assurance processing for ecological data." Proceedings of the Third IEEE Computer Society Metadata Conference. Bethesda, MD. April 6-7, 1999.

[6] Knowledge Network for Biocomplexity http://knb.ecoinformatics.org/software/eml/

[7] Michener, W. K., Brunt, J. W., Helly, J., Kirchner, T. B., and S. G. Stafford. 1997. Non-geospatial metadata for the ecological sciences. Ecological Applications 7:330-342.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C Recommendation 6 October 2000 http://www.w3.org/TR/REC-xml

[9] "SOAP Version 1.2 Part 0: Primer" W3C Working Draft 17 December 2001 http://www.w3.org/TR/soap12-part0/

[10] Federal Geographic Data Committee. FGDC-STD-001-1998. Content standard for digital geospatial metadata (revised June 1998). Federal Geographic Data Committee. Washington, D.C.

[11] Federal Geographic Data Committee http://www.fgdc.gov/metadata/contstan.html