SQLJ: Embedded SQL in Java

An Oracle Technical White Paper

May 15, 1998

Introduction

During the past year and a half, Java™ has become the language of choice for developing and deploying Internet/Intranet applications. Its promise of a unified portable application development solution that can execute on a simpler, lower cost network-centric IT infrastructure has prompted major development tool vendors and infrastructure providers to provide support for Java. Client-side Java interfaces and tools are being complemented by highly productive, scaleable, reliable, and performant Java servers as leading software vendors such as Oracle integrate Java Virtual Machines with their databases and other application platforms. Leading browser platforms are building Java Virtual Machines into their systems. Java *applets* and *servlets* are appearing all over the Web, bringing rich functionality to what was before only a static medium. Java's value proposition is so compelling that many large companies are already beginning to re-architect their business critical applications in Java.

This paper provides a technical overview of SQLJ, a new language that has emerged as a result of a multi-vendor effort to provide support for static SQL in Java programs. It does so in 5 sections:

- The first section provides a brief review of the **promise of Java and Oracle's Java strategy**, with an overview of the SQLJ product.
- The next section highlights **SQLJ's design goals, how it works, its most important features,** and how it compares with dynamic SQL APIs such as JDBC.
- The third section describes **the various configurations** in which SQLJ can be used.
- The fourth section summarizes **SQLJ's benefits.**
- Finally, the fifth section provides a clear description of **Oracle's future plans for SQLJ.**

1. The Promise of Java and Oracle's Java Strategy

Java has raised considerable interest both within the application developer community and within the corporate CIO community, because it promises to provide both a very productive application development language and the potential to significantly simplify and reduce the cost structure of the information technology infrastructure within companies today.

1.1 Java's Interest to Software Developers

Software developers are attracted by Java's power and productivity as an application programming language.

- **Modern, Object-oriented:** Java is provides a modern, object-oriented programming language with features such as garbage collection that make application development simpler and quicker than with other programming languages such as C++.

- **Component Model:** It provides component models - Java Beans and Enterprise Java Beans - which allow users the facility to very easily assemble and distribute applications using off-the-shelf components. As Java tools and Java servers are increasingly providing a rich development and execution environment for these components, they will become widely used in both client-side GUI development and server side component creation and distribution.
- **Internet-centric:** Most importantly, Java was designed from the ground up to facilitate Internet-centric application development with facilities to allow transparent partitioning and distribution of application components across a network. Java is, therefore, very rapidly becoming the predominant language to build enterprise applications.

1.2 Java's Interest to CIOs

While Java has attracted the attention of the software development community, it has also raised considerable excitement among Fortune 500 Chief Information Officers [CIOs]. CIOs are attracted by Java's promise of:

- **Lower cost I/T infrastructure:** By offering downloadable applets that execute in a browser, Java eliminates the need for pre-installation and configuration of software on a client, thereby, considerably lowering the cost of software maintenance.
- **Broad choice of products:** CIOs are also attracted by the growing variety of low cost tools, Java servers, and clients that are coming to market offering them a broad array of choices at modest prices.
- **Openness:** Finally, Java's openness and portability allows CIOs to train their developers on a single programming language and tool set while ensuring that they do not face proprietary lock-in from any single software vendor.

1.3 Oracle's Java Strategy

Oracle's Network Computing Strategy brings the Internet model of computing to enterprise applications. Oracle has had a significant Java project underway aimed at providing **a complete Java solution** to its enterprise customers including an enterprise class **Java server platform** and a highly productive set of **Java development tools.** Oracle Data Server Technologies, the division that builds Oracle's industry leading database server, has a number of projects underway to deliver client-side programmatic interfaces and Java integration with the Oracle database.

- **Client-side programmatic interfaces**: Oracle will offer two different client-side programmatic interfaces for Java developers - JDBC and SQLJ.
    - **JDBC Drivers**: It offers two different JDBC drivers - one for developers writing client-server Java applications or Java-based middle tier servers, the other for those writing Java applets. A specialized version of JDBC is also being developed to execute within the database server, allowing Java applications that execute on the server's Java VM to access data defined locally [i.e., on the same machine and in the same process] via JDBC.
    - **SQLJ**: Oracle is working with a number of partners to develop SQLJ - a standard way to embed SQL statements in Java programs. SQLJ offers a much simpler and more productive programming API than JDBC to develop database applications in Java.
- **Server**: Oracle is delivering a Java VM that is tightly integrated with, and leverages the performance and scalability advances of, the Oracle8 server to offer an ideal platform for traditional OLTP applications, intranet and internet-based applications, and e-commerce applications. The Oracle Java VM will support two different programming models: Integration with SQL allowing users to write traditional database stored procedures, triggers, and methods in Java. The Java VM will also provide a transaction server platform for distributed Java

components called Enterprise Java Beans that allow programmers to develop reusable server-side application components in Java. Browser-based or middle-tier Java or CORBA clients can communicate with server-side EJB components via an object-based protocol such as IIOP [CORBA's Internet Interoperability Protocol]. Oracle's offers a common set of services between this platform and its Application Server's Java platform.

- **Strategy**: In delivering its Java solution to customers, Oracle is focused on three primary differentiators:
  - A complete solution consisting of both a Java server platform and an integrated set of Tools
  - Enterprise Quality Platform: Fast, scaleable and operationally robust
  - Standards compliant: All of Oracle's products are 100% compatible with Java standards.

2. SQLJ - A Brief Overview and Description

Having provided an overview of Oracle's Java strategy in general, this section provides considerably more detail on SQLJ - an important component in this strategy. The section is divided into three parts:

- First: It describes SQLJ and its design goals
- Second: It describes how SQLJ works
- Third: It provides an overview of the most important features of SQLJ
- Fourth: It compares SQLJ code with equivalent JDBC code

2.1 SQLJ: The Product and Design Goals

SQLJ provides a standard way to embed SQL statements in Java programs. In writing a SQLJ application, a user writes a Java program and embeds SQL statements in it following certain standard syntactic rules governing how SQL statements can be embedded in Java programs. The user then runs a SQLJ translator which converts this SQLJ program to a standard Java program, replacing the embedded SQL statements with calls to the SQLJ runtime. The generated Java program is compiled using any Java compiler and run against a database. The SQLJ runtime environment consists of a thin SQLJ runtime library which is implemented in pure Java, and which in turn calls a JDBC driver targeting the appropriate database.

SQLJ is, therefore, similar to the ANSI/ISO "Embedded SQL" standards, which prescribe how static SQL is embedded in C, COBOL, FORTRAN, and other languages. For example, Oracle's pre-compiler product *PRO*C* is an implementation of the Embedded SQL standard in the host language C - those familiar with the Oracle pre-compilers may think of SQLJ as if it were *PRO*Java.* For Java developers who are unfamiliar with embedded SQL interfaces to databases, there are two important facets to SQLJ that require further exploration:

- **SQLJ is a Standard:** The Language Specification is a joint specification supported by all the leading database and database-tools vendors including Oracle, IBM, Sybase, Informix, Compaq/Tandem, and Javasoft. All these vendors have also cooperated to provide a Reference Implementation of the SQLJ Translator to ensure that the SQLJ implementations from the different database vendors are compatible and interoperable.
- **Design Goals for SQLJ:** SQLJ was primarily designed to significantly improve the productivity of Java developers building database applications. To facilitate this, its definition was focused around a few important design goals:

- *Compact and high-level interface*: SQLJ provides application programmers with a higher level interface resulting in more compact and error free code than JDBC for static SQL. SQLJ does not address dynamic SQL - users can however combine both SQLJ and JDBC in a single application to address both static and dynamic SQL respectively.

- *Translation-time syntax and semantic checking of SQL statements:* For static SQL, the SQLJ translator performs type-checking and schema-checking to detect syntax and semantic errors in SQL statements at program development-time rather than runtime. Programs written in SQLJ are therefore more robust compared to JDBC programs.

- *Multi-Vendor Interoperability*: SQLJ Applications were designed to be vendor independent in three important ways: [i] First, the syntax for embedding SQL in Java programs is a standard shared by all the SQLJ partners. [ii] Second, the partner's share a common SQLJ translator reference implementation. [iii] Third, since a SQLJ program performs database access using JDBC at runtime, SQLJ can access any data server for which JDBC drivers are implemented.

- *Vendor-specific customization:* The SQLJ standard also allows vendor-specific customizations - a binary SQLJ application includes a set of SQLJ *profiles* that describe the SQL operations appearing in the original program source. These profiles can be used to create vendor-specific customizations that can be installed into the binary SQLJ application. Two types of customizations are possible: [i] Profile customizations to improve SQL execution performance using optimization techniques. [ii] Customizations to grant access to vendor-specific features not otherwise available to SQLJ programs. Multiple customizations can be installed into the same SQLJ binary, so that the same binary can be used to execute SQL on databases from different vendors, and the execution of that operation will take advantage of the customization available for each vendor.

- *Flexible Deployment:* The code generated by the SQLJ translator is 100% standard Java code and can be executed in any standards-compliant Java Virtual Machine. This allows SQLJ programs to be partitioned easily across different tiers in a distributed architecture, and deployed in many different environments without any code changes.

2.2 SQLJ: How it Works

A SQLJ Application is a standard compiled Java program that was prepared from SQLJ source files. A SQLJ program is typically compiled in two steps. In the first step, a *SQLJ Translator* translates the SQLJ application into a Java application with calls to the SQLJ runtime replacing the SQLJ clauses - its output is a set of standard Java source files. In the second step, any Java compiler can be used to compile those Java files.

2.2.1 SQLJ Translator

A *SQLJ Translator* performs two important functions:

- **Translates the SQLJ source code**: The *SQLJ Translator* translates the SQLJ application into a Java application with calls to the SQLJ runtime replacing the SQLJ clauses, generating a set of standard Java source files.

- **Type Checking:** It performs SQL syntax-checking, schema-checking, and type-checking of host variables at translation time, if logon information to the database is provided. This checking is performed *statically*, i.e., all SQL statements in the program are checked irrespective of the actual code paths executed at application runtime.

Reference Implementation: The SQLJ partners (Oracle, IBM, Sybase, Informix, Compaq/Tandem, JavaSoft, and others) have collaborated to produce a *Reference Implementation* of a SQLJ Translator. The SQLJ Reference Implementation is written entirely in Java, and is designed so that it can be incorporated into Java development tools which will allow SQLJ translation and Java compilation to be performed in one step.

2.2.2 SQLJ Runtime

A compiled SQLJ application is a standard Java program and can run wherever a Java VM, the SQLJ runtime library and a JDBC driver are available. There are three important aspects to consider regarding executing SQLJ applications:

- **SQLJ Runtime:** At runtime, a SQLJ application communicates with a database through the SQLJ runtime library which is a thin layer of pure Java code above a JDBC driver. A variety of JDBC drivers can be used at runtime - the figure below illustrates the three alternative runtime configurations of a SQLJ program using a variety of different JDBC drivers to target Oracle.

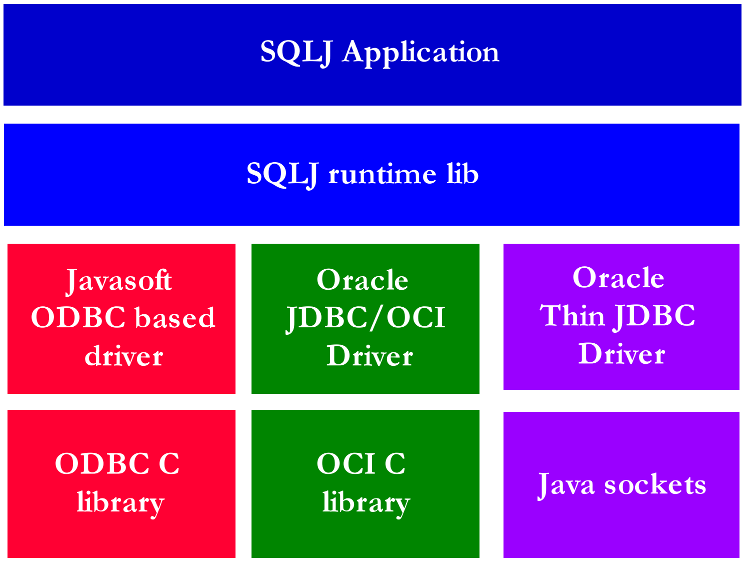

Figure: SQLJ Runtime Configurations

- **Type Safety:** SQLJ associates the properties of result-sets and database connections with the *types* of the Java objects that represent them, so those types can appear in the interfaces between separately developed Java components. For example, the shape of rows of an iterator object (the number and types of the columns) is encoded by its type (its class). That iterator-class can appear as the type of parameters and results in the interfaces between software components. Therefore, components can exchange SQL result-set data as type-safe, first-class objects with known row shapes, so that the SQLJ translator and Java compiler can check that accesses of column data are correct wherever it is used.
- **Binary portability:** Applications translated by the SQLJ Reference Implementation can access any database that is supported by an implementation of JDBC or a compliant implementation of the SQLJ runtime API. This property of *binary portability* allows compiled applications to be portable not only across platforms, but also across different vendors' databases. For example, a SQLJ Application, that contains SQL conforming to SQL92 Entry Level, augmented by

BEGIN..END blocks and calls to stored procedures, can connect to an Oracle, IBM, Tandem, or Sybase, database and interact successfully with any of these databases.

2.3 SQLJ Features

Having described what SQLJ is and how it works, let us now examine its most important features using an example. There are 5 important features each of which are described in more detail below.

- SQLJ Clauses
- Host Variables
- Result Sets Returned by Queries
- Database Connection Management
- Combining static and dynamic SQL: using SQLJ and JDBC together

2.3.1 SQLJ Clauses

Static SQL statements appear in a SQLJ program text as *SQLJ clauses*. A SQLJ clause is introduced by the token #sql, and contains a SQL statement inside curly braces. An executable SQLJ clause may appear where a Java statement may appear. Here is a SQLJ clause that contains a SQL UPDATE statement:

```
  #sql { UPDATE TAB SET COL1 = :x WHERE COL2 > :y AND COL3 < :z };
```

2.3.2 Host Variables

The inputs and outputs of SQL statements are passed through *host-variables*. A host-variable is a Java variable, parameter, or field that is embedded in a SQL statement, prefixed by a colon character. The standard JDBC types, such as boolean, byte, short, int, String, byte[], java.sql.Date, Integer, Double, etc. are valid host variable types in SQLJ. In addition, Oracle's SQLJ translator supports Oracle7 and Oracle8 specific types, such as ROWID, CLOB, BLOB, as well as Object and REF types.

The following example consists of two SQL table definitions, and a Java method containing SQLJ clauses that access those tables.

```
CREATE TABLE PARTS_MASTER
    (PART_ID NUMBER(8) PRIMARY KEY,
     PART_NAME VARCHAR(40),
     SUPPLIER VARCHAR(200));

CREATE TABLE MRP
    (PART_ID REFERENCES PARTS_MASTER,
     QUANTITY_ON_HAND NUMBER(6),
     REORDER_THRESHOLD NUMBER(6));


// Part of a SQLJ program, showing definition of one method:
  public class inventory {
  ...
    public void pullStock (int part, int quantity) throws OutOfStock {

      int on_hand, threshold;

      #sql { SELECT QUANTITY_ON_HAND, REORDER_THRESHOLD
               INTO :on_hand, :threshold FROM MRP
               WHERE PART_ID = :part FOR UPDATE };
      on_hand -= quantity;

      if (on_hand < threshold) {
        String supplier;
         #sql { SELECT SUPPLIER INTO :supplier FROM PARTS_MASTER
               WHERE PART_ID = :part };
        inventory.orderMore(part, quantity, supplier);
      }

      if (on_hand < 0) {
        #sql { ROLLBACK };
        throw new OutOfStock();
      } else {
        #sql { UPDATE MRP SET QUANTITY_ON_HAND = :on_hand
               WHERE PART_ID = :part };
         #sql { COMMIT };
      }
    }
  ...
  }
```

The above example shows that SQLJ is quite similar to ANSI/ISO Embedded SQL. It allows SQL statements to appear directly in program logic. At program development time, static analysis can detect errors in their SQL syntax, in their uses of tables and other schema definitions, and in their numbers and types of arguments and results.

2.3.3 Result Sets Returned by Queries

In a SQLJ program, a result-set returned by a multi-row query is manipulated by means of an *iterator* object, that iterates through the rows in the result-set. An iterator is an object of an *iterator-class*, which is a Java class that is defined by a declarative SQLJ clause that may appear where a class definition may appear. The clause defining a named-iterator class lists the Java names and types for columns in a row of a result-set. The following clause defines an iterator-class called AllStock:

```
  #sql iterator AllStock (String part, int quantity);
```

The above clause implicitly defines Java class AllStock with methods named part and quantity, of types String and int, respectively. Those *column-accessor methods* return the values of columns from rows of a result-set contained in an iterator of type AllStock. The following SQLJ program fragment defines a local variable of class AllStock, executes a query to populate that variable with an iterator object, and calls the column-accessor methods of the iterator and prints the column values:

```
  public void printStock () {
    AllStock iter;
    // Instantiate the iterator with a SQL query
    #sql iter = { SELECT PART_NAME AS "part",
                         QUANTITY_ON_HAND AS "quantity"
                  FROM PARTS_MASTER, MRP
                   WHERE PARTS_MASTER.PART_ID = MRP.PART_ID };
    // Now loop through the result rows
    while (iter.next()) {
      System.out.println("Part: " + iter.part() + ", Quantity: "
                         + iter.quantity());
    }
    iter.close();
  }
```

In addition, SQLJ provides support for defining *positioned iterators* that use traditional FETCH…INTO syntax to access query columns by position.

2.3.4 Database Connection Management

The example above contains no explicit management of database connections. Its SQL statements execute on the default database connection, which is global to the program. SQLJ programs may also manipulate multiple database connections. Users may explicitly declare a *connection-context* class, wherever a Java class declaration is permitted. For example:

```
#sql context PartsDB;                    // declares a class PartsDB
PartsDB pdb = new PartsDB( url, user, password );
```

Each SQLJ clause may designate a particular instance of a connection context, in square brackets, immediately following the token #sql, for example.

```
#sql [pdb] { INSERT INTO MRP VALUES (:id, :quantity, :threshold) };
```

Multiple connection context classes can be used in the same program to partition SQL statements that are executed on different schemas. At translation time, different SQL checking can be performed for each connection context class used.

2.3.5 Combining Static and Dynamic SQL - SQLJ and JDBC

A SQLJ program may contain both SQLJ clauses and JDBC calls, for static and dynamic SQL, respectively. The two paradigms interoperate at the level of database connections and result sets/iterators. For example, a SQLJ connection context can be initialized with an existing JDBC connection:

```
java.sql.connection conn = …;
PartsDb pdb = new PartsDB(conn);
```

It is also possible to extract a JDBC connection object from a SQLJ connection context. Similar conversions are possible between JDBC result sets and structured SQLJ iterators. For example:

```
AllStock iter;
#sql iter = { SELECT … };
java.sqlj.ResultSet rs = iter.getResultSet();
```

Thus, the dynamic SQL API for SQLJ is simply JDBC.

2.4 SQLJ code versus JDBC code

For SQL statements with input arguments, SQLJ clauses are often shorter than the equivalent dynamic SQL (JDBC) calls, because SQLJ uses host variables to pass arguments to SQL statements, while JDBC requires a separate statement to bind each argument and to retrieve each result. Contrast SQLJ and JDBC program fragments for the same single-row SELECT statement:

```
// SQLJ
  float w; java.sql.Date x; int y; String z;
  ...
  #sql { SELECT C1, C2 INTO :w, :x FROM TAB WHERE C3 = :y AND C4 = :z };


// JDBC
  float w; java.sql.Date x; int y; String z;
  ...
  PreparedStatement s = connection.prepareStatement(
    "SELECT C1, C2 FROM TAB WHERE C3 = ? AND C4 = ?");
  s.setInt(1, y);
  s.setString(2, z);
  ResultSet r = s.executeQuery();
  r.next();
  w = r.getFloat(1);
  x = r.getDate(2);
  r.close();
  s.close();
```

Unlike dynamic SQL, SQLJ permits compile-time checking of the SQL syntax, of the type compatibility of the host-variables with the SQL statements in which they are used, and of the correctness of the query itself with respect to the definition of tables, views, stored procedures, etc. in the database schema. Type-checking and schema-checking is also done where column-data is fetched from an iterator object (by a FETCH statement, or by column-accessor methods), because the class of the iterator object defines the number and types of columns in rows contained by that iterator.

3. SQLJ Deployment Configurations

Having understood how SQLJ works and what its most important features are, it is now appropriate to consider how SQLJ programs can be deployed in a variety of different configurations. This section is divided into two parts: first, it discusses platform requirements to run a SQLJ program; next, it discusses five different scenarios in which SQLJ can be deployed.

3.1 Requirements for SQLJ Deployment

The only requirements from a platform point of view to run a SQLJ program are the availability of:

- The SQLJ runtime library
- A JDBC driver - Oracle's SQLJ Translator can be used with any JDBC driver including JDBC/ODBC bridges, Oracle's JDBC/OCI driver, or Oracle's Thin JDBC Driver
- A Java Virtual Machine where SQLJ programs will execute.

3.2 Deployment Scenarios

SQLJ Programs can be deployed in a number of different configurations including either fat or thin clients, in middle tier Java web servers or application servers, or as stored programs on the Java Virtual Machine integrated with the Oracle8.1 database server. Since the SQLJ runtime library is a thin layer of pure Java code that sits above the chosen JDBC driver, the user must choose the JDBC driver best suited for the particular deployment configuration he needs. The remainder of this section illustrates how Oracle's SQLJ Translator can be used in combination with Oracle's own JDBC drivers in five different deployment configurations:

- For traditional two-tier client-server Java applications
- For multi-tier applications with a middle tier application server
- For Java Applets
- With Oracle's Net8 Connection Manager
- For Oracle8.1 Database Stored Programs

3.2.1 SQLJ for Client-Server Java Applications

Oracle's JDBC/OCI driver is targeted to two kinds of application developers - those who are beginning to use Java for traditional client-server application development in an Intranet environment and those who are choosing to develop Java-based middle tiers such as Java application servers. SQLJ can be used with Oracle's JDBC/OCI driver in a traditional two-tier client-server using SQLJ in combination with Oracle's JDBC/OCI Driver. The Java application makes calls to the SQLJ library which in turn translates calls via the JDBC/OCI driver across SQL*Net to communicate with Oracle Database Server. In this configuration, users will need to deploy the following libraries on each client:

- SQLJ Runtime Library
- JDBC/OCI Driver
- Oracle Client Libraries including OCI, SQL*Net and CORE libraries

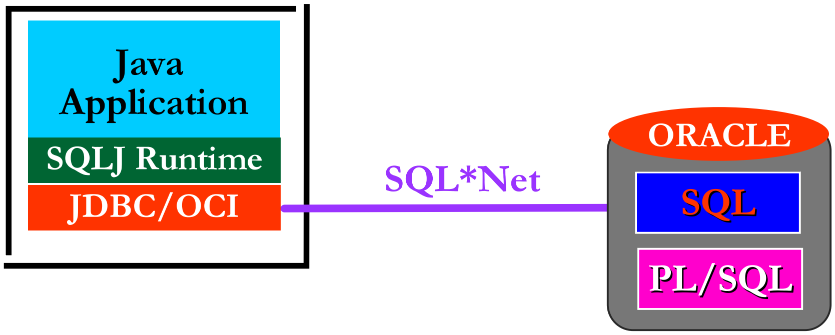

Figure: Using SQLJ in a Two Tier Client Server Configuration

3.2.2 SQLJ for Multi-tier Applications

In a 3-tier configuration, a browser-based client communicates with a Java-based middle tier either using a stateless protocol such as HTTP today or a stateful protocol such as IIOP in the future. The middle tier in turn executes the Java application logic written in SQLJ and communicates with the back end database server. The following diagram indicates how SQLJ can be used in combination with Oracle's JDBC/OCI Driver in a three-tier configuration either in an Intranet setting or an Extranet setting [where the web server is located behind the firewall].

The client is a web browser that communicates with the web server using the HTTP protocol. The Java application [the executable], the SQLJ runtime library, the JDBC/OCI driver, and the OCI, CORE and SQL*Net libraries are installed on the web server. The user can start/invoke the Java application in a number of different ways: The user can use a CGI script, Oracle's Web Request Broker API [if the application is deployed on Oracle's Web Application Server], or even an IIOP style invocation. Once the Java application has been invoked, it communicates with the backend database server using SQL*Net.

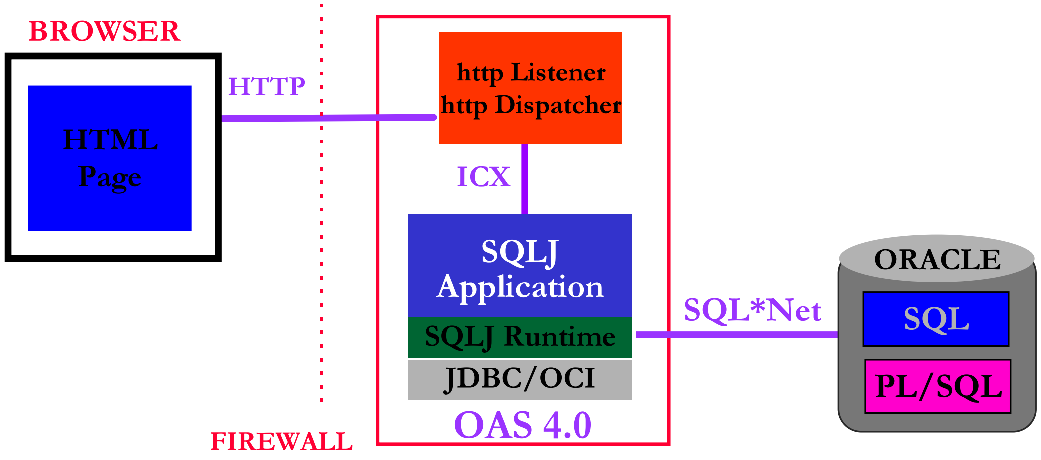

Figure: Using SQLJ in a Three Tier Architecture

There are three important deployment issues that arise from this form of application deployment - how to manage application state, how to improve the scalability of the application, and how to optimize application responsiveness. Users are directed to Oracle's JDBC technical white paper which discusses these issues in considerably greater detail.

3.2.3 SQLJ for Java Applets

SQLJ programs may be run as Java applets - Java applets execute in a browser and therefore cannot require any client-side installation. From the browser, the user simply clicks on a Uniform Resource Locator (URL) in a HTML page that contains a Java applet tag. When the user loads the page, the Java applet (which can be a translated SQLJ program), the SQLJ runtime library (which is 100% Java), and the necessary JDBC driver are downloaded into the browser from the web server. Once they are in the browser and begin executing in the browser's Java VM, the compiled SQLJ code establishes a connection with the database server via the JDBC driver. Since it needs to be downloadable along with the Java applet and the SQLJ runtime library, the JDBC Driver also needs to be a pure Java implementation. It also needs to be relatively small in size so that it can be downloaded quickly and execute in a performant manner on browser-based Java VMs.

Oracle's SQLJ Runtime and the Thin JDBC driver have been designed explicitly for such implementations. By eliminating the need for a client-side installation, Oracle's downloadable SQLJ Runtime and its Thin JDBC driver reduce the costs associated with maintenance and administration in an Intranet environment. They are even better suited for Extranet applications where client-side installations preclude the universal access that is fundamental to the success of the worldwide web. The following diagram shows the typical configuration in which the SQLJ Runtime and the Thin JDBC driver can be used with Java applets. The Java applet, SQLJ runtime library and Thin JDBC are downloaded into the browser after the user opens a URL. The SQLJ application communicates directly with the database server using SQL*Net. The web server and the database server can be on physically separate machines or on the same machine. In an Intranet deployment the entire configuration is behind a firewall; in an Extranet deployment, the web server and the database are both behind the firewall.

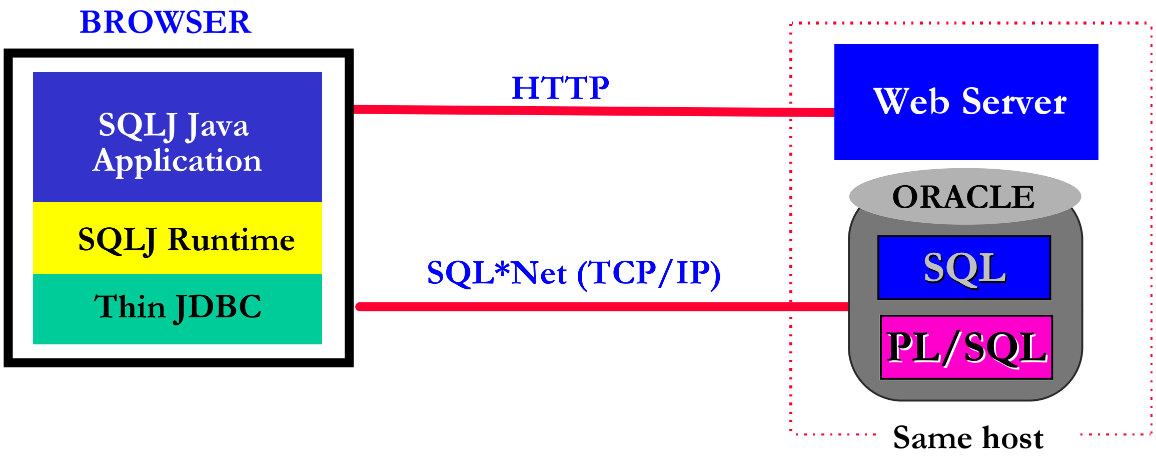

Figure: Using SQLJ with Thin Clients

The deployment configuration described above raises three important issues - how to handle Java security, how to manage application state, and how to support large numbers of browsers connecting in this two tier configuration. These issues are discussed in considerably greater detail in Oracle's JDBC Technical Whitepaper to which the user is referred.

3.2.4 SQLJ with Net8 Connection Manager

The SQLJ Translator and the Thin JDBC Driver can be used together with the Net8 Connection Manager to achieve significantly greater scalability in a two-tier or three-tier configuration on platforms with physical number of end point limitations. Oracle Connection Manager is a multipurpose networking service for Oracle environments providing client connection concentration, client connection access control, and multiprotocol connectivity. Oracle Connection Manager enables large numbers of users to connect to a single server by multiplexing multiple client database sessions across a single network connection thereby reducing internal memory usage within the database server [the server allocates memory internally to handle each connection or typically establishes a pool of connections that it swaps between a number of active connections]. By reducing the resource requirements within the database server, Oracle Connection Manager allows the Oracle database server to support a large numbers of concurrent users.

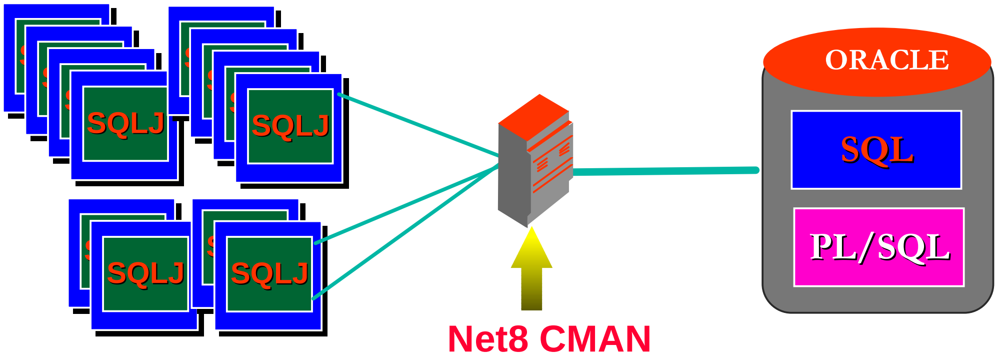

Figure: Using SQLJ with Net8 Connection Manager

3.2.5 Writing Database Stored Procedures in SQLJ

Oracle8.1 tightly integrates a highly scaleable and high performance Java Virtual Machine with the database kernel. It provides an ideal platform on which to run data intensive Java application logic. A key element of the Java VM's architecture is an efficient interface through which Java programs can access SQL and PL/SQL within the database. The Java VM provides two ways with which users can write such queries:

- **Embedded SQLJ Translator:** For static SQL queries, users can write embedded SQL in Java application code. The Java VM includes the SQLJ translator and Java compiler to transparently compile embedded SQL in Java programs to Java binaries. The Java VM also integrates the SQLJ runtime library which uses an embedded JDBC Driver to communicate with SQL and PL/SQL.
- **Embedded JDBC Driver:** Those choosing to write dynamic SQL can directly target JDBC - the Java VM incorporates a specialized implementation of Oracle's JDBC Driver corresponding to the Javasoft JDBC 1.2.2 specification that provides all of the functionality that it provides in a client-side environment but is optimized to run within the database server and provide efficient access to SQL data and PL/SQL subprograms on the local database.
- **Database Stored Procedures:** Traditional database stored procedures, triggers and methods can be implemented using SQLJ in the Oracle8.1 RDBMS. These database stored procedures implemented in Java can be invoked from three types of clients:
  - *Any Java client*: Java clients - fat clients, middle-tier Java clients or browsers - can invoke database stored procedures via JDBC or SQLJ
  - *Any SQL*Net client*: Any other SQL*Net client ie OCI, Pro*, Developer/2000 or any other tool that communicates with the RDBMS across SQL*Net
  - *Any CORBA client*: CORBA clients can also access these stored procedures using the CORBA standard IIOP protocol.

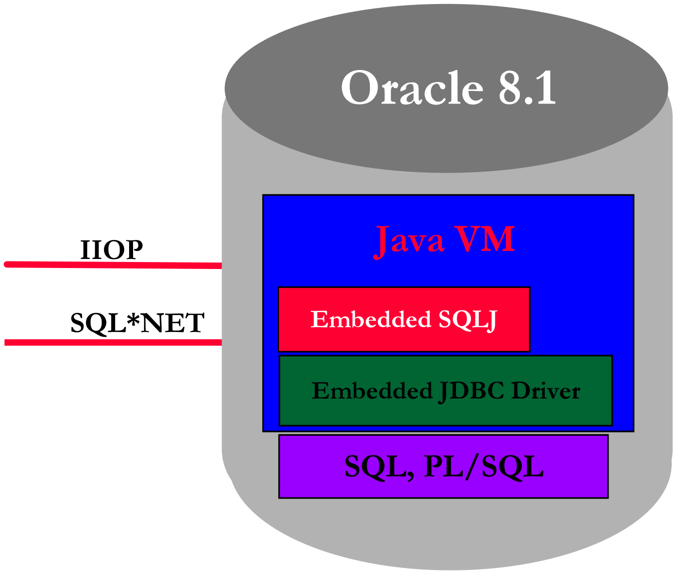

Figure: Using SQLJ for Database Stored Procedures

4. Benefits of Oracle's SQLJ Translator

Having understood some of the technical underpinnings of Oracle's SQLJ Translator and the numerous configurations in which it can be used, it is important to discuss the various benefits that Oracle's SQLJ Translator provides developers building database applications in Java. These benefits can be divided into two different categories:

- Benefits of SQLJ Standard
  - Improved application developer productivity
  - Database independence and multivendor interoperability
  - Vendor-specific customization
- Benefits of Oracle's SQLJ Translator
  - Access to Oracle-specific features
  - Compliance with SQLJ standard
  - Integration with AppBuilder for Java 1.0
  - Integration with many different browsers and Java VMs
  - Flexible deployment configurations - two tier and multi-tier
  - Intranet and Extranet deployment
  - Access to heterogeneous data sources

4.1 Benefits of SQLJ Standard

The SQLJ Partners - Oracle, IBM, Javasoft, Compaq Computer/Tandem, Sybase, Informix, and others - are all cooperatively developing a common standard - the SQLJ Standard - to integrate Java and SQL providing customers a highly productive, open, multivendor solution to build enterprise applications in Java. The SQLJ Standard shared by these vendors provides a number of important benefits.

4.1.1 Improved application developer productivity

SQLJ was developed cooperatively by the major database vendors primarily to provide Java application developers a simple and highly productive way to build database applications.

- **Comprehensive functionality**: SQLJ provides embedded SQL syntax to simplify database access for a variety of different facilities including transaction management, queries, DDL statements to create and drop schema objects, DML statements to manipulate the data, and stored procedure and function calls
- **Compact and high-level interface:** SQLJ provides application programmers with high-level embedded SQL syntax for easy database access. SQLJ programs have more compact code and a higher level programming abstraction compared to JDBC, and thus are less error prone and easier to understand and maintain.
- **Translation-time syntax and semantic checking of SQL statements:** The SQLJ translator performs type-checking and schema-checking to detect syntax and semantic errors in SQL statements at program development-time rather than runtime. In contrast, in a dynamic API like JDBC, the syntax of SQL statements and the types of the arguments are not known until runtime. Programs written in SQLJ are therefore more robust compared to JDBC programs, since they can be statically type-checked independent of the control flow at runtime.
- **Complements JDBC:** While SQLJ provides a highly productive interface for static SQL, users can easily combine calls to JDBC from within a SQLJ program to perform dynamic SQL operation. SQLJ provides a ConnectionContext Java object which can be used to create JDBC Statement objects needed for dynamic SQL operations. SQLJ, therefore, combines the best of

both worlds allowing users to type-check their static SQL without restricting their use of dynamic SQL within the same application.

4.1.2 Database Independence - Multivendor Interoperability

By cooperatively developing the SQLJ standard, Oracle, IBM, Sybase, Compaq, Informix and Javasoft and several other leading industry partners are developing a common, open standard that bridges Java and SQL. There are a number of elements to SQLJ's vendor independence.

- **Common SQLJ Specification:** All the partners share a common SQLJ specification is being submitted to the international ANSI/ISO standards body. It consists of 3 parts:
  - Part 0 **- The SQLJ Language Specification** - provides standard language syntax and semantics for embedding static SQL in Java programs. (`http://www.oracle.com/st/products/jdbc/`)
  - Part 1 - **The Stored Procedure Specification** - provides standards for implementing database stored procedures and functions in Java. This will allow customers who have written stored procedures in Java to easily migrate them between databases.
  - Part 2 - **The Stored Java Class Specification** - addresses standard ways to store Java datatypes and classes as objects in a database.
- **Portability:** SQLJ offers portability at two levels: [i] Java source for stored procedures can be moved from one vendor's platform to another; [ii] Binary portability: Compiled Java classes [Java bytecodes] from translated SQLJ programs can be moved to any compatible SQLJ platform and executed regardless of which platform did the original translation.
- **Vendor-neutral SQLJ syntax:** SQLJ syntax is designed to be database neutral - the SQLJ translator makes minimal assumptions about the SQL dialect.
- **Vendor-neutral runtime environment:** SQLJ's runtime environment consists of a thin layer of pure Java code that in turn communicates with the database server across a call-level API. The runtime environment is therefore vendor neutral in two important ways: [i] Nothing about the SQLJ language is JDBC-specific - it can be implemented with interfaces other than JDBC. Oracle's particular implementation uses JDBC as its runtime environment. [ii] Even a JDBC-specific SQLJ implementation is not restricted to any particular database vendor's JDBC driver. For instance, the Oracle JDBC driver can be used with a JDBC-ODBC bridge to communicate with another vendor's database.
- **Interoperable translator implementations:** Finally, the SQLJ partners are ensuring interoperability between their SQLJ translators by sharing a common SQLJ translator implementation. This will allow SQLJ to be moved seamlessly between compatible implementations from many different vendors and is consistent with Java's philosophy of **write-once run-anywhere**.

4.1.3 Vendor-specific customizations

According to the SQLJ standard, the SQL operations appearing in the original program source are placed into a set of *SQLJ profiles*. This facility provides vendors with the ability to customize a SQLJ application for a vendor's database. By creating and installing a specific customization into a profile, vendors may customize the SQLJ application for their platform. Customizations may used for the following reasons:

- To improve SQL execution performance.
- To provide full database portability.
- To grant access to vendor-specific features not otherwise available to SQLJ programs.

Multiple customizations can be installed into the same SQLJ binary, so that the same binary can be used to execute SQL on databases from vendors, and the execution of that operation will take advantage of the customization available for each vendor.

Oracle's own SQL Translator complies with the SQLJ Standard specification, but provides customizations for both performance and to access Oracle-specific features.

4.2 Benefits of Oracle's SQLJ Translator

In addition to providing the benefits described above that are common to all compliant SQLJ implementations, Oracle's own SQLJ translator and runtime library has a number of other important benefits that are described in the remainder of this section.

4.2.1 Access to Oracle-specific features

While complying with the SQLJ Standard, Oracle's SQLJ Translator has been customized with Oracle-specific extensions designed to simply and easily expose the database server's capabilities to Java application developers. SQLJ developers wishing to use these Oracle-specific features need to install the Oracle SQLChecker classes, the Oracle runtime customizer classes, and an Oracle JDBC driver. Oracle's SQLJ Translator provides support for a number of Oracle-specific features including:

- **Oracle Specific Types:** The following Oracle-specific types are supported in Oracle's SQLJ Translator:
  - ResultSet and Iterator Support.
  - ROWID Support.
  - Extended Output Stream Support.
  - Extended BigDecimal Support.
  - BLOB, CLOB, and BFILE Support.
  - Oracle8.1 Object and REF Support.
- **PL/SQL Stored Procedures:** Oracle's SQLJ Translator provides access to PL/SQL Stored Procedures and anonymous blocks.
- **Comprehensive Globalization/Multibyte Character Support:** Oracle's SQLJ Translator provides excellent support for national language character sets. Oracle's SQLJ Translator fully supports Java's Unicode escape sequences. Additionally, it allows the use of ISO8859_1 (Latin-1) characters directly in SQLJ programs. It also allows the use of Unicode characters inside of any SQL code fragment.
- **Java Stored Procedures:** With the integration of a Java Virtual Machine with Oracle8.1 and the embedding of a SQLJ translator within the VM, SQLJ can also be used to write traditional database stored procedures, functions and triggers in Java.

4.2.2 Standards Compliance

Oracle's SQLJ Translator, runtime library and generated SQLJ programs comply with a number of different Java and SQL standards.

- **JDK Versions:** Oracle's SQLJ Translator is compliant with JDK 1.1.1 - both the SQLJ translator itself and the generated SQLJ programs require Java 1.1.1 or later.
- **JDBC Drivers:** Oracle's SQLJ Translator can be used with any industry standard JDBC driver; the generated SQLJ programs does not restrict its use with Oracle's own JDBC driver. However, Oracle suggests the use of its own JDBC drivers particularly to access Oracle database-specific functionality.

- **SQLJ Standard Version:** When made available in production, Oracle's SQLJ Translator will be compliant with the SQLJ 1.0 specification. The current web release is compliant with an earlier version of the SQLJ specification.
- **RDBMS Versions:** SQLJ programs generated by Oracle's SQLJ Translator can be used with both the Oracle7 and Oracle8 JDBC drivers. The appropriate JDBC Driver needs to be chosen.

4.2.3 Integration with AppBuilder for Java

Oracle's AppBuilder for Java 1.0 release provides a number of features integrating SQLJ programs with the AppBuilder for Java development environment. Specifically, the AppBuilder for Java development environment provides a very rich environment for SQLJ developers to develop and deploy applications. It includes three specific capabilities:

- **Syntax-directed editing:** By providing a syntax-directed editor for SQLJ source programs, AppBuilder for Java 1.0 provides capabilities such as proper indentation as users type in SQLJ source into the tools source editor, and the ability to chroma-code SQLJ constructs.
- **Project Creation:** AppBuilder for Java 1.0 supports the inclusion of SQLJ files in projects and knows how to build SQLJ files. Further, the tool integrates the SQLJ translator allowing application developers to transparently invoke it to process SQLJ and Java files that have changed [make dependencies] - this eliminates the need for separate translation and compilation steps to a user. Finally, the tool allows developers to set and change the translate time options for SQLJ within the tool.
- **Debugging:** SQLJ also provides a set of runtime interfaces which is integrated with AppBuilder for Java customers to debug SQLJ programs at the source level within the tool. The debugging capability will be further enhanced with future releases of SQLJ and AppBuilder for Java.

4.2.4 Integration with Browsers and Java VMs

Oracle's SQLJ translator will be certified in a number of different configurations with most industry standard browsers and Java VMs.

- **Browsers**: Oracle's SQLJ runtime library used in combination with the Thin JDBC driver will be certified with the most popular browsers on the market today - Netscape Navigator 3.0 and 4.0 and Internet Explorer 4.0.
- **Java Virtual Machines:** Further, the SQLJ translator and runtime library will be certified with a number of different Java VMs including JDK, Oracle's AppBuilder for Java development tool, and others. For the complete list of supported configurations on different operating systsms and hardware platforms, users are encouraged to read platform specific SQLJ documentation or contact Oracle's Worldwide Support organization.

4.2.5 Flexible Deployment Configurations

SQLJ applications can be deployed in a variety of different configurations as described earlier in this paper. Specifically, they can be used for three kinds of applications:

- **Fat Clients:** SQLJ Applications can be deployed in traditional two-tier client-server configurations together with Oracle's JDBC/OCI Driver.
- **Thin Clients:** SQLJ Applications can be deployed in two-tier thin client-database server configuration together with Oracle's Thin JDBC Driver.
- **Middle Tier Web Servers or Application Servers:** Database-centric Java applications that execute in any middle-tier Java web server or application server can be written using SQLJ. Specifically, SQLJ applications targeting the Oracle database server can be executed on the Oracle Application Server's Java VM.

- **Database Stored Programs:** With Oracle8.1, SQLJ programs can execute on the database server's Java VM which provides a very high performance and scaleable platform to execute SQLJ code. Traditional database stored procedures and functions, triggers, or object-relational methods can then be implemented using SQLJ.

4.2.6 Intranet and Extranet Deployment

SQLJ applications can work across firewalls. Since the runtime environment for SQLJ applications is essentially a thin layer of pure Java code together with the JDBC driver selected by the user, SQLJ applications work with all the firewalls with which the specific JDBC driver chosen works. Oracle's own JDBC drivers can work in both an Intranet and in an Extranet setting. In an Extranet deployment, the drivers can be used with most industry leading firewalls which have been SQL*Net certified. Today, the following firewall vendors have certified their Firewalls with SQL*Net:

- **Stateful Inspection Firewalls**: Firewalls from Checkpoint, SunSoft, and CISCO Systems
- **Proxy-based Firewalls**: Firewalls from Milkyway Networks, Trusted Information Systems, Raptor, Secure Computing Corporation, and Global Internet are proxy-based and are all SQL*Net certified.

4.2.7 Access to Heterogeneous Data Sources

SQLJ applications can also be used to access a wide variety of heterogeneous data sources. By using Oracle's Transparent Gateway products, SQLJ can provide high performance transparent connectivity to over 25 different enterprise and legacy databases from Java applications or applets as shown in the figure below.

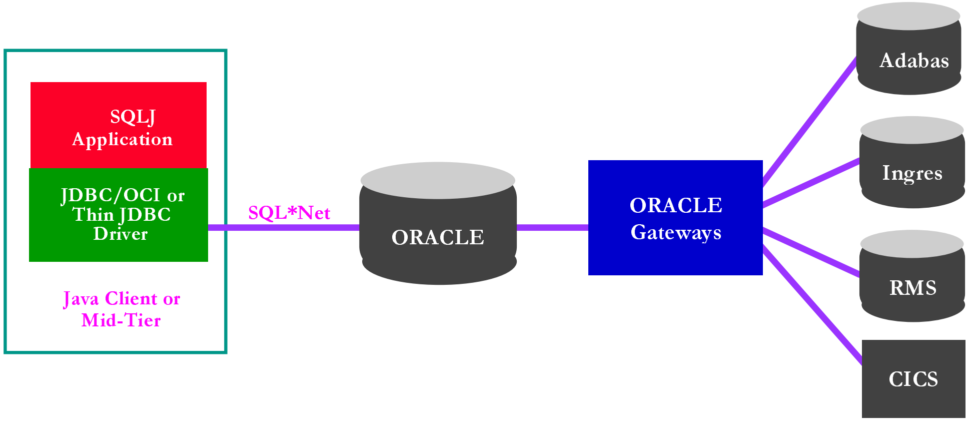

Figure: Using JDBC to access Heterogeneous Data Sources

4.2.8 Performance

Recognizing that Java application performance is one of the critical gating factors that can drive the widespread adoption of Java, Oracle has designed its SQLJ runtime environment with performance in mind. There are three important ways in which the SQLJ has been optimized to provide excellent performance:

- **Direct Database Access**: The Oracle-specific SQLJ profile has a number of important Oracle - specific customizations that are installed into the binary SQLJ application for better performance. For instance, Oracle is implementing advanced SQL precompilation techniques to improve SQL execution performance. SQLJ performance can therefore be competitive with the best JDBC drivers in the marketplace.

- **API Extensions**: Oracle provides a number of extensions to its SQLJ Translator to provide performance improvements. They include:
  - **Bulk operations**: Oracle's SQLJ implementation allow a user to set a number of rows to prefetch into the client during queries, thereby reducing round trips to the server. This dramatically improves the throughput of relational data.
  - **Execution Batching**: Oracle's SQLJ Translator allows a user to batch INSERTS and UPDATES to the server, further reducing round trips to the server.
  - **Define Query Columns**: It allow a user to inform the drive of the types of columns in an upcoming query, saving a separate define round trip to the database.
- **Optimized SQLJ Runtime**: Finally, Oracle's own SQLJ implementation is a very thin layer of Java code thereby reducing to the minimum overhead introduced by the SQLJ runtime itself.

5. Conclusions and Future Directions

The paper has provided an overview of SQLJ, a highly productive, open standard for Embedded SQL in Java supported by all the leading database vendors including Oracle, IBM, Sybase, Informix, Javasoft and Compaq/Tandem. SQLJ program can be deployed in a number of different configurations including two-tier client server applications, three-tier Intranet and Extranet applications, and to write database stored procedures, triggers and methods with Oracle8.1.

Oracle's SQLJ Translator conforms to the SQLJ standard but provides support for a number of database features. Oracle will aggressively evolve its SQLJ implementation as a critical aspect of its comprehensive server-centric Java strategy. Oracle will be making its SQLJ translator commercially available with Oracle8.1. Some of the additional features that Oracle will include in its Oracle8.1 SQLJ product are:

- **Oracle8 Datatypes**: Support for Oracle8 specific features such as Large Objects [LOBs] and its object-relational extensions such as object types and collections [varrays and Nested Tables]. Oracle7 types such as REFCURSORs and PL/SQL Tables will also be supported.
- **Improved globalization support**: Additional services in SQLJ, such as complete NLS support for its message files.
- **Performance and scalability improvements**: Further improvements in performance and scalability by further integration with features of Oracle's JDBC driver. SQLJ performance will reap the benefits of improvements in the JDBC drivers themselves, leveraging performance and scalability optimizations in the database server such as connection pooling, elimination of bind round trips, DML RETURNING syntax and optimizations in SQL*Net.

SQLJ: Embedded SQL for Java

May 15, 1998

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
415.506.7000
Fax 415.506.7200