|  |
| --- |
| **XML Query Language** |
| Querying XML data using SQL style mechanics. |
|  |
| This document contains the proposal for creating a new XML Query Language for Advanced Database, Comp 511, PennState Harrisburg. It outlines introduction to the task at hand, plan of attack, relevance toward advance database concepts, and lastly any references needed to ensure a quality deliverable. |
|  |
| **Allen Brubaker** |
| **9/15/2009** |
|  |

Contents

[Overview 3](#_Toc240963422)

[Detail of the Problem 4](#_Toc240963423)

[Plan of Attack 5](#_Toc240963424)

[Background 5](#_Toc240963425)

[Major Design Points 5](#_Toc240963426)

[Prerequisites for Design 5](#_Toc240963427)

[Pitfalls and Solutions 6](#_Toc240963428)

[Scenarios for Adequate Completion 6](#_Toc240963429)

[Milestones 7](#_Toc240963430)

[Resources 8](#_Toc240963431)

# Overview

There exists a disconnect between our current knowledge employing databases to not only simply store data but to access data in a manner that is relevant to the end users of the data, and between our true knowledge of the underpinnings of such databases and how, when one enters a properly written query in a predetermined language, data is both queried and returned. Being in an advanced database class, this gives us the perfect opportunity to tackle this inherent disconnect between knowledge of how to use a database software such as SQL Server and then the more subtle yet profound knowledge of what happens underneath.

In order to properly study such concepts, instead of directly studying SQL Server, it is our desire to write a new XML Query Language such that we would have hands on experience with handling all the underlying data storage of a database, which in this case would be XML, creating a query language that would allow one to search and query the underlying database efficiently, intuitively, and precisely, and finally to create a windows program with an accessible yet friendly user interface to act as the presenter and handler of the language, which in turn handles the underlying database. Creating a useful design and concept to not only manage data that should be allowed to be boundless in size, as well as creating a language to efficiently gather and present the data to end users, with a program for end users to test, introduces some issues. These issues consist of how to store data intuitively in order to then query the data, the syntax of the language that should be utilized, and the programming language that should be used to implement the project.

To face these issues we had to decide upon a useful programming environment to create, manage, and test the entire project, and lastly the style of the querying language. We decided it best to emulate the SQL query language style as closely as possible to give us both concrete goals for the different abilities we wanted our querying language to have (these might include the ability to sort, join, select, filter). Moreover, we chose to emulate a SQL querying style that users may need only minimal time in order to get acquainted with the new XML Language. Therefore, when they are presented with the final project, they can begin testing the various abilities of our query language rather than spend time having to learn the language.

As for the programming environment chosen, we decided to use a programming environment that we were most familiar with, this being the venerable .Net and more specifically C# with its many invaluable libraries, of which we will be making ample use not only to create the GUI with speed, but to ease the many parsing issues that will present itself when trying to query XML. To some, parsing may be thought of as the major component of our project, but to us we feel that allowing .Net to handle the parsing will allow us to focus on the querying mechanics for which this project was duly designed. This will give us more time to tackle and emulate more of the querying concepts and less of the drudgery of xml parsing. What follows is the breadth of our proposal including a detail of the problem to be solved including how this project is related to database concepts, strategic and tactical plans of attack, and lastly any resources that will be used.

# Detail of the Problem

Briefly introduced in the overview section of this proposal, the problem we are tackling with this project of creating an XML Query Language is the large disconnect between our understanding of how a database, such as SQL Server, takes arguments in the form of a language style understood by English speakers (such as SELECT, FROM, WHERE), then efficiently queries an underlying storage unit to return data in a format desirable to the user. The presented data could be sorted, grouped, joined across multiple data units or tables, filtered through various constraints, and although the underlying database is stored in static many times useless forms, when it is queried and joined across multiple tables with filters imposed on the query, the data that is returned is deeply meaningful and thoroughly enlightening for users as they ascertain in far greater ways perhaps subtle trends that their products have followed amongst their customers or even as they maintain employee payroll information with the added nicety of being able to track with accuracy and confidence the history of pay raises.

It is obvious that being able to store data in a database and query the data in a form meaningful to the users creates the entire beauty of the concept of a database itself. Though one can study the many esoteric database concepts of data mining, concurrent transactions, query optimizations and the like; understanding the rudimentary concepts of a database, which consists entirely of storing data and presenting data in as flexible a way as possible, is of utmost precedence. This project desires to study just that, and we intend to accomplish the goals of understanding how a database works by creating and managing our own data storage facility as XML files, designing a language easily understandable to end users, and creating the functions of the language necessary to perform adequate querying across the entire breadth of the data, which will most assuredly consist of multiple tables. We plan to emulate the SQL querying language as closely as possible, while managing ourselves the underlying XML storage facility.

Past attempts at creating an XML Query Language can be found in a project developed by Eric Liskow entitled *A New XML Query Language* (cs.hbg.psu.edu/comp519). Though his project was mainly a success, the majority of his efforts, and indeed many sleepless nights, were spent designing a means to effectively parse the xml data in response to the queries issued by the query language. Because we chose to utilize .Net and C# with its many useful libraries that ease creating and managing XML data sources, less work will be done in handling the less meaningful task of parsing, and more time can be devoted toward creating a truly intuitive querying language employing as many language mechanics as possible in attempts to imitate SQL. These include being able to select, sort, filter data, group data, join data with multiple styles of joins, and the like. The detailed plans of attack including the milestones inherent in the project, a schedule of the aforementioned, and feasibility of the project as it pertains to current background knowledge can be found in the subsequent section.

# Plan of Attack

Any successful project and likewise proposal should fully assess the design points of the project at hand. Furthermore, it should attempt to reduce the overarching goal of the project into concrete manageable steps adequate to complete the project within the given time constraints. What follows is our plan of attack--both a strategic plan of attack detailing many of the issues needed to be addressed in the implementation of the project, and a tactical plan of attack with milestones, schedules for when each should be completed, and any possibility for failures that might occur, along with any corresponding fallback positions.

## Background

Having a background not only in computer science but also in C, C++, C#, .Net, and WPF along with real world experience handling various large databases for the state of Pennsylvania uniquely qualifies me in being able tackle the problem at hand. Though I do not have specific experience per se with XML data structures, becoming well versed in the libraries required in the .Net to handle XML data as well as being able to design a way to implement and test via a user interface will not be an issue—instead, it is a required part of the project and one of the faculties I am excited to add to my repertoire of .Net programming experiences.

## Major Design Points

Below is listed the span of design points needed to be fully addressed for the project to be successfully completed.

* Create an intuitive GUI where the user can both enter a query and obtain the results of the query, with data stored in a table. The GUI will be created with Windows Presentation Foundation (WPF).
* Capture and parse the query string.
* Design objects and methods in .Net to uniquely handle one of the many requests a query string might issue. These may be selecting, sorting, ordering, filtering, grouping, and joining.
* Create objects and methods able to interface with the underlying XML data storage—whether it be creating new XML data tables or querying the ones that already in existence.
* Collect the results in the form of an anonymous object with fields corresponding to the individual columns found in the result of the xml query.
* Display the results to the user by attaching the returned data back to the GUI

## Prerequisites for Design

Each the above steps do not simply require that previous steps have been completed. On the contrary, various other prerequisites are required before a step can be properly addressed and completed. These prerequisites, including learning about the utilities available to me as presented by .Net, are listed below.

* Obtain the tools and references needed in order to embark on development. These can be found in the section entitled *Resources*.
* Research using the given references and the internet, more specifically the Microsoft Developer Network (MSDN), to find any libraries that may be useful in helping to create and parse XML data structures.
* Preliminarily brainstorm on paper the structure of the program, including the names of objects, the methods that should go in each object, whether they should be in one large project or in multiple projects of the solution. Decide if N-Tier architecture with best practices suit the application’s needs best, or if focus should be spent more on usability than on best practices.
* Decide upon the structure and layout of the XML storage facility—whether to create separate XML files for each table in the database or use one large file that contains in outermost tags the table names.

## Pitfalls and Solutions

Along the way potential pitfalls might surface. Forecasting potential failures along with designing specific solutions should these arise is vital to the success of the project. Below is listed any potential pitfalls that might arise as well as ways to handle them in speedy manner so as to abate and eliminate the propagating side effects that may threaten the health of the project.

* Procrastinating due to external stressors.
  + Solution:Well plan a schedule for when major milestones should be completed. Then stick to this schedule at all costs.
* Unable to fully comprehend the references that may arise because of a lack of experience with .Net.
  + Solution: Enlist the help of experienced friends that would explain the material more thoroughly.
* Not enough time to complete everything.
  + Solution:Be sure to tackle the major design points first, and if need be, exclude some of the features of the querying language (such as implement selecting and inserting, but perhaps exclude grouping of data). Modify the schedule as needed.
* Too much time to complete everything.
  + Solution: Implement more complex facets of the querying language, such as *group by* and *having* clauses.

## Scenarios for Adequate Completion

In case of time constraints, external issues, or other various problems that might arise, the below scenarios are presented, detailing the minimum, expected, and best-case possible outcomes in order to consider the project a success. The minimum scenario must be attained in order to consider this project completed. This should be accomplished before the final due date.

* **Minimum Scenario** – At minimum, a GUI should be created with the basic functionalities of selecting a subset of columns and inserting data into an underlying XML storage facility.
* **Expected Scenario** – The expected set of features that should be implemented are as follows. The minimum should be completed as well as implementing sorting, joining, and grouping mechanics.
* **Best Case Scenario** – The expected scenario should be fully implemented. In addition, creating more complex joins such as cross joins, natural joins, etc., adding an ability to sort already grouped data using the *having* command. Furthermore, more creative elements may be implemented such as allowing the user to query more than one set of statements, thus resulting in two tables being displayed on the GUI, similar to how SQL Server displays as many tables as the sets of queries are selected. Lastly, if time permits, optimize querying speed.

## Milestones

Below is listed the major milestones of the project, along with a description of the elements contained in them.

* Research Phase
  + Gather needed research materials.
  + Fully utilize research materials to gain a better understanding of the subject matter.
  + Interview experienced programmers for the best plan of attack.
* Concept of Design Phase
  + Brainstorm best approach to implement the application.
  + Record rough ideas on paper.
  + Refine sketch into finished concept of design.
* Construction Phase
  + Construct GUI.
  + Design XML backend.
  + Implement objects to handle the middle tier—the interface between GUI and data backend.
  + Connect major design elements together.
* Testing Phase
  + Personally test GUI first with sundry test cases to ensure robustness.
  + Enlist the help of others who may break the system.
  + Refine any tenuous elements found during testing.
* Documentation Phase
  + Comment the code.
  + Exhaustively document the functionality and design of the working parts along with how they contribute to the application as a whole.
  + Incorporate this document into a final report.

# Resources

MacDonald, M. (Feb 2008). *Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5, Second Edition.* Apress.

Microsoft. (n.d.). *MSDN Library*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/default.aspx

Troelsen, A. (Nov 2007). *Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition.* Apress.