



New trends in Java Development for IDS (JCC & pureQuery)

Satheesh Bandaram
IBM Corp.

Session Code: B18

Wednesday, April 30, 2008 • 04:40 p.m. – 05:40 p.m.

2008 IIUG Inform*i*x Conference



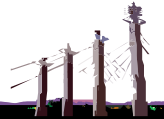
Session B18

New Trends in Java
Development for IDS

Satheesh Bandaram

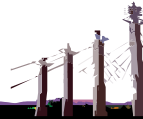
IBM Corp.

bandaram@us.ibm.com



What is JCC?

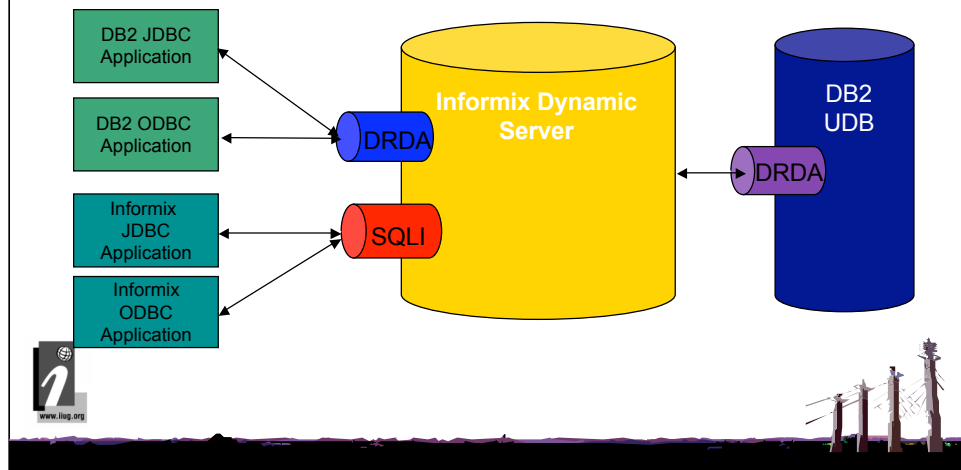
- Before IDS Cheetah release (11.10), IBM Informix JDBC driver provided Java connectivity to IDS server
 - This Java driver is IDS server specific
 - Had limited integration and testing with other IM products and tools
- New IBM Data Server Driver for JDBC and SQLJ (JCC) provides Java connectivity to all DB2 servers, IDS and Cloudscape
 - IDS server support was added in Cheetah release
- Provides JDBC or SQLJ access to data
- Very compact and doesn't need install
 - Very small footprint, about 2.5 MB
 - Provides pure-Java or Type 4 connectivity to IDS
- Provides both JDBC 3 and JDBC 4 drivers
- High Performance
 - Request chaining for Type-4, sendDataAsIs, deferPrepares





IDS server supports both drivers!

- Now IDS server can work with both DRDA and SQLI clients
- DRDA support enables all common clients to work with IDS
 - JCC, .Net, Ruby on Rails, PHP and Perl



Common Java Client for ALL IM data servers - Java Common Client [JCC]

Development is priming the [JCC] driver for IDS CHEETAH release

We have most of the base data types, transactions, cursors, ANSI support, statements, result sets, error handling, etc., completed.

A parallel development team working on VIPER 2 Line Items (many are applicable to IDS), SDO/DAS, JDBC 4.0, etc.,

Common Web Admin will be based on JCC and will support IDS via this enablement

Convergence includes QA as well with enhancements made to common test buckets

JCCDEVTESTs, IDS JDBC QA, Regression and System test buckets

All new enhancements for all databases will be worked into JCC

Informix JDBC connection URLs, environment variables, data source will be supported

Goal is to keep existing application impacts to a bare minimum. However,

Namespace will carry JCC namespace in order to distinguish between the drivers

JDBC specification implementation will ride on JCC semantics, unless there is a business need to retain multiple implementations

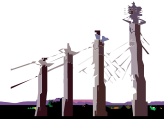
**DRDA protocol also drives some of the semantics as opposed to SQLI implementation

Legacy Informix JDBC driver in maintenance and will be deprecated over time



Value proposition of JCC/DRDA to IDS platform

- Single JDBC driver for ALL IM data servers
 - Application compatibility and migration
- All new enhancements for all databases will be worked into JCC
 - PureQuery technology needs JCC driver
 - Data Server Web Services and openJPA exploit JCC more
 - New Data Studio suite of tools will work better with IDS using this driver
 - WebSphere suite of products works very closely with JCC. Many JCC extensions created for WAS exploitation. High availability and performance features integrated into WAS suite very tightly
 - SAP and other vendors exploit JCC functionality to the fullest
- Many third party products and tools available for DRDA
- All JDBC development teams working together on same product



Common Java Client for ALL IM data servers - Java Common Client [JCC]

Development is priming the [JCC] driver for IDS CHEETAH release

We have most of the base data types, transactions, cursors, ANSI support, statements, result sets, error handling, etc., completed.

A parallel development team working on VIPER 2 Line Items (many are applicable to IDS), SDO/DAS, JDBC 4.0, etc.,

Common Web Admin will be based on JCC and will support IDS via this enablement

Convergence includes QA as well with enhancements made to common test buckets

JCCDEVTESTs, IDS JDBC QA, Regression and System test buckets

All new enhancements for all databases will be worked into JCC

Informix JDBC connection URLs, environment variables, data source will be supported

Goal is to keep existing application impacts to a bare minimum. However,

Namespace will carry JCC namespace in order to distinguish between the drivers

JDBC specification implementation will ride on JCC semantics, unless there is a business need to retain multiple implementations

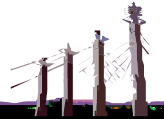
**DRDA protocol also drives some of the semantics as opposed to SQLI implementation

Legacy Informix JDBC driver in maintenance and will be deprecated over time



IDS features in JCC that are not in Informix JDBC driver

- JDBC 4.0 support
 - New major upgrade to JDBC specification that is part of JDK 1.6
 - Enhanced Large Object APIs
 - Set Client Info and other ease of use enhancements
 - Exception handling improvements, Connection management
- Full Mach 11 support only available in JCC driver
 - Work load management only exposed in JCC driver
 - Automatic Client reroute that can route client requests to alternates
- Full support for planned and unplanned failover handling
- Superior tracing and debugging mechanisms
- Full support for progressive references
- This list is likely to grow, with time



Common Java Client for ALL IM data servers - Java Common Client [JCC]

Development is priming the [JCC] driver for IDS CHEETAH release

We have most of the base data types, transactions, cursors, ANSI support, statements, result sets, error handling, etc., completed.

A parallel development team working on VIPER 2 Line Items (many are applicable to IDS), SDO/DAS, JDBC 4.0, etc.,

Common Web Admin will be based on JCC and will support IDS via this enablement

Convergence includes QA as well with enhancements made to common test buckets

JCCDEVTESTs, IDS JDBC QA, Regression and System test buckets

All new enhancements for all databases will be worked into JCC

Informix JDBC connection URLs, environment variables, data source will be supported

Goal is to keep existing application impacts to a bare minimum. However,

Namespace will carry JCC namespace in order to distinguish between the drivers

JDBC specification implementation will ride on JCC semantics, unless there is a business need to retain multiple implementations

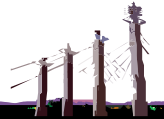
**DRDA protocol also drives some of the semantics as opposed to SQLI implementation

Legacy Informix JDBC driver in maintenance and will be deprecated over time



Migration challenges

- Some migration differences are likely
 - Name space differences need application changes
- IDS extensibility features not fully exposed in JCC
 - Some IDS supported data type support missing
- Some Datablade underlying mechanisms not yet supported
- Informix environment variables not supported
- Serial8 and BIGINT support missing. To be added in Sept 08, in FP2
- A few references to DB2 in client APIs
 - Easy to rename, but will cause migration problems to DB2 customers
- **You can help define IDS customer pain points for development team**
- This list is likely to reduce with time



Common Java Client for ALL IM data servers - Java Common Client [JCC]

Development is priming the [JCC] driver for IDS CHEETAH release

We have most of the base data types, transactions, cursors, ANSI support, statements, result sets, error handling, etc., completed.

A parallel development team working on VIPER 2 Line Items (many are applicable to IDS), SDO/DAS, JDBC 4.0, etc.,

Common Web Admin will be based on JCC and will support IDS via this enablement

Convergence includes QA as well with enhancements made to common test buckets

JCCDEVTESTs, IDS JDBC QA, Regression and System test buckets

All new enhancements for all databases will be worked into JCC

Informix JDBC connection URLs, environment variables, data source will be supported

Goal is to keep existing application impacts to a bare minimum. However,

Namespace will carry JCC namespace in order to distinguish between the drivers

JDBC specification implementation will ride on JCC semantics, unless there is a business need to retain multiple implementations

**DRDA protocol also drives some of the semantics as opposed to SQLI implementation

Legacy Informix JDBC driver in maintenance and will be deprecated over time

Switching from IBM Informix driver to JCC driver

JDBC application using IBM Informix driver

```

import java.sql.*;

public class TestInformixConnection
{
    public static void main (String[] args) throws
    Exception
    {
        Connection conn1;
        String URL = "jdbc:informix-sqli:" +
            "//myhost:1242/test:INFORMIXSERVER=testserver";

        try {
            Class.forName
                ("com.informix.jdbc.IfxDriver").newInstance();
        }
        catch (Exception e) { // Handle exception }
        try {
            conn1 = DriverManager.getConnection (URL,
            "testuser", "testpass");
        }
        catch (SQLException se) { // Handle exception }

        System.out.println("Established connection");
        conn1.close();
    }
}

```



JDBC application using Data Server driver (JCC)

```

import java.sql.*;

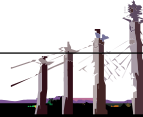
public class TestJccConnection
{
    public static void main (String[] args) throws
    Exception
    {
        Connection conn1;
        String URL = "jdbc:db2://myhost:1243/test:";

        // Load the JDBC driver ..
        try {
            Class.forName
                ("com.ibm.db2.jcc.DB2Driver").newInstance();
        }
        catch (Exception e) { // Handle Exception }

        try {
            conn1 = DriverManager.getConnection (URL,
            "testuser", "testpass");
        }
        catch (SQLException se) { // Handle Exception }

        System.out.println("Established connection");
        conn1.close();
    }
}

```



```

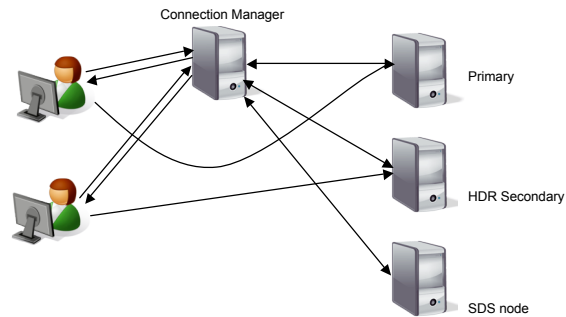
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class TestInformixConnection
{
    public static void main (String[] args) throws Exception
    {
        Connection conn1;
        String URL = "jdbc:informix-
sqli://myhost:1242/test:INFORMIXSERVER=testserver";

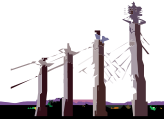
        // Load the JDBC driver ..
        try {
            Class.forName ("com.informix.jdbc.IfxDriver").newInstance ();
        }
        catch (Exception e) {
            System.out.println

```


Mach 11: Connection Manager



- Clients have the option of connecting to the servers directly or via the connection manager
- Clients connected via the connection manager will have the ability of automatic connection rerouting in case of failures
- Only JCC driver provides full support for Mach 11



Q: Does this work within ER?

A: Yes

Q: What is the failover hierarchy if the CMSM is down? Would be a manual or auto operation?

A: Alert and manual intervention.

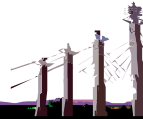
Q: How does CMSM come back up if it fails?

A: Alert and manual startup. However, there can be multiple CMSM agents running within a group configuration.

Two JAR files of the Driver

JAR file	Driver version	Level of JDBC support	Minimum level of Java
db2jcc.jar	3.50+	JDBC 3.0 and earlier	1.4
db2jcc4.jar	4.0+	JDBC 4.0 and earlier	6.0

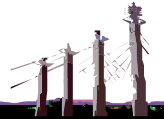
- Cheetah (11.10) IDS release was shipped with JCC 3.50. JCC 4.0 was certified later with a PID release of IDS
- Cheetah 2 (11.50) IDS release would need JCC 3.52 and JCC 4.2 to fully exploit Mach 11 and other IDS enablement features



JDBC 4: Service Provider Mechanism

- No need to load a driver using Class.forName method
 - `Class.forName("com.ibm.db2.jcc.DB2Driver");`
 - `Connection c = DriverManager.getConnection(url, user, password);`
- Auto-loading of a JDBC driver is through the service provider mechanism
- For-each loop support
 - Easier to navigate through SQLException without coding `getNextException()`
 - SQLException implements `Iterable<Throwable>`

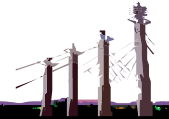
```
catch (SQLException sqle) {  
    for (Throwable e : sqle)  
        e.printStackTrace();  
}
```



JDBC 4: SQLException Improvements

New SQLException hierarchy

- 2 categories of SQLException, SQLNonTransientException, SQLTransientException
- 6 subclasses of SQLNonTransientException
 - SQLDataException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLSyntaxErrorException
- 3 subclasses of SQLTransientException
 - SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException

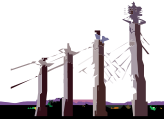


JDBC 4: Wrapper Interface and isValid

- Vendor-specific resources may be wrapped in some environment, such as in an application server
- Gain access to the vendor-specific resource in a standard way
- Most JDBC interfaces, such as Statement, ResultSet, and Connection, now extend the Wrapper interface

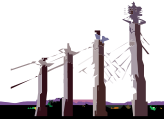
```
if (s.isWrapperFor(DB2Statement.class)) {  
    DB2Statement ds = s.unwrap(DB2Statement.class);  
    ds.setDB2ClientProgramId("...");  
}
```

- Validate whether a connection is still alive or not
 - The connection pool manager may take advantage of this information
- ```
do {
 Connection c = ...; // get a connection from pool
 if (c.isValid(timeout)) return c;
} while (hasNextConnectionInPool)
```



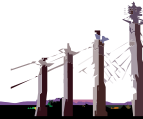
## JDBC 4: Connection Client Information

- Associate client-specific information with a connection  
c.**setClientInfo**("ApplicationName", "fooApp");  
c.**setClientInfo**("ClientUser", "fooUser");  
c.**setClientInfo**("ClientHostname", "fooHose");
- Monitoring tools can display this information for pinpointing the problematic connection  
String app = c.**getClientInfo**("ApplicationName");  
String user = c.**getClientInfo**("ClientUser");  
String host = c.**getClientInfo**("ClientHostname");
- Retrieve a list of supported properties  
DatabaseMetaData dm = c.getMetaData();  
**ResultSet** rs = dm.**getClientInfoProperties**();  
while (rs.next()) {  
    // Process  
}



## JDBC 4: Statement Events and poolable hint

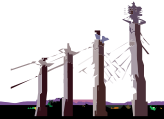
- Allow the statement pool manager to be notified of statement events
- The statement pool manager needs to implement `StatementEventListener` and register itself on `PooledConnection`  
`PooledConnection pc = ...;`  
`pc.addStatementEventListener(poolManager);`
- The listener is notified of statement closing and statement error events
- The event contains the statement, its associated connection, and the exception about to be thrown
- Give a hint to the statement pool manager about whether a statement should be pooled or not  
// in the application  
`s.setPoolable(true);`





## JDBC 4 support for IDS

- Several LOB handling improvements
  - Ability to free LOBs ahead of transaction boundary
  - Allow partial large object retrieval as a stream
  - Allow large object setting without a length
  - Enable creation of large objects through factory methods
  - JCC also added support for length-less streaming
- New datatypes, SQLXML and ROWID. Not applicable to IDS
  - SQLXML enables binary XML transport to servers
  - Provides text, SAX, DOM or StAX streaming directly
  - Adds national character data types



### Common Java Client for ALL IM data servers - Java Common Client [JCC]

Development is priming the [JCC] driver for IDS CHEETAH release

We have most of the base data types, transactions, cursors, ANSI support, statements, result sets, error handling, etc., completed.

A parallel development team working on VIPER 2 Line Items (many are applicable to IDS), SDO/DAS, JDBC 4.0, etc.,

Common Web Admin will be based on JCC and will support IDS via this enablement

Convergence includes QA as well with enhancements made to common test buckets

JCCDEVTESTs, IDS JDBC QA, Regression and System test buckets

All new enhancements for all databases will be worked into JCC

Informix JDBC connection URLs, environment variables, data source will be supported

Goal is to keep existing application impacts to a bare minimum. However,

Namespace will carry JCC namespace in order to distinguish between the drivers

JDBC specification implementation will ride on JCC semantics, unless there is a business need to retain multiple implementations

\*\*DRDA protocol also drives some of the semantics as opposed to SQLI implementation

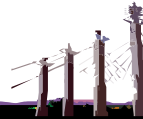
Legacy Informix JDBC driver in maintenance and will be deprecated over time



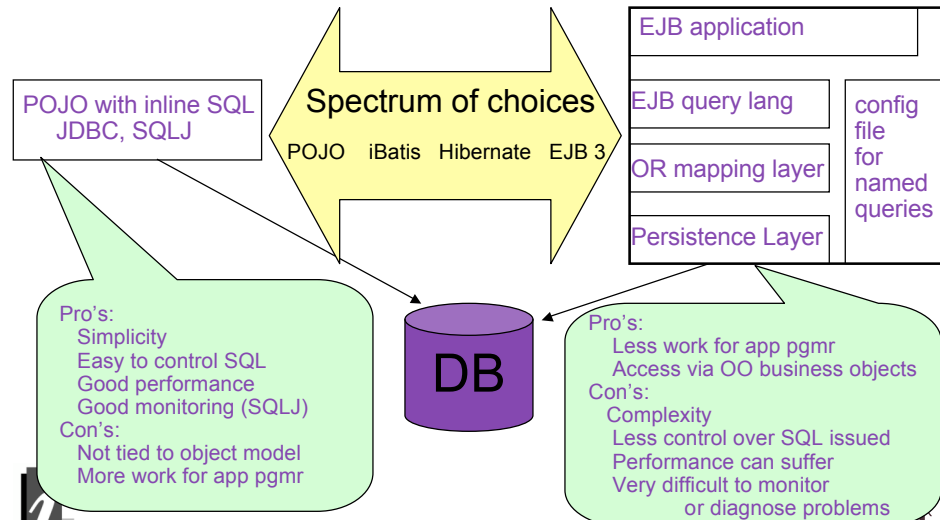


## *IBM Data Studio pureQuery*

- Reducing the cost of deploying and managing data



## Java Data Access – many forms



## What performance/diagnosis challenges?

## EJB Query Language:

```
SELECT object(e) FROM Employee e
WHERE e.dept=?1 AND e.salary>=?2
```

Query language is a subset of SQL. Doesn't have all the SQL features you want.

App query syntax is different from SQL query. How do you track problem SQL queries back to the app that issued the original query???

In most cases, queries map to JDBC. No ability to lock in access path at program deployment. No ability to search catalog to see which queries are issued by a given program.

EJB application

EJB query lang

OR mapping layer

Persistence Layer

config  
file  
for  
named  
queriesOR mapping  
Transform to SQL

## SQL issued to database:

```
SELECT * FROM PROD.EMP
WHERE DEPT=? AND SALARY>
```

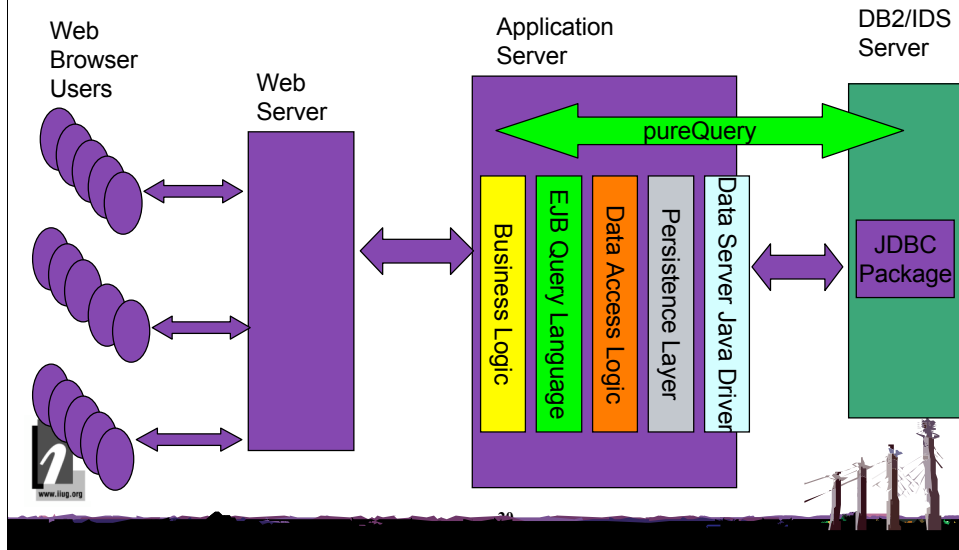
Often, app query is intercepted by persistence layer, and the resulting SQL query looks nothing like the app query.

- Resulting query might perform badly.
- Changing app query might not result in a similar change in the SQL query...

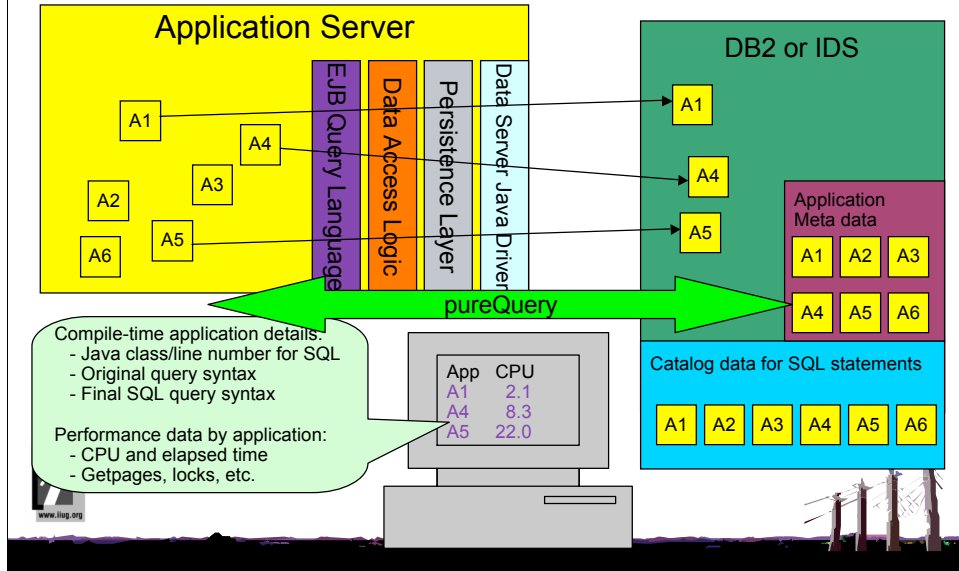


www.iiug.org

## Toughest issue for Web apps – Problem diagnosis and resolution



## pureQuery with IBM Runtime/Tooling



-Omegaman

-pureQuery is trying to unify monitoring

-Capture and save monitoring data in a database so that everyone can get to it

*pureQuery: Improved Java data access*

pure**Query**

**pureQuery is a high-performance Java data access platform focused on simplifying the tasks of developing and managing applications that access data.**

## **Optimizing Java database development and deployment**

- Integrated tools, APIs, and runtime
- Flexible deployment options with Static SQL support
- Unified API to query databases and in-memory Java objects with SQL



The benefits of using pureQuery extend throughout the development, deployment, management, and governance stages of the application life cycle.

pureQuery provides access to data in databases and in-memory Java objects via its tools, APIs, and runtime environment.

Embraces SQL as the common query language

Improves the Java data access life cycle

Development, deployment, management, and governance stages

Key component of IBM's end-to-end problem determination strategy

Before pureQuery

JDBC or SQLJ to access database

Tedious to develop, customize, maintain

Developer spends a lot of time on lower level data access instead of business logic

Technologies based on proprietary query languages

No visibility to the efficiency of the generated SQL

Problem determination difficult

Tied to single vendor

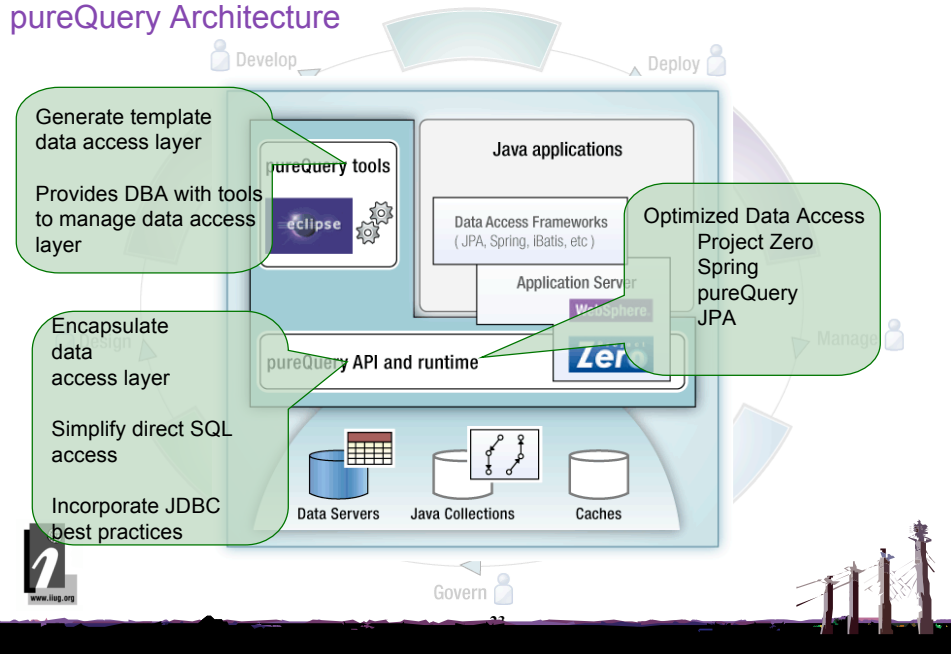
Not sophisticated enough to handle complex scenarios

Performance boost from Static SQL requires new skills

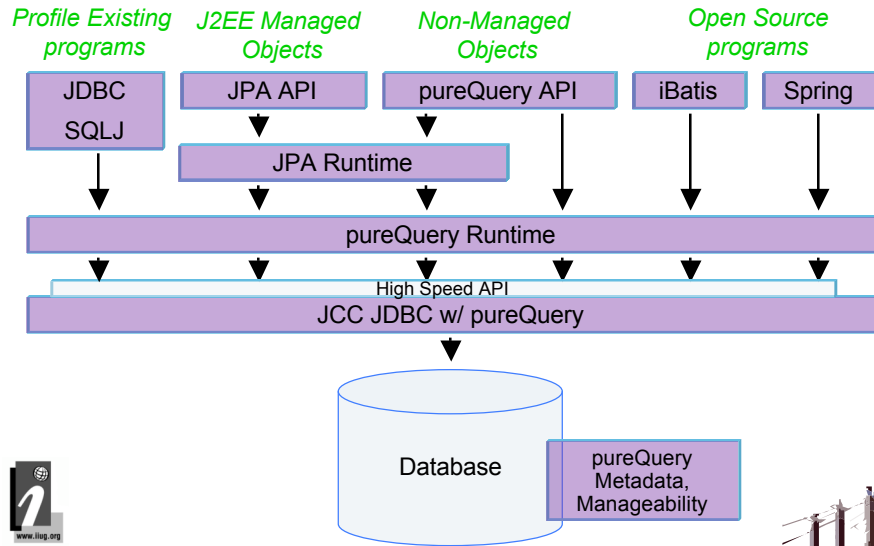
No simple integrated way to work with database and in-memory Java objects

With pureQuery you can use standard SQL to access data from databases or in-memory Java objects

## pureQuery Architecture

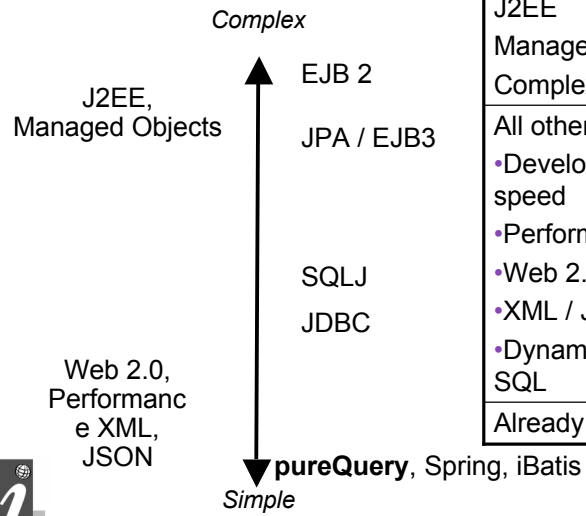


## Java On-Ramps to pureQuery





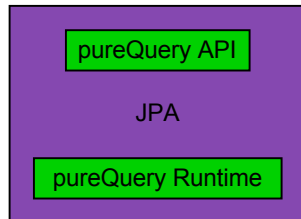
## Java Data API Space



|                                                                                                                                                                                            |                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| J2EE<br>Managed Objects<br>Complex O/R                                                                                                                                                     | JPA +<br><b>pureQuery</b> |
| All other cases <ul style="list-style-type: none"><li>• Development speed</li><li>• Performance</li><li>• Web 2.0 / AJAX</li><li>• XML / JSON</li><li>• Dynamic &amp; Static SQL</li></ul> | pureQuery                 |
| Already using SQLJ                                                                                                                                                                         | SQLJ                      |



## JPA and pureQuery together

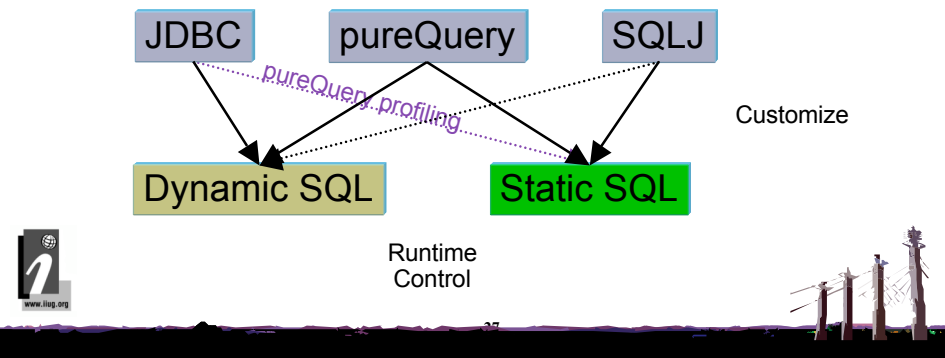


- EJBQL and runtime SQL generation based on object manipulation make the value of pureQuery even more important in the JPA setting
- IBM is enhancing our JPA implementation with both pureQuery APIs and pureQuery runtime lifecycle benefits
- JPA w/pureQuery enables problem determination, optimization, and governance connecting the EJBQL and business logic to the actual SQL and database operation
- JPA / EJB3 is a J2EE5 standard
- WebSphere is delivering JPA
- Apache openJPA is the only JPA implementation supported by more than one major vendor: BEA and IBM



## pureQuery enables wide use of Static SQL

- Static SQL
  - Highest speed
  - Greatest reliability
- JDBC is basic access, uses Dynamic SQL
- SQLJ adds Static SQL
- pureQuery supports both Static SQL and Dynamic SQL
  - Code to dynamic SQL, turn on static SQL at deployment



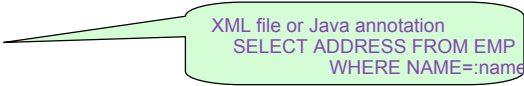
## Retrieve a single row from Database

**pureQuery:***Automatically Optimizes for 1 row*

```
addr = db.queryFirst("SELECT ADDRESS FROM EMP
WHERE NAME=?", String.class, name);
```

*-or-*

```
addr = getAddress(name);
```



XML file or Java annotation  
SELECT ADDRESS FROM EMP  
WHERE NAME=:name;

**SQLJ:**

```
#sql [con] { SELECT ADDRESS INTO :addr FROM EMP
WHERE NAME=:name };
```

**JDBC:**

```
java.sql.PreparedStatement ps = con.prepareStatement(
 "SELECT ADDRESS FROM EMP WHERE NAME=?");
ps.setString(1, name);
java.sql.ResultSet names = ps.executeQuery();
names.next();
addr = names.getString(1);
names.close();
ps.close();
```

Most SQL statements in SQLJ are a one-line Java expression. These one-line expressions contain familiar "host variable" syntax that makes it easy for the programmer to issue SQL statements that reference Java variables.

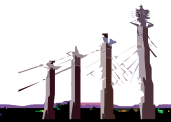
JDBC is a much lower-level programming interface. The application programmer has to use set and get methods to associate Java program variables with SQL statements. A typical SQL SELECT statement can easily take 50-100 lines of code using JDBC. The same SQL statement is a one-line expression in SQLJ.

*Developing with pureQuery: Flexible styles*■ **pureQuery support several programming styles**

- Inline style – SQL in application
  - Simplified and intelligent direct SQL access
- Method Style – Encapsulate SQL in Java interfaces
  - Annotated Method Style
    - ❖ Define SQL as Java annotations
  - Named query style – extension of Annotated Method Style
    - ❖ Define SQL in XML files. same format used by EJB 3.0 Java Persistence Architecture (JPA)

■ **Results provided**

- Java objects and collections
- JSON (Javascript Object Notation)
- XML



Support several API styles to fit well into all of the popular Java programming models/frameworks

Inline style (familiar JDBC and SQLJ approach)

Method style (similar to JDBC 4 ease of use enhancements). You define the interface and we write the code for you. Trying to generate most efficient code. Mapping is done at code generation

Named query style (similar to iBatis/JDO/Hibernate/JPA)

## Questions/Comments

