Level: Advanced
Works with: Notes/Domino
Updated: 01-Apr-2003

by
Ken
Yee

In **Part 1** of this article series, we covered how to integrate Amazon Web Services with Notes/Domino 6. We're sure a lot of readers would like to know how a similar application is created with J2EE because IBM has focused many of its efforts on that technology. Thus, in this article, we cover how to reproduce the Notes application built in the first article using J2EE technologies with an eye on how the techniques are different from Domino. This article is intended to give Domino developers an overview of the different ways a similar **Amazon Web Services** application can be developed using J2EE. This article assumes that you are an experienced Java developer.

First, we'll look at how to create the application using Dreamweaver and JavaServer Pages (JSPs); this technique is used for the simplest of Web applications for reasons we'll clarify later. Then, we'll look at how to create the application using Entity JavaBeans (EJBs), JavaBeans, and Struts; this technique is typically used for more robust and complex Web applications.

J2EE technologies and frameworks

The J2EE world has many different technologies and frameworks, for example, Cocoon (XML/XSLT), Velocity, ColdFusion MX, Java Server Faces, and so on, that run on the J2EE platform. Of course, the J2EE platform allows you to use any of these frameworks on any platform that J2EE runs on, giving you all the promise of a multi-platform environment.

A Domino server can also have a hybrid J2EE/Domino design that uses Domino forms for displaying the user interface and Java servlets to handle the form processing. This provides more flexible form handling and slightly better performance than a pure Domino application, but may be considered oddball by pure-J2EE developers.

Another alternative for a Domino/J2EE hybrid architecture uses a J2EE server (specifically JSP, such as Tomcat) to deliver the user interface, but Domino databases to store data. This is done using Domino JSP custom tags and was discussed in the *LDD Today* interview, "**Jeff Calow on new Web technologies in Domino 6**." Again, this provides slightly better performance than a pure Domino application, but most developers use it for improved control over the graphical layout of a Web site because Domino doesn't provide per-pixel placement in its native Web rendering.

These hybrid techniques don't provide a significant performance improvement over a site created with Domino, but they do provide alternatives that involve less work than a pure J2EE application, and this alternative is a comfortable way for Domino developers to try J2EE technology. We'll cover the hybrid Domino/J2EE technique using the Lotus Domino Toolkit for WebSphere Studio in Part 3 of this article series and use it to develop a Web UI for our Amazon books data which is stored in a Notes database.

What is J2EE?

J2EE is a standard set of technologies and APIs that Sun Microsystems defines as requirements before a J2EE server can be called compliant. There have been multiple versions of J2EE; the latest as of this writing is version 1.3; version 1.4 is in beta release. The technologies that are relevant from a Web development perspective are servlets, JSPs, JavaBeans, and EJBs.

For simple Web applications, you can use servlets and JSPs. Servlets and JSPs are supported by the Apache Jakarta Project's Tomcat engine. Both servlets and JSPs are more efficient than Notes/Domino agents because they are only loaded once when they are first used; by contrast, Java agents are reloaded each time they are called, so they are more processor intensive. For JSP support, you should be most concerned with JSP version 1.1 support. This version added JSP custom tags in which you do not need to code logic into JSPs, so you can treat JSPs as purely a presentation layer.

EJBs are typically associated with J2EE applications. They provide a way to wrap database access and business logic, so they can be distributed on multiple servers. When properly done, the database access is buried behind a business logic object so that Web site developers who use JSPs don't know how the underlying database is organized.

A good deal of the power of J2EE comes from the ability to partition each layer onto separate servers. Whereas Domino runs the database, business, and presentation/UI layers all on one Web server, J2EE separates each of these three pieces onto separate servers. In addition, each piece can be clustered so that you have redundancy at the layer level. Although you can cluster Domino servers, you can't cluster as finely or deeply, so a properly designed J2EE application can inherently scale further than a similar Domino application because you can spread it onto more machines. The downside is that this comes at a much higher cost in development and complexity.

Using an RDBMS database

As most Domino developers know, putting a few fields onto a form defines a Domino database. This design can be ad-hoc (that is, forms and views are designed without a formal design process) or designed using a formal process; you can add fields at will, and especially, change fields from single value to multivalue. Field sizes do not have to be specified up front, although you have to know the limitations of field sizes. After you create forms, views can display these fields, including multivalue fields, easily.

In the RDBMS (relational database management system) world, you have to put more effort into the initial design of the database because changing the layout of tables is more painful, primarily because of all the foreign key links and the field sizes that you must declare up front. In some cases, you may have to unload the tables, delete all the data, change the database design, then reload the data.

Multivalue fields also deserve further discussion. You can typically map a Domino document to a row in an RDBMS table. Each field of a Domino document can be mapped to a column in the table. However, each column of a table (that is, each field) can only store one value. In an RDBMS, you have to create a separate table with two columns. One column links to the form (the row in the table) using a foreign key. The other column links to another table that lists all the possible values for that multivalue field.

Security

Another major feature that Domino's infrastructure provides for you is user management and document-level security. User roles and access control are inherent in Domino's database structure. While an RDBMS has similar management of users (adding/deleting/grouping), most do not have custom role support; their roles are specific to RDBMS tasks, for example, insert row, delete row, change schema, and so on. To make matters worse, in a typical Web application, you may not use specific user logins to a database because all database access is done through one user so that you can pool database connections for performance.

Document-level security in an RDBMS is cobbled together by limiting database access to stored procedures with custom selects and views. There is no row level access control in an RDBMS. If you want to simulate this, you have to add your own table that lists the users or roles that have access to that particular table row.

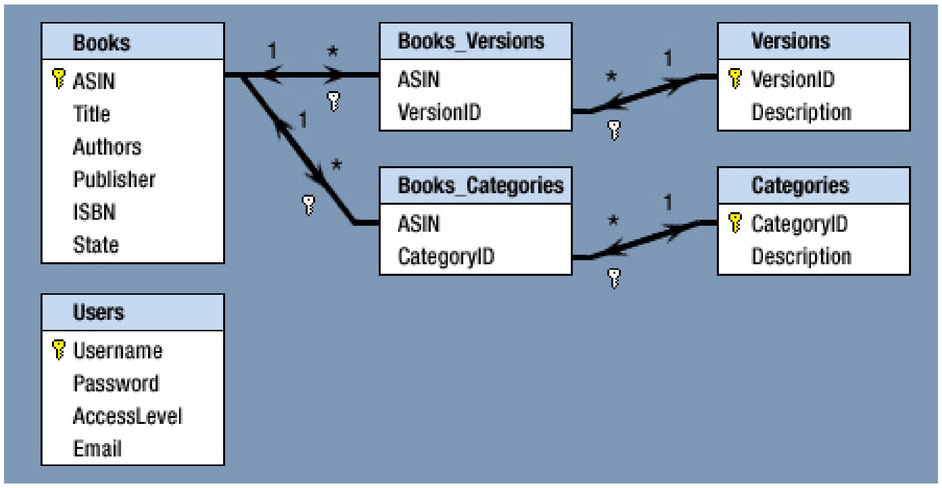
As you can see, most RDBMS Web applications won't have nearly the same level of security as Notes applications because Notes inherently has a tighter security model. However, Web applications can depend on a single point of access—the Web page. By depending on this single point of access, security can be added to RDBMS Web applications before database access. Unfortunately, depending on this one access point also prevents Web applications from being secure if developers attempt to put these on a disconnected laptop because a user can access the local database directly using any application that supports ODBC. This prevents standard Web applications from being used securely via replication, which Notes users take for granted.

Macromedia Dreamweaver provides a very simplistic form of security at the user interface level. Dreamweaver lets

Web developers add actions to Web pages—ASP, JSP, Cold Fusion—to provide login, to access control to pages, to hide parts of a Web page depending on what access level you have, and so on. The requirement is a user table that includes user name, password, and access level fields. There can be only one access level per user, so Dreamweaver is not role-based. All the security happens at the Web page level.

EJBs provide a method-level security model. Users can be assigned roles in the J2EE server, and the application developer can specify which roles can access particular methods in an EJB. Because EJBs are used by Web page designers or JavaBean writers (for encapsulating business logic), this provides an extra layer of security in case Web pages are hacked. In addition, the J2EE server can protect specific URLs (for example, everything in the /admin directory), so they can only be accessed by people assigned certain roles, such as the Admin role. Again, this occurs at a layer before RDBMS access, so if the user has direct access to the RDBMS, he can bypass any security you place in EJBs.

Database schema
A major part of the design of a multitier Web application is the design of the relational database that the application's data is stored in. The design has to support all the needed features of the Web application, and it's important that it be done right the first time because changing it is more difficult than changing design in Domino where we can add a few fields to a form or view at will. We'll set up our tables according to the following diagram:



Here is the SQL used to generate the tables. In this first part, the SQL code removes previous versions of the database in case the database design changed:

```
DROP TABLE "Books_Categories"
DROP TABLE "Books_Versions"
DROP TABLE "Versions"
DROP TABLE "Categories"
DROP TABLE "Books"
DROP TABLE "Users"
```

The following section of code creates all of the tables for the application:

```
CREATE TABLE "Books"
(
	"ASIN" varchar(30) NOT NULL PRIMARY KEY,
	"Title" varchar(255) NOT NULL,
	"Authors" varchar(255),
	"Publisher" varchar(255),
	"ISBN" varchar(30),
	"State" varchar(30),
	"LastFound" datetime
)

CREATE TABLE "Versions"
```

```
(
    "VersionID" int NOT NULL PRIMARY KEY,
    "Description" varchar(50) NOT NULL
)

CREATE TABLE "Categories"
(
    "CategoryID" int NOT NULL PRIMARY KEY,
    "Description" varchar(50) NOT NULL
)

CREATE TABLE "Books_Versions"
(
    "ASIN" varchar(30) NOT NULL,
    "VersionID" int NOT NULL
)

CREATE TABLE "Books_Categories"
(
    "ASIN" varchar(30) NOT NULL,
    "CategoryID" int NOT NULL
)
```

This next section of code adds all of the foreign key links seen in the previous diagram:

```
ALTER TABLE "Books_Versions" ADD CONSTRAINT "FK_book_version_ASIN_key" FOREIGN KEY (
    "ASIN"
)
REFERENCES "Books" (
    "ASIN"
)
ALTER TABLE "Books_Versions" ADD CONSTRAINT "FK_book_version_key" FOREIGN KEY (
    "VersionID"
)
REFERENCES "Versions" (
    "VersionID"
)

ALTER TABLE "Books_Categories" ADD CONSTRAINT "FK_book_categories_ASIN_key" FOREIGN KEY (
    "ASIN"
)
REFERENCES "Books" (
    "ASIN"
)
ALTER TABLE "Books_Categories" ADD CONSTRAINT "FK_book_category_key" FOREIGN KEY (
    "CategoryID"
)
REFERENCES "Categories" (
    "CategoryID"
)
```

The following code creates a table for Dreamweaver's access control management. We discuss Dreamweaver's access control management later in this article.

```
CREATE TABLE "Users"
(
    "Username" varchar(100) NOT NULL PRIMARY KEY,
    "Password" varchar(100) NOT NULL,
    "AccessLevel" varchar(100),
    "Email" varchar(100)
)
```

This section initializes parts of the database; this is the equivalent of adding selectable values to a dialog list field in Notes:

```
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (1, 'R3')
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (2, 'R4')
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (3, 'R4.5')
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (4, 'R4.6')
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (5, 'R5')
INSERT INTO "Versions" ("VersionID", "Description")
	VALUES (6, 'R6')

INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (1, 'Administration')
INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (2, 'Education')
INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (3, 'Infrastructure')
INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (4, 'Programming')
INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (5, 'Platforms')
INSERT INTO "Categories" ("CategoryID", "Description")
	VALUES (6, 'User')

INSERT INTO "Users" ("Username", "Password", "AccessLevel")
	VALUES ('admin', 'password', 'Administrator')
```

We're now ready to load our Amazon Web Services data into this database. If you haven't already installed Amazon Web Services, you can **download the developer's kit** from Amazon.com.

Importing from Amazon Web Services

To import the Amazon Web Services book information into this database, we create an appropriate class using the BookTracker interface we defined in Part 1 of this article series. We use JDBC to communicate with the database and to load the tables with the Amazon information.

```
import java.sql.*;

// implements the book update object using a JDBC database as a data store
public class JdbcBookTracker implements BookTracker {
	Connection connBooks;
		PreparedStatement findBook;
		PreparedStatement updateBook;
		PreparedStatement addBook;
```

The constructor sets up the RDBMS connection:

```
	// constructor
	public JdbcBookTracker(String driver, String datasource, String username, String password) throws
	Exception {
		// initialize JDBC connection
	Driver driverBooks = (Driver)Class.forName(driver).newInstance();
	connBooks = DriverManager.getConnection(datasource,username,password);
	}

	// Java pseudo-destructor
	public void finalize() throws Exception {
		// clean up JDBC connections
			connBooks.close();
		}
```

This is the "meat" of the database update code. It's similar to the code needed to load the Amazon Web Services data into a Domino application:

```
    // update book from amazon query
    public void updateBook(
        String title,
        String ASIN,
        String ISBN,
        String authors,
        String publisher,
        String rating,
        int numreviews)
        throws Exception {
    // check for blank ISBN
        if (ISBN == null) {
        System.out.println(title + " has a null ISBN!");
        }
    // check for blank ASIN
        if (ASIN == null) {
            System.out.println(title + " has a null ASIN!");
        }

    // see if we can find the book
        findBook = connBooks.prepareStatement("SELECT Title FROM Books WHERE ASIN = ?");
    findBook.setString(1, ASIN);
    ResultSet booksFound = findBook.executeQuery();
    boolean isInDatabase = !booksFound.next();
    booksFound.close();
    findBook.close();
        if (isInDatabase) {
            System.out.println("Adding " + title + " to database");
            // new book, so we have to add it to the database
            addBook = connBooks.prepareStatement("INSERT INTO Books
            (ASIN,Title,Authors,Publisher,ISBN,State,LastFound) VALUES
            (?,?,?,?,?,?,CURRENT_TIMESTAMP)");
                addBook.setString(1, ASIN);
                addBook.setString(2, title);
                addBook.setString(3, authors);
                addBook.setString(4, publisher);
                addBook.setString(5, ISBN);
                addBook.setString(6, "A");  // A for added
    //addBook.setTimestamp(7, new Timestamp(java.util.Calendar.getInstance().getTimeInMillis()));
              addBook.execute();
                addBook.close();
        } else {
                System.out.println("Updating timestamp for " + title);
                // update book timestamp
                updateBook = connBooks.prepareStatement("UPDATE Books SET LastFound =
                CURRENT_TIMESTAMP WHERE ASIN = ?");
                updateBook.setString(1, ASIN);
                updateBook.execute();
                updateBook.close();
            }
        }
}
```

Calling this is simply done with this code:

```
BookTracker tracker = new JdbcBookTracker("sun.jdbc.odbc.JdbcOdbcDriver", "jdbc:odbc:Amazon",
"sa", "");
getAmazonBooks("lotus domino", tracker);
```

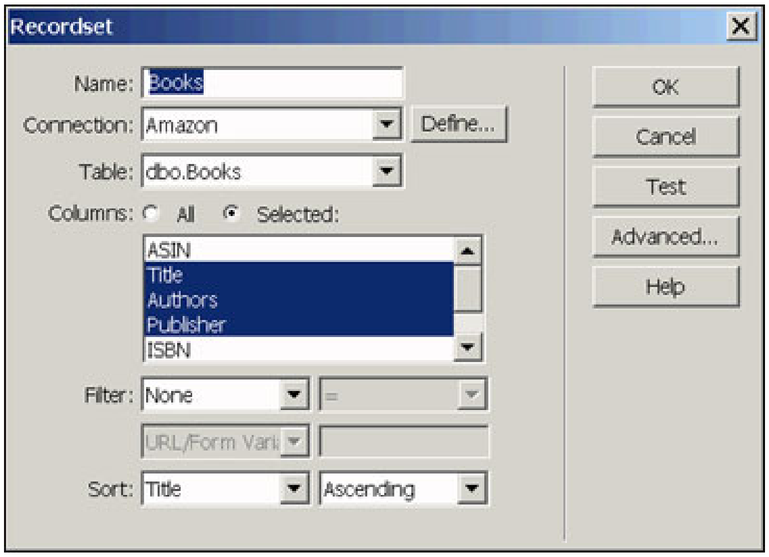
assuming you have an ODBC source named Amazon for the database with the schema we defined above, the default user name/password for a lot of databases is "sa"/"(that is, the password is blank and the admin user name is "sa"). After we have the data loaded, we can provide a user interface for our J2EE version of Amazon Web Services.

Forms and views
Forms and views are elements we take for granted in the Domino environment. For other Web technologies, you typically have to create three items: a page to display a document in read mode, a page to display a document in edit mode, and a page to display a view. You also have to create separate administration Web pages to handle users, roles, and groups. Most Web environments have a concept similar to Hide-Whens. These Web environments don't have the extensive paging, categorizing, or searching support of Domino views. However, you do have more control over how the final output looks than you have with Domino elements.

Using Macromedia Dreamweaver

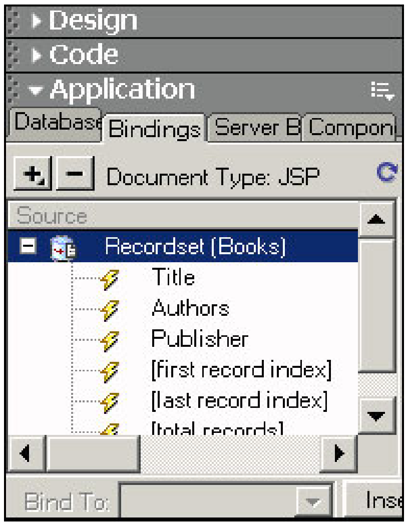
Macromedia Dreamweaver provides a rich visual editing environment for HTML as required for all Web development environments. Dreamweaver stands out from other tools by providing code editing, including macros for Web sites that use a page language such as ASP, ASP.Net, JSP, ColdFusion, and PHP. The extension macros allow you to automate login, implement Hide-When-like functionality, display result sets automatically, create custom result sets, and so on. You can edit and add your own code macros. There are also "snippets" which are code (HTML or script) macros that can be shared across different pages. For JSP support, Dreamweaver provides method/tag completion for both JavaBeans and JSP custom tag libraries.

Views in Dreamweaver
Let's start by creating a view that shows all the books. Start by adding a RecordSet in the Server Behaviors dialog to your JSP and choosing which table and table columns you want to show on the page:

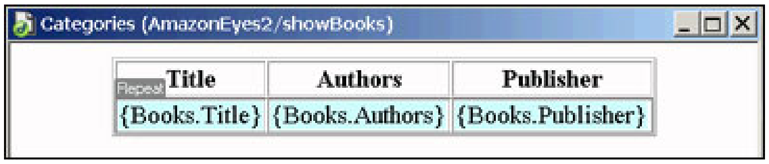

Recordset

Name: Books
Connection: Amazon
Table: dbo.Books
Columns: All [✓ Selected:]
ASIN
Title
Authors
Publisher
ISBN
Filter: None =
URL/Form Vari
Sort: Title Ascending

You can see that we've chosen to get the Title, Authors, and Publisher field from the database table Books; we'll also sort the results based on the Title field. After you do that, you can drag and drop fields from the Bindings window into your Web page:

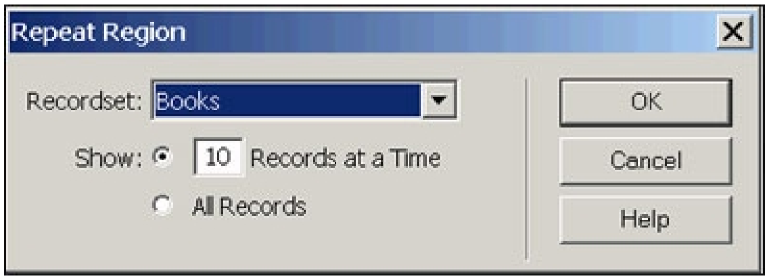


Because we are creating a view, we should create a table to display each document row:



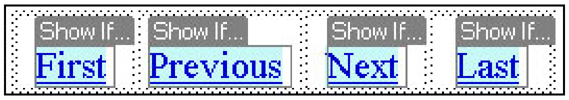
Title	Authors	Publisher
{Books.Title}	{Books.Authors}	{Books.Publisher}

Notice that there is a little "Repeat" indicator and highlighting for one row of the table to indicate that this is a Repeating Region. This is how Dreamweaver shows that the RecordSet will be displayed by iterating through the records and repeating that section of HTML. You can also specify how many of the records are displayed in the Repeat Region dialog box:

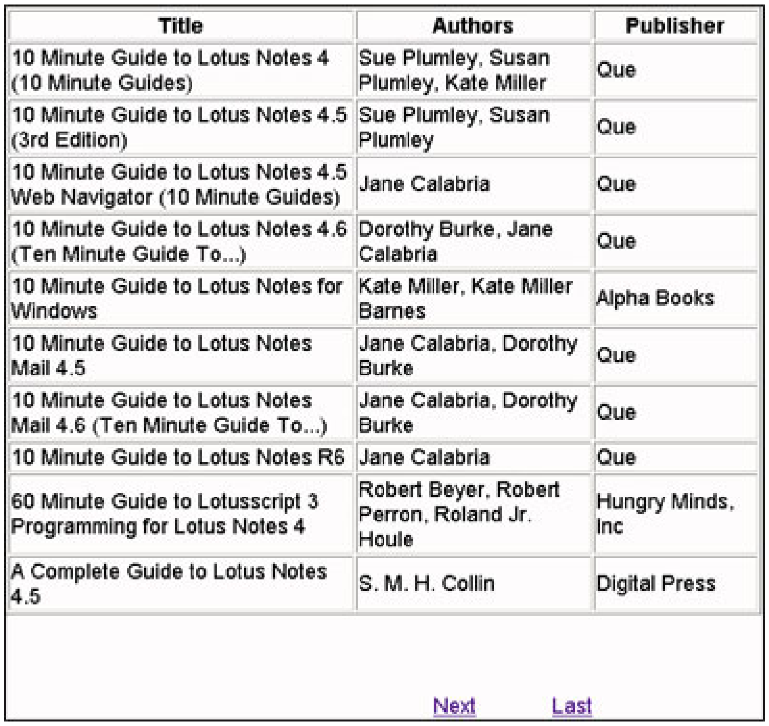


Recordset: Books
Show: [✓ 10 Records at a Time] All Records

Finally, you can add a navigation bar to page through the results. We can use a text or graphical navigation bar, but we'll use text for simplicity:



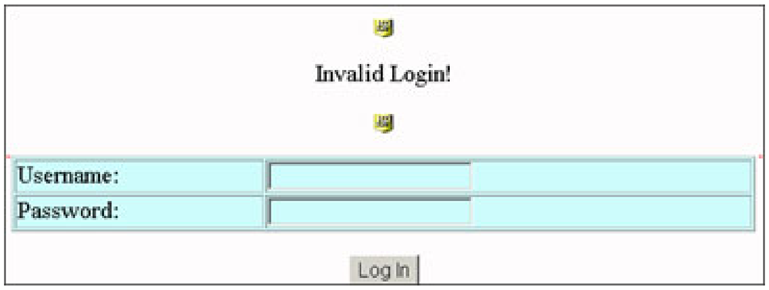
Now we can see what it looks like when we display this page in a Web browser:



Title	Authors	Publisher
10 Minute Guide to Lotus Notes 4 (10 Minute Guides)	Sue Plumley, Susan Plumley, Kate Miller	Que
10 Minute Guide to Lotus Notes 4.5 (3rd Edition)	Sue Plumley, Susan Plumley	Que
10 Minute Guide to Lotus Notes 4.5 Web Navigator (10 Minute Guides)	Jane Calabria	Que
10 Minute Guide to Lotus Notes 4.6 (Ten Minute Guide To...)	Dorothy Burke, Jane Calabria	Que
10 Minute Guide to Lotus Notes for Windows	Kate Miller, Kate Miller Barnes	Alpha Books
10 Minute Guide to Lotus Notes Mail 4.5	Jane Calabria, Dorothy Burke	Que
10 Minute Guide to Lotus Notes Mail 4.6 (Ten Minute Guide To...)	Jane Calabria, Dorothy Burke	Que
10 Minute Guide to Lotus Notes R6	Jane Calabria	Que
60 Minute Guide to Lotusscript 3 Programming for Lotus Notes 4	Robert Beyer, Robert Perron, Roland Jr. Houle	Hungry Minds, Inc
A Complete Guide to Lotus Notes 4.5	S. M. H. Collin	Digital Press

Note that it even hides the Prev/First parts of the navigation bar if you're on the first page. The navigation bar also understands how to page through results. Because your repeat region is set to display only 10 results at a time, when you click on Next, the navigation bar displays the next 10 results as expected. It also hides the Next/Last parts of the navigation bar when you're on the last page.

User authentication in Dreamweaver
To support editing a book and putting it in specific categories like we did in Notes, you have to provide a separate Web UI. However, first we have to create a login page because we don't want every user to be able to change book categories. We do this by providing a page with a few login fields and an error message if the login fails:

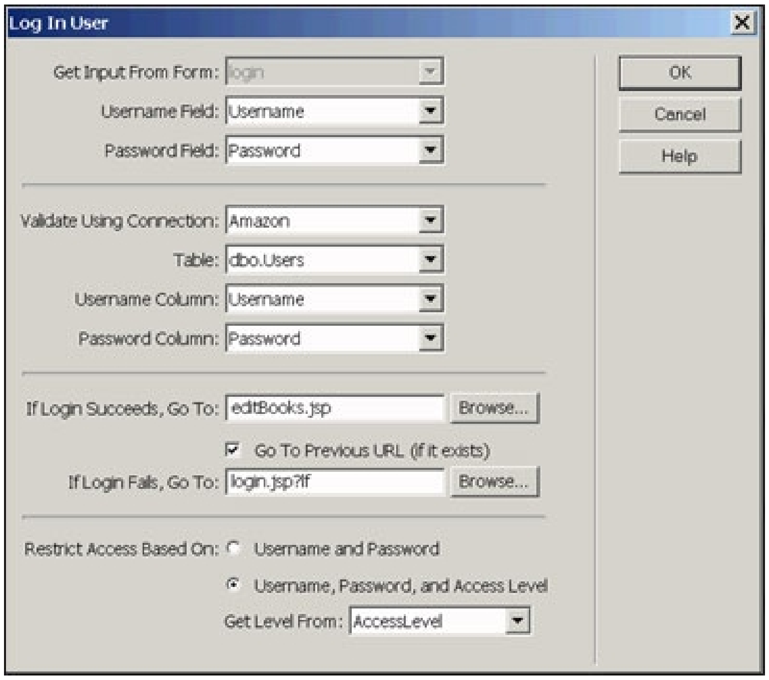


Invalid Login!

Username:	
Password:	

You do this by creating a form area, then adding a table with fields and field labels. The rectangular red area indicates the borders of a form. The form action is set up to jump back to this login page, so code has to be added to handle the actual login. We've also added a message (the text surrounded by the JSP icons) to show the user that the login failed.

We next add a Login Server Behavior to this page:



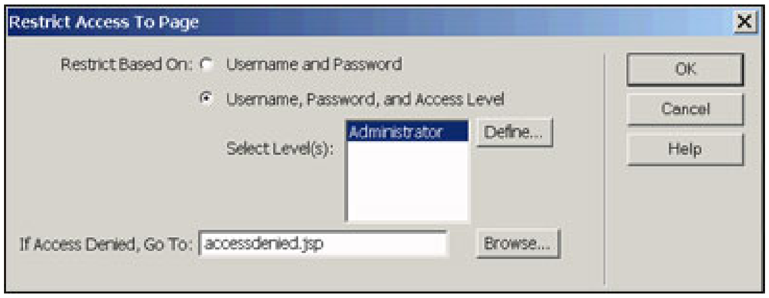
Get Input From Form: login
Username Field: Username
Password Field: Password
Validate Using Connection: Amazon
Table: dbo.Users
Username Column: Username
Password Column: Password
If Login Succeeds, Go To: editBooks.jsp
[✓] Go To Previous URL (if it exists)
If Login Fails, Go To: login.jsp?lf
Restrict Access Based On: ( ) Username and Password [✓ Username, Password, and Access Level]
Get Level From: AccessLevel

As you can see, this lets you specify which table in our database is used for authentication and the access level of the logged in user (the AccessLevel field). It also lets you specify where the user should go if he logged in successfully or not. In the case of a bad login, we added a querystring parameter of "lf." We also added some custom code (that's what those JSP icons indicate) to handle this:

```
<% if (request.getParameter("lf") != null) { %>
      <p align="center">Invalid Login!</p>
<% } /* end request.getParameter(lf) != null */ %>
```

Unfortunately, there is no built-in Dreamweaver Server Behavior for this as there is for hiding parts of a view's navigation bar. Because of this, you need to have an understanding of Java to implement the Dreamweaver equivalent of Hide-When's unless you're using a JSP custom tag library which has support for Hide-When's.

Access control in Dreamweaver
Dreamweaver has a convenient way of limiting access to specific pages unless a user has a specific access level. This is done by adding a Restrict Access Server Behavior to the page:



Restrict Based On: ( ) Username and Password [✓ Username, Password, and Access Level]
Select Level(s): Administrator
If Access Denied, Go To: accessdenied.jsp

In this case, only users with the Administrator access level in his or her login can access this page. The code that actually does this (you won't see this in the visual representation of the page) is:

```
<%
// *** Restrict Access To Page: Grant or deny access to this page
String MM_authorizedUsers="Administrator";
String MM_authFailedURL="accessdenied.jsp";
boolean MM_grantAccess=false;
```

```
if (session.getValue("MM_Username") != null && !session.getValue("MM_Username").equals("")) {
if (false || (session.getValue("MM_UserAuthorization")=="") ||
    (MM_authorizedUsers.indexOf((String)session.getValue("MM_UserAuthorization")) >=0)) {
 MM_grantAccess = true;
}
}
if (!MM_grantAccess) {
String MM_qsChar = "?";
if (MM_authFailedURL.indexOf("?") >= 0) MM_qsChar = "&";
String MM_referrer = request.getRequestURI();
if (request.getQueryString() != null) MM_referrer = MM_referrer + "?" + request.getQueryString();
MM_authFailedURL = MM_authFailedURL + MM_qsChar + "accessdenied=" +
java.net.URLEncoder.encode(MM_referrer);
response.sendRedirect(response.encodeRedirectURL(MM_authFailedURL));
return;
 }
%>
```

As you can see, it is intimately tied to the Login Server Behavior.

Dreamweaver weaknesses
Dreamweaver provides an adequate environment for editing page language Web pages (though it doesn't generate deployment descriptors for you). Unfortunately, the out-of-the-box behaviors violate one of the rules for good Web architecture: Keep the UI and logic separate. Doing this works for simpler sites, but for complex and robust sites, you really must keep the two separate. By mixing database access and code directly into the JSPs, it makes management of the Web site more difficult unless you stick with the Dreamweaver development environment or write your own behaviors. Even if you continue to use the Dreamweaver environment, it can get messy because when you upgrade to a new version of Dreamweaver, the Server Behaviors may change, so it can't match the code you already have, and then you'll have to edit the JSP code directly.

JSP tags
JSP custom tags and JavaBeans make it possible for you to separate the user interface and workflow logic. JSP custom tags look like custom HTML tags to Web developers. The JavaServer Page Standard Tag Library (JSTL) is a required part of JSP 1.3. JSTL includes the tags you can use for looping through collections and for implementing Hide-When's. It also provides XML parsing and transformation tags and lets you reference objects similarly to JavaScript. Lastly, it provides direct SQL access to JDBC sources, though you should avoid these tags if you're trying to separate the UI and logic.

As an example, our login page displayed an "invalid login" message if login failed. However, it required knowledge of how to write a Java if statement and what object to get a querystring parameter from:

```
<% if (request.getParameter("lf") != null) { %>
      <p align="center">Invalid Login!</p>
<% } /* end request.getParameter(lf) != null */ %>
```

Rewritten using JSP custom tags, this looks like:

```
<c:if test="${!empty param.lf}">
      <p align="center">Invalid Login!</p>
</c:if>
```

which is quite a bit simpler and doesn't require that Web page designers understand Java syntax.

All database access should be hidden inside JavaBeans to prevent tying the UI of a Web site to how the data is arranged in a database. Decoupled data keeps a change in the database schema from causing a ripple effect of changes in workflow logic and the user interface. When done properly, the only thing that JSP Web page designers need to know is how to read JavaBean values—so they can be displayed on appropriate parts of the page—and which URL (usually a servlet) to put into the POST section of the page.

EJBs and wrapping database access

Entity JavaBeans are used to wrapper database access. For simpler databases like ours, an EJB object represents each row of a table. Unfortunately, EJBs have been very tedious to write because you have to create three classes (and separate files) for each EJB: the home, the local interface, and the remote interface. The home class lets users create new instances of the bean or find existing instances (such as rows in a database). The local and remote interfaces are similar to Microsoft COM proxies. They allow access to the EJB in your local Java Virtual Machine (JVM) or on a different remote server.

Several tools are available to alleviate this pain. XDoclet is similar to Microsoft's MIDL compiler in that it generates the proxy classes from the classes that you want to use as EJBs; XDoclet also provides a database independent way of specifying queries and creates deployment descriptors for these access classes. MiddleGen provides a way to generate database access classes from the database itself. AndroMDA (formerly UML2EJB) provides a way to generate database access classes from a high level UML model of your application. Both MiddleGen and AndroMDA generate code that is then compiled by XDoclet.

XDoclet
XDoclet is a tool that lets you write only a single class for EJB for database access instead of a minimum of four classes. It does this by letting you tag your main class and methods, so the other required EJB classes are automatically generated. Here is a snippet of an EJB class that you would have to write for the Book table:

```
/**
        * @ejb.bean
        *    type="CMP"
        *    cmp-version="2.x"
        *    name="Book"
        *    local-jndi-name="amazon/BookLocalHome"
        *    view-type="both"
        *    primkey-field="asin"
        *
        * @ejb.finder
        *    signature="java.util.Collection findAll()"
        *    result-type-mapping="Local"
        *    method-intf="LocalHome"
        *    query="SELECT OBJECT(o) FROM Book o"
        *
        * @ejb.finder
        *    signature="java.util.Collection findByIsbn(java.lang.String isbn)"
        *    result-type-mapping="Local"
        *    method-intf="LocalHome"
        *    query="SELECT DISTINCT OBJECT(o) FROM Book o WHERE o.isbn = ?1"
        *    description="ISBN is not indexed."
        * @ejb.persistence table-name="Books"
        * @ejb.transaction type="Required"
        * @ejb:security-role-ref admin Administrator
        *
*/
public abstract class BookBean implements javax.ejb.EntityBean {

/**
        * Sets the title
        *
        * @param title the new title value
        * @ejb.interface-method view-type="both"
        * @ejb:permission Administrator
        */
        public abstract void setTitle(java.lang.String title);
```

You see that it defines how to map each method to a particular table and column and specifies which columns are primary keys, finder methods, and so on. You can also add various tags to generate deployment descriptors for JBoss, Weblogic, Orion, and WebSphere. The lines highlighted in bold were added manually to add role-based permission to a method (setTitle, in this case); this prevents users who are not administrators from changing the title of a book, even if the JSP invokes the EJB.

MiddleGen
MiddleGen is a tool that can automatically generate EJBs if you already have a database built. You give it a JDBC source from which it can extract database schema information. It can then generate the database access EJBs for you as well as a basic Web user interface using Jakarta Struts and all the other XML files required for a

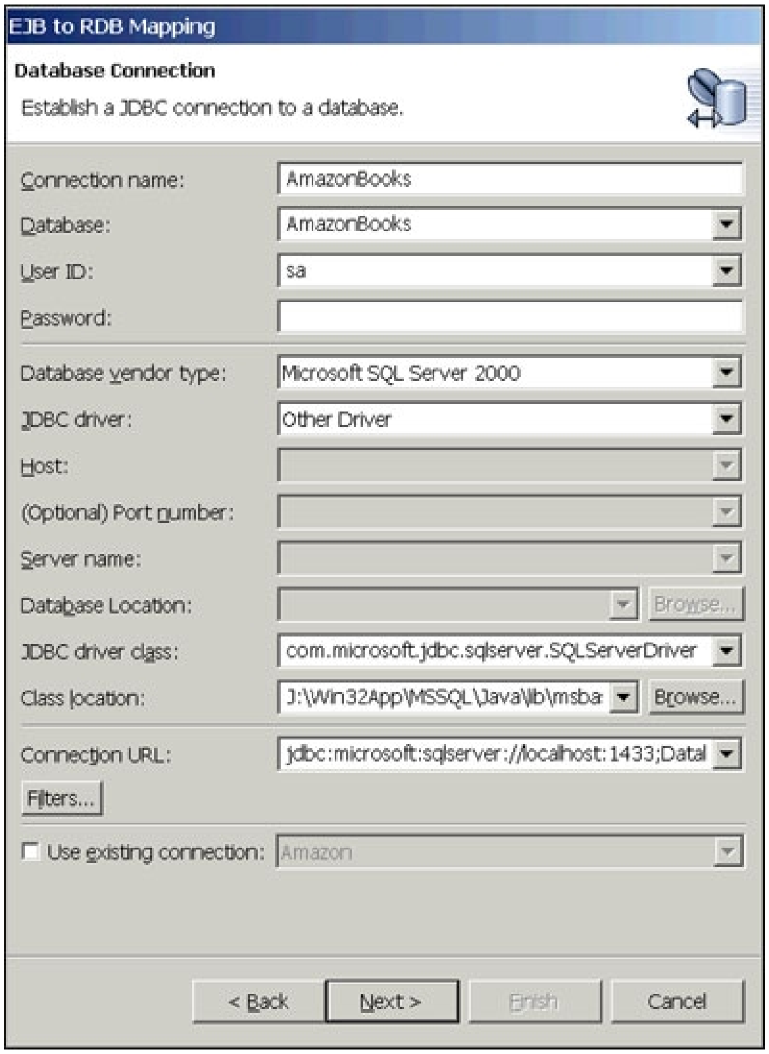
deployment EAR file you can drop into the deployment directory of your J2EE server. It does all this by creating files that can be used by XDoclet to create the local/remote interfaces. The previous code sample was generated by MiddleGen.

AndroMDA
AndroMDA takes a different approach. It lets you generate EJBs from a Unified Modeling Language (UML) project's files; UML tools can export XML which is analogous to the RTF format for word processors. This is what you'd want to do if you were creating a project from scratch and had no database design to start from. AndroMDA creates the appropriate XDoclet files for you similar to what MiddleGen does and automatically generates the appropriate EJB descriptors to handle one-to-many and many-to-many relationships.

Using WebSphere Studio

Once you've used open source tools and spent a while trying to get the right dependent tools installed, you'll appreciate how much integration IBM has done with the latest 5.0 version of WebSphere Studio. It not only provides a bottom-up approach to creating EJBs (the MiddleGen technique), but also a top-down approach in which it generates the database from existing EJBs; it also has a meet-in-the-middle approach where you map existing EJBs to specific tables/columns. There is no high level tool in WebSphere Studio that lets you create your project using UML diagrams yet, but hopefully, IBM's purchase of Rational will rectify this.

IBM has spent a lot of time creating wizards (dialog boxes that prompt users for all the information to complete a specific task) and deployment descriptor editors, though it still helps to understand how the underlying XML configuration files work. For example, after going through a connection dialog for the bottom-up EJBs:

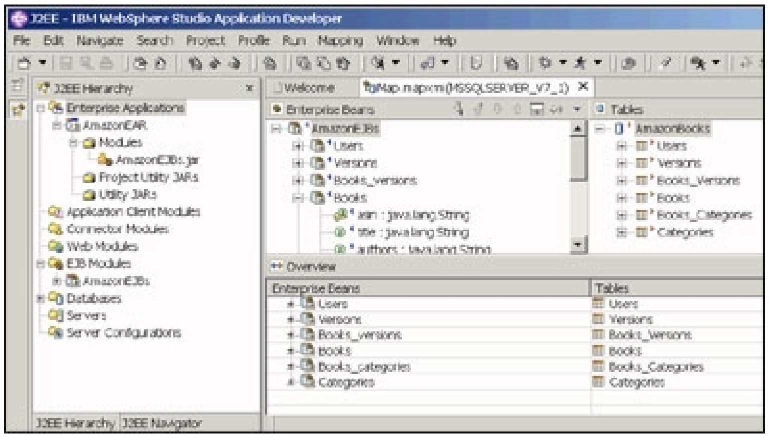

EJB to RDB Mapping

Database Connection

Establish a JDBC connection to a database.

Connection name:	AmazonBooks
Database:	AmazonBooks
User ID:	sa
Password:	
Database vendor type:	Microsoft SQL Server 2000
JDBC driver:	Other Driver
Host:	
(Optional) Port number:	
Server name:	
Database Location:	
JDBC driver class:	com.microsoft.jdbc.sqlserver.SQLServerDriver
Class location:	J:\Win32App\MSSQL\Java\lib\msba
Connection URL:	jdbc:microsoft:sqlserver://localhost:1433;Datal
☐ Use existing connection:	Amazon

You get a bunch of pre-built EJBs that map to your database (although it doesn't seem to be detecting the many-to-many tables properly and still generates EJBs for them in the current 5.0 release of WebSphere Studio Application Developer):



From the EJBs, you can create access beans and then use another wizard to generate access pages for the beans; the result is fairly simple, so you are probably better off coding your own JSPs or Struts pages once you have EJBs and/or access beans. Access beans are another layer on top of EJBs that are used to abstract access to the EJBs to look like plain JavaBeans (no Remote/Home interface issues), so the Web pages don't have to understand that they're dealing with EJBs; there are JSP custom tags to access plain JavaBeans' values, so Web page designers don't need to understand Java.

Conclusion

As you can see, development using J2EE technologies is more complicated than development in Domino. It requires more time and resources than similar applications in Domino because you need specialists to design each layer (JavaBeans/EJBs, database, and Web). The tools are evolving rapidly to ease some of the development burden, but rapid change also means incompatibilities between all the tools you need to use to take care of all the plumbing that Domino provides. In cases where you have huge databases and lots of Web activity, J2EE technologies should be a solution you consider; the separation of the UI/database/middleware and the ability to cluster each of these layers gives J2EE the headroom needed to create a huge Web site like eBay. If you're porting a Domino application to J2EE, be sure the extra investment in time and expenses is worth it.

If you venture into the standard Web development world of J2EE, keep in mind the following:

1. It's great that you have freedom of choice in the parts and plumbing (back-end database, Web UI, middleware), but this means you have to make sure they work well together. If you're adding a new piece of the software to your Web solution or development environment, be prepared to work through/around "features" and budget some time for this.
2. Be prepared to edit a lot of text files by hand. Whereas we have Notes databases with a nice user interface to configure things in the Notes world, most of the J2EE world still depends on editing XML files (like deployment descriptors) using a text editor. This includes learning Jakarta Ant, which is the Java world's equivalent of UNIX's "make" tool to help with generating EJBs.
3. Use Sun's JDBC/ODBC driver for prototyping, but don't use it for rollout. It has performance issues and other quirks (like requiring you to read column values in the same order you do the select if you're using Microsoft SQL Server for your database).
4. Errors in J2EE applications are frequently cryptic. You typically see a call stack with a Java exception, and this may not point you directly to the line that caused the problem.
5. Searching is something we haven't covered because that topic is complex enough to be a separate article. In the Domino world, enabling search is as simple as selecting a checkbox in Database Properties. In the J2EE world, you have to find a search engine, then come up with a way to feed data to the search engine, get results out, and format the results for display to the user.
6. Background agents were not covered. On an RDBMS, they're implemented via stored procedures which are not portable between databases. The newer RDBMS's can run Java stored procedures, so integration with our Amazon Web Services code is relatively simple.

We've covered quite a few techniques in a short amount of space. Each technique deserves more coverage than we've given it here, so if you want to see another article on a specific technique, please let us know.

In the next part of this series, we'll be creating a full application using the Domino WebSphere Studio Toolkit because this is the most likely approach Domino developers will be taking to integrate J2EE technologies.

ABOUT THE AUTHOR

Ken Yee has been a consultant and Lotus Business Partner since the inception of the program. He has done software development since the late 1980's and is always looking for interesting J2EE/Domino integration projects. His company, **KEY Enterprise Solutions**, has done Notes, Domino, IIS/ASP, Java, ActiveX/COM, and C++ development and administration projects for **Lotus**, **Inso/Stellent**, **Logica**, **eVelocity**, **World Bank**, and **Analysis Group**. KEY Enterprise Solutions maintains the **Notes/Domino FAQ** (the first Notes FAQ on the net) as a service to the Notes community and the **Java Servlet FAQ** for the Java community.