ParaSite: Mining the Structural Information on the World-Wide Web

by

Ellen Spertus

S.B., Computer Science and Engineering, MIT (1990)
S.M., Electrical Engineering and Computer Science, MIT (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1998

Author .....
Department of Electrical Engineering and Computer Science
February 6, 1998

Certified by .....
Lynn Andrea Stein
Class Of 1957 Career Development Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

ParaSite: Mining the Structural Information on the World-Wide Web

by
Ellen Spertus

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The World-Wide Web is potentially the world's largest knowledge base but only if new information retrieval techniques are developed to take advantage of its unique characteristics, particularly the semi-structured information within pages, across pages, and in page names. Because these types of structure are represented in such different ways, a large number of specialized tools have been required to gather structural information. I provide a relational database interface to the Web called Squeal, which encapsulates these different types of structure in a uniform manner, allowing the user to query the Web in Structured Query Language (SQL) as if it were a database. A novel "just-in-time" interpreter automatically retrieves information from the Web as implicitly demanded by user queries, a technique which could be applied not just to the Internet but to other sources of data too large to be precomputed into a database. The level of abstraction provided by Squeal allows the user to easily create agents that make full use of the previously-untapped information on the Web. One such "ParaSite" is a simple structure-based recommender system that compares favorably to the best text-based system.

Thesis Supervisor: Lynn Andrea Stein
Title: Class Of 1957 Career Development Associate Professor

Acknowledgments

The University of Washington (UW) was my academic home away from home for the last two years. I am grateful to Oren Etzioni and Dan Weld for taking me into their Internet Softbot group. Other UW folk who made me feel at home were Alan Borning, Lauren Bricker, Steve Hanks, Frankye Jones, David (Pardo) Keppel, Ed Lazowska, Sean Sandys, my office-mates Marc Fiuczynski, Jack Lo, Brendan Mumey, Kurt Partridge, and Xiaohan Qin, and of course Keith Golden.

My work benefited greatly from interaction at UW with Oren Etzioni, Marc Friedman, Keith Golden, Nick Kushmerick, Tessa Lau, Marc Langheinrich, Greg Lauckhart, Alon Levy, Neal Lesh, Greg Linden, Kurt Partridge, Mike Perkowitz, Rich Segal, Erik Selberg, Jonathan Shakes, and Stephen Soderland of the University of Washington. I'm particularly grateful to Rich for badgering me into realizing that the user need not explicitly request transfers from the Web to the SQL database.

Alberto Mendelzon, Gustavo Arocena, and George Mihaila of the University of Toronto have generously shared their time, code, and expertise. Their WebSQL system was a major influence on this work.

I also benefitted from code made freely available by Arthur Do (HtmlStreamTokenizer [8]), Santeri Paavolainen (MD5 [30]), Original Reusable Objects, Inc. (OROMatcher [29]), and Sriram Sankar, Sreenivasa Viswanadha, and Rob Duncan (JavaCC [36]).

This thesis is the culmination of many years as an MIT student. Faculty and staff who were especially helpful and encouraging over the years include Bill Dally, who introduced me to research and supervised my bachelor's and master's theses; John Guttag, for repeatedly helping me beyond the call of duty; Tom Knight, who always encouraged me with even my most random ideas; Marilyn Pierce, for her helpfulness and warmth; Jerry Sussman, who is MIT incarnate; and Bill Weihl, whom I could always go to for excellent advice. Other people who encouraged me over the years were Judy Goldsmith of the University of Kentucky and David Lewis of AT&T Research.

From the beginning of freshman year through the completion of three theses, Nate Osgood has provided me with invaluable emotional and intellectual support. I am privileged to have worked with such a wonderful person. I always enjoyed staying with him, Carol Collura and Norm Margolus, Kathy Knobe, my sister Andrea Spertus, and Lisa and Greg Tucker-Kellogg on my visits to MIT.

Philip Greenspun provided me with a much-needed education about databases. Other MIT folk with whom I had valuable discussions about my thesis include Michael Ernst (now at UW), David Karger, Brian LaMacchia (now at Microsoft), Richard Lethin (now at Equator Technologies Consulting), Henry Lieberman and Jim Miller.

I was generously funded by the National Science Foundation for three years of my graduate study and by the Intel Foundation for one particularly crucial year.

I am very grateful to my committee: Lynn Andrea Stein, Ken Haase, Tom Knight, and Pattie Maes. I was privileged to have such a strong and diverse committee, with Lynn's strength in AI, Ken's and Tom's broad backgrounds, Pattie's expertise with agents and collaborative filtering, and all of their interests in novel approaches to information retrieval. I also appreciate help from their assistants: Marie Lamb, Agnieszka Meyro, and Annika Pfluger.

I could not have done this without the support of my family. I am particularly grateful to my father and brother for stimulating my interest in computers and math.

I am not sure I could have done this without the tremendous support and assistance from my advisor Lynn Stein and my fiancé Keith Golden. Lynn took me on as a student under non-optimal circumstances and stuck with me despite the geographic obstacles and the many other demands on her time. She challenged me at just the right level, treating my ideas and work with respect but demanding that I then go and fully develop them. It is a rare teacher who manges to be both rigorous and compassionate. These meant even more to me than her vital intellectual contribution.

Keith Golden helped me in more ways than I could list, including discussing research with me, teaching me about AI, and showing me by example that a thesis could be completed gracefully and thoroughly, but most of all by loving me and believing in me. I feel that I'll be able to do anything I want with him at my side and have a lot of fun too.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World-Wide Web contains hundreds of millions of pages of data. Existing information retrieval techniques are inadequate for navigating this unprecedented cornucopia of semistructured information. Turning the Web into "the world's largest knowledge-base" has been proposed as one of the most important challenges in artificial intelligence [27]. This is the problem that I address.

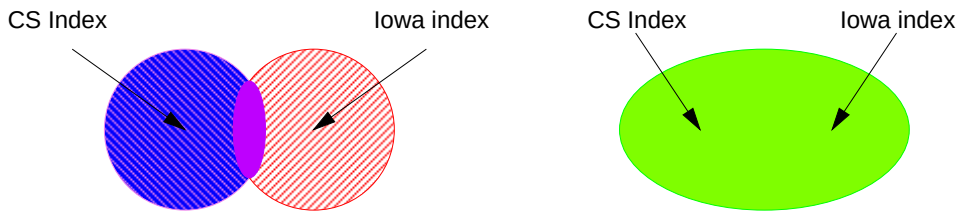
The success of the Web is a testament to the importance of *structure*. Users of the Web make use of both intra-document structure (via annotations indicating headers, lists, and formatting directives) and inter-document structure (hyperlinks). Even the uniform resource locator (URL) identifying a page is structured: typically a host name followed by a location in the file hierarchy. All of these types of information are used by human readers but largely ignored by automated Web tools. I argue that this information is potentially valuable to automated tools and show the feasibility of making use of structural information through a system that allows easy creation of such applications.

While information retrieval has been studied for decades, new techniques are needed because the domains examined have been very different from the Web, both quantitatively and qualitatively. The most studied domain is "flat" text collections, i.e., sets of articles or documents, each of which is a sequence of unannotated text. Most current Web tools are based on information retrieval techniques developed for flat text collections, ignoring the Web's structural information. Similarly, the Web is different from "classical" hypertext applications, such as an encyclopedia available on CD-ROM, which is static and authored by a single team, as will be discussed in more detail in the next chapter. Because of the novelty of the Web, new information retrieval paradigms are needed.

The solution I present is viewing the Web as a giant database, where structural relations are represented as database relations, which provides a uniform framework for accessing the diverse types of Web structure. This allows the construction of powerful "ParaSites", which mine and apply information available on the Web in ways unanticipated by the information's authors. Specifically, this dissertation provides the following:

1. an ontology specifying types of structural information to extract from the Web and a corresponding relational database schema
2. a set of algorithms for representing an unbounded set of information in a finite database
3. a system, Squeal, that implements the abstraction of the Web as database
4. a set of ParaSite applications, to show the value of tapping this underutilized structural information

In this introduction, I motivate the work by describing some ParaSites that can be easily built on top of Squeal.



(a) Intersecting circles **(b) An ellipse**

Figure 1-1: A geometric representation of material related to computer science and Iowa

1.1 ParaSites

Human beings, not machines, are the ultimate authorities on intelligent information retrieval. Machines just have the advantage of being able to logically process data more quickly than human beings can. The best approach to information retrieval (or any problem) is to assign to the computer and the human being the parts of the task that each is best at. Specifically, the computer should execute an algorithm that reduces a new task to a set of problems previously solved and documented by human beings. In other words, the computer should act as a parasite (ParaSite). Hence, my approach is to provide a system that breaks a task down into a set of queries on stored human knowledge (the Web) and puts them back together to produce an answer. The rest of this section describes some useful ParaSites that show the value of being able to access the Web's structural information.

1.1.1 Link geometry

Consider the set of pages within three hyperlinks of a computer science index. The pages within one link of the index are almost certainly related to computer science; the pages two or three links out have a lower probability, although still greater than that of random pages on the web. We can think of the set of pages within n links of an index I as being the interior of a circle (hypersphere) of radius n with center I. We could create two such circles with centers (foci) representing different subject indices and intersect them in an attempt to find material that is relevant to both subjects, as shown in Figure 1-1a, or we could build an ellipse from the two foci, as shown in Figure 1-1b. Indeed, crawls from Yahoo's Computer Science (CS) and Iowa indices meet, appropriately, at Grinnell (Iowa) College's Department of Mathematics and Computer Science (www.math.grin.edu), showing that link geometry can be a useful technique for finding pages on a given set of topics. This technique applies a very simple algorithm to human-created information on the Web to create a new and useful application.

1.1.2 A recommender system

One useful class of information retrieval applications is *recommender systems* [33], where a program recommends new Web pages (or some other resource) judged likely to be of interest to a user, based on the user's initial set of liked pages P. A standard technique for recommender systems is extracting keywords that appear on the initial pages and returning pages that contain these keywords. Note that this technique is based purely on the text of a page, independent of any inter- or intra-document structure.

Another technique for making recommendations is collaborative filtering [40], where pages are recommended that were liked by other people who liked P. This is based on the assumption that items thought valuable/similar by one user are likely to by another user. As collaborative filtering is currently practiced, users *explicitly* rate pages to indicate their recommendations. We can think of the act of creating hyperlinks to a page as being an *implicit* recommendation. In other words, if a person links to pages Q and R, we can guess that people who like Q may like R, especially if the links to Q and R appear near each other on the referencing page (such as within the same list). This makes use of intra-document structural information.

Accordingly, if a user requests a page similar to a set of pages $P_1, \ldots, P_n$, a ParaSite can find (through AltaVista) pages R that point to a maximal subset of these pages and then return to the user what other pages are referenced by R. Note that ParaSite does not have to understand what the pages have in common. It just needs to find a list that includes the pages and can infer that whatever trait they have in common is also exemplified by other pages they point to.

For example, the first page returned from AltaVista that pointed to both Computer Professionals for Social Responsibility ("www.cpsr.org/home.html") and Electronic Privacy Information Center ("www.epic.org") was a list of organizations fighting the Communications Decency Act; links included the Electronic Frontier Foundation ("www.eff.org") and other related organizations. Recommender systems will be discussed in greater detail throughout the thesis, where it will be shown that the ParaSite approach to this problem is very effective.

1.1.3 A home page finder

A new type of application made necessary by the Web is a tool to find users' personal home pages, given their name and perhaps an affiliation. Like many information classification tasks, determining whether a given page is a specific person's home page is an easier problem for a person to solve than for a computer. Consequently, ParaSite's primary strategy is not determining directly if a page "looks like" a home page but finding pages that *human beings* have labeled as being someone's home page. While there is no single stereotypical title for home pages, there is for the anchor text of hyperlinks to them: the author's name. For example, an AltaVista search for pages containing the name "Nicholas Kushmerick" returned 27 links, none of them to his home page. In contrast, a search for hyperlinks with *anchor text* "Nicholas Kushmerick" returned three matches, two of which were links to the correct home page and one to his email address. This is an example of taking advantage of inter-document structure.

Another class of useful structural information is intra-document structure. For example, if the name "Nicholas Kushmerick" appears in the title field of a page, that is more significant than if it appears in the body. The structure of the URL can also be used. The URL of Kushmerick's home page is: "http://www.cs.washington.edu/homes/nick/". This is easily recognized as a likely home page because:

1. The file name is the empty string. (Other stereotypical file names for home pages are "index.html" and "home.html".)
2. The final directory name is the user's email alias.
3. The penultimate directory name is "homes". (Another common penultimate directory for home pages is "people".)

This application will also be discussed at greater length later in this thesis.

1.1.4 Summary

These applications all make use of structural information on the Web. Squeal is the first system to allow their easy implementation. Furthermore, Squeal provides a clean interface that treats the different types of structure in a uniform manner, allowing the user to ignore the details of how the information is represented and to focus on the more meaningful aspects.

source	anchor	destination
www.ai.mit.edu	Lab for Computer Science	www.lcs.mit.edu
www.ai.mit.edu	MIT	web.mit.edu
www.ai.mit.edu	People	www.ai.mit.edu/people
www.lcs.mit.edu	AI Lab	www.ai.mit.edu

Table 1.1: Sample Relation **link** Representing Hyperlinks. Each row or "line" contains a **source** field, an **anchor** field, and a **destination** field. The first line represents the hyperlink labeled "Lab for Computer Science" from the page with URL "www.ai.mit.edu" to the page "www.lcs.mit.edu".

1.2 A Database Interface to the Web

Relational databases and Structured Query Language (SQL) were designed to provide users with a powerful way of accessing structured information. Squeal makes use of these technologies in both its user interface and its implementation. Specifically, it provides the user with the illusion that the Web is stored in a well-defined relational database and can thus be accessed through SQL. The Squeal implementation also stores information, although not the entire Web, in a relational database.

1.2.1 Background

A *relational database* [6] is used to represent sets of data. The objects and constraints of a database are defined by its *schema*. The only complex data structure is the *table*, which consists of zero or more *rows*, each of which is a set of one or more *columns*, each of which can hold data of a specific type, such as an integer or a character array. Table 1.1 shows a table named **link**, each of whose rows consists of a set of three columns, named "source", "anchor", and "destination". Such a database might be used to store information about hyperlinks among web pages, as illustrated with the sample data.

Structured Query Language (SQL) [7] is a declarative language for accessing the schema and data of a relational database. The command for retrieving data is SELECT with a required FROM clause and an optional WHERE clause. The FROM clause indicates what tables should be queried, and the WHERE clause constrains the cases for which data is returned. Some sample SELECT statements and the resulting tables follow:

What are the destinations of hyperlinks originating at "www.ai.mit.edu"?
SELECT destination
FROM link
WHERE source = 'www.ai.mit.edu'

destination
www.lcs.mit.edu
web.mit.edu
www.ai.mit.edu/people
www.ai.mit.edu

Show the values of all fields on lines with anchor text "MIT".
SELECT *
FROM link
WHERE anchor = 'MIT'

source	anchor	destination
www.ai.mit.edu	MIT	web.mit.edu

Figure 1-2: Cache SQL database to the Web

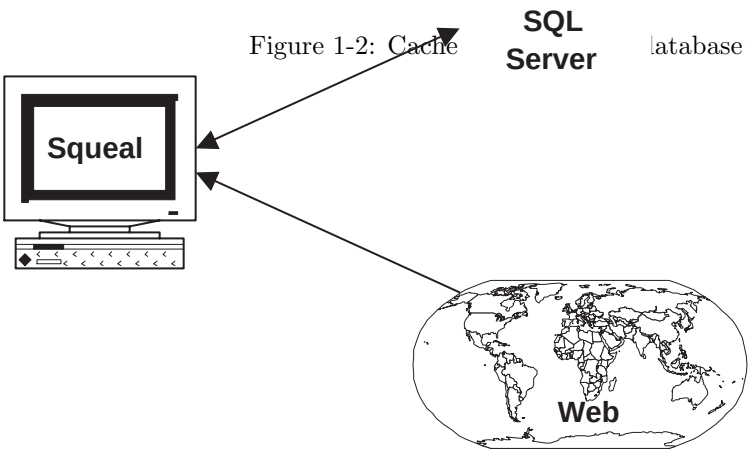


Figure 1-3: Structure of Data Transfer in the ParaSite System

What pages point to each other?
SELECT l1.source, l2.source
FROM link l1, link l2
WHERE l1.source = l2.destination AND
l2.source = l1.destination

l1.source	l2.source
www.ai.mit.edu	www.lcs.mit.edu
www.lcs.mit.edu	www.ai.mit.edu

It would be very useful for tools and users to be able to access the Web in this fashion.

1.2.2 Squeal

Despite the illusion presented by the **link** table, the Web is not really represented as a relational database. Instead, it is a collection of documents stored on many different machines. My system, Squeal, bridges the gap by retrieving and processing information from the Web as needed in order to maintain the illusion that the Web is a relational database. As Figure 1-2 illustrates, this is done by having an actual relational database act as a cache of the Web.

The structure of system data flow is shown in Figure 1-3. The user enters commands in Squeal on a client machine. The Squeal interpreter fetches the appropriate pages from the Web, parses them, places them in an underlying SQL database, and queries the database in SQL to answer user queries. This provides the user with the ability to query the Web as though it were a relational database.

Squeal's core is syntactically a subset of SQL, although interpreted in a different way. Unlike ordinary SQL interpreters, querying the database with the Squeal interpreter can cause the state to change. For example, consider the SQL command:

SELECT * FROM link WHERE source = 'www.ai.mit.edu'

In an ordinary SQL implementation, this would retrieve all lines of the **link** table whose **source** column is equal to "www.ai.mit.edu". The Squeal interpretation of the same statement is:

1. Check whether the system has already computed the links from the page "www.ai.mit.edu". If so, go to step 3.
2. Retrieve the page and parse it, putting the appropriate entries in the **link** table (possibly modifying other tables).
3. Do the ordinary SQL interpretation of the original command.

Except for a difference in response time, the user cannot tell whether the information was already in the database or if it had to be retrieved, processed, and inserted into the database. In other words, the database acts like a cache, as illustrated in Figure 1-2. The result is the illusion that the user is directly querying the Web. Once this abstraction is in place, the discussed ParaSites can be easily implemented.

1.3 Related work

Related work that serves as background is discussed in this section. A comparison of Squeal to other database interfaces to the Web appears in the conclusion, after Squeal has been more fully described.

1.3.1 Semantic networks

The precursors of this work hearken all the way back to the nineteen-sixties. Treating the Web as a knowledge base is reminiscent of semantic networks [31, 26], graphs whose nodes represent concepts or entities and whose arcs represent the relations among them. In the seventies and eighties, Woods [46] and Trigg [43] improved on the model by distinguishing among different types of relations between nodes, also applicable to the Web.

1.3.2 Structure within documents

Numerous hypertext researchers, including Furuta [12] and André et al. [3] have written about structured documents and the benefits they provide for consistency, reusability, and verifiability. Wilkinson and Fuller discuss the use of intra-document structure for both answering queries and displaying results [45]. Before the rise of HTML, the markup language most commonly considered was Standard General Markup Language (SGML), leading to the development of HyQ [18], a Lisp-like query language for the Hypermedia/Time-based Structuring Language (HyTime) extension to SGML.

1.3.3 Structure within and across web pages

Boyan et al. have observed that Web pages differ from general text in that they possess external and internal structure [5]. They use this information in Web tools to propagate rewards from interesting pages to those that point to them (also done by LaMacchia [21]) and to more heavily weight words in titles, headers, etc., when considering document/keyword similarity, a technique used earlier in Lycos by Mauldin and Leavitt [22]. Mauldin has made use of link information by naming a page with the anchor text of hyperlinks to it. Iwazume et al. have preferentially expanded hyperlinks containing keywords relevant to the user's query [16], although O'Leary observes that anchor text information can be unreliable [28]. LaMacchia has implemented or proposed heuristics similar to some mentioned in this paper, such as making use of the information in directory hierarchies [21]. Frei and Stieger have discussed how knowledge of the adjacency of nodes via hyperlinks can be used to help a user navigate or find the answer to a query [11].

1.4 Reader's Guide

Chapter 2 presents an ontology for the Web and Squeal, describing what types of information are made available to the Squeal user, more precisely defining the types of structural information available on the Web. Chapter 2 also provides examples of simple Squeal queries.

The Squeal language is defined in Chapter 3, including the algorithms used to implement the illusion that the Web is a relational database. These algorithms fetch and process information from the Web as implicitly required by user queries and would be useful in any domain where one wanted to provide a similar illusion by putting information in a database in such a lazy fashion.

The low-level details of the Squeal implementation, including how to add new relations, are presented in Chapter 4. While this chapter would be of value for modifying my code or reproducing the work, it contains no vital conceptual information and may be safely skipped by most readers.

Chapter 5 presents the three ParaSites that were implemented: a recommender system, a home page finder, and a moved page finder. For each application, the heuristics, Squeal code, and evaluation are provided. By demonstrating the simplicity and utility of structure-based applications expressed in Squeal, an argument is made for the value of the system.

Chapter 6 contains the conclusions drawn from this work, including a discussion of lessons learned and comparisons to existing work, as well as possible directions for future research.

Chapter 2

Ontology

A fundamental characteristic of the Web is the structural information it encodes, which has been obscured through there being a number of different ways to represent different kinds of structure. A key contribution of my work is providing a *uniform interface* to these different types of structure. This is done through the Squeal ontology, represented by a SQL database schema, which serves as the user's view of the Web, allowing the user to focus on structural information without needing to be aware of the representation.

2.1 Types of Information on the Web

The World-Wide Web consists of *pages* of data, addressed by *uniform resource locators (URLs)*. We focus on pages written in *hypertext markup language (HTML)*. HTML includes *tags* and *attributes* to specify intra-document information, such as a page's logical structure or how it should be visually rendered. HTML also incorporates inter-document information, such as what pages are connected by *hyperlinks*. In this section, I discuss what type of information about Web pages can be accessed through Squeal.

2.1.1 Uniform Resource Locators (URLs)

A URL consists of the following components:

- An access scheme, such as "http", which stands for "hypertext transfer protocol", followed by a colon (:).
- A machine name, preceded by two forward slashes (//).
- Optionally, a machine port number, preceded by a colon (:). The default port number is 80.
- A Unix-style (case-sensitive, slash-delimited) path name, including the file name.
- Optionally, a label preceded by a pound sign (#), specifying a reference within a document.

We concern ourself with the documents accessible via http. Some examples of URLs are:

- http://www.ai.mit.edu:80
- http://www.ai.mit.edu/projects/cs101/#course
- http://www.amazon.com/exec/obidos/ats-query/4304-6019362-659411

```
<H1>Massachusetts Institute of Technology</H1>

<H2>Spotlights</H2>

<UL>
<LI> Felice Frankel: The Power of Images
<LI> An Evening of Conversation with Noam Chomsky and Kathleen Cleaver
</UL>
```

Figure 2-1: HTML Specification of Internal Document Structure. The corresponding display is shown in Figure 2-2

Massachusetts Institute of Technology

Spotlights

- Felice Frankel: The Power of Images
- An Evening of Conversation with Noam Chomsky and Kathleen Cleaver

Figure 2-2: Appearance of HTML Internal Document Structure (Figure 2-1).

2.1.2 Hypertext Markup Language

All documents, except for some leaf nodes, are written in *hypertext markup language* (HTML) [32], which consists of ordinary text augmented with *tags* and *attributes*. Tag names are followed by zero or more attribute assignments, all enclosed in angle brackets. In some cases, subsequent text is affected until the appearance of a concluding tag, prefixed with a forward slash (/). This enclosed text is referred to as *anchor text*. For example, this piece of hypertext:

```
<FONT face="Times" color=red>fire</font>
```

contains the tag "FONT", which indicates that the anchor text ("fire") should appear in the font specified by the attributes, i.e., in the face "Times" and the color red. HTML is used to express information about a document's internal structure, appearance, hyperlinks, and meta-information. We will discuss these in the following separate sections, although a single tag can serve more than one of these purposes.

Internal structure

An HTML document can contain internal structure in the form of *headers* and *lists*. The tag "H1" is used to designate a top-level header, "H2" a second-level header, and so forth through "H6", a sixth-level header. Four tags are used to designate different types of simple lists: "OL" (ordered list), "UL" (unordered list), "MENU", and "DIR" (directory list). The "LI" (list item) tag appears before each item. Figure 2-1 shows some hypertext specifying header and list information. Figure 2-2 how it might be displayed to a user.

Appearance

Tags are used to express the font family (e.g., Times), style (e.g., italics), size, and alignment of text on a page. They are also used to include graphics, white space, and lines. Authors frequently

misuse header tags for their effects on display, even though the commands are meant for structural information.

Hyperlinks

Tags are used to designate hyperlinks from the current page to another location on the Web, typically another page. A link is designated with the "A" ("anchor") tag, and the "HREF" attribute is used to express the destination. When displayed by a browser, clicking on the anchor text causes the destination to be loaded.

Meta-information

The "META" tag is used to express meta-information meant for Web tools rather than for browsers. The "NAME" and "CONTENT" attributes are used to specify the type of information being given and its value, respectively. Common values of the "NAME" attribute are "description" and "keywords", both used by search engines to categorize, rank, and display URLs. While broader standards for metadata, such as the Dublin Metadata Core Element Set [44], have been proposed, none are in wide use.

2.2 Database Relations

Rather than invent a new language to represent structural information, we turn to a representation designed for this purpose: relational databases. This section presents the relational database schema through which the Squeal user views the Web. Once in this representation, Structured Query Language (SQL)can be used to access it.

The Squeal relations are divided into four categories: (1) basic relations, (2) tag and attribute relations, (3) relations built on tags, and (4) relations for second-hand information. Relation and column names appear in **bold face** and function names appear in *slant* font. After a relation is defined, its use is demonstrated through an annotated transcript, in which user commands are shown in **boldface** and annotations in *italics*. To keep the query results readable, I sometimes do not show all of the results returned by the system.

2.2.1 Basic Relations

In order to provide useful information about the Web, it is necessary that strings, URLs, and Web pages can be represented. This is done through the four basic relations:

1. **valstring**, which maps an integer to a string
2. **urls**, which describes the set of URLs representing the same page
3. **parse**, which represents the components of a URL
4. **page**, which represents information corresponding to a single page

valstring

The **valstring** relation, shown in Table 2.1 is used to map an integer **value_id** to a string. The **value_id**s are frequently used in other relations to represent strings. The primary purpose of this indirection it to represent strings as cleanly and efficiently as possible. The **vcvalue** field is used for strings of less than 256 characters, the maximum length of a varchar (variable-length character array) object supported by Microsoft SQL Server. The unlimited **textvalue** field is only used when needed for larger strings, because it requires at least 2048 bytes of storage. For every legal row in the relation, one of **vcvalue** or **textvalue** is null.[1] The SQL definition for this and the other relations is shown in appendix A.

[1]Because of limitations on operations involving TEXT fields, Microsoft SQL Server cannot enforce this constraint.

valstring		
colname	*type*	*notes*
value_id	INT	
vcvalue	VARCHAR(255)	
textvalue	TEXT	

Table 2.1: The **valstring** Relation. Note that this is the SQL data structure representing the table, indicated by the double border.

The *value_id* function returns the **value_id** associated with a string, adding a new row to the relation if necessary. The *value* function converts a **value_id** into a string.

```
CREATE PROCEDURE value @id INT
AS
SELECT COALESCE(CONVERT(TEXT,vcvalue),textvalue)
FROM valstring
WHERE value_id = @id
```

urls

The **urls** relation, shown in Table 2.2 is used to map one or more **value_id**s representing URL strings to a unique **url_id**. If there were a one-to-one correspondence between URL strings and **url_id**s, it would not be needed, but multiple URL strings can reference the same page. For example, both "http://www.ai.mit.edu" and "http://www.ai.mit.edu/index.html" reference the same document, so they would have the same **url_id**, with different **variant** numbers.

The *url_id* function returns the **url_id** associated with a URL string, adding a new row to **urls** and **value_id** if necessary. The *url* function converts a **url_id** into a URL string. This can be defined in SQL:

```
CREATE PROCEDURE url @id INT
AS
SELECT COALESCE(CONVERT(TEXT,v.vcvalue),v.textvalue)
FROM valstring v, urls u
WHERE v.value_id = u.url_id AND u.url_id = @id
```

The **url_id** associated with a string can change because a **url_id** is initially assigned when the URL is encountered, although it cannot be determined whether two URLs correspond to the same page until their contents have been retrieved. It is impractical to retrieve entire pages when a **url_id** is defined, so the only other option is to allow a **url_id** to change. For this reason, the user should call the *url_id* function when a **url_id** is needed, rather than storing the result of calling *url_id*. If the user creates a table with a column of type **url_id**, the value in the table will change if the **url_id** changes.

Strictly speaking, there is no way to tell if two URLs reference the same file or merely two copies of a file. Squeal uses the heuristic that two URLs reference the same page and should have the same **url_id** if they retrieve identical contents and the host component of the URLs are the same strings (ignoring case differences). Corresponding pages at separate mirror sites would not have the same **url_id**.

parse

The **parse** relation, shown in Table 2.3 encodes a parsed version of a URL string. This is useful for determining if two URLs are on the same host machine or if one is above the other in

urls		
colname	*type*	*notes*
url_id	INT	
value_id	INT	unique
variant	INT	default: 0

Table 2.2: The **urls** relation

parse		
colname	*type*	*notes*
url_value_id	INT	
component	CHAR(5)	{"host", "port", "path", "ref"}
value	VARCHAR(255)	
depth	INT	null when component $\neq$ "path"

Table 2.3: The **parse** relation

the directory hierarchy. The **component** field contains "host", "port", "path", or "ref", and the **value** field shows the associated value. For the "path" component, the **depth** within the file hierarchy is also indicated, starting at the file name. Table 2.4 shows the **parse** relation for the URL "http://www.ai.mit.edu/people/index.html". Note that one **url_id** can have multiple parses if multiple URLs correspond to that page.

page

The **page** relation, shown in Table 2.5 caches pages so they are not unnecessarily retrieved from the Web. It contains information from the last time a page was retrieved, including the text of a page, its size, a time stamp and the results of a Message Digest 5 (MD5) hash function [34] on the page's text, providing a quick way to tell if two pages contain the same text. The **valid** field encodes the results of the last attempted retrieval of the page: "y" if it was successfully retrieved, "n" if it could not be retrieved, or "b" if the page was over the maximum page size limit (to be discussed in the next chapter).

Figure 2-3 includes a transcript demonstrating these relations.

2.2.2 Tag and attribute relations

Because they encode the structure of pages and hyperlinks, tags and attributes are fundamental to Squeal, allowing users to access this information.

url_value_id	component	value	depth
950	host	www.ai.mit.edu	0
950	port	80	0
950	path	index.html	1
950	path	people	2
950	ref	[null]	0

Table 2.4: Parse relation for **url_value_id** = *value_id*('http://www.ai.mit.edu/people/index.html')

What **value_id** *corresponds to the string "http://www.ai.mit.edu/index.html"?*
SELECT value_id
FROM valstring
WHERE vcvalue='http://www.ai.mit.edu/index.html';

value_id
2606

What **url_id** *corresponds to the url represented by "http://www.ai.mit.edu/index.html"?*
SELECT url_id
FROM urls
WHERE value_id = 2606;

url_id
817

The url_id *operator is a shortcut for the above sequence:*
PRINT *url_id*(**'http://www.ai.mit.edu/index.html'**);
817

Show all the known URLs corresponding to this **url_id**.
SELECT *
FROM urls
WHERE url_id = 817;

url_id	value_id	variant
817	2606	1
817	11128	2

Show all the known URLs for this page.
SELECT v.vcvalue
FROM valstring v, urls u
WHERE u.url_id = 817 AND v.value_id = u.value_id;

vcvalue
http://www.ai.mit.edu
http://www.ai.mit.edu/index.html

How many bytes long is this page?
SELECT bytes FROM page WHERE url_id = 817;

bytes
5618

What is the host component of the URL?
SELECT value
FROM parse
WHERE component='host' AND url_value_id = 817;

value
www.ai.mit.edu

Figure 2-3: Transcript Illustrating the Basic Relations

page		
colname	*type*	*notes*
url_id	INT	
contents	TEXT	
bytes	INT	
md5	CHAR(32)	
when	DATETIME	

Table 2.5: The **page** relation

tag		
colname	*type*	*notes*
url_id	INT	
tag_id	INT	unique
name	CHAR(15)	
startOffset	INT	
endOffset	INT	

Table 2.6: The **tag** relation

tag

The **tag** relation, shown in Table 2.6, contains one line for each tag or set of paired tags encountered. (Single tags include "<hr>" (horizontal rule); paired tags include "<title>" ... "</title>".) Information stored includes the **url_id**, tag name, start and end character offsets within the document, and a unique **tag_id**, which is used to reference attributes.

att

The **att** relation, shown in Table 2.7, contains one line for each attribute name and value in a document. The name is a short string and the value an index into **valstring**. I treat the anchor text enclosed by a pair of tags to be the value of the attribute "anchor".

Figure 2-4 has a transcript demonstrating the use of the **tag** and **attribute** relations.

2.2.3 Relations built on tags

Information about the structure of headers and lists on a page and hyperlinks across pages can be inferred from the information in the **tag** and **att** tables, but we provide the user with specialized tables **header**, **list**, and **link** for greater convenience.

att		
colname	*type*	*notes*
tag_id	INT	
name	CHAR(15)	
value_id	INT	

Table 2.7: The **att** relation

How many times does each type of tag appear on "http://www.mit.edu"?

```
SELECT name, COUNT(*)
FROM tag
WHERE url_id = url_id('www.mit.edu')
GROUP BY name;
```

name	
!DOCTYPE	1
A	42
BODY	1
FNORD	15
HEAD	1
HTML	1
IMG	10
P	16
TITLE	1

What tags appear within the first hundred characters?

```
SELECT name, tag_id
FROM tag
WHERE url_id = url_id('www.mit.edu') AND startOffset < 100;
```

name	tag_id
!DOCTYPE	1087
HTML	1088
HEAD	1089
TITLE	1090

Show me the attributes of the "title" tag.

```
SELECT *
FROM att
WHERE tag_id = 1090;
```

tag_id	name	value_id
1090	anchor	1105

What is the text associated with the "title" tag?

```
SELECT vcvalue
FROM valstring
WHERE value_id = 1105;
```

vcvalue
SIPB WWW Server Home Page

What are the attributes associated with "img" (image) tags in the first 2000 characters?

```
SELECT a.name, v.vcvalue, t.tag_id
FROM valstring v, tag t, att a
WHERE t.url_id = url_id('www.mit.edu')
    AND t.startOffset < 2000 AND t.name='img'
    AND a.tag_id = t.tag_id AND v.value_id = a.value_id
```

name	vcvalue	tag_id
alt	MIT SIPB WWW Server	1098
src	http://www.mit.edu/gif/MITSIPB_cropped.GIF	1098
align	bottom	1126
src	http://www.mit.edu/gif/Icon_NEW.GIF	1126

Figure 2-4: Transcript Demonstrating the **tag** and **att** Relations

header		
colname	*type*	*notes*
url_id	INT	
struct	BINARY(6)	
ord	INT	
offset	INT	

Table 2.8: The **header** relation

list		
colname	*type*	*notes*
url_id	INT	
struct	BINARY(6)	
ord	INT	
offset	INT	

Table 2.9: The **list** relation

header

As discussed above, HTML supports six levels of headings (H1–H6). Through the **header** table, shown in Table 2.8, Squeal keeps track of the levels of headings at each point of change. The **struct** column consists of an array of six bytes, each of which corresponds to one level of heading. At the beginning of a page, all six bytes are initialized to zero. Whenever a <Hn> **tag** is encountered, byte $(n-1)$ is incremented, and all bytes with higher offsets are zeroed. This can be seen in the above transcript. The **ord** element is an ordinal number incremented with every new database line. The following transcript provides an example:

Show the header structure of "www.ai.mit.edu/people/ellens/cv.html"
SELECT h.struct, t.name, h.offset
FROM header h, tag t
WHERE h.url_id=437 AND t.url_id = 437
AND h.offset = t.startOffset;

struct	name	offset
1 0 0 0 0 0	H1	129
1 1 0 0 0 0	H2	586
1 2 0 0 0 0	H2	1820
1 3 0 0 0 0	H2	3246
1 4 0 0 0 0	H2	4691
1 5 0 0 0 0	H2	5591
1 6 0 0 0 0	H2	6518
1 6 1 0 0 0	H3	6680
1 6 2 0 0 0	H3	10618
1 7 0 0 0 0	H2	13645
1 8 0 0 0 0	H2	16948

list

As discussed above, There are four types of lists supported by HTML: ordered lists, unordered lists, menus, and directories. In the **list** relation, they are not differentiated among. Lists can be arbitrarily nested in HTML; Squeal keeps track of up to six levels of nesting of lists, via the **list**

- Initialization
  1. Set each of the 6 bytes of **struct** to 0.
  2. Set each of the 6 bytes of internal structure **lmax** to 0.
- When a new list begins
  1. If all elements of **struct** are 0, set i to 0. Otherwise, set i to the highest index such that $\mathbf{struct}[i] \neq 0$.
  2. $\mathbf{lmax}[i] \leftarrow \mathbf{lmax}[i] + 1$
  3. $\mathbf{struct}[i] \leftarrow \mathbf{lmax}[i]$
  4. $\forall j \mid i < j < 6, \mathbf{lmax}[j] \leftarrow 0$
- When a list ends
  1. Set i to the highest index such that $\mathbf{struct}[i] \neq 0$.
  2. $\mathbf{struct}[i] \leftarrow 0$

Figure 2-5: Rules for Managing List Structure

link		
colname	*type*	*notes*
source_url_id	INT	
anchor_value_id	INT	
dest_url_id	INT	
hstruct	BINARY(6)	
lstruct	BINARY(6)	

Table 2.10: The **link** relation

relation, shown in Table 2.9. The **struct** column contains an array of six bytes. The rules for managing the bytes are shown in Figure 2-5, and an example is shown in Figure 2-6.

link

The **link** relation, shown in Table 2.10, contains one line for each hyperlink encountered, storing the source **url_id**, anchor text **value_id**, destination **url_id**, and the **list** structure and **header** structure at which the hyperlink occurs. Because all of the information in the **link** table exists in other tables, it can be defined as a SQL view (virtual table), as shown in Figure 2-7 (although for practical reasons, it is implemented as an ordinary table).
Figure 2-8 contains a transcript demonstrating the **link** relation.

2.2.4 Relations for second-hand information

In addition to direct information about the text and hyperlinks of Web pages, indirect information can be obtained from Web tools. It is important however to distinguish between verified and second-hand information, hence the **rcontains** table, for *reported contents* of pages, and the **rlink** table, for *reported links*.

0	1	2	3	4	5	
-	-	-	-	-	-	
0	0	0	0	0	0	[start of document]
1	0	0	0	0	0	List1 begins
1	1	0	0	0	0	Sublist 1a begins
1	0	0	0	0	0	Sublist 1a ends
0	0	0	0	0	0	List 1 ends
2	0	0	0	0	0	List 2 begins
2	1	0	0	0	0	Sublist 2a begins
2	0	0	0	0	0	Sublist 2a ends
2	2	0	0	0	0	Sublist 2b begins
2	0	0	0	0	0	Sublist 2b ends
0	0	0	0	0	0	List 2 ends

Figure 2-6: Example of **struct** values of **list** at list tags

```
CREATE VIEW link
AS
SELECT DISTINCT t.url_id AS source_url_id,
    aAnchor.value_id AS anchor_value_id,
    u.url_id AS dest_url_id,
    h1.struct AS hstruct,
    l1.struct AS lstruct
FROM tag t, att aAnchor, att aHref, urls u,
    header h1, header h2, list l1, list l2
WHERE t.name = 'a' AND t.tag_id = aAnchor.tag_id AND
    aAnchor.name = 'anchor' AND t.tag_id = aHref.tag_id AND
    aHref.name = 'href' AND u.value_id = aHref.value_id AND
    h1.url_id = t.url_id AND h2.url_id = t.url_id AND
    h1.offset < t.startOffset AND h2.offset > t.startOffset AND
    h1.ord+1 = h2.ord AND l1.url_id = t.url_id AND
    l2.url_id = t.url_id AND l1.offset < t.startOffset AND
    l2.offset > t.startOffset AND l1.ord+1 = l2.ord
```

Figure 2-7: A SQL definition of **link** in terms of other relations

Show the value_id and anchor text of all links from "www.ai.mit.edu/projects/haystack/"

```
SELECT l.anchor_value_id, v.vcvalue
FROM link l, valstring v
WHERE l.source_url_id = 503 AND l.anchor_value_id = v.value_id;
```

anchor_value_id	vcvalue
115	Publications
120	People
1686	Introduction
1687	Download
1688	Installation
1689	Online Manual

Give me the url_id and URL of some pages that point to "www.ai.mit.edu"

```
SELECT l.dest_url_id AS url_id, v.vcvalue AS value
FROM link l, valstring v, urls u
WHERE l.dest_url_id = 1 AND u.url_id = l.source_url_id
AND v.value_id = u.value_id;
```

url_id	value
3	http://www.tns.lcs.mit.edu/
4	http://www-mtl.mit.edu/
5	http://www.csg.lcs.mit.edu:8001/
6	http://farnsworth.mit.edu/other_servers.html
7	http://www.frob.com/
9	http://cag-www.lcs.mit.edu/

See what other links from "www.tns.lcs.mit.edu" are in the same list as the link to "www.ai.mit.edu"

```
SELECT DISTINCT l.dest_url_id, v.vcvalue
FROM link l, link lAI, valstring v, urls u
WHERE l.source_url_id = 3 AND lAI.source_url_id = 3
AND lAI.dest_url_id = 1 AND l.lstruct = lAI.lstruct
AND l.dest_url_id = u.url_id AND v.value_id = u.value_id;
```

dest_url_id	vcvalue
1	http://www.ai.mit.edu/
2	http://www.lcs.mit.edu/
34	http://web.mit.edu/
78	http://farnsworth.mit.edu/
79	http://www.mit.edu/

Figure 2-8: Transcript Demonstrating the **link** Relation

rcontains		
colname	*type*	*notes*
url_id	INT	
value	VARCHAR(255)	
helper	CHAR(16)	

Table 2.11: The **rcontains** relation

rlink		
colname	*type*	*notes*
source_url_id	INT	
anchor	VARCHAR(255)	
dest_url_id	INT	
helper	CHAR(16)	

Table 2.12: The **rlink** relation

rcontains

The **rcontains** relation, shown in Table 2.11, indicates the claim by the search engine specified in the **helper** column that a certain page contains a certain piece of text. In the current implementation, legal values for the **helper** column are "altavista" and "lycos". (Details on how many such pages the **rcontains** table holds will be discussed in Section 3.6.) The SQL definition of **rcontains** is:

rlink

The **rlink** relation, shown in Table 2.12, contains search engine claims about hyperlinks among pages. Because of how the underlying search engine (AltaVista) works, the **source_url_id** field will always have a non-null value, as will the **helper** field, but exactly one of **anchor** and **dest_url_id** will be null. In other words, a line can indicate the reported source and destination of a hyperlink, or it can indicate a page and anchor text that reportedly appears in a hyperlink. The missing information, assuming the reported link actually exists, can be extracted from the **link** relation. The transcript in Figure 2-9 demonstrates the use of the **rlink** and **rcontains** relations.

2.3 Summary

We have shown the Squeal ontology, i.e., what information on the Web is available to the Squeal user and the abstraction through which it is accessed. Specifically, the user has access to the raw and parsed version of each URL (**urls**, **parse**), the raw text, size, and hash value of each page (**page**), including all of the **tag** and attribute (**att**) values. This information can be accessed directly or through the **header**, **list**, and **link** relations. The user can also ask what pages reportedly contain a specified piece of text (**rcontains**) or a described hyperlink (**rlink**). In the next chapter, we provide more detail on the syntax of user queries and how Squeal provides the information.

Find pages that reportedly contain "MIT AI Lab".

```
SELECT r.url_id, v.vcvalue
FROM rcontains r, valstring v, urls u
WHERE r.value LIKE '%MIT AI Lab%'
    AND u.url_id = r.url_id AND v.value_id = u.value_id;
```

url_id	**vcvalue**
829	http://www.ai.mit.edu/weather/weather.html
830	http://www.mic.atr.co.jp/ hajiri/html/AI.html
828	http://www.neurop2.ruhr-uni-bochum.de/MIT_lit.html
837	http://ymiseja.eenet.ee/IT/maillist/msg9.html
838	http://www.pmms.cam.ac.uk/ gjm11/jargon/jargappA.html
839	http://www.csa.runnet.ru/AI/faqs/lisp_6.htm

Find pages that reportedly point to "www.ai.mit.edu/people/ellens/gender.html".

```
SELECT r.source_url_id, v.vcvalue
FROM rlink r, valstring v, urls u
WHERE r.dest_url_id = 591 AND u.url_id = r.source_url_id
AND v.value_id = u.value_id;
```

source_url_id	**vcvalue**
542	http://www.cs.vassar.edu/
846	http://www.mcs.salford.ac.uk/TOUR/022.htm
847	http://www.cmu.edu/adm/uri/oncampus/opps.html
851	http://yaron.clever.net/sheizaf3.shtml

Find pages that reportedly contain "computer science" as anchor text.

```
SELECT r.source_url_id, v.vcvalue
FROM rlink r, valstring v, urls u
WHERE r.anchor = 'computer science'
AND u.url_id = r.source_url_id and v.value_id = u.value_id;
```

source_url_id	**vcvalue**
868	http://longwood.cs.ucf.edu/
869	http://emmy.smith.edu/
870	http://www.icsi.berkeley.edu/
871	http://www.engr.uvic.ca/

Figure 2-9: Transcript Demonstrating the **rlink** and **rcontains** Relations

Chapter 3

Squeal

The previous chapter described a relational database model of the Web. This abstraction is implemented by the Squeal interpreter, which determines what information from the Web needs to be brought into the database and how to do so. This chapter describes the Squeal language and the algorithms that support its interpretation. The next chapter goes into lower-level detail about the Java implementation of the Squeal interpreter.

The Squeal core is syntactically a subset of SQL, with a different semantics. For example, consider the possible interpretations of the following statement:

```
SELECT dest_url_id FROM link WHERE source_url_id = 7
```

The semantics of the statement in SQL are to query the **link** table and return to the user all values of **dest_url_id** whose associated **source_url_id** is seven. No changes are made to the database by the statement's execution. In contrast, the semantics in Squeal are to return the **url_id**s of all links from the Web page represented by **source_url_id**. As a side effect, the **link** table in the database representation may be modified, but users need not be aware of this. The purpose of Squeal is to give users the illusion that they are directly querying the Web.

When a statement is entered, the Squeal interpreter parses it, rejecting it with an error message if it is invalid and generating a parse tree otherwise. If the statement doesn't reference any of the special tables described in Chapter 2, it is passed through to the SQL server and the reply is passed back to the user. If special tables are referenced, the Squeal interpreter determines what information from the Web is needed in order to answer the query. If the information is not yet in the database, it fetches and parses Web pages and puts the needed information into the database. The rest of this chapter gives the syntax and semantics of Squeal and the algorithms for its interpretation. In addition to the core, which only includes legal SQL statements, Squeal contains some simple mechanisms for naming and abstraction, also described in this chapter.

3.1 Lexical Tokens

Figure 3-1 shows the lexical tokens used in the Squeal grammar. Strings may be delimited with either single or double quotation marks. Identifiers may start with optional "#" characters (for SQL temporary tables), then must have a letter, and then may be followed by any combination of letters, digits, and underscores. The language is case-insensitive. Statements are separated by semicolons, not shown in the below grammars. Comments are begun with a double slash ("//") and extend to the end of a line.

3.2 Expressions

The grammar for expressions is shown in Figure 3-2. The semantics of expression evaluation depend on whether an object is a first-class or passed-through data type. The complete Squeal grammar

```
<STRING>:     ("'" (~["'"])* "'") | ("\"" (~["\""])* "\"")

<LETTER>:     ["A"-"Z", "a"-"z"]

<DIGIT>:      ["0"-"9"]

<IDENTIFIER>: ("#")* (<LETTER>)+ (<LETTER>|<DIGIT>|"_")*

<NUMBER>:     (<DIGIT>)+
```

Figure 3-1: Lexical tokens

appears in appendix B. In this chapter, we simplify it slightly for expository purposes.

3.2.1 First-class data types

Integers and strings are first-class objects. The binary operations "+", "-", "*", and "/" are defined for integers, as is unary minus. Strings may be bounded with single or double quotation marks.

3.2.2 Passed-through data types

Squeal also recognizes table types, aliases, column names, relational operators, and logical expressions but does not evaluate them, instead passing them to the SQL server for interpretation, typically as part of the WHERE clause of a SELECT statement.

3.2.3 Special data types

While the representation of the **url_id** and **value_id** types are integers, they are treated as special for two reasons:

1. It is convenient for the interpreter to know they are not ordinary integers, so the corresponding string can be displayed to the user.
2. The **url_id** associated with a specific URL (specifically, associated with a specific **value_id**) can change, as discussed in Section 2.2.1.

If the user defines a column to be of type **url_id**, the interpreter knows to change the values if the **url_id** changes.

3.3 Simple Statements

Figure 3-3 shows the grammar for SQL statements. In this section, we will discuss the statements that do not access the Web or the database.

3.3.1 Basic statements

The grammar for the **print** and **let** statements is shown in Figure 3-4. The **print** statement, which can be abbreviated "?", prints an expression. For example, "print 2*3" prints 6. The **let** statement sets a variable to an expression value. If the variable has not been defined, it is created. Any variable can hold either a string or an integer. The expression in the optional "ELSE" clause is evaluated if the first expression is null. This is similar to the "COALESCE" statement in SQL.

```
expression:                 disjunctionExpression

disjunctionExpression:      conjunctionExpression ("OR" disjunctionExpression)?

conjunctionExpression:      negationExpression ("AND" conjunctionExpression)?

negationExpression:         ("NOT")? relExpression

relExpression:              sumExpression (rel_opsumExpression)?

rel_op:                     "<" | "=" | ">" | "<>" | ">=" | "<="
                            | "NOT IN" | "LIKE" | "NOT LIKE" | "IN"

sumExpression:              productExpression (("+")|("-") sumExpression)?

productExpression:          unaryExpression (("*")|("/") productExpression)?

unaryExpression:            ("-")? parenthesizedExpression

parenthesizedExpression:    "(" selectStatement ")"
                            | "(" expression ")"
                            | funcall
                            | cell
                            | <NUMBER> | <IDENTIFIER> | <STRING>
                            | aggregateExpression
                            | "*"

cell:                       <IDENTIFIER> "." (<IDENTIFIER> | "*")

aggregateExpression:        aggregate_op "(" agg_restrict? expression ")"

aggregate_op:               "AVG" | "MAX" | "MIN" | "SUM" | "COUNT"

agg_restrict:               "ALL" | "DISTINCT" | "UNIQUE"
```

Figure 3-2: Grammar for Expressions. . The symbol "|" represents disjunction. When an item is followed by "?", it is optional; when followed by "*", it may appear any number of times, including zero. The nonterminal `funcall` is defined below in Figure 3-5.

```
statement: printStatement
         | letStatement
         | callStatement
         | helpStatement
         | deffuncStatement
         | defprocStatement
         | inputStatement
         | outputStatement
         | quitStatement
         | fetchStatement
         | selectStatement
         | createStatement
         | dropStatement
         | deleteStatement
         | updateStatement
         | insertStatement
         | describeStatement
```

Figure 3-3: Grammar for **statement**

`letStatement` = "LET" <IDENTIFIER> "=" `expression` ("ELSE" `expression`)

Figure 3-4: Grammar for LET Statements. The variable is defined to be the value of the first expression, unless it is null and an ELSE clause is present, in which case the second expression is used.

```
callStatement:    funcall

funcall:          <IDENTIFIER> argList

argList:          "(" (expression ("," expression)*)? ")"

deffuncStatement: "DEFFUNC" <IDENTIFIER> symbolList expression

defprocStatement: "DEFPROC" <IDENTIFIER> symbolList (statement)+ "ENDPROC"

symbolList:       "(" (<IDENTIFIER> ("," <IDENTIFIER>)*)? ")"

helpStatement:    "HELP" "(" <IDENTIFIER> ")"
                  | "HELP" <IDENTIFIER>
```

Figure 3-5: Grammar for DEFFUNC, DEFPROC, CALL, and HELP

function	description
strcat	Concatenate any number of strings
directory	Return a URL with the file name removed
dirparent	Return the parent of the given directory
value_id	Return the **value_id** associated with a string. If no string is provided, return a number one higher than the highest **value_id**
value	Return the string associated with a **value_id**
url_id	Return the **url_id** associated with a string or **value_id**
url	Return the url associated with a **url_id**

Table 3.1: User-Callable Functions Defined at Squeal Start-Up

3.3.2 Function/procedure statements

Squeal supports built-in and user-defined *functions* and *procedures*. The body of a function is an expression, and the body of a procedure is one or more statements. The **deffunc** and **defproc** statements are used to declare functions and procedures, respectively, and they are called with the **call** statement. The **help** statement prints information about a built-in function. The grammar for these statements appear in Figure 3-5. Figure 3-6 shows a transcript demonstrating their use. Table 3.1 shows a list of functions defined when the system begins.

3.3.3 Control statements

The grammar for the control statements is shown in Figure 3-7. The INPUT statement specifies from where the Squeal interpreter should receive its input. If the expression is a file name, commands will be read from that file. If the expression is a number, commands will be read from the associated port. The OUTPUT statement indicates that ordinary output should be written to the associated file.

The QUIT statement closes the current input stream to the interpreter, ending the effect of the last INPUT statement. If QUIT is entered at the top input level, the interpreter is exited.

help strcat;
Concatenate any number of strings

? strcat("foo", "bar");
foobar
[printed]

deffunc double(x) strcat(x,x);
[defined function 'double']

? double('hello');
hellohello
[printed]

```
defproc sumdiff(a, b)
    ? a + b;
    ? a - b;
endproc;
```
[defined procedure 'sumdiff']

sumdiff(10,2);
12
8
[printed]

Figure 3-6: Transcript Demonstrating Squeal Functions and Procedures. Commands are in bold face. All other text, including that in brackets, is generated by the Squeal interpreter.

```
inputStatement:  "INPUT" expression

outputStatement: "OUTPUT" stringLiteral

quitStatement:   "QUIT" | "EXIT"
```

Figure 3-7: Grammar for INPUT, OUTPUT, and QUIT statements

table	defining columns	optional columns
link	**source_url_id, anchor_value, dest_url_id**	
page	**url_id**	
parse	**url_value_id**	
rcontains	**value**	**helper, num**
rlink	**anchor_value, dest_url_id**	**helper, num**
tag	**url_id**	

Table 3.2: Defining Columns for Tables

User query:

SELECT * FROM **link** WHERE source_url_id = 1

Normalized form:

SELECT $\mathcal{L}$.* FROM **link** $\mathcal{L}$ WHERE $\mathcal{L}$.source_url_id = 1

Squeal interpretation:

FETCH **link**(source_url_id = 1)

MSELECT * FROM **link** WHERE source_url_id = 1

Figure 3-8: Transformation of Squeal User Query into Internal Statements

3.4 Internal Statements: FETCH and MSELECT

In addition to the Squeal user commands, there are two internal commands that are useful to the Squeal interpreter: FETCH and MSELECT. FETCH does whatever is necessary to fill in lines of a table, given a *defining column* and a value, which can be used to define entire lines of the relation. For example, **source_url_id** is a defining column of **link** because, given a **source_url_id**, all of the values of associated lines can be computed. Table 3.2 shows the defining columns of each of the applicable tables, as well as columns that may optionally be defined in the case of the **rcontains** and **rlink** relations. The **helper** field specifies which search engine should be consulted, and **num** indicates the number of matching pages that should be retrieved. By default, this value is 10. MSELECT, standing for "manual select", simply passes a SELECT statement to the underlying SQL server. As the example in Figure 3-8 shows, SELECT statements are normalized and then broken down into a combination of FETCH and MSELECT statements. This process is discussed in Section 3.5.1.

The FETCH statement

While the Squeal interpreter can be configured to allow the user to enter FETCH and MSELECT statements, they are meant for internal Squeal use. The grammar of the FETCH statement is shown in Figure 3-9. We will consider three separate classes of FETCH statements. During this discussion, it is important to distinguish between *table types*, such as **link**, and *table aliases*, such as $\mathcal{L}$, as illustrated in Figure 3-8. If a query involves only one table alias, the user may omit it, in which case an alias is inserted during Squeal's query normalization process.

Basic FETCH statement The simplest type of FETCH statement does not include a "FROM" clause. How it is interpreted depends on what relation and columns are specified. Figure 3-10 shows how they are interpreted. If multiple columns are supplied, only the earliest one in the relation definition is used. For example,

FETCH **link**(**source_url_id**=1, **dest_url**=2)

is treated as:

```
fetchStatement  = "FETCH" <IDENTIFIER> "(" fetchList ")"
                  ("FROM" tableList
                  ("WHERE" logicExpression)?
                  ("GROUP BY" columnList)?
                  ("HAVING" logicExpression?)?

fetchList       = namedArgument ("," namedArgument)*

namedArgument   = <IDENTIFIER> "=" fetchExpression

fetchExpression = expression "|"expression
                | expression "&" expression
                | expression
```

Figure 3-9: Grammar for FETCH Statement. Nonterminals `columnList` and `orderList` are defined in Figure 3-11. Nonterminals `logicExpression` and `expression` are defined in Appendix B.

table	column
link	**anchor_value**
link	**dest_url_id**
rcontains	**value**
rlink	**anchor_value**
rlink	**dest_url_id**

Table 3.3: Relations Allowing FETCH Conjunctions or Disjunctions

FETCH **link**(**source_url_id**=1)

FETCH **statement with Conjunction or Disjunction** In addition to providing a single expression for a column, it is possible to provide a conjunction or disjunction. For example, consider the following statement:

FETCH **rcontains**(value='Golden'&'Spertus')

This asks Alta Vista for all pages that contain *both* of the words "Golden" and "Spertus" (of which there are 20). Similarly, consider the following statement:

FETCH **rcontains**(value='Golden'|'Spertus')

This finds pages that Alta Vista claims contain *either* "Golden" or "Spertus" (of which there are about 3000). Table 3.3 shows the columns that can be set to conjunctions or disjunctions.

FETCH **statement with FROM clause** To use the contents of a table column as part of the expression, one can supply a FROM clause to a FETCH statement. For example, consider the following FETCH statement, which references user-defined table "utable":

FETCH **link**(**source_url_id** = **url_id**) FROM utable WHERE **url_id** > 10

This is interpreted as:

For all values u of **url_id** in table **utable** such that $u > 10$,
FETCH **link**(source_url_id = u).

table

- **source_url_id**: fetch the page and parse links
- **anchor_value**: perform FETCH rlink(anchor_value=<anchor_value>)), then verify links
- **dest_url_id**: perform FETCH rlink(dest_url_id=<dest_url_id>), then verify links

page

- **url_id**: FETCH page from the Web

parse

- **url_value_id**: parse associated string

rcontains

- **value**: ask search engine for pages containing <value>

rlink

- **anchor_value**: ask search engine for pages with the associated string as anchor text
- **dest_url_id**: ask search engine for pages pointing to <dest_url_id>

tag

- **url_id**: FETCH page from Web or **page** table and parse

Figure 3-10: Interpretation of Simple FETCH Statements. When a search engine is used, it is by default Alta Vista with 10 responses retrieved, but these parameters can be changed via the **helper** and **num** columns, respectively.

nonterminal	statement	description
createStatement	CREATE	create a table
dropStatement	DROP	drop (delete) a table
describeStatement	DESCRIBE	describe an existing table
insertStatement	INSERT	insert rows into a table
updateStatement	UPDATE	change values in a table
deleteStatement	DELETE	delete rows from a table
selectStatement	SELECT	select rows from a table

Table 3.4: SQL Commands Supported by Squeal

```
selectStatement = ("SELECT"|"MSELECT") ("ALL" | "DISTINCT") ? selectList
                  "FROM" tableList
                  ("WHERE" logicExpression)?
                  ("GROUP BY" columnList)?
                  ("HAVING" logicExpression)?
                  ("ORDER BY" orderList)?

selectList      = selectItem ("," selectItem)*
selectItem      = expression ("AS" <IDENTIFIER>)?

tableList       = tableName ("," tableName)*
tableName       = <IDENTIFIER> (<IDENTIFIER>)?

orderList       = orderItem ("," orderItem)*
orderItem       = expression ("ASC" | "DESC")
columnList      = column ("," column)*
column          = <IDENTIFIER> ("." <IDENTIFIER>)
```

Figure 3-11: Grammar for Squeal Queries. The definitions of nonterminals `logicExpression` and `expression` appear in appendix B.

3.5 Squeal's SQL core

Table 3.4 lists the SQL commands supported by Squeal. None of them except for SELECT can be applied to the system tables described in chapter 2, only to user-defined tables. By default, the CREATE statement does not cause an error if the table being defined already exists; instead, it drops the old table and then creates a new one. SQL-compliant CREATE behavior can be achieved with the "-c" command-line option 3.6. Figure 3-11 shows the simplified grammar for SELECT statements.

As discussed above, not all syntactically-valid statements are acceptable in Squeal. Tables are divided into three categories, *user-readable*, *automatic*, and *derived*, as shown in Table 3.5. Different rules apply for queries depending on their category.

3.5.1 User-Readable Tables

User-readable tables include **urls** and **valstring**, as well as any tables created by the user. Squeal passes queries on these tables directly to the SQL client, without performing and checks or side effects. Changes are effected to **urls** through the **url_id** function (section 2.2.1), to **valstring** through the **value_id** function (section 2.2.1), and to user-defined tables through INSERT statements.

table	category
urls	user-readable
valstring	user-readable
link	automatic
page	automatic
parse	automatic
rcontains	automatic
rlink	automatic
tag	automatic
att	derived
header	derived
list	derived

Table 3.5: Categories of Tables. Each table is either *user-readable*, *automatic*, or *derived*. User-defined tables are user-readable.

3.5.2 Automatic Tables

Queries on automatic tables, such as **link** require analysis by the Squeal interpreter, which will turn them into FETCH and MSELECT statements. They are called "automatic" because the entries are automatically filled in response to user queries. Each automatic table has one or more *defining columns*, as discussed in section 3.4.

The most basic query involving an automatic table is of the form:

```
SELECT * FROM <table type> <table alias> WHERE (<col name> = <expression>)
```

If <col name> is a defining column for <table type>, this is transformed into:

```
FETCH <table type>( <col name> = <expression> )
MSELECT * FROM <table type> WHERE (<col name> = <expression>)
```

If <col name> is not a defining column, the interpreter rejects the statement.

The four key procedures involved in generating FETCH statements from SELECT statements are:

1. **transform**: top-level routine, which calls the below procedures and outputs required FETCH statements (Figure 3-12).
   (a) **merge**: recognize a conjunction idiom, possibly outputting a FETCH statement (Figure 3-14).
   (b) **findBound**: try to find a bound on a defining column of an automatic table appearing in the SELECT statement; may call itself recursively (Figure 3-13).
   (c) **refDependences**: help construct FETCH statements, adding references to tables upon which the current table depends; may call itself recursively (Figure 3-15).

3.5.3 Derived Tables

Derived tables are not computed directly. Instead, they are computed when their *parent table* is computed. For example, **header** for a **source_url_id** is computed as a side effect of computing **tag** for the same **source_url_id**. Table 3.6 shows the parent tables of each of the derived tables and the column they have in common with their parent.

Derived tables are dealt with similarly, but not identically, to automatic tables. All changes are to procedure **findBound**. Figure 3-16 shows the changes required to **findBound**.

Procedure **transform**(selectStatement)

1. Initialization
   (a) Let unboundedTables be the set of table aliases in selectStatement.
   (b) Let whereDef be the WHERE clause in selectStatement.
   (c) Let selectionTable by an empty hash table indexed on table aliases.
   (d) Let orderedKeys be an empty vector.
2. Call merge(selectStatement, unboundedTables)
3. Let remainingTables be |unboundedTables|.
4. For each table alias currentTableAlias in unboundedTables,
   call findBound(currentTable, unboundedTables, fetchClauses, whereDef)
5. Branch point on the value of |unboundedTables|:
   (a) remainingTables, fail.
   (b) > 0, go to step 3.
   (c) else, execute completion code:
      i. For each alias in orderedKeys, if the table is not a UserReadableTable
         A. Let fetchString = "FETCH ".
         B. Let tablesString = "".
         C. Let clausesString = " WHERE (1=1) "
         D. Let dependences be the empty set
         E. For each tuple [currentTableAlias, LHSstring, op, RHSstring, tablesReferenced] associated with alias in selectionTable where LHSstring $\neq$ the empty string
            1. Append to fetchString: LHSstring + op + RHSstring
            2. Add the elements of tablesReferenced to dependences.
            3. Let processed be the set containing alias
            4. call refDependences(alias, selectionTable, dependences, tablesString, clausesString, processed)
            5. Output clause: fetchString + ") FROM " + tablesString + clausesString
      ii. return success

Figure 3-12: Pseudocode for **transform**. Given a selectStatement, **transform** returns a set of clauses or fails. Code to handle simple syntactic items, such as commas, has been omitted.

derived table	parent table	shared column
att	tag	tag_id
header	tag	url_id
list	tag	url_id

Table 3.6: Relations Between Derived Tables and Their Parents

Procedure **findBound**(currentTableAlias, unboundedTables, fetchClauses, whereDef)
Branch point on the type of whereDef

1. disjunction (clause1 OR clause2)
   (a) Let returnValue1 be the result of findBound(currentTableAlias, unboundedTables, fetchClauses, clause1)
   (b) If returnValue = false, return false
   (c) Let returnValue2 be the result of findBound(currentTableAlias, unboundedTables, fetchClauses, clause2)
   (d) Merge any new clauses of fetchClauses with the same left-hand side, to create FETCH disjunctions (section 3.4).
   (e) Return returnValue2
2. conjunction (clause1 AND clause2)
   (a) Let returnValue1 be the result of findBound(currentTableAlias, unboundedTables, fetchClauses, clause1)
   (b) Let returnValue2 be the result of findBound(currentTableAlias, unboundedTables, fetchClauses, clause2)
   (c) Merge any new clauses of fetchClauses with the same left-hand side, to create FETCH conjunctions 3.4.
   (d) return (returnValue1 $\vee$ returnValue2)
3. negation (e.g, "tab.col $\neq$ 7"), return false
4. relational expression (LHS op RHS)
   (a) If (currentTableType is not a UserReadableTable and (*op* is not "=" or "LIKE") or
   (*LHS* is not of the form currentTableAlias.col) or
   (col is not a defining column for currentTableType)),
   return false
   (b) Let currentTableType be the type of currentTableAlias
   (c) Let referencedTables be the tables referenced in *RHS*
   (d) If |referencedTables $\cap$ unboundedTables|$\neq$ 0,
   return false
   (e) Let newClause be the tuple
   [currentTableAlias, ToString(LHS), op, ToString(RHS), tablesReferenced]
   (f) Add newClauses to fetchClauses
   (g) Remove currentTableAlias from unboundedTables
   (h) return true

Figure 3-13: Pseudocode for **findBound**. Returns true if an item was removed from unboundedTables, false otherwise.

Procedure **merge**(selectStatement, unboundedTables)

Look for WHERE clauses of the form:

t1.col1 = t2.col1 AND
t1.col2 = exp1 AND t1.col2 = exp2

where *t1* and *t2* are different instantiations of the same table T. If found, remove *t1* and *t2* from unboundedTables and execute:

FETCH T(col2 = (exp1 & exp2))

Figure 3-14: Description of **merge**

Procedure **refDependences**(topAlias, selectionTable, tablesReferenced, tablesString, clausesString, processed)

1. Let newDependences be the empty set
2. For each alias predAlias in tablesReferenced and selectionTable but not in processed
   (a) Put predAlias in processed.
   (b) Append to tablesString: TableType(currentTableAlias) + " " + currentTableAlias
      i. Append to tablesString: TableType(predAlias) + " " + predAlias
      ii. For each tuple [currentTableAlias, LHSstring, operator, RHSstring, currentTablesReferenced] corresponding to predAlias in selectionTable where currentTablesReferenced does not contain topAlias,
         A. Append to clausesString: " AND " + LHSstring + " " + operator + " " + RHSstring
         B. Add the elements of currentTablesReferenced to the set newDependences
3. If | newDependences |> 0, call refDependences(topAlias, selectionTable, tablesReferenced, tablesString, clausesString, processed)

Figure 3-15: Pseudocode for **refDependences**. The procedure adds table definitions and clauses of referenced tables to the current FETCH statement.

4. if whereDef is a relational expression (LHS op RHS)

   (a) If (currentTableType is not a UserReadableTable and (*op* is not "=" or "LIKE") or
   (*LHS* is not of the form currentTableAlias.col) or
   (col is not a defining column for currentTableType)),
   return false

   (b) Let currentTableType be the type of currentTableAlias

   (c) Let parentTable be the table from which currentTableType is derived

   (d) Let parentCol be the column through which it is derived

   (e) If *RHS* is parentTable.parentCol where parentTable is the parent table of currentTableType and parentCol is the column through which they are related,
   return true

   (f) If |referencedTables ∩ unboundedTables|≠ 0,
   return false

   (g) If *RHS* is a constant expression

      i. Let newClause be the tuple
      [parentTable, parentCol, ToString(RHS), tablesReferenced]

      ii. Add newClauses to fetchClauses

      iii. Remove currentTableAlias from unboundedTables

      iv. return true

   (h) return false

Figure 3-16: Changed Portion of **findBound** (Figure 3-13) for Derived Tables

```
usage: java FrontEnd [-d] [-log <filename>] [-maxpage #] [-tolinks #] <filename>*
        -d: debug
        -c: creation of tables that already exist illegal
        -log <filename>: output detailed information to a file
        -maxpage #: Maximum size (in Kbytes) of pages to load (default is 30)
        -tolinks #: Minimum number of pages to consider per query (default is 10)
```

Figure 3-17: Usage for Squeal

3.6 Command-Line Interface

Figure 3-17 shows the command-line interface to Squeal. If the "debug" (-d) flag is set, extra information is printed to the standard output stream. If the "creation" (-c) flag is set, it is illegal to redefine a table that already exists; otherwise, the new definition replaces the old one. If the "log" option is used, detailed debugging and status information is written to a file. The variable "maxpage" limits the size of pages to load; by default, the limit is 30 kilobytes. The variable "tolinks" controls how many links should be retrieved from a search engine after making a query; the default is ten. Input files can be provided on the command line.

Chapter 4

Implementation

This chapter goes into low-level detail about the implementation of the Squeal interpreter. It is likely to be of interest to anyone modifying the Squeal interpreter or building a similar system. Readers who are primarily interested in the conceptual ideas of the thesis or in using the system may want to skip to the next chapter, which discusses applications.

The Squeal interpreter is written in Java and runs on a Sparc-20/50 workstation. It communicates through a network using TCP/IP with an Intel-based machine running Microsoft SQL Server. This chapter describes the classes in the Java implementation of Squeal. Outside tools and classes used are:

- Java Compiler Compiler [36], discussed below
- The OROMatcher classes for Perl5 regular expressions [29]
- The HtmlStreamTokenizer class for parsing HTML [8]
- Santeri Paavolainen's implementation of MD5 [30, 34]

To distinguish standard Java classes from those defined for Squeal, Squeal class names are underlined.

4.1 Database state

The class DBstate is used to encapsulate database state. When Squeal is started, a connection to the SQL server is opened and an instance of DBstate is created. The connection is closed when Squeal is terminated. DBstate has one public member variable, **stmt1**, which is of type java.sql.Statement and is used to send a query or update to the SQL server.

4.2 Column

The column class is used to represent each column in tables on the SQL server. The class is needed in interpreting SELECT statements and in displaying results. Member variables (Figure 4-1 hold each column's name and type as well as the Table instance (defined below) of which they are part.

4.3 Tables

4.3.1 Purpose

In addition to the SQL tables stored on the SQL server, the Squeal implementation keeps information locally about every table accessible to the user or system. This is used for interpreting SELECT statements and when user tables are created, deleted, or modified. Every table is an instance of the Table class. Figure 4-2 shows the hierarchy beneath Table, including instance variables. All of

`public String name`
: The name of the column

`public String type`
: The type of the column

`public Table table`
: The Table instance of which this is a component

Figure 4-1: Member Variables for Column

Table
- **computation**
- **creation**
- UserVisibleTable
  - AutomaticTable (section 3.5.2)
    - **link** (section 2.2.3)
    - **page** (section 2.2.1)
    - **parse** (section 2.2.1)
    - **rcontains** (section 2.2.4)
    - **rlink** (section 2.2.4)
    - **tag** (section 2.2.2)
  - DerivedTable (section 3.5.3)
    - **att** (section 2.2.2)
    - **header** (section 2.2.3)
    - **list** (section 2.2.3)
  - UserReadableTable (section 3.5.1)
    - **urls** (section 2.2.1)
    - **valstring** (section 2.2.1)
    - UserDefinedTable

Figure 4-2: Class Hierarchy of Tables

`static Hashtable tables`
Hash table mapping table names to Table instances
`static Vector url_id_columns`
The set of all columns defined in the system of type **url_id** (section 2.2.1)
`public String name`
The name associated with the Table instance
`public Vector columns`
The set of Column instances associated with the Table instance

Figure 4-3: Member Variables for Table

creation		
colname	*type*	*notes*
tab	VARCHAR(15)	primary key
stamp	DATETIME	
def_value_id	INT	

Table 4.1: The **creation** relation

the tables discussed up to now have been subclasses of UserVisibleTable. These include the tables defined in the ontology as well as any tables created by the Squeal user. The Squeal internal tables, which are not subclasses of UserVisibleTable, are discussed later in this section.

4.3.2 Details

There are two static variables associated with Table:

1. *tables*, a HashTable that maps strings to the Table with that name
2. *url_id_columns*, a Vector containing the instances of Column that have type "url_id". This is needed to implement static method *url_id_fixup*, which updates the values of changed **url_id**s, as discussed in Section 2.2.1.

Associated with each Table instance are the table's name and a Vector containing the columns.

Figure 4-4 shows the public methods associated with Table. They support the creation and deletion of SQL tables and provide access to the Column information.

4.3.3 Squeal internal tables

Not previously discussed are the Squeal internal tables, **creation** and **computation**. These are not visible to the Squeal user.

creation

The **creation** table, shown in Table 4.1, is used to keep track of user-defined tables across execution sessions. Table names are limited to 15 characters, while the definition string is unbounded. When the user creates a table, a line is added to **creation**, a new SQL table is created on the server, and a UserReadableTable instance is created. If a table is deleted, the corresponding line is removed from **creation**, the SQL table is dropped, and the instance is freed. Upon Squeal start-up, all of the entries in **creation** are interpreted, to re-create the UserReadableTable instances.

`public static Table createTable(DBstate dbs, String def)`
Create a Table instance from a definition string, creating a table on the SQL server if necessary

`public static void init(DBstate dbs)`
For each definition in the **creation** table, call **createTable**

`public static Table getTable(String n)`
Return the Table instance associated with a name, or `null`

`public static void dropTable(DBstate dbs, String name)`
Permanently delete a Table instance and the associated SQL table, given its name

`public boolean existsP(DBstate dbs)`
Check whether a table with name `name` exists on the SQL server

`public String createIfNecessary(DBstate dbs)`
Create on the SQL server a table with the characteristics of the instance variable, unless the table already exists

`public Column columnMatch(String arg)`
Return the Column instance associated with a name, or `null`

`public static void url_id_fixup(int old_url_id, int new_url_id, ...)`
In all columns of type **url_id**, replace values of `old_url_id` with `new_url_id`

Figure 4-4: Methods Defined for Table

computation		
colname	*type*	*notes*
compute_id	INT	primary key
stamp	DATETIME	
tab	CHAR(15)	
column	CHAR(15)	
value	VARCHAR(255)	
helper	CHAR(15)	
num	INT	

Table 4.2: The **computation** relation

computation

Purpose The **computation** relation is used to keeps track of what implicit or explicit FETCH statements have been performed and when. This can be used to prevent unnecessary recomputation of recently-computed information or to facilitate recomputation of stale information. Each line of the **computation** relation can be thought of as a thunk [1] bundled with the time of its execution and a pointer to its results.

Details The columns of the **compute** relation are shown in Table 4.2. Each **computation** relation has a unique numeric **compute_id**. Also stored are the table name, column name, and column value. For statements using **helper** and **num** arguments, these also appear in the table. The **page**, **rcontains**, **rlink**, and **tag** tables also have a **compute_id** column, not previously discussed. This provides information about when each line of these tables was created. The other tables do not have **compute_id** relations, because they can be deduced from tables on which they depend. While the information currently is not used, it would be easy to modify the system to reload pages that were judged to have expired.

compute_id	stamp	tab	column	value
7875	Oct 8 1997 6:19PM	**rlink**	**source_url_id**	&3&5&

Table 4.3: Entry in **computation** table created in processing the Squeal statement "FETCH rlink(**dest_url_id**=3&5)"

For a recomputation to be prevented, the old and potential **tab** and **column** columns must be identical, and the **value** columns must be identical except in the case of conjunctions and disjunctions. For conjunctions and disjunctions, the **value** string is the set of terms, delimited by "&" or "|", respectively. Table 4.3 shows the **computation** entry created by the Squeal statement:

FETCH rlink(**dest_url_id**=3&5);

If a subsequent delimited conjunction is a substring of the **value** in the table, e.g., "&3&5&", the computation is not redone. For disjunctions, the later **value** must be a superstring of the original **value**. In any case, the terms must appear in the same order for a match to be noted, since textual comparison is used.

The **helper** and **num** fields are set when **tab** is **rcontains** or **rvalue**, to indicate which Web search tool was used as a helper and how many items were requested. If the user issues two FETCH statements involving **rlink** that have different **helper** values, two Web requests will be made. If two statements are identical except for the **num** values, the second FETCH will only be performed if its **num** value is the greater one.

4.4 Exceptions

Class ParasiteException represents Squeal exceptions. It can be instantiated and has the following four subclasses:

- QuitException, thrown when the "quit" statement is executed.
- UnboundVariableException, thrown when a reference to an undefined variable is made.
- UnsupportedProtocolException, thrown when an attempt is made to retrieve a URL whose protocol is not "http".
- UnsupportedURLException, thrown when an attempt is made to retrieve a malformatted URL.

4.5 Representation of variables

4.5.1 SymbolTable

The SymbolTable class is built on top of java.util.HashTable. The hash table component is used to map variable names (of type String) to values (of type Object). A member variable **parent** of class SymbolTable is either set to null or to a parent SymbolTable. Methods are defined to get or put the value of a symbol as well as to check whether a symbol is defined (Figure 4-5).

4.5.2 Bindings

The Bindings class is a subclass of java.Util.Hashtable and is used to represent stack frames. Specifically, the inherited hashtable is used to represent formal-actual argument pairs and a parent symbol table. A Binding instance is created when a Squeal procedure call is made. It is also used to implement FETCH statements. Methods are defined to add, remove, access, or check for the existence of a binding, as shown in Figure 4-6. The methods that create **String**s are used to support the **Computation** table.

`public synchronized boolean containsKey(Object key)`
Return true if **super**.containsKey(**key**) is a key in the constituent hash table.
If not and if **parent** is null, return false. Otherwise, return **parent**.`containsKey`(**key**).
`public synchronized Object get(Object key)`
If **super**.containsKey(**key**), return **super**.get(**key**).
If not and if **parent** is null, return null. Otherwise, return **parent**.`get`(**key**).
`public synchronized Object put(Object key, Object val)`
Execute **super**.put(**key**, **val**).

Figure 4-5: Methods Defined for SymbolTable

`public synchronized Object put(Object key, Object val)`
Call `parent.put(key, val)`
`public synchronized Object get(Object key)`
If **super**.containsKey(**key**), return **super**.get(**key**).
If not and if **parent** is null, return null. Otherwise, return **parent**.`get`(**key**).
`public synchronized Object remove(Object key)`
Call `parent.remove(key)`
`public void extend(Vector v)`
For each namedArg **na** (Section 4.9.1) in **v**,
call `put(na.name, na.value)`.
`public synchronized boolean containsArg(Object key)`
Return true if **super**.containsKey(**key**) is a key in the constituent hash table.
If not and if **parent** is null, return false. Otherwise, return **parent**.`containsKey`(**key**).
`public String toConjunction()`
Return a string representing each key-value pair (k,v) as "$< k >=< v >$",
delimited by " AND " (e.g., "x = 1 AND y = 2").
`public String keysString()`
Return a comma-separated list of the keys in the constituent hash table.
(This if for debugging purposes.)
`public String valuesString()`
Return a comma-separated list of the values in the constituent hash table.
(This if for debugging purposes.)

Figure 4-6: Methods Defined for Bindings

SimpleNode
- NodeWithRequiredName
- NodeWithOptionalName
- NodeContainingList
  - NodeContainingParenthesizedList
- BinaryNode

Figure 4-7: Class Hierarchy of Parser-Generated Nodes

4.6 Parser

Squeal is parsed by the Java Compiler Compiler (JavaCC) [36]. The Squeal grammar, with actions, is shown in Appendix B. JavaCC creates LALR parsers with lookahead one, except when greater lookaheads are explicitly specified in a region of the grammar. The maximum lookahead needed for the Squeal grammar is two.

This section describes the data objects produced by the parser, all of which are subclasses of SimpleNode, as shown in Figure 4-7. The parser prepends "AST" (for "abstract syntax tree") to terminal names to create class names. Table 4.4 shows SimpleNode and its direct subclasses. Parser node classes are shown in Table 4.4.

4.6.1 SimpleNode

The class SimpleNode is provided by JavaCC and modified for Squeal. Methods provided by JavaCC provide access to child nodes. A key method defined for Squeal is *toSQL*, which converts a SimpleNode to a SQL representation so it can be passed to the SQL server. If the node has only one child, the return value is the result of calling *toSQL* on the child. If there are multiple children, the results of recursive calls to *toSQL* are concatenated, separated by space characters. Subclasses of SimpleNode either inherit *toSQL* (as in the case of ASTnumericLiteral, which recursively calls *toSQL* for its one child) or override it (as in the case of ASTcell, which prints its two children separated by a period). Other methods allow a child node to be removed (in order to get rid of an unneeded argument to FETCH) and to find all the tables, cells, or variables referred to in a statement, which is necessary when interpreting SELECT statements. A complete list of methods is shown in Figure 4-8. The following sections describe methods defined for subclasses of SimpleNode.

4.6.2 NodeWithRequiredName

Because so many productions contain a string that needs to be retained in addition to the node type, there is a class NodeWithRequiredName, of which ASTfuncall is a subclass, with the name field set to the function name. NodeWithRequiredName defines methods *setName* and *getName* and overrides methods *toString* and *toSQL* to include the name.

4.6.3 NodeWithOptionalName

NodeWithOptionalName is similar to NodeWithRequiredName, except that setting the name field is optional. The methods *setName* and *getName* are defined, and *toString* is overridden, returning the name if one exists and the empty string otherwise.

4.6.4 NodeContainingList

NodeContainingList is used to represent nodes that consist of a list of child nodes. Specifically, it represents ASTselectList and ASTtableList. It overrides methods *toString* and *toSQL* and defines

`public String toString()`
Return the identifier associated with the instance (JavaCC)
`public int jjtGetNumChildren()`
Return the number of children of the node (JavaCC)
`public SimpleNode jjtGetChild(int i)`
Return the i^{th} child of the node (JavaCC)
`public Object toSQL(SymbolTable symtab)`
Return a legal SQL representation of the node and its children
`public void findTableNames(Vector v)`
Build a vector containing the tables referenced by this node and its children
`public void findCells(Vector v)`
Build a vector containing the ASTcells referenced by this node and its children
`public void findVariables(Vector v)`
Build a vector containing the ASTvariables referenced by this node and its children
`public void removeChild(SimpleNode child)`
Remove the specified child node

Figure 4-8: Methods Defined for SimpleNode, either by JavaCC (as indicated) or purely for Squeal.

SimpleNode	NodeWithRequiredName	NodeWithOptionalName
ASTcell	ASTcolumnDef	ASTaggregateExpression
ASTcolumnsList	ASTcomputeStatement	ASTselectItem
ASTconvertExpression	ASTcreateStatement	
ASTinputStatement	ASTdeffuncStatement	NodeContainingList
ASTlistExpression	ASTdefprocStatement	ASTselectList
ASTnamedArgument	ASTdeleteStatement	ASTtableList
ASTnamedArgumentList	ASTdescribeStatement	
ASTnegationExpression	ASTdropStatement	NodeContainingParenthesizedList
ASTnumericLiteral	ASTfuncall	ASTargList
ASTorderItem	ASTgroupbyDef	ASTsymbolList
ASTorderList	ASThavingDef	
ASToutputStatment	ASThelpStatement	BinaryOperation
ASTparenthesizedExpression	ASTinsertStatement	ASTconjunctionExpression
ASTprintStatement	ASTletStatement	ASTdisjunctionExpression
ASTquitStatement	ASTorderbyDef	ASTlogicExpression
ASTrelExpression	ASTrelRHS	ASTproductExpression
ASTselectStatement	ASTstringLiteral	ASTsumExpression
ASTstar	ASTsymbolLiteral	
ASTstatement	ASTtableName	
ASTunaryExpression	ASTvariable	
ASTupdateStatement	ASTwhereDef	

Table 4.4: Classes of Nodes Created by Parser

method *toVector*, which creates a Vector, each of whose elements is the SQL representation of a child node.

4.6.5 NodeContainingParenthesizedList

NodeContainingParenthesizedList is a subclass of NodeContainingList. It overrides *toSQL* to include parentheses around the list. It is used to represent ASTargList and ASTsymbolList.

4.6.6 BinaryOperation

BinaryOperation is used to represent binary logical and arithmetic operations. The *setOp* method is used to set the operator, and *toSQL* is overridden to call *doBinaryOp*, which, in the case of arithmetic operations, performs the operation if the values of both operands are known, otherwise returning the operator information with the operand in between, to eventually be passed to the SQL server.

4.6.7 Context

The behavior of *toSQL* may depend on the context in which it is called. For example, if the variable "x" is undefined within Squeal, the result of calling *toSQL* on the ASTvariable representing "x" depends on how it is used:

- If the entire statement is "PRINT **x**", then an UnboundVariableException should be thrown.
- If the entire statement is "SELECT **x** FROM usertable", the statement should be passed to the SQL server.

In client mode, the UnboundVariableException would be thrown; in server mode, the representation appropriate for server access is used. Another use is to provide proper syntax for **String**s. For example, the interpretation of **String** s, bound to "foo" depends on whether the value will be sent to the SQL server:

- If the entire statement is "userfunc(s)", *s* should be interpreted as **foo** (without quotes).
- If the entire statement is "SELECT * FROM usertable WHERE col = s", then *s* should be interpreted as **'foo'** (with single quotes).

The Context class has a single instance, which contains a stack, the topmost element of which indicates whether the system is in "client" or "server" mode. The appropriate value is pushed on to the stack when a statement is interpreted. By default, it is in "client" mode. The value "server" is pushed onto the stack upon interpretation of an INSERT, DELETE or UPDATE statement, and popped at the end of its interpretation. Similarly, "client" is pushed at a function call (in case one is nested in an INSERT statement, for example).

4.7 Output

Squeal supports five output streams, shown in Table 4.5. They are built on top of a hierarchy of subclasses of java.io.PrintWriter, shown in Figure 4-9. NullPrintWriter overrides all of PrintWriter's **write**, **print**, and **println** methods to ignore their arguments and do nothing. It is used for an output stream that the user is not interested in viewing. The class LoggingPrintWriter is instantiated for streams that the user wishes to view. The constructor takes three arguments: an output stream (e.g., **System.out**), a name (e.g., "status"), and a PrintWriter **logStream**, to which it echoes whatever it prints, with the stream name prepended. By default, **logStream** is of class NullPrintWriter and no log is created. If the user requests a log via the command-line interface (Section 3.6), it is of type LogPrintWriter, which prints everything it is passed to a file, with a time stamp attached. Figure 4-10 shows an excerpt from a sample log file. The numeric column is the number of milliseconds since the start time, divided by 128.

name	default value	purpose
StatusStream	System.out	prompts and other user-interface communication
OutputStream	System.out	results of computations
DebugStream	(none)	messages for debugging purposes
ErrorStream	System.err	errors
LogStream	(none)	all of the above

Table 4.5: Streams Used by Squeal

java.io.PrintWriter
    NullPrintWriter
    LoggingPrintWriter
    LogPrintWriter

Figure 4-9: Class Hierarchy Based on java.io.PrintWriter

```
0               Creating log at Wed Jul 09 17:16:36 PDT 1997
10      status  Opening database
30      debug   Push: AccessibleBufferedInputStream1dc613f5
30      debug   Debug: Completed initialization
30      status  ->
33      debug   Calling processTree with type class squeal.ASTstatement
33      debug   Calling processTree with type class squeal.ASTdeffuncStatement
```

Figure 4-10: Sample Log Output

`public static void main(String[] args)`
Top-level procedure, containing the call to *init* and the read-eval-print loop.
`public static void init(String[] args) throws ex.ParasiteException`
Code to process the command-line arguments, set up the input streams, and call other initialization routines.
`public static void printUsage()`
Print information about the command-line interface (section 3.6).
`public static Object processTree(SimpleNode node, SymbolTable symtab) throws ex.ParasiteException`
Return the result of evaluating the parsed Squeal statement or expression in *node*.
`public static Object inputFrom(Object argument) throws ex.ParasiteException`
Create an input stream to the specified file.
`static boolean popInputStack() throws java.io.IOException`
Pop the stack of input streams, indicating that input from a stream is complete. Return *true* if more input streams are on the stack, *false* if it is empty.
`public static Object Interpret(String s)`
Called in order to cause a String to be interpreted as a Squeal statement. *Interpret* calls the parser and then *processTree*.
`public static Object get(String s)`
Look up symbol *s* in the top-level symbol table, *globSymtab*.

Figure 4-11: Methods in FrontEnd

4.8 FrontEnd

The class FrontEnd contains the code to start the Squeal interpreter and to repeat the read-eval-print loop. During initialization, a member variable *globalSymtab* of class SymbolTable is created, which holds environment variables (section 3.6 and any values defined at the top-level in the interpreter. Another member variable, *inputStack*, maintains a stack of input streams, initially holding only standard input. Almost all of program execution occurs within the read-eval-print loop. In each iteration, a statement is read from the topmost input stream. If it is a file inclusion statement (*in*), a handle to the specified file is pushed onto the stack. Otherwise, the statement is passed to *processTree*, which consists of a giant case statement to handle different types of Squeal statements. If *processTree* throws a QuitException, then the input stack is popped and execution continues with the new top input stream (or execution ends if the stack is now empty). If *processTree* returns a value, it is printed, and the loop is repeated. FrontEnd also contains the method *Interpret*, which passes a String to the parser and executes it. The *get* method is used when other classes need to access *globalsymtab*. Figure 4-11 shows a complete list of methods in FrontEnd.

4.9 Miscellaneous

4.9.1 namedArg

The class namedArg is used by parsing routines to represent a formal-actual parameter pair, such as for the statement: "FETCH(**url_id**=7)". The member variables **name** and **value** contain the formal and actual parameters, respectively, "**url_id**" and 7 in this example. **namedArg** instances are later used for creating Bindings instances.

`public Vector colNames`
The names of the columns
`public Vector colsizes`
The maximum width of each column
`public Vector rows`
The set of rows, each element of which is a java.util.Vector containing the elements of each row `public int numCols`
The number of columns

Figure 4-12: Member Variables for SelectionResult

4.9.2 Cell

The class Cell is used to represent a cell in a SQL table, e.g., ("utable.colfoo"). Member variables **alias** and **column** contain the table alias and column, respectively, "utable" and "colfoo" in this example.

4.9.3 Junction

The abstract class Junction is used as a superclass for Conjunction and Disjunctions, which are used to represent conjunctions and disjunctions in FETCH statements. Each instance of Junction contains two member variables: **terms**, a java.util.Vector whose elements are the conjuncts/disjuncts, and **separator**, which is either "&" or "|". The method *toString* converts a Junction instance to its printed representation, e.g., "**url_id** = 6 | **url_id** = 7", and is used in constructing queries for the SQL server. The method *toQuotedString* is similar but quotes each term. The method *toList* returns a string containing a comma-separated list of the terms, useful for construction IN clauses of SQL statements.

4.9.4 Set

The class Set provides a simple implementation of mathematical sets. It implements methods *addElement*, *union*, *size*, and *contains*, as well as *elements*, which returns an Enumeration of the elements.

4.9.5 SelectionResult

The class SelectionResult is used to represent the results of a SQL SELECT query. Its member variables are shown in Figure 4-12. The method *display* prints the result to the specified java.io.PrintWriter. The method *toObject* returns the element of the table, if there is only one, and throws a ParasiteException otherwise. It is used for statements of the form: "LET n = (SELECT MAX(**url_id**) FROM **link**".

4.10 Functions and Procedures

4.10.1 UserCallableFunc

Figure 4-13 shows the class hierarchy for user-callable functions, based at UserCallableFunc, which is an abstract type. UserCallableFunc has a pair of static hash tables, **funcHash** and **helpHash**, used to map function names to the function object and help string, respectively. Methods, including constructors (which we have not usually shown), can be found in Figure 4-14.

UserCallableFunc
    UserDefinedFunc
    UserDefinedProc
    JavaDefinedFunc

Figure 4-13: Class Hierarchy for Functions and Procedures

`public UserCallableFunc(String fName, String helpString)`
    Add **(fName, this)** to the function look-up table (**funcHash**) and
    add **(fName, helpString)** to the help look-up table (**helpHash**).
`public UserCallableFunc(String fName)`
    Add **(fName, this)** to the function look-up table (**funcHash**) and
    add **(fName, "no help available")** to the help look-up table (**helpHash**).
`abstract public Object call(SymbolTable s, Vector v, DBstate dbs)`
    Apply the associated function with symbol table **s**, actual parameters and
    database statement **dbs**.
`public static UserCallableFunc getFunc(String fName)`
    Return the function object associated with **fName** (or null, if none is defined
`public static Object call(String s, SymbolTable symtab, Vector v, DBstate dbs)`
    Apply the function named **s** to symbol table **symtab**, actual parameters **v**,
    and database state **dbs**, throwing a Parasite Exception if no such function exists

Figure 4-14: Methods Defined for UserCallableFunc

```
public class JDFstrcat extends JavaDefinedFunc {
    public JDFstrcat() {
        super("strcat", "Concatenate any number of strings");
    }

    public Object call(Vector v, DBstate dbs) {
        StringBuffer outputSB = new StringBuffer();

        for (int i = 0; i ¡ v.size(); i++) {
            Object curObj = v.elementAt(i);
            outputSB.append(Utils.unquote(curObj.toString()));
        }
        return new String(outputSB);
    }
}
```

Figure 4-15: Java-Defined Function Strcat

4.10.2 UserDefinedFunc

The class UserDefinedFunc is used to represent functions defined by the Squeal user. As discussed in Section 3.3, the body of a function is a single expression. When the function is created, the argument list and body of the function are stored in hashed tables, keyed by the new function's name. Upon application, the system verifies that the number of actual parameters is equal to the number of formal parameters and extends the symbol table to include these bindings, then executing the procedure body and returning the result.

4.10.3 UserDefinedProc

The class UserDefinedProc is used to represent procedures defined by the Squeal user. As discussed in Section 3.3, the body of a procedure is a sequence of statements. The behavior of UserDefinedProc is identical to UserDefinedFunc, except that it is the value of the last *statement* (as opposed to the sole *expression*) that is returned.

4.10.4 JavaDefinedFunc

The class JavaDefinedFunc is used as a superclass for user-callable functions defined in Java. It contains the following abstract function, which must be defined in all its subclasses:

```
abstract public Object call(Vector v, DBstate dbs)
```

Figure 4-15 shows the Java code defining JDFstrcat, a function to concatenate strings. Table 3.1 on page 36 shows a list of all current subclasses of JavaDefinedFunc.

4.10.5 SQLfunc

The abstract class SQLfunc is used to represent relations that can be called explicitly through *fetch* or implicitly through *select*. For example, the class SQLfuncLink represents the **link** relation. Note that SQLfunc is not a subclass of UserCallableFunction and that its subclasses cannot be called like regular functions. The key abstract function that must be overridden in any subclass is:

```
abstract public Boolean call(Bindings bindings, DBstate dbs, int compute_id)
```

```
public class SQLfuncRlink extends SQLfunc
{
  public SQLfuncRlink(DBstate dbs) {
    super("rlink", dbs);
    createAutomaticTable(dbs, "rlink",
        "source_url_id url_id, *anchor varchar(255), *dest_url_id url_id");
  }

  public Boolean call(Bindings bindings,
                      DBstate dbs,
                      int compute_id) throws ex.ParasiteException {

    String helper = (String)bindings.get("helper");
    SearchEngine se = SearchEngine.getFunc(helper);
    int num = ((Integer)bindings.get("num")).intValue();
    if (bindings.containsKey("anchor"))
      return se.computePagesContainingAnchor(dbs, compute_id,
                                             num, bindings.get("anchor"));
    else if (bindings.containsKey("dest_url_id"))
      return se.computePagesReferencingDest(dbs, compute_id,
                                            num, bindings.get("dest_url_id"));
    throw new ex.ParasiteException("Unable to process this call to rlink");
  }
}
```

Figure 4-16: Code for SQLfuncRlink

The procedure takes an item of class Bindings (Section 4.5.2) representing the known assignments, the database state, and a **compute_id** (Section 4.3.3). It then adds rows as appropriate to the eponymous relation or throws an exception if unable to do so.

Figure 4-16 shows the implementation of SQLfuncRlink. In the constructor *SQLfuncRlink*, the automatic table (Section 3.5.2) **rlink** is created. Asterisks indicate defining columns 3.4 **anchor** and **dest_url_id**. The procedure call determines the **helper** and **num** bindings and then checks whether **anchor** or **dest_url_id** is bound, calling the corresponding function. If neither is bound, a ParasiteException is thrown.

4.11 SearchEngine

The abstract class SearchEngine encapsulates search engines that can be called by the system. The static method *getFunc*, used in Figure 4-16 maps from a string representing a search engine to the object. Instance methods are shown in Figure 4-17.

4.11.1 AltaVista

The AltaVista class supports all of the SearchEngine methods. For *computePagesContainingText* and *computePagesContainingAnchor*, **value** can be of type String or Junction. For *computePagesReferencingDest*, **value** can be an Integer, representing a **url_id**, or a Junction. Any other types for **value** yield an exception.

`abstract public Boolean computePagesContainingText(DBstate dbs, int compute_id, int num, Object value)`

Add to **rcontents** up to **num** pages reported to contain **value**, returning true on success, false otherwise.

`abstract public Boolean computePagesContainingAnchor(DBstate dbs, int compute_id, int num, Object value)`

Add to **rlink** up to **num** pages reported to contain a hyperlink with anchor text **value**, returning true on success, false otherwise.

`abstract public Boolean computePagesReferencingDest(DBstate dbs, int compute_id, int num, Object value)`

Add to **rcontents** up to **num** pages reported to contain a hyperlink with destination **value**, returning true on success, false otherwise.

Figure 4-17: Methods Defined for SearchEngine

4.11.2 Lycos

The Lycos class supports *computePagesContainingText* with **value** of type String or Junction. Calls to the other functions yield an exception, because Lycos does not support those types of searches.

4.12 Computation

The class Computation supports the FETCH statement, whether it is entered by the user or produced in interpreting a SELECT statement. For example, consider the following FETCH statement:

```
FETCH link(url_id=possible_id) FROM utable u WHERE u.score > 6;
```

These are the steps that take place in the public function *performComputation*:

1. If multiple `namedArgument` items are given, throw out all but the one whose **name** appear earliest in the table definition, retaining **helper** and **num** assignments.
2. Use SimpleNode.*findCells* (Section 4.6.1) to determine what columns are needed from the SQL server (e.g., **utable.possible_id**).
3. Request these columns (e.g., "SELECT u.possible_id FROM utable u WHERE u.score>6"), putting the results in SelectionResult **sr**.
4. For each row of **sr**
   (a) Bind each cell name (e.g., **u.possible_id**) to the row's corresponding value (e.g., 6).
   (b) If a row does not yet exist in COMPUTATION representing this computation:
      i. Call the appropriate function (**link**) with the appropriate bindings (**url_id**=6).
      ii. Add the appropriate line to COMPUTATION

4.13 Selection

The Selection class contains the methods and data for interpreting SELECT statements according to the algorithms described in Section 3.5.2. Each Selection instance consists of a Table, column name, relational operator (i.e., "=" or "LIKE"), and value, plus a Vector of aliases of Tables that must be bounded before the Selection can be determined. Consider the interpretation of the following statement from the Home Page Finder:

```
SELECT DISTINCT u.url_id, 20, 'Base of file name same as name of directory: '
     + v.vcvalue           FROM urls u, parse p1, parse p2, valstring v
WHERE u.url_id IN (SELECT DISTINCT url_id FROM candidate)
AND p1.url_value_id = u.value_id     AND p2.url_value_id = u.value_id
AND v.value_id = u.value_id      AND p1.depth+1 = p2.depth
AND (p1.value = p2.value + '.html'   OR p1.value = p2.value + '.htm');
```

Execution of *performSelectStatement* is as follows:

1. Local variable *unboundedTables* is set to the table aliases (e.g., {u, p1, p2, v})
2. If all tables are user-readable (section 3.5.1), send the statement to the SQL server and return. (In this instance, (*u* (**urls**) and *v* (**valstring**) are user-readable; *p1* and *p2* (**parse**) are not.)
3. If no WHERE clause is present, throw a ParasiteException with the message "In automatic mode, select statements of canonical tables must contain a nontrivial where clause."
4. Group together any conjunctions referring to the same column of the same alias. (In this instance, there are none.)
5. Local variable *numRemaining* is set to *unboundedTables.size*() (e.g., 4)
6. While *numRemaining* > 0
   (a) Call *buildComputeClauses*, which attempts to provide a bound on a member of *unboundedTables*. Procedure *buildComputeClauses* loops through passed variable *unboundedTables* until it finds a table that is bound (i.e., has a defining column set to a known value), as shown by the pseudo-code in figure 3-13. As a side-effect, it removes the bound element from *unboundedTables* and creates **Selection** instances, placing them in *selectionTable*.
   (b) If no tables were bound (i.e., *unboundedTables.size*() = *numRemaining*, throw a ParasiteException with the message: "Unable to provide bound on canonical tables: " + *unboundedTables*; otherwise, decrement *numRemaining*

   In our example, the tables are bounded in this order:

   (a) u, creating the following Selections:
      i. {name = "u", table = **urls**, lvalue = "u.**url_id**", op = "LIKE", rvalue = "(SELECT DISTINCT url_id FROM candidate candidate0)", dependences = { } }
      ii. {name = "u", table = **urls**, lvalue = "u.**value_id**", op = "=", rvalue = "v.value_id", dependences = {v} }
   (b) p1, creating the Selection {name = "p1", table = **parse**", lvalue = "p1.**url_value_id**", op = "=", rvalue = "u.**value_id**", dependences = {u} }
   (c) p2, creating the Selection {name = "p2", table = **parse**", lvalue = "p2.**url_value_id**", op = "=", rvalue = "u.**value_id**", dependences = {u} }
   (d) v, creating the Selection {name = "v", table = **valstring**", lvalue = "v.**value_id**", op = "=", rvalue = "u.**value_id**", dependences = {u} }
7. Determine the order in which the FETCH statement corresponding to each Selection must be computed, honoring the *dependences* values. In our example, an acceptable order is {u, p1, p2, v} because *p1*, *p2*, and *v* depend only on *u*.
8. For each Selection in *selectionTable* that is not a user-readable table, in order:
   (a) Construct the FETCH statement corresponding to the Selection, appending "where (1=1)".
   (b) Add constraints inherited from the dependences (shown in italics below).
   (c) Interpret the FETCH statement. In our example, these would be:

i. "FETCH parse(url_value_id = u.value_id) FROM urls u, valstring v WHERE (1=1) *AND u.url_id IN (SELECT DISTINCT url_id FROM candidate candidate0) AND u.value_id = v.value_id AND v.value_id = u.value_id*"

ii. "FETCH parse(url_value_id = u.value_id) FROM urls u, valstring v WHERE (1=1) *AND u.url_id IN (SELECT DISTINCT url_id FROM candidate candidate0) AND u.value_id = v.value_id AND v.value_id = u.value_id*"

9. Pass the original SELECT statement to the SQL server, which is now able to answer the query.

4.14 Utils

The class Utils contains many useful utility functions, for string processing, node manipulation, Web access, and SQL server access. It also provides an *assert* method, which, if its first argument is false, prints an error message and halts Squeal.

4.14.1 String Manipulation

A number of routines, shown in Figure 4-18, support general string manipulation and conversion among different formats, e.g., Squeal, SQL, and URL formats. For example, percentage signs are used in SQL queries to represent wild cards; *unpercent* removes them for Web queries. *HTMLify* converts Strings into a format appropriate for appearing in a URL, such as by replacing every space character (" ") with a plus sign ("+"). Similarly, to prepare a statement for the SQL server, special characters are protected by *doubleQuotes*, which replaces each single quotation mark with a pair of single quotation marks, and *quotePercents*, which puts brackets around each percentage sign The method *vectorToCommaSeparatedString*. The method *cleanText* assures that Strings returned from the SQL server are properly bounded.

4.14.2 SQL Server Access

Utils provides useful methods for accessing the SQL server, shown in Figure 4-19. The method *getSoleSelection* is used to make a query that returns a single result, such as the **vcvalue** associated with a defined **value_id**. If zero or more than one results are returned, a ParasiteException is thrown. (How this is handled is discussed in the next section.) The method *getSoleSelectionNoFail* is similar but calls *assert* to ensure that one and only one result is returned. It is used for system queries that should always return exactly one result, such as:

```
SELECT MAX(value_id) FROM valstring
```

The method *tableExists* checks whether a table exists on the SQL server. The five versions the method *doInsert* are used for inserting data in a variety of formats into a SQL table. The method *executeUpdate* is used to send an already-constructed String to the SQL server.

4.14.3 Conversion

Utils contains many routines (Figure 4-20) to convert from one type of data to another; for example, from the String "foo" to the associated **value_id**. Most conversions are implemented as simple queries, which are passed to *getSoleSelection*, such as:

```
SELECT value_id FROM valstring WHERE vcvalue = "foo"
```

If the operation fails, the appropriate modification is made to the proper tables (e.g., adding a line to **valstring**) and the query is re-attempted.

`public static String stringSubstituteFirst(String origString, String origSub, String newSub)`
Replace the first instance of *origSub* with *newSub* in *origString*.

`public static String stringSubstitute(String origString, String origSub, String newSub, boolean globalP)`
Replace every (if *globalP* is true) or the first (if *globalP* is false) occurrence of *origSub* with *newSub* in *origString*.

`public static String makeBound(String s, int limit)`
Return a String containing the largest left substring of *s* such that its length is less than *limit*.

`public static String unquote(String s)`
Remove quotation marks from the beginning and end of *s* if any are present.

`public static String unpercent(String s)`
Remove percentage signs from the beginning and end of *s* if any are present.

`public static String HTMLify(String s)`
Convert a String into a format appropriate for using as a URL, e.g., replacing "," with "%2C", etc.

`public static String doubleQuotes(String s)`
Make a legal SQL statement by replacing every single quote with a pair of single quotes.

`public static String quotePercents(String value)`
Replace percentage signs (%) in *value* with [%] in preparation for passing a statement to the SQL server. This effectively quotes the percentage sign.

`public static String cleanText(String s)`
Remove any characters appearing after the null character in *s*.

Figure 4-18: String-Manipulation Methods Defined for Utils

`public static Object getSoleSelection (String sqlString, DBstate dbs) throws ex.ParasiteException`
Pass the SQL statement *sqlString* to the SQL server and return the result. If more than one result is returned by the SQL server, throw a ParasiteException.

`public static Object getSoleSelectionNoFail (String sqlString, DBstate dbs)`
Like *getSoleSelection*, but assert failure (halting the system) if more than one value is returned.

`public static boolean tableExists(String tableName, DBstate dbs)`
Check whether a table named *tableName* exists on the SQL server.

`public static void doInsert(...)`
Create an INSERT statement assigning the specified columns to the specified values for the indicated table and send the statement to the SQL server.

`public static void executeUpdate(DBstate dbs, String q) throws ex.ParasiteException`
Send the String *q* to the SQL server.

Figure 4-19: SQL Server Access Methods Defined for Utils

```
public static int String_to_value_id(DBstate dbs, String value) throws
        ex.ParasiteException
```
Return the **value_id** of the line of **valstring** with *value* as the associated text, creating the line if necessary.

```
public static String value_id_to_String(DBstate dbs, Integer value_id)
        throws ex.ParasiteException
```
Return the text of the line of **valstring** that has *value_id* in the **value_id** field.

```
public static int urlstring_to_url_id(DBstate dbs, String urlstring)
        throws ex.ParasiteException
```
Return the **url_id** of the line of **urls** corresponding to the String *urlstring*.

```
public static int URL_to_url_id(DBstate dbs, URL url, ...)  throws
        ex.ParasiteException
```
Return the **url_id** corresponding to *url*.

```
public static int URL_to_value_id(DBstate dbs, URL url, int compute_id)
        throws ex.ParasiteException
```
Return the **value_id** associated with the string representation of *url*.

```
public static int value_id_to_url_id(DBstate dbs, int value_id, ...)
        throws ex.ParasiteException
```
Return the **url_id** corresponding to the URL whose string representation has a **value_id** of *value_id*.

```
public static URL url_id_to_URL(DBstate dbs, int url_id, ...)  throws
        ex.ParasiteException
```
Return the URL whose **url_id** is *url_id*.

```
public static String url_id_to_urlstring(DBstate dbs, int url_id, ...)
        throws ex.ParasiteException
```
Return the string representation of the URL whose **url_id** is *url_id*.

```
public static String URLtoContentString(URL url, DBstate dbs) throws
        ex.ParasiteException
```
Return the String corresponding to the contents of the page addressed by *url*.

Figure 4-20: Conversion Methods Defined for Utils

`public static Vector ProcessChildrenVector(SimpleNode node, SymbolTable symtab)`
Return a Vector containing the results of interpreting every child of *node*.
`public static Vector ProcessChildrenVectorSQL(SimpleNode node, SymbolTable symtab)`
Return a Vector containing the results of executing *toSQL* on every child of *node*.
`public static SimpleNode scoot(SimpleNode sn)`
Return the earliest descendant of SimpleNode *sn* that has either zero or multiple nodes (i.e., does not have only one child).

Figure 4-21: Methods Defined for Utils

4.14.4 Node Manipulation

Util contains a set of routines for operating on the SimpleNodes returned by the parser. The method *ProcessChildrenVector* calls FrontEnd.*processTree* on all of the children of the passed node, building up a Vector. This is used in interpreting FETCH statements and making Squeal function/procedure calls. The method *ProcessChildrenVectorSQL* is similar, except it evaluates nodes with SimpleNode.*toSQL*. It is used for parsing INSERT statements. The method *scoot* walks down the descendent tree from a node until it reaches a node with zero or more than one children, which it returns. It is used to quickly traverse single strands in the parse tree. These methods are listed in Figure 4-21.

Chapter 5

Applications

In this chapter, I describe some applications, written in Squeal. These applications were originally proposed in my thesis proposal [42].

5.1 Sibs: Finding Similar Pages

One common technique for finding pages similar to those in a given set P is to search for pages with words that appear frequently in the pages of P [13]. Another approach is collaborative filtering, where pages are recommended that were liked by other people who liked those in P; whether a user liked a page is determined by their explicit rating [40]. My technique is a variant type of collaborative filtering where liking is indicated not through explicit ratings but by observing hyperlinks. Specifically, a page that points to elements of P is likely to point to similar pages.

5.1.1 Basic Technique

One could use the following algorithm to find pages similar to $P1$ and $P2$:

1. Generate a list of pages R that reference $P1$ and $P2$.
2. List the pages most commonly referenced by pages within R.

Figure 5-1 shows the Squeal code implementing this algorithm. Figure 5-2 shows a transcript.

5.1.2 Optimizations

Some optimizations/variations are:

1. Only return target pages that include a keyword specified by the user (Figure 5-3). Adding the keyword "women" removed the links to the ACLU and PFAW.
2. Return the names of hosts frequently containing referenced pages (Figure 5-4). This brought to prominence Electronic Policy Network (efn.org) and Close Up Foundation (www.closeup.org).
3. Only return target pages that point to one or both of $P1$ and $P2$ (Figure 5-5). The only such page (besides the NOW home page, which points to itself) was the English Server's page on Feminism and Women's Studies ("http://english-www.hss.cmu.edu/feminism.html"), which points to both.
4. Only follow links that appear in the same list (Section 2.2.3) and under the same header (Section 2.2.3) as the links to $P1$ and $P2$ (Figure 5-6). This causes all of the pages to drop substantially below the original pages. Remaining pages include the ACLU and PFAW.

```
DEFPROC SimPagesBasic(page1id, page2id, threshold)
    // Create temporary data structures
    CREATE TABLE parent(url_id url_id);
    CREATE TABLE results(url_id url_id, score INT);

    // Insert into "parent" the pages that reportedly link
    // to both pages that we care about
    INSERT INTO parent (url_id)
    SELECT DISTINCT r1.source_url_id
    FROM rlink r1, rlink r2
    WHERE r1.source_url_id = r2.source_url_id AND
    r1.dest_url_id = page1id AND
    r2.dest_url_id = page2id;

    // Store the pages pointed to by the parent pages,
    // along with a count of the number of links to them
    INSERT INTO results (url_id, score)
    SELECT l.dest_url_id, COUNT(*)
    FROM link l, parent p
    WHERE l.source_url_id = p.url_id
    GROUP BY l.dest_url_id;

    // Show the URLs of pages most often pointed to
    // and the number of links to them
    SELECT v.vcvalue, COUNT(*)
    FROM link l, parent p, valstring v, urls u
    WHERE l.source_url_id = p.url_id
    AND l.dest_url_id = u.url_id
    AND u.value_id = v.value_id
    GROUP BY v.vcvalue
    HAVING COUNT(*) >= threshold
    ORDER BY COUNT(*) DESC;
ENDPROC;
```

Figure 5-1: Code for SimPagesBasic

The home page for the National Organization for Women (NOW)
? url_id('www.now.org');
1877

The home page for the Feminist Majority Foundation
? url_id('www.feminist.org');
1503

SimPagesBasic(1877, 1503, 3);

vcvalue	
http://www.now.org/	13
http://www.feminist.org/	7
http://www.aauw.org/	4
http://www.aclu.org/	4
http://www.feminist.org/gateway/womenorg.html#top	4
http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mmbt/www/women/writers.html	3
http://www.cs.utk.edu/ bartley/other/ISA.html	3
http://www.democrats.org/	3
http://www.feminist.org/fmf/graphics/navigate.map	3
http://www.igc.org/igc/womensnet/	3
http://www.pfaw.org/	3

Figure 5-2: Transcript of run of SimPagesBasic. The variable **tolinks** (Section 3.6) is set to 20. Acronyms include AAUW (American Association of University Women) and PFAW (People for the American Way).

```
SELECT v.vcvalue, r.score
FROM results r, valstring v, urls u, page pg
WHERE u.url_id = r.url_id
AND v.value_id = u.value_id
AND pg.url_id = u.url_id
AND pg.contents LIKE strcat("%", keyword, "%")
AND r.score >= threshold
ORDER BY r.score DESC;
```

Figure 5-3: Modification to SimPagesBasic (Figure 5-1) to require presence of keyword.

```
SELECT par.value, COUNT(*)
FROM link l, parent p, urls u, parse par
WHERE l.source_url_id = p.url_id
AND l.dest_url_id = u.url_id
AND par.url_value_id = u.value_id
AND par.component = 'host'
GROUP BY par.value
HAVING COUNT(*) >= threshold
ORDER BY COUNT(*) DESC;
```

Figure 5-4: Follow-On to SimPagesBasic (Figure 5-1) to return hosts.

```
SELECT v.vcvalue, r.score
FROM results r, valstring v, urls u, link l
WHERE u.url_id = r.url_id
AND v.value_id = u.value_id
AND l.source_url_id = u.url_id
AND r.score >= threshold
AND (l.dest_url_id = page1id OR l.dest_url_id = page2id)
ORDER BY r.score DESC;
```

Figure 5-5: Follow-On to SimPagesBasic(Figure 5-1) to only return pages pointing to one or more of the original pages.

5.1.3 Evaluation

The text-based approach to recommender systems is used by Excite (www.excite.com), which allows the user to request pages textually similar to those in P. Because Excite can only find pages similar to a single page, not a set of them, we only provide a single URL to each system for each round of the test.

Method

Five subjects agreed to evaluate the system. Each submitted a list of between 10 and 15 URLs that interested him. (All subjects were male.) For each of the URLs, I performed an Excite and ParaSite search for similar pages. One submitted page was "http://www.suntimes.com/ebert/ebert.html", with title value "Roger Ebert on Movies". We will refer to this as the seed URL.

Excite ratings The Excite engine takes a query as input and returns a set of URLs, each with a relevance score, summary, and a link offering to find "more like this". Figure 5-7 shows the first five listings returned for a query on "Roger Ebert".

While the first URL returned from Excite is slightly different from the seed URL, the associated pages have the same contents, which makes them the same for Excite's purposes. In order to get Excite's suggestions for more pages "like" the first one, we follow the appropriate hyperlink. Figure 5-8 shows the top 5 URLs returned by this second query (not showing their summaries and "more like this" links). It is from this list that we obtain the Excite-generated URLs similar to the seed URL. We take the first items returned from this second stage, skipping broken links and pages with the same content as the seed URL and broken links.

Of the first 25 user-submitted URLs that I received, 17 of them could be found in the Excite database.

Parasite ratings The code for Sibs is in Figure 5-9. It only considers links occurring at the same header and list levels as links to the user-specified page. The threshold for the number of common links was set to 2. In our evaluation, the "tolinks" command-line variable was set to 20 and "maxpage" to 30k (section 3.6).

Table 5.1 shows the top 5 URLs yielded by each system for the "Roger Ebert" query.

Of the 17 seed URLs for which Excite was able to generate similar URLs, ParaSite was able to find similar URLs in 13 cases. Table 5.2 summarizes the performance of Excite and ParaSite on the 25 potential seed URLs.

There were thirteen seed URLs for which sufficient pages were generated by each system. These seeds and the recommended URLs were given to the five subjects, with a request to rate the relevance, interestingness, and novelty of each recommended page relative to the seed. The complete instructions are shown in Figure 5-10. Seed URLs and the associated recommendations were displayed as shown in Figure 5-11, with each URL represented as a hyperlink to the specified page and

```
DEFPROC SimPageListHeader(page1id, page2id, threshold)
    CREATE TABLE parent(url_id url_id, hstruct BINARY(6), lstruct BINARY(6));
    CREATE TABLE results(url_id url_id, score INT);

    // Only put pages into "parent" if the links to page1id and
    // page2id appear at the same levels in the list and header;
    // store the hierarchy information with the url_id.
    INSERT INTO parent (url_id, hstruct, lstruct)
    SELECT DISTINCT l1.source_url_id, l1.lstruct, l1.hstruct
    FROM link l1, link l2
    WHERE l1.source_url_id = l2.source_url_id AND
    l1.dest_url_id = page1id AND
    l2.dest_url_id = page2id AND
    l1.lstruct = l2.lstruct AND
    l1.hstruct = l2.hstruct;

    // Store the pages pointed to by the parent pages
    // at the appropriate levels, along with a count of the number
    // of qualifying links
    INSERT INTO results (url_id, score)
    SELECT l.dest_url_id, COUNT(*)
    FROM link l, parent p
    WHERE l.source_url_id = p.url_id
    AND l.hstruct = p.hstruct
    AND l.lstruct = p.struct
    GROUP BY l.dest_url_id;

    // Show the URLs of pages most often pointed to
    // and the number of links to them
    SELECT v.vcvalue, COUNT(*)
    FROM link l, parent p, valstring v, urls u
    WHERE l.source_url_id = p.url_id
    AND l.dest_url_id = u.url_id
    AND u.value_id = v.value_id
    GROUP BY v.vcvalue
    HAVING COUNT(*) >= threshold
    ORDER BY COUNT(*) DESC;
ENDPROC;
```

Figure 5-6: Code for SimPageListHeader only counts links at the same level in the header and list hierarchies as the input pages.

78% Roger Ebert on Movies
URL: http://www.suntimes.com/ebert/
Summary: Roger Ebert on Movies. Search Roger Ebert's Reviews Current Roger Ebert Reviews Roger Ebert Features One Minute Movie Reviews Roger Ebert's The Great Movies Movie Answer Man.
More Like This: Click here to perform a search for documents like this one.

78% Russ Meyer
URL: http://www.well.com/user/jeffdove/russmeyer.html
Summary: A classic piece from 1973 in which Ebert sums up Meyer's career to that point and reflects on his own involvment with Beyond the Valley Of the Dolls. A 1973 interview by Kenneth Turan and Stephen F. Zito with some good biographical background and philosophy on filmmaking information from Meyer.
More Like This: Click here to perform a search for documents like this one.

77% Roger Ebert's Book of Film
URL: http://www.wwnorton.com/catalog/fall96/ebert.htm
Summary: For this delicious, instructive, and vastly enjoyable anthology, Roger Ebert has selected and introduced an international treasury of more than 100 selections that touch on every aspect of filmmaking and filmgoing. Here is a book to get lost in and return to time and time again–at once a history, an anatomy, and a loving appreciation of the central art form of our time.
More Like This: Click here to perform a search for documents like this one.

77% Roger Ebert's Book of Film
URL: http://www.wwnorton.com:81/catalog/fall96/ebert.htm
Summary: For this delicious, instructive, and vastly enjoyable anthology, Roger Ebert has selected and introduced an international treasury of more than 100 selections that touch on every aspect of filmmaking and filmgoing. Here as well are the novelists who have indelibly captured the experience of moviegoing in our lives (Walker Percy, James Agee, Larry McMurtry) and the culture of the movie.
More Like This: Click here to perform a search for documents like this one.

75% Siskel and Ebert's Web Picks
URL: http://w3.arizona.edu/ uab/picks.htm
Summary: Big Book Yellow Pages To help find the theaters.
More Like This: Click here to perform a search for documents like this one.

Figure 5-7: The First Five Listings Returned by Excite Query on "Roger Ebert". All hyperlinks are underlined.

Excite	ParaSite
www.suntimes.com/show/index.html *Showcase*	movieweb.com/movie/movie.html *MOVIEWEB: Home Page*
www.suntimes.com/bruno/bruno.html *Restaurant Reviews*	www.hollywood.com/ *Hollywood Online*
www.suntimes.com/savage/savage.html *Terry Savage on Personal Finance*	us.imdb.com *The Internet Movie Database (IMDb)*
www.suntimes.com/index/ *Chicago Sun-Times Online*	mirrors.yahoo.com/eff/speech.html *Black Thursday*
www.girlsonfilm.com/ *Girls On Film*	movieweb.com/movie/top25.html *MOVIEWEB: Box Office Stats*

Table 5.1: Top 5 Pages Returned by Excite and ParaSite with Ebert Seed URL (www.suntimes.com/ebert/ebert.html). For each page, the URL and title are listed, the title in italics.

99% Roger Ebert on Movies
URL: http://www.suntimes.com/ebert/

98% The Roger Ebert Movie Files
URL: http://www.suntimes.com/ebert/ebertser.html

96% Chicago Sun-Times Online
URL: http://www.suntimes.com/index/

96% Showcase
URL: http://www.suntimes.com/show/index.html

95% Chicago Sun-Times Online
URL: http://www.suntimes.com/index/index.html

Figure 5-8: Top Five Listings Returned by Excite "More Like This" Query on Ebert Seed URL.

```
DEFPROC Sibs(pageid, threshold)
    CREATE TABLE parent(url_id url_id, hstruct BINARY(6), lstruct BINARY(6));

    INSERT INTO parent (url_id, hstruct, lstruct)
    SELECT DISTINCT source_url_id, hstruct, lstruct
    FROM link
    WHERE dest_url_id = pageid;

    SELECT v.vcvalue, COUNT(*)
    FROM link l, parent p, valstring v, urls u
    WHERE l.source_url_id = p.url_id
        AND l.dest_url_id = u.url_id
        AND u.value_id = v.value_id
        AND l.hstruct = p.hstruct
        AND l.lstruct = p.lstruct
    GROUP BY v.vcvalue
    HAVING COUNT(*) >= threshold
    ORDER BY COUNT(*) DESC;
ENDPROC
```

Figure 5-9: Code for Sibs

URL	Excite (%)		ParaSite (#)	
	max	min	max	num
www.weather.com/weather/us/cities/TX_Austin.html	98	98	5	5
www.suntimes.com/ebert/ebert.html	95	92	4	5
us.imdb.com/search	99	98	n/a	0
www.mapquest.com/	95	82	3	5
www.americanair.com:80/aa_home.htm	98	98	3	5
www.texasbbq.com	–	–		
www.geodesic.com	96	94	3	4
www.mos.org/leonardo	82	80	?	?
www.amd.com	93	90	7	5
www.odci.gov/cia/dst/	93	87	n/a	0
www.cnn.com	–	–		
banking.wellsfargo.com	–	–		
www.ebay.com	99	96	3	1
www.metacrawler.com	–	–		
www.roguemarket.com	81	71	2	5
www.artbell.com	98	96	4	2
www.wsdot.wa.gov/regions/northwest/NWFLOW/	–	–		
members.aol.com/gwattier/washjob.htm	–	–		
www.activision.com	97	95	3	5
www.happypuppy.com	92	90	2	4
www.economist.com	86	73	3	5
www.owlnet.rice.edu/ indigo/gsotd/may96.html	90	84	3	5
wombat.doc.ic.ac.uk/foldoc/index.html	–	–		
www.wolfram.com/graphics/	–	–		
www.cs.ubc.ca/nest/imager/contributions/scharein/KnotPlot.html	99	93	6	5

Table 5.2: Performance of Excite and ParaSite on 25 Seed URLs. The top 4 or 5 URLs were picked from those generated by each system, assuming enough were generated. Except when the URL wasn't in Excite's database, indicated by dashes, Excite always generated more than five URLs. The maximum and minimum relevance ratings of the top 5 are listed. ParaSite was only run on URLs in Excite's database. The "max" column indicates the score of the best result, and "num" indicates the number of links available for the evaluation (i.e., with score 2 or more), to a maximum of 5. The URLs that were successful seeds for both systems are listed in bold face.

The purpose of this evaluation is to compare the Web pages suggested by two different systems in response to a seed URL provided by the user. In other words, the user enters a URL (s)he likes or considers interesting, and the system returns suggestions of more URLs to consider.

Each page of the evaluation consists of a seed URL and up to five URLs generated by each of the two systems. Take a look. You should print each of these pages to record your ratings and comments. For each returned URL, you will rate how relevant, interesting, and novel it is relative to the seed URL on a scale of 0 (not at all) to 3 (very much). Here are short descriptions of each of the criteria:

- **relevance**: how closely related to the seed URL
- **interestingness**: how interesting to someone interested in the seed URL
- **novelty**: how likely it is that the user didn't already know how to find the information on the page (i.e., is the suggested URL novel?)

If you were the person to submit the seed URL, give the ratings from your point of view. If you were not the person to submit the seed URL, imagine why they found the seed page interesting and try to answer from their point of view. Write down (or type in) your assumptions. Each URL uses the title of the page as the anchor text. Be sure to follow each link, because the title may not be accurate and may not give enough information. It should not usually be necessary to follow further links. You should also record your subjective impressions on the different recommendations and the different "feel" of the two systems. Please log the time you spend on each page. The first page will probably take you the longest. After that, you should take about 15-20 minutes per page, half on the ratings and half on comments.

Please contact me with any questions by email (ellens@cs.washington.edu), work phone (685-4087), or home phone (882-8669). After you have completed the forms, we can arrange a time for you to drop them off and for me to pay you. Thank you for participating.

There are **13** pages.

Figure 5-10: Instructions for evaluation of Recommender Systems

with the title of the page as the anchor text.

Results

As subjects pointed out, a rating for novelty seemed not to be applicable when a page was entirely irrelevant. For this reason, when "averaging" ratings, novelty was treated as zero when relevance was zero. To capture the simultaneous importance of the three metrics, I also list the product of the averages. A summary over all the pages is shown in Table 5.3. The complete ratings appear in appendix D. When scores for each subject are listed, individuals are identified by a single letter.

On average, the Excite pages were judged more relevant (1.84 vs. 1.36) and interesting (1.63 vs. 1.47) than the ParaSite pages, while the ParaSite pages were judged more novel (1.32 vs. 1.12) and had a higher product (4.58 vs. 4.29). The results of each page's evaluation of can be divided into three cases: those where all of the ParaSite averages are higher (3), where the Excite averages are higher (4), and where the results are mixed (6).[1] I discuss two pages in each category, including the most extreme.

Roger Ebert The recommendations for the Roger Ebert seed URL were shown in Table 5.1. The subject evaluations are shown in Table 5.4. Four of the Excite URLs were other pages at the *Chicago Sun-Times* site, which tended to get low ratings for relevance, interestingness, and especially novelty. The final Excite reference, "Girls on Film", a film-related zine was rated more highly, as were

[1] In one case, KnotPlot, all of the Excite ratings were higher except for novelty, where ParaSite performed better by 1%. I classified this as an entire win for Excite.

Ebert	Excite	0.95	1.30	0.55	0.92
	ParaSite	1.55	1.35	1.00	3.00
	Δ	63%	4%	82%	225%
Austin	Excite	0.95	0.95	0.25	0.23
	ParaSite	1.40	1.85	1.40	5.84
	Δ	47%	95%	460%	2459%
Mapquest	Excite	2.20	1.85	1.35	6.49
	ParaSite	0.90	1.40	0.78	1.59
	Δ	-59%	-24%	-43%	-76%
American	Excite	1.95	1.60	0.95	3.09
	ParaSite	0.99	1.11	1.45	7.00
	Δ	-49%	-31%	53%	127%
Geodesic	Excite	2.25	1.94	0.44	1.90
	ParaSite	2.31	2.13	1.75	9.13
	Δ	3%	10%	300%	381%
AMD	Excite	1.30	1.35	1.35	4.31
	ParaSite	1.81	1.69	0.88	3.34
	Δ	39%	25%	-35%	-23%
Rogue	Excite	1.30	1.35	1.35	4.31
	ParaSite	1.13	1.37	1.50	4.44
	Δ	-13%	1%	11%	3%
Art Bell	Excite	2.25	2.00	0.50	2.25
	ParaSite	0.6	1.00	0.90	0.85
	Δ	-73%	-50%	80%	-62%
Activision	Excite	1.80	1.45	2.15	5.73
	ParaSite	1.73	2.07	1.65	6.79
	Δ	-4%	43%	-23%	19%
Happy Puppy	Excite	2.63	2.19	1.13	7.15
	ParaSite	1.09	1.14	1.10	3.85
	Δ	-58%	-48%	-2%	-46%
Economist	Excite	1.90	1.70	1.80	6.32
	ParaSite	1.00	1.00	0.75	3.41
	Δ	-47%	-41%	-58%	-46%
Geek	Excite	1.94	1.44	0.94	3.12
	ParaSite	1.71	1.83	2.13	7.12
	Δ	-12%	28%	127%	128%
KnotPlot	Excite	2.45	2.05	1.85	9.95
	ParaSite	1.30	1.05	1.87	3.13
	Δ	-47%	-49%	1%	-69%
Average	Excite	1.84	1.63	1.12	4.29
	ParaSite	1.35	1.46	1.32	4.58
	Δ	-27%	-10%	17%	7%

Table 5.3: Averages of Ratings by Seed URL. Ratings were from 0 to 3, with higher numbers better. Seed URLs are listed in Table 5.2.

Seed URL: Roger Ebert on Movies

System A	System B
Showcase	MOVIEWEB: Home Page
Restaurant Reviews	Hollywood Online
Terry Savage on Personal Finance	The Internet Movie Database (IMDb)
Chicago Sun-Times Online	Black Thursday
Girls on Film	MOVIEWEB: Box Office Stats

Figure 5-11: Sample Evaluation Page for Recommender Systems. Underlined text indicates hyperlinks.

P	K	W	S	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
1 1 1	1 1 1	3 3 1	1 1 0	Showcase	1.5	1.5	0.75	1.69
0 0 0	0 1 1	2 3 1	1 1 0	Restaurant Reviews	0.75	1.25	0.25	0.23
0 0 0	0 1 1	1 3 1	1 1 0	Terry Savage on Personal Finance	0.5	1.25	0.25	0.16
0 0 0	0 1 0	3 3 1	0 0 0	Chicago Sun-Times Online	0.75	1	0.25	0.19
1 2 2	2 1 2	2 3 1	0 0 0	Girls on Film	1.25	1.5	1.25	2.34
				Excite averages	0.95	1.30	0.55	0.92
1 1 1	2 2 2	3 3 1	2 2 2	MOVIEWEB: Home Page	2	2	1.5	6.00
0 0 0	2 2 2	3 3 1	2 1 0	Hollywood Online	1.75	1.5	0.75	1.97
0 0 0	2 2 2	2 2 1	3 0 1	The Internet Movie Database	1.75	1	1	1.75
0 0 0	0 0 3	2 1 1	0 0 0	Black Thursday	0.5	0.25	0.25	0.03
1 2 2	1 2 2	3 2 1	2 2 1	MOVIEWEB: Box Office Stats	1.75	2	1.5	5.25
				ParaSite averages	1.55	1.35	1.00	3.00

Table 5.4: User Ratings of Ebert Recommendations. "P", "K", "W", and "S" are pseudonyms for the subjects. The subject who gave the seed is in bold face. The letters "r", "i", and "n" stand for "relevance", "interestingness", and "novelty", respectively.

four of ParaSite's recommendations, all film-related. The remaining ParaSite reference, to "Black Thursday", a Web censorship protest, was judged entirely irrelevant. A representative comment was:

> I think the reason the person [submitting the URL] found this seed page interesting was because they wanted movie reviews. System A [Excite] tended to give results that were more related to where the seed page existed, unlike System B [ParaSite] which gave me the impression that it actually read and interpreted the content. —K

Quantitatively, users found the ParaSite results more relevant (1.55 vs. .95) and novel (1 vs. .55) than the Excite results, and about equally interesting. The product of relevancy, interestingness, and novelty was much higher for ParaSite than Excite (3 vs. .92).

Austin weather The URLs returned by each system for the page entitled "The Weather Channel - Austin, TX" are shown in Table 5.5. Excite returned Weather Channel reports on other cities in Texas and Arkansas, while ParaSite generally returned information, not necessarily weather-related about Austin. The user ratings are shown in Table 5.6. Three of the users, including the one who submitted the seed URL, preferred the ParaSite listings. The fourth reviewer thought that the weather information returned by Excite was more relevant. As with the previous seed, one of the URLs returned by ParaSite was deemed almost entirely irrelevant. Quantitatively, the ParaSite pages were judged more relevant (1.4 vs. .95), interesting (1.8 vs. .95), and novel (1.37 vs. .25) than the Excite pages. The product of the ratings was much higher for ParaSite, 5.29 vs. .23.

Excite	ParaSite
/weather/us/cities/TX_Lamesa.html	www.intellicast.com/weather/aus/
The Weather Channel - Lamesa, TX	Austin, TX
/weather/us/cities/TX_Seminole.html	www.austin360.com
The Weather Channel - Seminole, TX	*Austin 360: THE city site for Austin*
/weather/us/cities/TX_Pecos.html	www.texasmonthly.com/
The Weather Channel - Pecos, TX	*TEXAS MONTHLY WWW RANCH*
/weather/us/cities/TX_Muleshoe.html	pilot.msu.edu/user/ander299/jokes.htm
The Weather Channel - Muleshoe, TX	*Jokes*
/weather/us/cities/AR_Hot_Springs.html	www.auschron.com/current/music.clubs/
The Weather Channel - Hot Springs, AR	*City Beat... 8 Nights of Austin Music*

Table 5.5: Top 5 Pages Returned by Excite and ParaSite with Austin Weather Seed URL (http://www.weather.com/weather/us/cities/TX_Austin.html). For each page, the URL and title are listed. For the Excite URLs, the host name was omitted above; it is "www.weather.com" in each case.

P	K	W	S	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
0 0 0	1 1 0	3 3 1	0 0 0	TWC - Lamesa, TX	1	1	0.25	0.25
0 0 0	1 1 0	3 3 1	0 0 0	TWC - Seminole, TX	1	1	0.25	0.25
0 0 0	1 1 0	3 3 1	0 0 0	TWC - Pecos, TX	1	1	0.25	0.25
0 0 0	1 1 0	3 3 1	0 0 0	TWC - Muleshoe, TX	1	1	0.25	0.25
0 0 0	0 0 0	3 3 1	0 0 0	TWC - Hot Springs, AZ	0.75	0.75	0.25	0.14
				Excite averages	0.95	0.95	0.25	0.23
3 3 3	2 3 2	3 3 1	3 2 3	Untitled	2.75	2.75	2.25	17.02
2 3 2	2 2 2	1 1 1	3 2 2	Austin 360...	2	2	1.75	7.00
1 2 2	1 1 2	1 2 2	2 2 2	TEXAS MONTHLY...	1.25	1.75	2	4.38
0 2 2	0 0 3	1 2 1	0 2 0	Jokes	0.25	1.5	0.25	0.09
1 2 2	0 0 3	1 2 1	1 1 0	City Beat...	0.75	1.25	0.75	0.70
				ParaSite averages	1.40	1.85	1.40	5.84

Table 5.6: User Ratings of Austin Weather Recommendations. "The Weather Channel" is abbreivated "TWC".

Excite	ParaSite
www.amd.com/K6/misc/articles.html *AMD-K6(TM) Related Articles*	www.intel.com/ *Welcome to Intel*
www.lubbockonline.com/news/040297/amd.htm *AMD unveils new K6 chip to rival Intel*	www.apple.com/ *Apple Computer*
www.pcworld.co.nz/pcwtop10/pcwtop10.shtml *@IDG New Zealand*	www.att.com/ *AT&T Home Page*
www.byte.com/art/9611/sec6/ART9.HTM *November 1996 / State Of The Art /* *The x86 Gets Faster with Age*	http://www.baynetworks.com/ Welcome to Bay Networks

Table 5.7: Top 4 Pages Returned by Excite and ParaSite for AMD Seed URL (www.amd.com). Because one of Excite's suggestions turned out to be in a foreign language, only the top 4 suggestions from each were used.

P	**K**	W	S	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
3 3 0	2 2 0	3 3 1	2 1 0	AMD-K6(TM) Related Articles	2.5	2.25	0.25	1.41
3 2 2	2 1 0	3 3 1	2 1 0	AMD unveils ...	2.5	1.75	0.75	3.28
3 3 2	1 1 3	3 3 3	2 1 1	IDG New Zealand	2.25	2	2.25	10.13
3 0 1	1 1 3	3 3 3	2 3 1	November 1996 ...	2.25	1.75	2	7.88
				Excite averages	2.38	1.94	1.31	5.67
3 3 1	3 3 0	3 3 1	3 2 2	Welcome to Intel	3	2.75	1	8.25
3 3 1	2 1 0	1 1 1	1 2 2	Apple Computer	1.75	1.75	1	3.06
2 2 1	1 1 0	0 1 1	2 1 1	AT&T Home Page	1.25	1.25	0.5	0.78
2 2 2	0 0 3	1 1 1	2 1 1	Welcome to Bay Networks	1.25	1	1	1.25
				ParaSite averages	1.81	1.69	0.88	3.34

Table 5.8: User Ratings of AMD Recommendations

AMD The URLs returned by each system for the "Advanced Micro Devices" (AMD) home page are shown in Table 5.7. Excite generated URLs for pages with information specific to AMD, as the titles suggest. ParaSite generalized differently, returning the home pages of other computer companies. As one user wrote:

> It depends on the orientation of the originator
> is s/he interested in the company, in chips, in investment, in employment or what? —P

Three out of the four users considered the Excite results superior, giving it a higher score on all metrics, especially relevance and novelty, as shown in Table 5.8.

Geek Site of the Day

The URLs returned by each system for the "Geek Site of the Day" (GSotD) are shown in Table 5.9. Because ParaSite only made four recommendations, only the top four Excite recommendations are listed. Two of the Excite recommendations were articles about GSotD, one reviewed GSotD and similar sites, and one was a GSotD archive. The ParaSite selections were more diverse: the first two were collections of cool/useless pages, the next was the home page of "CNET: The Computer Network", and the fourth was the Museum of Bad Art. User ratings are shown in Table 5.10. The Excite pages were considered more relevant (1.94 vs. 1.71), while the ParaSite pages were considered more interesting (1.83 vs. 1.44) and novel (2.13 vs. 0.94). Users disagreed as to which system was preferable:

Excite	ParaSite
www.webcrawler.com./News/site.06.html *WebCrawler*	cool.infi.net *Cool Site of the Day*
www.owlnet.rice.edu/ indigo/gsotd/pcnovice.html *PC Novice mentions GSOTD*	www.go2net.com/internet/useless/ *go2net | internet | The Useless Pages*
www.owlnet.rice.edu/~indigo/gsotd/sept95.html *Geek Sites of the Day, September 1995*	www.cnet.com *Welcome to CNET.COM*
www.newsherald.com/BUSINESS/B20.HTM *News Herald Business Wire: How to keep up with everything that's cool*	www.glyphs.com/moba/ *The Museum of Bad Art*

Table 5.9: Top 4 Pages Returned by Excite and ParaSite for GSotD Seed URL (www.owlnet.rice.edu/~indigo/gsotd/).

P	K	W	**S**	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
1 1 1	2 1 0	3 3 3	3 2 1	WebCrawler	2.25	1.75	1.25	4.92
2 1 0	2 1 0	1 1 0	1 0 0	PC Novice mentions GSOTD	1.5	0.75	0	0.00
3 3 0	2 1 0	3 3 3	1 0 0	GSotD, September 1995	2.25	1.75	0.75	2.95
3 2 3	1 1 3	0 1 1	3 2 1	News Herald Business Wire	1.75	1.5	1.75	4.59
				Excite averages	1.94	1.44	0.94	3.12
3 3 3	2 2 2	0 1 2	3 1 2	Cool Site of the Day	2	2.25	2	9.00
3 3 3	2 2 2	2 2 3	3 3 3	go2net: The Useless Pages	2.08	2.08	2.5	10.85
3 3 3	0 0 1	1 1 3	- - -	Welcome to CNET.COM	1.75	1.75	2	6.13
1 2 3	1 1 3	2 2 2	3 3 2	The Museum of Bad Art	1	1.25	2	2.50
				ParaSite averages	1.71	1.83	2.13	7.12

Table 5.10: User Ratings of Geek Site of the Day Recommendations. Dashes indicate where a user neglected to rate a page.

> System A [Excite] came up with one good suggestion. System B [ParaSite] came up with several. System B wins... —P
>
> I assume that the person wants sites that would be interesting or funny to the computer geek, such as things in poor taste. In this case I would choose system A. —W

MapQuest The URLs returned by each system for the "MapQuest!" home page are shown in Table 5.11. All 5 sites returned by Excite were highly-relevant map-related sites. The 5 sites returned by ParaSite were all related to travel but much less directly, such as tourist information about San Diego. The user ratings are shown in Table 5.12. Excite was rated better for all measures.

MapQuest The KnotPlot Site contains pictures of mathematical knots and links generated by a program called KnotPlot by its author Rob Scharein. The URLs returned by each system for the KnotPlot page are shown in Table 5.13. Three of the sites returned by Excite were knot-related sites on the same machine; one other was an announcement of a talk Scharein gave on KnotPlot; and the fifth was the home page of a topologist with a link to the KnotPlot Site. Of the five ParaSite suggestions, only one was of similar quality: Rob Scharein's home page. Three of the remainder were math-related pages, and one was the home page of an individual with no apparent relevance. The user ratings are shown in Table 5.14. The two systems got approximately the same score for novelty, and the Excite pages were judged significantly more relevant and interesting.

Excite	ParaSite
www.mckinley.com/magellan/Reviews/ News_and_Reference/Geography_and_Maps/ index.magellan.html *Geography & Maps Topics*	www.lib.utexas.edu/Libs/PCL/Map_ collection/Map_collection.html *PCL Map Collection*
netfind.aol.com/aol/Reviews/News_and_ Reference/Geography_and_Maps/ index.netfind.html *AOL NetFind: Reviews: Geography & Maps*	metro.jussieu.fr:10001/bin/cities/english *Subway navigator*
www.yumasun.com/-feat/Net/netmaps.html *Maps on Net*	pubweb.parc.xerox.com/map *Xerox PARC Map Viewer*
www.geosys.com/ *GeoSystems: Welcome!*	www.geocities.com/HotSprings/4150 *Inside San Diego*
elmo.webcrawler.com/select/trref.24.html *MapQuest*	www.geocities.com/HotSprings/4150/info.html *Inside San Diego: INFORMATION REQUEST*

Table 5.11: Top 5 Pages Returned by Excite and ParaSite for MapQuest Seed URL (http://www.mapquest.com).

P	K	W	**S**	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
2 2 3	2 2 2	3 3 1	3 2 1	Geography & Maps Topics	2.5	2.25	1.75	9.84
2 3 2	2 2 2	3 3 1	3 2 2	AOL NetFind...	2.5	2.5	1.75	10.94
1 2 1	2 2 2	3 2 1	3 2 2	Maps on Net	2.25	2	1.5	6.75
1 2 3	1 1 1	3 2 1	3 1 1	GeoSystems: Welcome!	2	1.5	1.5	4.50
0 0 0	2 0 0	3 3 1	2 1 0	MapQuest	1.75	1	0.25	0.44
				Excite averages	2.20	1.85	1.35	6.49
0 1 2	1 1 3	3 3 2	3 2 0	PCL Map Collection	1.75	1.75	1.125	3.45
1 1 1	0 0 3	2 3 2	1 3 2	Subway navigator	1	1.75	1.25	2.19
0 1 1	0 0 3	3 3 1	2 3 3	Xerox PARC Map Viewer	1.25	1.75	1	2.19
0 1 1	0 0 3	1 2 1	0 1 1	Inside San Diego	0.25	1	0.25	0.06
0 1 1	0 0 3	1 2 1	0 0 0	Inside San Diego: INFO...	0.25	0.75	0.25	0.05
				ParaSite averages	0.90	1.40	0.78	1.59

Table 5.12: User Ratings of MapQuest Recommendations

Excite	ParaSite
www.cs.ubc.ca/nest/imager/contributions/scharein/KnotSquare.html *The Knot Square*	www.cs.ubc.ca/spider/scharein *Rob Scharein's Main WWW Page*
www.cs.ubc.ca/nest/imager/contributions/scharein/knot-theory/fox-knot.html *A Fox's Quick Introduction to Knot Theory*	www.hk.super.net/~cismath *CIS Mathematics Department*
www.forum.swarthmore.edu/news.archives/geometry.announcements/article212.html *Untitled*	forum.swarthmore.edu/ *The Math Forum Home*
www.cs.ubc.ca/nest/imager/contributions/scharein/knot-theory/references.html *Excellent References on Knot Theory*	forum.swarthmore.edu/maw/ *Math Awareness Week 1997*
www.ma.utexas.edu/~sedgwick/ Eric Sedgwick	users.aol.com/wkrulac/bill.htm Bill Krulac's Home Page

Table 5.13: Top 5 Pages Returned by Excite and ParaSite for KnotPlot Seed URL (www.cs.ubc.ca/nest/imager/contributions/scharein/KnotPlot.html)

P	K	W	**S**	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
3 3 3	3 3 1	3 3 2	1 0 0	The Knot Square	2.5	2.25	1.5	8.44
3 3 3	2 3 1	3 3 2	3 3 2	A Fox's Quick Introduction...	2.75	3	2	16.50
3 1 2	2 1 3	2 2 1	1 0 0	Untitled	2	1	1.5	3.00
3 3 3	2 1 3	3 3 2	1 0 0	Excellent References...	2.25	1.75	2	7.88
3 3 3	2 1 3	3 3 2	3 2 1	Eric Sedgwick	2.75	2.25	2.25	13.92
				Excite averages	2.45	2.05	1.85	9.95
- - -	1 0 3	3 3 2	2 2 2	Rob Scharein's... Page	2	1.5	2.33	7.00
1 1 3	1 0 3	1 1 1	2 1 1	CIS Mathematics Department	1.25	0.75	2	1.88
2 2 2	1 1 3	2 3 1	1 1 1	The Math Forum Home Page	1.5	1.75	1.75	4.59
1 1 2	1 0 3	2 2 2	1 0 1	Math Awareness Week 1997	1.25	0.75	2	1.88
1 1 3	0 0 3	1 1 2	0 0 0	Bill Krulac's Home Page	0.5	0.5	1.25	0.31
				ParaSite averages	1.30	1.05	1.87	3.13

Table 5.14: User Ratings of KnotPlot Recommendations

URL	count	description
www.cs.ubc.ca/spider/scharein	6	Rob Scharein's home page
www.geom.umn.edu/	4	The Geometry Center
www.geom.umn.edu/~scharein/knotplot/		KnotPlot-related pages...
resource/www/CommandWindow.html	4	
resource/www/ControlPanel.html	4	
resource/www/KnotPlotManual.html	4	
resource/www/KPManOverView.html	4	
resource/www/ViewWindow.html	4	
www.math.uiowa.edu/knotplot/		KnotPlot mirror pages...
CommandWindow.html	4	
ControlPanel.html	4	
KnotPlotManual.html	4	
KPManOverView.html	4	
ViewWindow.html	4	
www.metacreations.com/	4	A computer visualization company
www.earlham.edu/suber/knotlink.htm	3	A different knots page
www.unitedmedia.com/comics/dilbert/	3	The Dilbert comic strip

Table 5.15: Pages Returned by ParaSite for KnotPlot with tolinks=40

URL	count	description
www.yahoo.com/	6	Yahoo!
www.cmpnet.com/	3	CMPnet: The Technology Network
www.geocities.com/HotSprings/4150/x.html	3	[broken link]
www.mckinley.com/	3	Magellan Internet Guide
www.netguide.com/	3	NetGuide: Your Guide to the Net
www.usps.gov/ncsc/	3	USPS zip code information

Table 5.16: Pages Returned by ParaSite for MapQuest with tolinks=40

Discussion

The ParaSite suggestions were judged more novel, while the Excite ratings were judged more relevant and interesting. In some cases, one system was markedly superior to the other. Some possible conclusions are:

1. The text-based approach is likelier than the structure-based approach to stay within the seed web site, yielding pages that users find more relevant but less novel.
2. Neither of the two approaches is always superior. Whether the text- or structure-based approach is better depends on the type of link and the user's purpose.
3. A superior system could be built by combining the two approaches.
4. The structure-based approach would have generated more useful results if more pages had been examined for each seed URL.

To test the last hypothesis, I reran the tests for "KnotPlot" and "MapQuest" with **tolinks** doubled to 40, meaning that the system examined up to forty pages pointing to the seed URLs. The KnotPlot run took an hour. Table 5.15 shows the URLs with ratings of three or greater. Because the subjects were no longer available to me, I have no ratings for the recommendations, but they appear to be much better than the original ParaSite recommendations. The MapQuest run took two-and-a-half hours. The results, shown in Table 5.16, do not appear to be much better. This suggests that, if enough pages are examined, the structure-based approach is superior to the text-based approach in most but not all cases. Further testing is desirable.

Related work

Collaborative filtering [40] is based on the assumption (also underlying parts of this work) that some people have the same taste as others. Consequently, if two people have some judgments in common, something liked by the first person should be recommended to the second person. The similar page finder can be seen as an application of collaborative filtering. While collaborative filtering systems have required users to explicitly rate items, there is no reason implicit ratings could not be used, as is the case with data mining [9].

Also related is bibliometrics, the statistical study of documents, which includes citation indexing [37]. Binary relations studied include *bibliographic reference*, where one paper references another; *bibliographic coupling*, where two papers share a common reference [17]; and *co-citation*, where two papers are referenced by a common source [41]. Co-citation is equivalent to ParaSite's judging two web documents similar if they are both referenced by the same page. As with links among web pages, it has been observed that there are many different motivations behind citations, including people's tendency to cite their own papers and to engage in quid pro quos [37]. The term "sitation" has been coined by Gerry McKiernan to describe the study of links between Web pages [35].

5.2 A Home Page Finder

In Section 1.1.3, heuristics for a program to find personal home pages were discussed. The primary trick was to look at hyperlinks to find pages that others have labeled as being a specific person's home page; i.e., to look for pages that are the destination of hyperlinks whose anchor text contains the person's name. Further heuristics take advantage of the tag structure within a page and the structure of candidate URLs. We can thus build an application to find home pages given a person's name:

1. Consider a page to be a candidate if it is the destination of a hyperlink with anchor text containing the person's name (and, ideally, nothing else).
2. Give a bonus to candidate pages with any of these characteristics:
   - A tag attribute contains the full name.
   - The full name appears within title or header tags.
   - The URL is of the form "<foo>/<foo>.html".
   - The URL file name is "index.html" or "home.html" or the empty string.
   - The final directory name in the URL starts with a tilde ( ) (the Unix convention for a user's home directory).
   - The penultimate directory name in the URL is "people" or begins with "home".
3. Display pages with the highest scores, allowing the user to find out the system's reasons.

The top-level code for the home page finder is shown in Figure 5-12. The rest of the code appears in Appendix C.

5.2.1 Sample Run

Here is a transcript of a run of the home page finder and subsequent queries:

HomePage('Pattie Maes');

```
DEFPROC HomePage(fullname)
    CREATE TABLE possibleParent(url_id url_id, urlstring VARCHAR(255), reason CHAR(255));
    CREATE TABLE candidate(url_id url_id, score INT, reason CHAR(255));
    CREATE TABLE results(url_id url_id, score INT);

    HomePageCore(fullname);
    // Fill in the results table
    INSERT INTO results (url_id, score)
    SELECT c.url_id, SUM(c.score) AS tot
    FROM candidate c
    GROUP BY c.url_id;

    // Display the results
    SELECT u.url_id, v.vcvalue, r.score
    FROM valstring v, results r, urls u
    WHERE u.url_id = r.url_id
    AND u.variant = 1
    AND v.value_id = u.value_id
    ORDER BY r.score DESC;
ENDPROC;
```

Figure 5-12: Top level code for home page finder. HomePageCore is defined in Appendix C.

url_id	vcvalue	score
4279	http://pattie.www.media.mit.edu/people/pattie/	33
4539	http://lcs.www.media.mit.edu/people/pattie/	23
4315	http://www.media.mit.edu/ pattie	16
4395	http://altavista.digital.com/cgi-bin/query?pg=aq&what=web &fmt=.&q=Pattie+Maes%0D%0A%0D%0A%0D%0A &r=Pattie+Maes&d0=&d1=	3
4380	http://www-pcd.stanford.edu/pcd-archives/ pcd-seminar/1993-1994/0012.html	1
4392	http://delegate.tokai-ic.or.jp:18080/InfoServ/ Artec/artec017.htm	1
4574	ahttp://www.vpro.nl/www/arteria/maxk/maxk-dop23.html	1
4584	http://lcs.www.media.mit.edu/people/lieber/ Teaching/Int-Int/Int-Int-Announcement.html	1

What are the scoring reasons behind the top page?

SELECT score, reason FROM candidate WHERE url_id = 4279 ORDER BY score DESC;

score	reason
10	File name is " "
10	Penultimate directory is: "people"
2	Anchor from "http://lcs.www.media.mit.edu/groups/agents/papers.html" is the full name: "Pattie Maes"
2	Anchor from "http://nif.www.media.mit.edu/" is the full name: "Pattie Maes"
2	Anchor from "http://www-white.media.mit.edu/vismod/demos/smartroom/contributors/contrib.html" is the full name: "Pattie Maes"
2	Anchor from "http://www-yano.is.tokushima-u.ac.jp/research/moo/ismoo-e.html" is the full name: "Pattie Maes"
1	Anchor from "http://aries.www.media.mit.edu/people/aries/home-page.html" includes the full name: "Prof. Pattie Maes"
1	Anchor from "http://lcs.www.media.mit.edu/groups/agents/papers.html" includes the full name: "Pattie Maes"
1	Anchor from "http://nif.www.media.mit.edu/" includes the full name: "Pattie Maes"
1	Anchor from "http://www-white.media.mit.edu/vismod/demos/smartroom/contributors/contrib.html" includes the full name: "Pattie Maes"
1	Anchor from "http://www-yano.is.tokushima-u.ac.jp/research/moo/ismoo-e.html" includes the full name: "Pattie Maes"

Give the reasons for the scores of the pages below 5.

SELECT c.url_id, c.score, c.reason
FROM candidate c, results r
WHERE r.score < 5 **AND c.url_id = r.url_id;**

url_id	score	reason
4380	1	Anchor from "http://www-pcd.stanford.edu/pcd-archives/pcd-seminar/1993-1994/0013.html" includes the full name: "Terry Winograd: "PCD 1/14: Pattie Maes, MIT, Learning Interface Agents""
4392	1	Anchor from "http://www.mediamatic.nl/whoiswho/Maes/PattieMaes.html" includes the full name: "Pattie Maes and Bruce Blumberg"
4395	2	Anchor from "http://www.mediamatic.nl/whoiswho/Maes/PattieMaes.html" is the full name: "Pattie Maes"
4395	1	Anchor from "http://www.mediamatic.nl/whoiswho/Maes/PattieMaes.html" includes the full name: "Pattie Maes"
4574	1	Anchor from "http://www.vpro.nl/htbin/scan/www/arteria/maxk/maxk-dop07.html" includes the full name: "061194: Pattie Maes"
4584	1	Anchor from "http://lcs.www.media.mit.edu/people/lieber/Teaching/Teaching.html" includes the full name: "Intelligent Interfaces Seminar <I>Pattie Maes and Henry Lieberman</I>"

The system didn't recognize that the top three pages returned were identical, because their servers have different names. Let's cause it to sum the scores of pages based on their MD5 hash value.

SELECT p.md5, SUM(r.score) as score
FROM page p, results r
WHERE p.url_id = r.url_id
GROUP BY p.md5;

md5	score
1c4697007709d157ca508d500edde0bc	72
75715d6305d48582c577ccbb7ce0a25a	1
8c6a9679daede982bda0af2a8a9023ae	1
9cf56d81c2a3be5368f8126f9721aadf	1
aa3c0dc8ebdd3409f46d5d5113eeb3dc	3
ca9cca3ec2c4acdf199906f9611ae28f	1

Show the totals by url_id.

```
SELECT r.url_id,
      (SELECT sum(r2.score) FROM
      results r2, page p2
      WHERE r2.url_id=p2.url_id AND p2.md5=p.md5) AS cnt
FROM results r, page p
WHERE r.url_id = p.url_id
AND r.url_id =
(SELECT MIN(r3.url_id) FROM results r3, page p3 WHERE p3.url_id =
   r3.url_id AND p3.md5 = p.md5);
```

url_id	score
4584	1
4574	1
4395	3
4392	1
4380	1
4279	72

Tell me which of the pages contains "Lieberman".

```
SELECT r.url_id
FROM results r, page p
WHERE p.url_id = r.url_id AND p.contents LIKE '%Lieberman%';
```

url_id
4584

5.2.2 Support for nicknames

A problem with the above home page finder is that it uses a single string to represent a person's name. A human being would know that "Ken Haase" and "Kenneth Haase" might be the same person, even though the strings are different. To implement a home page finder that can make use of different versions of the same name, we create the table **names**, shown in Figure 5-13, and the code in Figure 5-14.

5.2.3 Evaluation

Names were taken from David Aha's list of Machine Learning and Case-Based Reasoning Home Pages (http://www.aic.nrl.navy.mil:80/ aha/people.html), which was chosen because it was used in the Ahoy! study [39]. The home page finder without nicknames was used (Figure 5-12), and the Squeal interpreter was prevented from making use of Aha's list or its mirrors. Of the first fifteen hyperlinks appearing under the letter "A", twelve links still worked. We also excluded the link for "David Aha", considering him a special case. Of the eleven remaining names, ParaSite successfully found nine home pages (82%) and failed in two cases (18%), as shown in Table 5.17. In one successful case, ParaSite found a better link than the one on Aha's list! Aha's entry for Lloyd Allison pointed to a one-page profile (http://www.cs.monash.edu.au/people/profiles/lloyd.html), while ParaSite returned a link to a more traditional home page (entitled "Lloyd Anderson - Home Page") that contained roughly two dozen links (http://www.cs.monash.edu.au:80/ lloyd/index.html). ParaSite unequivocally failed in

number	name
1	Ken
1	Kenneth
1	Kenny
1	Kennie
2	Deborah
2	Debby
2	Deb
2	Debbie
3	Thomas
3	Tom
3	Tommy

Figure 5-13: The "names" Table

```
DEFPROC HomePageWithNicknames(firstName, lastName)
    CREATE TABLE possibleParent(url_id url_id, urlstring VARCHAR(255), reason CHAR(255));
    CREATE TABLE candidate(url_id url_id, score INT, reason CHAR(255));
    CREATE TABLE results(url_id url_id, score INT);

    LET nameNumber = (SELECT number FROM names WHERE name=firstName);

    FETCH HomePageCore(fullname=strcat(n.name, ' ', lastName))
    FROM names n
    WHERE n.number = nameNumber;

    // Fill in the results table
    INSERT INTO results (url_id, score)
    SELECT c.url_id, SUM(c.score) AS tot
    FROM candidate c
    GROUP BY c.url_id;

    // Display the results
    SELECT u.url_id, v.vcvalue, r.score
    FROM valstring v, results r, urls u
    WHERE u.url_id = r.url_id
    AND u.variant = 1
    AND v.value_id = u.value_id
    ORDER BY r.score DESC;
ENDPROC;
```

Figure 5-14: Top Level Code for HomePageWithNicknames. HomePageCore is defined in Appendix C.

name	# returned	# correct	notes
Agnar Aamodt	2	2	separate pages for English, Norwegian
Gennady Argre	0	0	no anchors found
Kamal Ali	1	1	
Carolyn Allex	3	3	all equivalent
Lloyd Allison	5	1	better match than Aha
Ethem Alpaydin	1	1	
Rick Alterman	2	1	
Klaus-Dieter Althoff	3	2 or 3	
Tim Andersen	1	1	not counting two bad links returned
Bill Anderson	3	0	other Bill Andersons
Chuck Anderson	4	1	other Chuck Andersons

Table 5.17: Results of Home Page Finder on Names from Aha's List. Pages were only considered correct if they pointed to the *home page* of the specified individual.

Lenore Blum
Lenore Blum 1943- Written by Lisa Hayes, Class of 1998 (Agnes Scott College) Lenore Blum was a bright and artistic child who loved math, art, and music from her original introductions to them. Blum finished high school at the age of 16, after which...
http://www.scottlan.edu/lriddle/women/BLUM.HTM, 5359 bytes, 27Apr97

Figure 5-15: Blurb returned from HotBot in response to the query "Lenore Blum 1943"

its search for Bill Anderson, returning pages relevant to other people with the name. In the case of Chuck Anderson, which I considered a win, links were returned both to the intended Chuck Anderson and to others, clearly a hazard of only using names for searches.

5.3 Bo Peep: Finding Moved Pages

Search engines frequently return obsolete URLs. In 1995, Selberg and Etzioni found that 14.9% of the URLs returned by popular search engines no longer point to accessible pages [38]. With the growth and aging of the web since their measurements, the percent of obsolete URLs returned may now be even higher. Currently, there are no utilities that try to track down moved pages.

5.3.1 Technique 1: Climbing the directory hierarchy

Figure 5-15 shows an AltaVista blurb containing a URL U_{bad} that is no longer valid. I describe a heuristic algorithm and application for finding a new URL U_{new} for the page, given U_{bad} and the title *title* of the page. In this example, U_{bad} ="http://www.scottlan.edu/lriddle/women/BLUM.HTM", and *title* = "Lenore Blum". (While the examples in this chapter involve topics that interest me, in no case are pages under my control involved in the results.)

We can create URL U_{base} by removing directory levels from U_{bad} until we obtain a valid URL. We can then crawl from U_{base} in search of a page with title *title*. This is based on the heuristic that someone who cared enough about the page to house it in the past is likely to at least link to the page now. Figure 5-16 shows code that creates a "candidate" data structure and seeds it with the parent of U_{bad}. Figure 5-17 shows how the **parse** and **valstring** tables might appear after the procedure is executed.

Figure 5-18 shows the complete program. The strategy is:

1. Let round = 0;
2. Insert the parent URL (as possible_url_id) and 0 (as round) into the **candidate** table.
3. Repeat:

```
DEFPROC bopeep_init(old_url_value_id)
    // Create a structure for candidates
    CREATE TABLE candidate (possible_url_id url_id, round INT);

    // Get an unused value_id
    LET new_value_id = value_id();

    // Copy all of the parse elements except for the file name
    // from the old_url_value_id to new_value_id
    INSERT INTO parse (url_value_id, component, value, depth)
    SELECT new_value_id, component, value, depth
    FROM parse
    WHERE url_value_id = old_url_value_id AND depth <> 1;

    // Set a null file name in the parse table for new_value_id
    INSERT INTO parse (url_value_id, component, value, depth)
    VALUES (new_value_id, 'path', '', 1);

    // Put the new url_id in the candidate table
    LET new_url_id = (SELECT url_id FROM urls WHERE value_id = new_value_id);
    INSERT INTO candidate (possible_url_id, round)
    VALUES (new_url_id, 0);
ENDPROC;
```

Figure 5-16: Bo Peep: Code to climb one up in the directory hierarchy

parse

url_value_id	component	value	depth
1	host	www.scottlan.edu	0
1	port	80	0
1	path	BLUM.HTM	1
1	path	women	2
1	path	lriddle	3
2	host	www.scottlan.edu	0
2	port	80	0
2	path		1
2	path	women	2
2	path	lriddle	3

valstring

value_id	vcvalue
1	http://www.scottlan.edu/lriddle/women/BLUM.HTM
2	http://www.scottlan.edu/lriddle/women/

Figure 5-17: Views of **parse** and **valstring** for a parent directory and child file

```
DEFPROC bopeep(old_url_value_id, title)
    bopeep_init(old_url_value_id);
    bopeep_loop(0, title);
ENDPROC;

DEFPROC bopeep_loop(round, title)

    // Check candidate table for matches
    SELECT DISTINCT c.possible_url_id
    FROM candidate c, tag t, att a, valstring v
    WHERE c.round = round AND
    t.url_id = c.possible_url_id AND
    t.name = 'title' AND
    a.tag_id = t.tag_id AND
    a.name = 'anchor' AND
    v.value_id = a.value_id AND
    v.vcvalue = title

    // Put children of current candidates into candidate table
    INSERT INTO candidate (possible_url_id, round)
    SELECT DISTINCT l.dest_url_id, round+1
    FROM link l, candidate c
    WHERE l.source_url_id = c.possible_url_id
    AND c.round = round;

    bopeep_loop(round+1, title);

ENDPROC;
```

Figure 5-18: Simple Implementation of Bo Peep

(a) Show all elements of **candidate** of the most recent round that have the sought-for title in their title field.

(b) Add the children of the newest **candidate** pages to **candidate**, with the round field set to the current value of round plus one.

(c) Let round = round + 1

Figure 5-19 shows a run of the program.

5.3.2 Technique 2: Checking with pages that referenced the old URL

Figure 5-20 shows a link containing a URL U_{bad} that is no longer valid. People who pointed to URL u_{bad} in the past are some of the most likely people to point to u_{new} now, either because they were informed of the page movement or took the trouble to find the new location themselves. Here is a heuristic based on that observation:

1. Find a set of pages P that pointed to u_{bad} at some point in the past.
2. Let P' be the elements of P that no longer point to u_{bad} anymore.
3. See if any of the pages pointed to from elements of P' are the page we are seeking.

A question is how to recognize when we've found the target page. We do this by letting the user supply a key phrase and announcing which of the linked-to pages contains that phrase in its title

? url_id('http://www.scottlan.edu/lriddle/women/BLUM.HTM');
1

SELECT value_id FROM urls WHERE url_id = 1
5

bopeep(5, "Lenore Blum");
url_id
1185

possible_url_id	vcvalue

possible_url_id	vcvalue

possible_url_id	vcvalue
63	Lenore Blum

The user manually halts the program and then requests the string associated with the matching URL...

? url(63);
http://www.scottlan.edu:80/lriddle/women/blum.htm

Figure 5-19: Transcript of Bo Peep

```
<A HREF="http://bunny.cs.uiuc.edu/funding/academicCareers.html">
ACM SIGMOD's database academic careers information</A>
```

Figure 5-20: Broken Link Appearing on "http://physio1.utmem.edu/PHYSIOLOGY/opp.html"

```
DEFPROC bopeep2(old_url_id, anchor)
    CREATE TABLE candidate (possible_url_id url_id);

    // Insert url_ids of pages that once pointed to old_url_id
    INSERT INTO candidate (possible_url_id)
    SELECT source_url_id
    FROM rlink
    WHERE dest_url_id = old_url_id;

    // Remove url_ids of pages that still point to old_url_id
    DELETE FROM candidate
    WHERE possible_url_id IN
    (SELECT source_url_id
    FROM link
    WHERE dest_url_id = old_url_id);

    // See if any of the pages pointed to by candidates have
    // the old anchor text within their title or header
    SELECT DISTINCT l.dest_url_id, v.vcvalue
    FROM candidate c, tag t, att a, valstring v, link l
    WHERE l.source_url_id = c.possible_url_id AND
    t.url_id = l.dest_url_id AND
    (t.name = 'title' OR t.name = 'h1') AND
    a.tag_id = t.tag_id AND
    a.name = 'anchor' AND
    v.value_id = a.value_id AND
    v.vcvalue LIKE strcat("%", anchor, "%");
ENDPROC;
```

Figure 5-21: Implementation of Bo Peep2

or first header. A second moved-page finder, based on this strategy, is shown in Figure 5-21. A transcript appears in Figure 5-22.

? url_id('http://bunny.cs.uiuc.edu/funding/academicCareers.html');
1253
[printed]

bopeep2(1253, 'academic careers');

dest_url_id	name	vcvalue
1362	H1	ACM SIGMOD's collection of information on academic careers and academic life
1362	TITLE	ACM SIGMOD's database academic careers information

? url(1362);
http://bunny.cs.uiuc.edu/sigmod/funding/academicCareers.html
[printed]

Figure 5-22: Transcript for Bo Peep2

Chapter 6

Conclusions

6.1 Lessons Learned

In addition to producing several useful artifacts, most notably the Squeal interpreter, there were several conclusions reached that have broader applicability.

6.1.1 A Relational Database Model of the Web

User Interface

By providing an ontology (Chapter 2), I have shown that the relational database model is useful for representing information about the Web, at both the user and implementation levels. The power of the relatively short Squeal applications (Chapter 4) demonstrates that powerful queries on the Web can be stated concisely. We also know from the success of SQL that even more powerful statements can be constructed. A major limitation of SQL is that it cannot represent transitive closure, e.g., the question of whether a path exists from page A to page B. This limitation is well-suited to the Web, because such queries cannot necessarily be answered (as will be discussed in greater detail in section 6.2.2 below).

Of course, simplicity and expressiveness are often at odds. Some classes of queries can be expressed in a more concise and intuitive fashion in languages designed for the Web than in a more general-purpose language. For example, the query "What is in the third table on page X?" can be expressed much more naturally in WebL [19] than in SQL/Squeal.

Internal Representation

The Squeal system uses the relational database model not only in the user interface but also in its implementation (Chapter 5). This provides the user with the added benefit of being able to directly access the internal SQL tables by querying the SQL server instead of the Squeal interpreter. Although not discussed earlier in this document, it is a powerful option: Users could have the Squeal system build up the tables over the region of the Web they care about and then access them directly through SQL. This would allow the user to easily make queries such as finding pairs of pages [in the prefetched portions of the Web] point to each other.

Building the Squeal interpreter on top of a SQL server black box meant that a Squeal did not need to know how to interpret SQL statements, only how to produce them. This greatly expedited development and allowed the use of an industrial SQL server. On the other hand, there is a performance cost associated with having the Squeal-to-SQL translator (in the Squeal interpreter) separate from the SQL interpreter (in the SQL server), since the two associated optimizers are also separate.

6.1.2 Using SQL syntax to specify computation

In ordinary SQL systems, a SELECT query does not modify the database; it only returns information about the state of the database at the time the query is made. The Squeal system is novel in fetching and parsing Web pages in response to a query. This makes Squeal a far more powerful language than SQL.

Implementable

As the algorithms (Chapter 3) demonstrate, it is possible to implement this more powerful query-language semantics for certain domains, including the Web. Specifically, it can be implemented when each table has one or more defining columns (section 3.4) whose value can be used to determine entire lines in the table. In the Web domain, **url_id** was the most common defining column (Table 3.2, since it is easy to determine information about a specific page by fetching and parsing it or by using it as an input to a search engine. There is no reason this technique could not be used for other domains where one or two fields of a relation determine the values of the others.

Useful language for user

The Squeal extensions to the SQL semantics let users specify Web operations declaratively, i.e., indicating *what* information they want, not *how* it should be retrieved. Declarative programs are often easier for people to use and are more easily optimized by machines than imperative languages. While there is a learning curve associated with declarative languages, Squeal's similarity to a language as popular as SQL should improve its accessibility.

6.2 Comparisons to Related Work

6.2.1 Structure within and across web pages

Boyan et al. have observed that Web pages differ from general text in that they possess external and internal structure [5]. They use this information to propagate rewards from interesting pages to those that point to them (also done by LaMacchia [21]) and to more heavily weight words in titles, headers, etc., when considering document/keyword similarity, a technique used earlier in Lycos by Mauldin and Leavitt [22]. Mauldin has made use of link information by naming a page with the anchor text of hyperlinks to it. Iwazume et al. have preferentially expanded hyperlinks containing keywords relevant to the user's query [16], although O'Leary observes that anchor text information can be unreliable [28]. LaMacchia has implemented or proposed heuristics similar to some mentioned in this paper, such as making use of the information in directory hierarchies [21]. Frei and Stieger have discussed how knowledge of the adjacency of nodes via hyperlinks can be used to help a user navigate or find the answer to a query [11].

6.2.2 Theoretical analyses of the Web

Abiteboul and Vianu model the Web as an infinite semistructured set of objects and discuss what queries are theoretically or practically computable [2]. They conclude that first-order logic and Datalog¬ queries are problematic when they include negation, the use of which is restricted in Squeal. They also observe that some queries have possibly infinite answers, such as the set of pages referencing a given page P. For the same query, Squeal only returns the set of pages known about by a given search engine, after filtering out those that no longer link to P. The practicality is achieved at the expense of completeness.

Mendelzon and Milo consider the Web to be finite and discuss two differences between the Web and conventional databases: restricted access (e.g., one-way links) and a lack of concurrency control, which allows the Web to change during a query's computation. While they cache pages so they do not (appear to) change during queries, other concurrency problems remain. For example, while having a finite answer, a request for the set of pages reachable from a page P might never terminate,

if pages are added to the set faster than they can be read [24]. This query cannot be expressed in the Squeal core because, like SQL, it does not support recursion. It can be expressed as a recursive Squeal procedure, in which case the application would not terminate.

6.2.3 Database interfaces to the Web

An extractor developed within the TSIMMIS project uses user-specified wrappers to convert web pages into database objects, which can then be queried [14]. Specifically, hypertext pages are treated as text, from which site-specific information (such as a table of weather information) is extracted in the form of a database object. This is in contrast to ParaSite, where each page is converted into a set of database relations according to the same schema.

WebSQL allows queries about hyperlink paths among web pages, with limited access to the text and internal structure of web pages and URLs [4, 25, 23]. In the default configuration, hyperlinks are divided into three categories, internal links (within a page), local links (within a site), and global links. It is also possible to define new link types based on anchor text; for example, links with anchor text "next". All of these facilities can be implemented in ParaSite, although WebSQL's syntax is more concise. While it is possible to access a region of a document based on text delimiters, one cannot do so on the basis of structure. For example, consider this sample WebSQL definition and query:

```
DEFINE CONTEXT LEFT = HR, RIGHT = P;

 SELECT a.href
 FROM Anchor a SUCH THAT base =  "http://www.x.y.z/songs.html"
 WHERE file(a.href) CONTAINS ".wav"
 AND a.context CONTAINS "beatles";
```

For each link on page "http://www.x.y.z/songs.html" whose destination contains ".wav", the system will examine the region from the nearest "HR" tag to the left of the link to the nearest "P" tag to the right of the link, returning the destination if the region contains "beatles". This same query could be specified in ParaSite, although it would be more verbose. A query that can be stated in ParaSite but not WebSQL is: "Find all links appearing within the first top-level list on a page." This query relies on knowing not just that a link is within a list item (which can be determined by examining nearby text) but its position in the overall structure of a page; e.g., in the second list appearing within the first top-level list.

W3QL is another language for accessing the web as a database, treating web pages as the fundamental units [20]. Information one can obtain about web pages are:

1. The hyperlink structure connecting web pages
2. The title, contents, and links on a page
3. Whether they are indices ("forms") and how to access them

For example, it is possible to request that a specific value be entered into a form and to follow all links that are returned, giving the user the titles of the pages. It is not possible for the user to specify forms in ParaSite (or in WebSQL), access to a few search engines being hardcoded. Access to the internal structure of a page is more restricted than with ParaSite (or with WebSQL). In W3QL, one cannot even specify all hyperlinks originating within a list.

One major way in which ParaSite differs from both systems is in providing a data model guaranteeing that data is saved from one query to the next and (consequently) containing information about the time at which data was retrieved or interpreted. Another way that ParaSite is unique is in providing equal access to all tags and attributes, unlike WebSQL and W3QL, which can only refer to certain attributes of the LINK tag and provide no access to attributes of other tags.

6.3 Future Work

6.3.1 Improvements to the System

The current system is a prototype and could be improved in a number of ways.

Integration into a SQL server

Squeal would be much more efficient and robust if it were integrated with a SQL database server. Currently, no guarantees are made as to atomicity, consistency, isolation, and durability. Compilation of Squeal queries would be more efficient than the current interpretation, as would be exposing Squeal queries to optimization.

Memory usage

Opportunities exist for more efficient memory usage. For example, text within nested tags is repeated in the database, instead of being referred to indirectly. A better approach might be to use offsets, which would also eliminate the redundancy of storing both the unparsed contents of a page in **page** and the parsed contents in other **tag**, **att**, and especially **vcvalue**.

Access to up-to-date information

Active database technology [15] could be used to update or invalidate pages in the database as they expire. We would also like to provide Squeal with direct access to a search engine's database, which would also minimize data transfer delays.

6.3.2 Further evaluation

It would be desirable to evaluate many structure-based heuristics, such as those that appear in this document, in depth. Some specific experiments that would be desirable are:

1. Do a larger test of the similar page finder, comparing it to both Excite and collaborative filtering.
2. Run the home page finder on a large test set. Measure not only the performance of the entire application but the effectiveness of each heuristic.
3. Measure the performance of the moved-page finder and each of its heuristics.

A broader experiment would be to make Squeal publicly available and see what people do with it. The best verification of my thesis would be for Squeal and applications built on it to be widely used.

Appendix A

SQL Database Schema for Squeal

These are the definitions for the SQL database schema defining Squeal (Chapter 2).

```
CREATE TABLE valstring
    (value_id INT NOT NULL,
    vcvalue VARCHAR(255) DEFAULT NULL,
    textvalue TEXT DEFAULT NULL,
    CHECK ((vcvalue = NULL AND textvalue != NULL) OR
            (vcvalue != NULL AND textvalue = NULL)))

CREATE TABLE urls
    (url_id INT NOT NULL,
    value_id INT UNIQUE NOT NULL,
    variant INT DEFAULT 1,
    PRIMARY KEY (url_id, variant))

CREATE TABLE parse
    (url_value_id INT NOT NULL,
    component CHAR(5),
    value VARCHAR(255) NOT NULL,
    depth INT DEFAULT 0,
    CHECK (component IN ('host','port','path','ref')))

CREATE TABLE page
    (url_id INT NOT NULL,
    bytes INT DEFAULT NULL,
    contents TEXT DEFAULT NULL,
    md5 CHAR(32) DEFAULT NULL,
    valid CHAR,
    CHECK (valid IN ('y', 'n', 'b')))

CREATE TABLE tag
    (tag_id INT NOT NULL,
    name CHAR(15) NOT NULL,
    url_id INT NOT NULL,
    startOffset INT NOT NULL,
    endOffset INT NOT NULL,
    PRIMARY KEY (tag_id))
```

```
CREATE TABLE att
    (tag_id INT NOT NULL,
    name CHAR(15) NOT NULL,
    value_id INT NOT NULL)

CREATE TABLE header
    (url_id INT NOT NULL,
    struct BINARY(6) NOT NULL,
    offset INT NOT NULL,
    ord INT NOT NULL)

CREATE TABLE list
    (url_id INT NOT NULL,
    struct BINARY(6) NOT NULL,
    offset INT NOT NULL,
    ord INT NOT NULL,
    tstamp TIMESTAMP)

CREATE TABLE link
    (source_url_id INT,
    anchor_value_id INT,
    dest_url_id INT,
    hstruct BINARY(6),
    lstruct BINARY(6),
    compute_id INT,
    tstamp TIMESTAMP)

CREATE TABLE rcontains
    (url_id INT NOT NULL,
    value VARCHAR(255) NOT NULL,
    helper CHAR(16) NOT NULL)

CREATE TABLE rlink
    (source_url_id INT,
    anchor VARCHAR(255) DEFAULT NULL,
    dest_url_id INT DEFAULT NULL,
    helper CHAR(16),
    CHECK ((dest_url_id = NULL AND anchor != NULL) OR
            (anchor = NULL AND dest_url_id != NULL)))
```

Appendix B

The Squeal Grammar

This is the grammar for Squeal in the format expected by the Java Compiler Compiler (JavaCC) [36].

```
options {
  IGNORE_CASE = true;
  MULTI = true;
}

PARSER_BEGIN(squealParser)
package squeal;
import SymbolTable;

public class squealParser {

  public static void main(String args[]) throws ParseError {
    squealParser parser = new squealParser(System.in);
    ASTstatement ast = parser.statement();
    ast.dump("");
  }

  public void reinit() { jjtree.reset(); }
}

PARSER_END(squealParser)

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
| < "//" (~["\n"])* "\n">
}

TOKEN :
{
        < SEMICOLON: ";">
|       < COLON: ":" >
|       < COUNT: "COUNT">
|       < SELECT: "SELECT">
|       < MSELECT: "MSELECT">
```

```
|       < UPDATE: "UPDATE">
|       < FROM: "FROM">
|       < ALL: "ALL">
|       < DISTINCT: "DISTINCT">
|       < UNIQUE: "UNIQUE">
|       < COMMA: "," >
|       < STAR: "*">
|       < PERIOD: "." >
|       < WHERE: "WHERE">
|       < GROUPBY: "GROUP BY">
|       < HAVING: "HAVING">
|       < LPAREN: "(" >
|       < RPAREN: ")" >
|       < NOT: "NOT">
|       < AND: "AND" | "&">
|       < OR: "OR" | "|">
|       < NEQ: "<>">
|       < LESS: "<">
|       < EQUALS: "=">
|       < GREATER: ">">
|       < GTE: ">=">
|       < LTE: "<=">
|       < LIKE: "LIKE">
|       < MATCHES: "MATCHES">
|       < BETWEEN: "BETWEEN">
|       < ORDERBY: "ORDER BY">
|       < SQUOTE: "'" >
|       < DQUOTE: "\"" >
|       < SSTRING: <SQUOTE> (~["'"])* <SQUOTE>>
|       < DSTRING: <DQUOTE> (~["\""])* <DQUOTE>>
|       < SLASH: "/">
|       < PLUS: "+">
|       < MINUS: "-">
|       < ASC: "ASC">
|       < DESC: "DESC">
|       < AVG: "AVG">
|       < MAX: "MAX">
|       < MIN: "MIN">
|       < SUM: "SUM">
|       < AS: "AS">
|       < DELETE: "DELETE">
|       < DROP: "DROP">
|       < QMARK: "?" >
|       < LET: "let" >
|       < DEFFUNC: "DEFFUNC">
|       < DEFPROC: "DEFPROC">
|       < ENDPROC: "ENDPROC">
|       < FETCH: "FETCH">
|       < PRINT: "PRINT">
|       < INTO: "INTO">
|       < HELP: "HELP">
|       < SET: "SET">
|       < IN: "IN">
|       < QUIT: "QUIT">
```

```
|       < EXIT: "EXIT">
|       < INSERT: "INSERT">
|       < VALUES: "VALUES">
|       < DESCRIBE: "DESCRIBE">
|       < UNLESS: "UNLESS">
|       < POUND: "#">
|       < NEW: "NEW">
|       < INPUT: "INPUT">
|       < OUTPUT: "OUTPUT">
|       < ELSE: "ELSE">
|       < CREATE: "CREATE">
|       < TABLE: "TABLE">
|       < CHAR: "CHAR">
|       < VARCHAR: "VARCHAR">
|       < BINARY: "BINARY">
|       < VARBINARY: "VARBINARY">
|       < INT: "INT">
|       < CONVERT: "CONVERT">
|       < VALUE_ID: "value_id">
|       < URL_ID: "url_id">
|       < ID: (<POUND>)*(<LETTER>)+(<LETTER>|<DIGIT>|<UNDERSCORE>)*>
|       < LETTER: ["A"-"Z", "a"-"z"]>
|       < UNDERSCORE: "_" >
|       < NUMBER: (<DIGIT>)+>
|       < DIGIT: ["0"-"9"]>
}


//// Statements

ASTstatement statement():{Token into = null;}
{
        <EOF> {return null;}
 |       nontrivialStatement() statement_separator() {return jjtThis;}
}


void nontrivialStatement()#void:{} {
  printStatement()
 |       letStatement()
 |       deffuncStatement()
 |       computeStatement()
 |       callStatement()
 |       defprocStatement()
 |  helpStatement()
 |       quitStatement()
 |       inputStatement()
 |       outputStatement()

        // SQL
 |       selectStatement(false)
 |       createStatement()
 |       dropStatement()
 |       deleteStatement()
```

```
 |       updateStatement()
 |       insertStatement()
 |       describeStatement()
}

void quitStatement():{}
{
        <QUIT> | <EXIT>
}

void helpStatement():{String s;}
{
        LOOKAHEAD(<HELP><LPAREN>)
        <HELP> <LPAREN> (s=identifier()) <RPAREN> {jjtThis.setName(s);}
|       <HELP> (s=identifier()) {jjtThis.setName(s);}
}

void statement_separator()#void:{}
{
        <SEMICOLON>
}

void selectStatement(boolean b):{Token cmd; String s=null; Token t = null;}
{
        (cmd=<SELECT>|cmd=<MSELECT>) (s=sel_restrict())? selectList()
        <FROM> tableList() restrict()
        ((t=<AS>|t=<INTO>) stringLiteral())?
         {jjtThis.setCommand(cmd.image);
          jjtThis.setInitialRestriction(s);
          if (t != null) jjtThis.setFileName();
          jjtThis.setIsSubSelect(b);}
}

String sel_restrict()#void:{}
{
        <ALL> {return "ALL";}
|       <DISTINCT> {return "DISTINCT";}
|       <UNIQUE> {return "UNIQUE";}
}

void subSelectStatement()#void:{}
{
        selectStatement(true)
}

void insertStatement():{Token id;}
{
        <INSERT> (<INTO>)? (id=<ID>) symbolList() insertStatementRHS()
         {jjtThis.setName(id.image);}
}

void insertStatementRHS()#void:{}
{
        <VALUES> argList()
```

```
|       subSelectStatement()
}

void updateStatement():{}
{
        <UPDATE> tableName() <SET> set_list() (whereDef())?
}

void describeStatement():{Token t;}
{
        <DESCRIBE> (t=<ID>) {jjtThis.setName(t.image);}
}

void deleteStatement():{Token t;}
{
        <DELETE> <FROM> (t=<ID>) (<WHERE> condition())?
         {jjtThis.setName(t.image);}
}

void dropStatement():{Token name;}
{
        <DROP> <TABLE> (name = <ID>)
         {jjtThis.setName(name.image);}
}

void createStatement():{Token name;}
{
        <CREATE> <TABLE> (name = <ID>) <LPAREN> columnDefList() <RPAREN>
         {jjtThis.setName(name.image);}
}

void columnDefList()#void:{}
{
        columnDef() (<COMMA> columnDef())*
}

void columnDef():{String name; String type;}
{
        (name = identifier()) (type=columnTypeDef())
         {jjtThis.setName(name); jjtThis.setType(type);}
}

String columnTypeDef()#void:{Token base, count;}
{
        (base=<INT>)
         {return base.image;}
|       (base=<URL_ID>)
         {return base.image;}
|       (base=<VALUE_ID>)
         {return base.image;}
|       (base=<CHAR>) <LPAREN> (count=<NUMBER>) <RPAREN>
         {return base.image + "(" + count.image + ")";}
|       (base=<VARCHAR>) <LPAREN> (count=<NUMBER>) <RPAREN>
         {return base.image + "(" + count.image + ")";}
```

```
|       (base=<VARBINARY>) <LPAREN> (count=<NUMBER>) <RPAREN>
        {return base.image + "(" + count.image + ")";}
|       (base=<BINARY>) <LPAREN> (count=<NUMBER>) <RPAREN>
        {return base.image + "(" + count.image + ")";}
}


//// Support for SELECT and FETCH statements

void selectList():{}
{
        selectItem() (<COMMA> selectItem())*
}

void selectItem():{String s=null;}
{
        expression() (<AS> (s=identifier()))?
        { if (s != null) jjtThis.setAlias(s); }
}

// Cells are allowed to handle transformed select statements
void namedArgument():{Token t=null;}
{
        (<NEW>)? symbolLiteral() (<PERIOD> symbolLiteral())?
        ((t=<EQUALS>)|(t=<LIKE>)) expression()
        { jjtThis.setOperator(t.image);}
}

void namedArgumentList():{}
{
        namedArgument() (<COMMA> namedArgument())*
}

void set_list()#void:{}
{
        namedArgumentList()
}

void computeList()#void:{}
{
        (namedArgumentList())?
}

void tableList():{}
{
        tableName() (<COMMA> tableName())*
}

void tableName():{String s; Token t2=null;}
{
        (s=identifier()) (t2=<ID>)? {jjtThis.setName(s);
          if (t2 != null) jjtThis.setAlias(t2.image);}
}
```

```
void restrict()#void:{}
{
        (whereDef())? (groupbyDef())? restrictEnd()
}

void whereDef():{}
{
        <WHERE> condition()
}


void restrictEnd()#void:{} {
        LOOKAHEAD(<HAVING>) havingDef() (orderbyDef())?
}

void groupbyDef():{}
{
        <GROUPBY> columnsList()
}

void havingDef():{}
{
        <HAVING> condition()
}

void orderbyDef():{}
{
        <ORDERBY> orderList()
}

void orderList():{}
{
        orderItem() (<COMMA> orderItem())*
}

void orderItem():{String s=null;}
{
        expression() (s = order_list_modifier())?
         { jjtThis.setModifier(s);}
}

String order_list_modifier()#void:{}
{
        <ASC> {return "ASC";}
|       <DESC> {return "DESC";}
}


void condition()#void:{}
{
        logicExpression()
}

void searchExpression()#void:{}
```

```
{
        logicExpression()
}

void columnsList():{}
{
        column() (<COMMA> column())*
}

void column():{Token t1; String s;}
{
        LOOKAHEAD(<ID> <PERIOD>) (t1=<ID>) <PERIOD> (s=identifier())
         { jjtThis.setTableName(t1.image);
           jjtThis.setColumnName(s);}
|       (s=identifier())
         { jjtThis.setColumnName(s);}
}

void convertExpression():{String def;}
{
        <CONVERT> <LPAREN> (def=columnTypeDef()) <COMMA> expression() <RPAREN>
         {jjtThis.setType(def);}
}


void aggregateExpression():{String s, r=null;}
{
        (s=aggregate()) <LPAREN> (r=agg_restrict())? expression() <RPAREN>
         {jjtThis.setName(s); jjtThis.setRestriction(r);}
}

void star():{}
{
        <STAR>
}

String aggregate()#void:{}
{
        <AVG> {return "AVG";}
|       <MAX> {return "MAX";}
|       <MIN> {return "MIN";}
|       <SUM> {return "SUM";}
|       <COUNT> {return "COUNT";}
}

String count()#void:{}
{
        <COUNT> {return "COUNT";}
}

String agg_restrict()#void:{}
{
        <ALL> {return "ALL";}
|       <DISTINCT> {return "DISTINCT";}
```

```
|       <UNIQUE> {return "UNIQUE";}
}

void inputStatement() :{}
{
        <INPUT> expression()
}

void outputStatement() :{}
{
        <OUTPUT> stringLiteral()
}

void printStatement() :{}
{
        (<QMARK>|<PRINT>) expression()
}

void letStatement():{Token t;}
{
        <LET> (t=<ID>) <EQUALS> expression() (<ELSE> expression())?
        {jjtThis.setName(t.image);}
}

void deffuncStatement():{Token t;}
{
        <DEFFUNC> (t=<ID>) symbolList() expression()
         {jjtThis.setName(t.image);}
}

void symbolList():{}
{
        <LPAREN> (symbolLiteral() (<COMMA> symbolLiteral())*)? <RPAREN>
}

void symbolLiteral():{String s;}
{
        (s=identifier()) { jjtThis.setName(s);}
}

void defprocStatement():{Token t;}
{
        <DEFPROC> (t=<ID>) symbolList()
        statement() (statement())*
        <ENDPROC>
         {jjtThis.setName(t.image);}
}

void callStatement()#void:{}
{
        funcall()
}

void computeStatement():{Token t; Token intoName = null; Token unless = null;}
```

```
{
        <FETCH> (t=<ID>) <LPAREN> computeList() <RPAREN>
        (<FROM> tableList() restrict())?
        (<INTO> (intoName = <ID>))?
        ((unless=<UNLESS>) condition())?
         {jjtThis.setName(t.image);
         if (unless != null)
           jjtThis.setUnlessClause();
         if (intoName != null)
           jjtThis.setIntoName(intoName.image);
         else
           jjtThis.setIntoName(t.image);}
}

void expression()#void :{}
{
   computedExpression()
}

void computedExpression():{Token t = null;}
{
        logicExpression()
}

void logicExpression()#void :{}
{
        disjunctionExpression()
}

void disjunctionExpression():{Token op = null;}
{
        conjunctionExpression() ((op=<OR>) disjunctionExpression())?
         {if (op != null) jjtThis.setOp(op.image);}
}

void conjunctionExpression():{Token op = null;}
{
        negationExpression() ((op=<AND>) conjunctionExpression())?
         {if (op != null) jjtThis.setOp(op.image);}
}

void negationExpression():{Token op = null;}
{
        ((op=<NOT>))? relExpression()
         {if (op != null) jjtThis.setOp(op.image);}
}

void relExpression():{String op = null;}
{
        sumExpression() ((op = rel_op()) sumExpression())?
         {jjtThis.setOp(op);}
}

String identifier()#void:{Token t;}
```

```
{
        ((t=<ID>)|(t=<URL_ID>)|(t=<VALUE_ID>)) {return t.image;}
}


String rel_op()#void:{}
{
        <LESS> {return("<");}
|       <EQUALS> {return("=");}
|       <GREATER> {return(">");}
|       <NEQ> {return("<>");}
|       <GTE> {return(">=");}
|       <LTE> {return("<=");}

|       LOOKAHEAD(<NOT> <IN>) <NOT> <IN> {return "NOT IN";}
|       <LIKE> {return "LIKE";}
|       <NOT> <LIKE> {return "NOT LIKE";}
|       <IN> {return "IN";}
}

void sumExpression():{String op = null;}
{
        productExpression() ((op=sum_op()) sumExpression())?
         {jjtThis.setOp(op);}
}

String sum_op()#void:{}
{
        <PLUS> {return("+");}
|       <MINUS> {return("-");}
}

void productExpression():{String op=null;}
{
        unaryExpression() ((op=product_op()) productExpression())?
         {jjtThis.setOp(op);}
}

String product_op()#void:{}
{
        <STAR> {return "*";}
|       <SLASH> {return "/";}
}

void unaryExpression():{}
{
        parenthesizedExpression()
|       <MINUS> parenthesizedExpression() {jjtThis.setNegated();}
}

void parenthesizedExpression():{}
{
        LOOKAHEAD(<LPAREN><SELECT>) <LPAREN> subSelectStatement() <RPAREN>
|       LOOKAHEAD(<LPAREN><MSELECT>) <LPAREN> subSelectStatement() <RPAREN>
```

```
|       <LPAREN> computedExpression() <RPAREN> {jjtThis.setParenthesized();}
|   LOOKAHEAD(identifier() <LPAREN>) funcall()
|       LOOKAHEAD(<ID> <PERIOD>) cell()
|       literal()
|       variable()
|       aggregateExpression()
|       star()
|       convertExpression()
}

void variable():{String s;}
{
        (s=identifier()) {jjtThis.setName(s);}
}

void literal()#void:{Token t;}
{
        numericLiteral()
  |     stringLiteral()
}

void numericLiteral():{Token t;}
{
        t=<NUMBER> {jjtThis.setNumber(t.image);}
}

void stringLiteral():{Token t;}
{
  t=<SSTRING> {jjtThis.setName(t.image.substring(1,
             t.image.length()-1));}
  |     t=<DSTRING> {jjtThis.setName(t.image.substring(1,
             t.image.length()-1));}
}

void funcall() :{String s;}
{
   (s = identifier()) argList()  {jjtThis.setName(s);}
}

void argList() : {}
{
        <LPAREN> (expression() (<COMMA> expression())*)? <RPAREN>
}

void cell():{ Token r, c;}
{
        (r=<ID>) <PERIOD> ((c=<ID>)|(c=<VALUE_ID>)|(c=<URL_ID>)|(c=<STAR>))
{jjtThis.setRelAndColName(r.image,c.image);}
}
```

Appendix C

Source Code for Home Page Finder

The home page finder is discussed in Sections 1.1.3 and 5.2.

```
DEFPROC HomePage(fullname)
    CREATE TABLE possibleParent(url_id url_id, urlstring varchar(255), reason CHAR(255));
    CREATE TABLE candidate(url_id url_id, score int, reason CHAR(255));
    CREATE TABLE results(url_id url_id, score int);

    HomePageCore(fullname);
    // Fill in the results table
    INSERT INTO results (url_id, score)
    SELECT c.url_id, SUM(c.score) AS tot
    FROM candidate c
    GROUP BY c.url_id;

    // Display the results
    SELECT u.url_id, v.vcvalue, r.score
    FROM valstring v, results r, urls u
    WHERE u.url_id = r.url_id
    AND u.variant = 1
    AND v.value_id = u.value_id
    ORDER BY r.score DESC;
ENDPROC;

DEFPROC HomePageCore(fullname)
    LET fullnameExp = strcat('%', fullname, '%');

    // Get possible parents that reportedly contain name as anchor text
    INSERT INTO possibleParent (url_id, urlstring, reason)
    SELECT DISTINCT r.source_url_id, v.vcvalue, 'Anchor reportedly contains full name:
        "' + fullname + '"'
    FROM rlink r, urls u, valstring v
    WHERE r.anchor like fullnameExp
    AND u.url_id = r.source_url_id
    AND u.variant = 1
    AND v.value_id = u.value_id;

    // Find pages with *just* the full name in anchor text
    INSERT INTO candidate (url_id, score, reason)
    SELECT DISTINCT l.dest_url_id, 2, 'Anchor from "' + p.urlstring + '" is
        the full name: "' + v.vcvalue + '"'
```

```
FROM link l, possibleParent p, valstring v
WHERE l.source_url_id = p.url_id
AND l.anchor_value_id = value_id(fullname)
AND v.value_id = l.anchor_value_id;

// Find pages with the full name anywhere in the anchor text
INSERT INTO candidate (url_id, score, reason)
SELECT DISTINCT l.dest_url_id, 1, 'Anchor from "' + p.urlstring + '" includes the full
    name: "' + v.vcvalue + '"'
FROM link l, possibleParent p, valstring v
WHERE l.source_url_id = p.url_id
AND v.value_id = l.anchor_value_id
AND v.vcvalue LIKE fullnameExp;

// Increment pages with name in attribute
INSERT INTO candidate(url_id, score, reason)
SELECT t.url_id, count(*), 'The name ("' + fullname + '") appears in ' +
    CONVERT(VARCHAR(5),COUNT(*)) + ' attribute value(s) on the page'
FROM tag t, att a, valstring v
WHERE t.url_id in (select distinct url_id from candidate)
AND a.tag_id = t.tag_id
AND v.value_id = a.value_id
AND v.vcvalue LIKE fullnameExp
GROUP BY t.url_id;

// Further increment pages with name within title or header
INSERT INTO candidate(url_id, score, reason)
SELECT t.url_id, 5, 'The anchor text of a "' + t.name+ '" tag contains the full name('
    fullname + '")'

FROM tag t, att a, valstring v
WHERE t.url_id IN (SELECT DISTINCT url_id FROM candidate)
AND (t.name='title' OR t.name LIKE 'h_')
AND a.tag_id = t.tag_id
AND a.name = 'anchor'
AND v.value_id = a.value_id
AND v.vcvalue like fullnameExp;

// Find pages with URLs of the form <foo>/foo.html
INSERT INTO candidate (url_id, score, reason)
SELECT DISTINCT u.url_id, 20, 'Base of file name same as name of directory: ' + v.vcvalue
FROM urls u, parse p1, parse p2, valstring v
WHERE u.url_id IN (SELECT DISTINCT url_id FROM candidate)
AND p1.url_value_id = u.value_id
AND p2.url_value_id = u.value_id
AND v.value_id = u.value_id
AND p1.depth+1 = p2.depth
AND (p1.value = p2.value + '.html' OR p1.value = p2.value + '.htm');

// Find pages named index.html or home.html or ¡empty¿
INSERT INTO candidate (url_id, score, reason)
SELECT DISTINCT c.url_id, 10, 'File name is "' + p.value + '"'
FROM candidate c, urls u, parse p
WHERE u.url_id = c.url_id
```

```
    AND p.url_value_id = u.value_id
    AND p.depth = 1
    AND (p.value LIKE 'home.htm%' OR p.value LIKE 'index.htm%' OR
    p.value LIKE '');

    // Find pages where final directory starts with ~
    INSERT INTO candidate (url_id, score, reason)
    SELECT DISTINCT c.url_id, 10, 'Final directory name starts with tilde: "' + p.value + '"'
    FROM candidate c, urls u, parse p
    WHERE u.url_id = c.url_id
    AND p.url_value_id = u.value_id
    AND p.depth = 2
    AND p.value LIKE '~%';

    // Find pages where the penultimate directory is named
    // "home%" or "people"
    INSERT INTO candidate (url_id, score, reason)
    SELECT DISTINCT c.url_id, 10, 'Penultimate directory is: "' + p.value + '"'
    FROM candidate c, urls u, parse p
    WHERE u.url_id = c.url_id
    AND p.url_value_id = u.value_id
    AND p.depth = 3
    AND (p.value LIKE 'home%' OR p.value LIKE 'people');
ENDPROC;
```

Appendix D

User Evaluations of Recommender Systems

Section 5.1 described the recommender system experiment. This appendix contains the user ratings that were not included there. The summary of the scores is repeated in table D.1, augmented with page numbers for the detailed ratings.

D.1 American Airlines

The URLs returned by each system for the "American Airlines" home page are shown in Table D.2. All 5 sites returned by Excite were within the American web site, leading to low ratings for novelty. Of the 5 sites returned by ParaSite, only one was at the American web site, three were to other travel-related pages, and one was to an entirely irrelevant page. The Excite pages were rated as being slightly more relevant, while the ParaSite pages were judged more interesting and slightly more novel. The user ratings are shown in Table D.3.

D.2 Geodesic Systems

The URLs returned by each system for the "Geodesic Systems" home page are shown in Table D.4. Geodesic Systems is a company that makes a C++ garbage collector. The results were similar to those for American Airlines: All of the Excite recommendations were within the same site as the seed URL, leading to low ratings for novelty. Of the 4 sites returned by ParaSite, only one was within the seed site, and three were to other software development pages. A representative comment was:

> If you're specifically interested in Geodesic then I would prefer system A [Excite]. If you're interested in tools which I think more people are then system B [ParaSite] would be better. —W

Correspondingly, the ParaSite pages were rated as being slightly more relevant and interesting and much more novel, as shown in Table D.5.

D.3 Rogue Market

The URLs returned by each system for the "Rogue Market" home page are shown in Table D.6. The Rogue Market is an exchange for virtual shares of celebrities and other popular culture icons. Excite's top two suggestions were pages that mentioned the Rogue Market, and the last three pages that contained one or both of the words "rogue" and "market". ParaSite's first suggestion was rated entirely irrelevant, the second was a fan club, and the last three were online gaming sites. As the person who provided the seed URL wrote:

Ebert	Excite	0.95	1.30	0.55	0.92
(p. 79)	ParaSite	1.55	1.35	1.00	3.00
	Δ	63%	4%	82%	225%
Austin	Excite	0.95	0.95	0.25	0.23
(p. 80)	ParaSite	1.40	1.85	1.40	5.84
	Δ	47%	95%	460%	2459%
Mapquest	Excite	2.20	1.85	1.35	6.49
(p. 83)	ParaSite	0.90	1.40	0.78	1.59
	Δ	-59%	-24%	-43%	-76%
American	Excite	1.95	1.60	0.95	3.09
(p. 80)	ParaSite	0.99	1.11	1.45	7.00
	Δ	-49%	-31%	53%	127%
Geodesic	Excite	2.25	1.94	0.44	1.90
(p. 121)	ParaSite	2.31	2.13	1.75	9.13
	Δ	3%	10%	300%	381%
AMD	Excite	1.30	1.35	1.35	4.31
(p. 81)	ParaSite	1.81	1.69	0.88	3.34
	Δ	39%	25%	-35%	-23%
Rogue	Excite	1.30	1.35	1.35	4.31
(p. 122)	ParaSite	1.13	1.37	1.50	4.44
	Δ	-13%	1%	11%	3%
Art Bell	Excite	2.25	2.00	0.50	2.25
(p. 122)	ParaSite	0.6	1.00	0.90	0.85
	Δ	-73%	-50%	80%	-62%
Activision	Excite	1.80	1.45	2.15	5.73
(p. 123)	ParaSite	1.73	2.07	1.65	6.79
	Δ	-4%	43%	-23%	19%
Happy Puppy	Excite	2.63	2.19	1.13	7.15
(p. 124)	ParaSite	1.09	1.14	1.10	3.85
	Δ	-58%	-48%	-2%	-46%
Economist	Excite	1.90	1.70	1.80	6.32
(p. 125)	ParaSite	1.00	1.00	0.75	3.41
	Δ	-47%	-41%	-58%	-46%
Geek	Excite	1.94	1.44	0.94	3.12
(p. 82)	ParaSite	1.71	1.83	2.13	7.12
	Δ	-12%	28%	127%	128%
KnotPlot	Excite	2.45	2.05	1.85	9.95
(p. 84)	ParaSite	1.30	1.05	1.87	3.13
	Δ	-47%	-49%	1%	-69%
Average	Excite	1.84	1.63	1.12	4.29
	ParaSite	1.35	1.46	1.32	4.58
	Δ	-27%	-10%	17%	7%

Table D.1: Averages of Ratings by Seed URL. Ratings were from 0 to 3, with higher numbers better. Seed URLs are listed in Table 5.2 on page 76.

Excite	ParaSite
aa_home/aadvantage/guide/earn.htm *Earning AAdvantage Miles*	www.microsoft.com/ *Microsoft Corporation Home Page*
aa_home/somespec.htm *Something Specials*	www.ual.com/ *United Airlines Official Homepage*
aa_home/aadvantage/aadvantage.htm *AAdvantage Home Page*	itre.ncsu.edu/index.html *The ITRE Home Page*
aa_home/aadvantage/guide/platinum.htm *AAdvantage Member Guide - Platinum Members*	www.americanair.com/ *American Airlines Home Page*
aa_home/aavac/aav_info/info2.html *Fly AAway Vacations Hotel Ratings*	www.easysabre.com/ *easySABRE*

Table D.2: Top 5 Pages Returned by Excite and ParaSite for American Airlines URL (http://www.americanair.com:80/aa_home.htm). The host name, "www.americanair.com" is omitted from the Excite URLs above.

P r i n	K r i n	W r i n	S r i n	Title	Average r	 i	 n	prod- uct
3 3 1	1 1 1	3 3 1	2 0 0	Earning Aadvantage Miles	2.25	1.75	0.75	2.95
3 3 2	1 1 1	3 3 2	2 0 0	Something Specials	2.25	1.75	1.25	4.92
3 3 1	1 1 1	3 3 1	1 0 0	Aadvantage Home Page	2	1.75	0.75	2.63
2 1 1	1 1 1	2 2 1	1 0 0	Aadvantage Member Guide...	1.5	1	0.75	1.13
2 3 2	1 1 1	3 3 2	1 0 0	FlyAAway Vacations...	1.75	1.75	1.25	3.83
				Excite averages	1.95	1.60	0.95	3.09
0 1 1	0 0 0	0 1 1	0 2 1	Microsoft Corporation Home Page	0	1	0	0.00
2 2 1	3 3 1	3 3 1	3 1 2	United Airlines Official Homepage	2.75	2.25	1.25	7.73
1 0 1	0 0 3	2 2 3	0 1 0	The ITRE Home Page	0.75	0.75	1	0.56
3 3 0	3 3 0	3 3 1	1 0 0	American Airlines Home Page	2.5	2.25	0.25	1.41
3 3 2	2 2 3	3 3 1	3 3 3	easySABRE	2.75	2.75	2.25	17.02
				ParaSite averages	1.75	1.80	0.95	5.34

Table D.3: User Ratings of American Airlines Recommendations

Excite	ParaSite
www.geodesic.com/news/news_hp-ux.html *Geodesic: Great Circle for HP-UX press release*	www.geodesic.com/GreatCircle/index.html *Geodesic - Great Circle*
www.microway.com.au/catalog/geodesic /welcome.html *Geodesic Systems Welcome Page*	www.parasoft.com/ *ParaSoft Corporation*
www.geodesic.com/comments/comments.html *Geodesic Systems- Comments*	www.pureatria.com/ *Pure Atria*
www.geodesic.com/GreatCircle/index.html *Geodesic - Great Circle*	www.roguewave.com/ *Rogue Wave Home Page*

Table D.4: Top 4 Pages Returned by Excite and ParaSite for Geodesic Systems URL (http://www.geodesic.com). Because ParaSite could only generate 4 suggestions, only the top 4 suggestions from Excite were used.

P r i n	K r i n	W r i n	S r i n	Title	Average r	i	n	prod-uct
3 3 0	2 2 0	3 2 2	1 0 0	GS: Great Circle for HP-UX...	2.25	1.75	0.5	1.97
3 3 0	2 2 0	3 3 2	1 0 0	GS Welcome Page	2.25	2	0.5	2.25
3 3 0	2 2 0	3 3 1	1 0 0	GS - Comments	2.25	2	0.25	1.13
3 3 0	2 2 0	3 3 2	1 0 0	Geodesic - Great Circle	2.25	2	0.5	2.25
					2.25	1.94	0.44	1.90
3 3 0	2 2 0	3 3 2	1 0 0	Geodesic - Great Circle	2.25	2	0.5	2.25
3 3 3	3 2 3	3 3 2	3 2 1	ParaSoft Corporation	3	2.5	2.25	16.88
3 3 2	2 2 3	1 1 1	2 1 1	Pure Atria	2	1.75	1.75	6.13
2 2 3	2 2 3	3 3 3	1 2 1	Rogue Wave Home Page	2	2.25	2.5	11.25
				ParaSite averages	2.31	2.13	1.75	9.13

Table D.5: User Ratings of Geodesic Systems Recommendations. For display purposes, the company name has been abbreviated to "GS".

Excite	ParaSite
www.stonesworld.com/2.0/corner/bbs/messages/12179.html *Stones Stock at 91.51 in the Rogue Market*	earth.simmons.edu/ *EnviroNet at Simmons College*
newsradio88.com/boot/archive/july_1997/july_17.html *WCBS NEWSRADIO 88 Boot Camp*	op.net/~jester *The Proudest Monkeys: The Dave Matthews Band Fan Club*
www.fool.com/FoolFAQ/FoolFAQ0007.htm *The Motley Fool: Fool FAQ - Bid/Ask Price*	smc.sierra.com/ *Sierra Internet Gaming System*
rogue.northwest.com/ mushro19/ *Mushrooms, Truffles & Matsutakes of the Northwest*	www.eonline.com/ *Welcome to E! Online - Front Door*
swerve.basilisk.com/basilisk_381.html *basilisk*	www.mindspring.com/~ogl/pcbt.htm *Puzzle, Card, Etc, Online Games*

Table D.6: Top 4 Pages Returned by Excite and ParaSite for Rogue Market Seed URL (http://www.roguemarket.com)

> System A [Excite] found me sites that mentioned the Rogue Market. System B [ParaSite] appeared to match what I was doing, having fun on the internet (Sierra, E!, Puzzle), gossip about stars, etc. —K

As shown in Table D.7, the two systems received similar numeric ratings, with the Excite pages judged more relevant and the ParaSite pages more novel.

D.4 Art Bell

The URLs returned by each system for the "Art Bell" web site are shown in Table D.8. Art Bell is a talk radio host who covers fringe topics. All 5 of Excite's recommendations were pages within the Art Bell site. None of the ParaSite suggestions were on the same site, and only one was relevant. Excite scored higher on relevance and interest, while ParaSite scored higher on novelty D.9.

P r i n	**K** r i n	W r i n	S r i n	Title	Average r	i	n	prod-uct
3 2 1	2 0 3	3 3 2	2 1 0	Stones Stock	2.5	1.5	1.5	5.63
3 3 3	0 0 3	3 2 3	2 2 1	WCBS NEWSRADIO 88	2	1.75	1.75	6.13
1 2 3	1 0 3	3 3 2	2 3 3	The Motley Fool	1.75	2	2.75	9.63
0 1 3	0 0 3	0 0 2	0 1 0	Mushrooms, Truffles	0	0.5	0	0.00
1 2 3	0 0 3	0 0 2	0 2 2	basilisk	0.25	1	0.75	0.19
				Excite averages	1.30	1.35	1.35	4.31
0 1 0	0 0 3	0 1 2	0 1 1	EnviroNet	0	0.75	0	0.00
0 0 0	1 0 3	0 0 2	1 1 1	The Proudest Monkeys	0.5	0.25	1	0.13
3 3 3	0 0 1	2 2 2	2 1 2	Sierra Internet Gaming	1.75	1.5	1.75	4.59
2 3 3	2 2 2	1 2 3	- - -	Welcome to E! Online	1.67	2.33	2.25	8.75
3 3 3	1 2 3	2 2 2	1 1 2	Puzzle, Card, Etc.	1.75	2	2.5	8.75
				ParaSite averages	1.13	1.37	1.50	4.44

Table D.7: User Ratings of Rogue Market Recommendations

Excite	ParaSite
www.artbell.com/satellite.html Art Bell - Satellite Reception Info	www.yahoo.com/ Yahoo!
www.artbell.com/canvas.html Art Bell - The Artist's Canvas	www.afrosquad.com/ AfroFrontDoor
www.artbell.com/stations.html Art Bell - Radio Station Affiliates	www.bigbook.com/ BigBook
www.artbell.com/ufos2.html Art Bell - UFO Photos Page 2	www.execpc.com/~krstic/mylinks.html Rick Krsticevic's Links
www.artbell.com/hotlinks.html Art Bell - Related Web Links	www.four11.com/ Four11 Directory Services

Table D.8: Top 5 Pages Returned by Excite and ParaSite for Art Bell Seed URL (http://www.artbell.com).

P r i n	**K** r i n	W r i n	S r i n	Title	Average r	i	n	prod-uct
3 3 0	2 2 0	3 3 1	1 0 0	Art Bell - Satellite Reception Info	2.25	2	0.25	1.13
3 3 0	2 2 0	3 3 2	1 0 0	Art Bell - The Artist's Canvas	2.25	2	0.5	2.25
3 3 0	2 2 0	3 3 2	1 0 0	Art Bell - Radio Station Affiliates	2.25	2	0.5	2.25
3 3 0	2 2 0	3 3 2	1 0 0	Art Bell - UFO Photos Page 2	2.25	2	0.5	2.25
3 3 0	2 2 0	3 3 3	1 0 0	Art Bell - Related Web Links	2.25	2	0.75	3.38
				Excite averages	2.25	2.00	0.50	2.25
3 3 0	0 1 0	1 1 0	0 1 1	Untitled	1	1.5	0	0.00
0 0 0	0 1 3	1 1 1	1 1 1	AfroFrontDoor	0.5	0.75	0.5	0.19
1 2 2	0 0 1	1 1 3	0 1 1	BigBook	0.5	1	1.25	0.63
2 3 2	0 0 3	1 1 3	2 2 1	Rick Krsticevic's Links	1.25	1.5	1.5	2.81
1 2 2	0 0 1	1 1 3	0 1 0	Four11 Directory Services	0.5	1	1.25	0.63
				ParaSite averages	0.6	1.00	0.90	0.85

Table D.9: User Ratings of Art Bell Recommendations

Excite	ParaSite
www.connix.com/~bruth/atari.html Entertainment Corner: Atari Cartridges	www.interplay.com/ Interplay Productions
www.primenet.com/~rworne/atari5200.shtml Classic Video Game Collecting	www.microsoft.com/ Microsoft Corporation Home Page
www.pribish.com/atari.list.8b.alphabetical.html Reinhard Pribish's Atari 400/800 Alphabetical List	www.cnet.com/ Welcome to CNET.COM
www.silcom.com/~halling/h-atari.htm Associated Enterprises	www.discovery.com/ Discovery Online
pw1.netcom.com/~kinggus/games/trade.html The Kingdom presents... Tradebait!	/www.ea.com/ Electronic Arts Online

Table D.10: Top 5 Pages Returned by Excite and ParaSite for Activision Seed URL (http://www.activision.com). Ellipsis in original.

P	K	W	**S**	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
3 3 3	1 1 3	3 2 3	1 1 0	Entertainment Corner...	2	1.75	2.25	7.88
2 2 3	1 1 3	3 3 3	1 1 1	Classic Video Game Collecting	1.75	1.75	2.5	7.66
2 2 3	1 1 3	3 2 2	1 0 0	Reinhard Pribish's Atari... List	1.75	1.25	2	4.38
2 2 3	1 1 3	3 2 2	1 0 0	Associated Enterprises	1.75	1.25	2	4.38
2 2 3	1 1 3	3 2 2	1 0 0	The Kingdom presents... Tradebait!	1.75	1.25	2	4.38
				Excite averages	3	2.67	2	16
- - -	3 3 1	3 3 2	3 2 2	Interplay Productions	1.25	1.5	0.75	1.41
0 0 0	3 2 1	2 2 2	0 2 1	Microsoft Corporation Home Page	1.5	2.25	2	6.75
1 3 3	0 0 0	3 3 3	2 3 2	Welcome to CNET.COM	0.67	1.67	2	2.22
1 3 3	0 0 0	1 2 3	- - -	Discovery Online	2.25	2.25	1.5	7.59
3 3 3	3 3 1	0 1 2	3 2 2	Electronic Arts Online				
				ParaSite averages				

Table D.11: User Ratings of Activision recommendations

D.5 Activision

The URLs returned by each system for the "Activision" home page are shown in Table D.10. Activision is a company that makes computer games. Excite returned 5 sites that listed cartridges for Atari games systems. As user S noted: "Activision made games for [Atari] once upon a time... the selected sites are very similar to each other but not to the seed url." ParaSite returned the home pages of five different companies, two of which make computer games. Users found the Excite selections slightly more relevant (1.8 vs. 1.73) and significantly more novel (2.15 vs. 2.65), while the ParaSite pages were judged more interesting (2.07 vs. 1.45). As one user summarized:

> System A [Excite] found Atari cartridge links and pages. Not useful. System B [ParaSite] found other gaming pages. I guess the usefulness of these would depend on what you felt like doing. —K

D.6 Happy Puppy

Table D.12 shows the URLs returned by each system for the "Happy Puppy" pages, the self-proclaimed "#1 Game Site on the Internet". Three of the four Excite recommendations were for pages on the same server. The fourth was a review of computer game sites. None of the ParaSite recommendations were on the Happy Puppy server. One was a computer gaming site, and one was a site for downloading shareware software. The two remaining suggestions were largely irrelevant.

Excite	ParaSite
happypuppy.com/games/kids.htm *Games - Happy Puppy's Kids Area*	hem.passagen.se/famjakob/musik.htm *Untitled*
www.happypuppy.com/cheats/ *Happy Puppy's Cheats Page*	www.gamesdomain.co.uk/ *Games Domain - The world's largest games site*
happypuppy.com/games/w3games.htm *Happy Puppy Games You Play on the WWW*	www.shareware.com/ *SHAREWARE.COM – the way to find shareware on the Internet*
netfind.aol.com/aol/Reviews/Hobbies/ Games/Computer/Game_Biz/ index.netfind.html *AOL NetFind: Reviews: Game Biz*	www.usc.edu *USCWeb, the University of Southern California*

Table D.12: Top 4 Pages Returned by Excite and ParaSite for Happy Puppy Seed URL (http://www.happypuppy.com).

P	K	W	S	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
3 3 0	3 0 0	3 3 3	1 1 0	Games - Happy Puppy's Kids Area	2.5	1.75	0.75	3.28
3 3 0	3 2 0	3 3 3	1 1 0	Happy Puppy's Cheats Page	2.5	2.25	0.75	4.22
3 3 0	3 2 0	3 3 3	1 1 0	Happy Puppy Games You Play...	2.5	2.25	0.75	4.22
3 3 3	3 2 2	3 3 3	3 2 1	AOL NetFind: Reviews: Game Biz	3	2.5	2.25	16.88
3 3 2	3 2 3	3 3 3	3 2 1	Games	0	0	0	0.00
				Excite averages	2.63	2.19	1.13	7.15
- - -	0 0 3	0 0 0	0 0 0	Untitled	0	0	0	0.00
3 3 3	3 3 2	3 3 3	3 2 1	Games Domain	3	2.75	2.25	18.56
3 3 3	2 2 1	1 2 2	2 1 1	SHAREWARE.COM	2	2	1.75	7.00
0 3 2	0 0 3	1 1 2	- - -	USCWeb	0.33	1.33	0.5	0.22
				ParaSite averages	1.33	1.52	1.13	6.45

Table D.13: User Ratings of HappyPuppy Recommendations

As Table D.13 shows, the Excite pages were judged more relevant (2.63 vs. 1.33) and interesting (2.19 vs. 1.52) than the ParaSite pages and equally interesting (1.13). As one user wrote:

> System A [Excite] ... came up with 3 references from the original home page. The last [reference looks] good.
> System B [ParaSite] came up with two great references. USCWeb is irrelevant, but interesting and novel. —P

D.7 Economist

The URLs returned by each system for the "Economist" home page are shown in Table D.14. *The Economist* is a weekly magazine that focuses on business and politics worldwide. One of the sites returned by Excite was a list of magazines containing news and commentary; a second was a list of business-related magazines. A third was a list of resources for economists, another an advertisement for a book about an economist, and the last one an essay by Engels. Note Excite's tendency to be literal (honing in on the word "economist") and to include lists that contain the name of the seed page. In contrast, ParaSite returned the home pages of two other news services (CNN and *The New York Times*), two statistic-intensive U. S. government bureaus, and the home page of an organization that supports educational research. The quantitative ratings of the systems were very similar, with Excite judged slightly more relevant (1.90 vs. 1.58) and novel (1.80 vs. 152) and about equally interesting (1.70 vs. 1.72), as shown in Table D.15. A subject summarized:

Excite	ParaSite
netfind.aol.com/aol/Reviews/News_and_Reference/ News_and_Commentary_Magazines/index.netfind.html *AOL NetFind: Reviews: News & Commentary Magazines*	www.cnn.com/ *CNN Interactive*
econwpa.wustl.edu/EconFAQ/EconFAQ.html *Resources for Economists on the Internet*	www.nytimes.com/ *The New York Times on the Web*
www.ecw.ca/Press/publications/stephen_leacock.htm *STEPHEN LEACOCK*	stats.bls.gov/blshome.html *Bureau of Labor Statistics*
www.uflib.ufl.edu/cm/business/stand.html *Business Library: Business Newsstand*	tikkun.ed.asu.edu/aera/home.html *American Educational Research Association*
leftside.uwc.ac.za/Archive/1844-DFJ/outlines.htm 1844: Outlines of a Critique of Political Economy	www.census.gov/ U.S. Census Bureau Home Page

Table D.14: Top 5 Pages Returned by Excite and ParaSite for Economist Seed URL (http://www.economist.com).

P	K	W	S	Title	Average			prod-
r i n	r i n	r i n	r i n		r	i	n	uct
3 3 2	2 2 2	3 3 1	3 1 1	AOL NetFind: Reviews: News...	2.75	2.25	1.5	9.28
3 3 3	2 1 3	3 3 1	1 1 1	Resources for Economists on the Internet	2.25	2	2	9.00
1 2 3	1 0 3	1 0 1	0 0 0	STEPHEN LEACOCK	0.75	0.5	1.75	0.66
3 2 2	2 2 2	3 3 1	3 2 1	Business Library: Business Newsstand	2.75	2.25	1.5	9.28
1 2 3	1 1 3	2 2 3	0 1 0	1844: Outlines of a Critique...	1	1.5	2.25	3.38
				Excite averages	1.90	1.70	1.80	6.32
- - -	1 3 0	2 2 2	2 2 2	CNN Interactive	1.5	2	1.33	4.00
2 2 1	1 1 0	3 3 1	3 2 2	The New York Times on the Web	2.25	2	1	4.50
2 2 3	1 1 3	3 3 2	1 2 2	Bureau of Labor Statistics	1.75	2	2.5	8.75
1 1 2	0 0 3	1 1 1	- - -	AERA	0.67	0.67	1	0.44
2 3 3	1 1 2	3 2 1	1 2 1	U.S. Census Bureau Home Page	1.75	2	1.75	6.13
				ParaSite averages	1.58	1.73	1.52	4.76

Table D.15: User Ratings of Economist Recommendations

It depends whether the originator is more interested in economics, current events or economic data. I am assuming that it is mainly the first two.
The first 2 suggestions by System A [Excite] are excellent. Most suggestions by both systems are rasonably relevant with System B [ParaSite] a bit more oriented to data and statistics.
A slight edge for System A [Excite]. —P

Bibliography

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996.

[2] Serge Abiteboul and Victor Vianu. Queries and computation on the web. In *The Sixth International Conference on Database Theory (ICDT)*, Delphi, Greece, January 1997.

[3] Jacques André, Richard Furuta, and Vincent Quint, editors. *Structured Documents*. Cambridge University Press, 1989.

[4] Gustavo O. Arocena, Alberto O. Mendelzon, and George A. Mihaila. Applications of a web query language. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Cruz, CA, April 1997.

[5] Justin Boyan, Dayne Freitag, and Thorsten Joachims. A machine learning architecture for optimizing web search engines. In Franz [10]. http://www.cs.cmu.edu/ reinf/papers/boyan.laser.ps.

[6] E. F. Codd. *The Relational Model for Database Management: Version 2*. Addison-Wesley, Reading, MA, 1990.

[7] C. J. Date and Hugh Darwen. *A Guide to the SQL Standard, 3rd Edition*. Addison-Wesley, Reading, MA, 1992.

[8] Arthur Do. *HtmlStreamTokenizer*, 1997. http://www-cs-students.stanford.edu/~do/html-streamer.html.

[9] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and padhr Smyth, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, Cambridge, MA, 1996.

[10] Alexander Franz, editor. *The AAAI-96 Workshop on Internet-based Information Systems*. AAAI, August 1996.

[11] H. P. Frei and Daniel Stieger. Making use of hypertext links when retrieving information. In *Proceedings of the Fourth ACM Conference on Hypertext*, pages 102–111, Washington, DC, 1992. Association for Computing Machinery.

[12] Richard Furuta. Defining and using structure in digital documents. In *Digital Libraries '94*, 1994.

[13] Claire L. Green and Peter Edwards. Using machine learning to enhance software tools for internet information management. In Franz [10].

[14] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[15] Eric N. Hanson and Jennifer Widom. An overview of production rules in database systems. *The Knowledge Engineering Review*, 8(2), June 1993.

[16] Michiaki Iwazume, Kengo Shirakami, Kazuaki Hatadani, Hideaki Takeda, and Toyoaki Nishida. Iica: An ontology-based internet navigation system. In Franz [10]. http://ai-www.aist-nara.ac.jp/doc/people/mitiak-i/aaai96/aaai96_5_12.html.

[17] M. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14:10–25, 1963.

[18] W. Eliot Kimber. HyTime and SGML: Understanding the HyTime HyQ query language 1.1. Available at "http://www.sgmlu.com/documents\HyTime\HyQ-1_1.txt", August 1993.

[19] Thomas Kistler and Hannes Marais. Webl – a programming language for the web. Technical Report 1997-029, Digital Systems Research Center, December 1997.

[20] David Konopnicki and Oded Shmueli. Information gathering in the world-wide web: The W3QL query language and the W3QS system. Available from "www.cs.technion.ac.il/~konop/w3qs.html", 1997.

[21] Brian A. LaMacchia. *Internet Fish*. PhD thesis, MIT, 1996.

[22] Michael Mauldin and John R. R. Leavitt. Web agent related research at the center for machine translation. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval*, 1994. http://fuzine.mt.cs.cmu.edu/mlm/signidr94.html.

[23] Alberto Mendelzon, George Mihaila, and Tova Milo. Querying the world wide web. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, Miami, FL, 1996.

[24] Alberto O. Mendelzon and Tova Milo. Formal models of web queries. In *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997.

[25] George A. Mihaila. WebSQL – an SQL-like query language for the world wide web. Master's thesis, University of Toronto, 1996.

[26] Marvin Minsky, editor. *Semantic Information Processing*. MIT Press, Cambridge, MA, 1968.

[27] Tom M. Mitchell. Challenge problems for artificial intelligence. Panel at AAAI-96, moderated by Bart Selman, 1996.

[28] Daniel E. O'Leary. The relationship between relevance and reliability in internet-based information and retrieval systems. In Franz [10]. http://www.usc.edu/dept/sba/atisp/AI/AI-Bus/internet/inter-rr.htm.

[29] Original Reusable Objects, Inc. *OROMatcher 1.0 User's Guide*, 1997. http://www.oro-inc.com/developers/docs/OROMatcher/index.html.

[30] Santeri Paavolainen. MD5 in Java JDK Beta-2. http://www.cs.hut.fi/~santtu/java/, 1996.

[31] M. R. Quillian. *Semantic Memory*. PhD thesis, Carnegie Institute of Technology, 1966.

[32] Dave Raggett. Html 3.2 reference specification. World-Wide Web Consortium technical report, January 1997.

[33] Paul Resnick and Hal R. Varian. Recommender systems (introduction to special section). *Communications of the ACM*, 40(3):56–58, March 1997.

[34] Ronald Rivest. The MD5 message-digest algorithm. Network Working Group Request for Comments: 1321, April 1992.

[35] Ronald Rousseau. Sitations: an exploratory study. *Cybermetrics*, 1(1), 1997. http://www.cindoc.csic.es/cybermetrics/articles/v1i1p1.html.

[36] Sriram Sankar, Sreenivasa Viswanadha, and Rob Duncan. *Java Compiler Compiler (JavaCC) - The Java Parser Generator*. Located at "http://www.suntest.com/JavaCC/".

[37] Jacques Savoy. Citation schemes in hypertext information retrieval. In Maristella Agosti and Alan F. Smeaton, editors, *Information Retrieval and Hypertext*, pages 99–120. Kluwer Academic Press, 1996.

[38] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the metacrawler. In *Proceedings of the 4th International World Wide Web Conference*, 1995.

[39] Jonathan Shakes, Marc Langheinrich, and Oren Etzioni. Dynamic reference sifting: A case study in the homepage domain. In *The Sixth International World Wide Web Conference*, April 1997. See "http://ahoy.cs.washington.edu:6060".

[40] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Computer-Human Interaction (CHI)*, 1995.

[41] H. Small. Co-citation in the scientific literature: a new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24:265–269, 1973.

[42] Ellen Spertus. Mining links. Thesis proposal, MIT Department of Electrical Engineering and Computer Science, September 1996.

[43] Randall H. Trigg. *A network-based approach to text handling for the online scientific community.* PhD thesis, University of Maryland, 1983. Also technical report TR-1346.

[44] S. Weibel, W. Cathro, and R. Iannella. The 4th dublin core metadata workshop report. *D-Lib Magazine*, June 1997. http://www.dlib.org/dlib/june97/metadata/06weibel.html.

[45] Ross Wilkinson and Michael Fuller. Integrated information aceess via structure. In Maristella Agosti and Alan Smeaton, editors, *Information Retrieval and Hypertext*, chapter 11. Kluwer Academic Publishers, 1996.

[46] William A. Woods. What's in a link: Foundations for semantic networks. In *Readings in Knowledge Representation*, chapter 11. Morgan Kaufmann, 1985.