RAW: A Relational Algebra for the Web

Thorsten Fiebig *Jürgen Weiss* *Guido Moerkotte*

Lehrstuhl für Informatik III
Fakultät für Mathematik und Informatik
Universität Mannheim
Seminargebäude A5
68131 Mannheim
Germany
email: *thorsten/weiss/moer*@pi3.informatik.uni-mannheim.de
web: http://pi3.informatik.uni-mannheim.de

Abstract

The main idea underlying the paper is to extend the relational algebra such that it becomes possible to process queries against the World-Wide Web. These extensions are minor in that we tried to keep them at the domain level. Additionally to the known domains (int, bool, float, string), we introduce three new domains to deal with URLs, html-documents or fragments thereof, and path expressions. Over these domains we define several functions that are accessible from the algebra within the subscripts of the relational operators. The approach allows us to reuse the operators of the relational algebra without major modifications. Indead, the only extension necessary is the introduction of a map operator. Further, two modifications to the scan and the indexscan are necessary. Finally, the indexscan which has the functionality of a typical meta-search engine is capable of computing a unified rank based on the tuple order provided by the underlying search engines.

1 Introduction

The Web [2] with its myriad of pages is a predestined target for the design of query languages dealing with unstructured data. Several such query languages are afloat already [9, 10, 11]. Hence, we do not (yet) give birth to (yet) another one. Instead, we take a look at what could be underneath. Adhering to the design principle *start simple, check expressiveness, expand when necessary*, we develop a core relational algebra to which the core of any of these query languages for the Web can be translated. This gives us the Relational Algebra for the Web, RAW for short. RAW is a slight extension of the common Relational Algebra. It introduces some new Web-specific domains and the map operator. For that, we do not need to model (relational or otherwise) the Web. This contrasts other approaches [11].

Due to our design principle, we could not use any more fanciful starting points like object-oriented algebras, but the future might reveal via the expressiveness check that relations are too restricted. So let us take the relations for granted and continue the RAW introduction.

The main idea behind any relational algebra in general and RAW in particular is that it consists of a set of algebraic operators with relations going in and out. And here already is the first important point of RAW: relations go *in*. Consider for example the request for

research documents containing "Data Warehousing".

It seems obvious that those documents which do so and are referenced by one of our favorite database research groups will be much more valuable than others. However, such a restriction is not easily expressed nowadays. Simply put, AltaVista cannot take advantage of our well-organized Netscape bookmarks. Not so with RAW. If we organize our hotlist as a relation or a set of relations (and even this is not necessary as we will see), the hotlist can readily be exploited by a RAW expression taking the relation(s) (or even our Netscape bookmarks) as input.

Sometimes, we have a request and do not know any good starting points. Then, we go to a search engine like AltaVista [6] or HotBot [5]. It is well-known, however that not every search engine suits every request and, even worse, not every search engine contains all the web pages of the world. The obvious solution is to go to a meta-search engine. But then, the retrieval language becomes very restricted [3]. Further, some search engines nicely rank or order their output in a user-useful way. This ranking or ordering gets lost via all existing meta-search engines. Both these disadvantages are solved via RAW operators. The RAW index scan (RAW can be seen as a physical algebra) provides a customizable ranking function integrating rank function that allows to order the output of this meta-search-engine-like relational algebraic operator. Subsequent RAW operators like selections provide for arbitrarily complex search predicates. These were the next two points for RAW.

The last point gives a more technical reason why we started by developing RAW and not a query language for the web. An algebra is easier to design, implement, evaluate and extend than a query language together with the necessary query optimizer and execution engine.

The rest of the paper is organized as follows. In section 2 we introduce the new Web-specific domains. In section 3 we describe the operator set of RAW and how RAW could be used to query the Web. Additionally we give a sketch of the implementation. The topic of section 4 is the integration of ranking results of different search engines. The last section contains our conclusions.

2 RAW's Relational Model

A relational schema $\mathcal{R}$ is a finite set of Attribute names $\{A_1, \ldots, A_n\}$. With each Attribute A_i a domain D_i is associated. A Relation R for a schema $\mathcal{R} = \{A_1, \ldots, A_n\}$ is a subset $R \subseteq D_1 \times \cdots \times D_n$. This is standard [1]. The only new things in RAW's relational model are new domains. Apart from the standard domains (int, float, string, bool) RAW features the following domains:

1. HTTP-Address
2. HTTP-Address-Path
3. HTML-Document-Path

These domains come with certain functions the same way as int comes with "+".

A HTTP-Address is an universal resource identifier (URI) as defined in [7]. The functions on this domain are, *get_accessmethod*, *get_servername*, *get_port*, *get_address*, *get_addresspath* and *get_filename*. They retrieve the according part of the HTTP-Address and return it as a string. Further functions concern the document referenced by a HTTP-Address. They are:

- *get_type*: HTTP-Address $\rightarrow$ String
  returns the mime type of the referenced document as a string,
- *get_size*: HTTP-Address $\rightarrow$ int
  returns the size of the referenced document,
- *contains*: HTTP-Address, String $\rightarrow$ Bool
  with an additional string parameter returns a bool indicating whether the string occurs in the document.

A HTTP-Address-Path consists of an alternating sequence of HTTP-Addresses and strings indicating the description of the reference. For example, if we have an HTTP-Address X, and the document referenced by X contains a link *<A HREF = Y>click here</A>*, then

X $\underline{\text{click here}}$ Y

is an HTTP-Address-Path. Again, this domain comes with plenty of functions like

- *length*: HTTP-Address-Path $\rightarrow$ int
  returns the length of the path,
- *append*: HTTP-Address-Path, HTTP-Address-Path $\rightarrow$ HTTP-Address-Path
  appends two paths,

- *contains*: HTTP-Address-Path, String → bool
  checks for the occurrence of a string within the underlined part of an HTTP-Address-Path,
- *first*: HTTP-Address-Path → HTTP-Address
  returns the first HTTP-Address in the path and
- *last*: HTTP-Address-Path → HTTP-Address
  returns the last HTTP-Address in the path.

We assume a HTML-document as a collection of HTML-fragments. Each fragment in a HTML-document could be addressed by a HTML-Document-Path. It describes the path leading to a HTML-fragment by a sequence of HTML-labels. For example,

<HTML> <HEAD> <TITLE>

describes the way to get to the title of a HTML-document. Some functions on this domain are

- *contains*: String, HTML-Path, String → Bool
  checks whether a String, usually representing a HTML-fragment, contains the given HTML-Path leading to a fragment which contains the last String,
- *contains*: HTTP-Address, HTML-Path,String → Bool
  checks whether a HTML-document contains the given path, leading to a HTML-fragment which contains the given String,
- *get_content*: HTTP-Address, HTML-Path → {String}
  returns a set of strings representing the HTML-fragments specified by the HTTP-Address and the HTML-Path,
- *get_content*: String, HTML-Path → {String}
  returns a set of strings representing the HTML-fragments specified by the HTML-Path.

Note that the last two functions return a set of strings. We will have to say more about it in the next section.

3 RAW

3.1 The Algebraic Operators

As the common Relational Algebra RAW consists of a set of operations. The operator set contains the standard set operators union, intersection, set-difference. Further, it contains the standard relational algebra operators. Let us denote the type of relation with schema $\{A_1, \ldots, A_n\}$ by $\{[A_1 : \tau_1, \ldots, A_n : \tau_n]\}$. Each Attribute of the relation is represented by its name A_i and its domain type τ_i.

- $SELECT_P$: $\{[A_1 : \tau_i, \ldots, A_n : \tau_n]\} \rightarrow \{[A_1 : \tau_1, \ldots, A_n : \tau_n]\}$,
  where $P : [A_1 : \tau_1, \ldots, A_n : \tau_n] \rightarrow$ *Boolean* is the selection predicate
- $PROJECT_{\{A_i:\tau_i,\ldots,A_j:\tau_j\}}$:$\{[A_1 : \tau_1, \ldots, A_n : \tau_n]\} \rightarrow \{[A_i : \tau_i, \ldots, A_j : \tau_j]\}$,
  where $\{A_i : \tau_i, \ldots, A_j : \tau_j\} \subseteq \{A_1 : \tau_1, \ldots, A_n : \tau_n\}$
- $JOIN_P$:$\{[A_1 : \tau_{A_1}, \ldots, A_n : \tau_{A_n}]\}, \{[B_1 : \tau_{B_1}, \ldots B_n : \tau_{B_n}]\} \rightarrow \{[A_n : \tau_{A_1}, \ldots, A_n : \tau_{A_n}, B_1 : \tau_{B_1}, \ldots, B_n : \tau_{B_n}]\}$,
  where $P : [A_1 : \tau_{A_1}, \ldots, A_n : \tau_{A_n}], [B_1 : \tau_{B_1}, \ldots B_n : \tau B_n] \rightarrow$ *Boolean* is the join predicate

As selection and join predicates, RAW accepts arbitrary boolean expressions possibly contain functions for the new data types introduced above. Additionally the *MAP* operator belongs to the operator set of RAW.

$MAP_{f,B}$:$\{[A_1 : \tau_{A_1}, \ldots, A_n : \tau_{A_n}]\} \rightarrow \{[A_1 : \tau_{A_1}, \ldots, A_n : \tau_{A_n}, B : \tau]\}$
f:$[A_1 : \tau_{A_1}, \ldots, A_n : \tau_{A_n}] \rightarrow \tau$

For each input tuple the map operator extends the input relation by the attribute B, that is the result of the evaluation of the function f found in the subscript. Possibly the result of function f is a set, in this case the operator generates one output tuple for each element in the set or in case of multiple set valued expressions, one output tuple for each combination.

So far, we are not able to follow paths that are not pre-specified. To resolve this, RAW was designed as a physical algebra. A typical physical relational algebra further contains the operators *SCAN* and *INDEX-SCAN* also present in RAW.

$SCAN_{[e,P,A]}$:$\{[A_1 : \tau_1, \ldots, A_n : \tau_n]\} \rightarrow \{[A_1 : \tau_1 \ldots, A_n : \tau_n, A : \tau]\}$
where τ = HTTP-Address-Path

The *SCAN* operator takes an input relation and the following further parameters. First, it takes an expression e that results in a set of strings. These strings are searched for all occurring HTML-Addresses. These are dereferenced by the *SCAN*. All HTML-Addresses that are contained in the referenced documents are again extracted and dereferenced and so on. The input relation is expanded by an additional attribute A (third parameter). The domain of A is HTTP-Address-Path. It contains all the paths that could be build by this extract/dereference process. More specifically, if t is an input tuple and $e(t)$ results in a string s containing the set of HTTP-Addresses H, *SCAN* first computes the set of paths *PSET* starting from addresses in H. For each path *PATH* in *PSET*, the tuple t is expanded by a new attribute whose value is *PATH*. Since there could be too many paths, the *SCAN* operator also takes predicates to restrict the output. For example, the length of the resulting paths can be restricted. Any other selection predicate that can be build with the special domain functions is accepted by the *SCAN* operator. This selection predicate P is its second parameter.

Let us illustrate the *SCAN* by an example. We are looking for all documents containing the string “Data Warehousing” that are referenced from our favorite database research group. Assume that a relation *FavDBResG* contains our favorite database research groups and has two attributes NAME and ADDRESS. The latter is assumed to be the HTTP-Address. Then, the query can be answered by

```
SCAN[get_address(ADDRESS),
    (contains(last(X),“Data Warehouse”) && length(X) < 3), RES]
        ( FavDBResG )
```

where we restricted the path length to at most 2. The predicate can have at most one free variable which is bound to the paths found by the extract/dereference process.

The RAW index-scan has the following signature.

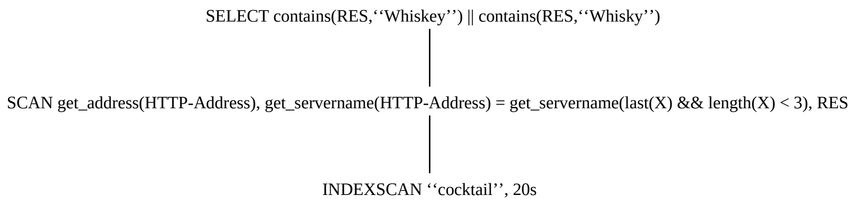
INDEXSCAN: *SString*, *Timeout*→ {[*Address*, *Ranking*, *SearchEngines*, *Title*]}

It takes a search string *SString* and a time-out *Timeout* as input and produces an output relation with the four attributes

1. *Address*, containing the HTTP-address of a resulting document,
2. *Ranking*, the rank value computed by *INDEXSCAN* (see section 4),
3. *SearchEngines*, a string containing the search engines that gave the HTTP-Address as a result. The names of the different engines are concatenated by “++”,
4. *Title*, containing the title of the document as a string,

3.2 More Sophisticated Examples

In this section we present some sophisticated query examples and their evaluation to illustrate the expressiveness of RAW. First, we are looking for cocktails that contain whiskey. This query is answered by the following RAW expression featuring an index-scan:

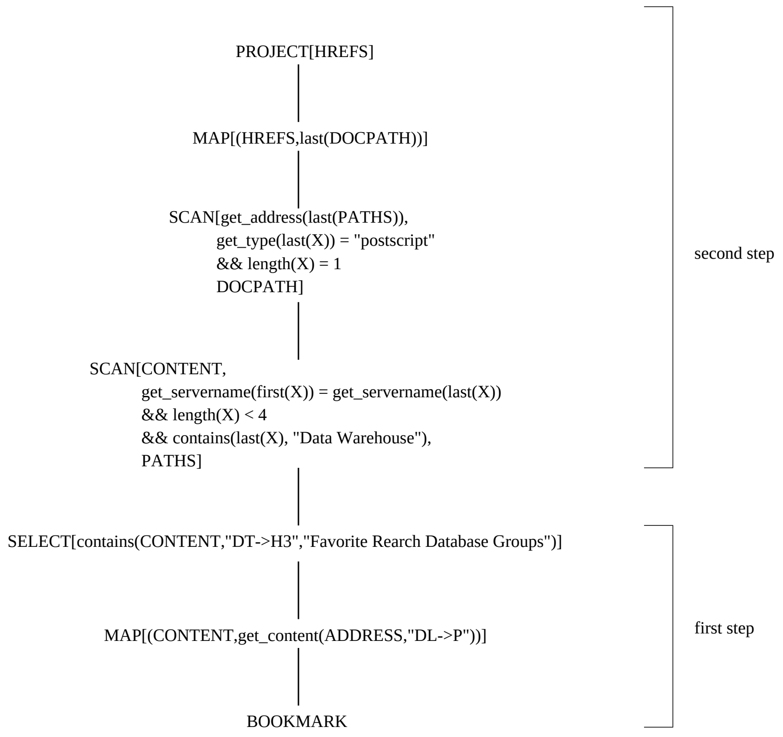

SELECT contains(RES,‘‘Whiskey’’) || contains(RES,‘‘Whisky’’)
SCAN get_address(HTTP-Address), get_servername(HTTP-Address) = get_servername(last(X) && length(X) < 3), RES
INDEXSCAN ‘‘cocktail’’, 20s

We first search for documents that contain the string “cocktail”. Afterwards we look for documents that are referenced by these and are on the same server. Last, we select those that contain either “whiskey” or “whisky”. As a time limit we gave 20 seconds. Already with this small time limit, the result contained 285 tuples.

Second, let us assume that we are interested in postscript documents which are referenced by the HTML-documents referenced by our Netscape bookmarks. As a further constraint we only want to look at postscript documents concerning “Data Warehouse”. The starting point of our query evaluation is the bookmark folder titled “Favorite Database Research Groups”. The evaluation of this query could be done in two steps:

1. extraction of the interesting bookmarks
2. finding the postscript documents

The following RAW-expressions answers our query.


PROJECT[HREFS]
MAP[(HREFS,last(DOCPATH))]
SCAN[get_address(last(PATHS)), get_type(last(X)) = "postscript" && length(X) = 1 DOCPATH]
SCAN[CONTENT, get_servername(first(X)) = get_servername(last(X)) && length(X) < 4 && contains(last(X), "Data Warehouse"), PATHS]
SELECT[contains(CONTENT,"DT->H3","Favorite Rearch Database Groups")]
MAP[(CONTENT,get_content(ADDRESS,"DL->P"))]
BOOKMARK
second step
first step

The query evaluation starts with the relation BOOKMARK, which only consists of the attribute ADDRESS. The relation contains only one URI, leading to our Netscape bookmark file, which is a HTML-document. It consists of a html definition list. The list contains single bookmarks and html definition lists representing a bookmark folder. Every bookmark and every folder has a title. The map operator extends our relation by the attribute CONTENT. The new attribute consists of a string representing a HTML-fragment. The fragment represents the content of a bookmark folder in the bookmark file. The selection of the interesting folder is done via the *contains* function.

The second step starts with a scan operation, which searches for HTML-documents that contains the phrase “Data Warehouse”. The input relation is extended by the valid paths expressions. These paths expressions are the starting point for the scan operation that searches for the postscript documents. The following map operation adds the HREFS attribute, which contains the URIs of the found postscript documents. The result of the final project operation is a relation with HTTP-Addresses of postscript documents.

3.3 The Implementation

Relations are represented as ASCII files. One line represents one tuple and different attributes are separated by an arbitrary separator character whose default is “|”. RAW itself is implemented as a bunch of programs in Perl and C++. Each program implements exactly one RAW operator. The communication between the operators is done via temporary files or (named) pipes. For the implementation of the scan operator the script w3mir[1] is used. Since plenty of the functions concerning the domain HTTP-Address refer to the actual document referenced, a buffer with fetched documents is managed by the RAW system. Another part of the implementation is concerned with the different errors that can occur. It carefully analyses the HTTP status codes and takes according actions. Analogous to the cache for successfully fetched documents, a bad-list for dangling references is managed. The technical details of the implementation go beyond the scope of this paper.

4 Ranking

As already mentioned, the index-scan of the RAW algebra provides for a unifying customizable ranking function to integrate the different orders produced by search engines. Hence, we must revise our data model first. Instead of relations being sets of tuples, we use ordered sets. For the development of the unifying customizable rank function, several facts discourage the integration of the ranks of search engines [8]:

- not every search engine provides an explicit rank, some do not even rank the documents
- different search engines use different ranking algorithms
- the ranking algorithms of the search engines are not known,
- the result of the ranking is represented differently, and
- different search engines have different document bases.

In this context it does not make much sense to explore the ranking values given by some search engines. Instead, we explore only the relative document orders provided by the search engines.

We think that the following factors are relevant for ordering a set of documents retrieved by several search engines:

1. the number of referencing search engines,
2. the rank of the reference in the query result, and
3. the quality of the referencing search engine.

[1]w3mir was originally developed by Oscar Nierstrasz (for further information see http://iamwww.unibe.ch:80/ scg/Src/Doc/htgrep.html)

The quality of the search engines determines the influence of the ranking of a particular search engine on the resulting ranking. Note that the last parameter might be query specific. Hence, this gives the first motivation for a customizable rank integrating function. Accordingly, the quality of the search engines can be specified either by the user or by the system. The specification by the system depends on the size of the search engines document base and on the number of relevant documents, i.e. the size of the query result.

Combining these factors we build a rank integration function, which merges the query results of different search engines. Therefor, we assume that every engine i returns an ordered list of references, in which the reference order corresponds to the document ranking. For every result list we build a tuple $(v_{i,j}, r_{d_j,i})$ for each reference occurring in the list where

- $v_{i,j}$ is a reference of document d_j by the search engine i.
- $r_{d_j,i}$ corresponds to the order of the reference in the query result
  (The first reference gets value 1, the second value 2 and so on.)

We start by eliminating those tuples which have the same value for v_j in all lists but keep the tuple with the lowest value of $r_{d_j,i}$ [2]. Then, we generate a tuple (v_{d_j}, r_{d_j}) for every reference. Again, v_{d_j} is a reference to a document d_j, and r_{d_j} is the now integrated new rank value. The integrated rank r_{d_j} is determined by the following equation:

$$r_{d_j} = \sum_{i=1}^{k} w_i \ r_{d_j,i}$$

where k is the total number of search engines and the weight factor w_i specifies the quality of the search engines. If this weight is not provided by the user, the system will use the the following equation:

$$w_i = \frac{1}{D_i \ |E_i|} \quad , D_i, |E_i| > 0$$

where D_i represents the number of documents in the document base of engine i and $|E_i|$ is the number of different references in the query result. We construct our final query result by using the tuples (v_{d_j}, r_{d_j}), sorted by the value of r_{d_j}. The document referenced by the tuple with the lowest value of r_{d_j} is assumed to be the best.

5 Conclusions and Future Steps

Although RAW already gives nice features and some expressiveness to evaluate queries against the Web, there are some disadvantages in its expressiveness. For example, the structure of HTML-documents cannot be exploited in a nice way. Simple path through the document are the only means provided by RAW. An object-oriented modeling of HTML-documents would allow much better queries [12]. For example, one could ask for lists with at least three entries. This is not possible within RAW. Hence, the question arises whether the domains should be further expanded or whether an algebra supporting the object-oriented data model should be applied. Future research will answer this question. Further points for future research are

1. It should be analyzed more carefully how far the existing query languages can be translated to RAW and how far RAW should be extended accordingly.
2. Optimization issues which are at the core of any algebraic approach should be investigated. A good starting point is [4].
3. The rank integrating function should be evaluated and compared with other (to be developed) rank integrating functions.

Finally, we will not be able to resist the temptation to design a query language that embraces the full features of RAW.

[2]We experimented with different rank functions and found this one the most useful.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1995.

[2] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Comm. of the ACM*, 37(8):76–82, August 1994.

[3] Paul De Bra. Finding Information on the Web. *CWI quaterly*, 8(4):289–306, December 1995.

[4] Kevin Chen-Chuan, Héctor García-Molina, and Andreas Paepcke. Boolean Query Mapping Across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, August 1996.

[5] Inktomi Cooperation. HotBot (tm). http://www.hotbot.com.

[6] Digital Equipment. AltaVista (tm) Search. http://www.altavista.digital.com.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, January 1997. http://www.ics.uci.edu/pub/ietf/http.

[8] Luis Gravano, Chen-Chuan K. Chang, Héctor García Molina, and Andreas Paepcke. STARTS: Stanford Proposal for Internet Meta-Searching. In *Procceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[9] David Konopnicki and Oded Shmueli. W3QS: A Query System for the World-Wide Web. In *Proceedings of the 21. VLDB Conference*, 1995.

[10] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proc. of the 6th. International Worshop on Research Issues in Data Engineering, RIDE '96*, February 1996.

[11] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proc. PDIS '96*, December 1996.

[12] Marc Volz, Karl Aberer, and Klemens Böhm. Applying a Flexible OODBMS-IRS Coupling for Structured Document Handling. In *Proc. of the 12. International Conference on Data Engineering (ICDE)*, pages 10–19, New Orleans, Louisiana, 1996.