The Jungle Database Search Engine

Michael Böhlen boehlen@cs.auc.dk
Linas Bukauskas linb@cs.auc.dk
Curtis Dyreson curtis@cs.auc.dk
Dept. of Comp. Science, Aalborg University Frederik Bajers Vej 7E, 9220 Aalborg st, Denmark

Abstract

Information spread in in databases cannot be found by current search engines. A database search engine is capable to access and advertise database on the WWW. Jungle is a database search engine prototype developed at Aalborg University. Operating through JDBC connections to remote databases, Jungle extracts and indexes database data and meta-data, building a data store of database information. This information is used to evaluate and optimize queries in the AQUA query language. AQUA is a natural and intuitive database query language that helps users to search for information without knowing how that information is structured. This paper gives an overview of AQUA and describes the implementation of Jungle.

1 Introduction

A recent study estimates that 80% of the data on the WWW is in the *hidden web* [LG98]. The hidden web refers to information that can be accessed on the WWW, but which current search engines cannot find (nor can the internet users who subsequently use those search engines). A major part of the hidden web consists of information tucked away in databases.

A database offers many advantages for maintaining a large information repository. Unfortunately, a database is a closed box to current search engines. Existing search engines are limited to searching the web of HTML text documents. There is no protocol to permit a search engine to look inside a database and index the information it finds. This means that search engines miss a lot of important information that is available on the WWW.

The ultimate goal of this project is to open a database to search engines. We call this opening of a database to external inspection *database advertising*. In database advertising, a database designer chooses to advertise some or all of the information in a database, making it available to search engines.

Currently the only way to advertise a database is to *publish* it in HTML. To publish a database, data is dumped into an HTML page, which a search engine subsequently indexes. Unfortunately, HTML is a very poor intermediary between a database and a search engine. HTML is a language to format text for presentation, not a data exchange language. Once data is put into HTML, it loses the structure that it had in the database. Indications are that XML will be a better data exchange language, however, there are many other problems to advertising in this manner.

A fundamental problem with publishing is that information in a database is highly *inter-related*. For example, in a library database, information about Niklaus Wirth may be spread among a number of stored tables. A database query dynamically constructs transient *relationships* between individual pieces of data. So a user could query for the books authored by Niklaus Wirth, or the books about Niklaus Wirth, or the publishers of books by Niklaus Wirth, etc. The number of potential, sensible, ordinary queries is quite large. Should all such queries be tried and the results published in HTML pages? Should only a few? If so, how few and how does a database advertiser determine which queries in advance? Our position is that it is unreasonable and far too expensive to publish the results of an exhaustive set of database queries.

In the rest of this paper we describe the implementation of the Jungle database advertiser. Jungle has a simple, yet usable query language, called AQUA, which is introduced in Section 2. Section 3 gives a detailed discussion of Jungle's architecture. We learned some important lessons in implementing Jungle which we summarize in Section 4. A short summary concludes this paper.

Jungle is currently available on the World-Wide Web at `http://www.cs.auc.dk/~linb/aqua`. All AQUA queries mentioned in the paper can be tried at the site.

2 AQUA

2.1 Vertical queries

A database can be thought of as having a hierarchy of levels, from the database, to tables within the database, to columns within a table, to individual attribute values within the columns. A user can look for databases about libraries as follows.

```
DB: "library"
```

AQUA interprets this query as a search for all databases described by the string "library". This can be the name of the database, or additional text comments about the database (some DBMS support text comments, and JDBC can extract these comments). The query returns the handle of each database object found.

The other levels (table, column, and value) can be queried in an analogous manner. So

```
TAB: "publication"
```

would find all the tables that are described by the string "publication", whereas

```
VAL: "Java"
```

will find all the tuples that contain the string "Java" in some attribute value.

The user can then narrow the search space to publication tables within the library databases as follows.

```
DB: "library" TAB: "publication"
```

Levels can be skipped in a vertical query, for instance the following query would obtain author values from any table in any database.

```
COL: "author" VAL: ""
```

In the above example, observe the difference between the levels higher than the specified `COL:` level and the levels lower than the `COL:` level. Higher levels can either be omitted or an empty string can be specified (`DB: ""`). There's no semantic difference. The reason is that a column can only be identified with respect to a database. Thus, the database has to be identified in any case. The situation is quite different for the value level. Omitting `VAL: ""` changes the semantics of the query. Specifically, no tuples will be returned.

2.2 Horizontal queries

One of the most useful features of AQUA is the ability to automatically join related information. AQUA users are naive, and will not know how the underlying databases structure their information. For instance a user looking for a book by Niklaus Wirth will usually not know, nor care, that this information is split into several tables in the Library database. AQUA, however, has enough intelligence to automatically construct this relationship during a horizontal query.

The relationships are derived from the imported and exported key information in the underlying databases. Conceptually, keys record the basic set of relationships between different tables in a database. AQUA uses the keyword `AND` to join information at the same level between tables in a horizontal query. For example, the relationship(s) of (Niklaus) Wirth to Java can be found as follows.

```
VAL: "Wirth" AND VAL: "Java"
```

3 Jungle Architecture

The Jungle prototype consists of a *Robot* that visits each remote database to populate Jungle's data store, and the AQUA *query evaluation engine.*

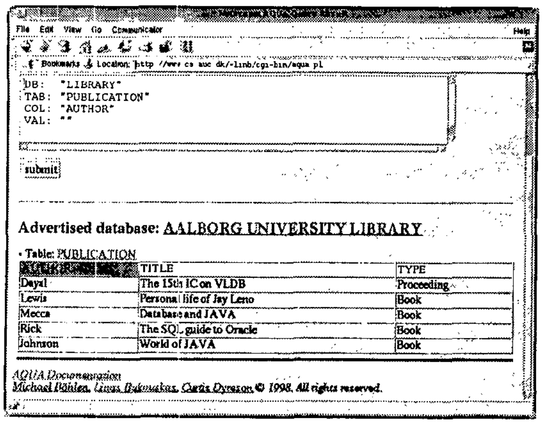
3.1 Robot

Jungle has a minimal amount of configuration. Basically, the only configuration information that Jungle requires is the location of each database's JDBC server, and a login name and password for that database. Jungle's robot then visits each remote database and performs the following steps to extract and index the information it finds.

First, the *schema grabber* lifts the *schema*, that is, the names of tables, the names of columns, any additional table or column descriptions such as synonyms (supported in some databases), column types, and the names of domains. The meta-data information is indexed. The meta-data index maps strings to database, table, column, or domain objects. The index is used in the evaluation of single-level and vertical AQUA queries to locate the appropriate object. Since it stores only meta-data, this index is relatively small.

Next, using the meta-data, Jungle generates and executes a sequence of SQL queries via JDBC to extract the values in *text* and *numeric* columns from each table. This data is also indexed. The data index maps strings or numbers to tuple objects. The data index is relatively large, but its only role is to improve performance, so that value-level queries can quickly identify which tuples contain relevant information.

Jungle applies different indexing techniques for text and numeric data. AQUA uses substring search inside a text field, so strings are indexed by decomposing (tokenizing) the string into a list of words. Only the first n letters of every word are stored to save space. In addition, all indexed words are translated to uppercase for case-insensitive searching. Numeric data is indexed by storing the minimal and maximal values from an entire column. This method could be improved since some numeric values occur more frequently than others, and ranges of values are common. So storing


File Edit View Go Communicator Help

Bookmarks Location: http //www cs auc dk/-linb/cgi-bin/aqua pl

```
DB:  "LIBRARY"
TAB: "PUBLICATION"
COL: "AUTHOR"
VAL: ""
```

submit

Advertised database: AALBORG UNIVERSITY LIBRARY

• Table: PUBLICATION

[illegible]	TITLE	TYPE
Dayal	The 15th ICon VLDB	Proceeding
Lewis	Personal life of Jay Leno	Book
Mecca	Database and JAVA	Book
Rick	The SQL guide to Oracle	Book
Johnson	World of JAVA	Book

AQUA Documentation
Michael Böhlen, Linas Bukauskas, Curtis Dyreson © 1998. All rights reserved.

Figure 1: Authors of publications in a library database

a distribution or a range of known values may improve query optimization.

Finally, *imported/exported* key information, if available, is retrieved. This key information will be used by AQUA to construct horizontal relationships among tuples in an underlying database.

3.2 AQUA query evaluation engine

The most complicated part of Jungle is the AQUA query evaluation engine. An AQUA query is a query for information in some advertised database. Some of the queries (e.g., single or vertical queries on the meta-data) can be answered directly from the information in Jungle's data store, but in general, Jungle (intelligently) converts an AQUA query to a set of SQL queries, has the queries evaluated by the appropriate remote database System, and collects and formats the results.

1. The AQUA **parser** is a recursive descent parser automatically generated from a BNF specification by JavaCup. The parser builds a parse tree of data extraction operations (e.g., database-level-search("library")).
2. The **plan generator** converts the data extraction operations to a set of SQL queries on the advertised databases. The SQL is generated using the meta-data and data indexes described in the previous section. For example, a horizontal value-level query will result in SQL queries to join related tables.
3. The plan **optimizer** determines those queries that can be answered without querying the underlying database (e.g., meta-data queries). Such queries have their results sent directly to the **HTML formatter**. The data index can be used to further prune unnecessary joins from the query evaluation plan.
4. The query is evaluated by opening a remote connection, feeding the generated SQL to the connected database, and collecting the results.
5. The **HTML formatter** formats the output in HTML as illustrated in Figure 1. A fixed HTML format is currently the only choice for reporting results. In future, we plan to add support for formatting with user-supplied templates.

4 Lessons learned

Jungle demos the feasibility of database search engines. Throughout the project we got many insights of which we would like to share the most important areas.

Value-level indexes A standard feature of every DBMS should be a value-level index that maps a string to the location (table, column, and/or tuple) of every value in the database that matches the string. This would help to create a new class of "value-added" database services such as database advertisers. Jungle currently has to build a value-level index on top of a database by *extracting* the relevant data from the database, thereby duplicating a part of the data in a database.

Database APIs Constructing this prototype would have been impossible without a standardized interface to database services. We found JDBC to be a very good API for our purposes, especially for extracting meta-data [HCF97]. The JDBC server technology was easy to use and provides an adequate level of security.

Database browsing It is important to combine declarative and browsing-based query languages. The Jungle search engine does do by integrating AQUA with the possibility for interactive heterogeneous [LSS93] database browsing [Mot86]. The result of queries is enriched with HTML links with encapsulated calls to the search engine. As shown in Figure 1 advertised database and table names are associated with an HTML link to support intuitive and ease-to-use browsing.

5 Summary

The paper describes the Jungle database search engine. Jungle allows users to query databases using AQUA. Vertical and horizontal AQUA queries allow to query advertised databases without knowing the underlying databases structures. We described the main components of the Jungle architecture, its robot, search engine, and data store. We described the main steps of the database advertisement and query evaluation phases.

References

[HCF97] G. Hamilton at al. *JDBC Database Access with JAVA: A Tutorial and Annotated Reference.* Addison Wesley, 1997.

[LG98] S. Lavrence et al. Searching the world wide web. *Science*, 280(4):98–100, 1998.

[LSS93] L. Lakshmanan et al. On the logical of schema integration and evolution in heterogenous database systems. In *DOOD'93*, December 1993.

[Mot86] A. Motro. Baroque: A browser for relational databases. In *ACM Trans.] on Office Information Systems*, volume 4, pages 164–181, 1986.