Achieving Business Intelligence ROI

Table of Contents

Achieving Business Intelligence ROI

Business Intelligence - pig or bonanza?

As competition increases and intensifies, organizations are discovering clever ways to systematically analyze data and provide alerts of changing business conditions in real time. These Business Intelligence (BI) applications drive greater productivity, new sources of revenue and competitive advantage. Below are a few examples:

- **Integrating competitive pricing information into price optimization or revenue management systems** — Extracting competitor pricing information from e-business sites so immediate action can be taken to maximize gross margins or respond to the actions of competitors.
- **Collecting and organizing content to populate enterprise portals and dashboards** — Crawling the Web, internal information sources and subscription services to automatically populate portals and dashboards displaying Key Performance Indicators (KPIs) and other pertinent content.
- **Real time monitoring of information sources to predict market movements** — Checking Web sites and subscription services for real time updates on information that could profoundly affect markets and risk.

Because of their promise, BI applications are expensive. Also, to no one's surprise, most developers prefer working on the components of these applications perceived to provide the most value to their organizations — analytics and presentation. Many developers find data gathering uninteresting and tedious. Unfortunately, without good data, cool analytics and presentation are akin to putting lipstick on a pig. Conversely, effective analytics and presentation combined with a robust and scalable platform for data collection results in a rapid Return On Investment (ROI)

Challenges

Challenges:

- Extracting data in semi-structured and unstructured formats
- Navigating arcane and disjointed data networks (including the Web)

The data to support most BI applications is generally swirling around in enterprise data networks that would make Rube Goldberg proud. Enterprise data networks typically encompass the World Wide Web, portals, subscription services, local file systems, email repositories and enterprise data stores. Much of this data is not structured for automated extraction (estimated as high as 75%), yet it is critical for effectively supporting applications like those described above. Different approaches have been taken to solve the dual problems of 1) extracting data in semi-structured and unstructured formats and 2) navigating arcane and disjointed data networks. Each of these approaches has its own advantages and disadvantages. Here we examine artificial intelligence and macro recording techniques.

Extraction

Using Artificial Intelligence (AI) is a popular approach to data extraction. By analyzing content characteristics and surfing behavior, AI can attempt

extraction of specific data (such as contact information from a Webpage).

This computerized "guesswork", however is usually too imprecise to support most BI applications.

A more precise approach is the macro recorder which provides a Graphical User Interface (GUI) to easily pinpoint data for extraction. The software records a user's actions while marking the desired data for later automatic replay. Though easy to precisely pinpoint required data, it is extremely tedious, fragile and inefficient — particularly if the data you seek appears on a variable number of pages and exist in several different formats.

A comprehensive query language is the best solution to handle complex network navigation and data extraction

Navigation

While you can use AI or macro recording techniques to extract information from a document, using these same techniques to navigate through a network to that document is a different story. Neither technology lends itself to efficient navigation through an enterprise data network, particularly when you consider the dynamic nature of the Web. Filling out Web forms and interpreting Java script are difficult to accomplish using AI or macro recorders.

To overcome this, many extraction tool vendors add a scripting language to handle the navigation and then use their extraction techniques once they get to the content. This inconsistent development environment often produces difficult to maintain applications, especially when navigation is sensitive to page content.

The best solution is a technology that navigates the data network and then applies extraction technologies based on the structure (or lack thereof) of the data. An integrated navigation and extraction engine directed by a concise, easy-to-learn notation, provides the precision often lacking in AI methods and eliminates the "clunkiness" of using one technique for extraction and another for navigation.

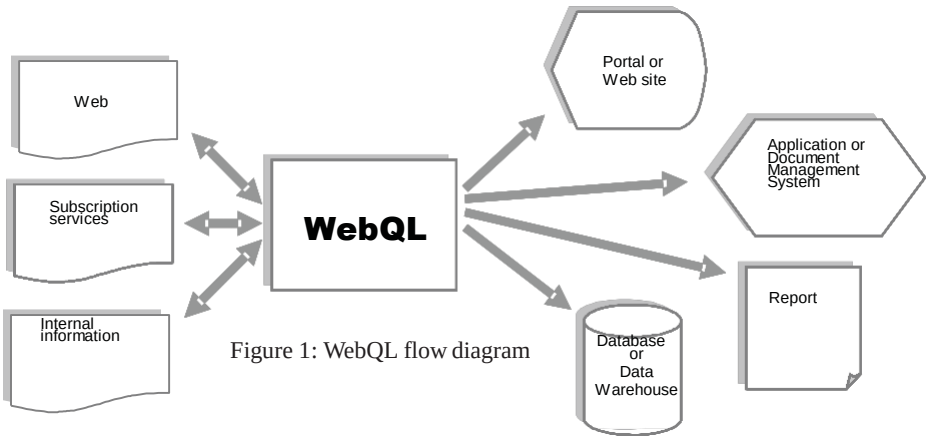

Figure 1: WebQL flow diagram

Is extending SQL the answer?

A comprehensive query language can solve the navigation/extraction puzzle. As an example, WebQL from QL2 Software, uses familiar SQL-like constructs to navigate through enterprise data networks and precisely extract data from a variety of information sources. Once it extracts the data, WebQL automatically reformats it into portals, reports, databases or applications.

Advanced WebQL features

- Concise notation based on familiar ANSI standard SQL including full support for joins, grouping, sorting and set operations
- Fully featured IDE includes syntax highlighting editor, graphical data flow monitor, and real-time delivery of results
- Standalone or server deployments available for Windows, Linux and Unix
- Integrated extraction and navigation techniques simplify the development and maintenance of applications
- Supports all modern technologies for parsing and extracting unstructured, semi-structured and structured data, including a number of advanced proprietary algorithms
- Simultaneous and uniform input and output of all common file formats – HTML, XML, PDF, DOC, CSV, TAB, images, databases, proprietary email formats, etc.
- Automatically employs sophisticated algorithms for page request throttling and parallel processing – no special coding required
- A variety of robust integration options – Web Services, Web browser, command line, GUI, ActiveX and APIs for various languages
- Comprehensive facilities for error trapping and reporting
- "Point-in-time" page archiving via HTML localization or graphical snapshotting of rendered pages
- Transparently supports all Web functionality – script execution, forms handling, cookies, user agents, frames, tables, authentication, etc.
- Query execution scheduling enabling "lights out" operation
- Integrated identity protection options

Similarities between WebQL and ANSI SQL

WebQL is designed to use the SQL language as its base. This allows SQL users to quickly ramp up to using the WebQL language. For example, assume the most basic query a SQL user is likely to initiate after connecting to a new database is:

```
select * from
some_table
```

This query retrieves all records from the database table some_table. WebQL offers two intuitive variants of this query. The first variant is:

```
select * from some_table@some_database
```

This query also retrieves all records from some_table, but WebQL has no inherent tables of its own so the @some_database syntax is necessary to indicate where the table lives.

Now for the second variant:

```
select * from http://www.website.com/
```

This query causes the URL http://www.website.com/ to be loaded, and a default set of columns to be populated with information retrieved from the page. To load all pages from a site instead of just one page, the following query does the trick:

```
select * from crawl of
http://www.website.com/
```

This `from` clause is just one of the clauses that SQL defines within a select statement. The others are `where`, `having`, `order by` (or `sort by`), and `group by`. WebQL supports all of these clauses, each behaving as a SQL user would expect.

Other language elements are shared too. WebQL expressions have all the same elements as expressions in SQL: strings are single quoted, identifiers may be double quoted, NULL values cause NULL results on Boolean operations, the string concatenation operator is ||, etc. Additionally, unions and joins (called connectors in the WebQL vernacular) are present in WebQL, as are column aliases, inner queries, and similar datatypes, including textual, numeric, Boolean and date/time values.

Where WebQL and SQL differ

There are a few important differences between WebQL and SQL due to the inherent nature of the problems each is designed to solve. The most obvious are the language extensions that allow WebQL to extract columnar data from unstructured sources, and route it to multiple arbitrary data sources simultaneously. Others that merit explanation are: JOIN syntax, named select syntax and datatypes.

JOIN syntax

The most noticeable difference is the syntax used to perform join operations. In SQL, tables are joined by selecting data from multiple tables within a single select statement. WebQL syntax requires that joins be expressed

more explicitly, visually similar to a traditional SQL union. For example, consider the following SQL query:

```
select table1.id, table1.col, table2.col
from table1, table2
where table1.id = table2.id
```

The WebQL equivalent of the former would be:

```
select as table1 *
from table1@db
join
select *
from table2@db
where table1.id = table2.id
```

The differences are attributable to the way WebQL accesses data. With traditional SQL, the RDBMS has complete knowledge about data size and form, and uses this information to decide the order data is loaded from tables and joined together. WebQL, on the other hand, has no knowledge of data size or potential response times. When pulling data from a Web site, for example, the load time for each page could vary from two seconds to two minutes, with sizes ranging from a few kilobytes to several megabytes. The syntax above allows the author to explicitly lay out the path data takes as the query executes.

Named Select Syntax

Another language extension demonstrated by the above query is the "named select" syntax. In the example, we use `table1` to name the first select statement. This feature allows a query writer to route output from a single SELECT statement to multiple child SELECT statements. This overcomes a disadvantage of the nested SQL syntax, which becomes a handicap when data retrieval is extremely time intensive. Named Select syntax practically eliminates reliance on temporary tables in WebQL.

Datatypes

The final important difference is the relatively small number of datatypes employed by WebQL. Most of the datatypes defined by the SQL standard are used to convey small differences in physical storage requirements and enforce stored data integrity. Since WebQL has no native data storage facilities, differences such as INTEGER as opposed to SMALLINT are not relevant. Also, WebQL has no need for precision specifications on its datatypes.

Supported File Formats

WebQL is capable of extracting data from virtually any document, image, or tabular format regardless of source. Once extracted the data can be automatically converted into virtually any format including:

- HTML
- XML
- PDF
- CSV, TSV, TXT
- RSS
- ZIP
- Office Files such as Word, Excel, Outlook, PowerPoint
- MIME
- DBF, TAR

WebQL APIs

WebQL provides multiple interfaces for programmatically controlling query execution from within applications or computing environments:

- Java interface for cross platform environments
- SOAP interface for Web Services
- C++ interface for high performance applications
- ActiveX/COM interfaces for the Windows environment
- VBA for the Microsoft Office environment
- .NET

Example: Creating a WebQL query

WebQL runs on Windows 2000 or Windows XP platforms and provides a graphical interface. WebQL queries are created using a fully featured notation editor. Figure 2 displays the concise, SQL-like notation.

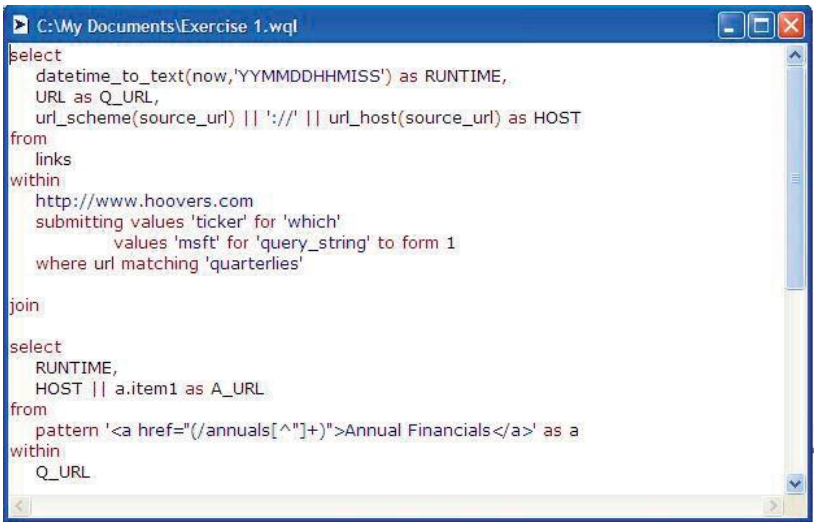

Figure 2: The WebQL Studio query window diagram

Once you complete the query notation, simply click the 'Run' button to execute the query (see Figure 3).

Figure 3: WebQL Studio – Run Query

In the following pages we will review options for viewing and bebugging in the in the WebQL Studio interactive execution window

While the query executes, WebQL displays an interactive execution window with multiple viewing and debugging options. For example, Figure 4 below displays the Messages tab. The lower pane of that window displays run time messages as the query executes. The top pane of that window displays a grid of extracted information in real time.

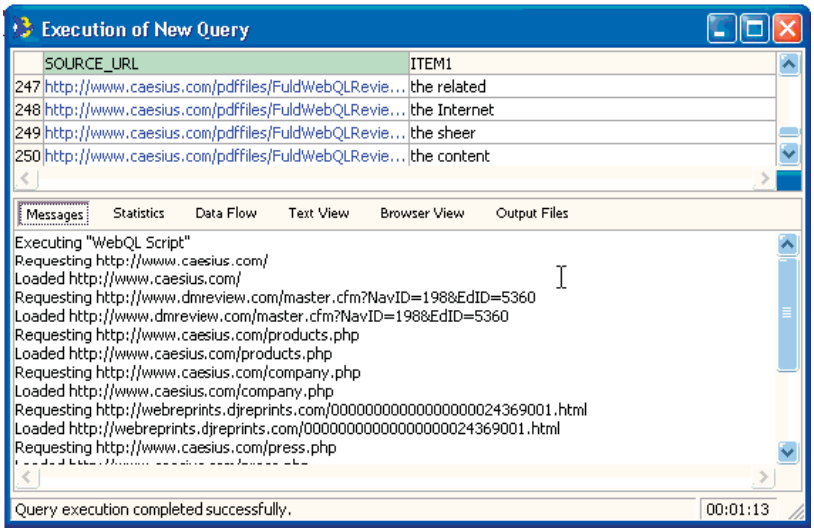

Figure 4: The Messages tab

The Statistics tab (see Figure 5) displays real time summary of page request results and network traffic volume.

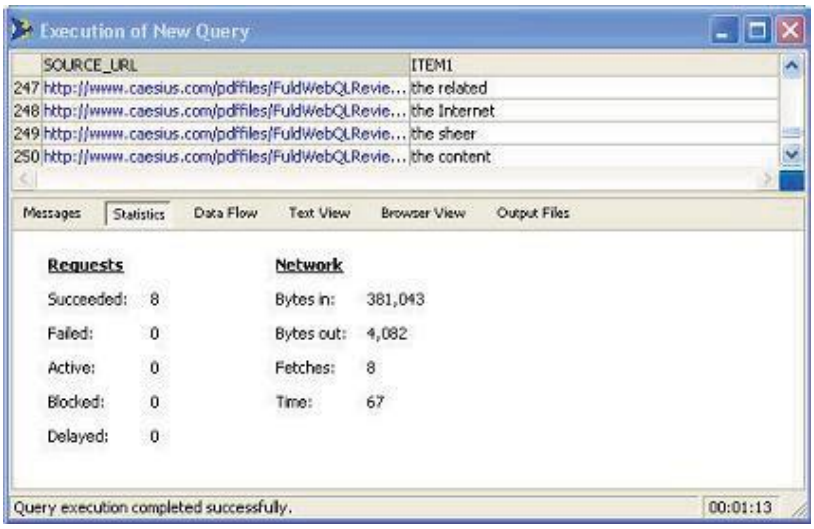

Figure 5: The Statistics tab

The Data Flow tab (Figure 6) displays a graphical flow chart of a query with color coded run time status indicators. Clicking any of the blocks takes you to the part of the query it represents.

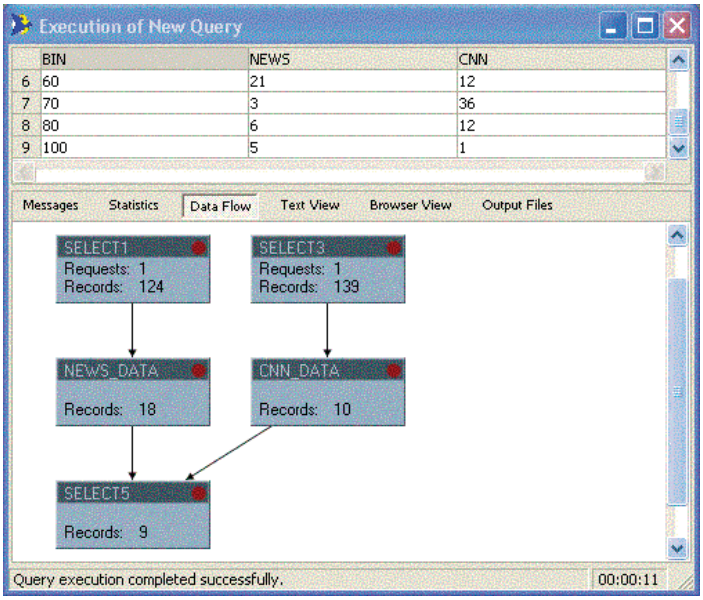

Figure 6: The Data Flow tab

At any time during query execution, the extracted information can be viewed in the lower pane either as text or HTML. In the lower pane of Figure 7 is a browser view of a Web page. To get this view, you click the link in the top window. This link was extracted during the query execution.

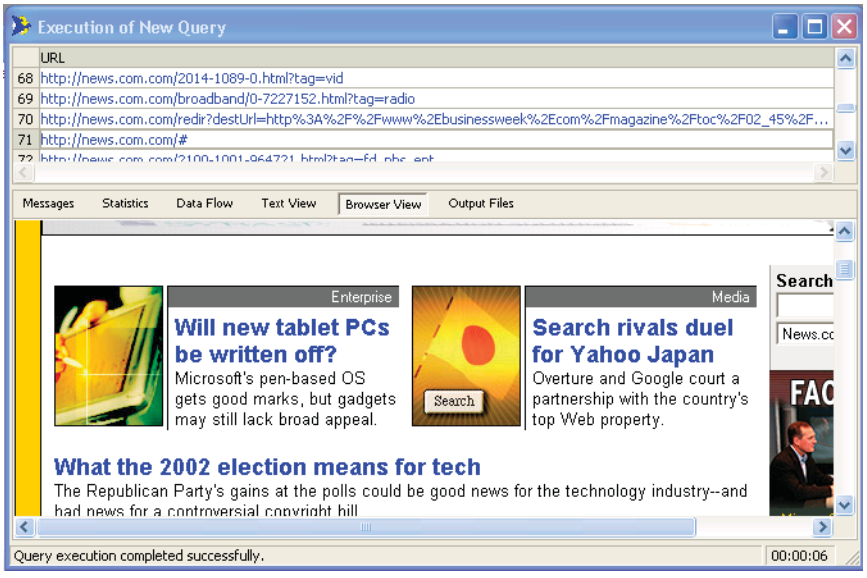

Figure 7: The Browser View tab

There are many format options for the data extracted by WebQL. The Output Files tab (see Figure 8) displays icons representing the output files specified in the query and created by WebQL. Clicking any of the icons displays the output in the specified file format.

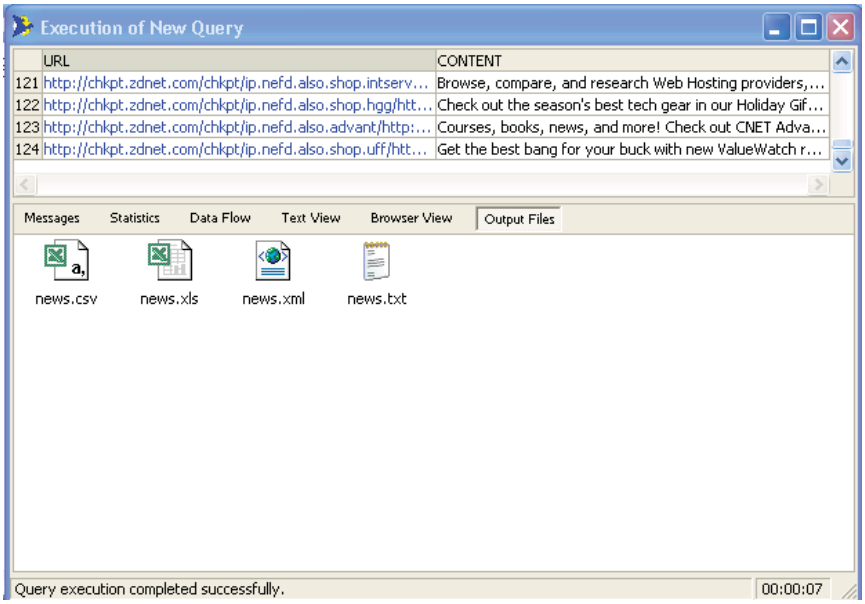

Figure 7: The Output Files tab

Summary

The expected ROI of expensive BI applications cannot be achieved without critical data. Polling your sales force to find out competitor's prices is not effective. Enterprise data networks (including the Web) are likely to contain the data these expensive applications need, but an effective platform is needed to pull it all together and integrate it. Point-and-click IDEs actually tend to penalize expertise, segmenting the task into very small chunks and preventing a holistic view of the solution. This fragmented approach requires even an experienced developer to deal with each small segment of the solution sequentially through the IDE interface rather than confronting the code directly.

Platforms based on SQL-like query languages to solve complex navigation and data extraction issues can deliver the data quantity, timeliness and precision BI applications require.

System Requirements and Platforms

WebQL has a distributable and scalable architecture, which allows it to be run on a single PC or a high performance multi-processor server. The exact hardware and system software platform required by WebQL depends on the information gathering application. The following is a configuration that meets most WebQL requirements:

- Pentium III or faster processor
- 512MB RAM suggested (256MB minimum)
- 25MB available hard-disk space (for installing WebQL only)
- Windows 2000, Windows XP, Linux, SUN Solaris
- High speed Internet connectivity

NOTES:

FOR MORE INFORMATION, PLEASE CONTACT:

QL2 Software, LLC
sales@QL2.com
www.QL2.com