Data Access for the J2EE™ Platform: The JDBC™ API and the J2EE™ Connector Architecture

John Goodson
Vice President, Product Operations
DataDirect Technologies, Inc.
Optimize Access To All Data

Learn how to optimize access to all data inside your company using the data access standards for the Java™ 2 Platform, Enterprise Edition (J2EE™), the Java Database Connectivity (JDBC™) API and the J2EE Connector Architecture.
Learning Objectives

• As a result of this presentation, you will be able to:
  – Design an architecture using J2EE technology to access your data
  – Understand differences and similarities between the JDBC API and the J2EE Connector Architecture
  – Design access methodologies to your data which utilize the optimization techniques discussed today
Speaker’s Qualifications

• John leads the R&D team at DataDirect Technologies

• John is on the Expert Panel for the JDBC API and “JDBC RowSet” JSRs and also was a member of the X/Open SQL Access Group

• John was a key contributor to the ODBC, OLE DB, and JDBC specifications

• John writes in several trade journals and speaks frequently on various data access topics (including speaking at the JavaOne℠ conference every year!)
There are abundant techniques available for optimizing your data engines, but few available for optimizing the data access APIs you’ll use to get that data.
Agenda

• The JDBC API and the J2EE Connector Architecture
• JDBC Specification, Fundamentals (brief)
• J2EE Connector Architecture Fundamentals (brief)
• The Role of XML – “JDBC Web RowSet”
• Designing for Performance
• Putting it all together
Data Access Architecture for the J2EE Platform

Application Component

xml

JDBC

DB2  Oracle

JCA

SAP
The JDBC Specification: Fundamentals

- Provides access to SQL data
  - Oracle, DB2, SQL Server, Sybase, Informix
- JDBC 1.0 API designed for basic functionality with emphasis on ease of use
- JDBC 2.0 API enhanced for advanced features and server-side use
- JDBC 3.0 API “rounded out” the API by providing missing features and optimizations
JDBC 3.0 API: Highlights

- Savepoints
- Retrieval of Parameter Metadata
- Pooling enhancements
- Retrieval of auto-generated keys
- Multiple open result set objects
- BLOB/CLOB updates
- SQL3 type enhancements
Demo
J2EE Connector Architecture Fundamentals

• Provides access to all non-SQL data
  – SAP, PeopleSoft, CICS, IMS, VSAM

• CCI is the client API that programmers will use

• Current version is 1.0

• Same types of objects as the JDBC API
“JDBC RowSets”
(The RowSets method of the JDBC API)

• Provides disconnected use of relational data
• The Web RowSet is in XML format
  – Can treat as an XML object!
• Remote results with SOAP, XML Writers, etc.
• Persist results as XML
Performance Issues

• Developing code that works is hard enough without having to make it work fast!

• Data sources are different – tough to write a single application that works well against any backend or to know all the specifics of any backend

• No exceptions are thrown to say your code is running too slow
High Level Guidelines

• The two slowest things in a data access architecture are:
  – Network requests
  – Disk I/O

• A close third is the inappropriate use of metadata methods

• And lots of other little “got ya’s”
Guidelines - Connections

• Connecting to a database/EIS system is the single slowest operation inside a data centric application

• Design for and use Connection Pooling
  – Pre-allocate connections
  – Closing connections does not actually close the physical connection to the backend
  – Connecting is simply reusing an active connection which does not cause a network I/O
Guidelines – Avoid Disk Access

• Committing transactions is expensive!
  – Network I/O
  – Numerous non-sequential disk I/Os

• Turn auto-commit off after connecting
  – WSCConnection.setAutoCommit (false);

• Distributed transactions are at least 4 times slower than local transactions
Guidelines – DatabaseMetaData Methods are slow

- Compared to other JDBC API methods, DatabaseMetaData methods are slow
  - Drivers have to issue complex queries in order to return the information mandated by the specification

- Cache information returned by DatabaseMetaData methods
Guidelines – Avoid null arguments and Search patterns

• Using null arguments or search patterns to DatabaseMetaData methods results in time consuming queries and, potentially, increases network traffic due to unwanted results
Example

• ResultSet rs = conn.getTables (null, null, "mytable", null);

• Should be ....

• ResultSet rs = conn.getTables ("cat1", "johng", "mytable", "TABLE");
Guidelines – Use ResultSetMetaData instead of getColumn

- getColumn
  - Query, potentially complex, sent to backend
  - Query executed on server
  - Driver must obtain result information, bind results, and retrieve results from server

- ResultSetMetaData
  - Simple query, which evaluates to no rows, sent to server – not executed
  - Only result set information obtained
Example

- PreparedStatement pstmt = conn.prepareStatement("select * from table where 0 = 1");
  ResultSetMetaData rsmd = pstmt.getMetaData();
  int count = rsmd.getColumnCount;
  ...

Guidelines – Various Little Things

- Avoid requesting long data unless you need it!
  - I.e. watch out for “select * from table”
- Reduce the size of results using
  - setMaxRows
  - setMaxFieldSize
- Understand your data types
Guidelines – Understand Prepared Statements

• Prepared Statements are not always as efficient as using the Statement object directly

• Use PreparedStatement if the query is to be executed more than once

• Use Statement if the query is to be executed once
Guidelines – a few more

- Use parameter markers for arguments to stored procedures … do not use literal arguments

- Use batches if available instead of PreparedStatement executions
Guidelines - RowSets

• RowSets eliminate network I/O and disk I/O until synchronization occurs

• RowSets eliminate locking and other resource usage on the backend
Guidelines – EIS Systems use Databases

• Remember that the JCA/CCI communicates to EIS Systems that ultimately use relational databases
  – All the rules discussed here apply!
Oracle – some specifics

- Use Rowid for fastest access to rows
- No Native scrolling
- Prepare does not result in a network round trip
  - PreparedStatement pstmt = conn.prepareStatement("my mama eats grits");
  - Prepares successfully!
Microsoft SQL Server - specifics

• Fastest access does NOT use server-side cursors

• Using server-side cursors allows for:
  – Multiple active open statements
  – Native scrolling
  – Real preparedStatements

• Native interface streams … therefore, you must read all rows from a query
DB2 – some specifics

- No native scrolling
- GRAPHIC data types are complex and hard to deal with
- 2 part queries only (schema.table)
Summary

• The JDBC API and the J2EE Connector Architecture can be used to build robust applications
• Design for performance and don’t rely on tuning your components later
• JDBC RowSets offer a unique architectural model but must be understood fully before use!
If You Only Remember One Thing…

Avoid Network I/O and Disk I/O.