Unified Data Access for Global Electronic Business

Thian-Huat Ong

Unified Data Access for Global Electronic Businesses

Thian-Huat Ong, California State University, Sacramento, USA, ongt@csus.edu

Abstract

Databases are critical in managing business knowledge assets. SQL is the de facto way of accessing data because of its simplicity, which allows end users to get what they want effectively. However, as businesses become more globally and electronically interconnected, the need for automated data access increases accordingly and the simplicity of SQL quickly disappears. This is primarily due to the large number of competing technologies that facilitate automated data access, with factors including different programming languages, different database application programming interfaces, different database management systems, and different operating systems. In global businesses or extended networks of businesses, it is common to have many different competing and sometimes incompatible technologies working together at the same time. This research provides a new solution, a unified way to access data regardless of the underlying technologies. Therefore, businesses can save time and cost by focusing on getting the data they want, instead of focusing on how to get the data.

Keywords: unified database access, 1-to-1 SQL mapping paradigm, programming-language-independent, DBMS-independent, software development productivity.

1. Background

SQL, Structure Query Language, is the fundamental way of accessing data stored in a database [1]. The language is intuitive, so many users are able to learn the language quickly and put it to use immediately. However, to process a large number of business transactions or to analyze vast amounts of business data, automated data access is needed. Database application developers must write programs that can access databases to retrieve and store data. Typically, the process depends on the choice of programming languages, database application programming interfaces (API), database management systems (DBMSes), and operating systems. A global business is highly likely to have a mixture of different components at the same time, even within a single department, because new technologies are introduced and integrated while legacy systems are still being used and maintained.

Competing technologies exist at the same time because they have their own values and strengths. However, managing many different technologies at the same time is highly cost ineffective, and it can be a challenge to find developers who are familiar with all these different technologies. The problem is a well recognized one, as many researchers and commercial products are producing many solutions to this difficult problem. For example, different worthwhile efforts have been put into creating a unified database application programming interface (API), such as ODBC [2], JDBC [3], and ADO.NET [4]. Each was successful, yet none has become a clear winner. Part of the reason is that each API is highly tied to its host language, yet no database application developers or businesses as a whole use only one programming language. Consequently, many language-independent solutions have been produced, such as using middleware solutions or standardizing on XML [5] as the data exchange or storage standard. Yet these solutions do not offer the same space or time efficiency and convenience.

Since the early days of databases, SQL has been the standard language to query databases. In the domain of accessing data programmatically a single standard similar to SQL has never existed for very long. Researchers and practitioners by now probably believe that this will never happen, as evidenced by the popularity of middleware solutions as well as many researchers championing the idea that XML should be the only standard that should exist to solve the data access problem once and for all. Contrary to the common belief, this research provides a unified implementation that shows otherwise. Better yet, this research provides a breakthrough way of accessing data that is intuitive but its performance rivals or exceeds existing approaches.

Unlike previous attempts which were closed source, the author believes that publishing the source codes of this research (online at http://www.tudbc.org) allows businesses to have better control over how they want to access their data and customize it to fit their needs.

2. TUDBC: Truly Unified Database Connectivity

Truly Unified Database Connectivity (TUDBC) provides a single unified API for accessing data, regardless of programming languages, application programming interfaces (APIs), database management systems (DBMSes), and operating systems. In the past, each programming language required a unique style of coding for data access programming. Figure 1 shows an example of how TUDBC is used consistently across different programming languages. Figure 1 also highlights the consistency of TUDBC across different types of languages. This is particularly important for global businesses with multiple databases, because a single unified API saves on both the time and cost of learning, using and managing data access. Because the programming logic remains the same, businesses will have greater certainty that the programs will work correctly.

SQL Statements

```
❶BEGIN TRANSACTION;
❷INSERT INTO STUDENT (NAME, AGE) VALUES ('John Doe',19);
❸INSERT INTO GRADE (ID, COURSE, GRADE) VALUES (@@IDENTITY,'MIS101','A');
❹COMMIT TRANSACTION;
```

TUDBC's One-to-One Mapping of SQL Statements

Source Codes for Java, C#, J#, VB.NET, Python, and Ruby

General Syntax: *`object.method(parameter...)`*

```
❶connection.transactionBegin();
❷studentInsertSQL.insert(student);
❸gradeInsertSQL.insert(student.id, grade);
❹connection.transactionCommit();
```

Source Codes for Perl, PHP, and C++ (same syntax but without '$')

General Syntax: *`$object->method($parameter...)`*

```
❶$connection->transactionBegin();
❷$studentInsertSQL->insert($student);
❸$gradeInsertSQL->insert($student->id, $grade);
❹$connection->transactionCommit();
```

Source Codes for C

General Syntax: *`classname_method(object, parameter...)`*

```
❶TUDBCConnection_transactionBegin(connection);
❷StudentInsertSQL_insert(studentInsertSQL, student);
❸GradeInsertSQL_insert(gradeInsertSQL, student->id, grade);
❹TUDBCConnection_transactionCommit(connection);
```

Figure 1. One-to-one mapping of SQL statements to various programming languages.

Two production-quality source codes are currently released on the website, including C# (and any other .NET languages such as VB.NET, F#, J#, etc.) and Java (another language widely used in businesses). Many other languages are actively being developed, including PHP (one of the most popular web application languages), Ruby (one of the newest programming language), C (one of the oldest programming language), C++ (one of the oldest object-oriented programming language), Perl, and Python. Keep in mind that any .NET programming languages (such as VB.NET, F#, J#, etc) can immediately use the C# library to begin using TUDBC. The coverage shows that TUDBC works for both the newer preferred object-oriented programming languages and the traditional procedural programming languages (such as C).

TUDBC is intuitive, as shown in the one-to-one mapping between SQL statements and TUDBC programming statements. The intuitiveness lets TUDBC achieve the same level of ease of use as SQL, which is not available in any past database APIs, middleware, or even XML. Past database APIs were usually more focused on a lower level of abstraction. For example, Figure 2 shows how one SQL statement translates into many programming statements, which loses one-to-one correspondence to the SQL statement. Furthermore, each language has significant differences that prevent developers from switching from one language to another easily. The implication for businesses is that business users can now focus more on business logic and getting the actual data, instead of being distracted by complex lines of codes about how to get the data. Furthermore, they are no longer tied to any particular database vendors, and they can freely choose the best performing vendors without changing source codes.

One SQL Statement

```
INSERT INTO STUDENT (NAME,AGE,GPA) VALUES ('John Doe',19,3.75);
```

Many Programming Statements

Source Codes for Java with JDBC

```
PreparedStatement pst = connection.prepareStatement(
    "INSERT INTO STUDENT (NAME,AGE,GPA) VALUES (?,?,?)");
pst.setString(1, "John Doe");
pst.setInt(2, 19);
pst.setDouble(3, 3.75);
pst.execute();
```

Source Codes for C# with ADO.NET

```
SqlCommand cmd = new SqlCommand(
    "INSERT INTO STUDENT (NAME,AGE,GPA) VALUES (@name, @age, @phone)",
    connection);
cmd.Parameters.Add("@name", SqlDbType.VarChar, 30);
cmd.Parameters.Add("@age", SqlDbType.SmallInt);
cmd.Parameters.Add("@phone", SqlDbType.Float);
cmd.Prepare();
cmd.Parameters["@name"].Value = "John Doe";
cmd.Parameters["@age"].Value = 19;
cmd.Parameters["@phone"].Value = 3.75;
cmd.ExecuteNonQuery();
```

Figure 2. Old APIs mapping one SQL statement to many programming statements.

Currently, TUDBC works with all top major database management systems, including Oracle, SQL Server, MySQL, and DB2. It also works with databases popular for small businesses such as Access and Derby. Many businesses have data in text files and Excel spreadsheets, and TUDBC offers the same consistent support for them as well. Because TUDBC source codes are published, users can extend TUDBC to support virtually any possible databases or they can wait for them to be included in the next release.

3. Performance

One of the common concerns with a unified solution is space and time efficiency, as evidenced in middleware solutions and XML. TUDBC has its own unique caching mechanism to cache both SQL statements and database connections. Therefore it is able to provide performance that is close-to-the-best or the best performance in various settings. Interested readers are referred to the website to read about the experiments in detail.

One significant feature of the performance enhancement is that it is hidden from developers. While developers still write programs in the same way, by simply using TUDBC they get built-in performance enhancement. This has significant implication for online businesses. Web applications usually demand high performance. Yet achieving high performance usually introduces a complex mechanism that is difficult to manage and debug. On the other hand, TUDBC provides performance without sacrificing intuitiveness or simplicity. Similarly, the performance enhancement means that large amounts of data analysis can be done more quickly, which is particularly important for global businesses with large data warehouses and complex analytical needs.

4. Conclusion

TUDBC provides a unified solution for global businesses with complex database requirements for multiple platforms and high performance. TUDBC is intuitive because it corresponds directly to SQL statements, which allows database application developers to think and write programs more intuitively in terms of SQL statements. TUDBC has unified and consistent support for different programming languages, different database application programming interfaces, different database management systems, and different operating systems. Last but not least, TUDBC offers close-to-the-best or the best performance in various comparison settings. The implication of TUDBC is that global businesses can now better manage their data access by using TUDBC, which has the potential of improving productivity by increasing quality while saving time and cost.

References

[1] Chamberlin, D.D. and Boyce, R.F., SEQUEL: A structured English query language. In Proceedings of the ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control, (Ann Arbor, Michigan, 1974), 249–264.

[2] Microsoft. *ODBC Overview*. http://support.microsoft.com/kb/110093

[3] Sun Microsystems. *JDBC Overview*. http://java.sun.com/products/jdbc/overview.html

[4] Sceppa, D. *Programming Microsoft ADO.NET 2.0 Core Reference*. Microsoft Press, 2006.

[5] World Wide Web Consortium. XML Standard. http://www.w3.org/XML/.