Unleash the Power of Java Stored Procedures

An Oracle White Paper
June 2002

ORACLE

Unleash the Power of Java Stored Procedures

EXECUTIVE OVERVIEW

Java stored procedures use procedural logic encapsulating multiple SQL statements that run within the database and are invoked in one call, thereby avoiding multiple network round trips. Java support in the database offers an open and portable alternative to proprietary procedural SQL extensions such as Oracle PL/SQL, for implementing stored procedures. Java stored procedures extend database functionality and programmability by bridging J2EE, XML, and non-J2EE worlds, using the database as an active Web client, EJB client, and Web services client. The goal of this paper is to present and explain those possibilities.

BACKGROUND

Motivations for Stored Procedures

Stored procedures allow the processing of a set of database operations in one call. By processing data locally within the database and returning just the results, stored procedures enhance the performance of data-intensive operations. You invoke stored procedures through a SQL interface, which hides their implementation from the requester. Their implementation and deployment can be delegated to database developers or administrators who are knowledgeable in writing efficient queries and SQL statements, thereby making efficient use of their skills. This application modularization makes for easier debugging, tuning, upgrading, and migration.

PL/SQL or Java-Based Stored Procedures?

Whether to use PL/SQL or Java stored procedures is the legitimate $10,000.00 question people ask each time I talk about Java stored procedures—so, without any delay, let's cut to the chase.

PL/SQL Stored Procedures

PL/SQL is the Oracle procedural extension to SQL. From its inception, PL/SQL has been designed and optimized for stored procedures and functions. It is well suited for encapsulating SQL operations with procedural logic and for manipulating all database object types. PL/SQL procedures and packages have been and will continue to be extensively used by a large community of Oracle developers and customers for implementing database operations and packages. Furthermore,

starting with Oracle9*i* Database, Release 1, PL/SQL runs natively compiled, and faster.

Cross-Platform portability: Compiled PL/SQL packages can be moved to different platforms—for example, from Solaris to Linux or Windows 2000—without re-compilation.

Java Stored Procedures

Java stored procedures represent an open, database-independent alternative to PL/SQL. Furthermore, Java stored procedures bring the power, richness, and object-orientation of the Java language.

Robustness

As opposed to PL/SQL, Java requires declaring exceptions that can be thrown by methods in any class, thereby making Java stored procedures more robust.

Reuse of Java skills: because Java is one of the dominant programming languages, it is likely that Java programmers already exist within your company; furthermore, new hires graduating from colleges have both Java and database skills.

Cross-Vendor and Cross-Platform Portability: all major databases vendors provide Java support in their database, either through a tight integration with the database runtime such as Oracle (see the OracleJVM section), or through a loosely coupled integration. Java stored procedures not only inherit cross-platform portability from the database but, moreover, cross-vendor portability—in other words, database-independence.

Application Partitioning: Java support in the database allows, whenever appropriate, well-scoped Java code to be moved from client-side or middle-tier into the database, or the opposite.

Complex database logic and powerful usage patterns: as this paper will explain, Java programming in the database allows implementing more complex database logic than PL/SQL and extends database functionality.

PL/SQL or Java: It Depends!

So the answer to the $10,000.00 question is, it depends! It depends on skills and application requirements. Use PL/SQL for SQL operations and access to all database types and objects; use Java, when you already have the skills, for SQL operations and for leveraging and interacting with the Java, J2EE, XML and Web services worlds. Oracle strategy is to enhance both languages and make them inter-operate; as an example, as of Oracle9*i* Database, Release 2, OracleJVM supports standard Java Debugging Wire Protocol, which allows debugging both PL/SQL and Java within the same JDeveloper or third-party IDE debugging session.

ENABLING TECHNOLOGIES

Programming the database using Java requires a Java Virtual Machine (JVM) as well as mechanisms for accessing SQL data from Java.

OracleJVM: Enabling Java Programming Within the Database

Starting with Oracle8*i*, Release 1 (Oracle 8.1.5), Oracle has offered a tightly integrated JVM within the database. Each database session may activate a virtually dedicated JVM during the first Java code invocation; subsequent users benefit from this already Java-enabled session. In reality, all sessions share the same JVM code and statics—only private states are kept and garbage collected in individual session space, providing for Java the same session isolation and data integrity as SQL operations: there is no need for a separate process for running Java. This session-based architecture provides a small memory footprint, and gives OracleJVM the same linear SMP scalability as the Oracle database. OracleJVM works similarly and consistently with every platform and OS on which the Oracle database runs. Java classes (and, optionally, the corresponding sources) are loaded in a database schema—through JDeveloper, third-party IDE, SQL*Plus, and *loadjava* utility—before they can be invoked. Java class invocation is secured and controlled through database authentication and authorization, Java 2 security (exposed to SQL through *Javasyspriv and Javauserpriv* roles), and *invoker's or definer's rights*[1]. OracleJVM provides a deployment time native compiler, which allows Java code (and, therefore, stored procedures) to be compiled once, stored in executable form, shared among users, and invoked more quickly and efficiently.

Oracle is continuously upgrading and improving OracleJVM performance and usability. Oracle9*i* Database, Release 2, complies with J2SE 1.3.x; any J2SE 1.3 code can run in the database and enable sophisticated data-driven computing, as discussed in the following sections.

For more details, code samples, and a discussion forum, go to the following Web site:

http://otn.oracle.com/tech/java/jsp/content.html

[1] For a full discussion of Oracle JVM security, see http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/java.920/a96656/security.htm

Loadjava Now Generates Call Spec

loadjava is a utility for loading Java classes and JAR files into the database. It is invoked either from the command line or through the `loadjava` method contained within the DBMS_JAVA class. In Oracle9*i* Database, Release 2, it provides new options for automatically publishing Java classes as stored procedures. *loadjava* creates PL/SQL wrappers (*Call Specs*) for methods contained in the processed classes, with the following requirements:

- The method must be a member of a public class.
- The method must itself be declared public and static.
- The method's signature must be "mappable, " according to mapping rules[2].
- If the return type is void, then a procedure is created.
- If the return type is arithmetic, `char` or `java.lang.String`, then a function is created and its return type is determined according to mapping rules.

Although Call Spec offers great benefits (see *"Sharing Business Logic Between Legacy and J2EE Worlds"*), in some situations you might not want to go through the PL/SQL wrapper. Oracle plans to provide a mechanism for directly invoking Java methods in the database, from external Java code, bypassing the Call Spec.

JDBC: One of the Most Essential Java APIs

JDBC is the most standard and common mechanism for accessing SQL data from Java[3]. Oracle furnishes:

- a type-4, pure Java, client JDBC driver, also referred to as a "thin" driver, for middle tier servers, client applications, and applets—also useable from within the database for accessing remote Oracle databases from Java stored procedures (similarly, a third-party pure JDBC driver can be loaded into OracleJVM for accessing third-party databases from within Java stored procedures)
- a type-2, client JDBC driver, based on OCI libraries for middle tier servers and client applications
- a type-2, server JDBC driver, sometimes referred to as a "kprb" driver, based on internal database libraries for faster, in-session SQL data access from Java stored procedures (and any Java code running within the OracleJVM)

Oracle furnishes complete support for JDBC 2.0 as well as significant support for JDBC 3.0 features and specific extensions.

---

[2]http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/java.920/a96656/newtools.htm#1010615

[3] "*JDBC support is one of the most essential and fundamental API's in the world of Java*" Rick Ross - in JavaLobby newsletter, April 2002

For more details, code samples, and a discussion forum, go to the following Web site: http://otn.oracle.com/tech/java/sqlj_jdbc/content.html

SQLJ

SQLJ is an ANSI SQL-1999 (SQL3 or SQL-99) multi-parts specification:

- Part 0 - *"Embedded SQL in Java":* specifies the ability to embed SQL statements in Java, similar to Pro*C. The Oracle implementation of SQL3 Part 0 is called "*SQLJ*" and supports dynamic SQL, beyond the standard, which requires embedding only static SQL. The Oracle strategy is to put SQLJ, functionally, on a par with JDBC, as a more productive and easier alternative to JDBC for both client-side and server-side Java. For more details, code samples, and a discussion forum, go to:http://www.oracle.com/forums/forum.jsp?id=424322
- Part 1 - *"SQL routines using Java":* specifies the ability to invoke static Java methods from SQL as procedures and functions. The Oracle implementation is referred to as Java Stored procedures or functions, and is the topic of this paper. For more details, code samples, and a discussion forum, go to: http://otn.oracle.com/tech/java/jsp/content.html
- Part 2 - *"SQL Types using Java*": specifies using Java classes as templates for user-defined data types in SQL. The Oracle implementation of SQLJ-Part-2 is known as *"SQLJ Object types"*. For more details, go to: http://technet.oracle.com/products/oracle9i/daily/jan17.html

COMMON USAGE PATTERNS OF JAVA STORED PROCEDURES

Java stored procedures are commonly used to implement the same database operations as PL/SQL, such as stored procedures, stored functions, and database triggers.

Calling Java Stored Procedures from SQL[4]

Java stored procedures and functions are callable, through their *Call Spec*, from:

- SQL DML statements (`INSERT`, `UPDATE`, `DELETE`, `SELECT`, `CALL` , `EXPLAIN PLAN`, `LOCK TABLE`, and `MERGE`)
- PL/SQL blocks, subprograms, and packages
- Database Triggers: the Java stored procedure is invoked through its *Call Spec* in the trigger body

---

4 http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/java.920/a96656/invokeap.htm

Calling SQL and Accessing Database Objects from Java

JDBC and SQLJ allow invoking SQL statements accessing database objects such as Collection types, PL/SQL Index by tables (OCI only), SQLJ Object types, NCHAR/Unicode data, BLOBs, CLOBs, Object Type Inheritance, and so on.

JPublisher

JPublisher is a powerful Java utility that simplifies transferring database object type instances—such as SQL object types, VARRAY types, nested table types, and object reference types—between the database and a Java program. For the Java developer, it eliminates the difficulty of manual mapping and coding. JPublisher translates database objects as well as PL/SQL package types into Java classes by:

- inspecting the provided database object type and generating a Java wrapper class that maps to the corresponding fields and methods
- encapsulating a PL/SQL package by generating the equivalent Java wrapper that contains calls to the PL/SQL packages

The generated Java wrapper can be called directly in your Java modules, as you would for any other Java methods.

Calling Java Stored Procedures from JDBC

Java stored procedures and functions are invoked from Java, either through *PreparedStatement* or *CallableStatement* (OUT and INOUT parameters).

In the following examples, we have omitted IN, OUT, and INOUT parameters, declarations, and registrations.

```
// SQL92 escape syntax: parameterless stored procedure
CallableStatement cstmt = conn.prepareCall("{CALL
proc}");
// SQL92 escape syntax: stored procedure
CallableStatement cstmt = conn.prepareCall("{CALL
proc(?,?)}");
// SQL92 escape syntax: stored function
CallableStatement cstmt = conn.prepareCall("{? = CALL
func(?,?)}");
```

Security—Data Access Control

One common use of Java stored procedures is to control and restrict access to Oracle data by allowing users to manipulate the data only through stored procedures that execute under their invoker's[5] privileges while denying access to the table itself. For example, you can disable updates during certain hours or give managers the ability to query salary data but not update it.

[5] *Invoker's and definer's rights* are SQL concepts that are used to dynamically authorize users when executing SQL, PL/SQL, or JDBC.

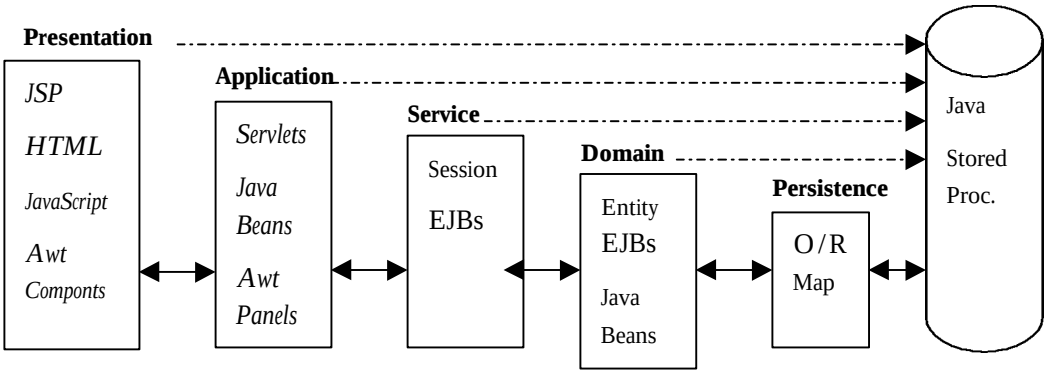
Implementation Transparency: 170 Systems, Inc."

Stored procedure implementations can change over time from PL/SQL to Java or the opposite, transparently to the requesters, as illustrated by the following quote from one of our major partners, 170 Systems, Inc.

"When our customers implement the 170 MarkView Document Management and Imaging System, the solution often incorporates sophisticated business logic. To date, this logic has been created using 170 MarkView configuration products in the form of PL/SQL code. We are now moving to Java stored procedures as an additional method for capturing this logic. The advantage this brings to the 170 MarkView product line is that this enables a new set of users - those who have Java rather than PL/SQL skills - to create and maintain business logic in the 170 MarkView solution."[6]

USING JAVA STORED PROCEDURES IN THE J2EE ENVIRONMENT

Typical J2EE Application Architecture



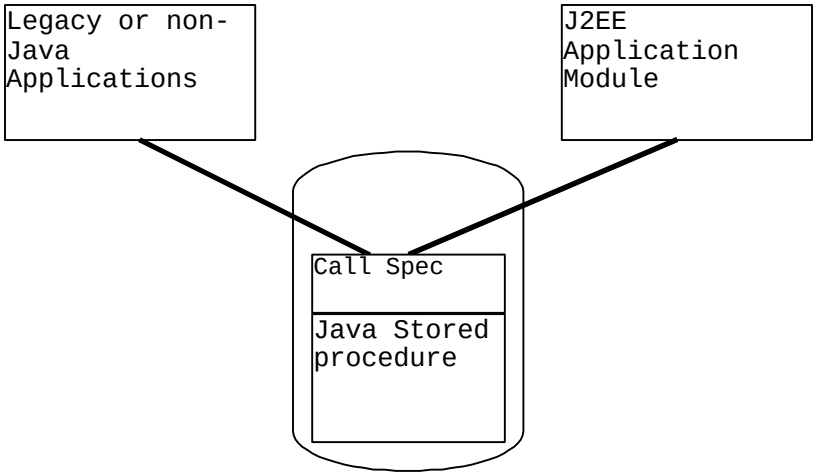
JSP, Servlets, and Session Beans Directly Calling Java Stored Procedures

Query-based JSPs and Servlets can directly invoke Java stored procedures, through JDBC/SQLJ, bypassing Session and Entity beans. For non-complex domain objects, Java stored procedures are often an easier and faster alternative to Entity Beans. In addition, as described in the following sections, BMP and CMP beans can also leverage Java stored procedures for auto-generating primary keys or implementing finders methods.

[6] Simon Poulding - 170 Systems, Inc.

Sharing Business Logic Between Legacy and J2EE Worlds

As the book *Mastering Enterprise JavaBeans*[7] explains, stored procedures allow sharing business logic between J2EE and non-J2EE worlds. Independently from their implementation, stored procedures are declared and known to the database catalog, through their SQL signature (Call Spec)—a PL/SQL wrapper in the case of Oracle. J2EE components, as well as legacy applications modules, can invoke the same stored procedure through the same Call Spec wrapper.



Autogenerating Primary Keys for BMP Entity Beans[8]

A BMP entity bean instance can be uniquely identified by the auto-generated primary key associated with the newly inserted data as return value for *ejbCreate()*. Retrieving this value within *ejbCreate()*, can be performed in one database operation by using a stored procedure that will insert the corresponding data, then retrieve or compute the primary key. Alternatively, you can insert the data using a simple INSERT statement, then retrieve the value of the pseudo-column (*ROWID*) of the newly inserted row, using JDBC 3.0 *RETURN_GENERATED_KEYS*; but the stored procedure approach is more portable across JDBC drivers and databases.

The following code snippet is excerpted from *EJB Design Pattern;* (OUT parameters registration have been omitted).

---

[7] *Mastering Enterprise JavaBeans, 2nd Edition* - Ed Roman, Scott W. Ambler, Tyler Jewell - Wiley

[8] This pattern is described in *EJB Design Patterns* - Floyd Marinescu - Wiley
(the original example uses a PL/SQL stored procedure for retrieving ROWID).

```
Public AccountPK ejbCreate(String ownerName, int
balance) throws CreateException
{
   try {
     CallableStatement call = conn.prepareCall{
            "{call insertAccount(?, ?, ?)}"};
           return new AccountPK(accountID);
    }
--
CREATE OR REPLACE PROCEDURE insertAccount{owner IN
varchar, bal IN number, newid OUT number)
AS LANGUAGE JAVA NAME
'GenPK.insertAccount(java.lang.String [])';
/
--
define insertAccount()as a public static Java method, in
public GenPK class:
```

- insert data
- compute unique key (passing out a sequence number out of a pre-allocated chunk or other mechanism)
- return primary key

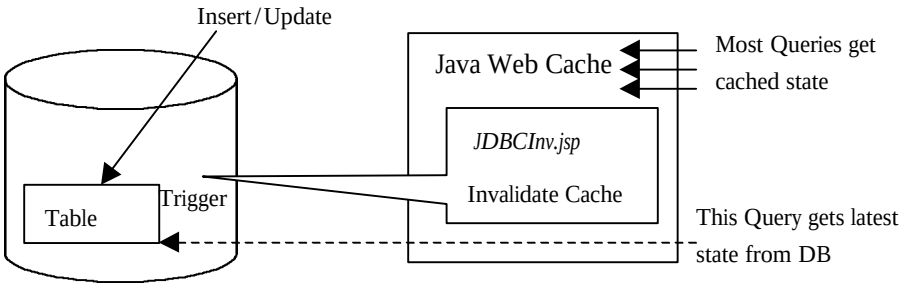
Mapping CMP Beans Finders Using Java Stored Procedures

You can use Java stored procedures in conjunction with OR mapping frameworks such as TOPlink, and CocoBase, to implement custom finder methods such as `findByStoredProcPK`—in contrast to `findByPrimaryKey`, which is generated by the EJB container. As an example: TOPLink allows you to define the EJB finders as *REDIRECT or NAMED finders*, then implement the query for the finder as described in the TopLink for Java documentation.

Cache Invalidation—Calling-Out to Web Components

Web Object Cache tags provide increased performance by letting you cache Java objects within an HTTP session. In order to refresh cached read-mostly data, such as product catalogs and price-lists, a Java stored procedure attached as a trigger to the database table will automatically call-out a JSP for invalidating a Web Object Cache object that maps its state to the database table. Upon firing the trigger, the very next query will force the state to be refreshed from the database. This pattern

is similar to “Publish event-driven Web content with JSP custom tags”[9]; using the database as sole source of truth, ensures capturing every data modification, regardless of its origin (other non-Java applications, legacy applications).



HttpCallout.java (excerpt)

```
public class HttpCallout {

   public static void getURL(String hostname, String portnum,
String url) throws InterruptedException {

     try {

       // process arguments

       String protocol = "http";

       String host = hostname;

       int port = Integer.parseInt(portnum);

       String page = url;

       // Grab HTTPConnection

 HTTPConnection con = new HTTPConnection(protocol, host, port);

       con.setTimeout(20000);

       con.setAllowUserInteraction(false);

       // Issue Get call

       HTTPResponse rsp = con.Get(page);


       // Grab Response

       byte[] data = rsp.getData();

       if ( data == null ) {

         System.out.println("no data");

       } else {
```

[9] http://www.javaworld.com/javaworld/jw-04-2002/jw-0419-event.html

```
          System.out.println("data length " + data.length);
          System.out.println(new String(data));
        }
      }
      catch ( Throwable ex ) {
        ex.printStackTrace();
      }
    }
}
```

Callout.sql

```
spool trigger.log
set echo on
connect scott/tiger
CREATE OR REPLACE PROCEDURE scott.callout_proc (host VARCHAR2,
port VARCHAR2,url VARCHAR2) AS LANGUAGE JAVA
NAME 'HttpCallout.getURL(java.lang.String, java.lang.String,
java.lang.String)'
;
/
show errors
CREATE OR REPLACE TRIGGER scott.callout_trig AFTER DELETE OR
INSERT OR UPDATE ON scott.callout
begin
    scott.callout_proc('dlsun693.us.oracle.com', '8888',
'/ojspdemos/cache/fseg/JDBCInv.jsp');
  end;
/
show errors
spool off
exit
```

Calling-Out to EJB Components in J2EE Servers

You can connect to EJB from Java Stored procedures, using either standard RMI over IIOP (against a J2EE 1.3 compatible server) or RMI over vendor specific transport protocol, such as ORMI (iAS/OC4J) or T3 (BEA). While providing RMI over IIOP for interoperability, each vendor has its own optimized protocol for RMI calls to their EJB container. Oracle9*i*AS, supports both RMI calls over IIOP and ORMI protocols. The following steps allow calling-out EJB in OC4J (this feature is a technology preview in Oracle9*i* Database, Release 2, and not yet supported for production). The complete code will be posted on the Oracle OTN Web site.

1. `loadjava -v -u sys/<passwd> -noverify -r -s -g public -genmissingjar gen.jar oc4jclient.jar jaas.jar ejb.jar`

   `(those jars are located under $j2ee_home/)`

2. `grant ejbclient to <ejb_user_schema> (as sysdba)`

3. `loadjava -v -r -u <ejb-user>/<passwd>`

   `cmpapp-client.jar and cmpapp-ejb.jar (ejb stubs)`

   `(located under` $j2ee_home/demo/cmp/client)

4. ```
   create or replace procedure myejb(args varchar2) as
   language java name
   'cmpapp.EmployeeClientTest(java.lang.String)';
   /
   ```

5. ```
   Call with
    set severoutput on
    call dbms_java.set_ouput(2000);
    call myejb('test');
   ```

see the results in your sqlplus environment.

USING JAVA STORED PROCEDURES IN XML AND WEB SERVICES ENVIRONMENTS

Oracle9*i* XML Database

Oracle9*i* Database release 2 provides *XMLType* for native XML storage and retrieval, as well as *XML Repository* for XML resources foldering, access control, versioning, and so on. Those capabilities are known under the collective name of Oracle9*i* XML Database (XDB). Java developers can write XML DB applications either through JDBC or directly within the database through Java stored procedures. These Java stored procedures are accessible from SQL and PL/SQL, through their *Call Spec*; another illustration of how Java stored procedures can

bridge SQL, XML and Java worlds. For more details about XDB, please visit the Oracle OTN Web site, @ http://otn.oracle.com/tech/xml/xmldb/content.html.

Extend Database Programmability with XML Develop Kit (XDK)

Because OracleJVM is J2SE 1.3 compatible, Java classes and JAR files can be loaded into the database. The XML Developer Kit is written in Java, and exposes its public methods, as Java stored procedures, extending the database's XML programmability. PL/SQL and Java code running within the database have access to the XML parser, the XSLT processor, XPath engine, and XML SQL Utility (XSU).

XML Parser for Java

The XML parser supports W3C XML1.0 recommendations, DOM 2.0 CORE, DOM 2.0 Traversal including Treewalker, Node Iterator and Node Filter, SAX20, SAX2-ext, and JAXP 1.1. PL/SQL and Java code running in the database access the XML parser through *xmlparser* and *xmldom* packages.

XML SQL Utility

XML SQL Utility (XSU) is a Java utility that generates an XML Document from SQL queries or a JDBC ResultSet, and writes data from an XML document into a database table or view (XML output can be produced as Text, DOM trees, or DTDs). This utility is exposed to the PL/SQL world through *dbms_xmlquery* and *dbms_xmlsave* packages.

DBMS_XMLQuery

This package is based on the XML SQL utility. To use it, create the following SQL script, *xmlq_test.*`sql`:

```
set echo on

set long 32000

var cb clob

declare

c dbms_xmlquery.ctxtype;

begin

c := dbms_xmlquery.newContext('select dname,
CURSOR(select ename,sal from p  where emp.deptno =
dept.deptno and emp.hiredate=:hiredate) as employees

from dept where deptno = :deptno');

  dbms_xmlquery.setTagCase(c,1);

  dbms_xmlquery.setBindValue(c,'deptno',10);

  dbms_xmlquery.setBindValue(c,'hiredate','09-JUN-1981');

  :cb := dbms_xmlquery.getXML(c);

  dbms_xmlquery.closeContext(c);

 end;
/

then use SQLPLUS to connect and run it

SQL> @xmlq_test

SQL> print cb
```

This will result in:

```
CB

------------------------------------------------------

<?xml version = '1.0'?>

<rowset>

   <row num="1">

      <dname>ACCOUNTING</dname>

      <employees>

         <employees_row num="1">

            <ename>CLARK</ename>

            <sal>2450</sal>

         </employees_row>

      </employees>

  </row>

</rowset>

SQL>
```

Database as Active Web Service Client

A Java stored procedure can be a Web service requester. Attaching such a stored procedure as a trigger allows the database to automatically invoke external Web services upon data-driven events. Non-Java modules such as PL/SQL procedures and DBMS packages can be encapsulated with a Java wrapper that will invoke external Web services, on their behalf. Here are the steps for a proof of concept:

- Load XERCES, SOAP, and JAXP jar files.
- Grant connect and resolve to the SOAP server/port.
- Load the SOAP client Java class.
- Create a PL/SQL package to wrap the SOAP client as trigger.

The complete code sample and instructions will be posted on the Oracle OTN Web site.

Calling-out an XML-RCP Server[10]

Here is a simple way of calling-out a remote XML-RPC server through SQL:

- Load RPC-Clients JAR files from http://xml.apache.org/xmlrpc/client.html, into OracleJVM.
- Load the following client code, *DbClass.java*, into the database.
- Invoke the remote XMl-RPC server through the *select DbClass.callOut('some value') from dual* statement.

```
public class DbClass
{
    public static String callOut(String valueIn) {
        String failReason = "no reason";
        try {
            // Use the Apache Xerces SAX Driver
XmlRpc.setDriver("org.apache.xerces.parsers.SAXParser");
            // Specify the server
            XmlRpcClient client =
             New XmlRpcClient("http://localhost:8585/");
            // Create request
            Vector params = new Vector();
            params.addElement(valueIn);
           // Make a request and print the result
                  String result =
      (String)client.execute("hello.sayHello", params);
//System.out.println("Response from server:  " +result);
            return "the answer " +  result;
        } catch (ClassNotFoundException e) { …
        } catch (MalformedURLException e) { …
        } catch (XmlRpcException e) { …
        } catch (IOException e) { …}
        return "failed:" + failReason;
    }
```

[10] Courtesy of Chris Field

```
}
```

This opens the door to complex queries, including federating data from Web services.

Database as Web Service Provider

Java as well as PL/SQL stored procedures that implement some business services can be exposed as Web service. Oracle9iAS release 9.0.2 provides J2EE-based Web services; furthermore, it allows encapsulating stateless database stored procedures with J2EE artifacts and exposed as regular J2EE Web Services. The stored procedure, which implements the service, is invoked via JDBC and runs in the database. Oracle plan to expose more database operations through Web services mechanisms.

For more details about exposing stored procedures as Web services, visit the following link at the OTN web site:
http://otn.oracle.com/tech/java/oc4j/pdf/Oracle9iAS-R2-J2EE.pdf

CONCLUSIONS

The integration of the database with a JVM that complies with J2SE, OracleJVM, allows extending database features and programmability through Java stored procedures. This paper highlights the simplicity and the power of Java stored procedures through new, emerging usage patterns, such as bridging and cooperating with J2EE components, XML applications, and Web services. As illustrated by the examples and code snippets[11], the ability to run Java code in the database opens the door for unlimited usage patterns. Some of those, such as EJBs Active Components for J2EE (AC4J)[12], which allow asynchronous, transactional then recoverable invocation of regular Enterprise JavaBeans in Oracle9*i*AS/OC4J through Java stored procedures, calling-out to JMS or in-session JMS/AQ invocation, and so on, are under work and not exposed in this paper.

---

[11] All code samples will be bundled and posted on the OTN website, @ http://otn.oracle.com/tech/java/jsp/content.html
[12] See Active Components for J2EE (AC4J) @ http://otn.oracle.com/tech/java/oc4j/pdf/oc4j_ent_jb_devguide_r2.pdf

Unleash the Power of Java Stored procedures
June 2002
Author: Kuassi Mensah
Contributing Authors: Cheuk Chau,
Ekkehard Rohwedder, Chris Field
Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the internet.