2008

Implementation of the NC-94 hybrid storage prototype on a binary version of CanStoreX

Niranjan Kumar
Iowa State University

Implementation of the NC-94 hybrid storage prototype on a binary version of CanStoreX

by

Niranjan Kumar

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Shashi K. Gadia, Major Professor
Leslie Miller
William J. Gutowski

Iowa State University

Ames, Iowa

2008

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The NC-94 dataset, that contains climate, soil and crop data for 30 years during 1971-2000 for all counties in the north central United States, is an important resource in the agricultural community. Analyzing the dataset would yield invaluable understanding for farmers, scientists, public, planners, and policy makers to improve crop practices and yields, undertake scientific studies, and developing policy.

In the parametric model and its query language ParaSQL, the concept of a dimension is built at the level of primitive values. A canonical storage for XML (CanStoreX) is a technology to store large XML documents, deemed to be in terabyte range, in a paginated form on the disk that is accessed easily and efficiently requiring very small amount of main memory. CanStoreX is used as a back-end for storing NC-94 data, hiding the heterogeneity in climate, crop, and soil data in order to allow the user a simple view of counties as objects where geographical and time dimensions are implicit and taken for granted.

This work has focused on loading the NC-94 database on the CanStoreX storage platform. The combination of existing parametric query constructs and an efficient storage structure will provide an important tool to researchers who wish to analyze the NC-94 dataset. The process of loading this database has also revealed important inconsistencies in the data, which we have tried to address and hence develop a more consistent view of the dataset. Previously only climate data was available and it was stored in an older version of CanStoreX where XML was stored in text form. The newer binary version of CanStoreX allows a readily available tree-like navigation in the paginated XML document. Addition of crop and soil data requires different internal representation in order to achieve a uniform view for users that is at par with the climate data. Further, the internals were conformed to use the version of CanStoreX where pages are stored in binary, rather than text form.

CHAPTER 1. INTRODUCTION

This chapter serves as an introduction to the various concepts and technologies which are part of this thesis. We begin by introducing the parametric data model, which forms the framework for some of the software applications encountered in this thesis. We also briefly introduce the structure of the eXtensible Markup Language (XML), which is the document encoding scheme used for our storage. The reader is then familiarized with the CanStoreX storage structure. We then look at the NC-94 agricultural dataset, which is the primary focus of this thesis. We conclude this chapter by examining prior work and the major contributions of this thesis. The organization of the remaining chapters is also detailed.

1.1. Parametric Data Model

Conventional data models do not capture space and time dimensions in data adequately, at the tuple level. As a result of this, the query of information is difficult. The parametric data model is an elegant approach to modeling real world objects. In this model, attribute values are functions over a parametric space that can have any number and types of dimensions [1]. With this in mind, we turn to data with various dimensions we are dealing with in this thesis.

- Temporal data – data that varies over time.
- Spatial data – data that varies with change in geographic location. When dealing with space on the surface of the earth, such data is also termed geographical.
- Spatiotemporal data – data that varies across time and space.

The parametric data model is well suited to store temporal, spatial or spatiotemporal data, as each can be modeled in terms of functions from the underlying parametric space.

Compared to interval-based models or relational models, a temporal version of the parametric model has been shown to have many advantages [2]. Further, the extension of parametric elements to model spatiotemporal data has been discussed and validated [3]. The suitability of the parametric model to store data from the NC-94 dataset is illustrated later in this chapter.

1.2. XML

XML or Extensible Markup Language is used to describe certain types of documents which in turn consist of *entities* [4]. These entities can be parsed to extract the data stored in them. XML documents follow a standard logical structure and are organized hierarchically through the use of *elements* which are delimited by using tags in the document. This is well suited to capture the dimensional structure of tuples in the parametric model [1]. XML offers many advantages over previous markup languages like HTML, because it provides customizable user defined data description options or meta-data tags, rather than standard predefined markup. The inherent tree structure provided by XML has also led to the development of several powerful Application Programming Interfaces (APIs) in a variety of programming languages. With the support of these APIs, XML data can be parsed efficiently, using parser API such as DOM [5] or SAX [6]. Since their resource usage vastly differs, the appropriate parser needs to be selected depending on the implementation [7]. DOM makes use of an in-memory tree model of the document to allow random access to any node in the structure. But the major drawback of this approach is that the entire document needs to be loaded in memory for processing to occur. SAX is an event-driven parser, which requires handlers to respond to the events it fires off as the document is processed linearly from beginning to end. This overcomes the disadvantages of the DOM parser by not requiring the entire document to be loaded in memory beforehand. But it it is inadequate where tree-based navigation is needed in an XML document. We now look at a particular storage technique that was developed to store and process large XML documents.

1.3. CanStoreX

The basic storage unit for any native XML database is an XML document [24]. CanStoreX is a native storage structure for XML, where XML documents are paginated across a disk, and each page or fragment of the original document is a complete XML document by itself. This implementation of canonical storage for XML was developed by Shihe Ma [8]. It been tested for large document sizes and has been shown to be an efficient method of storing XML documents for query processing. Here, as mentioned before, documents are stored as pages, with the requirement that each fragment of the document be a self-contained XML page. This

is made possible by using special nodes called c-nodes and f-nodes to link individual pages. By traversing these nodes and their subtrees, we can reconstruct an entire XML document paginated using this architecture. The original implementation by Ma did not scale up to more than 1 GB because of his textual page implementation which led to issues with memory allocation to Java objects. This was addressed by Daniel Patanroi, who used a binary representation of XML pages to improve efficiency in storage and access for larger documents up to 100 GB [9].

Several query engines and platforms have been developed to handle XML data in general. Coming to the CanStoreX implementation, XML query languages like XQuery [29] and Quilt [26], and query engine platforms like Kweelt [25] were extended and modified to work with it. More details relating to development on the CanStoreX platform is covered in section 1.5. Thus the efficacy of CanStoreX as a complete storage platform for XML documents has been validated.

1.4. NC-94 Agricultural Dataset

Analyzing survey data collected longitudinally over a significant time period will obviously benefit any type of practice, like agriculture, which is based on a large number of variable factors. This helps predict patterns and assist farmers in making sound decisions regarding their produce. Hence, with a focus on the effects of weather variation on agricultural practices, the North Central Regional Association of State Agricultural Experiment Station Directors (NCRA) decided to develop techniques and perform data collection to study and help support better crop management and reduce risk factors associated with this sector [11]. This was devised to benefit not only producers, but also provide information to enable formulation of policies affecting crop management.

Data collection and organization was primarily focused on the states in the north-central region of the United States that came under the purview of this association. Climate, crop and soil data were collected for these states. The result of these data collection and assimilation efforts was the NC-94 dataset. The exact parameters and time periods covered by the dataset are detailed in later sections. Several problems encountered during collection, led to the development of many innovative methods to compensate and extrapolate for missing data. It

is evident that this is an important dataset which can be used in various applications that can extract useful information to support a decision making process. The NCRA was not merely concerned with collecting this information, but also developing software and hardware applications to make use of this data in a meaningful manner. The NC094 committee has made many advances in this realm and several publications relating various projects by its participants are a testament to this fact.

The NC094 committee was active for 5 years (1999-2004) and was superseded by the current committee, NC1018. They will be overseeing the next iteration of this massive project (2004-2009), and have also renewed similar objectives while carrying on the work that the NC094 committee began [12].

Coming to the specific use of the NC-94 dataset in this thesis, we had sought to provide a platform where the three datasets available (Climate, Crop and Soil) could be loaded to facilitate efficient query processing and analysis within the framework of the parametric data model.

1.4.1. Organization of the NC-94 dataset

1.4.1.1. Geographic considerations

The data available was specifically designed to be granular at the geographical level of counties. In simpler terms, a county is the most fundamental level of classification, and various producers belonging to a particular county were contacted to get county-level information. The county identifiers are the limited spatial information provided by this dataset.

1.4.1.2. Time period considerations

The data covers a 30-year span from 1971 to 2000. The granularity assigned to time measurement differs vastly across the three datasets. For climate data, measurements were spanned across days, for crop data, values were recorded yearly, and for the soil data, there was no concept of time assigned to it. This can be reasonably justified by the claim that soil data is relatively time invariant with regards to the parameters that were being measured here.

1.4.1.3. Format of the data

The original dataset made available to us was organized as follows:

- Climate data – Comma-separated values file (.csv)
- Crop data – Microsoft Access 2003 file (.mdb)
- Soil data – Microsoft Access 2003 file (.mdb)

We had to make certain modifications and manipulation to the format of these datasets to make them suitable for storage on the CanStoreX platform. The nature of these changes are covered in Appendix .

1.4.2. Availability of GML data

Though strictly not part of the NC-94 dataset, access was also provided to the geographic boundaries of counties in the north-central region, in the format of a file specified in Geography Markup Language (GML). GML is an XML-based standard developed by the Open Geospatial Consortium to model geographic regions [13]. As the standard is XML-based, the CanStoreX storage technology is well suited for storing and accessing these maps. This is an interesting way of explicitly incorporating the space dimension in the NC-94 dataset where counties are represented only symbolically by their names or FIPS codes. It would allow us to; for example, determine if two counties are neighbors.

1.4.3. Using the parametric data model

As the NC-94 dataset involves space and time dimensions, the parametric framework is well suited to model it for storage, query, and analysis. In the parametric model each tuple would contain the entire spatio-temporal description of one county. For example, for a given county, the entire daily climatic information covering 30 years, together with the geographical information present in the GML maps will be available to a user in terms of a single tuple. In the traditional relational model the spatiotemporal information that is present in a tuple in our model would be structurally fragmented into 10,958 tuples, one for each day in the 30-year period and the geographical maps stored elsewhere. Such a representation is difficult to deal with. On the other hand, the user in the parametric model is not bogged down with the internals of storage structures and sees a county as a single object and deals with it as one

would in a natural language such as English. Moreover, the internal storage structures vary among climate, crop, and soil data. For example, in contrast to the climate that has 10,958 daily observations that require to be represented internally as one parametric tuple, the soil relation has only one observation that spans the whole 30-year period. Furthermore, even for each of the climate, crop, and soil relations, the representation and storage technology will continue to evolve. In the parametric model these details are absorbed by the system and the user is not expected to be conversant with the changing storage representations. Thus the applications based on such a model will have a long life.

For the NC-94 dataset, we devise the following storage artifacts.

- Climate- Spatiotemporal relation
- Crop – Spatiotemporal relation
- Soil – Spatial relation
- GML database – Spatial relation

However the user is only aware of climate, crop, and soil information that is organized by counties. From the user's point of view they are all spatiotemporal in nature and the questions to be asked of a specific county are in plain English. ParaSQL, which is an SQL-like language specifically designed for query of parametric data, is used. Just as SQL can be used to query for atomic values, ParaSQL can be used to query for counties, which represent atomic values here.

We will model tuples from these relations as containing parametric elements. Parametric elements are just subsets of the parametric space, and can be classified as spatial, temporal or spatiotemporal. According to the requirements of the parametric model, all parametric elements should be closed under the set theoretic operations of union, intersection and complementation. Since elements from these relations are also closed under these operations and can be categorized as either spatiotemporal or spatial elements, we can apply the parametric model here.

It is clear that the entire history and information about this real-world object, a "county", is preserved in one tuple, and hence it closely resembles how one would actually associate an object and data related to it – as a single entity. In our experiments we often restrict to counties in Iowa; however that can easily be extended to the whole north-central region.

1.5. Prior Work

The original implementation of the CanStoreX platform was done by Shihe Ma [8]. His version involved a plain text representation of the paginated XML. Parsing such a representation involved use of DOM utilities, which added the overhead of memory allocation to a large number of objects in the Java implementation. Pages had to be completely loaded in memory to allow document traversal. Daniel Patanroi [9] addressed this issue by using a binary page implementation. This version did not require dependence on DOM parser objects for page processing. Using a hierarchy of nodes, only the subtrees and root elements required for processing were loaded into main memory. This implementation is the most efficient version as of now.

The development of custom query processing modules based on existing platforms was then undertaken. Satyadev Nandakumar developed and implemented a parser for XQuery using the Kweelt platform [27]. Using conventional implementations of the DOM interface for parsing the documents would have meant loading entire documents on the fly when unnecessary. Hence Robert Stark developed a custom implementations that took advantage of caching in CanStoreX to only load elements as they are accessed [20]. Matt Swanson and Srikanth Krithivasan then implemented the XQuery evaluation engine on this version of CanStoreX [28].

1.6. Contributions

This work seeks to provide a unified and standard platform for the NC-94 dataset that can be used for advanced query processing using the parametric data model. The objectives we achieved were:

- Extending the existing storage to support all 3 datasets: crop, climate and soil. Previously only the climate dataset was represented.
- Revising and modifying the format of data in the NC-94 dataset, in light of inconsistencies discovered.
- Loading the datasets to the current stable binary version of CanStoreX.
- Upgrading the ParaSQL GUI and back-end system that supports it so that it can function with the binary version of CanStoreX.

This has led to the deployment of a coherent and consistent database to harness the power of ParaSQL over an efficient storage platform, CanStoreX.

The remaining chapters are organized as follows: Chapter 2 contains an overview of the organization of the NC-94 dataset and details on the manipulations we performed to maintain integrity and consistency. Chapter 3 details the loading of the datasets on the CanStoreX platform. Chapter 4 contains the upgrades performed on the ParaSQL GUI code. We discuss enhancements that were done to the existing CanStoreX code in Chapter 5. Chapter 6 contains conclusions and scope for future work.

CHAPTER 2. NC-94 DATABASE – ORGANIZATION AND MANAGEMENT

We were fortunate to have access to the entire NC-94 dataset since October 2007, since previously only the climate part of the dataset was available. The value of the NC-94 database to people and processes associated with the agricultural industry cannot be emphasized enough. The vast number of parameters measured alone serves to provide a wealth of information that can be used in a number of applications to model predictions, or extract relationship factors between climate change and crop yield. We now take a detailed look at how this database is organized. We then move on to certain manipulations we had to perform to keep the data consistent and suitable for loading to the CanStoreX platform.

2.1. Overview

This database can be divided into 3 distinct relations as briefly mentioned before: Climate, Crop and Soil. The spatial aspect in the climate, crop and soil relations is limited to just simple identifying numbers for each county and state- referred to as Federal Information Processing Standards (FIPS) codes [14, 15]. Each state is identified by a two digit numeric code and each county is identified by a three digit code. Combining these, we identify each county uniquely by a five digit code. Hence, the spatial granularity for all three datasets is a county. Thus this FIPS code is a unique identifier for counties common to all three relations. These relations cover 1051 counties in the 12 states that are in the North-Central Region (NCR) of the United States and contain climate, crop yield and soil data records spanning over 30 years.

Given the spatial and temporal expanse of this database, it is not unusual to note that there will be some missing data, either due to lack of resources for further investigation or an actual gap in a recording station’s log due to various reasons. The NC094 committee had addressed these issues by developing methods to extrapolate existing data and account for the missing records [12]. In spite of the same, we have noticed some inconsistencies through the course of our interaction with this database. We have strived to maintain the integrity and consistency of this database while performing certain changes that were required to address this.

We examine each of the 3 relations below. The detailed schema of each relation appears in Appendix A.

2.1.1. Climate relation

The climate relation is considered spatiotemporal because it consists of recorded parameters for each day in a year for each county in these states. The time granularity here is thus one day. The relation extends across 30 years (1971-200). This means that each of the 1051 counties has 10958 days associated with it, covering this span of 30 years (after accounting for leap years). Each day has particular climate parameters that were measured and recorded. As mentioned before, this dataset was originally available as a .CSV file.

2.1.2. Crop relation

The crop relation is also a spatiotemporal relation. It was formulated by collecting information from producers across various counties and records many parameters associated with each crop (see Appendix A). This file was provided in the Microsoft Access 2003 file format. The original complete relation spanning 32 years (1970-2001) is actually the result of a query on individual relations for each year (CCrop_1970 … CCrop_2001). Hence the granularity of time here is a year. One fortunate benefit of this relation was that the data available was not only for the 12 NCR states but also the entire continental United States. This led to a major design decision on our part which is detailed in the management section of this chapter. Further, apart from State and County codes that were part of the FIPS representation for a county, this relation also contained identifiers for districts. Groups of geographically adjacent counties are encapsulated in a district. This new spatial granularity provides a level of aggregation, which is used to store totals at the district level. To make the concept clearer, an illustration of the districts and counties in Iowa is in Figure 1, provided by National Agricultural and Statistics Services (NASS) [19]. The district numbers and boundaries are in boldface font.

Another important spatial aspect to this relation was the inclusion of Albers projection co-ordinates for the NCR counties in a separate relation ‘Countyinfo’. The Albers Equal Area Conic projection is a map projection method used to model geographic regions [16, 17]. Though we did not factor these co-ordinates into our particular implementation, due to the availability of a more detailed GML document, this information adds another valuable parameter to this database that can be used uniformly across the three datasets.

2.1.3. Soil relation

The soil relation is again indexed by county FIPS codes, but is different from the other relations in this database, because it is restricted to the spatial dimension, and does not have a granularity of time associated with it. We chose to accept this as a decision that the NC094 committee would have taken after careful consideration, and it can obviously be reasonably justified. The complication with this dataset is the large number of attributes in the relation - 105. Such a large number of attributes for a single county might seem questionable at first, due to its effect on tuple size, but on further examination, we realized that they are needed to accurately record the inherently complicated physical properties of soil.

Further, several statistical derivations were needed to accurately represent the quality and characteristics of the soil of an entire county in a single tuple. The values recorded are actually composite values calculated across all the soil types in a particular county [23]. The size of the actual soil dataset was very small because we have to deal with only one row of information per county, a total of 1051 tuples for the NCR states.

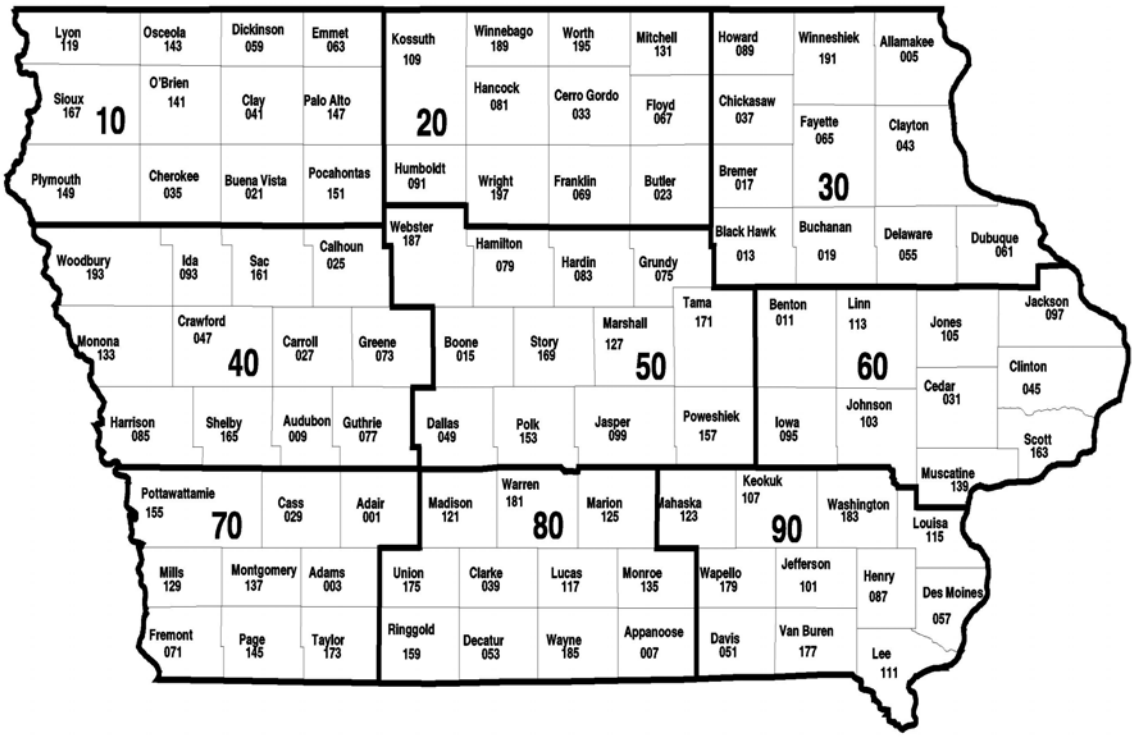

Figure 1. Boundaries of counties and districts in Iowa

Relation Name	Attributes	Description	Data Type
H2O	Region	5 digit FIPS code	Long Integer
	h20av	Parameter Measured	Double

Figure 2. Schema of a sub-relation in the Soil database

2.2. Database Management

We describe certain aspects of the datasets that we found did not match the quality and consistency of the surrounding data in the database. We then describe changes we effected to address these issues. The nature of these changes was not to modify existing data, because that would defeat the purpose of using this sterile database that had been created through years of data collection in the field. We sought to reorganize certain data, and in some cases, choose more optimal formats for storage, with the goal of a better final database that could be loaded on our platform, CanStoreX. The entire list of inconsistencies we noted is detailed in Appendix B. We cover the major findings in this section.

The climate relation was by far the most consistent and complete dataset. As briefly mentioned before, the climate relation had been available before, and was the focus of Noh's work on validating the parametric model for a spatiotemporal database [1]. His hybrid storage design for this dataset [18] is the one we continue to use and enhance. Below, we focus on the inconsistencies discovered and then go on to how we addressed them and also made modifications to suit our storage structure.

2.2.1. Inconsistencies across tables in soil dataset

The soil Access database provided to us had 3 relations (tables) in it. One was the main soil dataset, which was obviously the largest and most comprehensive, consisting of 102 attributes for 1051 counties. Apart from this, there were two auxiliary, smaller tables, with a relatively smaller number of attributes; containing parameters measured which were not present in the larger relation. One of these relations is shown in Figure 2.

We wanted a single, collated relation which could be loaded on to CanStoreX. At the same time, we did not want to lose any data that was already present, even if it was in a different table. Hence, we decided to merge the relations, to form a single soil dataset which would contain the data from the other smaller relations also.

But as noted in detail in Appendix B, there were inconsistencies across the county FIPS codes. Also, there was extraneous data from certain states not in the north-central region. To address issues of consistency across majority of the data, we decided to only collate the extra data for the counties that had a FIPS code match in the larger relation. With this decision, we compromised on information for a couple of counties (in NCR states) which were not present in the large dataset, but gained measurements for 1051 counties that did match.

2.2.2. Presence of NULL values in crop relation

As mentioned before, the final crop relation was the result of a query combining several tables in the Access database. In the final table, we noticed that several columns had NULLs recorded in them. On further examination, we also noticed that these NULL values did not correspond to a zero as we had first assumed. Due to this, the fields that had NULLs occurring in them were defined as Text data types in Access, in spite of the fact that a Double or Float would have sufficed.

This was a major hurdle to cross before we could load the dataset completely because we did not want to store these fields in a native format that did not correspond to their semantic meaning as a real world parameter. Also, looking at this as a storage concern, a single decision to store a String in place of a Float or Double would have enormous repercussions on the size of the stored database, considering the number of records in the crop relation just for Iowa – 25585 rows of data.

We were interested in determining if these NULLs were present due to a zeroing out of data for certain calculation purposes or they were genuinely due to a lack of information while conducting the field survey. Since both options would have diametrically opposite implications, further investigation was necessary. This issue was further complicated because the NULLs were present in some records for which data from other columns of the same record strongly suggested that a zero/NULL was not possible in that particular position.

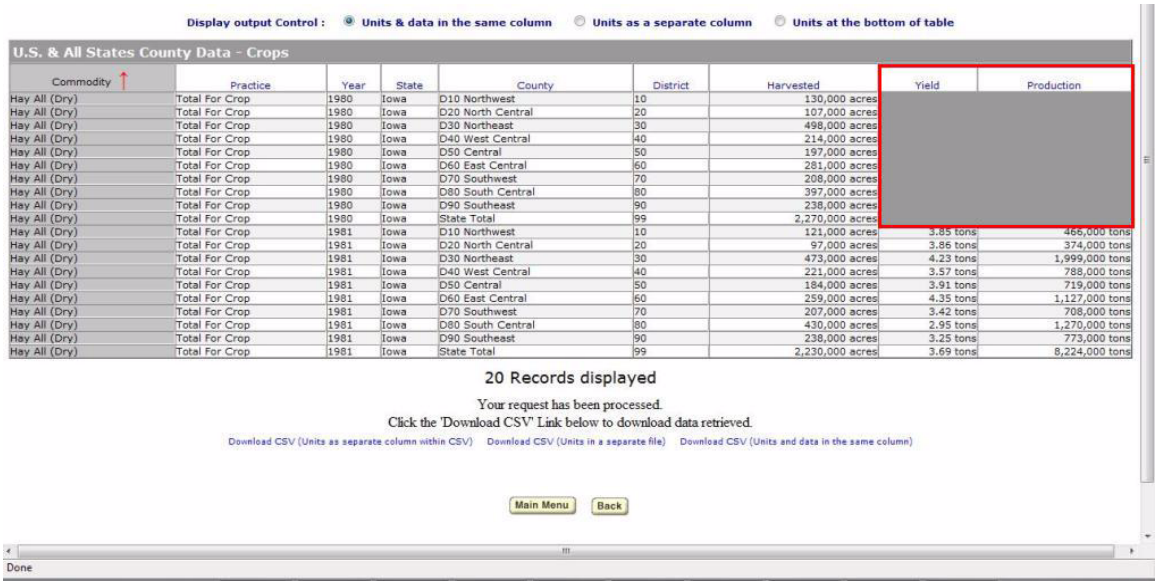

Figure 3. Screenshot from NASS displaying NULLs in Yield and Production

Illustrating this with a more concrete example, certain values in column ‘Planted’ would be NULL, but would have a corresponding value in the ‘Harvested’ field. It is obvious even to the casual observer that such a case is not possible unless the ‘Planted’ information was not accessible in this case. Thus, in any case a zero could be ruled out in this column unless explicitly noted.

Similarly, yield values were unusually presented as NULLs in many records, even when corresponding planted and harvested values were present. Yield is calculated by the National Agricultural and Statistics Services by using the Production, and Planted or Harvested parameters. In these cases, there was no production data itself to calculate yield. This is illustrated in Figure 3. This is a screenshot of a query run on the NASS website [19]. The NC-94 database consists of the same base dataset for these years. The area marked in red shows the NULL values in a query for a subset of Iowa crop data from 1980 and 1981.

We consulted with NASS staff on this issue, and it seems that for certain years they do not have survey data that covers some fields, and hence there are bound to be gaps in the data. Hence, we had to contend with these NULLs as just an anomaly in the data collection process and represent it as such.

Our approach to storing these values was based on considerations of optimal native storage configurations, and possible future availability of this missing data. Since the native format of

these fields was an Integer or Double/Float depending on its usage, we decided to store them using these formats instead of their Access definition – a String. To indicate that a discrepancy was currently present (lack of information), we decided to add a comment column to the database. This could be edited later if the data was updated or derived through extrapolation methods.

2.2.3. Common updates to address sub-optimal storage

One update performed was to import the climate dataset into Access, from its original CSV format. This was done to ensure a uniform format across the database that would ensure our loading module worked efficiently.

We also performed certain updates which were common to the three datasets to allow for an optimal storage configuration. This involved minor changes like rearranging fields in the root Access databases, to the introduction of new formats to better represent field values. The major changes are detailed below.

- Moving geographically relevant indexing data (FIPS codes, State and County names) to the first few fields in each relation. This translated to a more efficient tuplet (a single row of data from the tuple) for retrieval and presentation purposes.
- Certain fields that were assigned larger data types than required were ported to fit smaller more efficient data types. In particular, certain String value types were represented as Integers or Doubles in CanStoreX. This decision was taken to reduce the overall size of the final database after ensuring that no data would be lost through this process of truncation.
- The FIPS codes for representing a record, a five digit unique number was stored as four digits in cases where there was a leading zero for the two digit state number (States with codes between 01 – 09), because this field was stored as an integer in the Access database. As mentioned before, the crop relation actually had data for all states in the continental region of the United States. Though we did not load this data on to CanStoreX, we did want to ensure this data could be loaded and accessed accurately later if required. Hence to preserve the FIPS mandated requirement, we decided to store the FIPS code as a five character string, to allow the leading zero to be preserved. This was a major design change

to a field in the stored relation, but we feel it is justified in this case. We also updated the Access database through JDBC routines to reflect this new field and format.

Thus, the combined effect of these updates was a more intuitive, efficient database, which provided options for future expansion without compromising the consistency of existing data.

CHAPTER 3. INTEGRATING NC-94 WITH CANSTOREX

This chapter details the development of the loading module that we designed to populate the CanStoreX storage structure with the NC-94 database. Since Noh had designed the hybrid storage architecture [18] for storing the NC-94 dataset, combining conventional sequences of binary pages for relational data with an XML directory structure, we could not use the existing pagination routines in CanStoreX. Also, we could not retain all aspects of Noh's original design, because CanStoreX has different restrictions. We begin by examining some aspects of the hybrid storage structure we had to manipulate and then move on to the loading module and its details.

3.1. Background

The hybrid storage structure available consists of an XML based directory structure that serves as an index to the relational data. The structure is outlined in Figure 6. Thus, this format allows the storage of homogeneous (relational data), heterogeneous (XML data) and hybrid (XML and relational data) relations on a single platform. Previously, when Shihe Ma's pagination algorithms were being used, XML pages conformed to that particular format, and relational data was stored as a sequence of tuples on binary pages. With the implementation of the binary version of CanStoreX, we began storing XML data with the revised pagination algorithms and the revised page formats. But, the relational homogeneous data was still maintained in the previous formats, as those formats were still relevant.

3.2. XML Directory Structure

A system wide catalog (NC-94.xml) serves as a catalog of the 3 NC-94 datasets and the GML spatial data file. This catalog contains many user-defined tags that are used to describe characteristics of a given relation. Each tag is a type of *element* and consists of *attributes* which in turn have *values*.

3.2.1. NC-94 XML Catalog

A sample from the catalog is shown in Figure 4, describing the climate relation. For example, here, **RelationList** is a tag name (an *element*), and has an *attribute*, **number**, which

has a *value*, **3.** Using these values we can define the characteristics of a relation in terms of the fields it contains, the size and type of each field and other indexing information. This XML catalog is also paginated in CanStoreX, and hence can be used by any application that deals with querying this database.

This name of this catalog (in this case, *NC-94.xml*) is sufficient for the GUI application to extract information about each relation. It is parsed using a modification of the DOM interface after retrieving the root document element from its paginated location. The process of using iterators to parse and retrieve information from the document stored in CanStoreX is detailed in Chapter 4. Since using conventional DOM modules would have meant a significant increase in memory utilization because of the way DOM builds its in-memory representation of an XML document, CanStoreX uses implementations of the DOM interface tailored specifically to minimize memory use.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Database Name="NC-94" homogeniety="false">
  <RelationList number="3">
    <Relation NumberOfPgs="5086" StorageFragmentKey="FIPS" ctype="hybrid"
      name="climate" pageid="5091" SpatialGranularity="county"
      SpatialRepresentation="FIPS" TemporalGranularity="day"
      TemporalRepresentation="integer" rtype="spatiotemporal">
      <AttributeList number="9">
        <Attribute key="true" length="32" name="FIPS" pos="0" type="string" />
        <Attribute key="false" length="4" name="StFIPS" pos="1" type="integer" />
        <Attribute key="false" length="4" name="CoFIPS" partition="true" pos="2"
          type="integer" />
        <Attribute key="false" length="4" name="Year" pos="3" type="integer" />
        <Attribute key="false" length="4" name="Day" pos="4" type="integer" />
        <Attribute key="false" length="4" name="Radiation" pos="5" type="float" />
        <Attribute key="false" length="4" name="MaxTemp" pos="6" type="float" />
        <Attribute key="false" length="4" name="MinTemp" pos="7" type="float" />
        <Attribute key="false" length="4" name="Precipitation" pos="8" type="float" />
      </AttributeList>
      <SpatialRelation name="county_gml" />
    </Relation>
...
  </RelationList>
</Database>
```

Figure 4. Schema and characteristics of climate relation in NC-94 catalog

```
<Relation name="climate" homogeneity="true" TupleSize="64 bytes">
  <Tuple pageID="833">
    <KeyAttributes>
      <Attribute name="FIPS" value="19001" />
    </KeyAttributes>
    <ParametricElement domain="temporal">
      <Interval start="0" end="10958" />
    </ParametricElement>
    <Info totalPages="43" totalItems="10958" />
  </Tuple>
  <Tuple pageID="876">
    <KeyAttributes>
      <Attribute name="FIPS" value="19003" />
    </KeyAttributes>
    <ParametricElement domain="temporal">
      <Interval start="0" end="10958" />
    </ParametricElement>
    <Info totalPages="43" totalItems="10958" />
  </Tuple>
...
</Relation>
```

Figure 5. Snapshot of the XML catalog for the climate relation

An entry for each relation in this catalog contains a pointer to the subsequent second level XML index in the case of spatiotemporal relations, or a pointer to the first page of the actual relation in the case of the GML file. This catalog displays the capabilities of XML as an encoding scheme for meta-data. We are able to define, customize and update our descriptions for each attribute in a flexible and standardized environment.

Due to restrictions imposed by the existing implementation of our query application, ParaSQL GUI, we had to retain certain fields in this catalog. But, we also added certain attributes to the tags to allow for a more descriptive catalog. For example, **StorageFragmentKey** was added as an attribute to the **Relation** tag, with the value **FIPS**, for the climate and crop relations. For the climate and crop relations, tuples are organized by FIPS codes, i.e. counties with the same FIPS codes are stored together. Subsequent counties with different FIPS codes are stored on new pages as they are encountered. But, for the soil relation, since there is only one tuplet for each county, starting a new page for each different

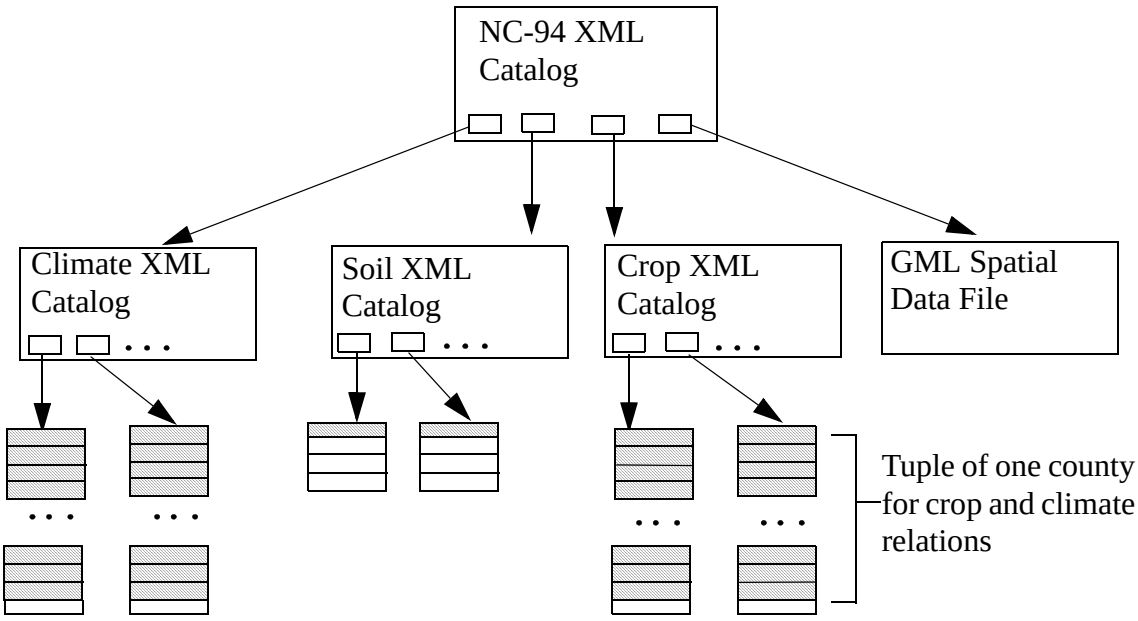

Figure 6. NC-94 hybrid storage structure

FIPS code would be a very inefficient way of storing this relation. Hence, this same attribute has a value **none**, for the soil relation.

3.2.2. Relation XML Catalog

We now look at the second-level relation catalog, which exists for the climate and crop relations. This catalog contains tuple pointers, i.e. the addresses of the pages where the actual tuples reside. This catalog also contains indexing information. It specifies which attributes are the key attributes, and also their actual values for each tuple. Hence, the key attribute value need not be retrieved from the physical storage location of the tuple, but can be read from this catalog itself, and thus save physical disk accesses.

This catalog is also paginated and stored at the location noted in the **pageid** attribute of this relation's entry in the NC-94 XML catalog. From Figure 5, we can see that this catalog details many characteristics about the stored relation that can be used for indexing purposes and to optimize query processing. Another noteworthy point about the XML relation catalogs is that they are generated at run time, when the actual relation is paginated on CanStoreX. In this method, the page numbers for each tuple are determined at loading time, just before the actual page is written and then updated in the catalog.

Apart from these XML catalogs, the CanStoreX platform has an XML based directory of its own, that serves as a catalog of the files stored on it. The structure of that catalog is very intuitive and is described later in this chapter. This presence of this catalog means that any application using files stored on CanStoreX can parse this XML document and retrieve the page number to begin using that particular file.

3.3. Data Types and Methods for Conversion

The previous loading module was no longer completely relevant because it was highly customized for loading the only relation available at that time, the climate relation. We set out to design a loading module that would be generic across the three datasets. One of our objectives was that it could be used reasonably efficiently to load all 3 datasets at a time. This way there would be a consistent view of the NC-94 database on the CanStoreX platform. Since the XML catalogs were generated at run time, while the relations were being paginated, the most efficient way to help maintain a uniform view of the database would be to load all 3 datasets at the same time.

The previous module also consisted of a set of utilities that were used to convert data types encountered while loading the dataset to the native byte storage format. These utilities formed the basis for our own conversion utilities. As mentioned before, there were many field data type definitions in the Access databases that did not match up to the actual data present in these fields. For example, the ‘Planted’ column in the crop relation was assigned a String type, even though it stored only an integer value. To address this, at the time of loading, we converted such fields to a more appropriate format. We defined new methods to store Double, Long and String values.

Addressing details of the String format, we looked to define a constant size byte representation of a String value. This is because we would be writing and reading data in terms of the fundamental unit of a tuplet. This implies that a tuplet will have to be of a fixed size to enable reading it by our tuplet iterators – detailed in the next chapter. Initially, in the unmodified datasets, the occurrences of String values were restricted to the crop dataset, and were used to specify the commodity being planted, the practice being measured and even the

name of the state and county. But, a design decision to represent the five digit FIPS codes as five character string values added a String valued column to each of the 3 relations.

On further examination, we determined that the maximum characters used up by any String value was always less than 32. Hence we decided to set the size of a byte-represented String as 32 characters. Finally, we used Java's native output stream writer to write the resulting bytes from a particular string value.

3.4. Loading Module

The loading module went through several revisions as and when we discovered inconsistencies in the dataset that had to be addressed, and hence led us to enhance our original code. We initially began by using Ma's pagination algorithms [8], and writing routines to load all 3 datasets. Since we were using an Access database for all three relations, we used Java's JDBC-ODBC bridge to read records from the datasets. At this point in time, though we have reorganized and updated the entire NC-94 dataset, we are loading only the records relevant to the state of Iowa, as we are currently interested in first providing an interface and useful query results at a local level. This system can be easily extended to support the entire NC-94 dataset as we have always kept scaling issues in mind while enhancing or updating our routines.

Once we were able to successfully load the three datasets by using Ma's pagination and Noh's storage managers, we shifted our focus to loading the datasets on CanStoreX, using the newer pagination algorithms. Any storage platform depends on a robust storage and buffer manager that can efficiently allocate and de-allocate storage space depending on the dynamic changes occurring due to applications writing in a non-sequential fashion. Usually, the buffer and storage manager are transparent to the end user who is only expected to provide the document they wish to store and its path of storage. After this, these two modules work in tandem to find available storage on the platform and update the allocation tables accordingly. In CanStoreX, a simple yet efficient page allocation system is used. A bitmap maintains the list of free pages and is updated every time a page is allocated or de-allocated. At the time of creation of the storage platform, the bitmap is initialized to the number of pages available in the storage structure.

The previous implementation of the loading module contained a more primitive storage and buffer manager, which allocated pages sequentially, given a page number to start with. In this module, the user had to provide as input a page number that would be used as the first page of the database on the storage platform. Unfortunately, this does not simulate an actual storage structure well, where the user merely provides a file for storage and is not concerned with how and where the file is actually stored. To model the behavior of the CanStoreX platform, we had to make use of its robust storage routines. Since the CanStoreX interface was originally written to paginate only XML documents, and not relational data, we had to port over its buffer and storage manager classes to be used in the loading module here.

Since the XML catalogs for each relation were being generated on-the-fly, as the relations were being paginated on CanStoreX, we use the original XML pagination routines from CanStoreX to store the XML documents after they are generated. The system wide NC-94 catalog is also updated with the locations of these relation catalogs. Now that the primitives used have been illustrated, the following sections detail the actual process of loading the datasets.

3.4.1. Input

Before the Access databases could be passed to the loading module, we needed to set up the initial version of the NC-94 XML catalog which would contain meta-data about each relation. This would allow a generic loading module like ours to determine the characteristics of each relation and call the appropriate routines. As detailed before, each relation has some subtle and major differences that prevent the design of a simple common loading routine that will work unchanged for all three relations. In spite of this, we directed our efforts toward designing one loader but with added complexity which would allow it to behave differently depending on the type and characteristics of a relation that is passed to it. This way, the loading process could occur with a single execution of the loading program. The 3 relations and the path of the catalog would be passed as parameters to the loading module.

A typical call to the loader would pass the following parameters to an instance of the loader class:

- Name of the relation in the NC-94 Catalog

- Path of the NC-94 catalog
- Name of the data source associated with this relation
- The type of database (as of now only Access databases are supported)

3.4.2. Parsing the NC-94 XML Catalog

As described earlier in this chapter, the NC-94 catalog consists of meta-data about each relation that we defined and assigned values, depending on the characteristics of each relation. We use a DOM parser to parse the catalog and based on the parameters provided to the instance of the loader, we look for the values of attributes that define how this relation should be paginated. For example, the climate and crop relations would have a value for the **TemporalGranularity** attribute as **day** and **year** respectively, while the soil relation would have a value of **constant**. We also decide on how the tuples should be paginated looking at the attribute **StorageFragmentKey**. We also use the results of parsing the catalog to ensure that all the attributes from the stored relation have been defined completely in a format suitable for storage. The parsing process also isolates the key attributes as defined by us. This is another advantage of using XML to define custom meta-data.

3.4.3. Writing Relational Data

Once the mode of writing had been determined, we executed a SELECT query to the underlying Access database using our Java connection object to retrieve records relevant to Iowa. This was easily accomplished using a simple filtering condition based on FIPS codes. Once we had a Java ResultSet object that was populated with the subset of records we were interested in, we began to write pages to the CanStoreX, tuple wise. This meant that we examined each row of the ResultSet to port each attribute present to the native byte representation depending on its high-level data type. Once all the attributes in a particular row were ported, we retrieved the next row and repeated this process.

The tuplet size, which was the sum of the lengths of all the attributes in a row, was fixed as explained before. The page size, previously 12 KB was changed to 16 KB to support routines from CanStoreX that interact with both the XML data and the relational data, namely the buffer and storage managers. We used an offset to determine the start positions of each attribute in the byte sequence that represented one tuplet- one row from the Access database.

The actual process of allocating a free page and writing to it was performed by the buffer and storage managers that are initialized when a storage device is “created” using the routines from existing CanStoreX code. Using the above process, we paginate the relational data as a sequence of bytes, and will be using the concept of fixed tuplet lengths to retrieve them for query, which will be detailed in the next chapter.

3.4.4. Writing XML Catalog Data

Coming back to the XML relation catalogs, we examine how they are also paginated on-the-fly as the relational data is written. Using the parameters passed to the loading module, we determine what needs to be written to the XML catalog. The tuplets are being written county wise (climate and crop relations), so we begin each tuple on a new page and this is used as an aid to determine when to add the tuple’s key attribute and temporal information to the relation’s XML catalog. Once all the relational data has been written, we stop writing to the catalog and end the document tags.

We paginate this catalog using the XML pagination routines from CanStoreX, and then obtain the location of this catalog, to update the NC-94 catalog. The location of the relation catalog is added to the **pageid** attribute of the relation’s meta-data. The address allows us to access this second-level catalog directly from the NC-94 catalog.

Paginating the GML spatial data file is uncomplicated. As mentioned before, the structure of this document is exactly like an XML document, on which it was modeled on. Hence we directly paginate this document, record its address and update the NC-94 catalog of the GML document’s location. One noteworthy point is that the CanStoreX pagination application automatically updates its public system catalog with the locations of each XML document paginated using its routines. We extended this to have the locations of the relational databases also recorded in the system catalog.

Thus, we were able to successfully load the datasets by defining suitable data types for the CanStoreX platform, a fixed tuplet size and using the robust CanStoreX buffer and storage manager to paginate the relational data.

CHAPTER 4. UPGRADING THE PARASQLGUI APPLICATION

Noh had created a flexible and efficient Graphical User Interface to take advantage of ParaSQL constructs to query the climate dataset available previously [1]. His application was based on a dataset that was paginated using Ma's algorithm. One of our primary objectives was to enable this application to work with data from the binary version of CanStoreX, since that was the most efficient version of this XML based storage platform. This was not a simple task to begin with, as the base ParaSQLGUI application code contained extensive references and hard coded customizations for the previous version of CanStoreX. With the use of the newer iterators from CanStoreX, we were able to phase out the previous page navigation routines, and maintain the same functions available previously. We begin by examining the architecture available to us with regards to navigating our paginated documents and extracting information stored in them. We then detail the conversion process and its results.

4.1. Iterators

In simple terms, an iterator is a basic data structure used to process chunks of data that have been grouped together in a larger storage unit. Essentially, the iterator takes advantage of the fact that each chunk of data has clear delimiters, which can separate it from other similar chunks. This delimiter can be a special character, or an identifiable sequence of flag characters, or in our case the fixed expected length of each segment (tuplet byte sequences), and special tags (XML document structure) that demarcate similar segments. Every iterator, regardless of the underlying data it operates on, needs to implement certain primitives that are used to 'iterate' over the data segments being read. These are:

- Open() – This routine needs to initialize the iterator and all the data structures it uses, and also check if there are actually any data segments that are present in the called storage unit or it is empty.
- HasNext() – This routine tests the storage structure to determine whether there are any more segments that can be read, or the end of legitimate data segments has been reached. It is similar to an end-of-file test used in common file processing routines
- GetNext() – This returns the next instance of the data segment being requested.

- Close() – This routine is used to close the iterator once processing has been completed. This ensures that memory and variables used by the iterator are de-allocated, and future calls to this iterator will not result in accessing these obsolete values.

Coming to the specific implementation of an iterator in code related to CanStoreX, we first examine the previous versions and their references in the ParaSQLGUI code. Noh was using iterators based on Ma's XML pagination. A *DiskDocument* was the root of the XML based storage. Navigation involved iterating through *DiskNodes* and *DiskNodeLists*, which were data structures defined by Ma to store the containers he used for his page structures. These data structures were based on extensions of the W3C DOM Node interface to interact with the paginated XML. But since CanStoreX had been upgraded to a binary version that used its own implementation of the DOM Node interface [20], we could not use these older iterators any more. The newer iterators took advantage of the binary implementation of CanStoreX to allow for lazy retrieval of DOM nodes. Previously, when a list of DOM child nodes was created from a parent, all the children would be loaded into memory, in a NodeList implementation, and hence consume precious memory, even if only one child needed to be accessed.

With the new implementation, DOM nodes are only loaded into memory as they are needed. Initially, the only check performed is to determine if there is actually a child node present for the given parent. This efficient implementation of iterating over DOM nodes is in turn used by more specialized iterators, like the *AttributeNodeIterator*, which is one of the main iterators we used in our effort to make the ParaSQLGUI application support CanStoreX paginated data. The only extra complexity with this model is that random access to a particular child node is not possible, and that particular node can only be reached by iterating over all siblings occurring before it, unless it happens to be the first child. This is not a drawback as such but just requires additional code to represent it. For each instance of this iterator that is created, we need a loop to process the all the children before the desired sibling. Without this additional code, any direct call to reference a child deeper than the first child in a parent node's sub-tree will result in an exception.

Further, the iterators in CanStoreX have also undergone many revisions to make the most efficient use of memory caching and optimal disk access. As mentioned before, default implementations of the DOM parser modules have been upgraded. A previous

ParaSQLGUI Structure	CanStoreX Replacement	Modules
DiskDocument	DiskIterator	Database Initialization, Relation Information
DiskNode	DOMNode	Relation Information, Tuplet Iterator, GML Relation Processing
NamedNodeMap	AttributeNodeIterator	Relation Information, Tuple Iterator, Attribute Processing
DiskNodeIterator	CSXNodeList	Relation Information

Figure 7. Replacements for ParaSQLGUI structures from CanStoreX

implementation of the *DOMNodeList* class was replaced by Stark [20], with the newer DOM primitives for CanStoreX, *CSXNodeList* and similarly the default *NamedNodeMap* implementation has been replaced with a *CSXNodeMap*.

Some of the previous iterators and their replacements from the newer CanStoreX implementation are detailed in Figure 7.

Looking at the iterators for the paginated relational data, the changes required were mostly in the initialization and setup routines. To begin iterating at the tuplet level, first the relation had to be loaded, and for this to occur, the paginated NC-94 catalog had to be deconstructed. This was done using the DOM node iterators mentioned above. Similarly, the XML relation catalog needed to be navigated for each tuple to be retrieved. The details of these transitions are explained in later sections. Another important application of the newer iterators was to navigate through the GML spatial data files. Here, the newer iterators had to be combined with routines from the Java Topology Suite, which is an API providing support for spatial function implementation [21].

4.2. Loading from CanStoreX

Previously, a *DiskDocument* was defined to be the root of the hybrid data file. In essence, the NC-94 XML catalog was always loaded at page 0 in the hybrid storage file, and the *DiskDocument* was hard coded to look at location 0 to retrieve this file. But in our case, the

NC-94 database is just one of the many files that could be on this platform. Hence we chose instead to look for this document in the system catalog of the CanStoreX structure. This was a more intuitive method of locating the file. We use a *DiskIterator* to begin the process of navigating the NC-94 XML document. At the time of loading the database into this application, only the root element of the NC-94 catalog is extracted and stored.

The buffer and storage manager from CanStoreX were also ported to this application because the older implementation was more closely related with the previous XML pagination method. Older packages that had modules to deal with the two types of pages, XML and relational were modified, since the XML page implementation currently in use had its own routines and iterators.

Since the buffer and storage managers used were updated, this directly affected another module dealing with initializing the database which was the information table in the GUI that displayed statistics on total disk access counts and query disk access counts. The previous implementation contained modules to increment access counts at the storage manager level, whenever a page was requested, either for the buffer or physical storage. To emulate the same behavior, we modified the CanStoreX routines to also maintain and update this statistical information while performing reading or writing operations, both at the buffer level and physical storage level. Other updates performed in the initialization stage involved modifying the global constants to recognize the newer 16 KB page size. Even though the page size was increased, this did not affect the relational page format, since the same number of bytes was allocated to header information.

4.3. Tuple Processing

Retrieving a tuple that matched the query criteria given by the user involves several complex steps of parsing, expression evaluation and repeated relation scans to filter matching tuples. A sample query has been illustrated in Figure 8 to show the various operators and expressions.

Since our modifications were restricted to the routines that handled the paginated datasets and XML catalogs, the existing parse routines were left unchanged. The expression tree generation modules did need modifications because they were hard coded to look for certain

```
select C.FIPS, C.Commodity, C.Yield
restricted to [1971,1973] + [1992,1995]
from USCrop C
where Population (C.FIPS) > 250000
```

Figure 8. Sample ParaSQL query on the crop relation

tokens which were unique to the climate relation. Once these routines were updated to ensure compatibility with the entire dataset, we examined the modules dealing with the stream processing of tuplets.

Tuplets are read from the underlying database on CanStoreX by using tuplet iterators, which are invoked on a particular relation by identifying tokens from the expression tree. A tuplet is aggregated by iterating over a byte sequence and using the attribute lengths defined in the NC-94 catalog to extract each attribute's value from that tuplet (row). This is where the fixed tuplet size plays a very important role. A fixed tuplet length is the only way we could use an iterator here to identify a legitimate data segment (in this case, a tuplet). Using incremental offsets, we can read an entire tuplet at a time. The structure of a tuplet with field sizes from the climate relation is given in Figure 9.

Attribute	FIPS	StFIPS	CoFIPS	Year	Day	Radiation	MaxTemp	MinTemp	Precip
Size in bytes	32	4	4	4	4	4	4	4	4
Example	19001	19	1	1971	1	6.8	6.72	- 7.02	0.0

Figure 9. Tuplet of climate relation with field sizes, as stored in CanStoreX

A tuplet is constructed and streamed into a string for display to the user. But before the string is displayed, the application determines which fields from the tuplet were requested by the user in the query and removes the extraneous fields. A tuple resulting from the query is buililt from many such tuplets and is displayed to the user on the screen. The location of the tuple is retrieved from the XML catalog of that particular relation. The routines that included code to retrieve this information from the relation catalogs had to be modified to use the newer *AttributeNodeIterator* and revised buffer and storage manager code. These modifications did

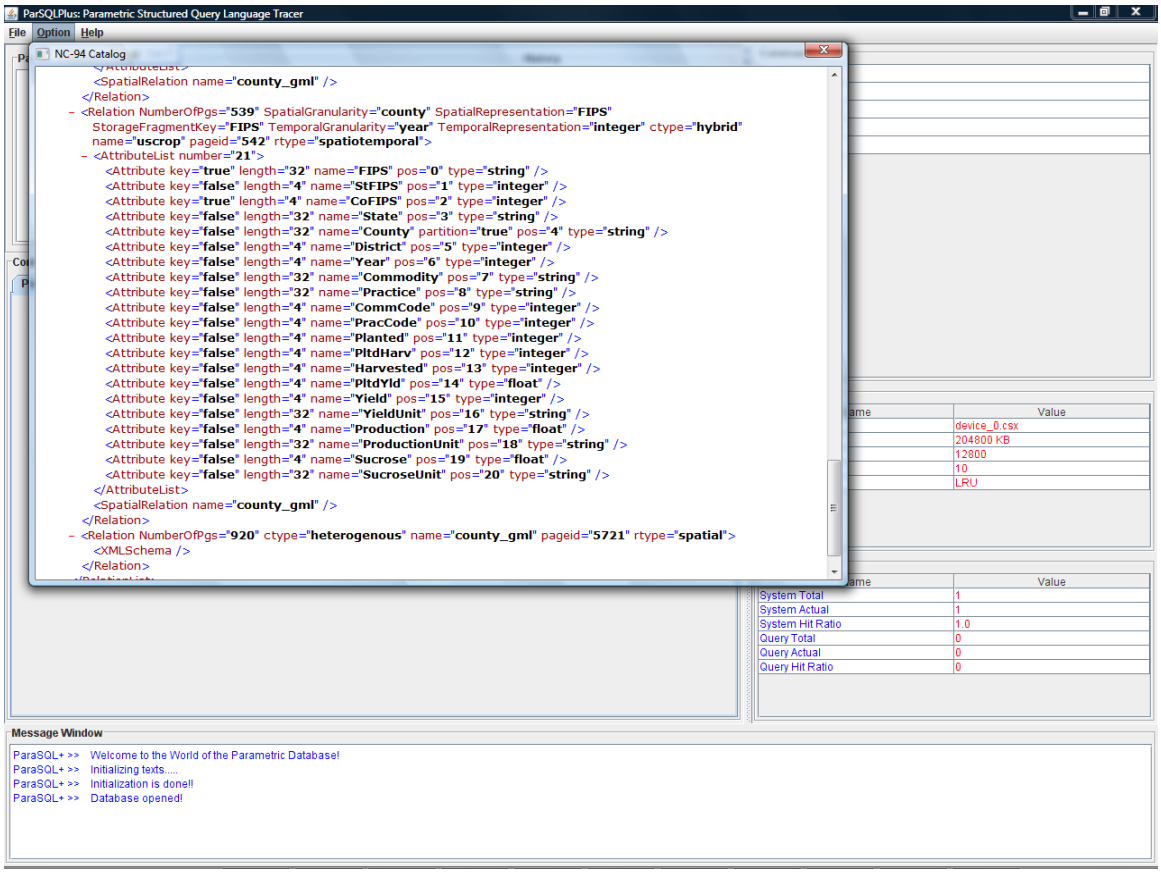

Figure 10. NC-94 Catalog displayed in the application

not change the underlying architecture of the tuple selection module, but only updated artifacts that were needed to access the pages on CanStoreX platform.

4.4. Changes to Temporal Condition Evaluation Code

The process of finding whether a given tuplet satisfied certain query criteria, namely, the temporal restriction criteria, was determined after a tuplet had been read from the relation and its string representation was constructed. The conditions specified in the RESTRICT clause of the query are parsed and stored in tokens in the expression tree. Then, the temporal attributes are extracted from the tuplet-string representation and evaluated. In the previous implementation, since only the climate dataset was available, the string representation of the tuplet was parsed using the embedded comma separators as delimiters and specific attributes were extracted using their position numbers in the string. But, with the availability of the crop relation, which was another spatiotemporal relation, the same position numbers could not be

used since the attributes and their positions were completely different from the climate relation. Thus to maintain the same string parsing method, certain conditions were implemented to make the parser aware that a different relation was being invoked.

4.5. Summary and Enhancements

The task of upgrading the ParaSQLGUI application to make it interact with data paginated in CanStoreX encompassed the following actions:

- Identifying all the routines that consumed XML data, and replacing older code with the iterators from CanStoreX.
- Using modified versions of the buffer and storage manager from CanStoreX to replace existing code.
- Modifying code that was customized to data from the climate relation.
- Updating global constants that were associated with the previous version of CanStoreX.
- With the completion of these tasks, certain enhancements were also performed to make more information available to a casual user of the system. The NC-94 catalog was made available for viewing so the user was aware of the characteristics and attributes of all the relations paginated on the platform.

Further, statistical reporting issues relating to query and disk access counts were also solved with the introduction of a transparent buffer and storage manager. A screen shot displaying the XML catalog from the ParaSQLGUI application is shown in Figure 10.

CHAPTER 5. CANSTOREX ENHANCEMENTS

This chapter deals with certain issues encountered while using the existing CanStoreX base code and the enhancements performed to address them.

5.1. Behavior of Attribute Iterator

The *AttributeNodeIterator* takes as input a DOM Node and sets up an iterator that is used to process the attributes of that node. This is done by calling the custom implementation of *CSXNodeList* and then loading attribute children of the parent node into memory as and when called. This is one of the main iterators we had utilized while upgrading the ParaSQLGUI code to work with CanStoreX. But, due to the nature of this module's programming, certain values were being retained between consecutive calls of the iterator. Even though a new instance of the iterator was being created, a call to retrieve attributes of the new node was returning exceptions due to a count variable not being synchronized with the new instance of the iterator.

5.1.1. Resolution of issue

A function to reset values of shared and common variables was included in *CSXNodeList* and called from the *open()* method of the *AttributeNodeIterator* class. This ensured that if the previous instance did not reset these values in its *close()* operation, the current instance could still create a new *CSXNodeList* with legal values of these variables.

5.1.2. Impact of modification on existing code

This new function impacts only all calls to *AttributeNodeIterator*. Other applications and implementations using *CSXNodeList* will not be affected, unless they explicitly invoke the reset function.

5.2. Usage of CSXNodeMap

The *AttributeNodeIterator* was previously defined to retrieve and store attributes using the DOM structure *NamedNodeMap*. As described before, by using this conventional structure

we were potentially wasting disk accesses in retrieving all the attribute children of a node, even if only a particular attribute node was going to be processed.

The conventional implementation of this DOM structure was replaced with the updated, memory efficient structure *CSXNodeMap* defined by Stark [20]. This was done to improve the performance of the *AttributeNodeIterator* when it was invoked on large attribute sets.

CHAPTER 6. CONCLUSION AND FUTURE WORK

This chapter contains some general conclusions about the outcomes of this work, and the scope for future work in this area.

6.1. Conclusions

The results of analyzing data from the NC-94 database are invaluable, beyond doubt. The NC-94 committee has demonstrated through the wealth of software and applications based on this database that research is far from finished in this area. With the flexibility and power of a storage platform like CanStoreX, we provide another avenue to effectively utilize the information in this database. We hope to be able to deploy this application (ParaSQLGUI interface to the NC-94 database, running on CanStoreX) on a large scale so that academic and industry experts in the agricultural realm could use this as a platform for their work.

6.2. Future Work

There are also still many ways to extend this application and further enrich its capabilities. One area would be to add more powerful constructs in the query clauses, like support for UNION and INTERSECT clauses. Further, a visualization module that displayed results graphically and dynamically responded to query results would be very useful to interpret the information coming out of this application.

The CanStoreX platform had been originally designed for pagination of XML documents, but has been shown to be suitable for even relational data stored as byte sequences in this work. Support for such data could be enhanced, with development of dedicated page formats and routines to process such data efficiently.

Certain solutions to address the inconsistencies in the NC-94 dataset could be revisited when more information is available about this data, and the datasets could be updated and loaded on this platform.

As of now the JTS suite has been used to interpret the GML spatial data and perform primitive operations on regions derived from that data. As more advanced tools emerge for

processing geospatial data, like GeoTools [22], they could be used to further enhance the number and complexity of operations on this data.

In conclusion, while we have provided a stable and flexible platform for query processing, there is tremendous scope to extend this platform and keep expanding it to support more features.

BIBLIOGRAPHY

[1] Noh, S.-Y. (2006). An XML-based implementation of the parametric data model for ad- hoc query of temporal and spatiotemporal data. Ph.D. thesis. Department of Computer Science. Iowa State University, 2006.

[2] Gadia, S. K. , Nair, S.S. (1993). Temporal Databases: A prelude to parametric data. In Temporal Databases: Theory, Design and Implementation, pages 28-66. Benjamin/ Cummings, 1993.

[3] Gadia, S. K. (1993). Parametric databases: seamless integration of spatial, temporal, belief and ordinary data. In *SIGMOD Record*, Volume 22, Issue 1, pages 15-20, 1993.

[4] Bray, T. , Paoli, J. , Sperberg-McQueen, C. M. , Maler, E. , Yergeau, F. (2006). Extensible Markup Language (XML) 1.0 (Fourth Edition) W3C recommendation, 16 August 2006.

[5] Le Hors, A. , Le Hegaret, P. , Wood, L. , Nicol, G. , Robie, J. , Champion, M. , Byrne, S. (2004). Document Object Model (DOM) level 3 core specification. Technical report, World Wide Web consortium (W3C) recommendation 07 April 2004.

[6] Megginson, D. (2001). SAX: A Simple API for XML. Technical Report, Megginson Technologies, http://www.saxproject.org/

[7] Guo, P. , Basu, J. , Scardina, M. , Karun, K. (2003). Parsing XML Efficiently, Oracle Magazine. September 2003.

[8] Ma, S. (2004). Implementation of a canonical native storage for XML. Master's Thesis. Department of Computer Science. Iowa State University, 2004.

[9] Patanroi, D. (2005). Binary page implementation of a canonical native storage for XML. Master's Thesis. Department of Computer Science. Iowa State University, 2005.

[10] Krithivasan, S. (2007). Implementation of a XQuery engine for large documents in CanstoreX. Master's Thesis. Department of Computer Science. Iowa State University, 2007.

[11] North Central Regional Association of State Agricultural Experiment Station Directors. Expected Outcomes. NC094: Impact of Climate and Soils on Crop Selection and Management. http://www.lgu.umd.edu/lgu_v2/pages/attachs/474_NC-94ExpectedOutcomes.html. September 2004.

[12] North Central Regional Association of State Agricultural Experiment Station Directors. NC1018: Impact of Climate and Soils on Crop Selection and Management (NC094 Renewal), http://www.nimss.umd.edu/homepages/outline.cfm?trackID=5094.

[13] Open Geospatial Consortium Inc. (2007). Open GIS Geography Markup Language (GML) Encoding Standard. Version: 3.2.1. August 2007.

[14] National Institute of Standards and Technology (1987). Federal Information Processing Standards Publications. 5-2: Codes for the Identification of the States, the District of Columbia and the Outlying Areas of the United States, and Associated Areas -- 87 May 28. May 1987.

[15] National Institute of Standards and Technology (1990). Federal Information Processing Standards Publications. 6-4: Counties and Equivalent Entities of the U.S., Its possessions, and Associated Areas -- 90 Aug 31. Modifications made, January 2005.

[16] Weisstein, E., W. Albers Equal-Area Conic Projection. MathWorld-- A Wolfram Web Resource. http://mathworld.wolfram.com/AlbersEqual-AreaConicProjection.html.

[17] United States Geological Survey. Map Projections. http://erg.usgs.gov/isb/pubs/MapProjections/projections.html. August 2006.

[18] Noh, S.-Y. (2006). Hybrid Storage Design for NC-94 Database Within the Parametric Data Model Framework. In *Workshop on Information Systems Information Technologies (ISIT 2006)*. In *Lecture Notes in Computer Science*, Volume 3981, pages 145-154, Berlin: Springer, 2006.

[19] United States Department of Agriculture. National Agricultural and Statistics Services. http://www.nass.usda.gov/

[20] Stark, R. , Gadia, S. K. (2006). Implementing a primitive version of DOM Interface for CanStoreX. Department of Computer Science. Iowa State University, 2006.

[21] Vivid Solutions. Java Topology Suite. http://www.vividsolutions.com/jts/jtshome.htm

[22] GeoTools: The Open Source Java GIS Toolkit. http://geotools.codehaus.org/

[23] Hollinger, S. , Illinois State Water Survey. NC-94 Files distributed on CD-Rom.

[24] Staken, K. (2001). Introduction to Native XML Databases. http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[25] Sahuguet, A. , Dupont, L. , Nguyen, T-L. . Kweelt. http://kweelt.sourceforge.net/

[26] Chamberlin, D. , Robie, J. , Florescu, D. (2000). Quilt: An XML query language for heterogeneous data sources. In *Proceedings of WebDB 2000 Conference*. In *Lecture Notes in Computer Science,* Springer-Verlag, 2000.

[27] Nandakumar, S. , Gadia, S. K. (2005). Implementing a parser for the XQuery grammar on Kweelt platform. Department of Computer Science. Iowa State University, 2005.

[28] Krithivasan, S. , Swanson, M. , Gadia, S. K. (2006). Building a XQuery application for CanStoreX on the Kweelt platform. Department of Computer Science. Iowa State University, 2006.

[29] Boag, S. , Chamberlin, D. , Fernandez, M. F. , Florescu, D. , Robie, J. , Simeon, J. (2007). XQuery 1.0: An XML query language. Technical Report, World Wide Web Consortium, 2007. W3C recommendation 23 January 2007.

APPENDIX A.

The schema of each of the three relations- Climate, Crop and Soil is presented here. The schema for the soil and climate relations was originally published as a supplement to accompany a distribution of the NC-94 data files [23]. These schemas also reflect the updates we had performed on the database, to add new fields and reformat exisitng field data-types.

The schema for the climate relation is presented in Figure 11. Each of the 1051 counties have 10958 tuples associated with them, covering every day of the year from 1971-2000.

The attributes in the crop relation are detailed in Figure 12. This relation spans 32 years (1970-2001). There are 21 attributes for each tuple, and the number of tuples per county varies, depending on the years for which data is available. But, for a given year, there will be only one tuple for a given commodity and practice type.

The attributes for the soil relation can be classified into two categories, those dependent on the layer of the soil, and those independent of soil layer. The layer independent parameters are shown in Figure 13 (a). The remaining parameters dependent on the layer of the soil are presented in Figure 13 (c). (Further, we have presented only those attributes which had a clear definition associated with them).There are a total of 9 soil layers. The top and bottom layer depths from the surface layer (in cm) are shown in Figure 13 (b).

Attribute	Description	Format
FIPS	Five digit unique county identifier	String
StFIPS	State identifier	Integer
CoFIPS	County identifier	Integer
Year	Year (1971-2000)	Integer
Day	Day of the year (1-366)	Integer
Radiation	Maximum solar radiation (MJ m^{-2})	Float
MaxTemp	Maximum temperature recorded for the day (deg C)	Float
MinTemp	Minimum temperature recorded for the day (deg C)	Float
Precipitation	Total precipitation received for the day (mm)	Float

Figure 11. Schema of Climate relation

Attribute	Description	Format
FIPS	Five digit unique county identifier	String
StFIPS	State identifier	Integer
CoFIPS	County identifier	Integer
State	Two character state abbreviation	String
County	County name	String
District	District number	Integer
Year	Year (1970-2001)	Integer
Commodity	Commodity/crop name	String
Practice	Cropping Practice measured	String
CommCode	Commodity Code	Integer
PracCode	Practice Code	Integer
Planted	Total Acreage Planted (Acres)	Integer
PltdHarv	Acreage Planted – Harvested	Integer
Harvested	Total Acreage Harvested (Acres)	Integer
PltdYld	Acreage Planted – Yield	Integer
Yield	Yield	Float
YieldUnit	Unit used to measure yield (BU/CWT/LBS/TON)	String
Production	Production calculated from yield and harvested/planted	Float
ProductionUnit	Unit used to measure production (BAL/BU/CWT/LBS/TON)	String
Sucrose	Sucrose measurement	Float
SucroseUnit	Unit used to measure sucrose (PCT)	Float

Figure 12. Schema of Crop relation

Attribute	Description	Format
FIPS	Five digit unique county identifier	String
StFIPS	State identifier	Integer
CoFIPS	County identifier	Integer
AcresT	Total land in county (Acres)	Float
Arable	Total arable land in county (Acres)	Float
PctArable	Percentage of land area arable	Float
PctSlopeT	Mean slope of total land in county	Float
PctSlopeA	Mean slope of arable land in county	Float
Drainage	Mean drainage classification[a] of county	Integer
DepthH2O	Depth to water table during wettest part of year (cm)	Float
DepthBed	Depth to bed rock (cm)	Float
MaxRoot	Maximum root depth (cm)	Float
PAV	Plant available water in top 2 meters of soil (cm)	Float
OMkgm-2	Organic matter mass in top 2 meters of soil (kg m^{-2})	Float

a. Drainage classification codes: 1 = Excessively well drained; 2 = Somewhat excessively well drained; 3 = Well drained; 4 = Moderately well drained; 5 = Somewhat poorly drained; 6 = Poorly drained; 7 = Very poorly drained.

(a) Schema of layer-independent soil parameters

Layer	Top	Bottom
1	0	10
2	10	25
3	25	50
4	50	75
5	75	100
6	100	125
7	125	150
8	150	175
9	175	200

(b) Layers of soil and distances from the surface in centimeters

Attribute	Description	Format
Sand	Percent sand in layer	Float
Silt	Percent silt in layer	Float
Clay	Percent clay in layer	Float
Liquid	Water content at liquid limit (cm)	Float
Plastic	Water content at plastic limit (cm)	Float
AvailH20	Plant available water content (cm)	Float
BulkDen	Bulk density of layer (Mg m^{-3})	Float
Omatter	Percent organic matter content	Float
Perm	Permeability (cm/hr)	Float

(c) Schema of layer-dependent soil parameters, all attributes are included for all layers

Figure 13. Layer-independent and layer-dependent soil data

APPENDIX B.

This section documents all the inconsistencies we discovered while preparing to load the NC-94 database on the CanStoreX platform. It also covers the updates we performed to maintain a consistent dataset.

Soil:

Original file: soils03.mdb

Inconsistencies noted:

- Tables *soilwaterold* and *H2o* have three attributes *H20av* and *SumofAvailH2O, AvgOfSand* which are not present in *ncrsoils-new*. Table *soilwaterold* has 119 of these records for state Kentucky (FIPS state code = 21), which are not present in larger table *ncrsoils-new*. It also has data for FIPS 29186 (State=MO, county=Ste. Genevieve County) which was changed to FIPS 29193. *ncrsoils-new* does not have data for 29186 or 29193. There is also data for FIPS 29510 (State=MO, county=St.Louis(city)), which was collapsed into 29189, but 29186 and 29510 are only found in *soilwaterold* and not in *ncrsoils-new*. Thus there are a total of 121 records which are not found in *ncrsoils-new*.
- The above noted inconsistencies are also present in the *H2o* table.
- Multiple values for drainage, bedrock depth and root depth for each county are recorded.

Revisions:

- *soilwaterold* and *H2o* have been collated into *ncrsoils-new* with the removal of 121 inconsistent records, since *ncrsoils-new* was the relation with largest amount of data. Hence a total of 3 columns have been added to *ncrsoils-new- H20av* from *H2o*, and *SumofAvailH2O, AvgOfSand* from *soilwaterold*.
- Attribute *Region* of *ncrsoils-new* was changed to ‘FIPS’ to more accurately represent the nature of its value. Table *ncrsoils-new* was renamed to *ncrsoils*.
- Attribute *NC94SoilData_County* was removed because another attribute *soils_County* was already present which had the same FIPS values. *soils_County* was renamed to *CoFIPS*.
- *FIPS, StFIPS* and *CoFIPS* were inserted at the front of the access relation so that the geographic data was accessible first, before soil data.
- Attributes that did not require a Long Integer storage value, will be ported to fit the regular Integer data type.

Crop:

Original file: NCR03crop.mdb

Inconsistencies noted:

- *Yield* has been defined as an Integer, so decimal values not recorded, and values >0 but <1 are showing up as 0.
- 1746 records of Iowa crop data where *Yield* or *Production* is NULL. 9908 records where *Planted* is NULL.

Revisions:

- Geographical data was moved to the front of the relation. The relation *uscrop* now begins with *FIPS, StFips, State, CoFips, County*, *District* and *Year.*
- Attributes that did not require a Long Integer storage value, will be ported to fit the regular Integer data type. These are StFips and CoFips.
- *Yield* was calculated again using *Production* and *Harvested* values, and stored as a float value.

ACKNOWLEDGEMENTS

I would like to express my gratitude to a lot of people who have helped me complete my research and whose support was invaluable during the composition of this work. First of all, I would like to thank my major professor, Dr. Shashi Gadia, without whose guidance this work would not have been possible. His patience and lucid explanations of concepts involved in my research are greatly appreciated. My discussions with him have led to many of the elegant solutions to the problems we faced in our implementation. I would also like to thank Dr. Leslie Miller and Dr. William Gutowski for their willingness to serve on my Program of Study committee. Thanks are also due to Daryl Herzmann, from the Iowa Environmental Mesonet, for providing us with a copy of the complete NC-94 database and his valuable input.

I am also grateful to all those people who I have met and befriended over the duration of my stay at Iowa State University. They have added great value to my life here and will be remembered for many years to come. My lab-mates Jia and Xinyuan also deserve special mention for their timely help and the fruitful discussions we have had. Finally, I would like to thank my parents and my brother, who have always been constant sources of inspiration and motivation at every stage in my life.