Exploring file based databases via an SQLite interface

Sander Canisius, Nicos Angelopoulos, and Lodewyk Wessels

Netherlands Cancer Institute, Amsterdam, Netherlands
{s.canisius,n.angelopoulos}@nki.nl

Abstract. We present a succinct yet powerful interface library to the *SQLite* database system. The single file, serverless 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. Finally, this paper discusses the strengths of the system and highlights possible extensions.

1 Introduction

SQLite[1][Allen and Owens, 2010] is a powerful, open source serverless 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. In the context of Prolog, *SQLite* can provide a flexible transaction-based extension to the in-memory predicate storage. In this paper we present the 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. As a result, there is an extensive body of literature on a variety of aspects of such integrations. We do not attempt a general overview of the area, but we present some of the research that is most clearly connected to our work. The *ODBC* library in *SWI-Prolog* [Wielemaker et al., 2012] 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. For instance the pioneering work of Draxler [1991], 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 reportedly, [Mungall, 2009], been ported to a number of Prolog systems, while it has also be extended and adopted for the *ODBC* library in the *Blipkit* software suite [Mungall, 2009]. Another approach which

[1] http://www.sqlite.org/

targeted machine learning and tabling as well as importing tables as predicates is MYDDAS, [Costa and Rocha, 2008].

Our current contribution does not deal with the theoretical or extensional aspects of integrating a database system to Prolog, in contrast, 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. In doing so, 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 integration.

The integration of flexible external database storage is crucial to the uptake of Prolog to new application and research areas which can assist with the up-keep of its development and its user penetration. Two such application areas are bioinformatics and web-servers. In the former, the ability to tap on to large data-sets is a central aspect due to the deluge of data publicly available resulting from the ever-growing number of high throughput technologies available to experimental biologists [for instance see Szklarczyk et al., 2011]. Our effort, along with other recent libraries, [Angelopoulos et al., 2012. In preparation.], takes a piece-meal approach into providing tools that can increase the penetration of Prolog into bioinformatics. These act complementary to more holistic approaches that provide a suite of programs within a single framework [Mungall, 2009]. In web-server applications, single file databases can facilitate scalable and effective interthread and back-end communication. Our library can enhance the excellent web facilities in modern open-source Prolog systems [Cabeza and Hermenegildo, 2001, Wielemaker et al., 2008].

The remainder of the paper is organised as follows. Section 2 describes the workings of the library. Section 3 presents some tests and possible applications, while the concluding remarks are in Section 4.

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* by means of the *C*-interface emulation that has been also used in the porting other low-level libraries [Angelopoulos et al., 2012. In preparation.]. We publish [2] 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. Installation and running are extremely simple and only depend on the location of the OS *SQLite* library.

2.1 Overall architecture

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*.

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

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

Table 1. Predicates table for Sqlite library. Top part: core functionality of the library, pertaining connection management and basic SQL query communication. Bottom section: auxiliary predicates on formatted querries and database introspection. There are no explicit predicates for importing tables, this is done wholesale for a database via a predicate option when a connection is initiated.

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 the third layer allows the integration and interaction with tables as Prolog predicates.

2.2 Low-level interface

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 top part of Table 1 lists the interface predicates to the core system. The basic opening and closing operations are implemented as :

```
sqlite_connect(+File, ?Conn)

sqlite_disconnect(+Conn)
```

The *C* code creates a unique, opaque term to keep track of open connections. However, this is not particularly informative to the users/programmers. To circumvent this, the library allows for aliases to connections that can act as mnemonic handles through which connections are known to the user. To establish such an alias it is sufficient to give a non variable, usually atomic, value to the connection variable of *sqlite_connect/2*.

As a running example we will use the connection to a large but simple protein database from *Uniprot*. It has two tables referenced on a single key and each having $286,525$ entries. Table 2 summarises the basic parameters of the database. Except if otherwise mentioned, the examples in this papers assume that the database detailed in Table 2 is open by :

```
sqlite_connect('uniprot.sqlite', uniprot)
```

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

Table 2. Structure of the database used as an example in the text.

To fine-tune details on the type of connection we wish to establish to the database file, we introduce an options argument in *sqlite_connect/3* . As is common practice, this is the last argument of the predicate. In addition to allowing a list of options from the set of possible options we also allowed single term options as a matter of convenience. The members of the recognised set are :

`alias(Alias)` An alternative way to register a connection alias. Note that in the case where `Conn` is also non-variable `Alias` and `Conn` should be unifiable.

`as_predicates(AsPred)` AsPred should be a Boolean value which when true instructs the library to create hook predicates that map each sqlite table to a Prolog predicate. These are created in the `AtMod` module, and it is the user's responsibility to make sure each predicate is unique in this module. See Section 2.4.

`at_module(AtMod)` Defines the module in which predicates should be defined when `AsPred` is `true`. By default, when this option is not present, predicates are defined in the `user` module.

`exists(Exists)` `Exists` should be a Boolean value. When this value is `false` the library does not throw an error if the provided file does not exist. The default value is `true` in which case an error is thrown if the *SQLite* file does not exist.

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 library also maintains the notion of a default connection which can be identified with *sqlite_default_connection/1*. This is usually set to the last connection opened and it is particularly useful in the common deployment scenario where a single connection is maintained.

The bulk of the traffic with *SQLite* is directed via *sqlite_query/2,3*. The version with arity 2 is simply a shortcut for applying the query on the default connection. The `SQL` argument of the predicates is an atom of valid *SQL* syntax, which when applied to the opened connection identified by `Conn`, results in rows that are returned one at the time in the form of a *row/n* term structure.

2.3 SQL wrapping predicates

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.

sqlite_format_query(Conn,SqlFormat,Row) Post a format style *SQL* query to *SQLite* `Connection` and get row result in `Row`.

sqlite_current_table(Conn,Table) Enumerate all tables in the database or query whether a specific table is part of a database.

sqlite_table_column(Conn,Table,Column) This predicate holds for each triplet of `Conn`, `Table` and `Column` such that the named table has the respective field (`Column`) in the context of the given database.

sqlite_table_count(Conn,Table,Count) Count is the number of data rows in `Table`; which is a table present in the database identified by `Conn`.

2.4 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 which the `WHERE` subclause 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`, the library constructs and asserts a clause in the Prolog database by using the following code :

```
Head =..[Name|Args],
Body = sqlite:sqlite_holds(Conn,Name,Arity,Columns,Args),
user:assert((Head :- Body)),
```

At runtime *sqlite_holds/5* ensures that the appropriate transformations take place and the results are fed back to Prolog as expected. By using predicated tables the number of rows for table *secondary_accessions/2* can be found with the following query :

```
?-
   Opt = as_predicates(true),
   sqlite_connect('uniprot.sqlite',uniprot,Opt),
   findall( A, secondary_accessions(A,_), As ),
   length( As, Len ).

As = ['A0A111', 'A0A112', 'A0A113', 'A0A131'|...]
Len = 286525.
```

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

3.1 Bioinformatics

The last decades have witnessed a phenomenal increase in the amount of biological knowledge that has been published and codified. This acceleration can be directly at-

tributed 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 (PPI) databases such as STRING [Szklarczyk et al., 2011] and HPRD [Keshava Prasad et al., 2009] and protein information databases such as the universal protein resource [*Uniprot*, The UniProt Consortium, 2012]. STRING collates information about protein interactions from a variety of sources. It currently contains information on $5,214,234$ proteins from $1,133$ organisms and holds $224,346,017$ interactions. HPRD holds human proteins and interactions between them. Currently there are $39,194$ interactions in HPRD. The *Uniprot* database includes a wide-ranging array of information on proteins. In this paper we concentrated on the mapping abilities between naming conventions for proteins. In the *SQLite* file we created from a recently downloaded dataset from *Uniprot*there are $286,525$ gene identifier mappings.

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.

3.2 Web-services

Engaging Prolog with the world wide web (WWW) in the role of a web-server has been well advocated and served by supporting libraries [Cabeza and Hermenegildo, 2001, Wielemaker et al., 2008]. Furthermore, there has been previous motivating work on systems that realise Prolog servers that mediate the web-publication of material stored in relational databases [Angelopoulos and Taylor, 2010].

The library presented here further facilitates the role of Prolog in this area. 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. However, the facilities offered by *DBMS* along with the presented predicated tables lead naturally to steadfast and scalable solutions.

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

ease of deployment - *SQLite* is arguably one of the easiest database back-ends to install and maintain. It depends on a single OS library and created databases are OS agnostic. In terms of the database it is thus trivial to migrate, move or upgrade the server.

filestore - One of the main success stories for *SQLite* has been in providing application specific filestore solutions. This fits well within a web-server setting where the server is provided with a clean back-end for all interactions with the filestore.

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. The library presented here facilitates such integrations in a straight forward, efficient and zero-configuration fashion that can facilitate rapid prototyping, collaborative coding and the development of reusable programs.

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 possibility would be to have a more refined model for imported predicates which will not necessarily import all tables from a database. With regard to predicated tables, an integration with the work of Draxler [1991] would improve performance on queries that can be mapped to *SQL* join operations. Ideally, one would like this to be done behind the scenes, away from any user intervention. Finally, it might be possible to integrate the *ODBC* and *SQLite* libraries in one package within *SWI*, creating a powerful, common interface in which user programs are largely agnostic to the back-end database system used.

The underlying development principles of our approach follows other recent developments [Angelopoulos et al., 2012. In preparation.]. The common aspect is that although inspired from a specific application area, in both cases bioinformatics, the approaches produced stand-alone libraries that can be of use to a wider audience. This approach complements projects that take a more integrative approach to program development. In the case of bioinformatics such a suite is the *Blipkit* [Mungall, 2009].

5 Acknowledgements

We would like to thank Jan Wielemaker for discussions, feedback and for making the *ODBC* library of the *SWI-Prolog* open source, on which our library is based. Also, we are thankful to the anonymous reviewers who made valuable suggestions.

Bibliography

Grant Allen and Mike Owens. *The Definitive Guide to SQLite*. Apress, 2nd edition, 2010. ISBN 1-4302-3225-0.

Nicos Angelopoulos and Paul Taylor. An extensible web interface for databases and its application to storing biochemical data. In *WLPE '10*, Edinburgh, Scotland, July 2010. URL `http://arxiv.org/pdf/1009.3771v1`.

Nicos Angelopoulos, Vitor Santos Costa, Joao Azevedo, Rui Camacho, and Lodewyk Wessels. Integrative statistics for logical reasoning, 2012. In preparation. URL `http://bioinformatics.nki.nl/~nicos/sware/real/`.

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.

J. Costa and R. Rocha. Global Storing Mechanisms for Tabled Evaluation. In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the 24th International Conference on Logic Programming, ICLP'2008*, number 5366 in LNCS, pages 708–712, Udine, Italy, December 2008. Springer-Verlag.

C. Draxler. *Accessing Relational and Higher Databases Through Database Set Predicates*. PhD thesis, Zurich University, 1991.

T. S. Keshava Prasad, Renu Goel, Kumaran Kandasamy, Shivakumar Keerthikumar, Sameer Kumar, Suresh Mathivanan, Deepthi Telikicherla, Rajesh Raju, Beema Shafreen, Abhilash Venugopal, Lavanya Balakrishnan, Arivusudar Marimuthu, Sutopa Banerjee, Devi S. Somanathan, Aimy Sebastian, Sandhya Rani, Somak Ray, C. J. Harrys Kishore, Sashi Kanth, Mukhtar Ahmed, Manoj K. Kashyap, Riaz Mohmood, Y. L. Ramachandra, V. Krishna, B. Abdul Rahiman, Sujatha Mohan, Prathibha Ranganathan, Subhashri Ramabadran, Raghothama Chaerkady, and Akhilesh Pandey. Human protein reference database2009 update. *Nucleic Acids Research*, 37(suppl 1):D767–D772, 2009.

Chris Mungall. Experiences using logic programming in bioinformatics. In Patricia Hill and David Warren, editors, *Logic Programming, 25th International Conference, ICLP 2009*, volume 5649 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2009.

Damian Szklarczyk, Andrea Franceschini, Michael Kuhn, Milan Simonovic, Alexander Roth, Pablo Minguez, Tobias Doerks, Manuel Stark, Jean Muller, Peer Bork, Lars J. Jensen, and Christian von Mering. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Research*, 39(suppl 1):D561–D568, 2011. doi: 10.1093/nar/gkq973. URL `http://nar.oxfordjournals.org/content/39/suppl_1/D561.abstract`.

The UniProt Consortium. Reorganizing the protein space at the universal protein resource (uniprot). *Nucleic Acids Res*, 40:D71–D75, 2012.

Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. Swi-prolog and the web. *Theory and Practice of Logic Programming*, 8(3):363–392, 2008.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.