Proc. of the Second Intl. Conference on Advances in Information Technology — AIT 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5939-1

Extensible Database Communication Modification Framework

Pinaet Phoonsarakun
The Sirindhorn International Thai-German Graduate School of Engineering, King Mongkut's University of Technology North Bangkok
Bangkok, Thailand

Kamol Limtanyakul
The Sirindhorn International Thai-German Graduate School of Engineering, King Mongkut's University of Technology North Bangkok
Bangkok, Thailand

Alexander Adam
dimensio informatics GmbH
Chemnitz, Germany

Wolfgang Benn
Chemnitz University of Technology
Chemnitz, Germany

***Abstract*— Current databases use many different protocols to communicate with their clients. Applications running on that communication protocols have to implement support for each of them. In this paper, we propose an abstraction layer, that enables an application to be applicable to many database protocols, such as the database protocol TNS for Oracle database, TDS for Microsoft SQL Server, DRDA for IBM DB2, and so on, using only one abstract interface. On this layer, there will be various primary abstract functions that database protocol applications can customize or integrate them for their own particular purposes, such as SQL rewrite, analysis, timing, result set cache, direct generation of result sets, intrusion detection, etc. The aim of this paper is to develop and propose this abstraction layer. Finally we show some examples of applications utilizing the proposed abstraction layer, they are able not only to perform SQL rewrite and timing, but also support the database protocol TNS, TDS, and DRDA.**

***Keywords*— abstraction layer, database management systems, SQL, database protocol, TNS, TDS, DRDA**

I. Introduction

Computer technology helps humans to calculate things and obtain the results extremely quickly and accurately. The technology is then used and applied to enhance many aspects of human activities, such as entertainment, education, exploration, communication, industrial production, transportation, business, finance, etc. In doing that, a large amount of data needs to be managed properly and securely, especially when data need to be exchanged between devices from great distances. This is why database management systems and computer network communication and security have become important, to help technology deal with data exchange and management. On the market, there are many database management systems (DBMS) available, e.g. Oracle database [1], Microsoft SQL Server [2], IBM DB2 [3][4], etc., and each of them uses its own database protocol, which possesses syntax and semantics as the rules for the data exchange and management between devices, database clients and servers.

Also there are many APIs (application programming interfaces) and abstractions available and some of them are able to support most of DBMS vendors, e.g. JDBC [5] (java database connectivity) and ODBC [6] (open database connectivity) can support Oracle database, Microsoft SQL Server, IBM DB2, etc. These APIs and abstractions are provided only at the application level.

At the network level, there are specialized tools for particular purposes but they are all specific to some database protocol(s) and do not provide any common API or abstraction, as detailed more in the section Related Work.

Therefore, there is currently no abstraction layer for all database protocols at the network level. Once there is such an abstraction layer, it will enable database protocol applications to support many database protocols, e.g. TNS (transparent network substrate) [1] for Oracle database, TDS (tabular data stream) [2] for Microsoft SQL Server, DRDA (distributed relational database architecture) [3] [4] for IBM DB2, etc. In this abstraction layer, there will be various primary abstract functions concerning events of database communication, from connection until disconnection. Database protocol applications can customize or integrate these abstract functions for their own particular purposes, such as SQL rewrite, analysis, timing, intrusion detection, result set cache, direct generation of result sets, etc. The design of the abstraction layer is shown in Fig. 1.

The abstraction is also necessary for an application that wishes to be integrated into existing IT systems without modifying any code of database client or server. One example use case of the abstraction is SQL rewrite which has the purpose to accelerate database response time, as illustrated in Fig. 2. There will be an extra delay belonging to the SQL modification. By the improvement of response time, the delay can become less significant. Furthermore, with the abstraction,

Proc. of the Second Intl. Conference on Advances in Information Technology — AIT 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5939-1

the SQL modification will have more potential for different database protocols.

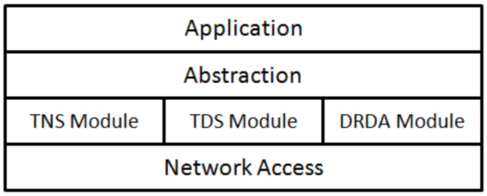


Figure 1. The abstraction layer design

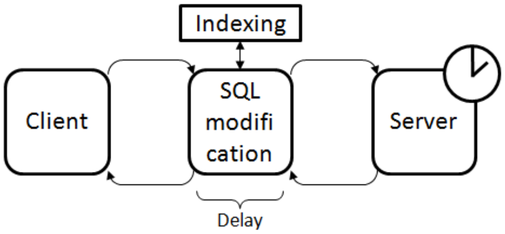


Figure 2. The SQL modification at the network level

Therefore, the goal of this paper's work is to develop and propose an abstraction layer for database protocols at the network level and analyze its performance.

II. Related Work

One language used for storing, retrieving, or manipulating data in relational databases is the SQL (structure query language). Analyzing and modifying SQL statements to optimize them can accelerate the information search process [7].

As mentioned in the introduction, there are APIs and abstractions that can support many relational databases, such as JDBC and ODBC, but they all are at the application level.

At the network level, there are specialized tools that can do some tasks but they are all specific to some database protocol(s), as described in the following.

TDS protocol analyzer [8] is specified for the database protocol TDS, which is used by Microsoft SQL Server. It has several features which include packet capture, packet analysis, packet storage, traffic statistics, and vulnerability warning.

GreenSQL [9] is specified for database protocols of relational databases MySQL and PostgreSQL by its open source version. It is also available for the TDS protocol in its commercial version [10]. This tool has several features, such as a firewall, filtering SQL statements with malicious intent against the database server, e.g. SQL injection vulnerabilities.

Security Testing Framework [11] is specified for the database protocol DRDA. It is used to test the database protocol by violating the syntax rules and semantics of the protocol.

Furthermore, the tools do not provide any common API or abstraction layer.

Consequently, for the work presented in this paper, it was decided to bring an abstraction layer onto the network level on top of database protocols, as shown in Fig. 1. On the abstraction layer, there are various abstract functions. All of these functions are based on C/C++ programming language.

The work presented in this paper used a network proxy to access network traffic to analyze and understand the database protocol used between a database client and the database server.

III. The Abstraction Layer for Database Protocols

In Fig. 1, the abstraction layer will have various abstract functions. These functions are generalizations that arise from broad similarities of all database protocol modules and will be called or invoked by one of the modules when the right event is met with the right parameter(s), to perform some specific tasks. These functions are callback functions and virtual, thus not implemented. They have to be overloaded to do something more useful/application specific/else. They are defined as standardized interfaces to cooperate with the lower layer. This makes it easier to change the implementation of the functionality provided by the layer. Moreover, the remainder of the system remains unchanged when a layer's implementation is changed as long as the layer provides the same functionality to the layer above it, and uses the same functionality from the layer below it.

A. *Abstract Functions*

In the whole process of data query between a database client and the database server, there are generally five phases, i.e. query preparing (prep), binding (bind), execution (exec), result set fetching (fetch), and closing (free), as shown in Fig. 3. The arrows to the right are client part and the opposites are server part. Abstract functions on the abstraction layer are based on the whole process.

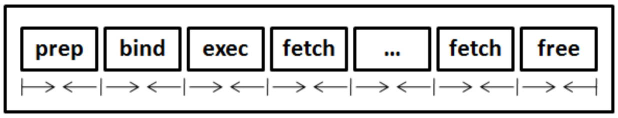
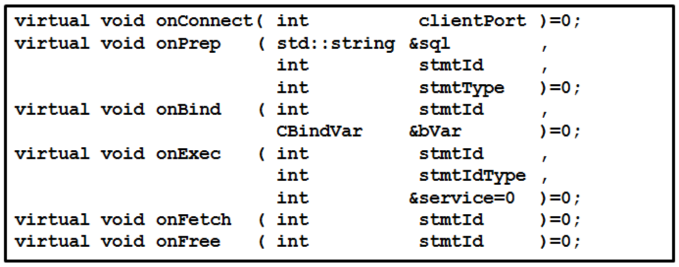


Figure 3. The whole process of data query

On the abstraction layer, the primary abstract functions listening to the DB client side are detailed in Fig. 4.



```
virtual void onConnect( int          clientPort )=0;
virtual void onPrep    ( std::string &sql        ,
                         int          stmtId     ,
                         int          stmtType   )=0;
virtual void onBind    ( int          stmtId     ,
                         CBindVar    &bVar       )=0;
virtual void onExec    ( int          stmtId     ,
                         int          stmtIdType ,
                         int         &service=0  )=0;
virtual void onFetch   ( int          stmtId     )=0;
virtual void onFree    ( int          stmtId     )=0;
```

Figure 4. The primary abstract functions listening to DB client

- `onConnect` will be called or invoked with the parameter `clientPort` when a database protocol module detected a connection attempt.
- `onPrep` will be invoked with the parameters when a database protocol module detected a phase of query preparing from client. This function allows an application to rewrite the SQL statement in a DB protocol packet by using the parameter `sql`. The parameter `stmtId` is used to refer to the SQL statement for the whole query process. The parameter `stmtType` indicates the type of the SQL statement whether it is a normal statement or prepared statement, etc.
- `onBind` will be invoked with the parameters when a database protocol module detected a phase of query binding from client. This function allows an application to modify values of the binding variables in a DB protocol packet by using the parameter `bVar`. The parameter `stmtId` indicates which SQL statement the binding variables belong to.
- `onExec` will be invoked with the parameters when a database protocol module detected a phase of query execution from client. The parameter `stmtId` indicates which SQL statement is going to be executed. The parameter `stmtIdType` indicates the type of `stmtId` whether it is depending on `onPrep` or `onExecDb`. If the `stmtIdType` is the latter, the `stmtId` must be replaced by the `stmtId` of the very first `onExecDb` event (note that `onExec` and `onExecDb` are not the same function). Otherwise the `stmtId` will be stuck to the `onPrep` event until it is closed by the `onFree` or `onFreeDb` event. In most cases, `stmtIdType` is depedning on `onExecDb`. The parameter `service` allows an application to use some special service from the database protocol module. The application returns `service` valued 0 for no special service; valued 1 in order to let the module to cache all result sets of the SQL statement and reply all of the result sets when found the same SQL again; valued 2 in order to generate direct result set independently from database; valued 3 to block the execution. If `service` is set to 2, the abstract function `onAnsDir` will be invoked to supply the service.

In some cases, events `onPrep`, `onBind`, `onExec` can be combined into one DB protocol packet, but not in separate packets, a TDS packet with ProcID 13, namely "Sp_PrepExec", for instance. Therefore the events `onPrep`, `onBind`, and `onExec` will be invoked consecutively by such one DB protocol packet.

- `onFetch` will be invoked with the parameter when a database protocol module detected a phase of query result set fetching from client. The parameter `stmtId` indicates which SQL statement is going to fetch more result sets.
- `onFree` will be invoked with the parameter when a database protocol module detected a phase of query closing from client. The parameter `stmtId` indicates which SQL statement is going to be closed or has been already closed.

On the abstraction layer, the primary abstract functions listening to the DB server side are detailed in Fig. 5.

```
virtual void onPrepDb ( int          stmtId     )=0;
virtual void onBindDb ( int          stmtId     )=0;
virtual void onExecDb ( int          stmtId     ,
                        CResultSet  &resultset  )=0;
virtual void onFetchDb( int          stmtId     ,
                        CResultSet  &resultset  )=0;
virtual void onFreeDb ( int          stmtId     )=0;
virtual void onDirAns ( int          stmtId     ,
                        CResultSet  &resultset  )=0;
virtual void onDisconn( int          clientPort )=0;
```

Figure 5. The primary abstract functions listening to DB server

- `onPrepDb` will be invoked with the parameter when a database protocol module detected a phase of query preparing from server. The parameter `stmtId` indicates which SQL statement this event belongs to.
- `onBindDb` will be invoked with the parameter when a database protocol module detected a phase of query binding from server. The parameter `stmtId` indicates which SQL statement this event belongs to.
- `onExecDb` will be invoked with the parameters when a database protocol module detected a phase of query execution from server. The parameter `stmtId` indicates which SQL statement the result set belongs to. This function allows an application to modify the result set from database by using the parameter `resultset`.

In some cases, events `onPrepDb`, `onBindDb`, `onExecDb` can be combined into one DB protocol packet, but not in separate packets. For instance, if a client sends a TDS packet with the ProcID 13 to the DB server, the events `onPrepDb`, `onBindDb`, `onExecDb` will be invoked consecutively by such one DB protocol packet afterwards.

- `onFetchDb` will be invoked with the parameters when a database protocol module detected a phase of query result set fetching from server. The parameter `stmtId` indicates which SQL statement the result set belongs to. This function also allows an application to modify the result set from database by using the parameter `resultset`.
- `onFreeDb` will be invoked with the parameter when a database protocol module detected a phase of query closing from server. The parameter `stmtId` indicates which SQL statement has been already closed.
- `onAnsDir` will be invoked with the parameters if the returned value of `service` of the last `onExec` event is set to 2, in order to supply the service of generating direct result set, independently from database. An application will have to provide the result set for the generation by using the parameter `resultset`. The parameter `stmtId` indicates

which SQL statement the generating result set will belong to.

- `onDisconn` will be invoked with the parameter when a database protocol module detected a disconnection. The parameter `clientPort` indicates which client port the disconnection belongs to.

The struct variables `CBindVar` and `CResultSet` can be found in the Appendix in [12].

The abstract function that is used to inform matters to an application is shown in Fig. 6.

```
virtual void onException( int           eNum      ,
                          std::string   eMsg      )=0;
```

Figure 6. The abstract function for an exception event

- `onException` will be invoked with the parameters when the lower layer of the abstraction layer needs to inform an application about matters concerning warnings, errors, or the output that the application returned to a DB module does not meet a DB protocol standard, for example, the length of SQL statement is too long, Oracle database is not able to support the length greater than 4000 characters, etc.

IV. Result

The paper's work separates tests into two groups. All integrations in this section base on the abstract functions from subsection 3.1 in this paper.

The databases used for both test groups are Oracle database version 11g, Microsoft SQL Server version 2008, and IBM DB2 version 10. The computer specification is as follows: CPU: Intel(R) Core(TM) i5 CPU 660 @ 3.33 GHz, RAM: 4 GB DDR3 Speed 1333 MHz, HD: 500 GB, OS: Linux Debian Squeeze.

The first test group used data for the tests as shown in Table 1.

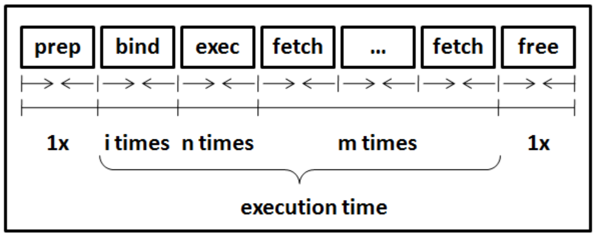


Figure 7. The SQL timing analysis concept

One possible use case is SQL timing. Its purpose is to optimize SQL statements, known as SQL tuning [13], through monitoring and measuring SQL performance. The SQL timing analysis concept is shown in Fig. 7. The two abstract functions in Fig. 8 are provided on top after integration of all abstract functions in the subsection 3.1, except `onDisconn` and `onAnsDir`. An SQL timing application integrated the two abstract functions, then was tested with the Microsoft SQL Server, and had the result as follows.

RequestDataPacket:

```
0x0301017d000001001600000012000000020000000000000000000001000000ffff0d00
020000001260404ffffffff0000e758000904d000005800400076005f006e006f002000
69006e0074002c00400076005f006e0061006d00650020006e00760061007200630068
006100720028003200380030029002c00400076005f00740061007800200066006c00
6f00610074000000e788000904d00000880053004500 4c0045004300540020002a0020
004600520004f004d00200045004d005000200057004800450052004500200065006d00
70006e006f003d00400076005f006e006f00200061006e00640020065006e0061006d
0065003d00400076005f006e0061006d00650020061006e006400200074006100780 0
3d00400076005f0074006100780005400076005f006e006f000026040487030000740
0076005f006e0061006d00650000e73002090 4d00000120041006c006500780061006e
0064006500720006400076005f007400610078000006d08080000000000e05240
```

Client Port:	`1053`
Timestamp:	`1357424616.111147`
Time Calls:	`req_SP_PREPEXEC`
Statement ID:	`0`
SQL:	`SELECT * FROM EMP WHERE empno=@v_no and ename=@v_name and tax=@v_tax`
BindingVariable:	`@v_no=903 @v_name=Alexander @v_tax=75.5`

ReplyDataPacket:

```
0x040100ac003401008104000000000000800380565006d0070006e006f000000000009
00af0a000904d000000565006e0061006d00650000000000090028086800690072006 5
004400610074006500000000000009006d0803740061007800d1870300000a00416c6578
616e64657220039b2e0b0800000000000e05240ff1100c100010000000000000000790000
0000ac00000001000000000000002604040100000 0fe0000e000000000000000000000
```

```
Client Port:      1053
Timestamp:        1357424616.115147
Time Calls:       ans_SP_PREPEXEC
Resultset:        empno | ename      | hireDate  | tax
                  903   | Alexander  | 01-Jun-07 | 75.5
Statement ID:     1
ResponseTime:     4000 (usec)
```

Note that the ResponseTime is from the subtraction of Timestamps.

TABLE I. EMP TABLE

EMPNO	ENAME	HIREDATE	TAX
901	Martin	01-FEB-06	70.5
902	Andreas	01-MAR-06	70.5
903	Alexander	01-JUN-07	75.5
904	Dirk	01-DEC-08	70.5
905	Hartmut	01-MAY-09	50.5

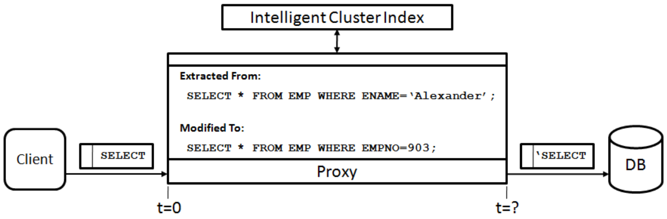
```
virtual void onClientSqlTime( int           clientPort  ,
                              std::string   timeType    ,
                              timeval       timeStamp   ,
                              int           stmtId      ,
                              std::string   sql         ,
                              CBindVar      bVar        ) = 0;
virtual void onServerSqlTime( int           clientPort  ,
                              std::string   timeType    ,
                              timeval       timeStamp   ,
                              int           stmtId      ,
                              CResultSet    resultset   )=0;
```

Figure 8. The abstract functions for SQL timing after integrated the primary abstract functions

Another possible use case is SQL rewrite. Rewriting a SELECT statement in SQL can speed up the search process [14]. By doing this, indexes are used in the WHERE clause [15], as shown in Fig. 9. This SQL rewrite application integrated mainly three abstract functions from the subsection 3.1, that are `onPrep`, `onBind`, and `onExec`. After the abstraction integration, the SQL rewrite application was tested and able to rewrite SQL statements in network traffic.

Moreover, it was tested with the three databases with different SQL statement lengths varying from 43 to 3,985 characters to analyze the performance of each DB protocol module of the abstraction, as shown in Fig. 10. The length was increased by adding more conditions in the WHERE clause. The delay was measured by subtracting the timestamp at a data packet arrived at a DB protocol module and the timestamp at the data packet released from the module. The delay of each length was the average delay of repetition 40,000 times. The graphs can be seen that delay increases dramatically and they are the same in all DB protocol modules.

One real use case is to rewrite SQL statements with the Intelligent Cluster Index [15], shown in Fig. 9. The test was conducted with Oracle database version 11g. The table size after the table join condition of the SQL statement [12] was 59,986,052 records. The connection without the database external index took 32.106 seconds, while the connection with the external index took 0.05 second and the delay in average was 479 microseconds. The factor of improvement was 642 faster. We integrated the primary keys of Oracle database's results into the query as an additional WHERE-condition. The database first used these primary keys and evaluated the rest of the conditions afterwards. So the amount of data was reduced before the expensive tests were made.



```
SELECT * FROM EMP WHERE ENAME='Alexander';
SELECT * FROM EMP WHERE EMPNO=903;
```

Figure 9. The SQL rewrite with indexing overview

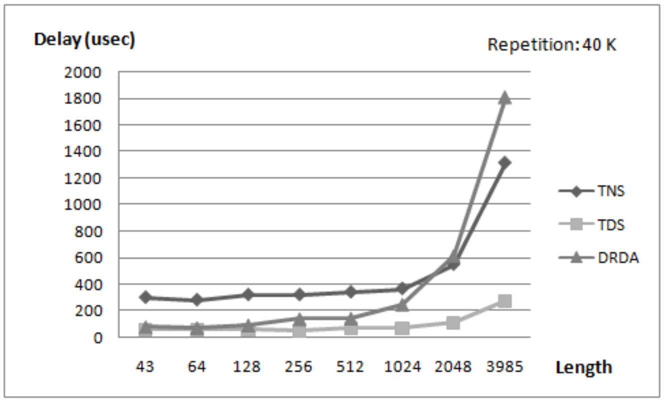


Figure 10. The relational graph testing of SQL length and delay of each DB protocol module

V. Conclusion

The work presented in this paper has designed and developed an abstraction layer at the network level on top of the database protocols TNS, TDS, and DRDA. Moreover the abstraction can be extended to support further database protocols easily. On the abstraction, there are various abstract functions based on the whole process of data query between a database client and the database server. These functions will be integrated into further database protocol applications, such as SQL rewrite, timing, analysis, intrusion detection, result set modification, result set cache, direct generation of result sets, etc. Also the paper's work has developed an abstraction integration for SQL timing and rewrite. The SQL timing integration is able to provide information for further SQL analysis. The SQL rewrite integration is able to modify SQL statements with various lengths and even provides the integration of a database external index. The influence of the abstraction layer is acceptable for SQL statement lengths below 2048.

The challenges that this paper's work has encountered were that in some cases all database protocols could not fit together completely, e.g. statement ID type in the abstract function `onExec`, and the lack of TNS documentation.

The future works of this paper are to extend the database protocol module to support further database protocols, such as MySQL, PostgreSQL, etc., and develop all of the modules to fully supply all of the abstract functions. We will also further analyze and improve performance of each DB protocol module.

References

[1] D. Litchfield, *The Oracle Hacker's Handbook: Hacking and Defending Oracle*. John Wiley & Sons, 2007.

[2] *[MS-TDS] Tabular Data Stream Protocol Specification*. Microsoft Corporation, December 14, 2011.

[3] *DRDA V5 Vol. 1: Distributed Relational Database Architecture (DRDA)*. The Open Group, July 2011.

[4] *DRDA V5 Vol. 3: Distributed Data Management (DDM) Architecture*. The Open Group, July 2011.

[5] R.M. Menon, *Expert Oracle JDBC Programming*. Apress, 2005. ISBN: 978-1590594070.

[6] Robert Signore, John Creamer, Michael O. Stegman, *The Odbc Solution: Open Database Connectivity in Distributed Environments*. Mcgraw-Hill, 1995. ISBN: 978-0079118806.

[7] D. K. Burleson, *Oracle Tuning The Definitive Reference*. Krittell, North Carolina, United States of America: Rampant, 2005.

[8] L. Guo and H. Wu, "Design and Implementation of TDS Protocol Analyzer," in Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference., 2009, pp. 633-636.

[9] (2012) GreenSql. [Online]. http://www.greensql.net/

[10] (2012) Product Overview. [Online]. http://www.greensql.com

[11] M. AboElFotoh, T. Dean, and R. Mayor, "An Empirical Evaluation of a Language-Based Security Testing Technique," 2009.

[12] P. Phoonsarakun, "Framework for SQL Modification and Analysis," Master's Thesis, The Sirindhorn International Thai-German Graduate School of Engineering (TGGS), King Mongkut's University of Technology North Bangkok (KMUTNB), Bangkok, 2012.

[13] J. Goodson and R. A. Steward, *The Data Access Handbook: Achieving Optimal Database Application Performance and Scalability*. Prentice Hall, March 2009.

[14] A. Abbasi, *Oracle 10g Database Administration Concepts and implementation Made Simple*. 2008.

[15] S. Leuoth, A. Adam, and W. Benn, "Profit of extending standard relational databases with the Intelligent Cluster Index (ICIx)," in Control Automation Robotics & Vision (ICARCV), 2010 11th, Singapore, 2010, pp. 1198-1205.