Squeal: SQL Access to Information on the Web


Ellen Spertus
Dept. of Mathematics and Computer Science
Mills College
5000 MacArthur Blvd.
Oakland, CA 94613
spertus@mills.edu

The World-Wide Web contains an abundance of semi-structured information, including hypertext links between pages, structure within hypertext pages, and structure within the addresses of pages (URLs). Because of the difficulty in harnessing this structural information, many Web tools fail to make use of it, instead treating the Web as though it were a flat text collection. We introduce Squeal, a SQL-like language for querying the Web as though it were in a relational database. We describe simple but powerful applications built on top of Squeal, which we call "ParaSites" because they make effective use of this underutilized information on the Web, often in ways unintended by the information's authors.

Squeal
Relational databases provide a powerful abstraction for manipulating data. The division of data into relations helps users comprehend the data and allows them to specify queries in Structured Query Language (SQL). While the Web is too large and quickly-changing to be stored in a relational database, it is useful to provide users with the illusion that it is. We provide this through the Squeal language and interpreter, which allows users to make SQL queries involving Web information. The Squeal interpreter determines what information needs to be fetched and parsed from the Web in order to answer the question. The retrieved information is cached in a local database in case it is needed again.

The Squeal schema includes relations for representing the following types of Web information:

- the contents of a page
- hypertext links between pages
- different URL strings corresponding to the same page
- the parsed version of a URL
- tags and attributes appearing on a page
- the header and list hierarchy on a page
- if a search engine reports that a page contains a specified string
- if a search engine reports that a page contains a specified link

For example, hyperlinks are expressed through the link table, which expresses a relation between a source URL, anchor text, and a destination URL. In order to ask what pages are pointed to by the MIT AI Lab home page, the user would enter:

```
SELECT destination FROM link
WHERE source = "www.ai.mit.edu"
```

The Squeal interpreter would retrieve the page, parse it, then return a list of the destinations of links on the page. Instead of (or in addition to) specifying the source page, the user could provide another field. For example, to ask for pages containing hyperlinks with "artificial intelligence" as anchor text, the user would enter:

```
SELECT source FROM link
WHERE anchor = "artificial intelligence"
```

To answer this, the Squeal interpreter would ask a search engine, such as AltaVista, what pages contain "artificial intelligence" as anchor text. Because no such engine can ensure returning all such pages, the list is likely to be incomplete. The Squeal interpreter then retrieves and examines the pages, returning to the user a list of pages that it has verified as having the desired anchor text. While recall will be less than 1, the precision is guaranteed to be 1. A similar process is used to request pages that link to both "www.ai.mit.edu" and "www.lcs.mit.edu":

```
SELECT L1.source FROM link L1, L2
WHERE L1.destination = "www.ai.mit.edu"
AND L2.destination = "www.lcs.mit.edu"
AND L1.source = L2.source
```

Queries involving multiple Squeal- and user-defined tables are also possible and can be found in the appendix.

ParaSites
We have built a number of ParaSites on top of Squeal that exploit the Web's structural information, including a personal home page finder, which uses the following algorithm to try finding the home page of the person with name N:
1. Generate a list of candidate pages from the destinations of hyperlinks whose anchor text is N.

2. Give bonus points to candidate pages where one or more of the following conditions hold:
   - The full name appears within title or header tags (e.g., "<title>Lynn Stein's home page</title>"
   - The URL is of the form "... foo/foo.html" (e.g., "/users/las/las.html"
   - The final directory name in the URL starts with a tilde (e.g., "~las/home.html"

This algorithm, with additional heuristics described elsewhere, is quite effective.

The contrast between text-based and structure-based approaches can be seen most directly in different recommender systems for Web pages. With either approach, a user specifies seed pages. A text-based system, such as Excite, then searches for pages that contain the same words, and return these pages to the user. The ParaSite approach is to find parent pages that point to the seed URLs, then to return the pages pointed to most frequently by the parents, i.e., the seed pages' siblings. The underlying philosophy is that human beings are best at deciding what pages are alike, so we should associate pages with each other if they co-occur as destinations of hyperlinks on multiple pages. This is entirely analogous to mining citation indexes. A small user study showed the ParaSite approach to be superior in some ways to the purely text-based approach. We predict the best system would combine both approaches.

**Conclusions**

We have provided a taste of the Squeal programming system for accessing structural information on the Web and have outlined some effective applications that can be easily written in Squeal, demonstrating the utility of the Web's structural information and how Web tools, such as AltaVista, can be harnessed.

**Bibliography**


**Appendix**

This is a slightly-simplified version of the code for a program to find and display possible home pages for someone with name name. SQL keywords are in capital letters, and Squeal tables are in bold face.

```
CREATE TABLE candidate (url URL, score INT);

// Generate candidates that are the destinations of hyperlinks whose anchor text is name, giving each of these pages 2 points.
INSERT INTO candidate (url, score)
SELECT L.destination, 2
FROM link L
WHERE L.anchor = name;

// Add 1 point to all of the candidate pages in which name appears as an attribute value for any tag.
INSERT INTO candidate(url, score)
SELECT c.url, 1
FROM candidate c, tag t, attribute a
WHERE t.url = u
AND a.tag_id = t.tag_id
AND a.value = name;

// Add 3 points for pages named 'home.html" or 'home.htm".
INSERT INTO candidate(url, score)
SELECT DISTINCT c.url
FROM candidate c, parse p
WHERE c.url = p.url
AND p.depth = 1
AND p.value LIKE 'home.htm%';

// Display results, best first
SELECT c.url, SUM(c.score) AS total
FROM candidate c
GROUP BY c.url
ORDER BY total DESC;
```
Abstract

The overwhelming amount of information sources now available through the internet has increased the need to combine or integrate the data retrieved from these sources in an intelligent and efficient manner. A desirable approach for information integration would be to have a single interface, like the SIMS information broker [1], which allows access to multiple information sources. An example application is to retrieve all the menus of restaurants from Joe's Favorite Restaurants site which have been rated highly by the Department of Health. This task, if performed manually, would require a significant amount of work for the user. An information broker, like SIMS, would allow access to multiple information sources, abstracting away the need for the user to know the location or query access methods of any particular source. SIMS stores knowledge about the data contained in each of these sources, as well as the relationships between the sources in the form of a model, called a domain model. The first step in creating the domain model is to determine which data instances appear in multiple sources, e.g. which restaurants from Joe's web site, like “Art's Deli,” also appear on the Health Department’s site. Once the common data instances are determined, the relationship between the data in the sources can be modeled in the domain model as subset, superset, equality, or overlapping.

A special case for information integration is when data instances can exist in inconsistent formats across several sources, e.g. the restaurant “Art's Deli” can appear as “Art's Delicatessen” in another source. For the integration process each source can be seen as a relation; therefore, integrating the sources requires performing a join on the two relations by comparing the instances of the primary keys. Since the data instances in the sources are represented in inconsistent formats they can not be compared using equality, but must be judged according to similarity. To determine the similarity between strings we developed general domain-independent transformation rules to recognize transformations like substring, acronym, and abbreviation. For example, “Deli” is a substring transformation of “Delicatessen.” A probabilistic similarity measure is calculated for each of the transformations between the strings of the two data instances; and then they each are combined to become the similarity measure of the data instances for that attribute. When comparing the data instance “Art's Deli” with “Art's Delicatessen,” the probabilities are calculated for the string transformations of “Art's” to “Art’s” and “Deli” to “Delicatessen.” These probabilities are combined to be the similarity measure for the two instances. After the similarity measures are determined for each of the attributes, Name, Address and Phone, then they are combined to measure the similarity.
for the two tuples. The most probable matching between
the two sets of tuples is then determined. We are
employing a statistical learning technique in order to
iteratively refine the initial probability measures to
increase the accuracy of the matches. Once the mapping
is known then a mapping table or function can be created
using the instances from the primary key attribute.

Some related work has been conducted by Huang & Russell
[2] on matching tuples across relations using a
probabilistic appearance model. Their approach also
incorporates the idea of comparing the tuples based on all
of the shared attributes. To determine similarity
between two instances, they calculate the probability that
given one instance it will appear like the second instance
in the other relation. Calculating these probabilities
requires a training set of correctly paired tuples (like the
tuples in the figure). Unfortunately, appearance
probabilities will not be helpful for an attribute with a
unique set of instances, like Restaurant Name. Since
“Art’s Deli” only occurs once in the set of instances,
knowing that it appears like “Art’s Delicatessen” does not
help in matching any other instances. In our approach
the initial probability measures are calculated for the
strings of instances that match using a specific
transformation rule; therefore, having matched “Art’s
Deli” with “Art’s Delicatessen” will increase the
probability of “Deli” as a substring of “Delicatessen.”
This will increase the probability for other instances
which use that specific transformation.

Other related work by Cohen [3] determines the
mappings by using the IR vector space model to perform
similarity joins on the primary key. In this work
stemming is used to measure similarity between strings;
therefore, in the figure “CPK” would not match
“California Pizza Kitchen.” But, if the values from the
other attributes were taken into consideration, the tuples
would match. In experiments where an entire tuple is
treated as one attribute accuracy was reduced. As
illustrated by the third example, “The Beverly” from Joe’s
site would equally match the tuples for “Art’s
Delicatessen” and “Cafe Beverly.” Our approach would
be able to handle all of these examples, because it
combines all of the measurements calculated individually
for each shared attributes to determine the correct
mapping.

References

Information Mediator. Advanced Planning Technology,
editor, Austin Tate, AAAI Press, Menlo Park, CA, 1996.

Identification in Bayesian Context. Proceedings of

structured information sources containing text. The
SIGIR-97 Workshop on Networked Information Retrieval,
1997.