# Creating the Pilot AJAX Enabled Query Tool

Rachel Ford
rford191@uwsp.edu

Robert Dollinger
rdolling@uwsp.edu
Department of Computing and New Media Technologies, University of Wisconsin-Stevens Point, Stevens Point, Wisconsin, 54481-3897, United States of America

## Abstract

The Web Based AJAX Enabled Query Tool (AEQ tool) is an educational software designed to simplify and enhance database related learning. The goals of this tool were to: (1) Provide students with an easy-to-use, remotely accessible database interface that seamlessly integrated access to multiple database clients (MSSQL, MySQL, Oracle). (2) Provide students with easy means for interaction outside of class through live chat and a shared working mode. (3) Provide instructors with an intuitive interface for organizing both class participants and also class materials. These goals could only be met through a combination of internet technologies. In addition to employing traditional client-server processing approaches throughout the application, the AEQ tool relies heavily on AJAX technology in order to provide the desired functionality while delivering a seamless user experience.

**Keywords:** Query tool, Rich Internet Applications, AJAX, databases, educational software

## 1. INTRODUCTION

A number of challenges face students enrolled in database courses, particularly introductory courses. While some of these challenges, such as understanding relational database concepts and SQL syntax are, of course, to be expected, others can overwhelm and even distract from the purpose of the class. This is particularly true for students who are new to SQL and database concepts, and who must initially struggle both with learning these concepts and syntax as well as adapting to complex, oftentimes very different, database interfaces (such as MSSQL, Oracle, MySql, etc.). Furthermore, even for the advanced student who is comfortable enough with these topics as to be able to master the interfaces, as well as for the beginning student, remote access to databases can be problematic; while there are a number of means to combat this issue, none are without their own difficulties. Finally, even where they are available, tools incorporating class content and providing the means for students and instructors to easily interact and communicate about the course and its related materials, off campus and on, during class and after, are separate from student workspaces.

The AEQ tool was designed to combat these challenges and simplify and enhance the student's learning experience (as well as the instructor's teaching experience): to provide instructors with the means to add and organize class content; to provide students and instructors with the means to share work and communicate easily; to provide a robust interface for seamless integration of all available databases; and to provide reliable remote access to databases, class content, and class instructor(s) and participants. The AEQ tool is not designed to necessarily *replace*, but to *enhance* and *work in conjunction with*, individual database interfaces and classroom tools, and to simplify introductory learning, multiple database access, and remote access.

To implement an application that met each of these goals, a unique combination of traditional web processing techniques as well as the AJAX technology (Gibbs, 2007; McClure, 2006; Moore, 2007; Pars, 2007; Woolston, 2006; Zakas, 2006) was employed. This paper examines the implementation of this tool resulting from a senior capstone project, and the combination of technologies and approaches used. The remainder of this paper is organized as follows: section 2 discusses the functional breakdown of the main application components, and section 3 examines the architecture of the application. Section 4 overviews the technologies employed, and section 5 examines the particulars of implementing the application's main functional components. Finally, section 6 discusses some functional limitations of the application at present, and explores potential for future development and enhancement of the AEQ tool.

## 2. APPLICATION COMPONENTS

The AEQ tool consists of several major components, each of which interacts to perform unique tasks:

- SQL Launcher
- Content Viewer and Manager
- Classroom and User Administration
- Class Interaction Tools

### SQL Launcher

The SQL Launcher provides a Web based, unified interface to execute SQL queries and commands against MSSQL, MySQL and Oracle databases (although future expansion to include support for any number of other database systems is fully possible). The SQL Launcher can be used as an alternative to DBMS specific Windows based client interfaces (see Figure 1).
During initial class setup (handled through the administration tools), the course instructor or administrator would determine which types of databases would be available to students, and create those databases accordingly. After this setup, the AEQ tool will create and store the corresponding connection strings, and retrieve them at login. Switching from one database to the other is as easy as selecting the appropriate database in a group of radio buttons. Thus running the same SQL code against the various systems is made very easy.

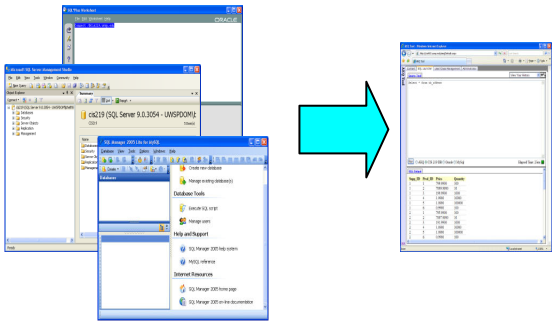
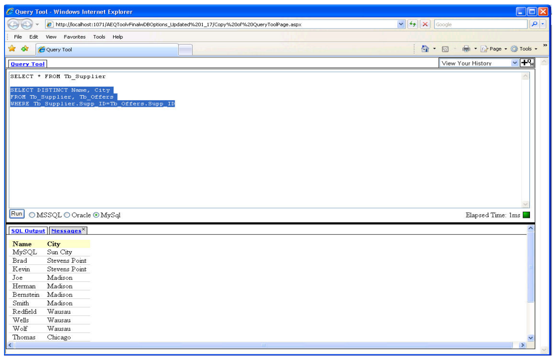


**Figure 1. Equivalent Functionality**

The SQL Launcher presents two main areas: the Query pane and the result pane (Figure 2).


Query Tool - Windows Internet Explorer
http://localhost:1071/AEQToolvFinalwDBOptions_Updated%201_17/Copy%20of%20QueryToolPage.aspx
File Edit View Favorites Tools Help
Query Tool
Page Tools
Query Tool
View Your History
Run
SQL Output
Messages

```
SELECT * FROM Tb_Supplier

SELECT DISTINCT Name, City
FROM Tb_Supplier, Tb_Offers
WHERE Tb_Supplier.Supp_ID=Tb_Offers.Supp_ID
```

( ) MSSQL ( ) Oracle [✓ MySql] Elapsed Time: 1ms

| Name | City |
|---|---|
| MySQL | Sun City |
| Brad | Stevens Point |
| Kevin | Stevens Point |
| Joe | Madison |
| Herman | Madison |
| Bernstein | Madison |
| Smith | Madison |
| Redfield | Wausau |
| Wells | Wausau |
| Wolf | Wausau |
| Thomas | Chicago |

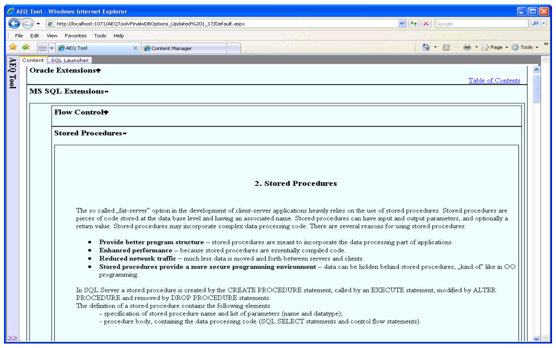
**Figure 2. The SQL Launcher Executing a Query**

The size of both areas is dynamically adjustable so that the two can trade the available space in the browser window. In the query area one can edit and/or paste any amount of SQL code, then highlight a selection from it and run the highlighted code. If nothing is selected the entire code in the window will be executed. The result pane will display either the results returned by the SQL queries or the messages returned by the target server, each in its own tab.

## Content Viewer and Manager

The Content Viewer and Manager is the most complex part of the AEQ application, and are meant to accompany the SQL Launcher in order to enhance the educational benefits and usability of the tool. It provides a structured and intuitive interface to serve the needs of both students, who are able to view the available course content, and instructors, who have the possibility to add/edit/delete and manage the content of their courses in a flexible way. Two different views, a Student View and an Instructor View of the course content are provided. Instructors have access to both views, while students only have access to the Student View.

The Student View allows users to navigate through the course content in two ways: by using a context sensitive Table of Contents control where a click on an item brings forward the corresponding full content or by expanding and hiding the full content items as desired (Figure 3). Furthermore, implementation is provided for users to copy and paste SQL code into the SQL Launcher for testing and study.


AEQ Tool - Windows Internet Explorer
http://localhost:1071/AEQToolvFinalwDBOptions_Updated%201_17/Default.aspx
File Edit View Favorites Tools Help
AEQ Tool
Content Manager
Page
Tools
AEQ Tool
Content
SQL Launcher
Oracle Extensions+
Table of Contents
MS SQL Extensions-
Flow Control+
Stored Procedures-

2. Stored Procedures

The so called „fat-server" option in the development of client-server applications heavily relies on the use of stored procedures. Stored procedures are pieces of code stored at the data base level and having an associated name. Stored procedures can have input and output parameters, and optionally a return value. Stored procedures may incorporate complex data processing code. There are several reasons for using stored procedures:

- **Provide better program structure** – stored procedures are meant to incorporate the data processing part of applications.
- **Enhanced performance** – because stored procedures are essentially compiled code.
- **Reduced network traffic** – much less data is moved and forth between servers and clients.
- **Stored procedures provide a more secure programming environment** – data can be hidden behind stored procedures, „kind of" like in OO programming.

In SQL Server a stored procedure is created by the CREATE PROCEDURE statement, called by an EXECUTE statement, modified by ALTER PROCEDURE and removed by DROP PROCEDURE statements.
The definition of a stored procedure contains the following elements:
- specification of stored procedure name and list of parameters (name and datatype);
- procedure body, containing the data processing code (SQL SELECT statements and control flow statements).

**Figure 3. Student View of the full course content**

Much more functionality is available to instructors in order not only to view, but also to manage, their course content. The main tool is a Table of Content Manager where instructors can dynamically rearrange the chapters of their course in a tree structured pattern (Figure 4).

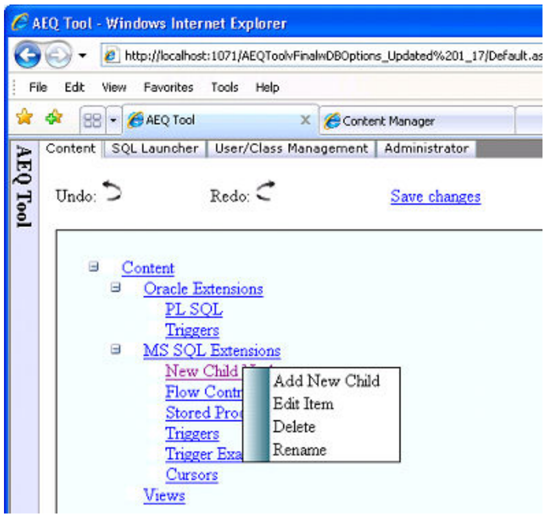


**Figure 4. Content Manager with options for managing a node**

Content can be added to a new node through a text editor and upload tool in several ways: simple typing, copy-and-paste from a different source, as well as by uploading a file. In addition an instructor may choose to designate parts of the content as SQL (ensuring that SQL formatting is provided in Student View and making it easier to transfer this content into the SQL Launcher); otherwise, all content is read as text. Content is designated as SQL by selecting that portion of the content and pressing the SQL button; likewise, SQL formatting can be removed in the same manner (Appendix A). Finally, an instructor managing multiple courses or sections can at any time select another class/section to manage or view from a dropdown list.

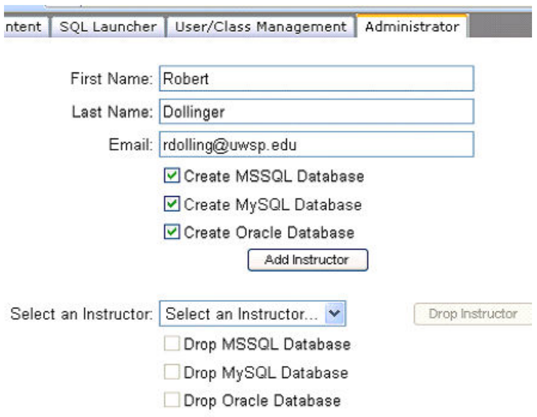
## Classroom and User Administration Tool

Since the AEQ tool is a complex and powerful application which, through some of its functions, manages resources like campus databases, it must ensure protection from unauthorized access. Proper security and role management features are integrated into the AEQ Administration Tool.

In order to provide user account support, the AEQ application takes advantage of several of the new Membership, Roles, and Profile features built in to ASP.NET 2.0 (Mitchell 2006). These built in features are used for authenticating, adding, creating, deleting, and modifying user account information, which is stored in a specific set of required tables. These tables are created automatically, and have been placed in the AEQ back-end database. The infrastructure created by the ASP.NET 2.0 Membership, Roles, and Profile features is programmatically used to provide the user account management functionality within the AEQ application.

Three different roles have been defined for the AEQ application: (1) Administrator, (2) Instructor and (3) Student.

**Administrator –** this role is the only one that provides access to the Administrator tab. Typically, there would be one single administrator for a given installation of the AEQ application. Administrator users are the only ones that cannot be created through the available functions of the AEQ application. One would create an administrator through the ASP.NET Web Site Administration Tool at application installation.
The main task of the administrator is to add and remove Instructor users (Figure 5).



First Name: Robert
Last Name: Dollinger
Email: rdolling@uwsp.edu
✓ Create MSSQL Database
✓ Create MySQL Database
✓ Create Oracle Database

Select an Instructor: Select an Instructor...
☐ Drop MSSQL Database
☐ Drop MySQL Database
☐ Drop Oracle Database

**Figure 5. The Administrator tab**

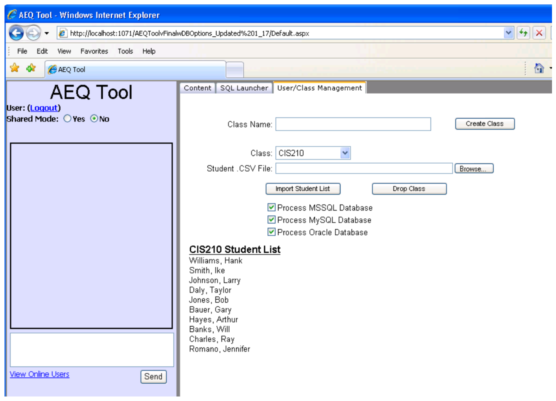
The administrator tab also provides the functionality for dropping an instructor.

**Instructor** - only instructors and the administrator have access to the User/Class Management tab, which allows them to create classes, import students to each class, and create corresponding student accounts. This set-up makes it possible to start with the single administrator role when installing the application, and then create and remove all other needed accounts from within the application itself.

The goal of the User/Class Management tab is to support instructors with some of the routine management tasks they perform at the start and end of each semester: (1) create a new class and accounts for the students in that class and (2) remove a class and all dependent information associated to it: students and accounts as well as working databases.

On the User/Class Management tab, the instructor (or administrator) can choose to modify an existing class or create a new class (Figure 6).

Both the Administrator and the Instructor roles have access to the functions of the User/Class Management tab. However, while an administrator can manage all the existing classes in the application, instructors can only see the classes created by themselves.



User: (Logout)
Shared Mode: Yes [✓ No]

Class Name:
Class: CIS210
Student .CSV File:
✓ Process MSSQL Database
✓ Process MySQL Database
✓ Process Oracle Database

CIS210 Student List
Williams, Hank
Smith, Ike
Johnson, Larry
Daly, Taylor
Jones, Bob
Bauer, Gary
Hayes, Arthur
Banks, Will
Charles, Ray
Romano, Jennifer

**Figure 6. User/Class Management tab**

**Student** – this is the most restrictive role in the application. None of the functionality related to the Administrator and Instructor role are available to student users. Students are limited to access the course content viewer, the SQL Launcher and the features in the chatroom (section 2.4).

## Class Interaction Tools

As one of the main objectives in the development of the AEQ application was to promote collaborative work among students, and between students and instructors, the

AEQ application includes three features that work together to achieve this goal: (1) The Chatroom, (2) The Active Users Window and (3) Shared Mode Support.

**The Chatroom** – is a chat feature that is implemented within the main design of the application, so that it is available at any time no matter the kind of activity the user is performing. The feature consists of a chat window and a text box used to send messages, both located on the left side of the application window. The area occupied by this feature can be adjusted horizontally or can be minimized to leave almost the entire window space for other activities. Users can send public messages and view all the messages sent by other users. Messages received while the user is logged in are displayed along with the timestamp of the message and the first and last name of the user who sent it. The list of messages is refreshed automatically every two seconds in all active user browsers.

**The Active Users Window** – complements the functionality of the Chatroom by showing the list of users currently online and their session mode, i.e. shared or not. This list is refreshed at two seconds intervals to maintain a constant, accurate report of who is logged in. The users list is displayed in a separate browser window that can be dragged anywhere, resized, closed, and reopened by clicking the 'View Online Users' link. By convention, a user is considered online for 2 minute after his/her Last Activity Date. A user without action for more than 2 minutes will be automatically removed from the list.

**Shared Mode Support** – every user can view his/her activity history by clicking on the icon in the upper-right corner of the SQL Launcher. The history consists of the past executed queries and their truncated outcome as logged in the AEQ database. Only the first 10 rows returned by a query are included in the history. Whenever a user enters the shared mode he/she makes his/her own history available to all other users who then can view, copy and test on their own the queries just produced by their peers. Any user entering shared mode is automatically added to the drop-down list by the history view list icon, and that student's history (since entering shared mode) is made available. This feature is an important part of the collaborative work support in the AEQ application, since it allows the students to truly share their work and experience, while being able to critique, comment and inquire about one another's work and results through the Chatroom feature. The two features nicely complement each other in order to provide powerful support for various types of collaborative activity.

## 3. APPLICATION ARCHITECTURE

The architecture of the AEQ tool is typical for AJAX applications. It consists of interacting components that are, potentially, located on three different computers (Appendix B).

**The AJAX Client** – consists of the JavaScript code executed by the browser on the end user machine. The code is downloaded as a result of the page requests from the AEQ Web Site, or even as a response to specific AJAX requests. The AJAX client takes care of the immediate user interaction that does not require the Web Server's participation, issues background AJAX requests upon specific user actions, receives and processes the data returned by the Web Server. E.g. a user click on the "Run" button in the SQL Launcher sends the SQL query in the query window as a parameter of an AJAX request to the Web Server to be forwarded to the selected target database. The query result is returned as part of the response to the AJAX request and is processed accordingly by the AJAX client: display the result set in case of success or the error message if the query failed. The AJAX Client consists of over 4,000 lines of JavaScript, roughly divided into the same functional modules as the application, with the addition of a shared module that handles AJAX requests and (typically more generic processing) that is done by at least two modules (see Appendix C). Each functional module performs two general tasks: issuing requests specific to its needs, and appropriately processing returned data/messages. Shared Mode Support is the one module that relies both on the shared AJAX functionality as well as another module (the SQL Launcher); this is because Shared Mode Support is a multi-user extension of individual SQL Launcher history.

**The AEQ Web Site** – is located on a campus Web Server and consists on the middle-tier logic of the application. It receives regular Web page or AJAX requests from the AJAX Client, interacts with the backend and/or target databases, and responds to the client requests. E.g. for example, when an AJAX request to execute an SQL query/command is received, the AEQ Web Site component connects to the selected target database and submits the SQL query/command for execution. The results from the database are then forwarded back to the AJAX client. The AEQ Web Site depends on separate handlers for Administrative Tools, SQL Launcher, Content Viewer/Manager, as well Interactive Tools; each of these handlers relies on other classes for specific processing. Some of these classes -- such as DBManager, and UserEngine -- are used throughout these handlers, and others are more specific to the processing at hand. For example, the server-side components and dependencies of the Administrative Tools include the Administrator and UserManagement aspx pages (both wrapped in the master application page), a shared handler, and four other classes (Figure 7).

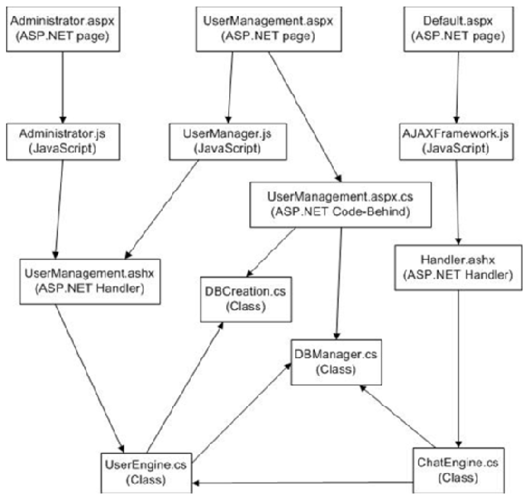

Administrator.aspx (ASP.NET page)
UserManagement.aspx (ASP.NET page)
Default.aspx (ASP.NET page)
Administrator.js (JavaScript)
UserManager.js (JavaScript)
AJAXFramework.js (JavaScript)
UserManagement.aspx.cs (ASP.NET Code-Behind)
UserManagement.ashx (ASP.NET Handler)
DBCreation.cs (Class)
Handler.ashx (ASP.NET Handler)
DBManager.cs (Class)
UserEngine.cs (Class)
ChatEngine.cs (Class)

**Figure 7. Dependencies and Interactions of the Administrative Tools**

**The AEQ Backend Database** – is located on a campus database server and stores data that is vital to the functioning of the application (Appendix D): course lists, course content, student/user lists, passwords and permissions, information about the registered target databases and other. Users (stored in an ASP.NET created table) are linked to their available database connections, their query history, their messages (in the Chatroom), and their classes; classes are linked to their content.

In addition to the three components above, the AEQ application also interacts with a number of target databases towards which the user's queries are directed. These databases can rely on different types of Database Management Systems, and are created by the class instructors for the specific teaching needs of their classes.

## 4. TECHNOLOGIES EMPLOYED

In order to provide a seamless user interface and a Windows application-like user experience, the AEQ tool incorporates a number of technologies. The application was designed in ASP.NET 2.0, with server-side code written in C#.NET. The AEQ tool relies on an MSSQL backend database (for class and user management, and course content). The application is heavily dependent on AJAX technologies, incorporating XMLHttpRequest's, Microsoft's AJAX libraries and controls, modified third party freeware AJAX controls, and custom controls. Generally, data is represented on and transmitted from the server as XML, and transformed on the client side through JavaScript and CSS.

Indeed, XML is the main vehicle of data exchange used throughout the application. While, occasionally, JSON's lightweight, contained format provided the best fit for portions of the application, generally the robust, self-descriptive, easily navigable XML was preferred. This was for several reasons: firstly, the XML tree structure proved an identical fit for some data used in the application (particularly the outline-like class content structure); secondly, there is ample support for both server and client-side navigation, transformation and manipulation of XML data; thirdly, complex XML data structures can be easily stored in and retrieved from a database; finally, the available tools for XML processing allow simple, straightforward, lightweight solutions to operations such as restructuring a data tree. Also taken into consideration was the fact that some handling of XML data would need to be provided when retrieving data from student and instructor databases, regardless of the preferred exchange format, due to the possibility that queries might either be run on XML data columns or formatted to return XML data.

While, as noted previously, the application was designed to work with a MSSQL backend database, implementation for creating and accessing three types of databases was designed: MSSQL, MySQL and Oracle. The incorporation of a larger number of database types in the future is a fully viable option, simply requiring the creation of a wrapper class for each additional type.

The AEQ tool was designed to be run in Internet Explorer, however the application was built with the future incorporation of cross-browser compatibility in mind; this means that browser specific JavaScript was avoided, and, when possible, portions of the application were built for and debugged in multiple browsers; and, when not possible, proper internal documentation and stub methods for future development were provided.

## 5. EXAMINATION OF SPECIFIC IMPLEMENTATION OF MODULES

While the base technologies employed throughout the application were the same, the requirements of the AEQ tool necessitated significant differences in the approaches used for implementing each functional module.

Each module exists independently, but is wrapped inside the main application frame in order to work in unison to produce the desired functionality. The wrapper is an aspx page that handles the majority of the details associated with linking pages and functionality. It is also here that the JavaScript code for dynamic resizing of portions of the screen, as well as most of the class interaction tools, are linked. This page is not a MasterPage, but behaves similarly to one.

### SQL Launcher

The SQL Launcher was constructed as a separate application, and later wrapped in the framework of the main application. In addition to the base .aspx webpage, the SQL Launcher consists of a webhandler, where most of the server side processing occurs, several C# classes for managing custom query tool exceptions and handling database access, and JavaScript and CSS for client-side processing and formatting.

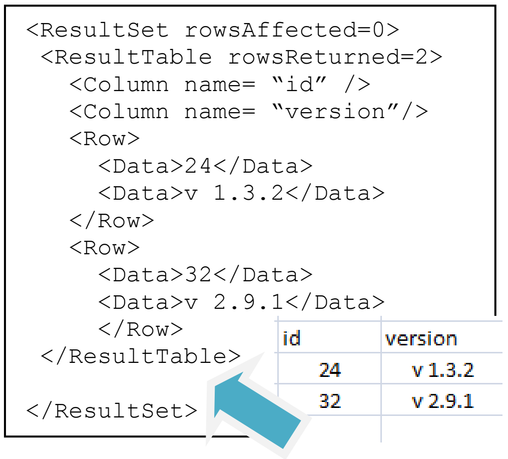
The process for using the SQL Launcher is as follows: after the user presses "Run", client-side JavaScript issues an XmlHttpRequest. The QueryToolHandler receives this request and calls the appropriate methods to process it; the results (be they data, messages, or errors) are returned to the client in an XML format; the client transforms and formats the data using JavaScript and CSS.
The user may choose to run queries on any database he or she has permissions to access by simply selecting a radio button. The ability to access multiple database types, such as the three for which implementation is provided, MSSQL, MySQL and Oracle, is achieved through the existence of an abstract class, DatabaseHandler, from which all other, database-specific classes are derived. This class contains abstract method headers for a number of methods that one would expect to find in such a class, such as creating, altering and fetching connection strings (Appendix E). It is then left up to the individual classes to implement these methods as appropriate.

While the MSSqlHandler, MySqlHandler and OracleHandler classes provide implementation for the MSSQL, MySQL and Oracle databases (respectively), the application is designed to make use of polymorphism in such a way that adding an additional type of database would be as simple as deriving another class from DatabaseHandler, and including it with the project (Appendix F). After the creation and inclusion of this new class, the application would then be able to access and process databases of whatever type for which the class provided implementation. This is possible because the application does not handle database specific objects in processing, but rather uses DatabaseHandler objects, leaving the database specific details to the derived classes.

When a user runs a query, it could potentially return any combination of row-column results, XML data, messages or errors. The QueryTool processes these types of return data differently in order to return pure XML to the client. When a query results in XML data, the result is returned unmodified.

When a query generates errors or messages, each message or error is assigned its own XML node; interestingly enough, Oracle returns general messages (such as Print statements) alongside of genuine error messages, so these are processed by the AEQ tool in the order in which they are received.

For this reason, the SQL Launcher processing of these messages, both on the server side, and later on the client side, is very generic: the message, whatever its content, is simply presented to the user, allowing its content (rather than, for instance, font color or size changes) to indicate its purpose. Finally, when a query returns row-column data, the SQL Launcher generates XML modeling the table structure. Column names are preserved before the data. Each row is represented as a "Row" element, and each cell inside that row is represented as a "Data" element (Figure 8). When a table contains columns of XML data as well as other data types, the XML data is simply inserted into the appropriate "Data" element(s) in the return XML document.



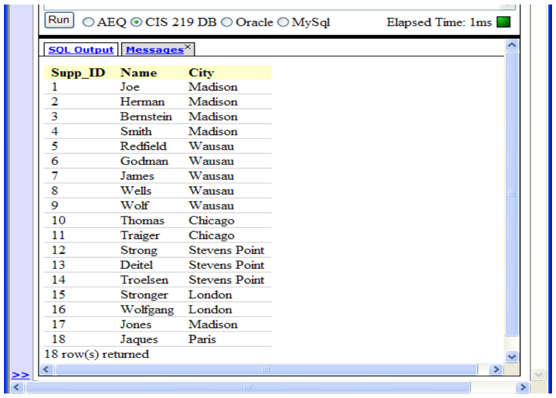
```
<ResultSet rowsAffected=0>
 <ResultTable rowsReturned=2>
   <Column name= "id" />
   <Column name= "version"/>
   <Row>
     <Data>24</Data>
     <Data>v 1.3.2</Data>
   </Row>
   <Row>
     <Data>32</Data>
     <Data>v 2.9.1</Data>
     </Row>
 </ResultTable>

</ResultSet>
```

| id | version |
|---|---|
| 24 | v 1.3.2 |
| 32 | v 2.9.1 |

**Figure 8. XML Representation of a Simple Table**

For all return types, the number of rows affected and the number of rows returned are sent back to the client, and displayed to the user (much as is done in the database clients).

After the XML is generated and returned by the handler, client side JavaScript processes and displays it in a table format in the output pane of the SQL Launcher (Figure 9). Similar to the MSSQL database interface, XML data is previewed in its appropriate column, but can be opened for viewing in a separate tab. As noted previously, the number of rows returned or rows affected is also displayed in the output pane. Furthermore, the client displays the query execution time, along with an icon indicating whether the execution was successful or not (green signifying success, red signifying failure).



Elapsed Time: 1ms 🟩

| Supp_ID | Name | City |
|---|---|---|
| 1 | Joe | Madison |
| 2 | Herman | Madison |
| 3 | Bernstein | Madison |
| 4 | Smith | Madison |
| 5 | Redfield | Wausau |
| 6 | Godman | Wausau |
| 7 | James | Wausau |
| 8 | Wells | Wausau |
| 9 | Wolf | Wausau |
| 10 | Thomas | Chicago |
| 11 | Traiger | Chicago |
| 12 | Strong | Stevens Point |
| 13 | Deitel | Stevens Point |
| 14 | Troelsen | Stevens Point |
| 15 | Stronger | London |
| 16 | Wolfgang | London |
| 17 | Jones | Madison |
| 18 | Jaques | Paris |

18 row(s) returned

**Figure 9. Result pane of the Query Launcher after successful execution of a query**

## Content Viewer and Manager

Of all portions of the application, the Content Viewer and Manager perhaps rely most heavily on client-side processing and AJAX technology, as this portion of the application provides functionality for dynamic navigation, display, manipulation and creation of class content. While the Viewer and Manager perform very different tasks, they also share some similar and some identical processing, and so make use of some of the same client and server side code (Figure 10).

Class content is stored as an XML tree in the application MSSQL database (Appendix G); this tree is automatically generated when the class is created. Instructors may then alter this initial, empty tree to create whatever structure best suits the class' needs, with but one stipulation: that actual text or SQL content can only exist in leaf nodes. Despite the downsides of passing potentially large XML files from the server to many clients simultaneously, XML seemed the most appropriate fit for a classroom setting: XML's tree structure lent itself perfectly to the tree structure of a class outline. Furthermore, XML can be passed easily enough between the server and clients, and clients and the server; and, while not only does the self documenting nature of XML make clear the purpose of individual nodes in a way that would not have been difficult to mimic using a lighter weight data-interchange format such as JSON, but also the clear structure makes processing significantly simpler and more straightforward.

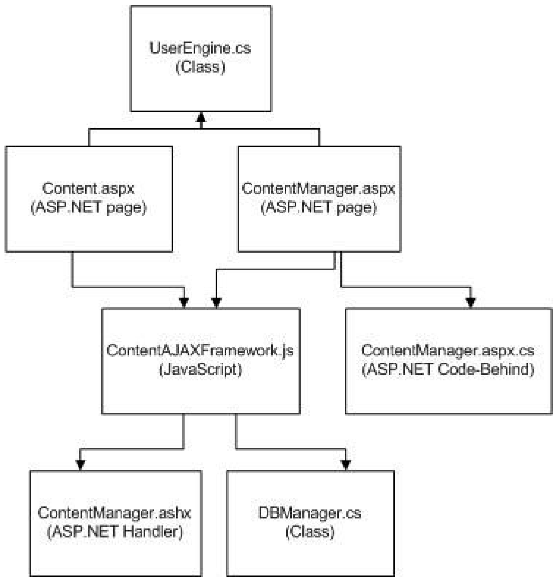
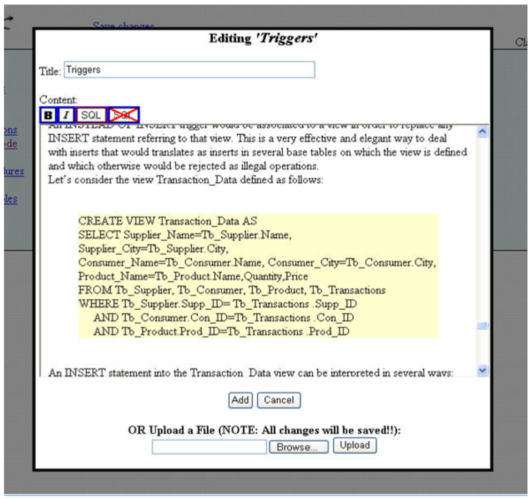


**Figure 10. Overview of Content Viewer and Manager structure, showing common code and classes**

On the client side, the XML tree is represented by a tree control that can be dynamically rearranged by dragging and dropping from one location to another (Appendix H); every content node in the XML tree is a node in the tree control. Instructors are provided with the functionality to rename, reorganize, delete, edit and, of course, create nodes; in addition to typing or pasting text, instructors can upload files inside a leaf node (Figure 11). Furthermore, if an instructor feels that an error has been made, he/she may choose the Undo or Redo option at any time after at least one change has been done or undone (respectively).
Each of these manipulations, additions, etc., alters the XML Content tree. To prevent what could be very excessive server traffic (were these changes constantly sent to the database), changes, until the instructor chooses to save them, are not saved to the database, and exist only on the client browser.



INSERT statement referring to that view. This is a very effective and elegant way to deal with inserts that would translates as inserts in several base tables on which the view is defined and which otherwise would be rejected as illegal operations.
Let's consider the view Transaction_Data defined as follows:

```
CREATE VIEW Transaction_Data AS
SELECT Supplier_Name=Tb_Supplier.Name,
Supplier_City=Tb_Supplier.City,
Consumer_Name=Tb_Consumer.Name, Consumer_City=Tb_Consumer.City,
Product_Name=Tb_Product.Name,Quantity,Price
FROM Tb_Supplier, Tb_Consumer, Tb_Product, Tb_Transactions
WHERE Tb_Supplier.Supp_ID= Tb_Transactions .Supp_ID
   AND Tb_Consumer.Con_ID=Tb_Transactions .Con_ID
   AND Tb_Product.Prod_ID=Tb_Transactions .Prod_ID
```

An INSERT statement into the Transaction_Data view can be interpreted in several ways:

**Figure 11. The SQL and text editor**

This functionality is made available through a combination of generic, AJAX enabled JavaScript tree and context menu controls, an application specific SQL and text editor, application-specific JavaScript XML tree processing, and a simple web handler that retrieves from and saves to the database.

While the Content Viewer functionality is not as complex as the Content Manager, it utilizes much of the same lower level XML tree processing and loading as does the Manager; here again, the structure and self-descriptive nature of XML matches the needs of the application perfectly. Furthermore, CSS and JavaScript transformation seemed most appropriate for both the Viewer and the Manager, as both require, at one level or another, very similar dynamic processing, which, combined with the modular structure of the JavaScript code, allowed for extensive code reuse.

The Viewer consists of two main parts: a Table of Contents control (a tree structure, represented by a styled HTML list, in which content nodes are represented by their titles, and are linked to their node content via the node id), and the Content View.

When the Content Viewer is initially loaded, content node titles are fetched in order to build both the Table of Contents as well as the main page, which will display the titles of all first level nodes. After this, any attempt to load content at any sub-level (either by expanding a main level node, or by navigating via the Table of Contents) will issue an XmlHttpRequest to the web handler to fetch the node at that level; the exception to this is if the data has already been loaded -- in that case, it is merely displayed. Otherwise, the requested XML node(s) will be fetched from the database by the web handler, and returned to the client. On the client side, the returned XML will be processed and formatted through JavaScript and CSS.

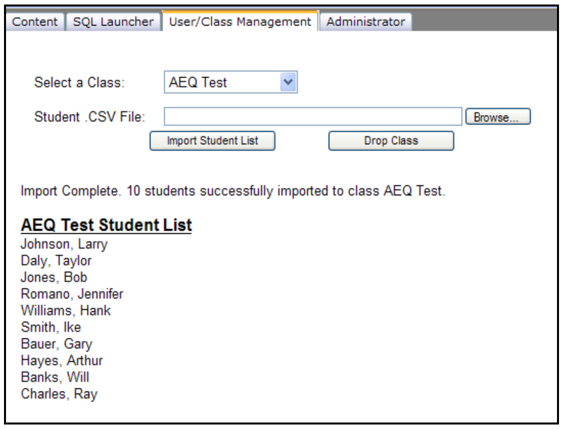
Nodes in Content Viewer can be collapsed and expanded at will. A collapsed node does not lose its data (i.e. a fresh call is not made to the server should the user expand the node).

## Classroom and User Administration

The Administration portion of the application most resembles traditional web applications, as it runs predominately on the server side. Nonetheless, in order to maintain a uniform user experience across the application, the Administrative portion of the AEQ tool makes use of AJAX technology where appropriate to minimize postbacks and page refreshes (in keeping with the Windows application-like feel of the application).

The main purpose of the administrative tools is to create users (instructors and students), to create classes, and to associate users with classes. The AEQ tool employs built-in features of ASP.NET 2.0 to define Membership, Roles and Profiles, and provides an interface for easily and quickly creating users and assigning them appropriate roles (Administrator, Instructor, Student). Only administrators can create and remove instructors; both administrators and instructors can create and delete students, just as both can create, manage and remove classes. Students, of course, cannot create, modify or delete other users or classes.

While instructors are created individually (and separately from class creation and management) by an instructor, an instructor or administrator user can both create and manage a class in the same work area (Figure 12).



Import Complete. 10 students successfully imported to class AEQ Test.

AEQ Test Student List
Johnson, Larry
Daly, Taylor
Jones, Bob
Romano, Jennifer
Williams, Hank
Smith, Ike
Bauer, Gary
Hayes, Arthur
Banks, Will
Charles, Ray

**Figure 12. Managing users associated to a test class**

Class management options which would trigger a postback are available inside an iFrame -- which is invisible to the user, but allows for partial page refreshes. For instance, when an instructor uploads a classlist CSV file, a postback is triggered inside the iFrame, which refreshes and displays the outcome of the upload attempt; the main page, however, does not refresh. This helps provide a more stable, Windows-like "feel", as can be found elsewhere throughout the application.

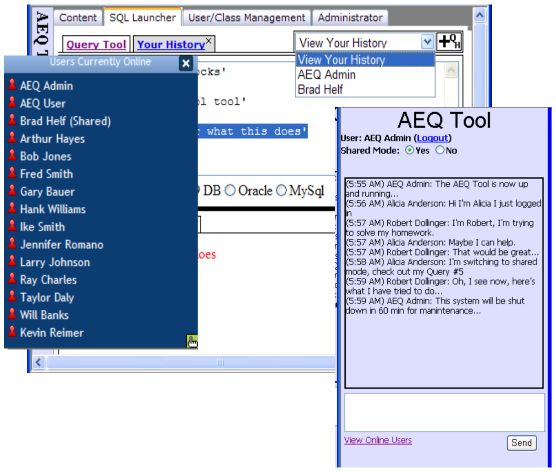
## Class Interaction Tools

Three features designed to work in unison to enhance classroom interaction are the chatroom, active user list, and shared mode (Figure 13). The active user list shows not only who is currently logged in, but also who is working in Shared Mode; the chatroom allows all logged in users (working in Shared mode or otherwise) to communicate; and Shared Mode History allows users to view the Shared History of any user working in Shared Mode.

Both the Chatroom and User List are AJAX driven. The Chatroom refreshes every two seconds to provide constant updates of user messages. Unlike other portions of the app, where potentially complex structures are neatly represented as XML, chat messages are wrapped and exchanged in JSON; JSON, lacking the verbosity of XML, provides a lighter, cleaner transfer of data, perfect for use in instances such as this, where there are no nested structures or complexity of data, but simply sets of three pieces of data (user name, time stamp and text message).

Chat and the User List, along with the Shared Mode option, are embedded into the main application; this ensures that they are always available to any logged-in user, regardless of his/her activity at any given moment. Access to Shared Mode History (the queries run, and a subset of the first ten results, messages, or errors returned, by individuals working in Shared Mode) is available in the SQL Launcher. Unlike the Chatroom and User List, where all users can interact equally and in the same way, Shared Mode History, it should be noted, is not an interactive workspace where multiple persons can collaboratively work on a query or queries, but rather a window in which one can *view* (but not change, without importing to one's SQL Launcher window) another user's query and result history.

Because multiple users might be working in shared mode at any given time, and a student or instructor might wish to view the Shared history of any number of these users, each user's Shared history is displayed in its own tab. These tabs are the JavaScript controls used in the SQL Launcher, and actually display alongside the Query Tool.



AEQ Admin
AEQ User
Brad Helf (Shared)
Arthur Hayes
Bob Jones
Fred Smith
Gary Bauer
Hank Williams
Ike Smith
Jennifer Romano
Larry Johnson
Ray Charles
Taylor Daly
Will Banks
Kevin Reimer

User: AEQ Admin (Logout)
Shared Mode: [✓ Yes] No

(5:55 AM) AEQ Admin: The AEQ Tool is now up and running...
(5:56 AM) Alicia Anderson: Hi I'm Alicia I just logged in
(5:57 AM) Robert Dollinger: I'm Robert, I'm trying to solve my homework.
(5:57 AM) Alicia Anderson: Maybe I can help.
(5:57 AM) Robert Dollinger: That would be great...
(5:58 AM) Alicia Anderson: I'm switching to shared mode, check out my Query #5
(5:59 AM) Robert Dollinger: Oh, I see now, here's what I have tried to do...
(5:59 AM) AEQ Admin: This system will be shut down in 60 min for manintenance...

**Figure 13. AEQ Interactive tools: User list, Shared Mode, and the Chatroom**

## 6. LIMITATIONS, AND LOOKING FORWARD

The current implementation of the AEQ Tool is a fully functional pilot version, however it is not without its limitations. Among them are: at present, xqueries and xpath expressions do not evaluate; furthermore, while tables, schemes, etc. can be created, altered, and dropped, there is no way to directly explore or visualize database content; administrative functions, while providing ample support for the tasks described, do not provide for customization of accounts, database access for individuals and groups, etc.; and, at present, there is no concept of private chatrooms or shared, interactive workspaces within the application.

Addressing these limitations is certainly a part of the future of the AEQ Tool. Furthermore, student and instructor feedback gathered through test runs of the application in classroom settings is certain to shape priorities and impact future revision, assessment and development of the tool.

## 7. CONCLUSION

The AEQ tool, as a fully functional pilot application, provides a solid base for future refinement and development. It not only provides an opportunity to test, refine and enhance the specific implementation choices made during the development of the application, but also the concepts behind the AEQ tool. As much as the actual building of the application, the prospect of deploying the application for use is a promising one, because it too will provide an opportunity to reconsider, validate or invalidate, and, later, perfect the approaches used in this deployment; this cycle of testing and use will, in a very real sense, lend direction to the future of the application. The lessons learnt in the building and deploying of this pilot version of the AEQ Tool will help identify the strengths and weaknesses of the concepts and approaches used in creating the application. This knowledge, in turn, may lead to some of the approaches and ideas behind the tool being reworked, transformed or even abandoned, just as it may lead to the extension, enhancement and continuation of others; regardless, it will ensure a stronger, more reliable application that can effectively and efficiently meet a real classroom need.

## 8. REFERENCES

Gibbs Matt, Wahlin Dan (2007) Professional ASP.NET 2.0 AJAX, Wiley Publishing, Inc.

McClure Wallace B., Cate Scott, Glavich Paul, Shoemaker Craig (2006) Beginning AJAX with ASP.NET, Wiley Publishing, Inc.

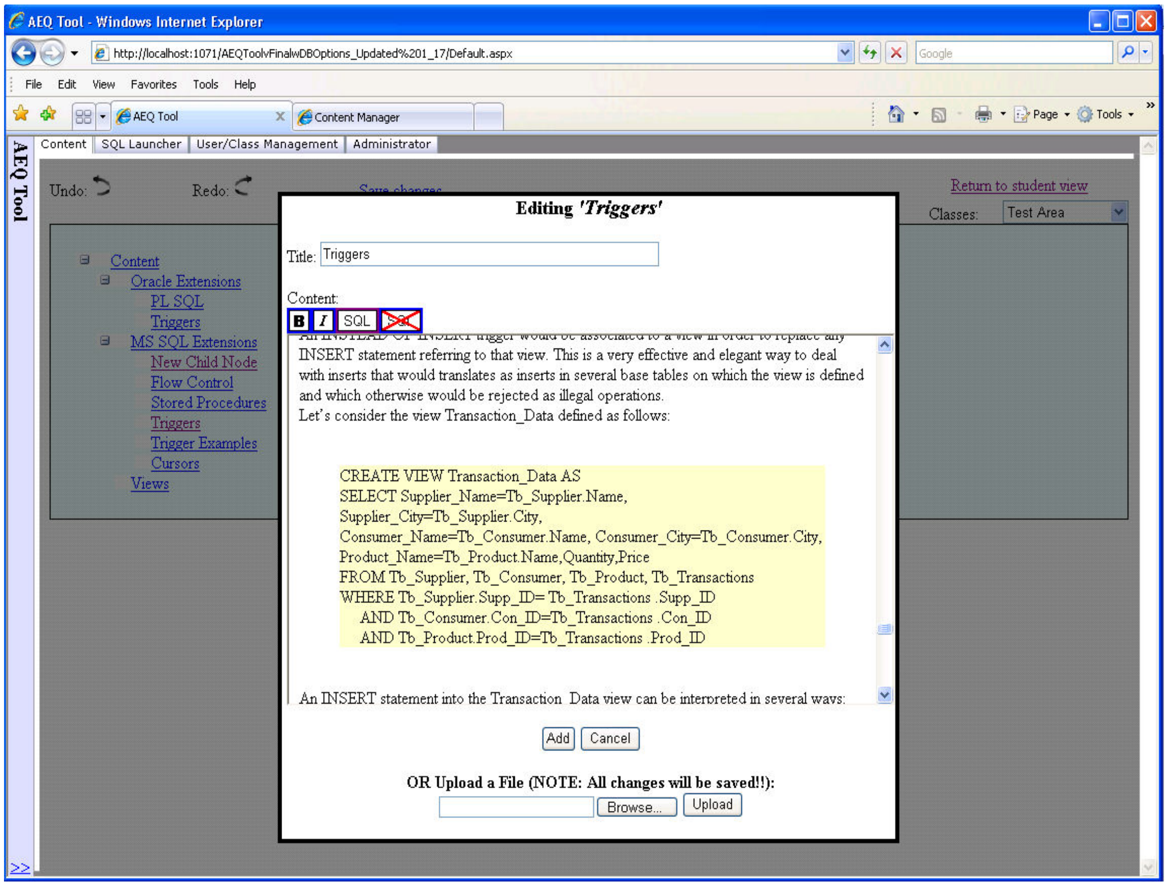
Mitchell Scott (2006) Examining ASP.NET 2.0's Membership, Roles, and Profile - Part 3, http://aspnet.4guysfromrolla.com/articles /040506-1.aspx

Moore Dana, Budd Raymond, Benson Edward (2007) Professional Rich Internet Applications: AJAX and Beyond, Wiley Publishing, Inc.

Woolston Daniel (2006) Pro AJAX and the .NET 2.0 Platform, Apress.

Zakas Nicholas C., McPeak Jeremy, Fawcett Joe (2006) Professional AJAX, Wiley Publishing, Inc.

## Appendix A



An INSTEAD OF INSERT trigger would be associated to a view in order to replace any INSERT statement referring to that view. This is a very effective and elegant way to deal with inserts that would translates as inserts in several base tables on which the view is defined and which otherwise would be rejected as illegal operations.
Let's consider the view Transaction_Data defined as follows:

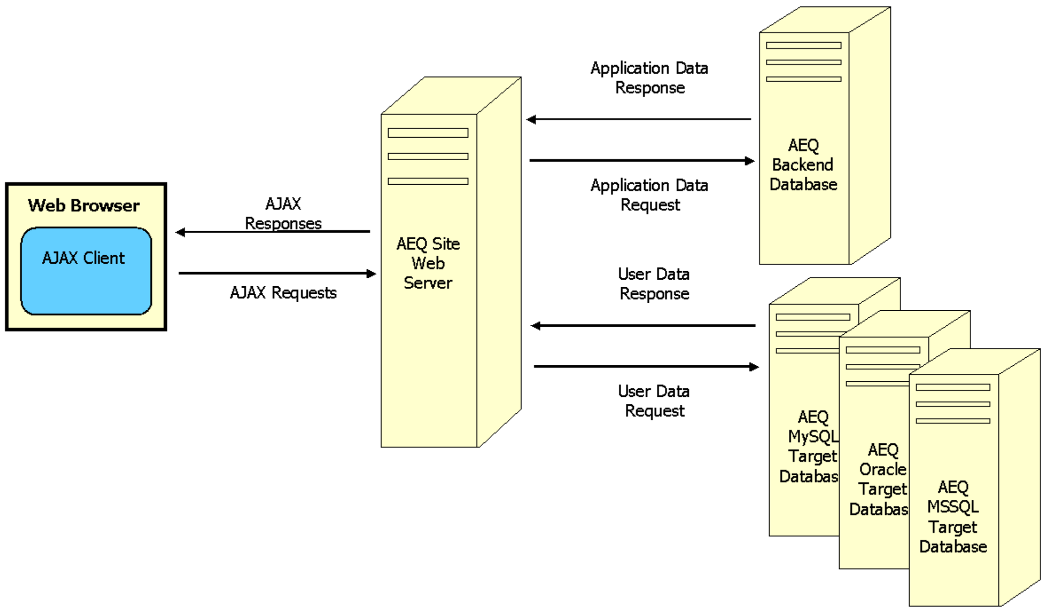
```
CREATE VIEW Transaction_Data AS
SELECT Supplier_Name=Tb_Supplier.Name,
Supplier_City=Tb_Supplier.City,
Consumer_Name=Tb_Consumer.Name, Consumer_City=Tb_Consumer.City,
Product_Name=Tb_Product.Name,Quantity,Price
FROM Tb_Supplier, Tb_Consumer, Tb_Product, Tb_Transactions
WHERE Tb_Supplier.Supp_ID= Tb_Transactions .Supp_ID
   AND Tb_Consumer.Con_ID=Tb_Transactions .Con_ID
   AND Tb_Product.Prod_ID=Tb_Transactions .Prod_ID
```

An INSERT statement into the Transaction_Data view can be interpreted in several ways:

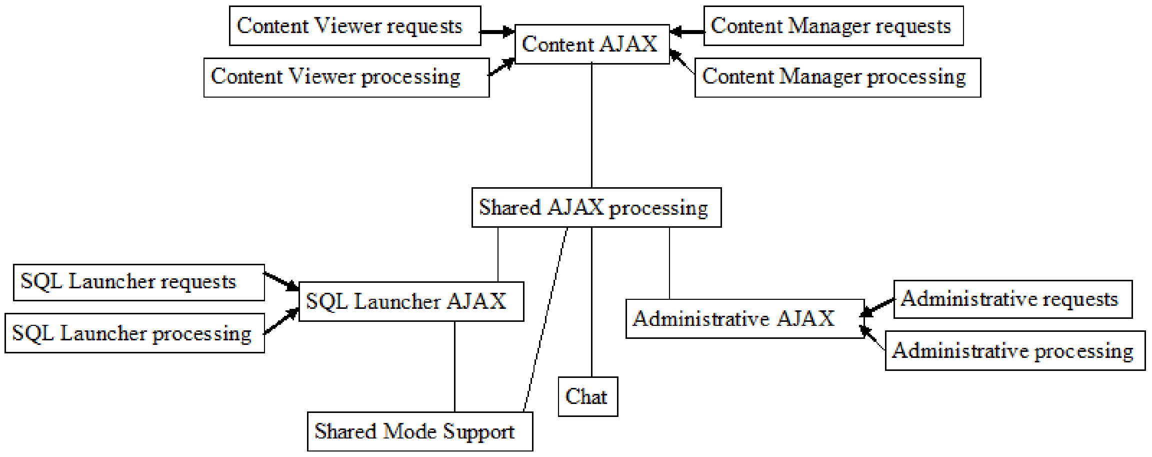
**Content Editor with text and SQL content**

## Appendix B

**Architecture of the AEQ Application**

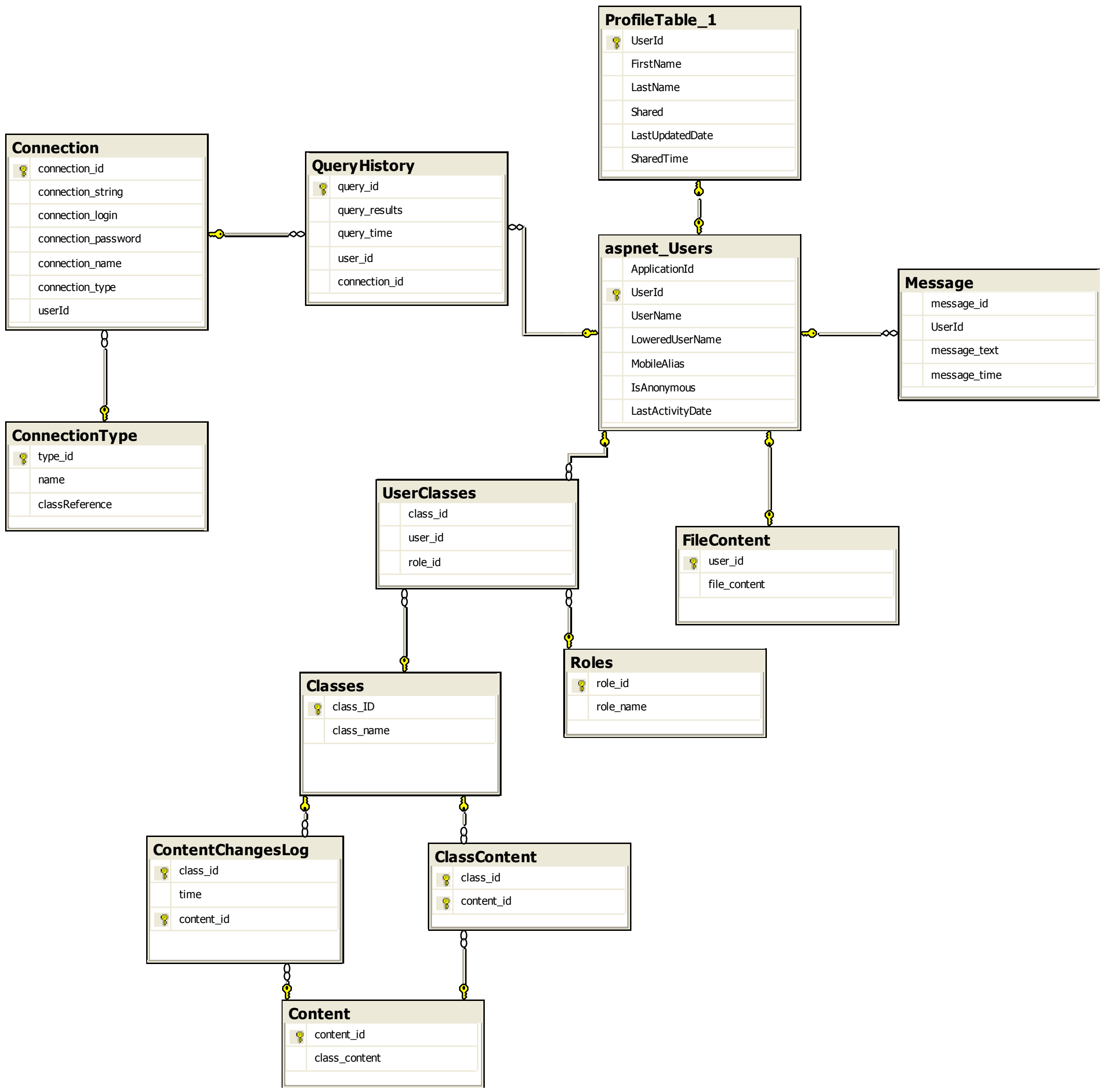


## Appendix C



**The basic functional structure of the AEQ tool AJAX client (note that Shared Mode Support borrows functionality from SQL Launcher as well as shared functions)**

## Appendix D



**Entity Relationship Diagram of AEQ backend database (ignoring ASP.NET provided tables)**

## Appendix E

```
public abstract class DatabaseHandler
{
  public event MessageEventDelegate MessageEvent;
  public abstract DbConnection createConnection(string connectionString);
  public abstract DbCommand createCommand(string commandString);
  public abstract DbConnection getConnection();
  public abstract DbCommand getCommand();
  public abstract bool configureConnection();
  public abstract ErrorMessage[] getOutput();
  ...
}
```

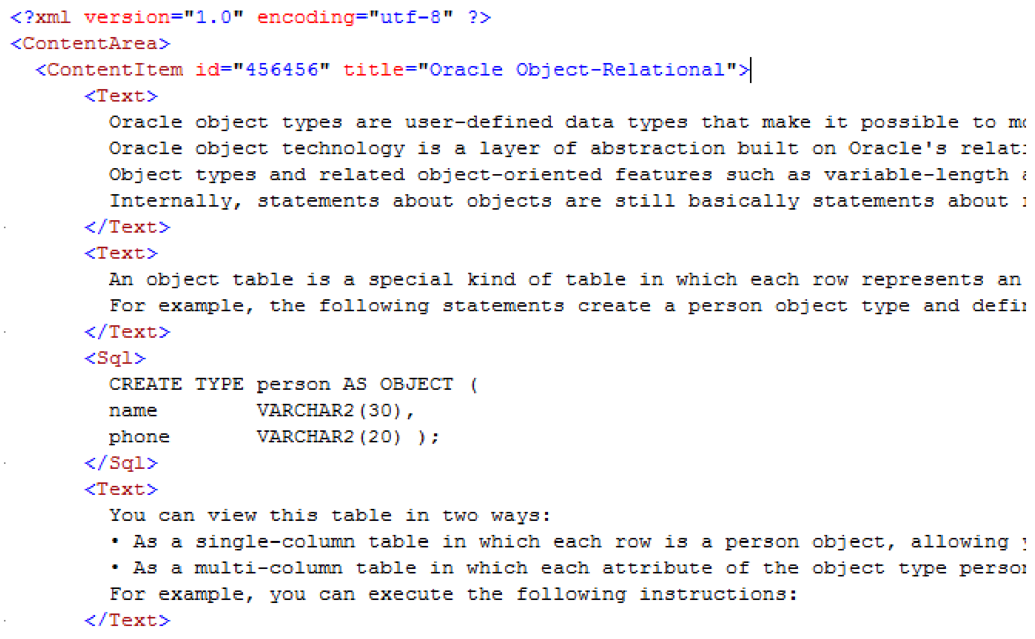
**Code from the DatabaseHandler class**

## Appendix F

```
public class XYZHandler : DatabaseHandler
{
    private ArrayList messages = new ArrayList(10);
    private XYZCommand command = null;
    private XYZConnection connection = null;
    public override DbConnection createConnection(string connectionString)
    {
      connection = new XYZConnection(connectionString);
      return connection;
    }
...

}
```

**Sample implementation for an XYZ database**

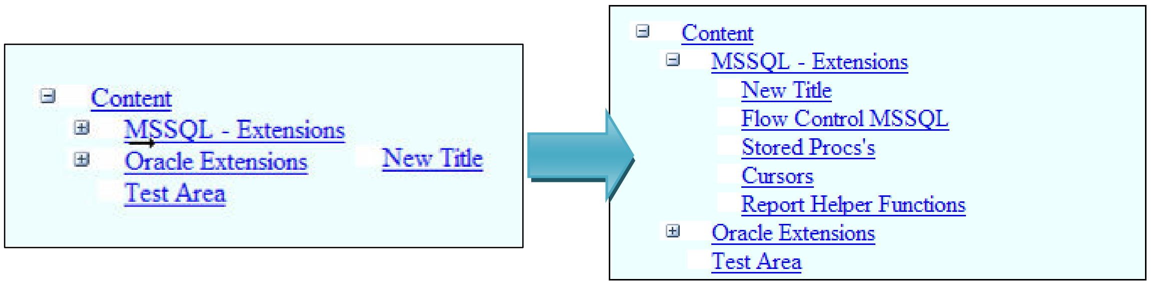
## Appendix G



```
<?xml version="1.0" encoding="utf-8" ?>
<ContentArea>
  <ContentItem id="456456" title="Oracle Object-Relational">
      <Text>
        Oracle object types are user-defined data types that make it possible to mo
        Oracle object technology is a layer of abstraction built on Oracle's relat:
        Object types and related object-oriented features such as variable-length a
        Internally, statements about objects are still basically statements about :
      </Text>
      <Text>
        An object table is a special kind of table in which each row represents an
        For example, the following statements create a person object type and defi:
      </Text>
      <Sql>
        CREATE TYPE person AS OBJECT (
        name        VARCHAR2(30),
        phone       VARCHAR2(20) );
      </Sql>
      <Text>
        You can view this table in two ways:
        • As a single-column table in which each row is a person object, allowing y
        • As a multi-column table in which each attribute of the object type perso:
        For example, you can execute the following instructions:
      </Text>
```

**Example Content XML tree, illustrating leaf Content node (it will contain no Content nodes)**

## Appendix H



**Content Manager, before and after dragging a node to a new location**