DATABASE TRENDS AND APPLICATIONS

Solutions for the Information Project Team • www.dbta.com Volume 15, Number 11 • November 2002

Java Objects in the Database

By Lee Fesperman

Java not only provides a viable alternative to the proprietary stored procedure languages offered with most relational databases, it provides a object-oriented language allowing new levels of capability.

Java is more than just a universal stored procedure language. Objects defined in Java can be cataloged and stored in the database and used as values in database columns. When cataloged in the database Java Classes can be used as functions in SQL and Stored Procedures. When used in columns their type is a Java class that is mapped to the database type internally. Java column values are active objects whose methods can be accessed in SQL commands or when retrieved locally. Java Objects defined in the database eliminate the need for O/R mapping tools and provide powerful O-O database capability such as inheritance, encapsulation, and polymorphism.

Java Data Objects (JDO) provides one approach to data access by supporting the construct of a business object persistence manager that is outside of the database. JDO specifies mapping between persistent data objects and information in external data stores but does not define how the mapping should be performed. In addition to adding another layer to your database applications it requires a new query and manipulation paradigm - JDOQL, for accessing the data by Java programmers.

Java Classes in the FirstSQL/J Database

The flexibility of cataloging Java Classes in the database provides many advantages:

- Java classes are made persistent without special treatment and O/R Mapping tools
- Supports simplified SQL access to persistent Java objects and methods
- Supports complex object relationships
- Provides an object-oriented approach to database development

Cataloging Classes in the Database

In order to create a class in the database it must be cataloged along with its methods and what classes it inherits. Creating a database class can use the following basic form:

CREATE CLASS class-name FROM 'java-name' [INHERITS class-list]

java-name is a string containing the fully qualified name of the Java class being cataloged. *java-name* may specify a *Java Class* or a Java Interface. In the optional INHERITS clause, class-list is a comma-separated list containing the user-defined base class (if any) and any user-defined interfaces implemented. The base class or *primary* interface is listed first. Any class or interface listed must be cataloged in the database with the database name used in the database class.

Dropping a database class uses the basic form:

DROP CLASS class-name [RESTRICT | CASCADE]

The ability to easily define the type of a database column as a Java class is an important capability. Internal conversion of Java types to database types means columns in database tables can then use the Java class for their type definition. Once a database class is cataloged it may be used in defining object columns.

For examples, we will use a Money class, supporting a variety of currencies:

Money Class
Constructors:
Money(BigDecimal amt, String currency);
Money(double amt, String currency);
Note: *currency is a string name of a currency - 'USD', 'Euro', …*

Methods:
String getCurrency(); // get currency type string
String toString(); // get amount with standard formatting
BigDecimal decValue(); // get numeric amount
double doubleValue(); // get numeric amount

We can then use the Money class to represent the amount in a sales order table:

```
CREATE TABLE sales_orders
(ord_id int,
 cust_id int REFERENCES customers,
 ord_date date,
 ord_amt Money, -- the data type of ord_amt is the Money class
 PRIMARY KEY(cust_id, ord_id)
);
```

In the new table - *sales_orders*, the *ord_amt* column uses the Java class Money as its type. In each row of the *sales_orders* table, the *ord_amt* column will either be null (empty) or contain a Java object instantiated from the Money class.

A SQL command manipulates an

Java Objects

Continued

object column by calling its methods:

```
SELECT sales_orders.cust_id,
sales_orders.ord_id,
sales_orders.ord_amt.toString()
FROM   sales_orders
WHERE
sales_orders.ord_amt.getCurrency
() = 'USD';
```

Object columns are assignable to other object columns.

The NEW clause provides values for an object column:

```
INSERT INTO sales_orders
VALUES (2451, 1, current_date,
NEW Money(2.99, 'USD'));
UPDATE sales_orders
SET    ord_amt = NEW
Money(5000, 'Euro')
WHERE cust_id = 5444
AND ord_id = 4;
```

A method returning a database object can also supply a new value for an object column.

You can also call class or static methods directly, using the class name. For example, the Java class, Money, is cataloged in the database and has a static method - convert() that converts a Money object from one currency to another. A query can use it to convert all order amounts to the same currency:

```
SELECT cust_id, ord_id,
Money.convert(ord_amt, 'Euro')
FROM   sales_orders;
```

Java Methods in the Database

When the database class is created it includes the methods from the original Java class. As a result, SQL commands can include calls to Java methods.

Static method calls can use the form:

[[catalog-name .] schema-name .] class-name . method-name (...)

class-name must be the database name of a cataloged Java class.

Instance method calls use the form:

(instance-expression) . method-name (...)

instance-expression is a SQL expression returning an object.

In the rows of the database table, the value of a column defined with Java class is an instance (object) of the Java class. Instances are created using the constructor for the class. Column values are active instances and their methods are callable in SQL commands. When the client retrieves a column value defined as a Java class, it is an active object that is often executed in the Client's JVM. Both class and instance methods may be accessible.

Stored Procedures in Java

Since the beginning, stored procedure languages have been proprietary to each database vendors, with no commonality. Using more portable languages, like C++, for server procedures has raised issues of safety (an errant procedure could crash the server) and security. Now, with most DB vendors supporting it, Java is becoming the stored procedure language of choice, promising portability and safety.

Stored procedures can be implemented as Java methods. A client application calls a Java stored procedure through JDBC or ODBC using standard syntax. The server translates this syntax into direct calls to user defined Java methods cataloged in the database.

Conclusion

Java objects in SQL databases are an excellent synergy of Object-Oriented Java development and SQL Relational Databases. It greatly extends the expressive power of SQL while making object-oriented database techniques widely available.

The consistent use of Java in all layers of a database application delivers not only object persistence and convenient manipulation of objects to object-oriented applications but also the power of objects integrated with a relational database and SQL. Java, as the basis for the object capabilities, ensures the widest availability of services and flexibility for database application developers.

Lee Fesperman is a leading database developer and creator of FirstSQL/J. This ORDBMS provides SQL 92 Intermediate Level and full object capabilities in the database using Java Classes. More information at www.firstsql.com.