XML in a Data Warehouse Design: Performance Evaluation Utilizing Geological Data

Behrooz Seyed-Abbassi, University of North Florida, 4567 St. Johns Bluff Road S, Jacksonville, FL 32224; E-mail: abbassi@unf.edu
Lori Stowers Pusey, University of North Florida, 4567 St. Johns Bluff Road S, Jacksonville, FL 32224; E-mail: puseyl@aetna.com

ABSTRACT
The structure of XML data presents challenges when determining efficient ways to map it to a relational data warehouse. One of these challenges is the presence of multi-valued child elements. Translating multi-valued XML elements to a relational data warehouse may require the consideration of non-traditional approaches to the design of the data warehouse. Traditionally, the star schema has been the design of choice for data warehouses. However, the semi-structured nature of XML data may make the use of alternatives, such as the snowflake schema, more efficient. This paper presents the implementation details of methods for preserving XML data in a relational data warehouse that are based on the star and snowflake design schemas and compares the methods quantitatively and qualitatively for geological data sets.

Keywords: XML, data warehouse, star schema, snowflake schema, relational database, geological data

1. INTRODUCTION
Extensible Markup Language (XML) has become the accepted standard for exchanging data over the Internet [1]. As more organizations choose to collaborate and share information on the Web, the demand to access and mine XML data is dramatically increasing which in turn is resulting in the need to utilize design methodologies for effective storage and retrieval or warehousing of the XML information.

Many research projects, as well as software corporations (such as, Oracle [2], Microsoft [3], and IBM [4]), have initiated techniques and various methods for creating, storing, and retrieving XML information [5][6], including a variety of proposed methodologies for translating XML data to a relational environment [7]. Several published works cite the need to store XML data in a file system or a relational database environment, as well as address some of the difficulties that arise from the differences between the structure and semantics of XML data and that of relational data [8][9].

As noted by Shanmugasundaram, native XML databases do not have the "sophisticated storage and query capability already provided by existing relational database systems" and "do not allow users to query seamlessly across XML documents and other data stored in relational database systems" [10]. Two complications presented by XML data have been identified as recursion and multi-valued (or set-valued) elements [11].

In a relational data warehouse, the main issue in storing XML data is the potential inability to extract all the information necessary from the XML document and definition to develop a data warehouse design that accurately represents the XML data [12]. Another problem is the differences in "expressive power" of the relationships presented in an XML document's definition [13]. The source of these issues is the semi-structured nature of XML.

The presented research work addresses design issues of storing multi-valued XML data with many-to-many relationships, which results in multiple associations within a relational data warehouse. Two alternative methodologies for preserving this type of XML data are evaluated. Both methods involve decomposing the XML elements and storing their values in a relational table within the data warehouse. The first proposed alternative makes use of the snowflake schema in the design of the data warehouse tables, while the second alternative utilizes the traditional star schema. The details of the implementation of these alternative methods, along with quantitative and qualitative comparisons of the methods are described.

The paper is divided into the following descriptive sections. Background information on XML and the relational paradigm are given in sections 2 and 3 respectively, with XML storage and mapping in a relational environment considered in section 4. The alternative methodologies and the comparative study are described in section 5, followed by more specific information about the implementation processes in section 6. The testing results from running SQL queries against each method's data tables are given in section 7, with section 8 providing the conclusions of the research.

2. XML CHARACTERISTICS
XML is a text-based language with user-defined tags that preserve the semantics and relative context of information. The tags add flexibility to XML documents and semantic representation. XML has become the preferred language for exchanging data over the Web for two reasons. It is based on a standard and, therefore, vendor-neutral, and since it is text-based, XML can be viewed within any text editor [1]. In order to allow the consumer of an XML document to verify its validity, a document type can be used in conjunction with the document to define the allowable structures.

The three basic methods for storing XML documents are as a flat file, XML database, and relational database [14]. The flat file approach allows for the access of a specific XML document through the traditional file system hierarchy. The XML database approach involves the "direct access to XML documents and fragments of documents, and the ability to query across those documents and fragments" [14]. The relational database approach is more complex and involves decomposing (shredding) the XML document and storing element values in table fields. Of the three approaches for storing and accessing XML data, only the relational database allows for the creation of complex queries, can be integrated with other relational data, and provides mechanisms for transaction management and recovery [14].

3. RELATIONAL PARADIGM
Relational database, introduced by E. F. Codd in 1970 [15], is presented by collections of tables (or relations) that contain data items with similar properties (or attributes). In a table, each column corresponds to one of the attributes, and each row (or tuple) represents a piece of data (or record) containing those attributes. Data warehouses, which typically contain historical data or facts and are structured specifically for querying and reporting, can take advantage of the structure of relational database for design and implementation. However, because the data is historical or factual and unlikely to change, data warehouses tend to be denormalized and the data treated as read-only [15].

Two common schemas used for a data warehouse are the star and snowflake [16]. The star schema contains a single fact table that holds factual data (typically transactions) surrounded by multiple dimension tables that contain reference data (i.e., components that comprise each fact or transaction) [15]. The fact table consists of a unique primary key for each fact, a foreign key reference to the primary key from each of the dimension tables, and optional measure data (such as quantity). The star schema models a many-to-one relationship between the fact table and each of the dimension tables.

The snowflake schema allows dimension tables from a star schema to be "organized into a hierarchy by normalizing them" [16]. This schema, therefore, allows for multi-valued attributes associated with a dimension table to be modeled using additional, hierarchical tables. However, because a data warehouse can contain an enormous number of records, performing joins between numerous tables can be costly in terms of query response time.

Typically, the star schema is used to represent the multi-dimensional data warehouse model and is better suited for querying, over the snowflake schema, because the data is denormalized [17]. The more normalized snowflake schema, while providing advantages when maintaining the data [17], requires additional table joins for queries that access the normalized data and can, therefore, increase the query response time [18].

4. STORING XML DATA IN RELATIONAL TABLES

There are three basic techniques to decompose (or shred) XML data for storage in relational tables: no decomposition, partial decomposition, and total decomposition [19][20]. They are differentiated by the extent to which the XML data is decomposed and by the way that XML is stored in relational tables. The decompositions into relational tables generally fall into two categories: those that start with an XML document and those that do not. Each approach varies in the number of relational tables that are created and in the structural information and element values that are captured.

The no decomposition approach (document-centric) involves storing the entire XML document as text in a flat file or relational table's field (e.g., as a character large object-CLOB) [20]. In partial decomposition, XML data is selected, decomposed, and stored in tables to allow quick access to specific data through SQL queries [19]. Total decomposition (data-centric) involves decomposing the entire XML document and storing its elements' and attributes' values in table fields. The third approach fully exploits the query power of relational databases, but is a complex implementation due to the differences in structure between XML documents and relational tables, and cardinality uncertainties inherent in XML schemas.

A schema-based approach for storing XML data in a relational data warehouse proposed by Golfarelli starts by translating an XML document into a graph and an algorithm applied to the graph to create an attribute tree from which a data warehouse conceptual design is developed. Golfarelli's method, unlike other methods, emphasizes the determination and evaluation of relationship cardinalities within the graph. These cardinalities show a to-one or to-many relationship between elements and their child elements (or sub-elements) [13].

Because of their structure, XML documents benefit from either the partial or total decomposition for storing XML information in a relational data warehouse. Both methods can handle multi-valued sub-elements in XML format using Golfarelli's methodology for to-one and to-many associations.

5. DATA WAREHOUSE DESIGN METHODOLOGIES

The focus of the proposed methodologies is on the problem of how to best capture multi-valued XML data within the confines of a relational data warehouse. Because of the inherent structure of a data warehouse, the fact table often contains a very large number of records, which means that performing table joins between the fact and dimension tables can be costly in terms of query response times. This is one reason why the denormalized star schema is generally the preferred design for a data warehouse.

The first methodology decomposes non-multi-valued elements and stores their values in data warehouse dimension table fields similar to Golfarelli's approach. However, instead of dropping multi-valued child or sub-elements altogether, the method is modified in order to capture the multi-valued sub-element data by storing the data in XML format within a field. Because the multi-valued elements are not decomposed, no table hierarchies are created and the star schema is maintained.

The second proposed alternative uses a variation of the star schema that includes additional hierarchical tables, thus creating a snowflake schema. This method, like the star schema, fully decomposes the XML data and captures its element values in the data warehouse tables and makes use of an additional table to hold the decomposed multi-valued element data. The additional table is then linked via a foreign-key field to the dimension table that holds the parent element data; thus creating a one-to-many relationship between the dimension table and the added table creating a normalized snowflake.

6. IMPLEMENTATION PROCESS

The following subsections summarize the implementation process for the two proposed design methods, including the XML data chosen for translation and storage, the design of the data warehouses, and the queries applied to the data tables.

6.1 Geological XML Data

To evaluate the performance of XML in the design of a data warehouse, the initial testing has been completed with text-based geological XML data to provide the foundation for future expansion to support more complex semi-structured data. The implementation process utilized the U.S. Geological Survey's (USGS) website [21] for historical XML data available in their earthquake advisories. The historical earthquake data consists of the date and time or origin, geographic coordinates (latitude and longitude), depth, magnitude, station used for the magnitude measurement, region, and additional magnitudes. A sample data format for an earthquake is presented below with the field names and data examples separated by :=.

Date:= 1/1/2004
Origin Time (UTC) (HH:MM:SS):=20:59:31.9
Latitude:= 8.310 S
Longitude:= 115.788 E
Depth:= 45
Magnitude:= 5.8
Station No. Used:= 119
Region:= Bali, Indonesia
Additional Magnitudes (Formula/Value/Station):= Mw/5.8/GS, Mw/5.8/HRV, mb/5.5/GS, Ms/5.4/GS

The earthquake data from 2000-2005 were converted to a modified XML earthquake advisory format and grouped by month resulting in 72 XML documents. To populate the data warehouses with a sufficient amount of data to show query access time differences between the methodologies, the data within the 72 documents (i.e., 6 years worth of data) was quadrupled to create over 100 years of data.

6.2 Implementation of the Data Warehouse Design

A fact table within the data warehouse consists of all of the data associated with an earthquake advisory, including the data for the advisory in which the event is reported as well as the event data. The following data items provided a complete earthquake advisory record or fact:

- Advisory title
- Advisory publish date
- Event date
- Event latitude, longitude, and depth
- Event region
- Event magnitudes (one or more)
- Magnitude source
- Magnitude formula
- Magnitude value

The design for each data warehouse as shown in Figures 1 and 2 includes a single fact table and dimension tables to hold the event advisory data. Time data is represented in a separate dimension table. The fact table contains foreign keys for the publish date and event date that refer to the time dimension table's primary key.

A separate dimension table is used to hold the publisher name. Using a separate dimension table avoids unnecessary duplication and leaves room to include advisories published by other organizations in the future. A separate dimension table is also used to hold the advisory title and link for the same reasons. The fact table contains foreign keys that link it to the publisher and advisory dimension tables.

Figure 1. Star schema data warehouse logical schema

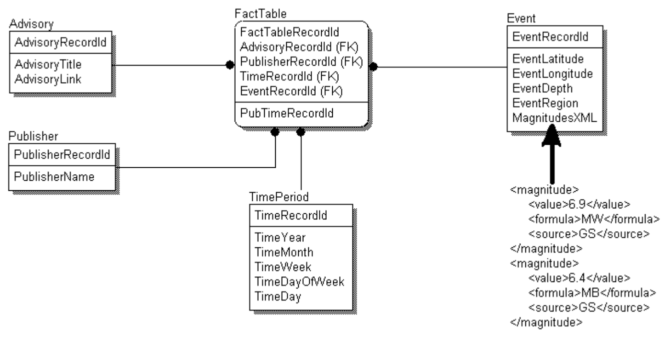
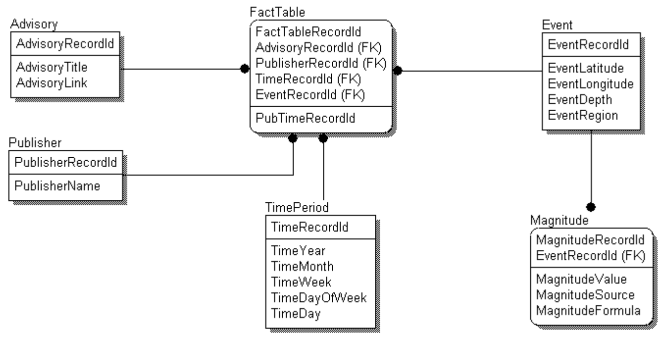


Figure 2. Snowflake schema data warehouse logical schema



The event region, latitude, longitude, and depth are represented in their own dimension table and the unique identifier included as a foreign key in the fact table. Magnitude data is stored differently for the two methods. The star schema method stores magnitude data in XML format for each event as presented in Figure 1. This means that the magnitude data is maintained in the original XML document format and stored as text within a table field called the Event table.

The snowflake schema method uses a separate, hierarchical table to store the multi-valued magnitude data. This means that each magnitude is maintained in a separate row with the source, formula, and value each contained within a column. In addition to the general structure of the snowflake schema's fact and dimension tables, Figure 2 shows the multi-valued XML data broken down into individual fields and records, and stored in a separate table. The table contains a foreign key reference to the primary key of the dimension table, which holds the parent element data.

6.3 Queries

Queries used to compare response times between the two methods are representative of the types of queries that might actually be applied to the earthquake advisory data in the real world. The queries are also designed to test the technique each method uses to store the multi-valued magnitude data. These queries, in a business language format, are:

1. Retrieve all earthquake event records
2. Retrieve a single earthquake event record for a given date and region
3. Retrieve all earthquake event records that had a magnitude >= 7
4. Retrieve earthquake event records that had a magnitude equal to the maximum magnitude across all records
5. Retrieve the average magnitude over all earthquake event records

The goal when translating these queries to SQL is to extract the information with a single query to the data warehouse, when possible. Multiple queries, however, are necessary in some cases. The SQL queries for the star schema method, where magnitude data is stored as XML for each event, involved using an extraction function applied to the XML text in order to retrieve each individual element's data value. Because there may be multiple magnitude source, magnitude formula, and magnitude value nodes for each event, these nodes must be accessed via a subscript. In addition, because the number of subscripts necessary to access the source, magnitude, and value nodes for each record varies, queries use the maximum number of subscripts and require additional processing to filter any null values returned. The SQL queries for the snowflake schema require an additional join of the Magnitude table.

7. RESULTS

The data tables for the two data warehouses were created using Oracle 10g [2] server with a dual Xeon processor. All queries were executed five times and the average of the five runs taken as the representative response time for that query.

In addition, indices were created in an attempt to maximize query response times. An index was created for each table's primary key and functional indexes were created for magnitude values within each data warehouse. The response times for Queries 1, 2, and 3 for the Star Schema and Snowflake Schema methods show less than an order of magnitude difference, while the Query 4 and 5 response times for the two methods show several orders of magnitude difference. The variations between the two sets of queries are a result of the differing query complexities caused by the manner in which the multi-valued XML data is stored and, therefore, must be accessed in the two methods. Queries 1, 2, and 3 involve either no or

Figure 3. Response times (milliseconds) for queries without calculations or aggregations

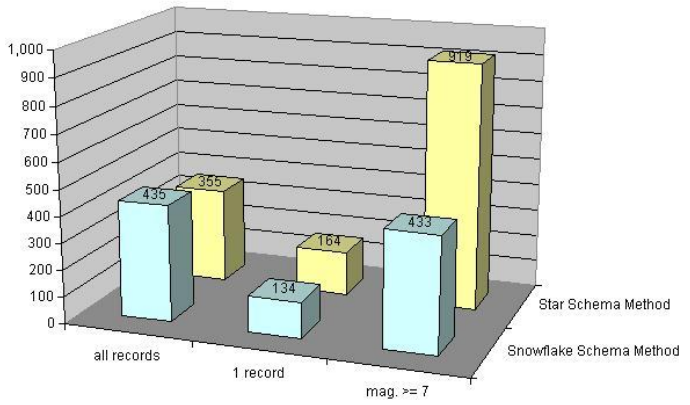


Figure 4. Response times (milliseconds) for queries with calculations or aggregations

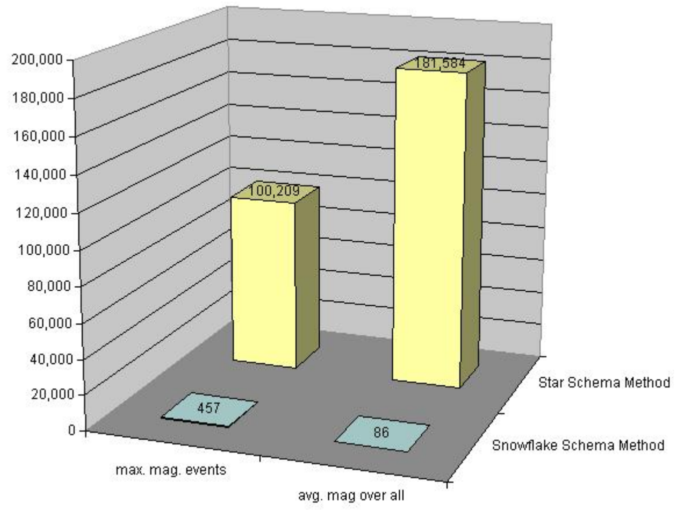
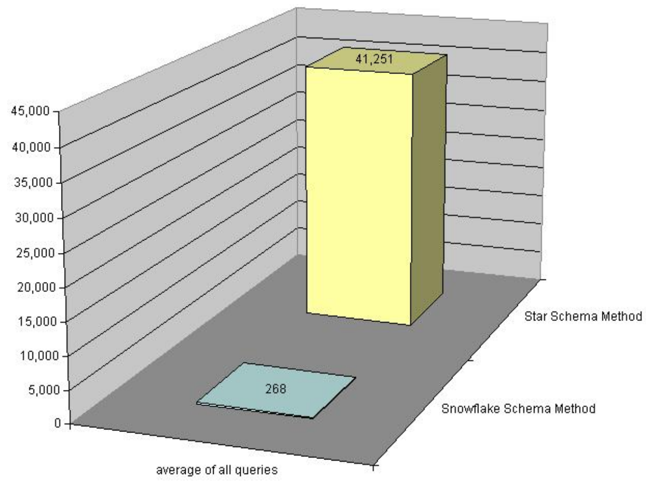


Figure 5. Response time average over all queries (milliseconds)



simple filtering, while Queries 4 and 5 involve calculations or aggregations of the multi-valued magnitude data, which is stored differently in the Star Schema and Snowflake Schema methods.

The response times (Figure 3) for Query 1 (retrieve all records) show that the Snowflake Schema is slightly slower than the Star Schema. Although the query for the Star Schema makes use of an XML extraction function, which is built into Oracle, to extract the magnitude data, the Snowflake Schema method actually takes slightly longer (435 vs. 355 ms for the Star Schema method). This is most likely due to the increased time it takes to join records, which is greater for the Snowflake Schema method because it has an extra table.

The response times for Query 2 (retrieve a single specific record) show that both the Star Schema and Snowflake Schema methods are comparably efficient at retrieving a single record (164 ms for the Star Schema method vs. 134 ms for the Snowflake Schema method). Even though the Snowflake Schema method requires the extra join for Query 2, the criteria for retrieving the single record translates into a much faster execution plan that filters for the search criteria before scans on the hierarchical table are performed.

The response times for Query 3 (retrieve all records that have a magnitude >= 7) show the Snowflake Schema method to be almost twice as fast as the Star Schema method (433 vs. 919 ms for the Star Schema method). The query for the Star Schema requires a comparison for each of the multi-valued XML elements, which adds extra filter criteria. The Snowflake Schema method, however, can take advantage of the relational database's built-in ability to join and filter.

Queries 4 and 5 (Figure 4), which involve aggregations of the multi-valued magnitude data, have response times that show differences between the Star Schema and Snowflake Schema methods that are several orders of magnitude in size.

The response times for Query 4 (retrieve all records with the maximum magnitude) show that the Snowflake Schema method is much faster than the Star Schema (457 vs. 100,209 ms for the Star Schema method). Response times for Query 5 (retrieve the average magnitude of all records) also show the Snowflake Schema method to be faster (86 vs. 181,584 ms for the Star Schema method).

Like Query 3, Queries 4 and 5 require access to and extraction of the multi-valued magnitude data. With the Star Schema method, Queries 4 and 5 must access the XML elements twice, once to perform a calculation or summation and a second time to perform an aggregation. Because of the need to extract the XML text elements in the Star Schema method, Queries 4 and 5 become cumbersome, while the Snowflake Schema method, again, takes advantage of the relational database's built-in ability to join and filter.

The response time averages for all 5 queries (Figure 5) show that the snowflake schema method demonstrates a response time average that is 153 times faster than that of the star schema method. These results demonstrate that in this case, where the XML documents contain multi-valued magnitude data, the decomposition of the magnitude data into a snowflake schema is advantageous over attempting to maintain the traditional star schema.

8. CONCLUSIONS

A comparison of the methods from a quantitative standpoint, based on the query response times, shows that the snowflake schema method is consistently faster, and markedly faster in most cases, than the star schema. The star schema method has some response times that are comparable to the snowflake schema method. However, queries that require calculations (e.g., sum) or aggregations (e.g., average) of data values stored as XML demonstrated much slower response times with the star schema.

A comparison of the two methods from a qualitative standpoint, based on the experiences gained from implementing the methods, also shows that the snowflake schema method is preferable over the star schema. The star schema method required the use of a subscript when accessing multi-valued elements within the XML data. In addition, the star schema method required additional filtering when accessing a variable number of multi-valued elements and post-query processing when performing aggregates. The snowflake schema method required only one query and did not require filtering or post-query processing.

The implementation in this research has demonstrated that when storing multi-valued XML data, the snowflake schema demonstrates faster query response times than the star schema and may, therefore, be preferred over the star schema in cases involving the storage of semi-structured XML data. Finally, the type of data and the access methods necessary to retrieve the data should be an important consideration when making decisions regarding the design of a data warehouse.

9. REFERENCES

[1] http://www.w3.org/XML/Core/#Publications
[2] http://www.oracle.com/index.html
[3] http://www.microsoft.com/
[4] http://www.ibm.com/us/
[5] Kanne, C. and Moerkotte, G. Efficient Storage of XML Data. Proceedings of ICDE (San Diego CA, March 2000), 198.
[6] Eisenberg, A. and Melton, J. SQL/XML Is Making Good Progress. SIGMOD Record 31, 2 (June 2002), 101-108.
[7] Florescu, D. and Kossman, D. Storing and Querying XML Data Using an RDBMS. Bulletin of the Technical Committee on Data Engineering 22, 3 (September 1999), 27-34.
[8] Kanne, C. and Moerkotte, G. Efficient Storage of XML Data. Proceedings of ICDE (San Diego CA, March 2000), 198.
[9] Tian, F., DeWitt, D., and Zhang, C. The Design and Performance Evaluation of Alternative XML Storage. SIGMOD Record 31, 1 (March 2002), 5-17.
[10] Shanmugasundaram, J., Krishnamurthy, R., and Tatarinov, I. A General Technique for Querying XML Documents Using a Relational Database System. SIGMOD Record 30, 3 (September 2001), 20-26.
[11] Shanmugasundaram, J., et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. Proceedings of the 25th VLDB Conference (September, 1999), 302.
[12] Xylem, L. A Dynamic Warehouse for XML Data of the Web. Bulletin of the Technical Committee on Data Engineering 24, 2 (June 2001), 40-47.
[13] Golfarelli, M., Rizzi, S. and Vrdoljak, B. Data Warehouse Design from XML Sources. Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP (November, 2001), 40-47.
[14] Graves, M. Designing XML Databases. Prentice Hall, Upper Saddle River, NJ, 2002.
[15] Connolly, T. and Begg, C. Data Systems: A Practical Approach to Design, Implementation, and Management. 2nd ed., Addison-Wesley, New York, 1999.
[16] Elmasri, R. and Navathe, S. Fundamentals of Database Systems. 3rd ed., Addison-Wesley, New York, 2000.
[17] Chaudhuri, S. and Dayal, U. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, 26, 1 (1997), 517-526.
[18] Seyed-Abbassi, B. Designing an Optimized Data Warehouse for Data Mining. Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (July, 2001), 214-219.
[19] Appelquist, D. XML and SQL: Developing Web Applications. Addison-Wesley, Boston, MA, 2002.

[20] Quin, L. Open Source XML Database Toolkit. Wiley Computer Publishing, New York, 2000.
[21] http://neic.usgs.gov/neis/epic/epic_global.html or http://eqint.cr.usgs.gov/neic/cgi-bin/epic/epic.cgi?SEARCHMETHOD=1&FILEFORMAT=4&SEARCHRANGE=HH&SYEAR=2000&SMONTH=&SDAY=&EYEAR=2005&EMONTH=&EDAY=&LMAG=1&UMAG=9.9&NDEP1=&NDEP2=&IO1=&IO2=&SLAT2=0.0&SLAT1=0.0&SLON2=0.0&SLON1=0.0&CLAT=0.0&CLON=0.0&CRAD=0&SUBMIT=Submit+Search