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 *proSQLitet*'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* 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

K. Sagonas (Ed.): PADL 2013, LNCS 7752, pp. 222–227, 2013.

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

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		

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[1] 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 to a large but simple protein database[2] 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

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

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