Beyond JDBC™

JDBC is a standard Java™ application programming interface to relational databases defined by Sun Microsystems. Although a useful technology, the use of JDBC forces application developers to generate SQL statements explicitly. This requires a lot of hand-coding of SQL statements and processing the results. It is a very mundane, error-prone and time-consuming job.

JDX provides an efficient, intuitive and object-oriented interface to relational data. JDX exploits the JDBC standard to talk to relational backends but application programmers are freed from having to deal with the complexity of working with two different programming paradigms (object-oriented Java and relational SQL).

Imagine the productivity gains resulting from such an intuitive programming interface, avoiding impedance mismatch between two different programming models, greatly reduced program size, ease of maintenance, having tools to generate relational schema from Java class definitions and vice-versa…!

Some of the issues to consider while using raw JDBC

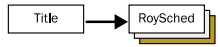
- Generation of SQL statements (SELECT, INSERT, UPDATE and DELETE) for each class
  - Do you have to write these statements manually ?
  - What if you have hundreds of classes for your application?
- Hard coding of database column names **Aaargh!**
- What if a new attribute is added to a class or an attribute name changes?
  - All corresponding statements have to be updated.
- What if an attribute type changes?
  - The getXXX call has to be changed appropriately.
  - May involve database changes also.
- What if a class hierarchy is involved (e.g., a class Political Title may inherit from Title and its objects may come from a different table)?
  - We have to potentially collect objects from multiple tables. Lots of changes at many levels.
- What if the class structure is more complex (more references, more levels)?
  - The code becomes exponentially complex!
- What if we want to do directed queries for a complex object (i.e., follow some references and ignore a few of them etc.)?
  - How to specify such a query?
  - Do we repeat the hard-coded SQL statements in different parts of the code ?
- How easily can this code be maintained/enhanced ?
- ***Wouldn't you rather be devoting more time to business logic?***

JDX™
(J-Database Exchange™)

SoftwareTree™
Simplify Data Integration

www.softwaretree.com

JDBC vs. JDX Example: Assume 2 classes - Title and RoySched. Each Title object has an array of RoySched objects. Primitive attributes of Title objects come from titles table and that of RoySched objects come from roysched table. We are trying to retrieve Title object(s) corresponding to a title_id stored in a String variable 'tid'.JDBC Code.



JDX™
(J-Database Exchange™)

JDBC Code

```
// Assuming a Connection object 'con'
// has been obtained to the database
// Retrieve the Title object

Statement stmt = con.createStatement();

// First fetch the titles table row

String query = "SELECT title_id, type, price,
title, ytd_sales, pub_id," + " pubdate, royalty,
advance, notes FROM titles" + " WHERE
title_id = '" + tid + "'";

ResultSet rs = stmt.executeQuery(query);

Title title = new Title();

while (rs.next()) {
     title.title_id = rs.getString("title_id");
     title.type = rs.getString("type");
     title.price = rs.getBigDecimal("price", 2);
     title.title = rs.getString("title");
     title.ytd_sales = rs.getInt("ytd_sales");
     title.pub_id = rs.getString("pub_id");
     title.pubdate =
       rs.getTimestamp("pubdate");
     title.royalty = rs.getInt("royalty");
     title.advance =
       rs.getBigDecimal("advance",2);
     title.notes = rs.getString("notes");
     break; // Simplifying assumption - only one
       // row is returned.
}

// Now fetch all the corresponding roysched
// table rows
Vector royScheds = new Vector();
RoySched roySched;

query = "SELECT title_id, lorange, hirange,
royalty FROM roysched" + " WHERE
title_id='" + title.title_id + "'" + " ORDER
BY royalty";

rs = stmt.executeQuery(query);

while (rs.next()) {
   roySched = new RoySched();
   roySched.title_id = rs.getString("title_id");
   roySched.lorange = rs.getInt("lorange");
   roySched.hirange = rs.getInt("hirange");
   roySched.royalty = rs.getInt("royalty");
   royScheds.addElement(roySched);
}

stmt.close();

// Initialize title's roysscheds attribute with
// the collection of roysched objects

title.roysscheds = new
            RoySched[royScheds.size()];
royScheds.copyInto(title.roysscheds);
```

JDX Code

```
// Assuming a handle 'jdx1' to the JDX service
// object has been obtained.
// Retrieve the Title object(s). In general,
// many qualifying objects may be returned.

Vector queryResults = jdx1.query("Title",
"title_id = '" + tid + "'", -1, 0, null);

Title title = (Title) queryResults.firstElement();
```

By using JDX, similar efficiencies are also gained for inserting, updating or deleting objects.

JDX, J-Database Exchange, Software Tree logo are trademarks of Software Tree. Java and JDBC are trademarks of Sun Microsystems, Inc.

Software Tree™
Simplify Data Integration

Software Tree, Inc.
650 Saratoga Avenue, San Jose, CA 95129.
Tel: 408-557-6769. Fax: 408-557-6799

Copyright © 1997-2002, Software Tree, Inc., All rights reserved

www.softwaretree.com