Automated Internet Data-mining - IQL

Cătălin STĂNESCU
Bucharest University of Economic Studies
Department of Economic Informatics and Cybernetics
ROMANIA


Abstract: This article presents the Internet Query Language (IQL) tool – an open-source software for automated data mining, processing, presentation and formatting. It automates complex multi-threaded tasks through a specialized query language adapted to hierarchical data while employing data processing techniques common to the internet medium and familiar to current developers. The tool is modular and it's various components can be employed for other use cases.

Key-Words: IQL, data-mining, data-processing, hierarchy, parallelism

1. Introduction

The internet is the largest source of raw information available. On a daily basis more information is exchanged online than there was total in storage 50 years ago for the entire world, and it is growing at an exponential rate, expected to reach 1 zettabyte by 2015. While it is instantly reachable and free of charge, and despite its huge information potential, working with data from the internet on a large scale has proven challenging. The primary reason for this is the fact that it is presented in formats that are usually unsuited for easy processing. While some attempts have been made to present the information in parser-friendly formats through web services, the results have been limited as work needs to be done for every specific instance. Other attempts include specialized parsers, but these too take resources to make, and only solve one particular resource at a time.

In this paper we present a tool which tries to automate information gathering processes from the world wide web. Since the main area of interest is the browsable internet, which has HTML as its main underlying structure, the software uses a hierarchy as its internal data representation. The interface can be both a command-line and a graphical user interface, and the language is not too dissimilar from SQL, making the learning curve easier for developers.

The retrieval of web-pages is done in parallel, with a heuristic indicator of progress. It can work with any input formats that can be represented in a hierarchical structure, like HTML, XHTML, XML, tables and lists, retrieved from local or remote sources. Processing can be done in a complex manner through the language's commands.

2. Hierarchical Structures

Most data on the internet is represented in some form of markup language - normal text complemented by semantic and stylistic information. The most ubiquitous one is the Hyper Text Markup Language. Virtually all websites on the internet use it to send information to browsers.

The information provided by the markup language itself can be just as important as the actual information it complements. It is therefore important in any processing of such data for the hierarchical and attributes information to be not only preserved but also usable. The internal representation is HTML-compatible and stores everything that could be used to transform the data. On a normal web page, DOM transformations are normally done through a scripting language like Javascript, and more recently by Javascript frameworks like jQuery. The reason for the popularity of jQuery is its ease of use, and especially its CSS-like selectors, through its Sizzle engine,
which is perfect for easily identifying elements of interest in a hierarchical model. A specialized version of this selector is implemented directly in the IQL language, with small modifications to also allow for manipulation of text-nodes.

2.1. HTML

HyperText Markup Language is a markup language recommended by the Web Consortium used to create web pages that can be shown inside a navigator such as a browser. The purpose of the language is primarily for presentation of content – paragraphs, fonts, tables, etc. – than to describe the document semantics. A valid HTML document should start with the DOCTYPE declaration indicating the HTML version of the document, as well as the desired backwards compatibility level. The data type markup is done using HTML tags. This varies according to the HTML version used, some being deprecated as they were emulating styling functionality which has since become the job of the CSS language, and other being added to expand the semantic range of the language. The textual content of the document is added between the tags, taking care to escape the special HTML characters into entities. The same principle applies to the attribute values. Between certain tags this rule may be ignored, such as the <script>, <style> and <textarea> elements, which expect content of type CDATA. This type of content can also be introduced using the <![CDATA[ ]]> tags, which is a convenient way of adding a lot of text without the need to escape all special characters. The special characters are: “<”, “>”, “&”, ‘<’ and ‘>’. The language is case-insensitive.

2.2. XML

Extensible Markup Language is a meta-language recommended by the Web Consortium for transmitting hierarchical information, or creating other markup languages such as XHTML, RDF, RSS, MathML, SVG, OWL, etc. Files in XML format are often used by web servers through languages like PHP, ASP, Python, Perl or more recently Javascript (through Node.js) to store and transmit information between web services – such as WSDL, or between a web service and users – such as RSS. The special characters in XML are identical to the ones used in HTML. The presence of these characters in an invalid context is however not allowed, resulting in an error being displayed. The language is also case-sensitive. Just like HTML can have different document types through the use of DTD’s, so can XML have separate meta-documents describing it’s structure and formalizing the rule-based restrictions or structure models.

3. Query language parser

The IQL syntax parser was built using an Abstract Syntax Walker (ASW). Unlike most other language interpreters like Abstract Syntax Parsers (ASP) which are generally context agnostic, ASW emulates roads in its interpretation methodology. This allows for syntactic definitions like “steps”, “forks”, “lateral” (optional paths) and “loops” which require a single syntax definition, where ASP’s generally require multiple redundant syntax definitions placed in multiple places of the specialized parser. ASW also allows for context-sensitive interpretation. Take for example the “+” token – it can be interpreted only as an operator in an AST, while it may actually have different interpretations depending on the context in which it is used. It is generic and modular, allowing for the “walking” code o be completely separated from the language syntax definitions, and can be used to parse other languages like C, C++, Python, Perl, Haskell, etc. The language definitions are stored in objects that can also be used for storing the data of the parsed code within themselves, allowing for completely atomic structural sections of the syntax, with minimal redundancies.
4. Extracting information from generated structures

The query is the primary means of describing the actions to undertake to the program. It is therefore important that the syntax allow for power of expression while maintaining the relevant syntactic rules and validations of logical operations.

The IQL syntax structure is similar to other query languages like SQL in that it is a special-purpose programming language designed for managing structured data. Unlike standard query languages like MySQL, PostgreSQL and MSSQL where the clauses have rigid ordering, here the clauses may take any order as they are independent from each other. The reason for this difference is that it is better suited for working with hierarchical data as opposed to the tabular nature of other database management systems.

Most types, operators and functions follow the SQL standard. Operators follow expected precedence rules. Types of operators by parity are:

- **unary**, dealing with a single operand. Depending on its positioning it can be:
  - left unary (or lunary), to the left of the operand, ex. "-", "!", "NOT"
  - right unary (or runary), placed to the right of the operand, ex. "IS NULL";

- **binary**, dealing with two operands, ex. "=". ":=". "LIKE", the range operator "..";

- **ternary**, dealing with three operands, ex. "... BETWEEN ... AND ...";

The available operators are in descending order of precedence are:

- `..` is the binary range operator;
- `%` allows formatting text according to patterns;
- `+` and `-` are either binary or left unary operators, depending on context;
- `*` and `/` are binary math operators
- `||` concatenates a list of expressions, producing a string;
- `<>` is the NULL-safe comparison operator;
- `>`, `<`, `>=`, `<=`, `=`, `!=` compare expressions;
- `|=` returns true if the values are equal, or the latter begins with the former followed by a dash;
- `*=` returns true if the previous value contains the latter;
- `~=` returns true if the previous value is contained in the white-space delimited list of the latter;
- `$=` returns true if the previous value ends with the latter;
- `^=` returns true if the previous value starts with the latter;
- LIKE compares the value using a wildcard;
- REGEXP compares the value using a regular expression;
- REGEXP_MATCH does a regular expression match, and returns hierarchical nodes containing the groups and captures defined in the expression;
- REGEXP_MATCH_ALL does the same as REGEXP_MATCH but doesn't stop after the first match;
- REPLACE replaces the given value with the latter within the expression;
- URI_BASE transforms a relative web address into an absolute one using the latter value as a base;
- IS [NOT] NULL returns true if the given value is, and respectively is not, NULL;
- URI_FORMAT is a special syntax operator that formats a valid URI according to the given values;
- BETWEEN returns true if the numerical value lies between two given values inclusively;
- NOT negates the given expression;
- AND with `&&` returns the boolean AND result of the given values;
- XOR returns the boolean XOR result of the given values;
- OR with `||` returns the boolean OR result of the given values;
- `:=` sets the variable or attribute value;

Functions can be deterministic or non-deterministic. By the type of input they expect they can be:
simple, which takes a simple input, ex. “IF”, “SUBSTR”, “CASE”;
aggregate, which take lists as input and produce a single output value; these can receive any type including subqueries as parameters, ex. “AVG”, “MIN”, “COUNT”;
The available functions are:
- AVG returns the aggregated average of the passed values;
- CASE is a special-syntax function that executes a series of predicate comparisons for selecting a given value;
- COUNT returns the number of sub-values from the passed aggregate value;
- DATA is a specialized non-standard function that works with different clauses to insert special values of error or meta-data;
- DATE creates a date from a given value;
- DATEDIFF returns the time-span between two given dates;
- DOMAIN returns the domain from a valid URL address;
- IF evaluates the first parameter and returns the second parameter if true, the third one otherwise;
- MIN returns the minimum aggregate value;
- MAX returns the minimum aggregate value;
- RAND is a non-deterministic function that returns a random number between 0 inclusively and 1 exclusively;
- SUBSTR and SUBSTRING return a string sub-selection;
- TEXT returns the textual value of a given expression;
- TRIM returns the given string with the edge white-space eliminated.
Non-standard types include Regex for pattern matching strings, and the input-type “?”. The commands available are:
- DELETE removes a node from the hierarchy;
- EXTRACT selects nodes for subsequent actions (original left in place);
- GET retrieve a remote resource using the GET HTTP command;
- GROUP BY group nodes according to multiple criteria (can work on multiple branch levels);
- INSERT moves nodes between parents;
- LIMIT selects nodes based on their position within their parent (can work on multiple branch levels);
- MAKE creates nodes based on the selector format;
- POST retrieve a remote resource using the POST HTTP command;
- ORDER BY sort nodes according to multiple criteria, or shuffle (can work on multiple branch levels);
- OUTPUT provides options for outputting data in different formats (such as options to minify or prettify);
- READ retrieve a local resource;
- SELECT selects nodes for subsequent actions (original is left in place);
- SET sets node attribute values or local and global variables;
- WITH and WITHIN switch the command context to (and respectively within) a specific selection of nodes for further processing.
Additionally, “BEGIN” and “END” tokens can be used to specify a series of commands for a subquery. The syntax parser is modular and can easily be extended.
Comparison is done in a way similar to the comparison method used in languages like SQL. If one of the values is NULL, and the comparison isn't done with a special null-aware operator like <=>, the result will be NULL. Otherwise the two values will be compared using a “natural” comparison algorithm which will result in the expected sorting for numbers, text or combinations of these two.
The selector follows the CSS selector model, also used by jQuery's Sizzle engine. The tokens used are:
- non-prefixed tokens for tag names: “div”, “section”;
- dot-prefixed for classes: “.myclass”;
- number-sign-prefix for id's: “#myid”;
- colon-prefix for pseudo-classes: “:button”, “:checked”, “:first-child”. Certain pseudo-classes may contain
special syntax, ex. “:nth-child(2)” and even subqueries, ex. ”:not(:first OR :last)”;
- bracket notation for attribute checks and sub-selectors:
  - “[div.red]” has to contain a descendant of type “div” with class “red”;
  - “a[@href ^= ‘mailto:’]” will finds email links;
- special tokens like “*” for any element, “**” for any node, “*text*” for text-nodes and “*comment*” for comment nodes (the last three are extensions to the classic SQL selector).

Query execution can take a long time, depending on factors such as network latency and query path. Execution time is hard to predict, so a heuristic approach is taken to estimate progress using factors like download elements count, individual size (when available) and download concurrency.

While the download of data is done on multiple threads, the manipulation of in-memory structures is only allowed through shared memory locks, preventing race conditions.

The main use of the tool is from the command-line, however it can also easily be used from other projects through its classes. Object values can be passed through the special input-type “?” which can also have a specified index “?4” or name “?myvar”.

Variables are objects separate from the main hierarchy. They may be specific to the current session (local), or global. Session variables can be accessed with “LOCAL”, “SESSION” or “$” prefixes, and globals with “GLOBAL” or “$$” prefix.

Often data may contain special non-alphanumeric characters. In this case the grave accent “`” may be used to escape data. For example, to select an HTML page’s doctype element, we can run “SELECT `!doctype` ;” while a named-input may be accessed with “?” “my named input” . This also works for variables, selectors, and other tokens.

Prior examples should have highlighted a certain ambiguity in the syntax. For example what is “+”, is it addition or is it a selector-descendant-relation? The answer is that it depends on the context. The parser doesn’t interpret tokens blindly like in most parsers, but rather it interprets them on-the-fly depending on the current parsing context. For example, “SELECT 1+2;” will return 3, while “MAKE a[b,c] SELECT b+c;” will select the c element. This allows operators, functions and selectors to have the same names without the parser getting confused.

5. Experimental data

For illustration purposes, getting the current BNR top exchange rates is made by using the following expression with the results presented in figure 1.

```
GET 'http://www.bnr.ro/nbrfxrates.xml'
SELECT rate ORDER DESC
```

![Figure 1. Rate request results](image1)

Getting historical quotes from yahoo’s RSS feed is made using the following expression with the results presented in figure 2.

```
GET 'http://rss.groups.yahoo.com/group/famous-quotes-/rss'
SELECT item
```

![Figure 2. Quotes request results](image2)

The output above contains many quotes, with many from the same people. The data is stored in memory and is available for further processing. In the following expression we illustrate how we can further group the items from the results of the previous expression using
the GROUP command, with the results presented in figure 3.

```sql
GROUP BY text(`dc:creator`) AS author
[@name=max(text(`dc:creator`))]
[SELECT text(description)]
```

Figure 3. Group command results

The above examples were run using the command-line version of the tool. There is also a Graphical User Interface version that is more user-friendly, and also contains additional features. The query can be inserted into the top half of the window. It has syntax highlighting specific to the IQL language, as well as keyword auto-completion. The context menu contains a query history list.

The bottom half is the preview pane, and shows the results data in various formats. On the left the hierarchical data is shown on a node basis, with highlighted syntax and a context-menu with formatting options. The right side shows a preview of how the data would look as rendered inside a browser. It has two modes – HTML and XML. The middle row contains controls for the output settings, program execution, as well as the data history options.

To get Bill Gates' two latest twitter updates, we can run the query as shown in figure 4, producing the results shown.

Figure 4. Query example using the GUI

The output may also be shown as it would render inside a browser. Fig. 5 shows a simple web page retrieval with minor modifications. A base element is created into the head of the page to insure relative links within the page still work as expected. Another modification made to the page is that results links are prepended with a bit of text to illustrate the ease with which such modifications can be made.

Figure 5. Query results with HTML output rendering

Xkcd.com is one of the most popular online comics, offering funny and informative images on various subjects like programming, technology and mathematics. If we wanted to extract a list of all images on their server we could do so without much effort. Pages containing the images are numbered
sequentially, however the images themselves is unpredictable and varies from one to the next.

In Fig. 6 we generate all the page addresses using the “..” operator, then we extract the images from each page. It's worth pointing out that although there are 100 images being downloaded in parallel, there is an imposed limit of 10 concurrent downloads per server in order to prevent accidentally overloading it and in effect resulting in a DoS attack. It should also be noted that not only does the query run in a relatively quick time-frame (under 30 seconds for this case), but all the meta-information is kept such as the image tooltip which is an integral part of all xkcd comics.

We select the state links with their names and links. Then within each link we get the page and extract relevant data like population count and surface. Since we want the results sorted, we use the ORDER clause, which has two syntax modes. In this case we use the form which allows simultaneous sorting of descendants within their common parent at multiple levels. Although the sorting uses textual data, the sorting is done correctly since the comparison is not done in a binary mode, but using “natural” comparison. Finally, we create the table itself, preceded by a bit of styling information for presentational purposes.

If we wanted to get a list of Romania’s states in descending order of population count, we could use http://judetel.rou.ro which contains a list of them on their front page. However data about them, like population count and surface are each contained in individual pages for each state.

In figure 7 we illustrate an example of a query that can achieve that. First we define a variable to contain the URL for the resource we will use to retrieve the data. The reason for this is that the URL is used in multiple places – once to retrieve the initial page containing the list of states, then again to make the individual state links absolute.

To illustrate cross-page downloads, the following statement retrieves the first page then downloads the next pages in parallel. The results are presented in figure 8:

```
GET 'http://www.gamespot.com/games.html?platform=5&mode=all&sort=sco re'
```
To achieve the same results as the simple examples above, a custom crawler would have to be built which takes time, testing and is often hard to get right.

6. Conclusions

As the amount of information stored online is increasing at an exponential rate, businesses and organizations dealing with it need efficient ways to work with it. Sharing, extracting and manipulating data is becoming an integral part of their regular activity. Therefore, the need to deal with large quantities of data in a format that's not always friendly to standard parsers will only increase over time.

Many companies including Wolfram Research (1), True Knowledge (2) and Google (3) have attempted to extrapolate data from the internet for easy processing and querying, but they suffer from shortcomings like limited data-coverage, inexact results or limited querying capabilities.

The most capable tool build so far is IBM’s Watson (4), which is able to understand human language to a high degree of accuracy, and parse huge amounts of data to produce a meaningful answer. However with requirements more similar to a supercomputer than a normal PC, it’s still a long way from becoming widely available.

The amount of information is growing exponentially, and with over 300 exabytes (5) of data on the cloud finding a way to make it readily accessible to automation is crucial.

Notes

(1) Wolfram Alpha is an answer-engine developed by Wolfram Research. It is an online service that answers factual queries directly by computing the answer from structured data, rather than providing a list of documents or web pages that might contain the answer as a search engine might. http://www.wolframalpha.com

(2) True Knowledge aimed to directly answer questions posed in plain English text, which is accomplished using a database of discrete facts. http://www.trueknowledge.com

(3) Google Squared was an information extraction and relationship extraction product from Google. http://en.wikipedia.org/wiki/Google_Squared

(4) Watson is an artificial intelligence computer system capable of answering questions posed in natural language http://en.wikipedia.org/wiki/Watson_(computer)


References


[4] Ivan Ion, Niculescu Silviu, Catalin Boja, Clonarea bazelor de date, Revista Română de Informatica si Automatica, Bucureşti