proSQLite: Prolog file based databases via an *SQLite* interface

Sander Canisius[1], Nicos Angelopoulos[1,2], and Lodewyk Wessels[1,2]

[1] Bioinformatics and Statistics, Netherlands Cancer Institute, Amsterdam, Netherlands
[2] The Netherlands Consortium for Systems Biology (NCSB)
{s.canisius,n.angelopoulos}@nki.nl

Abstract. We present a succinct yet powerful interface library to the *SQLite* database system. The single file, server-less approach of *SQLite* along with the natural integration of relational data within Prolog, render the library a useful addition to the existing database libraries in modern open-source engines. We detail the architecture and predicates of the library and provide example deployment scenarios. A simple bioinformatics example is presented throughout to illustrate *proSQLite*'s main functions. Finally, this paper discusses the strengths of the system and highlights possible extensions.

Keywords: databases, *SQL*, Prolog libraries, *SQLite*.

1 Introduction

SQLite [1] is a powerful, open source server-less database management system that requires no configuration as its databases are stored in a single file. Ran from a lightweight operating system (OS) library executable, it can be deployed in a number of scenarios where a traditional server-client database management system (DBMS) is not possible, advisable or necessary. This paper presents an implementation of a Prolog library that uses the C-interface to communicate with the *SQLite* OS library.

The relational nature of Prolog makes its co-habitation with relational database systems an attractive proposition. Not only databases can be viewed and used as external persistent storage devices that store large predicates that do not fit in memory, but it is also the case that Prolog is a natural choice when it comes to selecting an inference engine for database systems. The *ODBC* library in *SWI-Prolog* [18] is closely related to our work since we have used the library as a blue print both for the C-interface code and for the library's predicates naming and argument conventions.

The field of integrating relational databases has a long tradition going back to the early years of Prolog [8]. For instance the pioneering work of Draxler[7], although based on writing out *SQL* rather than directly interrogating the database, provided extensive support for translating combinations of arbitrary Prolog and table-associated predicates to optimised *SQL* queries. The code has been ported to a number of Prolog systems[13]. Another approach which targeted machine learning and tabling as well as importing tables as predicates is *MYDDAS*, [5]. An early *ODBC* interface for *Quintus*

predicate name/arity	moded arguments	predicate name/arity	moded arguments
sqlite_connect/2	+File, ?Conn		
sqlite_connect/3	+File, ?Conn, +Opts	*sqlite_format_query/3*	+Conn, +SQL, -Row
sqlite_disconnect/1	+Conn	*sqlite_current_table/2*	+Conn, -Row
sqlite_current_connection/1	-Conn	*sqlite_table_column/3*	+Conn, ?Table, -Column
sqlite_default_connection/1	-Conn	*sqlite_table_count/3*	+Conn, +Table, -Count
sqlite_query/2	+SQL, -Row		
sqlite_query/3	+Conn, +SQL, -Row		

Table 1. Predicates for *proSQLite* library. Left: connection management and SQL queries. Right: auxiliary predicates on formatted queries and database introspection.

Prolog was *ProDBI* [12]. Prolog has also been used to implement a database management system based on the functional data model [10]. In this contribution we concentrate on describing an open-source modern library that can be used out-of-the-box with a zero configuration, community supported database system. We hope that the library will be a useful tool for the logic programming community and provide a solid basis in which researchers can contribute rather than having to reinvent the basic aspects of such integrations.

2 Library specifics

Here we present the overall architecture of the system along with the specific details of the three component architecture. Our library was developed on *SWI 6.1.4* under a *Linux* operating system. It is also expected to be working on the *Yap 6.3.2* [6] by means of the *C*-interface emulation [16] that has been also used in the porting other low-level libraries [2]. We publish[3] the library as open source and we encourage the porting to other Prolog engines as well as contributions from the logic programming community to its further development. Deployment is extremely simple and only depend on the location of the *SQLite* binary.

Our library is composed of three main components. At the lower level, written in *C*, the part that handles opening, closing and communicating with the *SQLite* OS library. The *C* code is modelled after, and borrows crucial parts from the *ODBC* library of *SWI*. On top of the low-level interface, sit two layers that ease the communication with the database. On the one hand, a set of predicates allow the interrogation of the database dictionary, while a third layer associates tables to Prolog predicates.

The heart of the library is its interface to *SQLite*. This is implemented in *C* and has strong affinity to the *ODBC* layer in *SWI*. The left part of Table 1 lists the interface predicates to the core system. Management predicates allow users to open, close and interrogate existence of connections to databases. The *C* code creates a unique, opaque term to keep track of open connections. However, this is not particularly informative to the users/programmers. More conveniently, the library allows for aliases to connections that can act as mnemonic handles. As a running example we will use the connection

[3] `http://bioinformatics.nki.nl/~nicos/sware`

table name	population	columns
secondary_accessions	286525	secondary_accession, primary_accession
identifier_mapping	3044651	uniprot_accession, identifier_type, target_identifier

Table 2. Structure of the uniport example database which stores protein identifier maps.

to a large but simple protein database[4] from *Uniprot*. It has two tables referenced on a single key and having $286,525$ and $3,044,651$ entries. The single file *SQLite* database is $184Mb$ in size. Table 2 summarises the basic parameters of the database

The type of connection we wish to establish to the database file is controlled by *sqlite_connect/3* . The user can interrogate all open connections and the existence of a specific connection via *sqlite_current_connection/1.* This predicate backtracks over all open connection if it is queried with a variable as its argument. The bulk of the traffic with *SQLite* is directed via *sqlite_query/2,3* through which data in tables can be added, deleted and queried. We include in the library a small number of predicates that assist user interaction with databases. These include parametrised query strings, interrogating the database dictionary and simple aggregate operations.

The formatted query mechanism provides a means for parametrised queries. This is useful for encoding common patterns of queries in an application. The function of the rest of the wrapping predicates follows directly their naming. The information they provide is gathered from the database dictionary which is managed by *SQLite*. For illustration purposes and for comparison with alternative ways of obtaining the counts of a table, we show in the code that follows how to use backtracking to obtain all tables in the *Uniprot* database along with their populations.

```
?- sqlite_current_table(uniprot, Table),
   time(sqlite_table_count(uniprot, Table, Count)),
   write(Table:Count),nl,fail.

% 7 inferences,0.007 CPU in 0.007secs (99% CPU,1013 Lips)
secondary_accessions:286525
% 7 inferences,0.083 CPU in 0.083 secs (100% CPU,85 Lips)
identifier_mapping:3044651
```

2.1 Tables as predicates

With the *as_predicates/1* option of *sqlite_connect/3* we can direct the library to create linking predicates for each table in the database. That is a predicate is created for each table in the underlying database. The predicates are created at module identified by option *at_module/1.* It is the responsibility of the user to ensure there are no name clashes. Once thus declared, the table predicates behave as normal Prolog predicates. The system makes simple transformations when filling the predicates with results from the database. Currently, this is by mean of creating an *SQL* `SELECT` statements in

[4] `http://bioinformatics..nki.nl/~nicos/sware/prosqlite/uniprot.sqlite`

which the `WHERE` sub-clause is formed from the ground arguments of the corresponding goal. For a table with name `Name` and columns that have a one-to-one correspondence with the list of variables in `Args`, and `Module` being the module provided at the `at_module` option of *sqlite_connect/3* . Predicates that correspond to database tables interact as if defined by a number of facts: each table row corresponds to a fact assertion to the Prolog database. To illustrate, we show the predicated goals for the two queries from the preceding section. Times are shown from a run on a Linux dual-core $3.16GHz$ desktop computer.

```
?- findall(S, secondary_accessions(S, 'P64943'), All).
   All = ['A0A111', 'Q10706'].

?- sqlite_current_table(uniprot, Table),
   findall(C,sqlite_table_column(uniprot,Table,C),Cs),
   length( Cs, Arity ), length( As, Arity ),
   Pred =.. [Table|As],
   time( ( findall(1,Pred,Ones),
               length(Ones,Count))),
   write(Table:Count), nl, fail.

%286,560 inf,0.561 CPU in 0.575 secs(98% CPU,510692Lips)
secondary_accessions:286525
%3,044,689inf,10.486CPUin10.516secs(100%CPU,290360Lips)
identifier_mapping:3044651
```

Predicated tables only depend on *SQL* transformations and as such are not specific to *SQLite* but can be easily ported to other interface libraries such as *ODBC*.

3 Applications

The last decades have witnessed a phenomenal increase in the amount of biological knowledge that has been published and codified [9]. This acceleration can be directly attributed to the evolution of high throughput technologies such as genome wide expression assays, microscopy and deep sequencing. One important way in which biological knowledge is codified is in the form of databases and ontologies. These include protein-protein interaction databases such as STRING [14] and HPRD [11] and protein information databases such as the universal protein resource *Uniprot* [15].

Prolog is a powerful platform for bioinformatics research and analysis. Its ability to query relational datasets and express recursive searches succinctly are of particular interest to ontologies and databases tabulating millions of relations. One of the main roadblocks hindering the use of Prolog in this research area is the lack of effective tools that give access to the resources available. Databases form the basic layer of biological knowledge available. The use of effective tools to connect databases in efficient, resilient and integrative manners to the logic engine can assist in narrowing this gap. Currently, we use *proSQLite* as one of the possible caching mechanisms in *pubGraph* a graph search tool that mines the citation relations from the *PubMed*[5] website to built visualisations of the relational networks.

Engaging Prolog with the world wide web (WWW) in the role of a web-server has been well advocated and served by supporting libraries [4, 17]. Furthermore, there has been previous motivating work on systems that realise Prolog servers that mediate the web-publication of material stored in relational databases [3]. The library presented here further facilitates the role of Prolog in this area. Particularly, with small to medium size web services. The main benefits of *SQLite* in this context are:

persistence - Prolog based servers need persistent storage of data. It is conceivable that such data can be realised as external files managed privately.

threading - Web-servers are inherently multi-threaded and the ability to communicate through a shared file-based database provides further plurality.

ease of deployment - *SQLite* is arguably one of the easiest database back-ends to install and maintain.

filestore abstraction - One of the main success stories for *SQLite* has been in providing application specific filestore solutions. This fits well within a web-server setting.

Currently, a large number of applications are reported to be using *SQLite* as an embedded database transaction system that is used to store application data in a uniform and robust manner. These include major open source projects such as the *Firefox/Mozilla*[6] browser and the *Powerdns*[7] DNS server. The embedded nature of *SQLite* reduces overheads and simplifies installation. Applications can use the layer to abstract their interactions with the operating system. Databases are stored in single files and are cross-platform compatible.

4 Conclusions

We presented a stable and efficient library for integrating a file-based DBMS to modern open-source Prolog engines. We have argued that Prolog is a powerful platform for data analysis and computational research in bioinformatics and for the realisation of agile web-servers that require minimal programming effort. Biological knowledge captured in the growing list of databases can be efficiently reasoned with, within logic programming. There are a number of possible extensions that can be envisaged on top of the presented library. These are not necessarily specific to this library but can also be of relevance to similar approaches such as the *ODBC* library of *SWI*. One such extension is *d*b_facts[8] which implements term based table interactions for *proSQLite* and *ODBC* databases. It also allows for a notation that selects columns from tables independently of their position in the respective table. This would allow decoupling of a table's precise list of constituent columns from accessing specific fields, making code easier to maintain as additions to the database structure do not need to be propagated to parts of the code that are not accessing the new columns.

[5] `http://www.ncbi.nlm.nih.gov/pubmed`
[6] `http://www.mozilla.org/`
[7] `http://doc.powerdns.com/gsqlite.html`
[8] `http://bioinformatics.nki.nl/~nicos/sware/db_facts`

Acknowledgements

We would like to thank Jan Wielemaker for discussions and for making the *ODBC* library of the *SWI-Prolog* open source. Also we thank Graham Kemp for valuable comments. This work was (co)financed by the Netherlands Consortium for Systems Biology (NCSB) which is part of the Netherlands Genomics Initiative/Netherlands Organisation for Scientific Research. SC and NA contributed equally.

References

1. G. Allen and M. Owens. *The Definitive Guide to SQLite*. 2010.
2. N. Angelopoulos, V. S. Costa, R. Camacho, J. Wielemaker, J. Azevedo, and Lodewyk Wessels. Integrative statistics for logical reasoning, 2012. Conditional accept PADL'13.
3. N. Angelopoulos and P. Taylor. An extensible web interface for databases and its application to storing biochemical data. In *WLPE '10*, Edinburgh, Scotland, July 2010.
4. D. Cabeza and M. Hermenegildo. Distributed WWW programming using Ciao Prolog and the PiLLoW library. *Theory and Practice of Logic Programming*, 1(3):251–282, May 2001.
5. J. Costa and R. Rocha. Global Storing Mechanisms for Tabled Evaluation. In *Proc. of the 24th Int. Conf. on Logic Programming, ICLP'08*, number 5366 in LNCS, pages 708–712, Udine, Italy, 2008.
6. V. S. Costa, R. Rocha, and L. Damas. The YAP Prolog system. *Journal of Theory and Practice of Logic Programming*, 12:5–34, 2012.
7. C. Draxler. *Accessing Relational and Higher Databases Through Database Set Predicates*. PhD thesis, Zurich University, 1991.
8. P. M. D. Gray and R. J. Lucas, editors. *Prolog and Databases, Implementations and New Directions*. Ellis Horwood Ltd, Chichester, U.K., 1988.
9. P.M.D. Gray, G.J.L. Kemp, C.J. Rawlings, N.P. Brown, C. Sander, J.M. Thornton, C.M. Orengo, S.J. Wodak, and J. Richelle. Macromolecular structure information and databases. *Trends in Biochemical Sciences*, 21:251–256, 1996.
10. G.J.L. Kemp, J.J. Iriarte, and P.M.D. Gray. Efficient Access to FDM Objects Stored in a Relational Database. In *Directions in Databases: Proceedings of the Twelfth British National Conference on Databases*, pages 170–186, 1994.
11. T. S. Keshava Prasad, Renu Goel, Kumaran Kandasamy, Shivakumar Keerthikumar, Sameer Kumar, et al. Human protein reference database 2009 update. *Nucleic Acids Research*, 37(suppl 1):D767–D772, 2009.
12. R. Lucas and Keylink Computers Ltd. *ProDBI: ODBC Interface for Quintus Prolog*. Kenilworth, UK, Keylink Computers Ltd, 1997.
13. C. Mungall. Experiences using logic programming in bioinformatics. In *Logic Programming, 25th International Conference, ICLP 2009*, pages 1–21. 2009.
14. D. Szklarczyk, A. Franceschini, M. Kuhn, M. Simonovic, A. Roth, P. Minguez, T. Doerks, M. Stark, et al. The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Research*, 39(suppl 1):D561–D568, 2011.
15. The UniProt Consortium. Reorganizing the protein space at the universal protein resource (uniprot). *Nucleic Acids Res*, 40:D71–D75, 2012.
16. J. Wielemaker and Vítor S. C. On the portability of prolog applications. In *Practical aspects of Declarative Languages*, pages 69–83, 2011.
17. J. Wielemaker, Z. Huang, and L. van der Meij. SWI-Prolog and the Web. *Theory and Practice of Logic Programming*, 8(3):363–392, 2008.
18. J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.