

00101 01011000  
10000 01000101  
10010 01001001

IDUG® North America


01000101 01011000 01010000 01000101 01010010  
01000100 01010101 01000111 00100001 00100000  
01001110 01000011 01000101 00100000 01001001

Experience IDUG

**Session: H13**

## Java DB2 Performance with Data Studio and pureQuery

**David Beulke**  
Dave Beulke & Associates  
a division of *Pragmatic Solutions, Inc.*

 **IDUG**  
The Worldwide DB2 User Community

**Cross Platform**

Java is becoming the most popular development language for today's systems. Since there are so many java coding techniques and java object options, java can be particularly difficult to design, complex to code, and problematic for achieving your performance goals. This presentation will take you through my experiences over the last four years tuning client java systems running on z/OS and distributed system. It will also highlight the new Data Studio and pureQuery features and its ability to code your java database access in many different fashions to match whatever application requirements are necessary.

In addition, discussions will also highlight pureQuery that can enhance performance and tuning capabilities by statically binding your java against the DB2 system. This enhances the accountability of your SQL, provides many security advantages and debugging capabilities. Through this presentation you will learn the java system architectures, coding techniques and what questions to ask to fix your java system performance issues.

Dave@DaveBeulke.com - 703 798-3283



- Member of the inaugural IBM Data Champion program
- One of 45 IBM DB2 Gold Consultant Worldwide
- Past President of International DB2 Users Group - IDUG
- Best speaker at CMG conference & former TDWI instructor

- Author of Syspedia- find, understand and integrate your data faster!

- [www.Syspedia.com](http://www.Syspedia.com)

**BLOG at**

- Columnist for DB2 Magazine

- Former editor of the IDUG Solutions Journal

[www.davebeulke.com](http://www.davebeulke.com)

- | Consulting                        | Educational Seminars                 |
|-----------------------------------|--------------------------------------|
| • CPU Demand Reduction Guaranteed | • DB2 Version 9 Transition           |
| • DB2 Performance Review          | • DB2 Version 9 Java Performance     |
| • Database Design Review          | • Data Warehouse Performance Seminar |
| • Security Audit & Assessments    | • SOA Architecture Review            |
| • Migration Assistance            | • DB2 Business Intelligence          |

- Extensive experience in performance of large databases and DW systems
  - Working with DB2 on z/OS since V1.2
  - Working with DB2 on LUW since OS/2 Extended Edition

- Co-Author of certification tests

- IBM DB2 V8 & V7 DBA certification & Business Intelligence certification test

1 01011000 01000100 01010101 01000111 01000001 01000000 01000101 01011000 01010000 01000101 01012110 01001001 01001001 01000011 01000101 01000000 01001001 01000100 01000101 01011000 01010000 01000101  
IDUG 2009 North America Copyright 2009 - PSI Dave@DaveBeulke.com

David Beulke is an internationally recognized DB2 consultant, author and lecturer. He is known for his extensive expertise in database performance, data warehouses and internet applications.

He is an augural member of the IBM Data Champion program, member of IBM DB2 Gold Consultant program, Past President of the International DB2 Users Group (IDUG), a columnist for DB2 Magazine, co-author of the IBM V8 and V7 z/OS DB2 Administration Certification exam, co-author of the Business Intelligence Certification exam, former instructor for The Data Warehouse Institute (TDWI), and former editor of the IDUG Solutions Journal.

His consulting and educational expertise helps clients with developing new systems, tuning performance problems or reducing costs on their mainframe, UNIX and Windows systems. His clients save millions of dollars in CPU charges, avoided unnecessary hardware upgrades and improved application development through his customer focused solutions, performance tuning expertise and expert business designs.

# Objectives

- Discuss the issues and performance problems discovered in various client systems over the last three years
- Analyze the Data Studio features and java pureQuery programming methods and options – Inline statements, (OR) Object Relational Mapping, Java Methods or Named queries.
- Understand the advantages of pureQuery, static access paths, flexible coding options, transform dynamic to static access and improved performance
- Realize the improved debugging and performance advantages of using static pureQuery techniques
- Analyze how to implement and leverage pureQuery, with the improvements to Visual Explain to capture all the java SQL information just like your COBOL applications

Data Studio and pureQuery offers the ability to code your java database access in many different fashions to match whatever application requirements are necessary. Using inLine statements, Java methods, object relational mapping, or named queries pureQuery offers the java programmer the flexibility to match their coding style with the application requirements.

In addition, pureQuery enhances the accountability of the java SQL code since it can now be statically bound against the DB2 system and database objects. Statically binding the java SQL to the system provides many advantages which are highlight in the presentation.

# Agenda

- Java Application Problems
- Data Studio - pureQuery – Background
- Examples Inline programming method
- Programming Java Methods database values
- Coding for performance
- pureQuery Java programming advantages
  - pureQuery Capture Mode
  - pureQuery Cross reference

These are the topics covered in this presentation. Additional information can also be found on the following web sites.

## Understanding pureQuery, Part 1:

### **pureQuery: IBM's new paradigm for writing Java database applications**

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0708ahadian/>

## **IBM Data Studio Information Center**

<http://publib.boulder.ibm.com/infocenter/dstudio/v1r1m0/index.jsp>

## **pureQuery rapid application development**

[http://www.ibm.com/developerworks/edu/dm-dw-dm-0711surange-i.html?S\\_TACT=105AGX11&S\\_CMP=LP](http://www.ibm.com/developerworks/edu/dm-dw-dm-0711surange-i.html?S_TACT=105AGX11&S_CMP=LP)



# Connections - Basics

- Use a server connection pool
  - One per ??
    - Web page
    - Application
    - Web server
- Always time out threads after a certain period
  - Only define the number needed
    - Know your minimum and maximum expected
- Recycle and reuse all the threads connected

The connections to the database are the majority of the issues encountered with java applications. The problems usually fall into three categories. Too many connections, not released connections or duplicate connections.

All of these situations provide difficulty in debugging, maintaining transaction integrity and the ability to tune the application transaction environment.

# Connections

- How many does the application really need?
  - Database
    - DB2 LUW, DB2 z/OS & Oracle in one transaction
  - Queues
    - Inbound and outbound
  - Web and App Server connection threads
- Parallelism and connection state
  - Mind the state of all of these connections
    - How long each is active
    - How long the transaction UOW is!

Copyright 2009 - PSI Dave@DaveBeulke.com  
IDUG 2009 North America 6

Distributed java applications can use a lot of resources against the database, servers and overall operating environment. The connections to the various resources can be a major issue for debugging and logging against the environment.

Sometimes the number of message queues can become an issue and minimizing the number of connections, the time that they are open and the amount of messages per transaction are critical for achieving performance.

Also managing the message queue activity is important to make sure the queue is robust enough to handle the message workload. Remember it is a queue and not a high performance database.

# Application Scope

- UOW & Transaction Scope
- Persistence cache control
- Hibernate and persistence layer issues
  - Lazy, Evict, etc.....
  - Regular SQL versus Hibernate SQL
  - Optimistic-lock
- Blocking and Commit Considerations
- Logging Considerations
  - How many logs are your TX writing to?

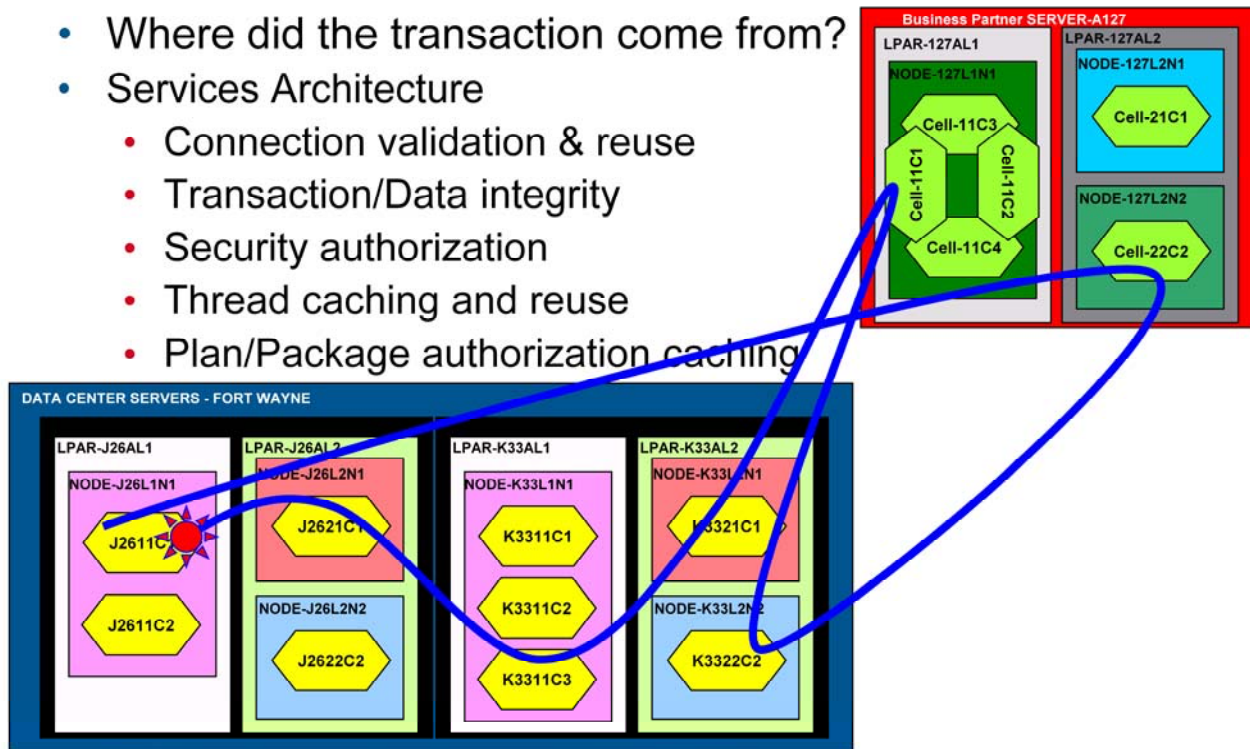
Copyright 2009 - PSI Dave@DaveBeulke.com  
IDUG 2009 North America

Transaction scope sometimes gets lost within the design team of the distributed java applications. Since the java development world of small reusable services are spread across the development staff sometimes the java web service does not concern itself with the UOW and the transaction scope. This can be a major problem as the number of services called or involved in a transaction escalate.

Sometimes the database access has totally been handled through a persistence layer such as Hibernate. These persistence layers are very convenient but are just another layer of debugging and complexity. In most cases they are just another problem by rewriting the SQL, flushing their persistence poorly or forcing another layer for the transactions to go through.

# Common Errors

- Where did the transaction come from?
- Services Architecture
  - Connection validation & reuse
  - Transaction/Data integrity
  - Security authorization
  - Thread caching and reuse
  - Plan/Package authorization caching



SOA architectures are the current fashion for application development. Managing these new environment requires extra attention to the how and when the application or the many different services interact with the database. The load balanced server configurations can have many different hardware capacity, memory allocations, software levels/releases and each cell can cause a different performance problem.

AJAX and dynamic application can create many connections and fire many SQL statements against the database environment. Try to use the new pureQuery Java application connection because it is static and pre-authorizes the database objects, plan/package authorization and security profile.

In addition to all the capacity and performance issues, the services need to maintain their transaction integrity. Services need to make sure they notify other services when any error or SQL return code causes a rollback.



# Common Errors

- Exception Error handling may not be appropriate
  - All errors are not exceptions
    - -811 and +100 SQLCode
- GET DIAGNOSTICS statement
  - Information about the last SQL statement
    - How many rows effected/errors
    - Example
      - Retrieve information that is similar to what is returned by the SQLCA plus Row Count

```
EXEC SQL GET DIAGNOSTICS CONDITION 1
:dasqlcode = DB2_RETURNED_SQLCODE,
:datokencnt = DB2_TOKEN_COUNT,
:datoken1 = DB2_ORDINAL_TOKEN_1,
:datoken2 = DB2_ORDINAL_TOKEN_2,
:datoken3 = DB2_ORDINAL_TOKEN_3,
:datoken4 = DB2_ORDINAL_TOKEN_4,
:datoken5 = DB2_ORDINAL_TOKEN_5,
:dasqlerrdlb = DB2_MESSAGE_ID,
:damsqtext = MESSAGE_TEXT,
:dasqlerrp = DB2_MODULE_DETECTING_ERROR,
:rcount = ROW_COUNT,
:dasqlstate = RETURNED_SQLSTATE;
```

Only basic java exception handling is generated within the Data Studio pureQuery generated modules. Good news is that error handling is at least present. Bad news is that it may not be what the java application needs to control or complete the processing logic.

For example, SQL and java exceptions can happen for a variety of reasons the generic exception may not be appropriate for situations such as more than one row qualifies in the result set (-811), or row not found (+100). Application developers need to generate test cases for these and other expected and unexpected application processing errors.

# Common Errors

- Commit Scope is a problem within java applications
  - The connection auto-commit mode to false:
    - `(Data) data) .setAutoCommit(false);`
  - Number of connections within the code executed
    - Every module is creating a connection instance
      - Understand which modules get a connection to the database  
minimize the times a connection is created
- Very important for transaction integrity, rollback and service scope analysis
  - `db.rollback();`

The commit scope of the java application development needs to be fully communicated and understood by the java development team. Since every application is using a services architecture these days it is very important to understand the integrity commit scope of all the applications. pureQuery automatically generates setAutoCommit statements within its module code and needs to be managed properly by the programmer to maintain transaction service integrity.

In addition to the setAutoCommit a database connection is usually generated with a pureQuery module. The connection information can be great to relieve the java application developer from this tedious task. The common error is that a large number of connections can be required by the application. Another potential issue is that the commit scope may not be valid because when a connection is started a Unit-of-Work (UOW) is started. Starting a new UOW may not be desired because this new service or transaction may need to be in the same overall transaction or service commit scope of other dependent activities. Make sure to verify the commit scopes of the java application and the number of connections used throughout the application.

# Coding for performance

- Static Bind provides the performance boost
- Removes dynamic overhead for every statement executed
  - User Authorization
    - Can also removed user from the table access
    - Now only need package execute authority
  - Object verification
    - Existence check of table(s), columns etc
  - Lock in access path
    - No dynamic changes

The biggest impact Data Studio and pureQuery are static bind capabilities for all the java applications. By providing static binds the accountability, monitoring and pinpointing of the SQL within the java source code is possible. All this information will provide great debugging and performance improvements for everyone's java applications.

The Visual Explain facilities are also integrated into Data Studio and are great for explaining the static bind SQL application information within the tool. This provides the java application developer with all the regular debugging and access table and index access path dependency information.

# pureQuery - Background

- Started out same time as Viper - DB2 9 for LUW
  - Data Studio - Beta Summer 2007
  - Data Studio and pureQuery are two different things
- Data Studio many function and features
  - Developer and DBA Administration product
- pureQuery is new static java processing
  - All platforms – z/OS, LUW, i5/OS and IDS
    - SQL and XML Result sets
    - Many types of java API access

The release of Data Studio Beta in the summer of 2007 was the beginning of pureQuery and the major enhancements to java within DB2 9 for zOS and LUW systems. These enhancements allow the java developer the flexibility to tailor the java SQL to the java EE or servlet or EJB implementation.

This pureQuery flexibility along with the ability to return Web 2.0 artifacts such as JSON, XML along with its integration with Eclipse and xQuery provide all types of usage and style possibilities.

# pureQuery -The next Gen in Java DB2 connectivity

- Provides the best of all options
  - Static database module
    - Improves security and all aspects of performance
- Developer flexibility for all types of coding requirements
  - Handles inline(JDBC/SQLJ)
  - Method calls(java/JDBC4)
  - named query(Hibernate/JDO/iBatis/JPA)
    - Simplifies developers and administrators' lives
  - queryResult = getCust(custid);
- Provides all the module information of static SQL
  - Java module SQL text stored within DB2 catalog
  - Connection/debugging back to the java class module
  - Full performance statistics tooling

The initial advantages of pureQuery were its ability to provide database access in many different styles that are being offered by many different java frameworks, development products and java architectures. These ideas from the Spring Framework, openJPA, Service Data Objects, Java Data Objects, Hibernate, iBatis, and the Java Persistence API required distinct SQL coding styles to access the database. Each camp of these java community frameworks and development products provided great specialized access architectures for particular types of processing access but were lacking for other types of application situations.

pureQuery's flexibility to provide all these different coding styles and the ability to mix and match them within a single java class eliminates the controversy of what framework or style is best. The java coding style that works best for the application is best and eliminates the industry discussion noise and moves everyone to making the application work efficiently.

pureQuery also moves the java development community toward accountability for their application SQL by statically binding the SQL into the DB2 system and database. The years of java programs being lumped together within large JAR class files and embedded within Websphere WAR application files are gone and individual SQL statements within particular java class files can be monitored, analyzed and pinpointed within every java application. pureQuery statically bound SQL programs improves accountability by also locking in a database access path and the security profile of the use and access types.



## pureQuery – Static Web v??.? connectivity

- Handles mapping between connectivity and object
  - Provides connection to data stores
    - XML and memoryCached objects
- Single API with the flexibility for all types of objects
  - Same API for XML, Joins/SQL or in-memory objects
  - Developers will put everything into memory objects
  - Debugging with virtual objects values!
- API improves on previous set/get routines
  - Reduces the data – program variable mismatches

pureQuery also provides access through SQL and xQuery to XML and memory cached objects within a java frameworks persistence layers. This enables pureQuery developers to leverage the latest developments with in-memory objects and in-memory databases.

This pureQuery interface provides developers the ability to put any type of simple SQL or multi-table join complex SQL or xQuery object access into a java class and map it appropriately to the underlying data store. This capability provides great debugging, monitoring and analysis capabilities for all types of virtual and static data objects. This also helps every SQL or xQuery pureQuery process to match data types, domains and ranges to further eliminate data mismatches and error situations.

# Features List

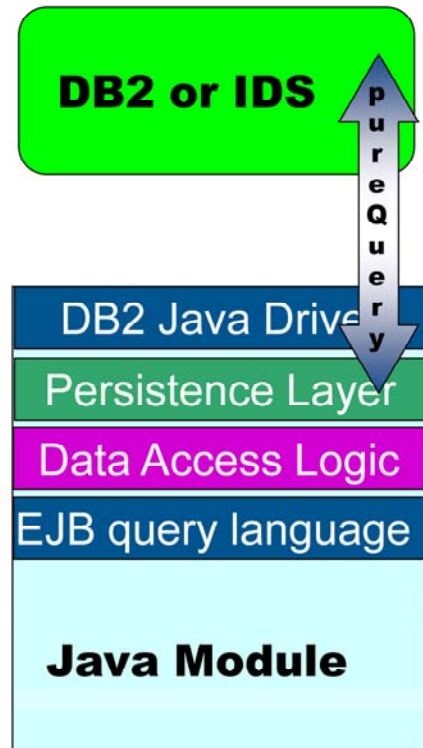
- Performance improvements with static java modules
  - Static - ANSI Standard SQL92
    - Eliminates object, user & access path authorization
    - Permanent Access Path
    - Security – privileges are package based
      - Users don't need table access authorization
  - Scalability
    - Short path length
    - Can be used as JDBC or bound and be static
- Monitoring
  - Package level accounting records produced
  - Package traceability from distributed thread
    - Traceability for problems, locks and SQL considerations
  - Package information stored in the DB2 Catalog
- Integrated Debugging environment for Java programs, stored procedures and java stored procedures

pureQuery provides the ability to statically Bind a java class SQL module to the database. This is similar to what many shops have done for many years with COBOL and many other application development languages. Statically Binding a java module against the database system is just like a COBOL Bind against the database system and it provides all the same benefits also.

The benefits of static java modules are that it eliminates object, user and access path authorizations. These tasks were very cumbersome for JDBC applications verifying, authorizing and developing an access path for every SQL statement within every transaction execution. The DB2 system mechanisms of Dynamic Statement Cache and EDM pool caching of dynamic statements did a wonderful job of minimizing the JDBC dynamic statement costs but statically bound java application completely eliminate these costs. Since many large corporations execute 100's of millions of dynamic JDBC java application transactions daily, statically bound java application will reduce CPU demands substantially for many corporations.

# Debugging Information

- Monitor each Java SQL executed
  - Application Name
  - Java class name
  - Java method name
  - Java object name
  - Source code line number
  - Source code context
  - Transaction name
  - Last compile timestamp
- Just like a normal static CICS or batch package
  - Cost estimate
  - Visual Explain
    - Elapsed Time and CPU time
  - SQL text generated



Statically bound pureQuery java application modules provide a great deal of information for monitoring and performance analysis within the run-time environment. The ability to track the java application modules via, application name, java class name, and java method name provides a lot of flexibility for monitoring and pinpointing java application code issues. These monitoring and tracking options go even further providing the Source code line number and compile timestamp of the java code for absolute debugging and performance analysis.

Data Studio also integrates Visual Explain capabilities directly into the menus so that the developers can quickly reference the access path information and fully understand the database tables and indexes referenced to retrieve the underlying data.

# Features List

- Improves Java programming API options

- Method style
  - Persistence Layer
  - Similar to JDBC 4



- Free - any DBMS
- Works with JDBC

- Object Relational Mapping Layer
  - Inline style – JDBC and SQLJ
  - Verification of database tables
  - Hibernate style



- Corporate Red Hat
- java & .net framework

- Named query style
  - JPA or OpenJPA standard acceptable
  - iBatis or JDO



- Apache
- DAO Removed

- EJB Query Language

- Query database, Cache, Collections or XML
  - Use SQL to query all type of objects

The iBatis framework with its DAO mapping provided named query style java database data access objects. These mapping were the layer that sometimes separated out the SQL developer from the java programmer. This separation also led to many hours debugging performance problems and inconsistencies from the DAO layer to the java code that referenced the data.

Hibernate with its Object Relational Mapping (ORM) framework was great as long as the ORM framework matched the object view of the data. When the objects were merged or separated the SQL needed to be changed also. Hibernate's ORM framework was also difficult to map multiple table joins because as the result set changed the ORM mapping needed to be changed.

Both Hibernate and iBatis frameworks successfully abstracted the data persistence layer to inline and ORM objects that are then referenced in the java application code. This architecture abstraction presents a problem because it does not tie back to the source SQL database access and each architecture has its preferred inLine or ORM coding style. The coding style hinders developers from using the best java coding style for the application module of service design. pureQuery allows all styles of programming and ties back to the SQL through the static Bind process for monitoring and accountability.



# Old way - manual coding

Five parts of three different statement types

- Statement types
  - Dynamic, DAO or SQLJ

```
<%@ page import='java.util.ArrayList' %>
<%@ page import='java.util.*' %>
<%@ page import='java.sql.*' %>
<%@ page import='javax.naming.*' %>
<%@ page import='javax.sql.*' %>
```

```
<%
// =====Pool Connection=
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");
if(envCtx == null )
throw new Exception("Boom - No Environment Context");
// the following matches the resource name defined in dbv20.xml
DataSource ds = (DataSource) envCtx.lookup("jdbc/db001");
// =====Pool Connection=
// =====ifs=
if (ds != null) {
Connection con = ds.getConnection();

if(con != null) {
// =====ifs=
```

```
query = "SELECT AMOD_MMBR_LIB, AMOD_MMBR, "
+ "AMOD_ENVR_ID, AMOD_APPL_ID, "
+ "AMOD_MMBR_TS, AMOD_EXTR_TS "
+ "from APPL_GUIDES "
+ "WHERE AMOD_ENVR_ID LIKE '" + envrname2 + "%' "
+ "AND AMOD_APPL_ID LIKE '" + applname2 + "%' "
+ "AND AMOD_MMBR LIKE '" + mmbrname2 + "%' "
+ orderbyOption2;
```

```
Statement stmt = con.createStatement();
log(getClass() + ": did the stmt ");

ResultSet rs = stmt.executeQuery(query);
log(getClass() + ": executed the query: " + query);

ResultSetMetaData md = rs.getMetaData();
log(getClass() + ": got the metadata array
```

```
while (rs.next()) {
for (int i=1; i<=colcount; i++) {
String applhldr = (rs.getString(i));
colnbr = colnbr +1
}
}
```

```
// important to cleanup
rs.close();
stmt.close();
con.close();
```

DaveBeulke@cs.com

IDUG 2009 North America

Data Studio generates all the different parts of the SQL code. In this slide we show the old classic manual coding style of JDBC code. This slide shows how loose the program code was for a typical JDBC program.

Note the various statements to check the JDBC Context and the various connection statements. All of these JDBC SQL parts are tightly generated from Data Studio helping to eliminate coding syntax errors and the tedious coding standard java interfaces.



# Data Studio pureQuery streamlines coding



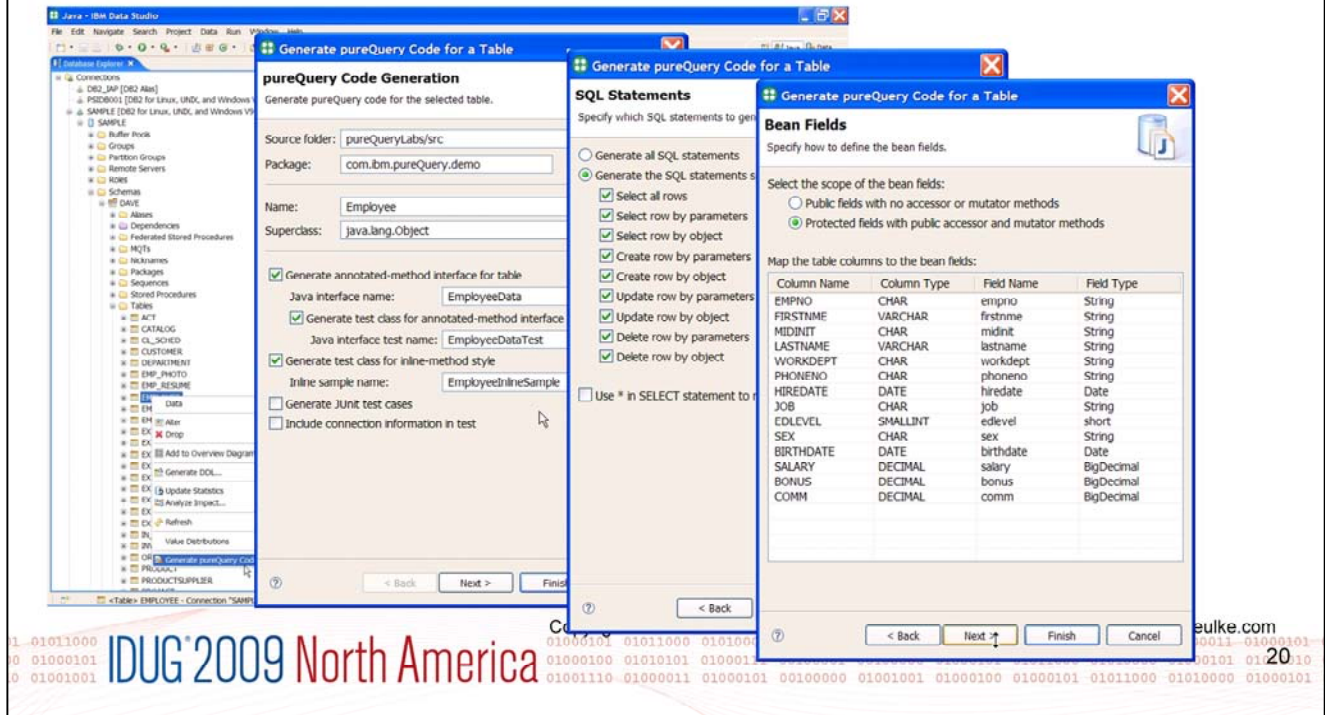
- Generates any style of coding desired
  - Methods, inline, ORM, memory and XML
- Generates java and SQL statement code
  - DBMS Connection, commit, SQL statement, exception handling, table-data type matching
- Generates sample output print statements to show the resultSets of the generated SQL statements
  - Debugging is easy with Data Studio execution capabilities

Data Studio with its ability to generate all the different styles of calling the database are a great productivity asset to the java developer. This capability provides the java developer with the ability to quickly generate error free SQL statements to retrieve, insert, update and delete database rows.

In addition to generating of the SQL statements, Data Studio also generates DBMS connection code. This handles one of the more confusing items for the database developer. Also the generated code also handles commit situations, and the database exception handling of errors within the java module.

# Any programming style

- Data Studio generates it easily



The inLine programming style is common to the iBatis framework. pureQuery provides a simple way to generate the code for this style of programming.

Looking at the capabilities within Data Studio, within several clicks the SQL code is generated, the exception handling along with a module that can be tested and validated.

# inLine Style

```
Java - CustomerInLineSample.java - IBM Data Studio
File Edit Source Refactor Navigate Search Project Data Run Window Help

CustomerInLineSample.java
package com.ibm.pureQuery.demo;

/**
 * A class to access PDQ_SC.CUSTOMER inline.
 */
import java.util.Iterator;

public class CustomerInLineSample {

    public static Data db = null;

    /**
     * @param args
     */
    public static void main (String[] args) {
        try {
            if (args.length < 1 ) {
                SampleUtil.println("All required arguments were not provided.");
                return;
            }
            db = SampleUtil.getData ("jdbc:db2://localhost:50000/SVC", "dave", args[0]);
            db.setAutoCommit(false);
            Iterator<CustomerInLine> getCustomerInLines = db.queryIterator ("select CID, NAME, COUNTRY, STREET, CITY, PROVINCE, ZIP, PHONE from PDQ_SC.CUSTOMER");

            CustomerInLine bean = null;
            if (getCustomerInLines.hasNext()) {
                bean = getCustomerInLines.next();
                ((ResultIterator<CustomerInLine>)getCustomerInLines).close();
            } else {
                SampleUtil.println("Result set is empty.");
                db.rollback();
                return;
            }

            CustomerInLine getCustomerInLine = db.queryFirst ("select CID, NAME, COUNTRY, STREET, CITY, PROVINCE, ZIP, PHONE, INFO from PDQ_SC.CUSTOMER where CID = ?", bean);
            SampleUtil.printClass(getCustomerInLine);

            db.update("update PDQ_SC.CUSTOMER set NAME = :name, COUNTRY = :country, STREET = :street, CITY = :city, PROVINCE = :province where CID = :cid", bean);
            getCustomerInLines = db.queryIterator ("select * from PDQ_SC.CUSTOMER", CustomerInLine.class);
            SampleUtil.println("Results for update (bean)");
            SampleUtil.printAll(getCustomerInLines);

            db.update("delete from PDQ_SC.CUSTOMER where CID = :cid", bean);
            getCustomerInLines = db.queryIterator ("select * from PDQ_SC.CUSTOMER", CustomerInLine.class);
            SampleUtil.println("Results for - delete (?)");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Copyright 2009 - PSI

Dave@DaveBeulke.com

1 01011000  
0 01000101  
0 01001001  
IDUG\*2009 North America

01000101 01011000 01010000 01000101 01010010 01001001 01000101 01001110 01000011 01000101  
01000100 01010101 01000111 00100001 00100000 01000101 01011000 01010000 01000101 01  
01001110 01000011 01000101 00100000 01001001 01000100 01000101 01011000 01010000 01000101

In this inLine style example, it shows the connection information, the SQL and the resultSet iterator to retrieve all the database table SQL data. This nice routine quickly retrieves the data and presents it.

Note the generated AutoCommit(false) within the code. Special analysis and handling of the commit scope of a module needs to be done to make sure the module or service is handling the work properly.

Also note the rollback within the module that could be referenced if there are no rows retrieved through the SQL statement. These generated statements are fine the way the code is generated and works now but may need to be changed if the module or transaction logic changes.

# Java Methods database values



- Generates all SQL statement types

```
package com.ibm.pureQuery.demo;

/* An interface that has methods for PQD_SC.CUSTOMER. */
import java.util.Iterator;

public interface CustomerData {
    // Select all PQD_SC.CUSTOMER
    @SuppressWarnings("unchecked")
    Iterator<Customer> getCustomers();

    // Select PQD_SC.CUSTOMER by parameters
    @SuppressWarnings("unchecked")
    Iterator<Customer> getCustomers(int cid);

    // Select PQD_SC.CUSTOMER by Customer object
    @SuppressWarnings("unchecked")
    Iterator<Customer> getCustomers(Customer c);

    // Create PQD_SC.CUSTOMER by parameters
    @SuppressWarnings("unchecked")
    int createCustomer(String name, String country, String street, String city, String province, String zip, String phone, String info);

    // Create PQD_SC.CUSTOMER by Customer Object
    @SuppressWarnings("unchecked")
    int createCustomer(Customer c);

    // Update PQD_SC.CUSTOMER by parameters
    @SuppressWarnings("unchecked")
    int updateCustomer(String name, String country, String street, String city, String province, String zip, String phone, String info);

    // Update PQD_SC.CUSTOMER by Customer object
    @SuppressWarnings("unchecked")
    int updateCustomer(Customer c);

    // Delete PQD_SC.CUSTOMER by parameters
    @SuppressWarnings("unchecked")
    int deleteCustomer(int cid);

    // Delete PQD_SC.CUSTOMER by Customer object
    @SuppressWarnings("unchecked")
    int deleteCustomer(Customer c);
}
```

Within one module all the necessary java SQL methods can be generated for an application manipulating table data. All of the statements provide the developer with many options for retrieving and manipulating the table data.

In addition to these statements the developer can add any additional SQL statements to the module. When adding these new SQL statements Data Studio will automatically validate the SQL, the column names and the overall syntax of the SQL statement.

# Coding for performance

- Example Bind statement

```
• java com.ibm.pdq.tools.StaticBinder -url
  jdbc:db2://DSNPROD:50000/CUSTOE -user dave -password pwd
  <bind-options> com.ibm.pureQuery.demo.CustomerDataImpl
```

DB2 for Z/OS options:

```
.-ACTION (REPLACE) -+-----+
(1) |               '-REPLVER (version-id) -' |
>>-----+-----+
'-ACTION (ADD)-----'

.-DBPROTOCOL (DRDA)----. .-DEGREE (1)---. .-EXPLAIN (NO)---.
>>-----+-----+-----+-----+
'-DBPROTOCOL (PRIVATE) -' '-DEGREE (ANY) -' '-EXPLAIN (YES) -'

.-IMMEDWRITE (NO)---. .-ISOLATION (RR)---. .-NOREOPT (VARS)---.
>>-----+-----+-----+-----+
'+IMMEDWRITE (FH1) -+ '+ISOLATION (RS) -+ '-REOPT (VARS)---'
'-IMMEDWRITE (YES) -' '+ISOLATION (CS) -+
'-ISOLATION (UR) -'

>>-----+-----+-----+-----+
'-OPTHINT (hint-ID) -' '-OWNER (authorization-ID) -'

>>-----+-----+-----+-----+
|               |               |
|               |               |
|               |               |
'-PATH (-+---schema-name-+-+)-'
'-USER-----'

>>-----+-----+-----+-----+
.-RELEASE (COMMIT)-----.
'-QUALIFIER (qualifier-name) -' '-RELEASE (DEALLOCATE) -'

>>-----+-----+-----+-----+
.-SQLERROR (NOPACKAGE)---. .-VALIDATE (RUN)---.
'-SQLERROR (CONTINUE)---' '-VALIDATE (BIND) -'

1.-0
0 0
0 0:
```

DB2 Database for Linux, UNIX, and Windows options:

```
(1) .-BLOCKING UNAMBIG-. .-DEGREE 1---.
>>-----+-----+-----+-----+
+BLOCKING ALL-----+ +DEC 15-+ '-DEGREE ANY-'
'-BLOCKING NO-----' '-DEC 31-'

.-EXPLAIN NO---. .-EXPLSNAP NO---. .-FEDERATED NO---.
>>-----+-----+-----+-----+
'-EXPLAIN YES-' +EXPLSNAP ALL-+ '-FEDERATED YES-'
'-EXPLSNAP YES-'

>>-----+-----+-----+-----+
.-INSERT DEF-. .-ISOLATION CS-.
'-FUNCPATH schema-name-' '-INSERT BUF-' +ISOLATION RR-+
+ISOLATION RS-+
'-ISOLATION UR-'

>>-----+-----+-----+-----+
'-OWNER authorization-ID-' '-QUALIFIER qualifier-name-'

>>-----+-----+-----+-----+
.-SQLERROR NOPACKAGE-.
'-QUERYOPT optimization-level-' '-SQLERROR CONTINUE--'

.-SQLWARN YES-. .-STATICREADONLY NO---. .-VALIDATE RUN---.
>>-----+-----+-----+-----+
'-SQLWARN NO--' '-STATICREADONLY YES-' '-VALIDATE BIND--'

Dave@DaveBeuik.com
```

```
1
1010000 01000101 01010010 01001001 01000101 01001110 01000011 01000101
1000111 00100001 00100000 01000101 01011000 01010000 01000101 01
1000101 00100000 01001001 01000100 01000101 01011000 01010000 01000101
```

The pureQuery process has all the Bind options of z/OS and LUW platforms.



# Coding for performance



- Static Bind puts everything into the DB2 catalog
  - Has all the standard objects within the DB2 LUW and z/OS catalog
    - All dependencies and objects are in the catalog
      - Query the objects through catalog queries
      - For example:

```
SELECT PKGSHEMA, PKGNAME, REMARKS, PKG_CREATE_TIME, PKGVERSION,  
ISOLATION, BLOCKING, BOUNDBY, TOTAL SECT, ISOLATION, EXPLICIT_BIND_TIME,  
LAST_BIND_TIME  
FROM SYSCAT.PACKAGES  
WHERE PKGSHEMA = 'NULLID' AND PKGNAME LIKE 'Custom%'
```

- Get all information on the DB2 java application just like all the COBOL DB2 applications

The Static Bind process for java application puts all the standard elements within the DB2 Catalog. This helps everyone by allowing catalog queries to highlight the various dependent table and index objects. This also allows all the standard tools and third part products to get object information through their normal package dependencies interfaces.

You can also query all the bind parameters for a particular process to understand all of its unique settings.

## pureQuery programming advantages

- Generates and validates SQL
  - All SQL parts - connect, SQL and exceptions
- Provides tooling for java application static Bind
  - Security enhanced
    - User Authorization removed from the table
      - Now only need package execute authority
  - Object verification at Bind time
    - Table and column existence checking
  - Access path generated and locked during Bind
    - Visual Explain integrated into Data Studio

Data Studio and pureQuery provide many new facilities to help the java application developer be more productive and write better SQL. The first facility of generating the SQL from the database, columns directly helps eliminate table and column errors along with column data type mismatches. Next the ability to generate ORM, method and all types of java SQL modules allows the developer to generate the best coding style for the application.

Next the Static Bind capabilities bring out all the classic advantages of static processes over dynamic processes within a database system. Pre-compiling and binding the java SQL application code verifies security profiles, object existence and definition and the most important process of generating a database access path that can be locked in for the processing. This access path generation can be very time and resource intensive and doing it once instead of for every execution of the process can save huge amounts of CPU for many enterprises.

# pureQuery programming advantages

- Single pureQuery API can work with any java
  - Improved developer productivity by generating data access modules through Data Studio
    - XML, in-memory, simple or complex SQL
- Uses standard JDBC - so portable across databases
  - Produces true data 'objects' for the OO designs
    - Goes beyond 'get' and 'set' routines
- Can use all other types of java modules
  - Generic, XML, JSON or custom coded

Data Studio with its ability to generate all types of data access objects provides a huge productivity improvement for java application developers. Generating standard SQL, complex SQL XML or in-memory data objects is straight forward.

Also the Data Studio Database Explorer and integrated testing provide great single click testing and database discovery capabilities. These capabilities provide new ways for the java application developer to test and see the result of their SQL or XML.

Data Studio and pureQuery also provide a way to integrate any type of java application code. This can be used to integrate or create all types of java data objects for any OO design point. This is especially important for application developer or efforts that require XML JSON or customer driven interfaces.

# SQL Capture Feature

- Existing dynamic JDBC application to Static SQL
  - Improve security with Static SQL
  - Improved performance
  - Improve debugging & maintenance
- Run existing application and capture SQL
  - No changes necessary to the application
    - Add pureQuery libraries to the external jars
    - Execute application
    - Bind capture file
      - Run static and new statements can be
        - Run dynamically or
        - Rejected and not executed

The process for enabling pureQuery within your project is very straight forward.

- 1) Add pureQuery libraries
- 2) Set Capture On within the settings
- 3) Execute all aspects of the application
- 4) Bind the “captured SQL file”
- 5) Execute the application as a static Java application
- 6) Or reject any lines not captured into your static application environment

## Cross Reference SQL with source module

- Impact analysis for java SQL modules
  - Improves debugging analysis
    - SQL traceability back to the source module
- Improve problem SQL faster
  - Trace back to source module and improve SQL
- Understand module and data dependencies
  - Through DS code outline display
  - Through SQL DB2 Catalog dependencies
    - Debugging and performance testing is much easier

Within Data studio with pureQuery you can cross reference the SQL statements to the source module. Within Outline mode of the module you can quickly see the SQL reference table and the various columns used by the module.



# FAQ – You should also know



- Does pureQuery work with DB2 z/OS Version 8?
  - Yes, the pureQuery benefits and modules can be bound and used with version 8
- Does pureQuery Capture work with mixed C# and java modules?
  - Yes, Capture works from the JDBC connection so as long as the SQL goes through the JDBC connection it will be captured and can be bound to the environment.
- What if my app completely generates the SQL statement number of WHERE clause predicates?
  - The SQL can be captured and bound and then run statically. Any additional SQL will be run dynamically like it was originally.
- If a SQL statement is not Captured and the program is setup to be run Static does the application crash or execute the SQL that wasn't captured dynamically?
  - The allowDynamic setting controls when the application SQL will be captured. When it is set to true, the default it will run and capture the application SQL. When it is set to false SQL that is not captured and bound will result in an error.

What other questions do you have?

# pureQuery programming advantages

- Static Bind removes java dynamic overhead
  - Reduces the Dynamic Statement Cache
  - Reduces EDM Pool overhead
  - Allows better memory tuning capabilities
- Faster overall execution by .001 per transaction
  - 20 million java dynamic transactions per day
  - 333 minutes of CPU saved per day!
    - Chargeback \$10-\$38 per minute = \$12,654.00/day
    - 250 business days > \$3 million per year
    - Reduced CPU demand > 5.5 hours of CPU per day
  - Your mileage may vary

The most important advantage of Data Studio and pureQuery are its capabilities to do a static bind for java SQL applications. In addition to all the static bind advantages highlighted on the previous slide, having a static java application environment helps the system reduce memory allocations. Since the workload is static it no longer requires a large system Dynamic Statement Cache, a large EDM Pool or a large number of server connections.

Static SQL and static processes within DB2 system reduce the system resources required to execute the SQL processing. For example it can reduce the overall CPU demand and result in significant charge back savings within an enterprise.

Session: H13  
Session Title: **Java DB2 Performance  
with Data Studio and pureQuery**

**David Beulke**

David Beulke Associates

A division of Pragmatic Solutions, Inc

**Dave@DaveBeulke.com**

**703 798-3283**