15 Data Integration on the Web

The World Wide Web offers a vast array of data in many forms. The majority of this data is structured for presentation not to machines but to humans, in the form of HTML tables, lists, and forms-based search interfaces. These extremely heterogeneous sources were created by individuals around the world and cover a very broad collection of topics in over 100 languages. Building systems that offer data integration services on this vast collection of data requires many of the techniques described thus far in the book, but also raises its own unique challenges.

While the Web offers many kinds of structured content, including XML (discussed in Chapter 11) and RDF (discussed in Chapter 12), the predominant representation by far is HTML. Structured data appears on HTML pages in several forms. Figure 15.1 shows the most common forms: HTML tables, HTML lists, and formatted "cards" or templates. Chapter 9 discusses how one might extract content from a given HTML page. However, there are a number of additional challenges posed by Web data integration, beyond the task of wrapping pages.

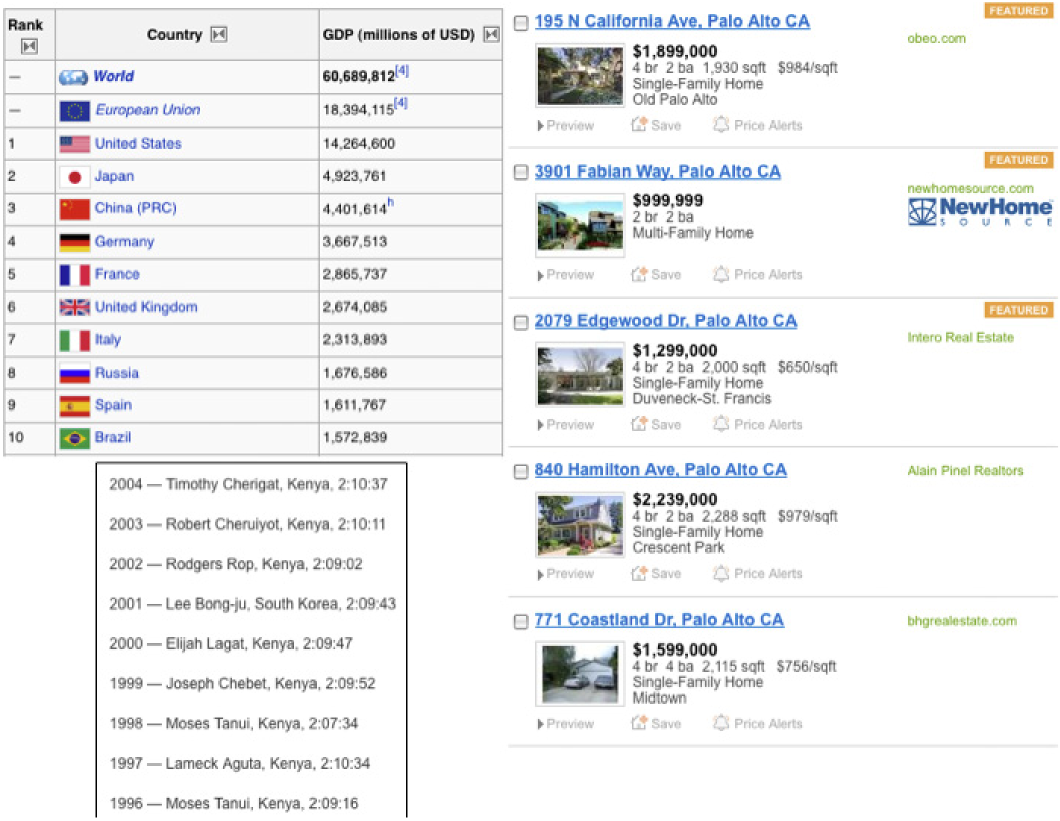
SCALE AND HETEROGENEITY

According to conservative estimates, there are at least a billion structured HTML data sets on the Web. Naturally, the quality of the data varies — it is often dirty, wrong, out of date, or inconsistent with other data sources. As discussed previously, the data may be in different languages. In many cases, we must have general and scalable techniques to make use of the data without requiring large amounts of human-administrator input.

MINIMAL STRUCTURAL AND SEMANTIC CUES

While to a viewer these data sets appear structured, a computer program faces several challenges to extract the structure from the Web pages. First, the visual structure evident in an HTML page may not be mirrored by consistent structure in the underlying HTML. Second, HTML tables are used primarily as a formatting mechanism for *arbitrary* data, and therefore the vast majority of the content that is formatted as HTML tables is actually not what we would consider high-quality structured data. When tabular data does appear on the Web, it comes with very little schema. At best, tables will have a header row with column names, but often it is tricky to decide whether the first row is a header row. The relations represented by the table are typically explained in the surrounding text and hard to extract. For example, the table on the bottom left of Figure 15.1 describes the list of winners of the Boston Marathon, but that fact is embedded somewhere in the surrounding text. Lists on Web pages present additional challenges. They have no schema at all, and every item in a list represents an entire row in a database. Hence, we need to first segment

Principles of Data Integration, DOI: 10.1016/B978-0-12-416044-6.00015-6



Rank	Country	GDP (millions of USD)
—	*World*	60,689,812[4]
—	*European Union*	18,394,115[4]
1	United States	14,264,600
2	Japan	4,923,761
3	China (PRC)	4,401,614[h]
4	Germany	3,667,513
5	France	2,865,737
6	United Kingdom	2,674,085
7	Italy	2,313,893
8	Russia	1,676,586
9	Spain	1,611,767
10	Brazil	1,572,839

2004 — Timothy Cherigat, Kenya, 2:10:37

2003 — Robert Cheruiyot, Kenya, 2:10:11

2002 — Rodgers Rop, Kenya, 2:09:02

2001 — Lee Bong-ju, South Korea, 2:09:43

2000 — Elijah Lagat, Kenya, 2:09:47

1999 — Joseph Chebet, Kenya, 2:09:52

1998 — Moses Tanui, Kenya, 2:07:34

1997 — Lameck Aguta, Kenya, 2:10:34

1996 — Moses Tanui, Kenya, 2:09:16

□ 195 N California Ave, Palo Alto CA
FEATURED
obeo.com
$1,899,000
4 br 2 ba 1,930 sqft $984/sqft
Single-Family Home
Old Palo Alto
Preview Save Price Alerts

□ 3901 Fabian Way, Palo Alto CA
FEATURED
newhomesource.com
NewHome SOURCE
$999,999
2 br 2 ba
Multi-Family Home
Preview Save Price Alerts

□ 2079 Edgewood Dr, Palo Alto CA
FEATURED
Intero Real Estate
$1,299,000
4 br 2 ba 2,000 sqft $650/sqft
Single-Family Home
Duveneck-St. Francis
Preview Save Price Alerts

□ 840 Hamilton Ave, Palo Alto CA
Alain Pinel Realtors
$2,239,000
4 br 2 ba 2,288 sqft $979/sqft
Single-Family Home
Crescent Park
Preview Save Price Alerts

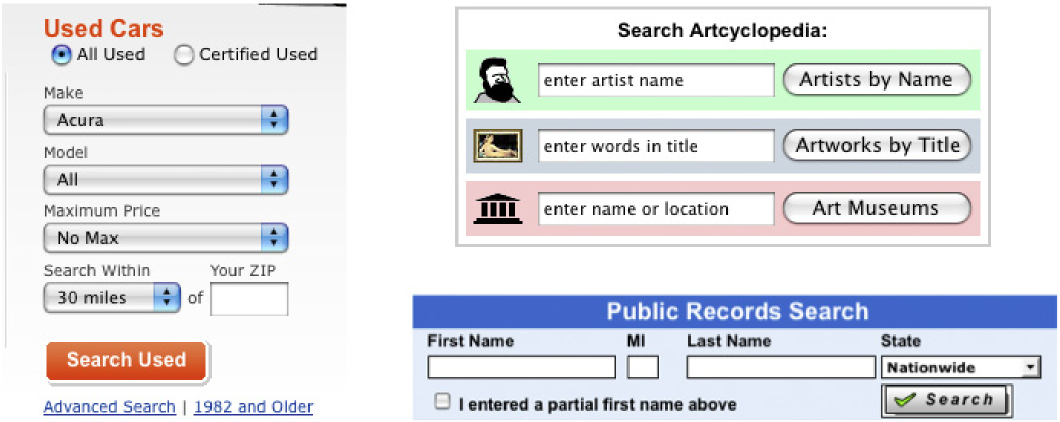
□ 771 Coastland Dr, Palo Alto CA
bhgrealestate.com
$1,599,000
4 br 4 ba 2,115 sqft $756/sqft
Single-Family Home
Midtown
Preview Save Price Alerts

FIGURE 15.1 The different kinds of structured data on the Web. The top left shows an HTML table of country GDPs, and the bottom left shows part of an HTML list of the winners of the Boston Marathon. On the right, we see a search result for real estate listings, where each result is formatted as a card.

each list item into a row of cells. Cards display the attributes of an object in a template that is repeated for every object. To extract the data, we must know the template's specific layout structure.

DYNAMIC CONTENT, ALSO KNOWN AS THE "DEEP WEB"

Many of the pages containing structured data are generated dynamically in response to user queries that are posed using HTML forms. A few example forms are shown in Figure 15.2. In some domains, such as cars, jobs, real estate, public records, events, and patents, there may be hundreds or thousands of forms in each domain. However, there is also a *long tail* effect — a large number of domains for which there are few forms. Examples of more exotic domains include quilts, horses for sale, and parking tickets in particularly well-organized municipalities.



Used Cars

● All Used ○ Certified Used

Make: Acura

Model: All

Maximum Price: No Max

Search Within: 30 miles of Your ZIP

State: Nationwide

☐ I entered a partial first name above

FIGURE 15.2 Sample HTML forms from the Web. The form on the left enables searching for used cars. The form on the top right is for searching art works, and the one on the bottom right is for searching public records.

Since form-based sources offer a structured query interface, much of the work on data integration on the Web is focused on providing uniform access to a multitude of forms. Accessing this data conveniently is even more important because the data are often hidden from Web search engines. Specifically, crawlers of Web search engines typically collect pages by following hyperlinks from pages they have already indexed. However, pages generated by form invocations are dynamically created on the fly, and typically have no incoming links. For this reason, this content has been referred to as the *deep Web* or *invisible Web*, and contrasted with the *surface Web*. The number of such form sources is estimated to be in the tens of millions, and estimates of the amount of content on the deep Web vary from being the size of the surface Web to being two orders of magnitude bigger.[1]

15.1 What Can We Do with Web Data?

In contrast to other types of data integration, Web sources often provide greater heterogeneity, fewer cues that help an automated system reason about what to do, and larger scale. Worse, at scale, we are often limited in the amount of human or even automated processing we can do on an individual source. So a natural question is whether and how we can harness Web data.

The answer is that broadly, we can use data sources on the Web for a variety of services, ranging from data integration to improving search. Before describing the specific techniques for extracting and querying such data, it is instructive to first consider the possible uses of structured data on the Web.

[1] These are estimates of sheer size, not of useful content.

DATA INTEGRATION

A natural goal in the presence of many data sources on a particular topic is to provide a single query interface that retrieves data from all these sources. For example, instead of expecting users to visit multiple job search sites, we can create a single site that integrates the data from multiple sites. Such engines are typically referred to as *vertical-search* engines.

A slightly different goal is to create a *topical portal* that integrates all the data on a particular domain. For example, consider a site that integrates all the information about the database research community, including its researchers, conferences, and publications. Here, different aspects of the data come from different sites. For example, data about researchers may come from one set of sources, while data about publications will come from another. In contrast, a vertical-search engine typically integrates data from multiple sites that all offer the same kind of information (e.g., cars for sale).

A third category is transient or "one-off" data integration tasks. Here, we may combine data from multiple sources in order to create a data set that will only be needed briefly. For example, consider a disaster relief effort where we need to rapidly create an online map displaying shelters with their contact details, or a student working on a course project who needs to collect information about water availability and GDP for various countries. Unlike the first two data integration tasks, here we do not expect technically skilled users, which leads to a different system design.

IMPROVED WEB SEARCH

There are several ways to use structured data to improve Web search. One can think of Web search as a special case of data integration, where no joins are performed (only unions).

As noted above, the deep Web, whose contents are not available to search engines, presents a significant gap in the coverage of search engines and therefore in search quality. Another opportunity lies in using the layout structure as a relevance signal. For example, if we find a page with a table that has a column labeled population and a row that has Zimbabwe in the left column, then that page should be considered relevant to the query "zimbabwe population" even if the occurrence of population on the page is far from the occurrence of Zimbabwe.

Another long-standing goal for Web search is to return facts as answers. For example, if the user queries for "zimbabwe population" and the appropriate answer exists in a table, we can return that number with a pointer to the source data. We can be even more ambitious and return *derived* facts. For example, for the query "africa population" we could calculate the answer from a table that contains the populations of all countries. Today's Web search engines offer factual answers to queries in very limited domains (e.g., weather, sports results) and with the support of contracted data sources.

Finally, often the intention of our search is to find structured data. For example, as part of a class assignment we may be looking for a data set containing crime rates in different American cities. Today there is no explicit way of telling the search engine that we are looking for a table of data, and therefore we are limited to the tedious process of scanning the results manually for those containing structured data.

We now discuss several bodies of work that try to provide the services described above. Section 15.2 describes two bodies of work for leveraging the content of the deep Web: search engines (e.g., cars for sale), which apply many of the ideas of virtual data integration systems, and deep-Web surfacing, which attempts to crawl through forms and find useful HTML pages. Section 15.3 discusses topical portals that attempt to assemble a broad range of data on a particular domain (e.g., find all that is known about database researchers). Finally, Section 15.4 discusses the common case of data integration on the Web where users need to integrate data sets for transient tasks (e.g., correlating coffee production of a country with its population).

15.2 The Deep Web

The deep Web contains data on a broad range of domains and in many languages. Users access content on the deep Web by filling out forms. When a query is submitted, an answer page is dynamically generated. Our discussion here is focused on deep-Web sources that provide data on a particular domain (e.g., cars, patents) and not on forms that provide generic search (e.g., www.google.com) or forms that provide a "search this site" functionality.

Before we describe how to integrate data across deep-Web sites, we review the basic constructs of Web forms. An HTML form is defined within a special HTML form tag (see Figure 15.3 for the HTML specifying the form shown in Figure 15.4). The action field identifies the server that will perform the query processing in response to the form submission. Forms can have several input controls, each defined by an input tag. Input controls can be of a number of types, the prominent ones being text boxes, select menus (defined in

```
<form action="http://jobs.com/find" method="get">
  <input type="hidden" name="src" value="hp"/>
  Keywords: <input type="text" name="kw"/>
  State: <select name="st">
              <option value="Any"/>
              <option value="AK"/>
              <option value="AL"/>
              ...
         </select>
  Sort By: <select name="sort">
              <option value="salary"/>
              <option value="startdate"/>
              ...
           </select>
  <input type="submit" name="s" value="go"/>
</form>
```

FIGURE 15.3 HTML for defining a form. The form defines the set of fields, their types, menu options, and the server that will process the request.

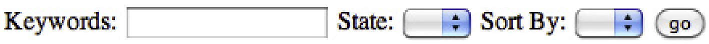


FIGURE 15.4 The rendering of the form specified in Figure 15.3.

a separate select tag), check boxes, radio buttons, and submit buttons. Each input has a name, which is typically not the name that the user sees on the HTML page. For example, in Figure 15.3 the user will see a field Keywords, but the name by which it can be identified by the browser or a Javascript program is kw. Users select input values either by entering arbitrary keywords into text boxes or by selecting from predefined options in select menus, check boxes, and radio buttons. In addition, there are hidden inputs whose values are fixed and are not visible to users interacting with the form. These are used to provide the server additional context about the form submission (e.g., the specific site from which it came). Our discussion focuses on the select menus and text boxes. Check boxes and radio buttons can be treated in the same way as select menus.

The example in Figure 15.3 includes a form that lets users search for jobs. When a form is submitted, the Web browser sends an HTTP request with the inputs and their values to the server using one of two methods: GET or POST. With GET, the parameters are appended to the action and included as part of the URL in the HTTP request. For example, the URL

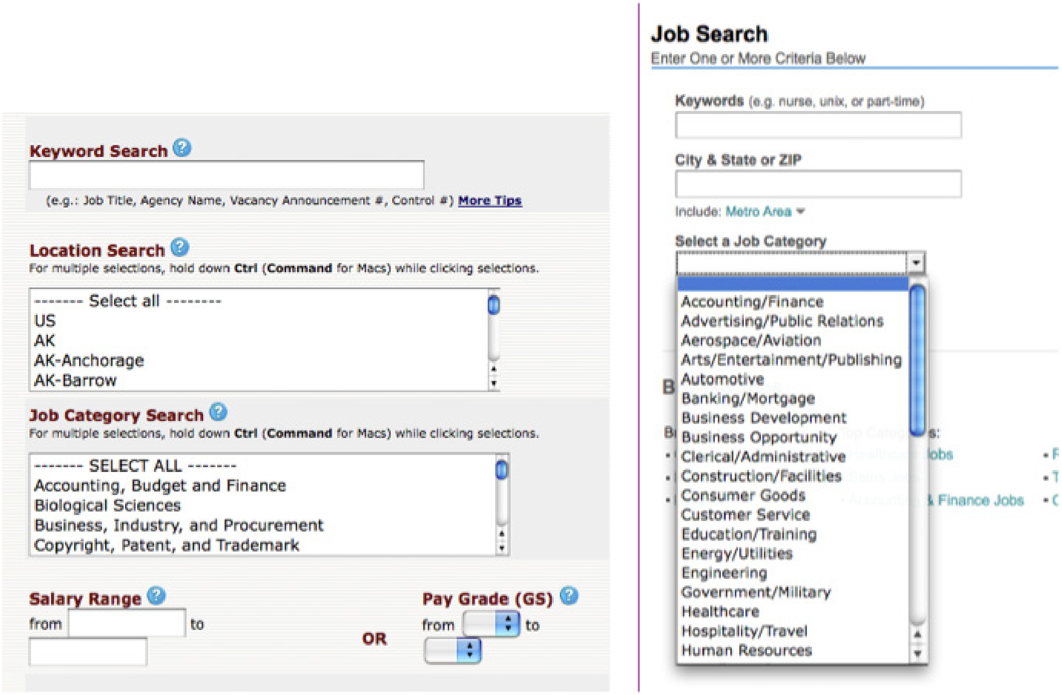
```
http://jobs.com/find?src=hp&kw=chef&st=Any&sort=salary&s=go
```

defines the query looking for chef positions in *any* state and sorting the results by salary. With POST, the parameters are sent in the body of the HTTP request and the URL is simply the action (e.g., `http://jobs.com/find`). Hence, the URLs obtained from forms that use GET are unique and encode the submitted values, while the URLs obtained with POST are not. One reason that this distinction is important is that a search engine index can treat a GET URL as any other page, thereby storing result pages for specific queries.

There are two main approaches for querying on the deep Web. The first (Section 15.2.1) builds vertical-search engines that integrate data on very narrow domains from thousands of deep-Web sources. The second (Section 15.2.2) attempts to crawl past forms and add pages to the search engine index from millions of sites on any domain.

15.2.1 Vertical Search

We begin by describing vertical-search engines, whose goal is to provide a single point of access to a set of sources in the same domain. In the most common domains of vertical search, such as cars, jobs, real estate, and airline tickets, there are thousands of sites with possibly relevant content. Even after restricting to sites that are relevant to a particular metropolitan area, there may be several tens of relevant sites — too many to browse manually. Figure 15.5 illustrates that even for simple domains such as job search, there is


Keyword Search
(e.g.: Job Title, Agency Name, Vacancy Announcement #, Control #) More Tips
Location Search
For multiple selections, hold down Ctrl (Command for Macs) while clicking selections.
------- Select all --------
US
AK
AK-Anchorage
AK-Barrow
Job Category Search
For multiple selections, hold down Ctrl (Command for Macs) while clicking selections.
------- SELECT ALL -------
Accounting, Budget and Finance
Biological Sciences
Business, Industry, and Procurement
Copyright, Patent, and Trademark
Salary Range
from
to
OR
Pay Grade (GS)
from
to
Job Search
Enter One or More Criteria Below
Keywords (e.g. nurse, unix, or part-time)
City & State or ZIP
Include: Metro Area
Select a Job Category
Accounting/Finance
Advertising/Public Relations
Aerospace/Aviation
Arts/Entertainment/Publishing
Automotive
Banking/Mortgage
Business Development
Business Opportunity
Clerical/Administrative
Construction/Facilities
Consumer Goods
Customer Service
Education/Training
Energy/Utilities
Engineering
Government/Military
Healthcare
Hospitality/Travel
Human Resources

USAJobs.com

Hotjobs.yahoo.com

FIGURE 15.5 Two different Web forms for searching job listings.

significant variability between the fields of different forms. Hence, a vertical-search engine needs to resolve heterogeneity. As such, vertical-search engines are a specialization of virtual data integration systems as we have described in earlier chapters. We consider each of their components in turn.

MEDIATED SCHEMA

The mediated schema of a vertical-search engine models the important properties of the objects under consideration. Some of the attributes in the mediated schema will be *input attributes* and will appear in the form that the users access. Other attributes will be *output attributes* and will only be seen in the search results pages. For example, a mediated schema for job search that integrates data from the two sources shown in Figure 15.5 would include the attributes shown in the form, such as category, keywordDescription, city, state, and the attributes openingDate and employingAgency that are only shown with the results. Note that in some cases, the attributes shown on the form may be minimum and maximum values for attributes of the schema (e.g., minPay).

SOURCE DESCRIPTIONS

The source descriptions in a vertical-search engine are also relatively simple because logically the sources expose a single relation. Hence, the main components of the source descriptions are (1) contents of the source (e.g., books, jobs, cars), (2) selection conditions (e.g., relevant geographical location, price ranges), (3) the attributes that can be queried on, (4) attributes that appear in the results, and (5) access-pattern restrictions, i.e., which input fields, or combinations thereof, are required to pose queries (see Section 3.3).

Example 15.1

A source description for the USAJobs.com site shown in Figure 15.5 would look as follows:

Contents:	jobs
Constraints:	employer = USA Federal Government
Query attributes:	keywords, category, location, salaryRange, payGrade
Output attributes:	openingData, employingAgency
Access restrictions:	at least one field must be given

Constructing the source descriptions can be done using the techniques described in Chapter 5. In particular, since there is quite a bit of similarity between different forms in a particular domain, schema matching techniques based on learning from experience (see Section 5.8) are especially effective.

The type of heterogeneity we see on the Web does present some unique challenges to query reformulation. In particular, forms often use drop-down menus to enable the user to make selections. For example, a real estate site may have a drop-down menu specifying the kind of property to search for (single-family home, condo, vacation property, etc.). However, these categories do not always line up with each other easily. For example, in some cities the site may feature the option lakefront property, and there will be no corresponding category in other cities. A similar challenge is dealing with ranges (e.g., price ranges). The ranges of the mediated schema may not map nicely onto ranges in each of the data sources, and therefore the system may need to reformulate a single range into a union of ranges.

WRAPPERS

There are two components to the wrappers of vertical-search engines: (1) posing the query to the underlying site and (2) processing the returned answer. Posing the query is relatively simple. After determining which query needs to be posed on a particular site, the system needs to create the HTTP request that would have been generated if the user posed the appropriate query directly on that site. For sites that use the HTML GET method for querying back-end databases, this amounts to creating a URL that contains the query parameters. For sites using POST, the query parameters need to be sent as part of the HTTP request.

There is an interesting wrinkle here, though, because some sites have more complex interactions. For example, it is common for real estate sites to guide the user through several steps until she can query for houses. In the first step the user will select a state, and in the second step she will select a county from that state. Finally, the third page the user encounters allows her to query for houses in the chosen county. Of course, the goal of the vertical-search engine is to abstract all these details from the user. Hence, if the user poses a query on a particular city, then the system needs to know how to arrive at the appropriate county search page.

In terms of processing the HTML pages returned for queries, we have two main options. We can either present the HTML page to the user as is (with an option of easily navigating between answers from different Web sites), or try to extract structured results from the returned page. The former option is much easier to implement and also has the advantage of being more friendly to the underlying sources. Many Web sites rely on ad revenue to support their operation and would greatly prefer that the vertical-search engine drive traffic to their site. Parsing the results from the answer pages requires applying information extraction techniques (see Chapter 9). However, these techniques typically require training machine learning algorithms per site, making it harder to scale the system up to a large number of sources.

15.2.2 Surfacing the Deep Web

Conceivably, we could follow the approach of building vertical-search engines for each of the thousands of domains that exist on the deep Web. However, such an approach would be impractical for several reasons. First, the human effort to create such a broad schema covering many domains would be prohibitive. In fact, it is not clear that such a schema can be built. In addition to covering many domains, the schema would have to be designed and maintained in over 100 languages and account for subtle cultural variations. Trying to break down the problem into smaller pieces is also tricky, because the boundaries of domains are very loosely defined. For example, starting from the domain of biology, it is easy to drift into medicine, pharmaceuticals, etc. Of course, the cost of building and maintaining source descriptions for millions of sources is also prohibitive. Finally, since we cannot expect users to be familiar with the mediated schema, or even easily find the one domain in which they are interested out of the thousands available, the system must be designed to accept keyword queries in arbitrary structure and on any domain. The problem of deciding whether a keyword query can be reformulated into a structured query in one of these many domains is a challenging and yet-unsolved problem.

A more practical approach to providing access to the full breadth of the deep Web is called *surfacing*. In this approach, the system guesses a relevant set of queries to submit to forms that it finds on the Web. The system submits these queries and receives HTML pages that are then entered into the search-engine index (hence, they become part of the surface Web). At query time, the surfaced pages participate in ranking like any other page in the index.

Except for freeing us from the effort of building a mediated schema and mappings, the main advantage of the surfacing approach is that it leverages the indexing and ranking systems of the search engine, which have become carefully tuned large software systems. The URLs that are stored in the index are the dynamic URLs associated with specific queries on the forms. Hence, when a user clicks on a result that is a surfaced page, she will be directed to a page that is created dynamically at that moment.

The main disadvantage of the surfacing method is that we lose the semantics associated with the page. Suppose we surfaced pages based on filling a form on a patents Web site and entering "chemistry" into the topic field. The word "chemistry" may not appear on some of the resulting pages, and therefore they may not be retrieved for a query asking for "chemistry patents." While this problem may be alleviated by adding the word "chemistry" into the document when it is indexed, it would be impractical to add all of chemistry's subfields (e.g., material science, polymers) or related fields.

Surfacing a deep-Web site requires an algorithm that examines an HTML form and returns a set of relevant and well-formed queries to submit to it. To scale to millions of forms, such an algorithm cannot involve any human intervention. We now describe the two main technical challenges that such an algorithm faces: (1) determining which subsets of fields to use for providing inputs and (2) determining good values to put into text fields.

(1) Determining input combinations: HTML forms typically have more than one input field. Hence, a naive strategy of enumerating the entire Cartesian product of all possible values of all inputs can result in a very large number of queries being posed. Submitting all these queries would drain the resources of the Web crawler and may often pose an unreasonable load on Web servers hosting the HTML forms. Furthermore, when the Cartesian product is very large, it is likely that a large number of the result pages are empty and hence useless from an indexing standpoint. As an example, a particular search form on cars.com has five inputs and a Cartesian product yields over 240 million URLs, though there are fewer than a million cars for sale on cars.com.

A heuristic for choosing input combinations that has proved useful in practice is to look for *informative inputs*. Intuitively, a field f (or a set of fields) in a form is informative if we obtain qualitatively different pages when we vary the values of f and hold all the other values constant. Consider a job search site that has, among others, the input fields state and sort by. Filling in the different values for state would lead to fetching pages that contain jobs in different states and would probably be relatively different from each other. In contrast, filling in the different values for the sort by field would only change the order of the results on the page; the content of the page would remain relatively the same.

Finding good input combinations can be done in a bottom-up fashion. We first consider each of the fields in the form in isolation and determine which ones are informative. We then consider pairs of fields, where at least one of them is informative, and check which pairs are informative. Continuing, we consider sets of three fields that contain an informative pair, and so on. In practice, it turns out that we rarely need to consider combinations that include more than three fields, though we may have to consider a few field combinations for a particular form.

(2) **Generating values for text fields:** When a field is a drop-down menu, the set of values we can provide it is given. For text fields, we need to generate relevant values. The common method of doing so is based on *iterative probing*. Initially, we predict candidate keywords by analyzing the text on pages from the site containing the form page that are available in the index. We test the form with these candidate keywords, and when valid form submissions result, we extract more keywords from the resulting pages. This iterative process continues until either new candidate keywords cannot be extracted or a prespecified target number of words is reached. The set of all candidate keywords can then be pruned to select a smaller subset that ensures diversity of the exposed database contents.

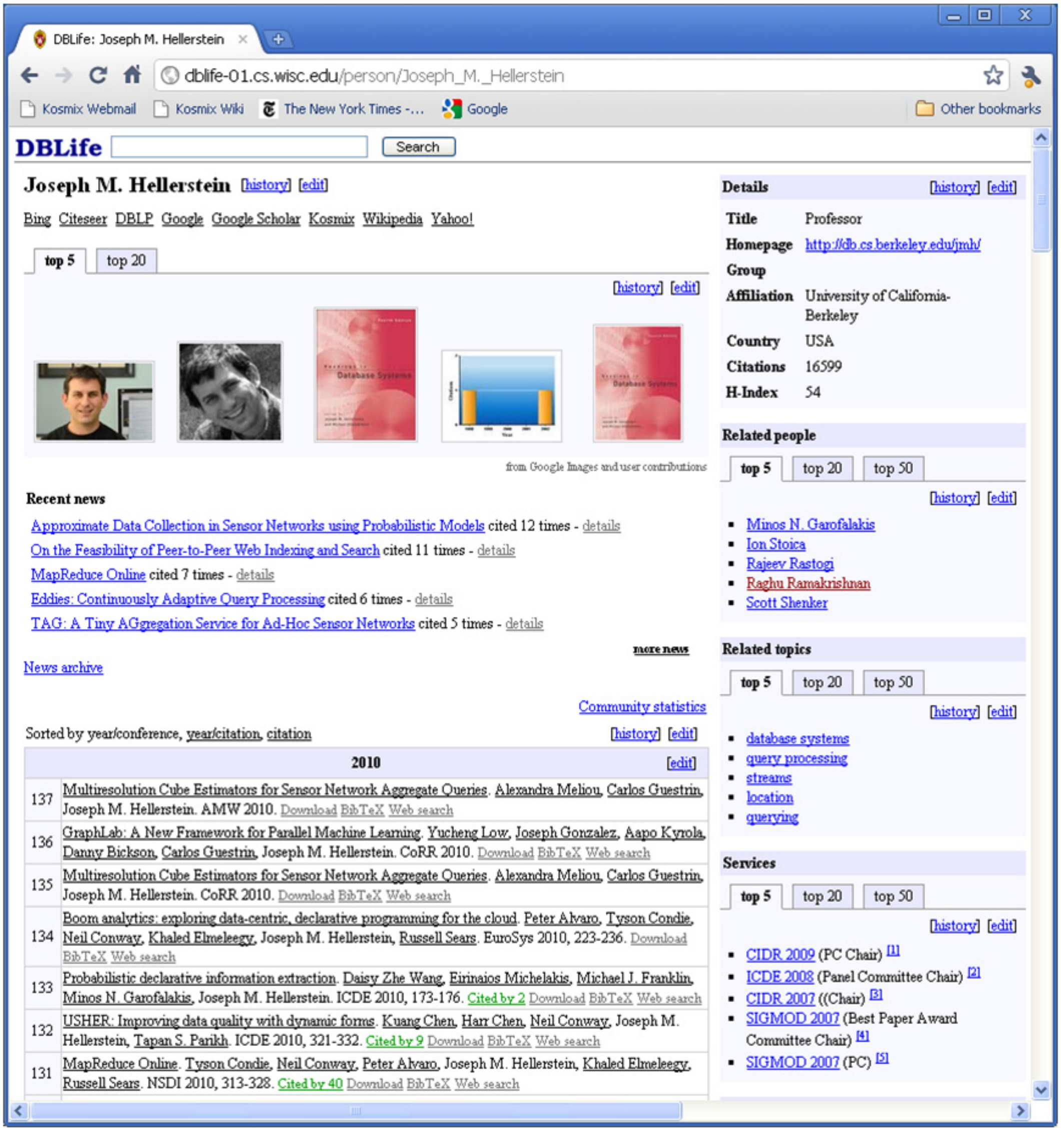
COVERAGE OF THE CRAWL

The surfacing algorithm will get some of the data from the deep-Web site, but does not provide any guarantees as to how much of the site it has covered. In fact, typically we do not know how much data exist in a deep-Web database, so it would be impossible to check if we have it all. In practice it turns out that it suffices to surface some of the content from the deep-Web site, even if it is not complete. With a good enough sample of the site's content, there is a substantial increase in the number of relevant queries that get directed to that site. Moreover, once some of the site is surfaced, the search engine's crawler will discover new deep-Web pages by following links from the surfaced pages. Finally, we note that the surfacing technique is limited to deep-Web sites that are powered by the GET method, not POST. In practice, many sources that accept POST also accept GET requests.

15.3 Topical Portals

A topical portal offers an integrated view of an entire topic. For example, consider a portal that pulls together all the information about researchers in a particular research community. The site would fuse data about individuals in the field, their advisers and students, their respective publications, the conferences and journals in the field, and members of program committees of conferences. As an example, the site can create *superpages* for individuals, as shown in Figure 15.6.

Topical portals are different from vertical-search engines in the type of integration they perform. Conceptually, the data underlying both types of sites are a table. A vertical-search engine integrates *rows* from different sources (i.e., a union of many sites). For example, each deep-Web site contributes a set of job listings, each represented as a row in the table. In contrast, a topical portal integrates *columns* from different sites (i.e., a join of many sites). For example, the column about publications could come from one site, while the column about affiliation would come from another. Hence, somewhat ironically,


DBLife: Joseph M. Hellerstein
dblife-01.cs.wisc.edu/person/Joseph_M._Hellerstein
Kosmix Webmail
Kosmix Wiki
The New York Times -...
Google
Other bookmarks
Search

DBLife

Joseph M. Hellerstein [history] [edit]

Bing Citeseer DBLP Google Google Scholar Kosmix Wikipedia Yahoo!

top 5 top 20

[history] [edit]

from Google Images and user contributions

Recent news

Approximate Data Collection in Sensor Networks using Probabilistic Models cited 12 times - details
On the Feasibility of Peer-to-Peer Web Indexing and Search cited 11 times - details
MapReduce Online cited 7 times - details
Eddies: Continuously Adaptive Query Processing cited 6 times - details
TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks cited 5 times - details

more news

News archive

Community statistics

Sorted by year/conference, year/citation, citation [history] [edit]

2010	[edit]
137	Multiresolution Cube Estimators for Sensor Network Aggregate Queries. Alexandra Meliou, Carlos Guestrin, Joseph M. Hellerstein. AMW 2010. Download BibTeX Web search
136	GraphLab: A New Framework for Parallel Machine Learning. Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, Joseph M. Hellerstein. CoRR 2010. Download BibTeX Web search
135	Multiresolution Cube Estimators for Sensor Network Aggregate Queries. Alexandra Meliou, Carlos Guestrin, Joseph M. Hellerstein. CoRR 2010. Download BibTeX Web search
134	Boom analytics: exploring data-centric, declarative programming for the cloud. Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, Russell Sears. EuroSys 2010, 223-236. Download BibTeX Web search
133	Probabilistic declarative information extraction. Daisy Zhe Wang, Eirinaios Michelakis, Michael J. Franklin, Minos N. Garofalakis, Joseph M. Hellerstein. ICDE 2010, 173-176. Cited by 2 Download BibTeX Web search
132	USHER: Improving data quality with dynamic forms. Kuang Chen, Harr Chen, Neil Conway, Joseph M. Hellerstein, Tapan S. Parikh. ICDE 2010, 321-332. Cited by 9 Download BibTeX Web search
131	MapReduce Online. Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, Russell Sears. NSDI 2010, 313-328. Cited by 40 Download BibTeX Web search

Details [history] [edit]

Title	Professor
Homepage	http://db.cs.berkeley.edu/jmh/
Group	
Affiliation	University of California-Berkeley
Country	USA
Citations	16599
H-Index	54

Related people

top 5 top 20 top 50

[history] [edit]

- Minos N. Garofalakis
- Ion Stoica
- Rajeev Rastogi
- Raghu Ramakrishnan
- Scott Shenker

Related topics

top 5 top 20 top 50

[history] [edit]

- database systems
- query processing
- streams
- location
- querying

Services

top 5 top 20 top 50

[history] [edit]

- CIDR 2009 (PC Chair) [1]
- ICDE 2008 (Panel Committee Chair) [2]
- CIDR 2007 ((Chair) [3]
- SIGMOD 2007 (Best Paper Award Committee Chair) [4]
- SIGMOD 2007 (PC) [5]

FIGURE 15.6 A superpage created by integrating many sources of information about Joseph M. Hellerstein.

vertical-search engines perform horizontal integration, while portal sites perform vertical integration.

A typical, but somewhat naive, approach to building a topical portal would be the following. First, conduct a *focused crawl* of the Web that looks for pages that are relevant to the topic. To determine whether a page is relevant, the crawl can examine the words on the page, the entities mentioned in it, or links to and from the page. Second, the system would apply a variety of general information extraction techniques (see Chapter 9) to find facts on the pages. For example, the system would try to extract relations such as advisedBy and authoredBy. Finally, given all the facts that were extracted from the pages, the system would put them together into a coherent knowledge base. The main challenge in this step is to reconcile multiple references to the same real-world objects, a topic we discussed in detail in Chapter 7.

A more effective approach to building topical portals relies on the observation that there is a collection of a few sites that provide most of the high-quality information in the domain. Furthermore, these sites are typically known to experts in the field. In our example, the combination of the DBLP[2] Web site, the sites of the top conferences and journals in the field, and the homepages of several hundreds of the top researchers in the field already provide a significant portion of the information about the database research community. Based on this approach, building the site proceeds in two steps.

INITIAL SETUP

We begin by deciding on the main sets of entities and relationships that we want our site to model. In our example, the entity sets may be person, conference, programCommittee, and publication, and the relationships can be authoredBy, advisedBy, and memberOf. We then select an initial set of sources in the domain, denoted by S_{int}.

We create extractors for each of the entity sets and relationships. These extractors will be more accurate than general-purpose extraction algorithms because they are aware of the specific patterns relevant to the relationships they are meant to extract. Therefore, we obtain a seed of accurately extracted entities and relationships. Moreover, if we want even more accurate data, we can tailor the extractors to be aware of the specific structure of particular sites. For example, an extractor that is aware of the DBLP site can yield a very high-quality database of authors and their publications. Another side benefit of this approach is that the problem of reference reconciliation from such sites also tends to be easier because these sites are more careful about maintaining unique naming conventions for individuals.

EXTENDING THE PORTAL

We consider two of the many methods to extend the data coverage of the portal. The first method relies on the heuristic that any important piece of information related to the topic will ultimately appear on one of the sites in S_{int} or on a page that is linked from a page in S_{int}. For example, if a new workshop is being organized on a novel area of database research, it is likely that we will find a pointer to it from one of our core sites. Hence, to extend the portal we monitor the initial collection of sites on a regular basis to expand it by finding new pages that are reachable from these sites.

The second method is based on *collaboration* among the users of the portal site. The portal provides a set of mechanisms that allow the users to correct data and add new data to the system. For example, users can be asked whether two strings refer to the individual or to correct an erroneous extraction. One of the key challenges in providing these mechanisms is to offer incentives for members of the community to collaborate. A simple

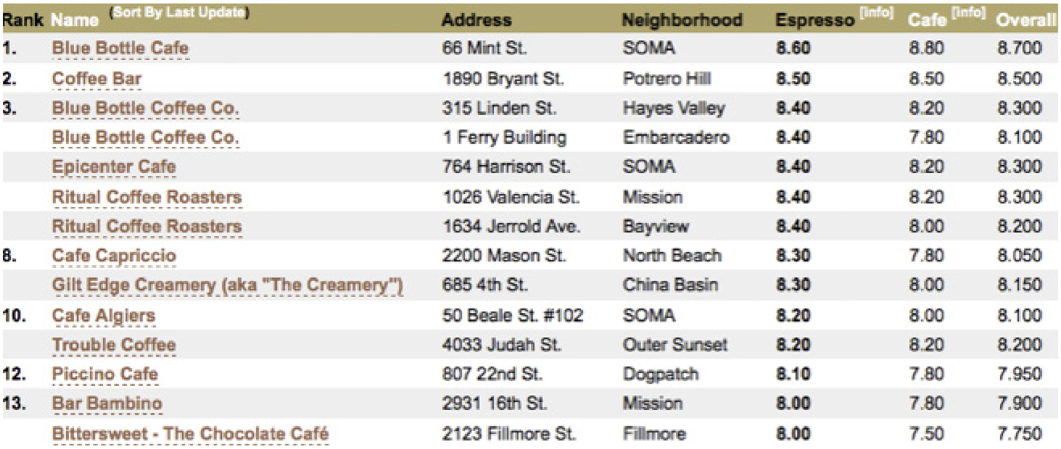
[2] A well-known site that lists publications in computer science conferences and journals.

kind of incentive is immediate positive feedback: the user should instantly see how his correction improves the site. A second set of incentives is based on publicly recognizing major contributors. Finally, the system must also ensure that it is not easily misled by erroneous or malicious contributions (e.g., from graduate students seeking a premature promotion).

15.4 Lightweight Combination of Web Data

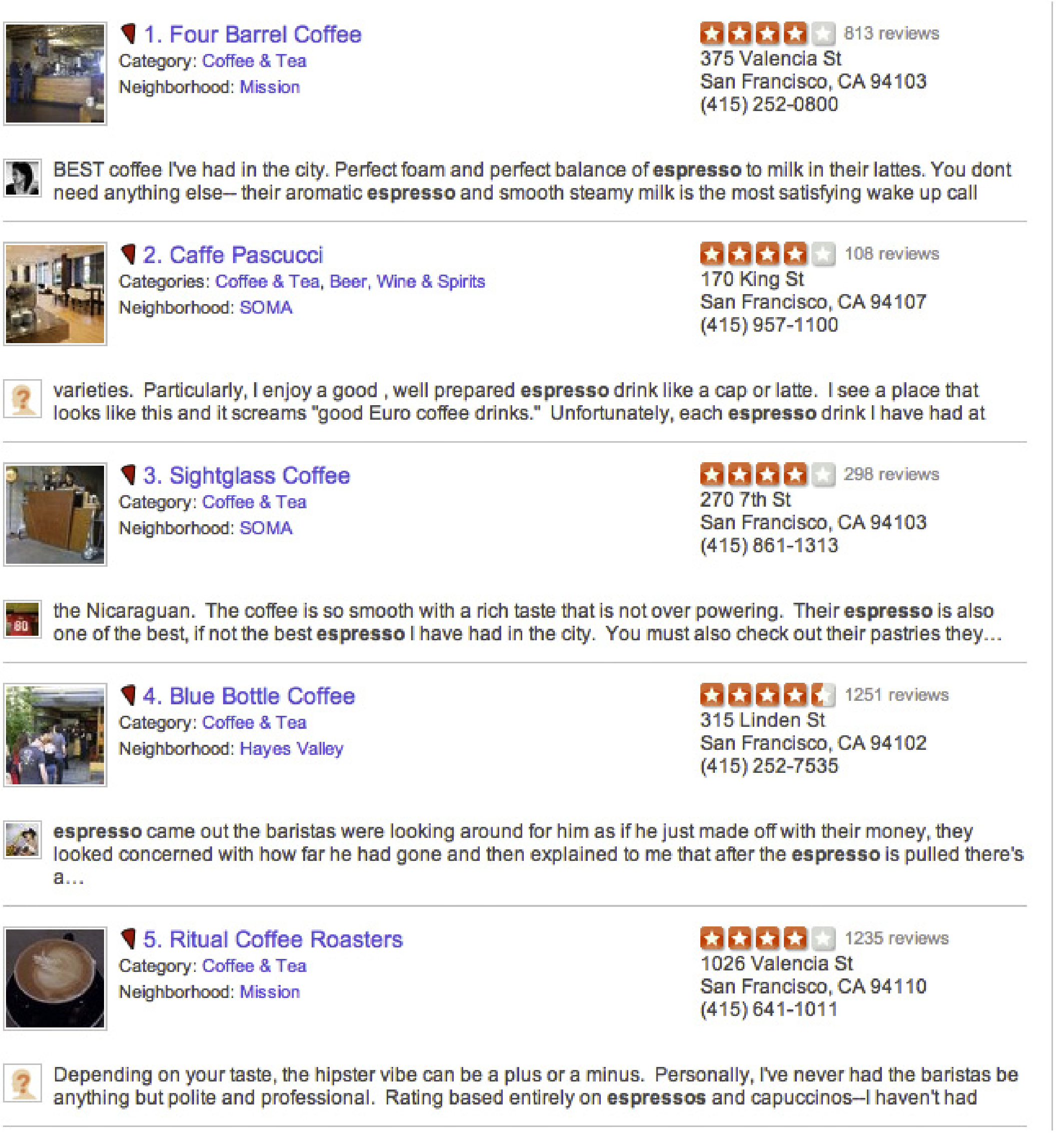
The rich collection of structured data sources on the Web offers many opportunities for ad hoc data integration. For example, consider a coffee enthusiast who wants to compile a list of cafes with their various quality ratings, using data of the type shown in Figures 15.7 and 15.8, or a disaster relief effort where we need to rapidly create an online map displaying shelters with their contact details and respective capacity. Similarly, consider a journalist who wants to add a data visualization to a news story that has a lifetime of a few days. In some scenarios we may integrate only data that are available on the Web, while in others we may integrate Web data with our own private data. Such combinations of data are often called *mashups*, and they have become a common way of providing data services on the Web.

Unlike standard data integration applications where it is expected that many similar queries will be posed against a relatively stable federation of sources for an extended period of time, the kinds of data integration we discuss here may be transient or even "one-off" tasks. The disaster relief mashup will hopefully be needed for only a short amount of time, and the coffee enthusiast may only need the list of cafes during his trip to the area. Hence, the challenge of lightweight data integration on the Web is to radically reduce the time and effort needed to set up the integration. In fact, there should not even be an explicit setup stage — data discovery, collection, cleaning, and analysis should be seamlessly intertwined.



Rank	Name (Sort By Last Update)	Address	Neighborhood	Espresso [Info]	Cafe [Info]	Overall
1.	Blue Bottle Cafe	66 Mint St.	SOMA	8.60	8.80	8.700
2.	Coffee Bar	1890 Bryant St.	Potrero Hill	8.50	8.50	8.500
3.	Blue Bottle Coffee Co.	315 Linden St.	Hayes Valley	8.40	8.20	8.300
	Blue Bottle Coffee Co.	1 Ferry Building	Embarcadero	8.40	7.80	8.100
	Epicenter Cafe	764 Harrison St.	SOMA	8.40	8.20	8.300
	Ritual Coffee Roasters	1026 Valencia St.	Mission	8.40	8.20	8.300
	Ritual Coffee Roasters	1634 Jerrold Ave.	Bayview	8.40	8.00	8.200
8.	Cafe Capriccio	2200 Mason St.	North Beach	8.30	7.80	8.050
	Gilt Edge Creamery (aka "The Creamery")	685 4th St.	China Basin	8.30	8.00	8.150
10.	Cafe Algiers	50 Beale St. #102	SOMA	8.20	8.00	8.100
	Trouble Coffee	4033 Judah St.	Outer Sunset	8.20	8.20	8.200
12.	Piccino Cafe	807 22nd St.	Dogpatch	8.10	7.80	7.950
13.	Bar Bambino	2931 16th St.	Mission	8.00	7.80	7.900
	Bittersweet - The Chocolate Café	2123 Fillmore St.	Fillmore	8.00	7.50	7.750

FIGURE 15.7 A table of cafe ratings that can be found on an HTML page.



1. Four Barrel Coffee
Category: Coffee & Tea
Neighborhood: Mission

813 reviews
375 Valencia St
San Francisco, CA 94103
(415) 252-0800

BEST coffee I've had in the city. Perfect foam and perfect balance of **espresso** to milk in their lattes. You dont need anything else-- their aromatic **espresso** and smooth steamy milk is the most satisfying wake up call

2. Caffe Pascucci
Categories: Coffee & Tea, Beer, Wine & Spirits
Neighborhood: SOMA

108 reviews
170 King St
San Francisco, CA 94107
(415) 957-1100

varieties. Particularly, I enjoy a good , well prepared **espresso** drink like a cap or latte. I see a place that looks like this and it screams "good Euro coffee drinks." Unfortunately, each **espresso** drink I have had at

3. Sightglass Coffee
Category: Coffee & Tea
Neighborhood: SOMA

298 reviews
270 7th St
San Francisco, CA 94103
(415) 861-1313

the Nicaraguan. The coffee is so smooth with a rich taste that is not over powering. Their **espresso** is also one of the best, if not the best **espresso** I have had in the city. You must also check out their pastries they...

4. Blue Bottle Coffee
Category: Coffee & Tea
Neighborhood: Hayes Valley

1251 reviews
315 Linden St
San Francisco, CA 94102
(415) 252-7535

espresso came out the baristas were looking around for him as if he just made off with their money, they looked concerned with how far he had gone and then explained to me that after the **espresso** is pulled there's a...

5. Ritual Coffee Roasters
Category: Coffee & Tea
Neighborhood: Mission

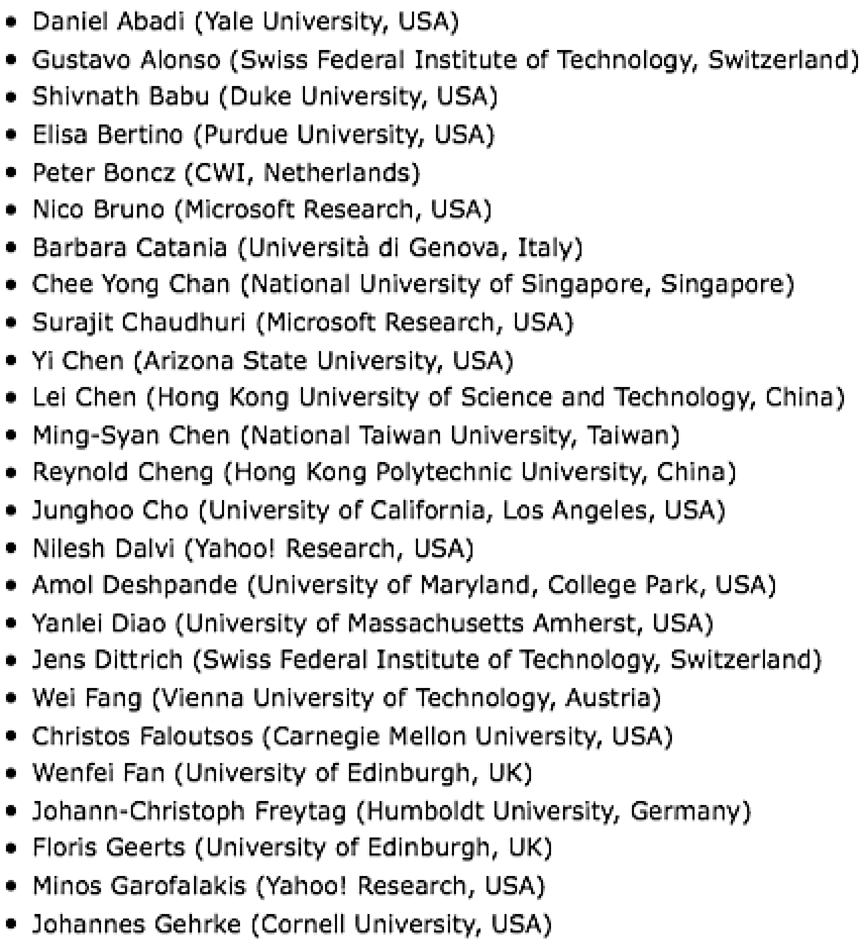
1235 reviews
1026 Valencia St
San Francisco, CA 94110
(415) 641-1011

Depending on your taste, the hipster vibe can be a plus or a minus. Personally, I've never had the baristas be anything but polite and professional. Rating based entirely on **espressos** and capuccinos--I haven't had

FIGURE 15.8 A list of cafes in San Francisco from yelp.com.

As mentioned earlier, the data integration problem is further complicated by the fact that the data are only partially structured. For example, tables, which constitute one of the main sources of structured data on the Web, have relatively little in the way of schema. We may know the headers of the columns if the first row is clearly marked (as in Figure 15.7), but often it is tricky to decide whether the first row (or rows) is a header row. Other elements of schema, such as the table name or data types, are nonexistent. The relation represented by the table is typically clear from the surrounding text to a person looking at it. Data types can sometimes be inferred by the data itself. Lists on the Web present additional challenges. As shown in Figure 15.9, every item in a list represents an entire row in a table. To convert the list into a table we need to segment each list item into multiple cells, which can be tricky. In addition, lists do not typically have a header row with attribute names.

We divide the data integration challenge into three components: locating the data on the Web, importing the data into a structured workspace, and combining data from multiple sources.



- Daniel Abadi (Yale University, USA)
- Gustavo Alonso (Swiss Federal Institute of Technology, Switzerland)
- Shivnath Babu (Duke University, USA)
- Elisa Bertino (Purdue University, USA)
- Peter Boncz (CWI, Netherlands)
- Nico Bruno (Microsoft Research, USA)
- Barbara Catania (Università di Genova, Italy)
- Chee Yong Chan (National University of Singapore, Singapore)
- Surajit Chaudhuri (Microsoft Research, USA)
- Yi Chen (Arizona State University, USA)
- Lei Chen (Hong Kong University of Science and Technology, China)
- Ming-Syan Chen (National Taiwan University, Taiwan)
- Reynold Cheng (Hong Kong Polytechnic University, China)
- Junghoo Cho (University of California, Los Angeles, USA)
- Nilesh Dalvi (Yahoo! Research, USA)
- Amol Deshpande (University of Maryland, College Park, USA)
- Yanlei Diao (University of Massachusetts Amherst, USA)
- Jens Dittrich (Swiss Federal Institute of Technology, Switzerland)
- Wei Fang (Vienna University of Technology, Austria)
- Christos Faloutsos (Carnegie Mellon University, USA)
- Wenfei Fan (University of Edinburgh, UK)
- Johann-Christoph Freytag (Humboldt University, Germany)
- Floris Geerts (University of Edinburgh, UK)
- Minos Garofalakis (Yahoo! Research, USA)
- Johannes Gehrke (Cornell University, USA)

FIGURE 15.9 An HTML list containing some of the program committee members of VLDB 2008.

15.4.1 Discovering Structured Data on the Web

The first step in data integration is to locate the relevant structured data on the Web. Data discovery typically starts by posing keyword queries, but search engines have been designed to index large collections of text files and turn out to be much less effective at searching for tables or lists. There is no way for the user to specify to the engine that she is interested in structured data. Without special support, the only option available to the user is to tediously scan the search results one by one for those containing structured data.

To support searches specifically for structured data, we need to discover which Web pages contain high-quality structured data and to mark those documents in the Web index (or create a separate repository of these documents). The discovery problem is challenging because many Web page designers use the HTML table construct to format unstructured data content nicely (e.g., forum responses, calendars, table of contents). In fact, less than 1% of the HTML tables on the Web contain high-quality tabular data.

Once we have a corpus of structured data, ranking results needs to consider *where* the keywords in the query match on the page. Two important kinds of hits are on the *header* row and on the *subject* column. Hits on the header row, which presumably includes the attribute names, should be given more weight because the attribute name applies to *all* the rows in the table. For example, if we search for "country olympic medals," then a hit on a column name olympic medals would be more meaningful than other occurrences of the phrase. The majority of tables on the Web have a subject column, which is a column that contains the entities the table is about (e.g., countries). If we search for "population zimbabwe," then a table that contains Zimbabwe in the subject column would be better than one that contains Zimbabwe elsewhere in the table. The latter table may actually be about cities and have a column for their country or may be describing different variants of corn and include a column describing in which countries they are grown. The subject column tends to be closer to the left-hand side of the table (except, of course, for tables in Hebrew and Arabic), but finding it algorithmically (when it exists) can be tricky. Needless to say, like any ranking signal, these are merely heuristics that need to be combined with other signals to arrive at a final ranking. In general, any preprocessing that can recover the semantics of the table, such as phrases describing the sets of objects represented in the table and the relationships modeled in the table will be extremely useful in ranking. In particular, the relationships modeled by a table are notoriously hard to discover. For example, a table describing the coffee production of different countries may have column headers such as name, 2001, 2002, ..., 2010. Deeper analysis is required to infer that the column 2001 describes the coffee production of the country in the name column for 2001.

15.4.2 Importing Data

Once we locate the data, we need to import them into a workspace where the data are put into tabular form and proper column headers are assigned. In some cases, such as the table shown in Figure 15.7, we may be lucky because the structure is nicely reflected in the HTML. However, consider the data shown in Figure 15.8. The data are structured but organized into cards, where each attribute appears in a particular place in the card. Extracting the different parts of each card requires knowledge of the card's internal structure. Similarly, if we import data from a list, we need to segment each list item into a row of cells. We consider two sets of methods for importing data: fully automatic and user supervised.

Fully automatic techniques for extracting data rows from cards or lists consider several aspects of the data, such as punctuation marks, changes in data type, and identifying entities that commonly appear elsewhere on the Web. Note that neither of these cues would work well in isolation, and they may not even suffice when applied together. In Figure 15.9, punctuation will enable separating the name of the program committee member from the affiliation. However, the punctuation within the affiliations is not consistent across the list items. Most of the list items have a comma separating the institution name from the country, but in some cases (e.g., University of California, Los Angeles) there is an additional comma that would confuse the importer. To resolve such ambiguities, an import algorithm can benefit from looking at the entire list as a whole. In our example, by looking at the entire list we can notice that there is always a country name before the right parenthesis, so we can use that signal to align the rest of the affiliations. Another powerful heuristic is to use the appearance of strings in table cells on the Web as a signal that these strings refer to entities. Specifically, if a particular string (e.g., University of California, Los Angeles) appears by itself in many cells of tables on the Web, that is a pretty good signal that it refers to an entity in the world. The wrapper construction techniques discussed in Chapter 9 are often applicable in this context as well.

User-supervised techniques are based on generalizing import patterns that are *demonstrated* by users and bear some resemblance to techniques for training wrappers (Chapter 9.3). Consider the data in Figure 15.8. The user would begin by selecting "Four Barrel Coffee" and pasting it into the leftmost cell of the workspace. The system would attempt to generalize from this example and propose other cafe names, such as Sightglass and Ritual. By doing so, the system will propose additional data rows. Next, the user will select the address of Four Barrel Coffee and paste it into the second column, to the right of the name. Similarly, she will copy and paste the phone number into the third column and the number of reviews into the fourth. The system will generalize from these examples and will fill in the other rows in the table with the corresponding data.

COLUMN HEADERS

After importing the data into a table, we need to propose headers for the columns of the resulting table if they do not already exist in the data. In some cases, such as phone numbers or addresses, it is possible to analyze the values in a column and propose a header. In other cases, we may consult a repository of tables that do have column headers. To find a header for a column C, we search for columns C' that have significant overlap of values with C and choose the most common column header among that set. For example, in the list shown in Figure 15.9, we can propose the column header country for the last column after noticing that it has high overlap with other columns with the same column header.

Another technique for proposing headers is to mine the Web for particular text patterns. For example, phrases of the form "countries such as France, Italy, and Spain" will appear very frequently on the Web. From this sentence we will be able to mine with high degree of confidence that France is a country. Using the same technique we will be able to derive that France is a European country and even a Western European country. If the same class (e.g., country) applies to a majority of the values in a particular column, we can propose the class as a label for the column. In general, we want to attach to the column any label that will help to retrieve it in response to relevant queries.

15.4.3 Combining Multiple Data Sets

Once we have the data, there are two challenges in combining multiple data sources. Since the collection of data sources is huge, the first challenge is to find *what* to combine your data with. Since the data are not always completely structured, the second challenge is to specify *how* to combine the data.

To find related data, we can consider two cases: adding more rows by union or adding more columns by join. To add more rows to a table T, the system needs to find tables that have the same columns as T, even if the column headers do not match up exactly. To find more columns, the system first needs to find the interesting join columns in T, and then find other tables that have a column that overlaps with the join column. For example, to find tables that join with the coffee data in Figure 15.7, the system would first postulate that the column with cafe names is a likely join column, and then search for tables that have overlapping values.

To specify how to join the data, we can employ the principle of demonstration by example again. Consider the task of joining the coffee data in Figure 15.7 with the data in Figure 15.8 after they have been imported. The user would highlight the cell with the value "Ritual Coffee" in the table created from Figure 15.7 and drag it onto the corresponding cell created for the data in Figure 15.8. By doing so, the user has told the system which columns she wishes to join. The user can also drag other columns of the row of Ritual Coffee into the appropriate row in the table to specify which columns should be kept in the joined table. The system can generalize from this example and apply the same process to other rows.

15.4.4 Reusing the Work of Others

As we consider data integration tasks on the Web, it is important to keep in mind that many people are trying to access the *same* data sources and perform similar, if not identical, tasks. If we can leverage the collective work of many people, we can develop very powerful techniques for managing structured data on the Web. For example, when a person makes the effort to extract a structured data set from a Web page and import it into a spreadsheet or database, we can use that as a signal that the data set is a valuable one and record the spreadsheet name and column headers she gave in the spreadsheet. When someone manually combines two data sets, we can infer that the two sources are related to each other and that the pair of columns used for the join are from the same domain, even if they are not identical. When someone transforms an HTML list into a table, that effort can also be recorded for future uses even if the data change a bit over time. The same transformation may be applicable to other lists on the Web with similar characteristics. Of course, the key to enabling these heuristics is to record actions across a large number of users, while being sensitive to privacy concerns. Cloud-based tools for data management have the potential for providing such services since they log the activities of many users.

15.5 Pay-as-You-Go Data Management

Data integration on the Web is an extreme example motivating pay-as-you-go data management. In contrast to the kinds of data integration systems described thus far in the book, pay-as-you-go systems try to avoid the need for the initial setup phase that includes creating the mediated schema and source descriptions. Instead, the goal is to offer a system that provides useful services on a collection of heterogeneous data with very little up-front effort. The following example illustrates a different kind of motivating scenario.

Example 15.2

Consider an example of a nonprofit organization trying to collect data about the world's water resources. Initially, the engineers at the organization will be handed a collection of data sources (typically in the form of spreadsheets or CSV files). There are several challenges they need to face. First, they will be unfamiliar with the data. They will not know all the terminology used to collect data about water, nor will they know the contents of the data sources and the relationships between them. Second, some of the sources may be redundant, out of date, or subsumed by others and therefore not of much use.

Hence, the first step in creating a data integration application is to explore the data in a rather unstructured fashion. The mediated schema can only be created after they have some familiarity with the data. They will only create semantic mappings to the select set of data sources they deem useful. Of course, they will also spend significant effort to clean the data and reconcile references where needed.

The above scenario is very typical in domains where data have been collected by independent experts who want to start collaborating. The key observation is that requiring the engineers to put all the effort of setting up the data integration system up front may not be practical or even possible. Instead, the system should enable the engineers to only invest the effort where the return on the investment is clear, while still providing value even when all the sources are not integrated.

Dataspace systems have been proposed as a concept for supporting pay-as-you-go data management systems. Broadly, there are two main components to such a system:

bootstrapping the system with useful services with no or little human intervention and guiding the user to improve the semantic coherence over time.

For the first component, providing keyword search over a collection of data coupled with effective data visualization is already a useful step. Going further, dataspace systems can employ the techniques we described for automatic schema matching, clustering of data sources, and automatic extraction of metadata.

For the second component the challenge is to channel the attention of the users to opportunities for improving the metadata in the system. Improvements to metadata could be to validate schema mappings, spend some effort on reference reconciliation, or improve the results of information extraction. As we described earlier, reusing the work of others and crowdsourcing some tasks are two powerful methods that can be employed here. Some of these ideas are discussed in the next chapter.

Bibliographic Notes

The Web has been one of the main driving forces behind research on data integration. In fact, the inspiration behind the initial work of the authors of this book was Web data integration [181, 323, 381]. A survey on data integration and management in the early days of the Web is given in [229]. Some of the early research prototypes that attempted to integrate information from deep-Web sources are [37, 199, 244, 361, 381], and more recent work in this area is described in [294, 296, 297, 581]. Junglee and Netbot were two early startups in the area of vertical search, focusing on job search and integrating data from shopping sites. Later, vertical-search engines appeared in other commercially viable domains, such as airline tickets and other categories of classified ads.

Several papers have offered estimates of the size of the deep Web [70, 409]. In [163] the authors describe methods for estimating the size of a given database on the hidden Web. The work we described on surfacing the deep Web is based on the surfacing in the Google search engine [410]. According to [410], their system was able to surface pages from millions of forms, from hundreds of domains, and in over 40 languages. The pages it surfaced were served in the top-10 results for over 1000 queries per second on the Google search engine at the time of writing. One of the main lessons learned from that work is that deep-Web content was especially helpful for long-tail queries rather than the most popular ones, for which there was sufficient data on the surface Web. Earlier work on iterative probing, one of the techniques used for surfacing Web pages, includes [55, 106, 321, 467]. Relatedly, in [550] the authors describe a technique that takes text as input and decides how to use the text to fill the fields of an HTML form.

The investigation of tables on the Web began with the works of [247, 565]. The first investigation of the entire collection of HTML tables on the Web was described in [101, 102]. That work showed that there are over 150 million high-quality HTML tables on the Web, even when restricting attention content to English. The authors collected these tables into a corpus and built a search engine that incorporated some of the ranking methods we described in this chapter. They also showed that the collection of schemas of