Supporting Web-Based Database Application Development

Quan Xia¹  Ling Feng²  Hongjun Lu³

¹National University of Singapore, Singapore, xiaquan@comp.nus.edu.sg
²Hong Kong Polytechnic University, China, cslfeng@comp.polyu.edu.hk
³Hong Kong University of Science & Technology, China, luhj@cs.ust.hk

Abstract

This paper discusses our experiences of designing and implementing a pure JAVA database proxy server with a JDBC compatible driver for Intranet/Internet database application development. In particular, we present a shared database connection strategy and flexible caching facilities to address the scalability problem. Web clients with the same access privilege can maintain their logical connections with shared physical connections to the database. Thus, not only a large number of users can be entertained by limited physical resource, the connection cost is also not indulged for each individual client. Web clients can also express their cache requirements explicitly in the JDBC protocol so that a large number of clients can be served with improved response time. The effectiveness of such strategies is demonstrated through a set of experiments.

1. Introduction

During the past decade, the power and maturity of the database technology have made database systems and related applications most popular in every sector of the economy and whole society. Its effective data organizing, storing, and retrieving techniques play a critical role in the traditional information management field. However, it seems that the popularity of databases is taken over by the rapid development of the Internet technology and the World Wide Web (known as "WWW" or "Web"). The reason for the Web's success is largely due to its simplicity. It allows users to publish and retrieve information easily via the hypertext interface. It is compatible with other existing protocols, such as gopher, ftp, and telnet, etc. Moreover, it provides users with the ability to browse multimedia documents in an open environment available on many different platforms, requiring little cooperation between information providers and users. Although the information privacy and security issue of the Internet is still a major concern, organizations can reduce the risk using the Intranet, which provides the Internet services for internal use and uses gateways to connect to the Internet outside the organization. It is expected that the Internet and Intranet will be the major vehicle for future data and information exchange. On one hand, they free database application developers from the network maintenance. Once a database is linked to the Web, it is accessible from any Internet-link computer in the world. Moreover, because the Web browsers are available for almost all platforms, they eliminate the needs for designing different application interfaces across different platforms. Thus, more and more researchers and developers today are building Web-based database applications.

Different from traditional database applications which are built based on centralized or distributed database systems and usually for internal use of an organization, Web-based database application developers face a lot of challenges from such new environment. One of them is scalability requirement: it is often the case that a hot Web site is accessed by a huge number of users simultaneously. Thus, to increase the scalability of Web-based database applications, efficient control over users' database accesses is necessary.

In this paper, we describe our design and implementation of a pure JAVA database proxy server, jdWeb, with a JDBC compatible driver for Internet/Intranet database application development. Such proxy server acts as a middle-ware [3] to provide Web clients with accesses to various Database Management Systems (DBMS). A Secure Socket Layer (SSL) communication protocol is utilized to establish secure communications between clients and the server. To increase the scalability of the server, we adopt a shared database connection strategy and flexible client- and server- side caching facilities. Web clients with the same access privilege can maintain their logical connections with a shared physical connection to the database. Thus, not only a large number of users can be entertained by limited physical resource; the connection cost is also not indulged for each individual client. In addition, Web clients can express their cache requirements explicitly so that a large number of clients can be served with improved response time.

¹ On leave from the National University of Singapore
The remainder of the paper is organized as follows. Section 2 discusses the issues related to connecting Java applications to DBMS. To improve the scalability of Web-based database applications, we present a shared database connection strategy and flexible caching facilities in Section 3. Section 4 describes our design and implementation of a pure JAVA database proxy server, together with a JDBC compatible driver for Internet/Intranet database application development. Some preliminary experiments and results are described in Section 5. Finally, Section 6 concludes the paper with some discussions of future work.

2. Accessing Databases via the Web

Currently, most Web-based database applications access databases through CGI (Common Gateway Interface) programs [3]. CGI is a standard for external gateway programs to interface with information servers such as Web servers. A gateway program, also referred to as a CGI script, is executed at the server site, started by the Web server upon a client request. The gateway program receives the parameters supplied by the client, accesses the database through the underlying DBMS, and sends the results back to the server. The server then sends them to the clients who made the request. This approach provides a simple, easy-to-use method of executing programs from within an HTTP server. It can be used to create virtual documents and to interface with servers outside the normal HTTP server. The CGI interface has many advantages, including portability of server software, and wide availability of public domain gateway programs and development tools. However, since most gateway programs are external to the server, a process has to be created for each request with database accesses. Creating a process is time-consuming and expensive in terms of the server’s main memory, and can also exhaust resources available to server applications. More importantly, most gateway programs are specific, application dependent and out of the normal HTTP server. The CGI interface has many advantages, including portability of server software, and wide availability of public domain gateway programs and development tools. However, since most gateway programs are external to the server, a process has to be created for each request with database accesses. Creating a process is time-consuming and expensive in terms of the server’s main memory, and can also exhaust resources available to server applications. More importantly, most gateway programs are specific, application dependent and written in language such as C or C++. There are often the cases that new gateway programs have to be developed for new applications [6].

A recent trend in Internet programming is to use Java [9,14]. Java, being robust, easy to use, easy to understand, and automatically downloadable on a network, is an excellent language basis for Web-based applications. Apparently, Web-based database applications, the results of the combination of Java programming and database technology will make disseminating information easy and economical. Businesses can continue to use their installed databases to store their data and to publish web pages containing Java applets that use information obtained from those databases, which is certainly a better way of getting out information to external customers. Internally, businesses can connect their employees to those databases via an Intranet even if they are using different machines running different operating systems. Furthermore, development time for new applications is short. Installation and version control is greatly simplified. A new application or an update of an existing one can be put on the server, and everybody has access to it.

In order to facilitate the development of Web-based database applications using Java, a standard API, JDBC, has been defined [16]. JDBC consists of a set of classes and interfaces written in the Java programming language that provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API. Through JDBC, it is possible to establish a connection with a database, send SQL statements, and receive results. The basic function of a JDBC driver is to provide certain mechanism to make Java applications able to talk to a DBMS that are usually not written in Java. In principle, this can be achieved in four different ways.

1) The driver translates JDBC calls into ODBC calls, which is then processed by an ODBC driver. Since ODBC provides connectivity to almost all the commercial DBMS, this method provides multiple database accesses to the Java clients with one JDBC driver. However, the ODBC binary code, and in many cases database client code, must be loaded on each client machine that uses this driver. As a result, this kind of driver is at most appropriate on a corporate network where client installations are not a major problem. Besides, the performance degrade caused by ODBC, one extra layer between the clients and DBMS, is another consideration.

2) The driver translates JDBC calls into the native API calls of the DBMS to be connected. Most DBMS provide standard API calls so that applications can talk to the DBMS. However, this method also requires that some binary code be loaded on each client machine. Although the performance could be improved because DBMS API calls are usually more efficient than ODBC calls, a client who need access DBMS from different vendors must have different drivers and necessary binary code installed.

3) The driver directly translates JDBC calls into the network protocols used by DBMS to be connected. This approach is similar to the previous one. It allows a direct call from the client machine to the DBMS server and is an excellent solution for Intranet access. However, since many of these protocols are proprietary, the database vendors themselves will be the primary source for such drivers.

4) The driver translates JDBC calls into a DBMS-independent net protocol, which is then translated into a DBMS protocol by a server. This net server middle-ware is able to connect its pure Java clients to many different databases. The specific protocol used depends on the vendor.
Among the above four approaches, the first three attempt to connect database server directly from Java applets. In addition to the brief discussion mentioned above, there are a number of concerns over these approaches.

- It is difficult, if not impossible, to establish multiple database connections in an applet. Because of the security restriction of Java, a Java applet can only establish network connection from which it was downloaded. In such working scenery, database server and Web server must reside at the same machine. Multiple database connections can not be established from a single Java applet, unless all the database servers are resident on the same machine.

- JDBC drivers provided by the DBMS providers are based on native proprietary protocols, which may not be compatible with each other. Before database connections can be established, drivers must be available to the client side. This usually makes the applet over weighted, due to carrying driver packages; and sometimes a special local setup procedure is also required to setup the drivers.

- Since Java applet cannot exchange data with other local applications or access to the local file system, all data retrieved from databases must be processed by the applet itself, and can not be easily saved or transferred to other local applications for further process. However, in practice, such function is usually desirable. For example, an image retrieved from a database may need to be processed by some other applications rather than displayed by the browser directly.

- A database server usually has a limitation on the number of concurrent connections, either because of the hardware resource or software license restrictions. When a database is open to the Internet, it would be most likely to have a large number of concurrent users. Providing reasonable service to virtually infinite number of users with limited physical database connections could be an important issue.

- Most DBMS were not designed for the environment such as Internet. Although they have authorization mechanisms to control the access to the data stored in databases, they usually do not provide secured network communication mechanism for data transmission. Most of them use standard socket or remote procedure calls. When the database systems move to the Internet, such native DBMS communication protocol is vulnerable to hackers and eavesdroppers who can intercept and even alter the transmitted data.

- As mentioned above, the restrictions of Java allow Java applets establish connection only with the Web server from which it is downloaded. In order to establish direct connections, the database server has to be setup on the same machine of Web server, and this machine is usually put outside the firewall [10] of the company. Such arrangement makes the database server even more vulnerable to the outside intruders.

Comparatively, the fourth approach is the most flexible alternative. There is indeed one fundamental difference between this approach and the others. While other approaches follow a two-tier model, where a Java applet or application talks directly to the database, this approach uses services provided by a ”middle tier”, hence forming a three-tier, or multi-tier model. In addition to addressing the issues mentioned above, the extra middle tier also makes it possible to maintain control over multiple users’ accesses and the kinds of updates that can be made to the data. Besides, both client- and server-side cache mechanisms can be effectively realized as well. These facilities are indeed necessary to meet the high scalability requirement posed by the Web. In our system being developed, we employ such multi-tier architecture. Detailed access control and cache strategies are elaborated in the following section.

3. Improving the Scalability of a Database

Proxy Server

The Web constitutes a global information universe. Anyone can access the Web as long as he/she is connected to the Internet. It is often the case that a hot Web site is accessed by a huge number of clients simultaneously, which usually does not happen in traditional database applications. However, the number of clients that can be efficiently served by a database server is limited. In order to increase the scalability of databases opened to the Internet users, efficient control over users’ database accesses is necessary. In this section, we describe a shared database connection strategy and flexible caching facilities to address such scalability problem.

3.1 Two types of database connections

A Connection represents a session between a client and a specific database. Within the context of a connection, SQL statements are executed and results are returned. In addition to the normal dedicated connection between a Java client and Web-based database, we present another different kind of database connection, i.e., shared connection.

Dedicated connection

Dedicated connection is a proprietary database connection to Web clients. It is for a specific database that can be interacted with in a specific sub-protocol. When a client applies for such connection, a physical database
connection will be established, and the client can issue SQL statements to access the database. This connection will not be released until another disconnect statement is sent out by the client, or by a time-out mechanism to prevent a disconnected Web client from occupying the connection. The connection manager of the proxy server manages all aspects about a connection, including the data source identifier, the sub-protocol, the state information, the DBMS SQL execution plan ID or handle, and any other contextual information needed to interact successfully with the underlying DBMS. The implementation of the connection is transparent to the programmers.

Shared Connection

Connecting to a database is an expensive operation. Moreover, the number of physical database connections and the number of allowed concurrent usage of a DBMS are often limited, either because of license limitation or server resource limitation. But for Internet database applications, large number of concurrent users, which is much more than the number of available physical connections is usually expected. In order to serve large number of transactions with reasonable response time, we introduce another kind of connection -- shared connection. Web clients with the same access privilege (e.g. users who check the stock prices) can maintain their logical connection through a shared physical connection to the database. When a client requests such connection, no physical database connection will be created; instead, the client is allowed to use a shared connection pool. That is, when the client submits a transaction, an idle connection will be fetched from the connection pool and assigned to this client. After the transaction completes its execution, the allocated connection will be returned back to the common connection pool, though the client can logically maintains the connection to issue the next transaction. If there is no more connection available when a client's request arrives, the request will be added into a waiting list. Whenever free connection is available, it will be allocated to the request with highest priority in the waiting list. Figure 1 outlines the database operations under two connection schemes.

Shared connection strategy can perform quite well in such working scenes like large number of read-only requests, with similar data requirements and access privilege, etc. In reality, lots of Web-based applications nowadays have such characteristics.

Besides the advantage of improving response time, shared connection strategy also provides us a flexible way to control multiple users’ accesses from the Web. By dividing users into different groups and allocating physical connections to each group rather than individual user, we are able to prevent users of lower priority from occupying too many physical database connections and from competing connections with those users of higher priority.

**Database Operations for User U**

1. \( C := \text{Get\_Database\_Connection}(U); \)
2. \( \text{foreach} \) transaction \( T \) issued from user \( U \) \( \text{do} \)
3. \( \text{Process\_Transaction}(T, C); \)
4. \( \text{Disconnect\_Database}(C); \)

**Get\_Database\_Connection(U)**

5. if requested connection of \( U \) is dedicated then
6. \( \text{wait until a physical connection } C \) with the database is created;
7. else
8. \( U \) is assigned to a shared connection pool \( C \);
9. return \( C \).

**Process\_Transaction(T, C)**

10. if \( C \) is a dedicated connection then
11. execute transaction \( T \) through physical connection \( C \);
12. else
13. if there is no idle physical connection in the shared connection pool \( C \) then
14. \( T \) enter the shared connection pool \( C \)’s waiting queue;
15. else
16. get an idle physical connection from the pool \( C \);
17. execute transaction \( T \) through such idle connection;
18. return such connection to the pool \( C \) after completing \( T \);
19. check \( C \)’s waiting queue and serve its first transaction (if any) using any idle connection of the pool \( C \);
20. endif
21. endif.

**Disconnect\_Database(C)**

22. if \( C \) is a dedicated connection then
23. release physical connection \( C \);
24. endif.

---

**Figure 1. Connection scheme and database operations**

---

**3.2 Caching query results**

Caching query results is probably the most widely used methods to improve response time on Web application. Query results can be cached at the server side as well as the client site. Most servers cache the frequently accessed data based on certain algorithms. The client itself can also cache the remote data retrieved so that the subsequent use of the data can be fetched locally without requesting remote access again. However, in both cases, what data is to be cached is determined by certain hard-coded caching mechanisms. Furthermore, the cache invalidation is also conducted based on such mechanisms.
In order to provide better services to large number of clients, it would be better to adopt more flexible caching mechanisms. One possibility is to allow the database proxy server to accept some hints from the client applications. Such hints can suggest whether the query results should be cached or what copy, the database copy or cached copy should be returned. For example, a client application, where time efficiency is more important than accuracy, can give a hint that if there is a cached copy, the cached copy of data can be returned without a new database access. On the other hand, a client also can require that, the query results should be stored at server side for later access when it is needed.

In such case, instead of transferring the data back to the client directly, the proxy server will send back a URL that points to the cached file on the server to the client. The client can use this URL to retrieve the file at any time it wants, or transfer it to an external tool. In this way, the data can also be processed in an external application or saved to the local file system.

With such flexible caching facilities, the number of interactions between Web clients and the DBMS are reduced; hence more database applications can be supported. Moreover, the network congestion problem can also be alleviated.

4. Design and Implementation of jdWeb

To support Web-based database application development and improve the scalability of the database proxy server using above discussed multiple connection strategies and flexible caching mechanism, we designed and implemented a pure Java application package, jdWeb. It consists of two parts. On the server side, jdWeb is a web/database proxy server. As a middle-ware, it provides multiple database driver control, encryption and security, concurrency control and data caching service. On the client side, jdWeb comes with the jdWeb JDBC driver suitable for both Java applets and Java applications. Java developers can use the jdWeb JDBC driver to develop light-weighted Java applets to access multiple databases from Internet through jdWeb database proxy Server. The jdWeb JDBC driver can work with all Java enabled browsers and all versions of Java Development Kit (JDK). Figure 2 depicts the system architecture where jdWeb is used.

The major design objectives of jdWeb include the following:

- **Provide installation free, light weighted drivers for Java applets.** jdWeb provides a JDBC driver (jdWeb JDBC driver) for application developers. Instead of connecting physical databases directly, jdWeb JDBC driver translates JDBC calls into DBMS-independent net protocol, and sends it to the jdWeb proxy server; then jdWeb proxy server accesses the database through native DBMS protocol (or ODBC calls if the protocol is not available) and sends back the results. Such implementation not only reduces the network traffic, but also makes the Internet database application zero installation and the immigration and update of the application completely isolated from the users.

- **Provide multiple database connections.** Because the actual physical database connections are established on the jdWeb proxy server, there is no restriction on multiple database accesses to the clients. Internet clients can establish multiple connection with databases on the Intranet transparently.

- **Provide encryption and Security.** To enhance the security of data communication, jdWeb provides Secure Socket Layer (SSL) as the standard communication protocol, which enables client and server applications to select from a set of hybrid encryption to establish secure communication links.

- **Provide improved scalability.** To allow the server to provide services for a large number of Internet clients with limited physical database connections, jdWeb implements two types of database connections, dedicated database connection and shared database connection. With shared connection, users within a group with the same access privilege share the same set of physical connections while maintaining logical connection for better performance.

- **Provide client- and server- side caching.** jdWeb provides extended functions to cache certain query result on both client- and server-side. It allows transferring the data through standard URL connection. The data can then be more efficiently cached and retrieved.
transferred to external applications or saved to local file system.

In the following two subsections, we will describe the design and implementation of the jdWeb driver and the jdWeb database proxy server, respectively.

4.1 The jdWeb JDBC driver

The jdWeb JDBC driver is a set of API to help users developing Internet database applications with the jdWeb proxy server. As a pure Java software package that implements JDBC API, the jdWeb JDBC drive is platform independent and can be distributed via Web servers or installed as a local Java package in any client computers. Besides, it supports secure multiple database connections and shared physical database connections.

JDBC interface

The JDBC API [12] is a set of programming interfaces which are based on X/Open SQL Call Level Interface (CLI). Modeled after the Open DataBase Connectivity (ODBC) specification, the JDBC API contains a set of objects and methods for issuing SQL statements, table updates, and calls to stored procedures. The basic components (classes or interfaces) are:

1) DriverManager, a static class capable of managing multiple JDBC drivers in a Java Virtual Machine execution environment.
2) Driver, an interface for all JDBC drivers capable of accessing one or more database systems.
3) Connection, an abstraction of a client/server database connection.
4) Statement, an abstraction of a database query or operation.
5) ResultSet, an abstraction of the database query results.
6) DatabaseMetaData, an abstraction of the information about database.
7) ResultSetMetaData, an interface to the information of the result set itself.
8) A few other classes and interfaces that represent other aspects of a database system.

The jdWeb JDBC Driver is implemented in full compliance with the JDBC API. However, there are additional features in the jdWeb JDBC Driver that extend and complement the capability of the JDBC API. These extensions are mainly to support shared database connections, and client- and server- side caching functions as mentioned in the previous section.

Extensions to JDBC interface

jdConnection class provides support for both dedicated connection and shared connection. Once a successful connection is established, a client can interact with the data source using SQL statements. In JDBC, there are four main types of statements: jdStatement, CachStatement, jdPreparedStatement, and jdCallableStatement. The jdConnection object has the createStatement(), prepareStatement(), createCachstm(int cachsize), and prepareCall() methods to create these statement objects. When a client applies for a special class statement, CachStatement, the driver spawns a new Java thread for this CachStatement. The thread, running in the background, establishes a separate network connection with jdWeb proxy server, and fetches the ResultSet from database. When a CachStatement is executed, the jdWeb proxy server will transfer the result set to the client, which will then be cached at the client side, as indicated by the client. In addition, jdWeb driver provides new method getURL( ) in jdResultSet to allows clients to specify whether to cache the current retrieved data at the server side. Later the client application can use this URL to retrieve the file, or transfer it to an external tool through Web browser external helper by using method AppletContext.showDocument (URL).

4.2 Server side implementation: jdWeb proxy server

jdWeb proxy server is the server side implementation of jdWeb package. It is the middleware between the Internet database application and the database server. The jdWeb proxy server translates the DBMS independent calls from JDBC driver into a DBMS request, and sends back the result to the JDBC driver. The jdWeb proxy server is able to connect all Java clients to different databases, through either ODBC or native JDBC driver. To support extended JDBC functions of jdWeb, it also provides concurrent connections and server side cache facilities. It has four main parts: connection manager, concurrent manager, security manager and administrator interface.

Connection manager

The Connection Manager administrates the connections with the client side application. Once the connection manager receives a connection request from the client side, it will first identify the connection type. If the connection requested is a dedicated connection, it will swamp a threat, and try to establish a database connection for it. If the connection request is a shared connection, the connection manager will check the authentication to make sure that the request comes from an authorized group user. If the check passes, connection manager will confirm the connection to the client, and wait for transaction requests from the user. In other words, the
connection manager implements the major functions outlined in Figure 1.

**Concurrent manager**

The function of the Concurrent Manager is to maintain shared database connection among a group of jdConnection users using shared connections. It allows system administrator to define groups, and to allocate a certain number of database connections to each group with certain access right. When the jdWeb proxy server starts, the concurrent manager will first check the configuration file, and set up a concurrent connection stack for each shared connection group. For each stack, a number of database connections is allocated. When a request from a client arrives, the concurrent manager will allocate a connection to it if there is any unoccupied connection available in the stack, otherwise it will block the request until a connection is available. When the client's statement completes its execution, the connection is returned to the concurrent manager.

**Security manager**

The Security Manager has two tasks. The first one is to verify whether the IP address of the client is an authenticated one. The jdWeb proxy server allows system administrators to define lists of IP addresses from which access is permitted or forbidden. For example, if a system administrator want to prohibit all accesses from 137.202.192.xx, he can just simply add "137.202.192.xx" to the forbidden list. Then the security manager will reject any access request from this network section. Such forbidden and access list is stored in a configuration document, and is loaded when the jdWeb proxy server is initialized. The second function of security manager is to maintain information of users' authentication. As mentioned earlier, shared connections are provided for group users. The security manager allows administrator to define a logical group for database users who will share some common connections.

**Administrator interface**

The administrator interface is the console of a jdWeb proxy server. Through it the administrator can monitor the execution of the server such as the connection time, and allows administrator to configure the server (e.g., define the groups, allocate connections to different groups, etc.). Administrator interface also allows the administrator to define the log level and check log file.

**5. Effectiveness of the Shared Connection Scheme**

Both the database proxy server and the extended JDBC driver have been implemented and under testing. While comprehensive field-testing is still undergoing, results from some small scale preliminary experiments are promising. The database proxy server is implemented on a Pentium/200 PC with 64M RAM. The DBMS used is Ms SQL server 6.0.

In order to study the effectiveness of shared connection, experiments were conducted with different configurations. Two parameters were used in the experiment setting. The first is the ratio between the number of concurrent clients and the available physical connections, e.g. the average number of clients per physical database connections available. Another parameter is the size of each application, measured by number of transactions, \( N_T \). Here we assume all transactions are similar queries to the database.

The elapsed time for client applications with different configurations was measured and shown in Figure 3. The results indicated that shared connection does provide benefits when the number of concurrent client applications is larger than the number of available physical connections. In Figure 3, when number of concurrent clients is equal to the available physical connections, client applications can be completed within the same period of time for both shared connection and dedicated connection mechanism. If the number of concurrent clients is larger than the number of available connections, applications running under the shared connection scheme completed faster than under the dedicated scheme. This is expected, as under dedicated scheme, some of the client applications have to wait for being connected before their execution.

Figure 4 depicts the maximum waiting time for a client. When each client has one line, the client application can be connected to the database right after its request. With the dedicated scheme and less number of physical connections, a client has to wait until one of the other clients releases a line. The waiting time is proportional to the number of concurrent client applications and the length of their transactions. If all the clients are working in the shared scheme, the waiting time will increase when the number of concurrent clients increases. However, the increment is lesser and not affected by the total execution time of other client applications.

![Figure 3: Connection Scheme and Elapsed Time.](image-url)
6. Conclusion

In this paper, we described our work on designing and implementing a pure Java middleware, the **jdWeb**, for developing Web-based database applications. The implementation has been completed and experiments are being conducted to study its performance. The initial tests indicate the following:

- The jdWeb proxy server and jdWeb JDBC driver provide developers a secure, robust and flexible way to develop multi-tier Internet database applications. An application running at the client side can use the single jdWeb driver to access multiple databases through the jdWeb proxy server.

- Extended features in the driver and server support shared connection for a group of users. It can effectively handle a large number of users to use a few physical database connections simultaneously and provide necessary control over multiple database accesses.

- Besides the advantage of improving the scalability of Web-based database servers, the server side caching facility can be used to break the constraint of Java applet, allowing the data retrieved from the database to be saved or transferred to other external tools and processed on the client side.

- The **jbWeb** provides Secure Socket Layer (SSL) as the standard communication protocol for the Internet database applications. SSL enables client and server applications to select from a set of hybrid encryption to establish secure communication links.

In addition to performance evaluation, there are still a number of issues to be addressed to provide seamless connection between Java applications and DBMSs. For Internet-based applications, scalability is still a major issue. The current implementation of shared connection in jdWeb can be further improved by carefully allocating physical connections among various kinds of Internet/Intranet user groups. The caching and cache invalidation mechanisms are also a topic to be further studies.

**References**

5. JavaSoft (Sun Microsystems), JDBC Drivers API, 1997.
9. B. Nance. Examining the Network Performance of JDBC.