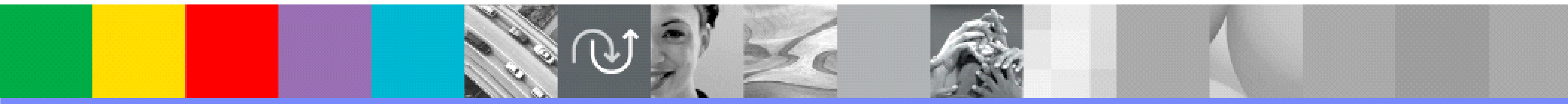
IBM Software Group

An IBM Proof of Technology

IBM Data Studio pureQuery For DBAs and Application Developers on z/OS (v2.2)

Presentations



PoT.IM.09.1.077.00

IBM Data Studio pureQuery
For DBAs and Application Developers
On z/OS (v2.2)

DETAILED AGENDA

TOPICS Part I

01 – Data Studio Environment and Overview – 35 minutes

Presentation - 8 slides – 15 minutes

This topic covers the concepts of IBM Data Studio and introduces Eclipse platform's perspective and views concepts. How IBM Data Studio helps you to manage a complete life cycle of data is also explained. An introduction to the aspects of Data Studio is also given from database administrators (DBAs) and application developer's point of view.

Lab 01 Exercises – 20 minutes

- Explore Data Studio
- Introduction to settings, perspectives & views
- Using the Database Explorer view
- Show how to use Data Studio to debug a stored procedure

02 – pureQuery Environment and Overview – 35 minutes

Presentation - 18 slides – 20 minutes

This topic covers the concepts of why pureQuery is useful for Java™ developers and DBAs. It gives the architecture and explains pureQuery and how it can be used in different scenarios.

Lab 01 Exercises – 15 minutes

- Create a pureQuery project

03 – pureQuery Tools – 55 minutes

Presentation – 18 slides – 25 minutes

This topic covers the main features and capabilities of pureQuery tools and how it can increase the productivity of Java developers and cut short on development time. A different aspect of application development is also explained in this topic.

Lab 02 Exercises – 30 minutes

- Show how to generate pureQuery code automatically for tables
- Show pureQuery capabilities of context sensitive help, pureQuery outline views.
- Show how to generate pureQuery code for SQL procedures.
- Show how to generate pureQuery code for SQL scripts

04 – pureQuery API – 55 minutes

Presentation – 14 slides – 15 minutes

This topic covers pureQuery APIs and explains different aspects of API used for query and update and batching capabilities.

Lab 03 Exercises – 40 minutes

- Show an example method style API with hands on experience
- Show an example inline style API with hands on experience for different capabilities
- Show an example of stored procedure returning a result set and how pureQuery code can be generated.

TOPICS Part II

05 – pureQuery Runtime – 50 minutes

Presentation – 19 slides – 25 minutes

This topic explains concepts of dynamic and static SQL. It introduces DB2 packages and how to lock an access path. The application using package security model is also discussed. It explains how to bind a Java application to a database for static SQL support. It also covers the steps that a DBA will take to turn dynamic SQL to static SQL at the run time.

Lab 04 Exercises – 25 minutes

- Steps to setup runtime
- Bind all interfaces using default values
- Execute a Java program

07 – pureQuery Explain – 45 minutes

Presentation – 9 slides – 25 minutes

This covers the steps that an application developer takes to create an explain plan for the queries during development time in order to write the best queries. After enabling pureQuery runtime, this shows the steps that a DBA can take to create explain plans for the all the queries that were turned to static from dynamic.

Lab 05 Exercises – 20 minutes

- Dynamic query SQL explain plan for the application developers
- Static query SQL explain plan for the DBAs

07 – Optimize existing JDBC Applications using pureQuery – 55 minutes

Presentation – 22 slides – 20 minutes

This covers the steps that an application developer or DBA takes to capture, configure and bind SQL statements from applications that are not using pureQuery model of development. For example, hibernate, JPA or any other persistence framework that an application is using can use pureQuery client optimization features to turn dynamic SQLs to static and thus lock an access path for the runtime.

Lab 05 Exercises – 35 minutes

- Capture, configure and bind SQL statements from the applications for which source code is available.
- Capture, configure and bind SQL statements from the applications for which source code is not available.

08 – pureQuery Advanced Concepts – 50 minutes

Presentation –16 slides – 20 minutes

This shows how application developers can use pluggable extensions to monitor the performance of database calls. This topic also demonstrates how to write result set handlers to customize the output data.

Lab 06 Exercises – 30 minutes

- Exercise to demo JPA XML files for externalizing SQLs, result set handlers and pluggable extensions.

Document & PoT Revision History

Version Number	Revision Description	By Whom	When
2.2	New – IBM Data Studio pureQuery for DBAs and Application Developers on z/OS	Vikram Khatri vikram.khatri@us.ibm.com Burt Vialpando burt.vialpando@us.ibm.com	04/20/2009

IBM Data Studio pureQuery for DBAs & Application Developers on z/OS (v2.2)

SCHEDULE OF TOPICS COVERED

PoT Topic	Session Length (Minutes)	Running Total (Minutes)	Start Time	End Time
PoT Greetings, Intro and logistics	0:05	0:05	9:00 AM	9:05 AM
Data Studio Introduction - Presentation	0:15	0:20	9:05 AM	9:20 AM
Data Studio Introduction – Lab	0:20	0:40	9:20 AM	9:40 AM
pureQuery Introduction - Presentation	0:20	1:00	9:40 AM	10:00 AM
pureQuery Introduction – Lab	0:15	1:15	10:00 AM	10:15 AM
[>>>>>> Break <<<<<<<<<]	0:10	1:25	10:15 AM	10:25 AM
pureQuery Tools - Presentation	0:25	1:50	10:25 AM	10:50 AM
pureQuery Tools - Lab	0:30	2:20	10:50 AM	11:20 AM
pureQuery API - Presentation	0:15	2:35	11:20 AM	11:35 AM
[>>>>>> Lunch <<<<<<<<<]	1:00	3:35	11:35 AM	12:35 PM
pureQuery API – Lab	0:40	4:15	12:35 PM	1:15 PM
pureQuery Runtime – Presentation	0:25	4:40	1:15 PM	1:40 PM
pureQuery Runtime – Lab	0:25	5:05	1:40 PM	2:05 PM
pureQuery Explain – Presentation	0:25	5:30	2:05 PM	2:30 PM
[>>>>>> Break <<<<<<<<<]	0:15	5:45	2:30 PM	2:45 PM
pureQuery Explain – Lab	0:20	6:05	2:45 PM	3:05 PM
pureQuery for JDBC Applications - Presentation	0:20	6:25	3:05 PM	3:25 PM
pureQuery for JDBC Applications - Lab	0:35	7:00	3:25 PM	4:00 PM
pureQuery Advanced – Presentation	0:20	7:20	4:00 PM	4:20 PM
pureQuery Advanced – Labs	0:30	7:50	4:20 PM	4:50 PM
Wrap-up	0:10	8:00	4:50 PM	5:00 PM

IBM Software Group

IBM Data Studio pureQuery
For DBAs and Application Developers on z/OS

Part 1

An IBM Proof of Technology

ON DEMAND BUSINESS™

Version 2.2 May 4, 2009

Vikram S Khatri and Burt Vialpando

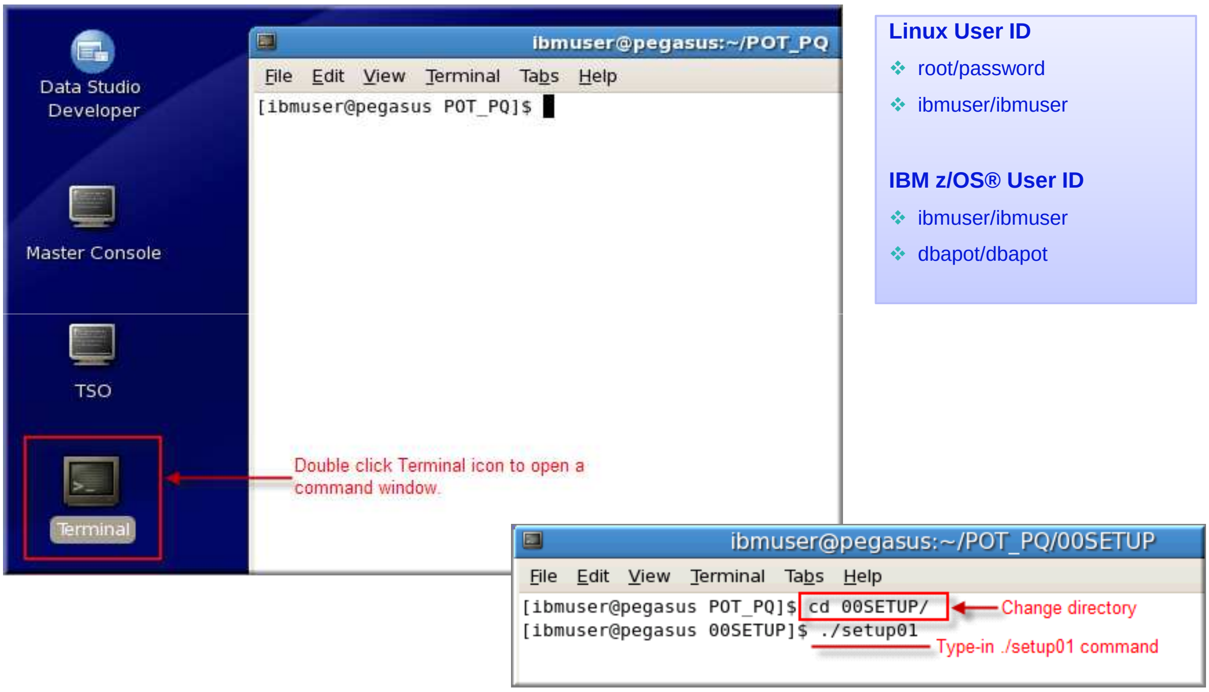
PoT Overview
Agenda – Core Topics – Part 1

Topics part 1	**Data Studio Environment Overview**	**- Lab 01**
	pureQuery Environment Overview	**- Lab 02**
	pureQuery Tools	**- Lab 03**
	pureQuery API	**- Lab 04**
Topics part 2	**pureQuery Runtime**	**- Lab 05**
	pureQuery Explain	**- Lab 06**
	Optimize existing JDBC™ Applications	**- Lab 07**
	pureQuery Advanced Concepts	**- Lab 08**

PoT Overview
Assumptions

- **For Core Topics**
  - Reasonable technical knowledge of relational database is assumed
    - We will not teach relational database principles here
  - Reasonable understanding of Java™ Programming language is assumed
    - We will not teach concepts of Java programming language here
  - The goal is to demonstrate IBM® DB2® pureQuery regardless of your current database administrator (DBA) skill set in DB2
- **For Advanced Topics**
  - Reasonably deep programming principles in Java is assumed
- **The labs will be done in a Linux® environment**
- **For brevity:**
  - “Application Development” or “Application Developer” may be referenced as “AD”

Start z/OS Virtual Machine
IPL (Initial Program Load) the machine (It takes 5-6 minutes)


Data Studio Developer
Master Console
TSO
Terminal
ibmuser@pegasus:~/POT_PQ
File Edit View Terminal Tabs Help
ibmuser@pegasus:~/POT_PQ/00SETUP
File Edit View Terminal Tabs Help

```
[ibmuser@pegasus POT_PQ]$
```

Double click Terminal icon to open a command window.

```
[ibmuser@pegasus POT_PQ]$ cd 00SETUP/
[ibmuser@pegasus 00SETUP]$ ./setup01
```

Change directory

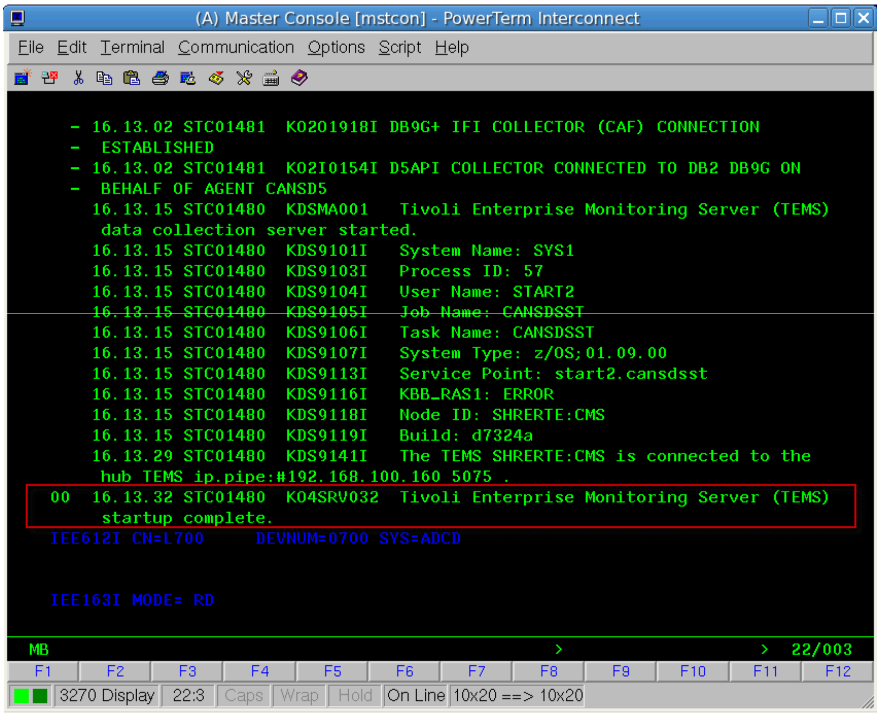
Type-in ./setup01 command

Linux User ID
- root/password
- ibmuser/ibmuser

IBM z/OS® User ID
- ibmuser/ibmuser
- dbapot/dbapot

Your Personal z/OS

After IPL


(A) Master Console [mstcon] - PowerTerm Interconnect
File Edit Terminal Communication Options Script Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
3270 Display 22:3 Caps Wrap Hold On Line 10x20 ==> 10x20

```
    - 16.13.02 STC01481  K0201918I DB9G+ IFI COLLECTOR (CAF) CONNECTION
    -  ESTABLISHED
    - 16.13.02 STC01481  K02I0154I D5API COLLECTOR CONNECTED TO DB2 DB9G ON
    -  BEHALF OF AGENT CANSD5
      16.13.15 STC01480  KDSMA001   Tivoli Enterprise Monitoring Server (TEMS)
       data collection server started.
      16.13.15 STC01480  KDS9101I   System Name: SYS1
      16.13.15 STC01480  KDS9103I   Process ID: 57
      16.13.15 STC01480  KDS9104I   User Name: START2
      16.13.15 STC01480  KDS9105I   Job Name: CANSDSST
      16.13.15 STC01480  KDS9106I   Task Name: CANSDSST
      16.13.15 STC01480  KDS9107I   System Type: z/OS;01.09.00
      16.13.15 STC01480  KDS9113I   Service Point: start2.cansdsst
      16.13.15 STC01480  KDS9116I   KBB_RAS1: ERROR
      16.13.15 STC01480  KDS9118I   Node ID: SHRERTE:CMS
      16.13.15 STC01480  KDS9119I   Build: d7324a
      16.13.29 STC01480  KDS9141I   The TEMS SHRERTE:CMS is connected to the
       hub TEMS ip.pipe:#192.168.100.160 5075 .
  00  16.13.32 STC01480  K04SRV032  Tivoli Enterprise Monitoring Server (TEMS)
       startup complete.
  IEE612I CN=L700      DEVNUM=0700 SYS=ADCD


  IEE163I MODE= RD

 MB                                                 >             >  22/003
```

Please remember

- Enter key – Right CTRL
- Attention key - Pause

IBM Software Group

Data Studio Environment Overview

Lab 01 Introduction to Data Studio

An IBM Proof of Technology

8 slides

Client Business Drivers and Challenges

Respond, Align and Compliance

Respond to opportunities and changes

- Innovation and speed is required for survival, but constant state of change creates development and deployment complexity
- Service Oriented Architecture (SOA)enables business agility, but need to service-enable data assets
- Hard to meet quality and delivery objectives with escalating regulations

Align with business priorities

- Many roles to keep in synch across business, architecture, development, administration
- Lack of common vocabulary with well understood semantics
- Lack of tools that provide integration or compatibility across roles

Compliance to privacy and security

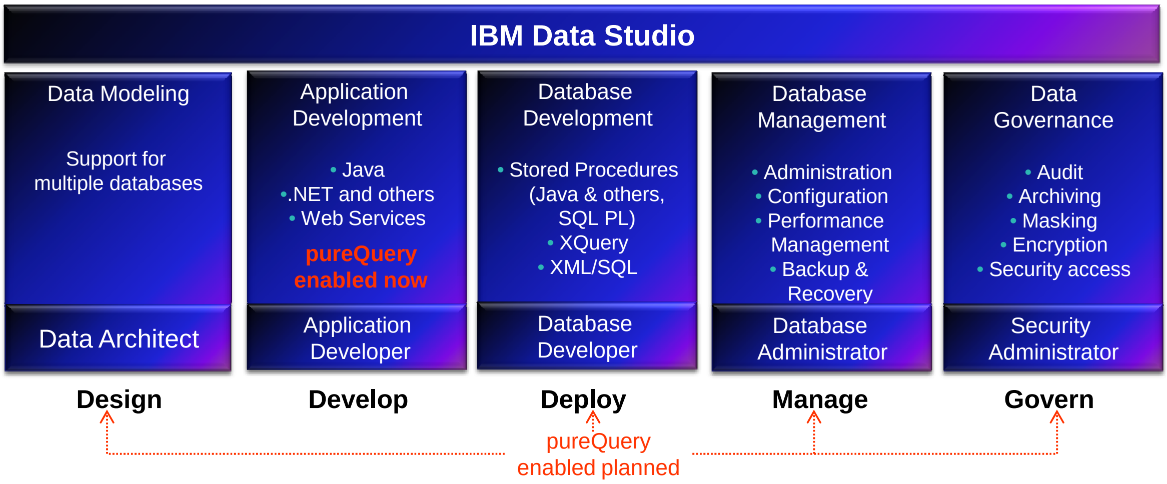
- Achieving service level agreements is very complex and people intensive
- Policy specification, implementation, and audit are largely manual processes
- Security and retention policies require greater granularity than typically available

---

IBM Data Studio Overview

Data Life Cycle Management

- **IBM Data Studio is a family of integrated database development and management tools for:**
  - DB2 Linux, UNIX® and Windows® (LUW) version 8.x onwards
  - DB2 for IBM System i® Version 5R2, 5R3 and 5R4
  - DB2 for IBM System z® Version 7, 8, 9 and Derby Version 10.x onwards
  - IBM Informix® version 9.x onwards (on all supported platforms)



IBM Data Studio

Data Modeling
Support for multiple databases
Data Architect
Design

Application Development
- Java
- .NET and others
- Web Services

pureQuery enabled now
Application Developer
Develop

Database Development
- Stored Procedures (Java & others, SQL PL)
- XQuery
- XML/SQL

Database Developer
Deploy

Database Management
- Administration
- Configuration
- Performance Management
- Backup & Recovery

Database Administrator
Manage

Data Governance
- Audit
- Archiving
- Masking
- Encryption
- Security access

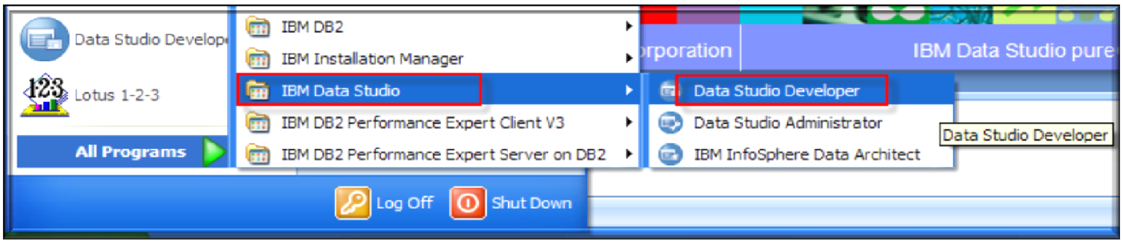
Security Administrator
Govern

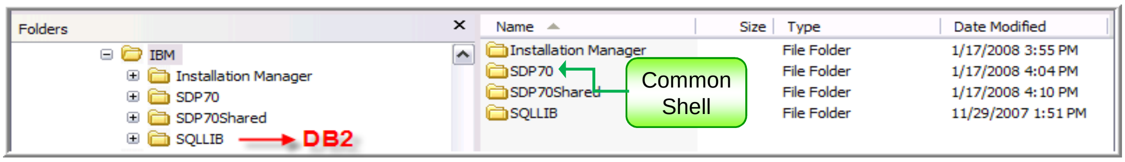
pureQuery enabled planned

Common Shell

DSD, DSA, IDA

- **Add tools to existing installed products and common shell sharing**
- **Single workspace – allows you to be more productive**
- **No import, export or extending the capability required**





IBM Data Studio Administrator (DSA)	"C:\Program Files\IBM\SDP70\eclipse.exe" -product com.ibm.datastudio.administrator.product.ide
IBM Data Studio Developer (DSD)	"C:\Program Files\IBM\SDP70\eclipse.exe" -product com.ibm.datastudio.developer.product.ide
IBM Infosphere Data Architect (IDA)	"C:\Program Files\IBM\SDP70\eclipse.exe" -product com.ibm.rational.data.architect.product.ide

---

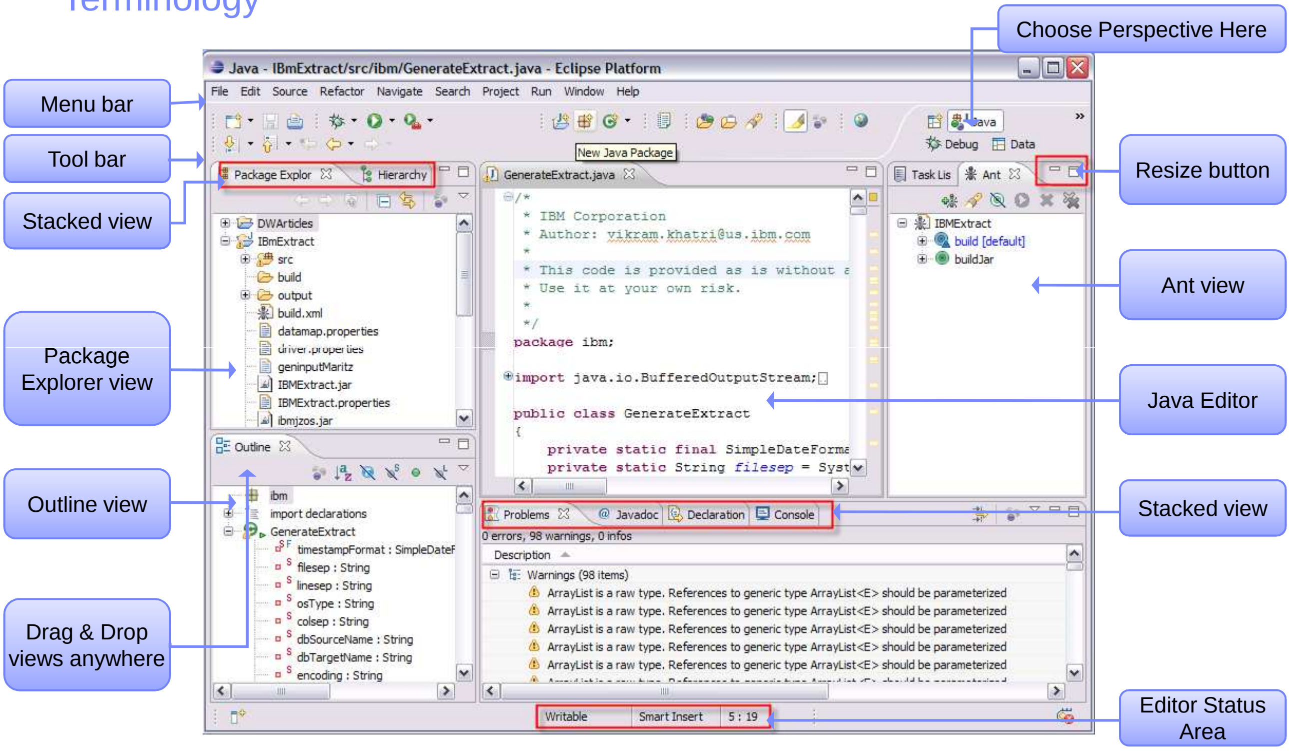
Eclipse Integrated Development Environment

Why use vs. the others (e.g. JDeveloper, NetBeans, JBuilder, etc.)

- **What is Eclipse?**
  - Open extensible Integrated Development Environment (IDE)
  - Developed by IBM, donated to the open source community and managed by eclipse.org
  - Eclipse perspectives are visual containers for a set of views and editors they are different for each context.
  - Tools operate on files on user's workspace
    - Workspace holds 1 or more top level projects
    - Projects map to directories in the file system
    - Tree of folders and files and termed as resources
    - Plug-in access via workspace and resource APIs
- **Benefits of using Eclipse**
  - Is a fully Integrated Development Environment supporting
    - Java, C/C++/C#, PERL, PHP, HTML, JSP™, EJB™ …
  - Integration platform for tools and open platform for application development tools
  - Advanced drag and drop features
  - Easily add new tools to existing installed products
    - common shell sharing
  - Eclipse platform runtime:
    - Is a micro kernel that discovers plug-ins at runtime
    - builds global plug-in registry
  - All IBM Rational® products are Eclipse based

Eclipse

Terminology



```
/*
 * IBM Corporation
 * Author: vikram.khatri@us.ibm.com
 *
 * This code is provided as is without a
 * Use it at your own risk.
 *
 */
package ibm;

import java.io.BufferedOutputStream;

public class GenerateExtract
{
    private static final SimpleDateForma
    private static String filesep = Syst
```

```
0 errors, 98 warnings, 0 infos
Warnings (98 items)
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
```

Choose Perspective Here

Menu bar

Tool bar

Stacked view

Package Explorer view

Outline view

Drag & Drop views anywhere

Resize button

Ant view

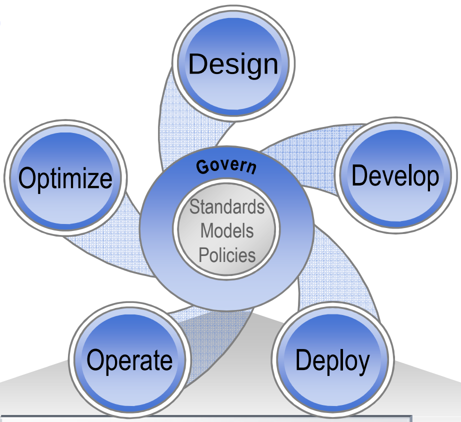
Java Editor

Stacked view

Editor Status Area

IBM Data Studio

Value Preposition



Improve Productivity
Java, XML, SOA development
Task automation
Team alignment

Improve Quality of Service
Performance optimization
Increased security
Increased quality

Improve Efficiency
CPU Capacity
Error reduction
Problem isolation

IBM InfoSphere™ Data Architect
A collaborative data design tool to understand information assets and their relationships, model data, and enforce enterprise standards for data quality and consistency

High performance Unload
High speed data extract for data migration and recovery

IBM Data Studio Developer
An integrated development environment for rapidly creating and testing database and pureQuery applications and services.

IBM Data Studio pureQuery Runtime
A high-performance Java data access platform to improve performance, security, and manageability of Java connections to databases.

IBM Data Studio Administrator
An administration environment to reduce application outages by automating and simplifying complex DB2 structural changes

Performance Expert
Improve availability with early problem detection
Isolate problems faster and with fewer resources
Free up DBA time to focus on value-creation activities

IBM Data Studio Developer
For Application Developer and DBA

IBM Data Studio Developer is an integrated database development environment that speeds application design, development, and deployment while increasing data access performance and manageability.

Enhance developer productivity

- Drag and drop creation of Web services for any SQL, XQuery, or stored procedure
- Provide a seamless SQL/Java experience including SQL assistance, validation, execution, and analysis
- Generate a data access layer using Java objects, JSON, or, XML
- Enhance problem isolation correlating problem SQL with issuing code, even when using frameworks that generate the SQL

Provide expert like performance for Java data access

- Facilitate use of JDBC and SQL data access best practices
- Improve DB2 performance, predictability, and manageability by enabling transparent activation of static SQL (i.e. no change to the application)

IBM Software Group

pureQuery Environment Overview

Lab 02 Create pureQuery Project

An IBM Proof of Technology

ON DEMAND BUSINESS™

18 slides

Problem Statement

Java Developers

High Cost of Application Development

- No support for SQL development within Java IDE
- No way to generate and customize optimized SQL code and test cases
- No way to leverage the power of SQL with in-memory Java data

Unpredictable application behavior

- Unpredictable performance due to runaway, ad-hoc and unexpected queries
- No link between the application and its executed SQL, which makes troubleshooting time consuming
- Difficulty meeting Service Level Agreements (SLA) and availability targets

---

IBM Data Studio Developer

5 key ways to enhance productivity

Focus

pureQuery for Java Data Access

- Develop / Deploy / Debug Dynamic & Static

Manage Database Objects

- Create/Alter/Drop

Queries

- Develop / Run SQL, SQL/XML, XQuery

Stored Procedures

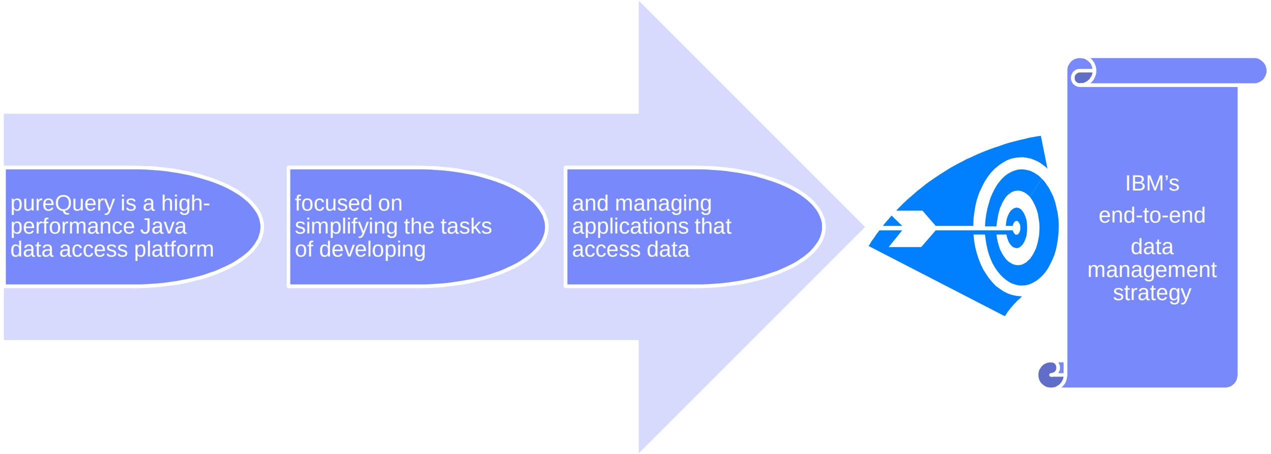
- Develop / Deploy / Debug SQL/Java

Data Studio Web Services

- Develop / Deploy

What is pureQuery?

Definition



pureQuery is a high-performance Java data access platform

focused on simplifying the tasks of developing

and managing applications that access data

IBM's end-to-end data management strategy

Why pureQuery?

Issues with conventional model of data access

JDBC and SQLJ

Pros	Cons
✓ Simplicity	✘ Not tied to object model
✓ Easy SQL Control	✘ More work for the application programmer
✓ Good performance	✘ Disconnect between developing env and db
✓ Good monitoring (SQLJ)	✘ Time consuming

Application that use alternative query languages (Hibernate, iBatis, EJB…)

Pros	Cons
✓ Less work for programmers	✘ Complexity
✓ Access via Object Oriented business objects	✘ Less control over SQL
	✘ Performance can suffer
	✘ Difficult to diagnose or monitor problems

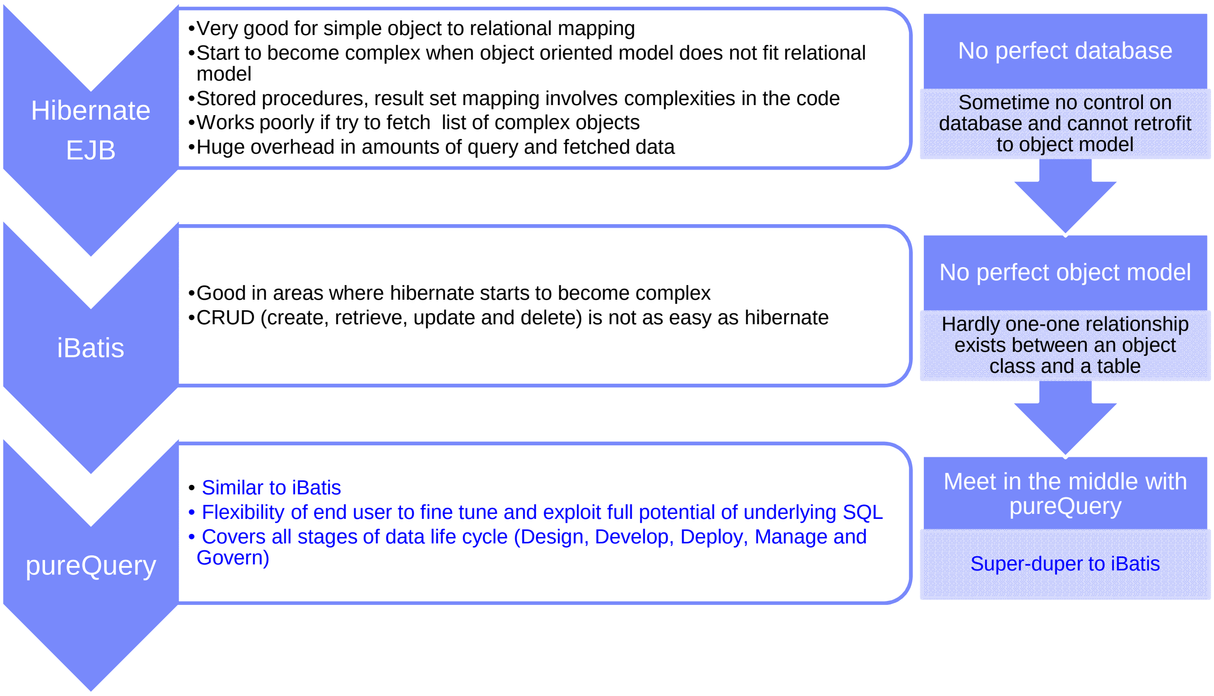
Why pureQuery?

Drawbacks of existing solutions

- ☹ **Java objects do not match with relational entities**
- ☹ **Hibernate, Toplink etc., force you to learn another object query language like HQL …**
- ☹ **These hide actual native SQL from the developers and DBAs since these queries transform themselves into native query language at runtime.**
- ☹ **Less visibility to the efficiency of generated SQL, thus problem determination is more difficult**
- ☹ **Proprietary object query language is not sophisticated enough to handle more complex relational queries**
- ☹ **IDEs have very limited or no integration with the Java editor to help construct standard SQL inside Java applications**
- ☹ **Hibernate, Toplink etc., provide primitive integration between Java and their object query language during application development**

Why pureQuery?

Challenges of popular data access framework



Hibernate
EJB

- Very good for simple object to relational mapping
- Start to become complex when object oriented model does not fit relational model
- Stored procedures, result set mapping involves complexities in the code
- Works poorly if try to fetch list of complex objects
- Huge overhead in amounts of query and fetched data

No perfect database

Sometime no control on database and cannot retrofit to object model

iBatis

- Good in areas where hibernate starts to become complex
- CRUD (create, retrieve, update and delete) is not as easy as hibernate

No perfect object model

Hardly one-one relationship exists between an object class and a table

pureQuery

- Similar to iBatis
- Flexibility of end user to fine tune and exploit full potential of underlying SQL
- Covers all stages of data life cycle (Design, Develop, Deploy, Manage and Govern)

Meet in the middle with pureQuery

Super-duper to iBatis

Why pureQuery?

Managed vs. Non-managed objects

Managed objects / object Centric

- Database is a black-box repository for the storage and retrieval of objects
- Developers want to know as little as possible about database

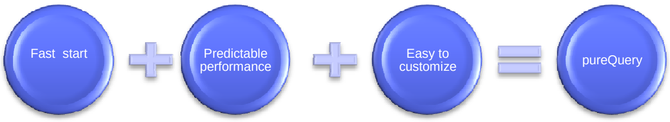
Experience shows 3 times more I/Os with hibernate data access and performance problems

Non-managed objects / data centric

- The database is an important server for the storage and retrieval of information.
- Architects and developers carefully design and optimize the database and data access code
- Data is accessed by more than one application

Why pureQuery?

Solutions and capabilities



- ☺ **Use native SQL with pureQuery to have simple database access layers**
- ☺ **Use standard SQL not just for databases, but also for in-memory collections**
- ☺ **Auto generate the data access layer with suggestions to CRUD (create, retrieve, update and delete) statements**
- ☺ **Auto generate test applications and JUnits for the generated code**
- ☺ **SQL integration inside Java editors is a hallmark of pureQuery tooling**
- ☺ **Flexibility in application development from SQL or Java beans**
- ☺ **Work with JPA XML format to keep all SQLs in one location outside Java**
- ☺ **Build DB2 applications using SQL for consistent performance**
- ☺ **Developers can focus on business logic and customization in auto generated code**
- ☺ **Build pureQuery on all IBM databases and platforms**
  - DB2 Linux, UNIX and Windows (LUW) version 8.x onwards
  - DB2 for System i Version 5R2, 5R3 and 5R4
  - DB2 for System z Version 7, 8, 9
  - Derby Version 10.x onwards
  - Informix version 9.x onwards (on all supported platforms)

Why pureQuery?

What it means to a Application Developer

1. **Makes data access tasks much easier from within Java editor**
    - ➢ An environment that allows you to detect problems earlier in the development life cycle
    - ➢ Code SQL same way you code Java in Java editor and be equally productive
    - ➢ Never leave Java editor and be equally productive in Java and SQL
    - ➢ Auto generate simple data access layer with much less code than JDBC
    - ➢ Easily map between names of columns and object names
    - ➢ Easily change bean field assessors
    - ➢ Generate data access layer from table, views, procedures, stored procedures, nick names, beans, SQL …
2. **pureQuery API is a thin layer which sits on top of JDBC**
3. **pureQuery encapsulates best practices of JDBC programming in its API**
4. **Do not worry about coding differently for static SQL**
5. **For serious object relational (OR) mapping, use JPA on top of pureQuery**

Why pureQuery?

What it means to an Application DBA

1. **Makes problem determination much easier**
    - ➢ Application meta data is stored in catalogs with pureQuery
    - ➢ Effective monitoring with tooling
2. **pureQuery runtime allows to bind Java classes to make dynamic SQLs to static**
3. **pureQuery runtime works with all databases that have a JDBC driver**
4. **A single source of tooling for DBAs as well as application developers**

Why pureQuery?

Productivity features

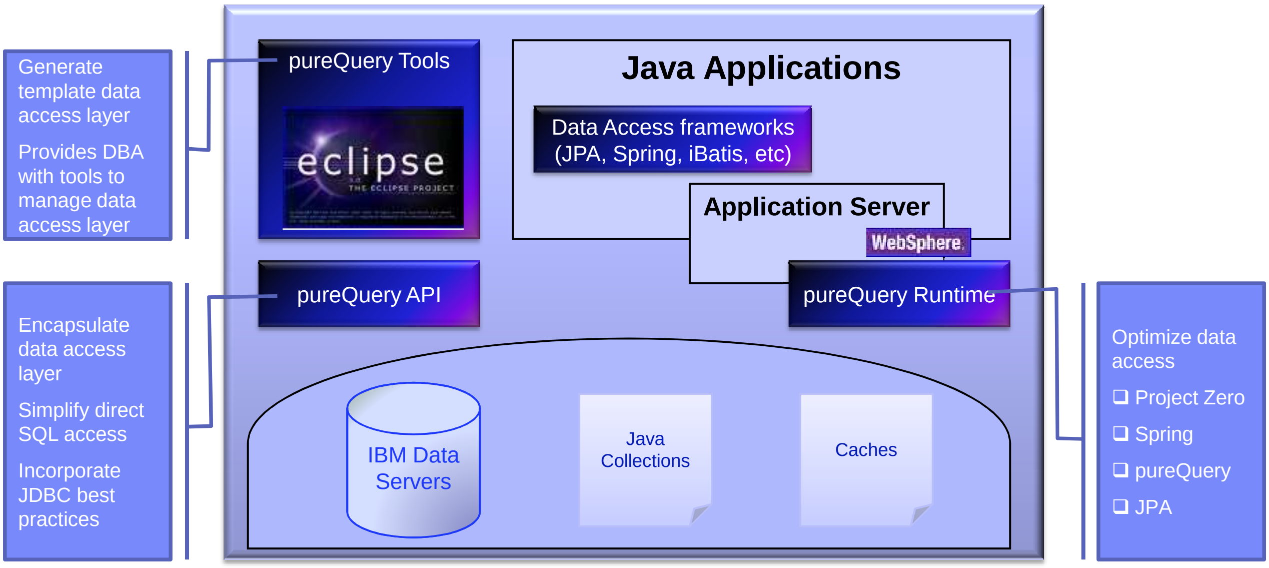
Requirements	pureQuery scenario
Need JDBC like code but much simpler	Use pureQuery in-line style API
Need SQL decoupled from application	Use of pureQuery Method style API *
Need JDBC like code without much exception handling code	Use pureQuery with Spring framework *
Database is designed first and applications comes second	Bottom-up approach from database *
Need all bean annotations and application SQL in one external XML file	Generate JPA XML mapping file
Migrate from JDBC to pureQuery	Bottom-up from SQL *
Have beans and database (Migrate from EJB)	Meet-in-the middle approach (Bean to database mapping) *
Have mapped beans (Migrate from EJBs)	Bottom-up from beans *
Need to work with database and in-memory data	Same SQL to use against database and in-memory objects *
Need reliable and consistent performance without more work	Static SQL without any code change *

* Covered in detail in later sections

---

pureQuery Architecture

IBM Database Servers



Generate template data access layer

Provides DBA with tools to manage data access layer

Encapsulate data access layer

Simplify direct SQL access

Incorporate JDBC best practices

pureQuery Tools

pureQuery API

Java Applications

Data Access frameworks (JPA, Spring, iBatis, etc)

Application Server

pureQuery Runtime

IBM Data Servers

Java Collections

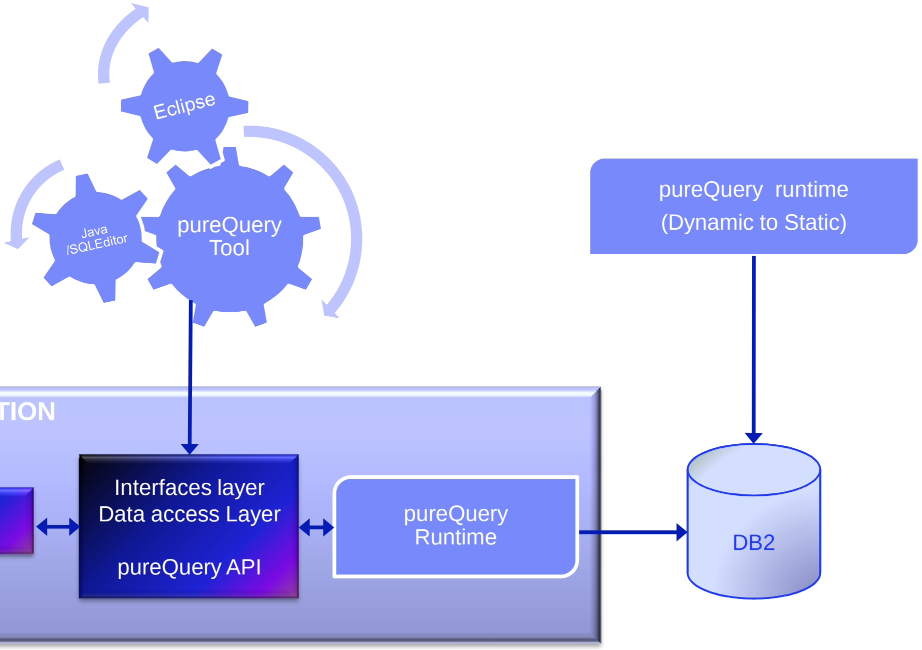
Caches

Optimize data access

- ❑ Project Zero
- ❑ Spring
- ❑ pureQuery
- ❑ JPA

pureQuery in Java Application

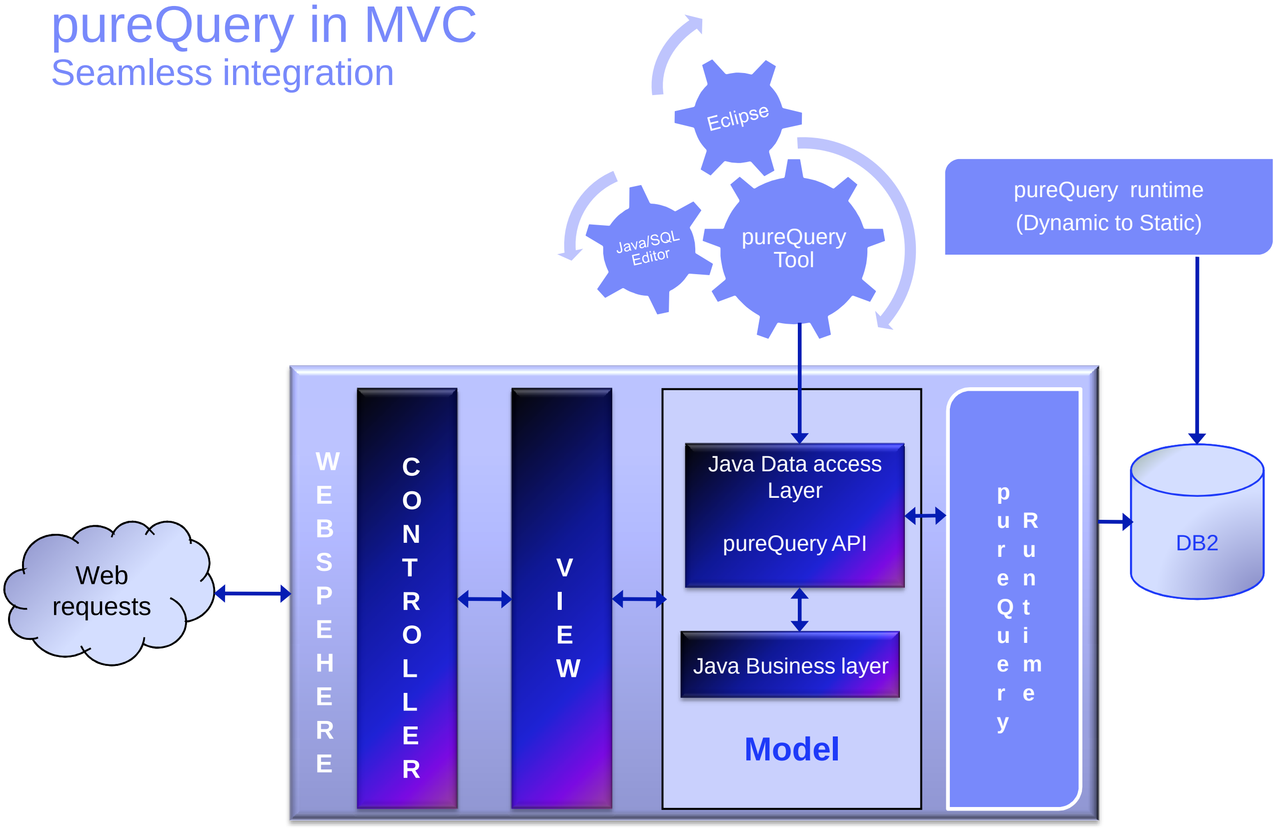
Seamless integration



---

pureQuery in MVC

Seamless integration



What is pureQuery?

pureQuery is Tools, API and Runtime

Tools
API
Runtime

1. **Tools**
   - ☐ SQL content assist
   - ☐ SQL execution (Run SQL)
   - ☐ SQL routine debugger
   - ☐ SQL / XQuery / XML editor
   - ☐ E-R diagramming
   - ☐ Browse and update statistics
   - ☐ Connection management with LDAP …
   - ☐ SQL validation
   - ☐ SQL Analysis (Explain plan)
   - ☐ Java routine debugger
   - ☐ XML schema editor
   - ☐ Data distribution viewer
   - ☐ Security access control
   - ☐ Data web services
2. **API**
   - Single API for all databases
   - Styles
     - In-line (Similar to JDBC or SQLJ)
     - Annotation Method (Similar to JDBC 4)
     - Named Query (Similar to Hibernate / JPA)
3. **Runtime**
   - Deploy dynamic to static either through GUI or Command line

Covered in labs

---

pureQuery Scope

What it is like and what it is not like

pureQuery is:

👍 Tools
API
Runtime

👍 similar to iBatis
👍 similar to JPAs runtime
👍 similar to Spring's DAO

pureQuery is NOT:

☒ OR (Object Relational) mapping model
☒ like Hibernate
☒ an application framework like Spring or EJB
☒ like Oracle's Toplink
☒ JPA

Value Proposition of pureQuery in Data Studio

What it means to you

pureQuery for DB2

- SQL is a first class citizen in Java development environment
- Static SQL for better performance
- Single API for in-memory and relational objects
- Access path locked-in at deployment for reliable/consistent behavior
- Application origin captured for all SQL statements for rapid problem identification
- Optimize existing JDBC applications using pureQuery to turn dynamic SQL into static for reliable/consistent performance
- Multiple versions of access path for easy fall back to prior versions

pureQuery for other databases

- Single API for queries for both relational and XML types
- Simple API syntax to eliminate the need for “get” or “set” methods
- Runtime will be existing JDBC, which is portable across all databases

---

pureQuery Tools

Lab 03 Explore pureQuery Tools

Tools
API
Runtime

An IBM Proof of Technology

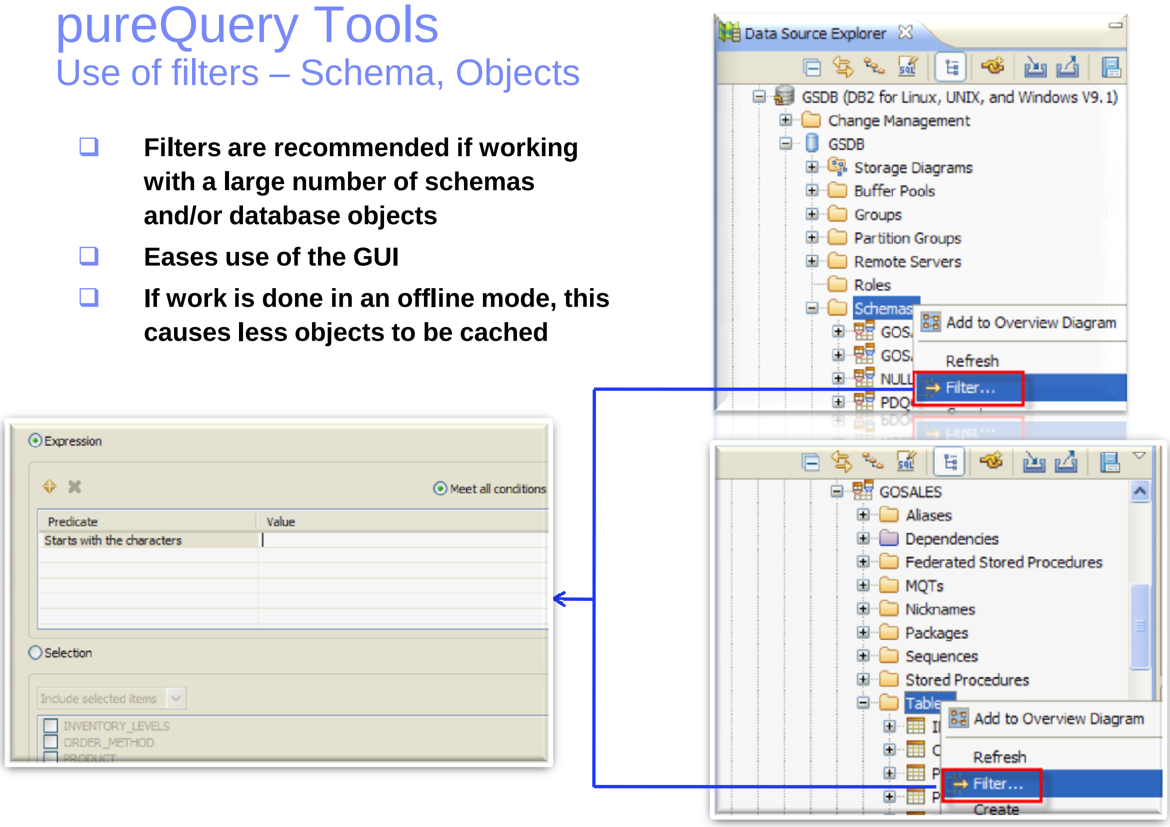
ON DEMAND BUSINESS™

18 slides

pureQuery Tools

Use of filters – Schema, Objects

- Filters are recommended if working with a large number of schemas and/or database objects
- Eases use of the GUI
- If work is done in an offline mode, this causes less objects to be cached

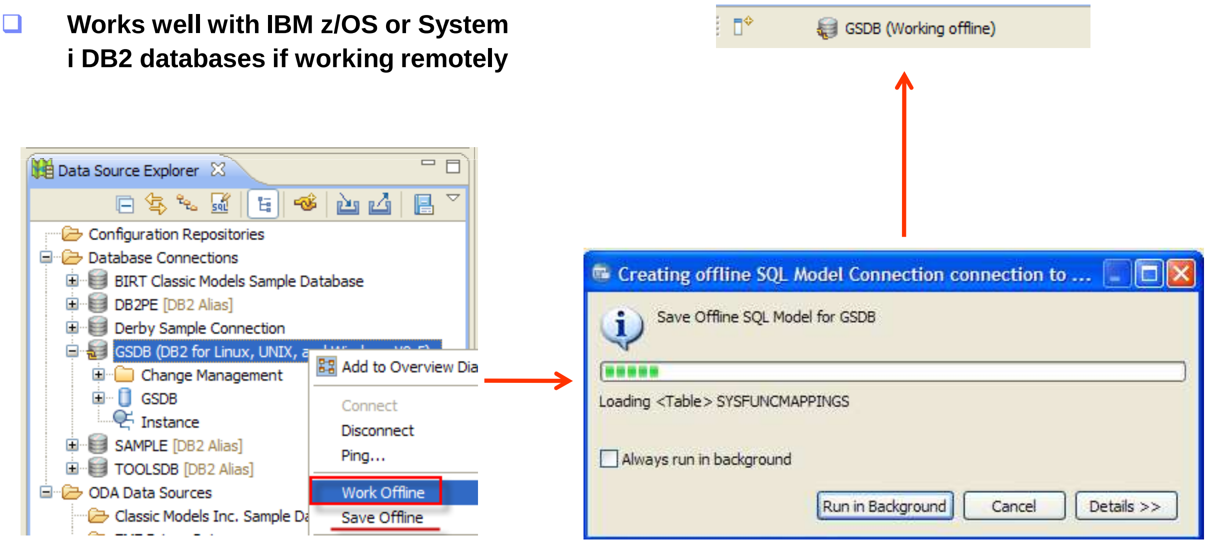


---

pureQuery Tools

Work in offline mode with a disconnected database

- Allows you to work in offline disconnected mode
- SQL context assist is still available in offline mode
- Works well with IBM z/OS or System i DB2 databases if working remotely



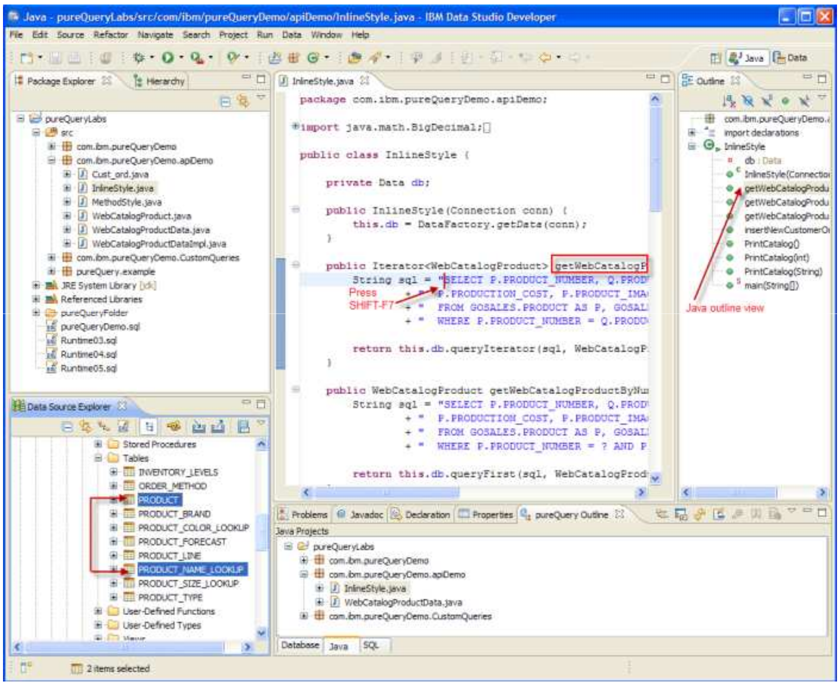
Save Offline SQL Model for GSDB

Loading <Table> SYSFUNCMAPPINGS

pureQuery Tools

Open definition in Data Source Explorer

- Similar to Java language outline view
- Column or Table name highlighted in database explorer view for easy reference



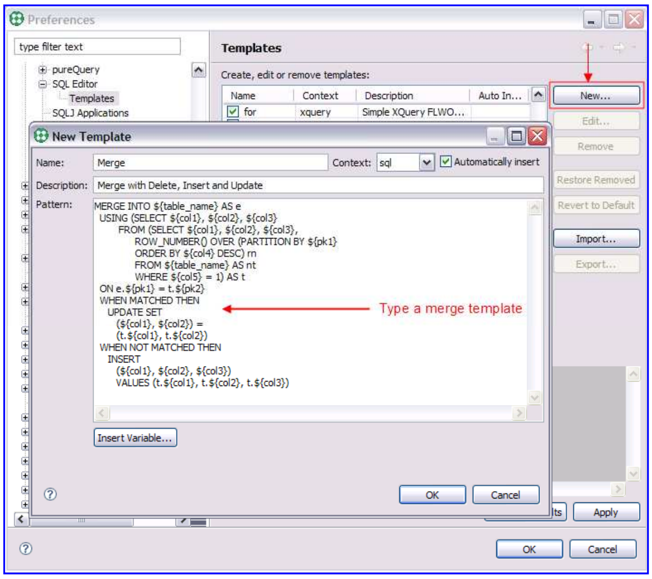
```
package com.ibm.pureQueryDemo.apiDemo;

import java.math.BigDecimal;

public class InlineStyle {

    private Data db;

    public InlineStyle(Connection conn) {
        this.db = DataFactory.getData(conn);
    }

    public Iterator<WebCatalogProduct> getWebCatalogP
        String sql = "SELECT P.PRODUCT_NUMBER, Q.PROD
                + "  P.PRODUCTION_COST, P.PRODUCT_IMA
                + "  FROM GOSALES.PRODUCT AS P, GOSAL
                + "  WHERE P.PRODUCT_NUMBER = Q.PRODU

        return this.db.queryIterator(sql, WebCatalogP.
    }

    public WebCatalogProduct getWebCatalogProductByNu
        String sql = "SELECT P.PRODUCT_NUMBER, Q.PROD
                + "  P.PRODUCTION_COST, P.PRODUCT_IMA
                + "  FROM GOSALES.PRODUCT AS P, GOSAL
                + "  WHERE P.PRODUCT_NUMBER = ? AND P

        return this.db.queryFirst(sql, WebCatalogProd
```

---

pureQuery Tools

SQL templates and customizations

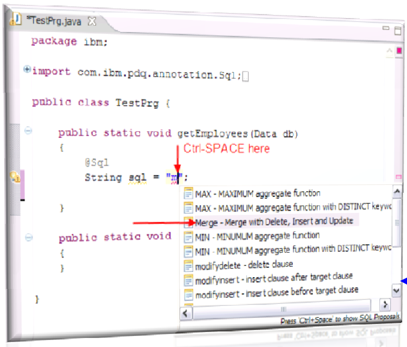
- Create your own SQL templates
- Use templates to write SQL that is frequently reused
- Use tabs to change the variable names after inserting SQL statement from the template through SQL context assist



Name: Merge
Context: sql
☑ Automatically insert
Description: Merge with Delete, Insert and Update
Pattern:

```
MERGE INTO ${table_name} AS e
 USING (SELECT ${col1}, ${col2}, ${col3}
      FROM (SELECT ${col1}, ${col2}, ${col3},
           ROW_NUMBER() OVER (PARTITION BY ${pk1}
           ORDER BY ${col4} DESC) rn
           FROM ${table_name} AS nt
           WHERE ${col5} = 1) AS t
 ON e.${pk1} = t.${pk2}
 WHEN MATCHED THEN
   UPDATE SET
     (${col1}, ${col2}) =
     (t.${col1}, t.${col2})
 WHEN NOT MATCHED THEN
   INSERT
     (${col1}, ${col2}, ${col3})
     VALUES (t.${col1}, t.${col2}, t.${col3})
```

Preferences
type filter text
pureQuery
SQL Editor
Templates
SQLJ Applications
Templates
Create, edit or remove templates:
Name
Context
Description
Auto In...
for
xquery
Simple XQuery FLWO...
New...
Edit...
Remove
Restore Removed
Revert to Default
Import...
Export...
New Template
Type a merge template
Insert Variable...
OK
Cancel
Apply
OK
Cancel



```
package ibm;

import com.ibm.pdq.annotation.Sql;

public class TestPrg {

    public static void getEmployees(Data db)
    {
        @Sql
        String sql = "m";

    }

    public static void
    {
    }

}
```

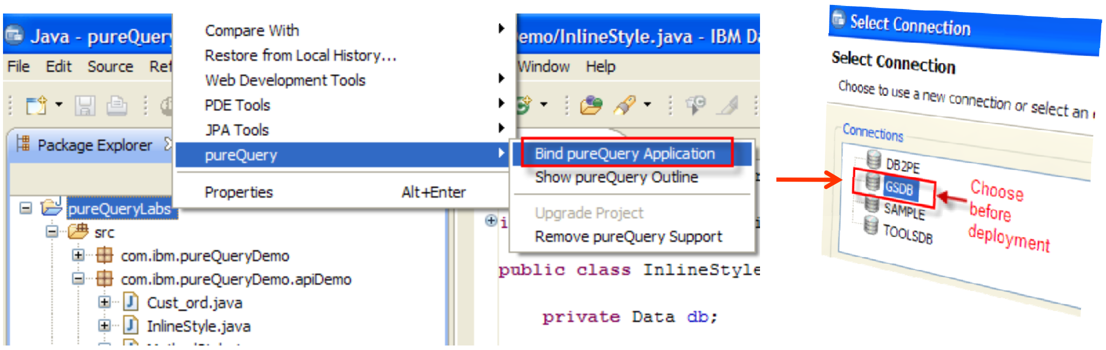
*TestPrg.java
Ctrl-SPACE here
MAX - MAXIMUM aggregate function
MAX - MAXIMUM aggregate function with DISTINCT keywc
Merge - Merge with Delete, Insert and Update
MIN - MINUMUM aggregate function
MIN - MINUMUM aggregate function with DISTINCT keywc
modifydelete - delete clause
modifyinsert - insert clause after target clause
modifyinsert - insert clause before target clause
Press 'Ctrl+Space' to show SQL Proposals

Use newly created merge template in your Java code using SQL assist

pureQuery Tools

Ease development to production migration

- ❑ **Applications can be developed on a development database**
  - **Yet deployed on a production database**
- ❑ **Just change the connection name for deployment**
- ❑ **Avoid having to qualify the identifiers that belong to the current schema**
  - **Change the current schema name easily through the GUI**

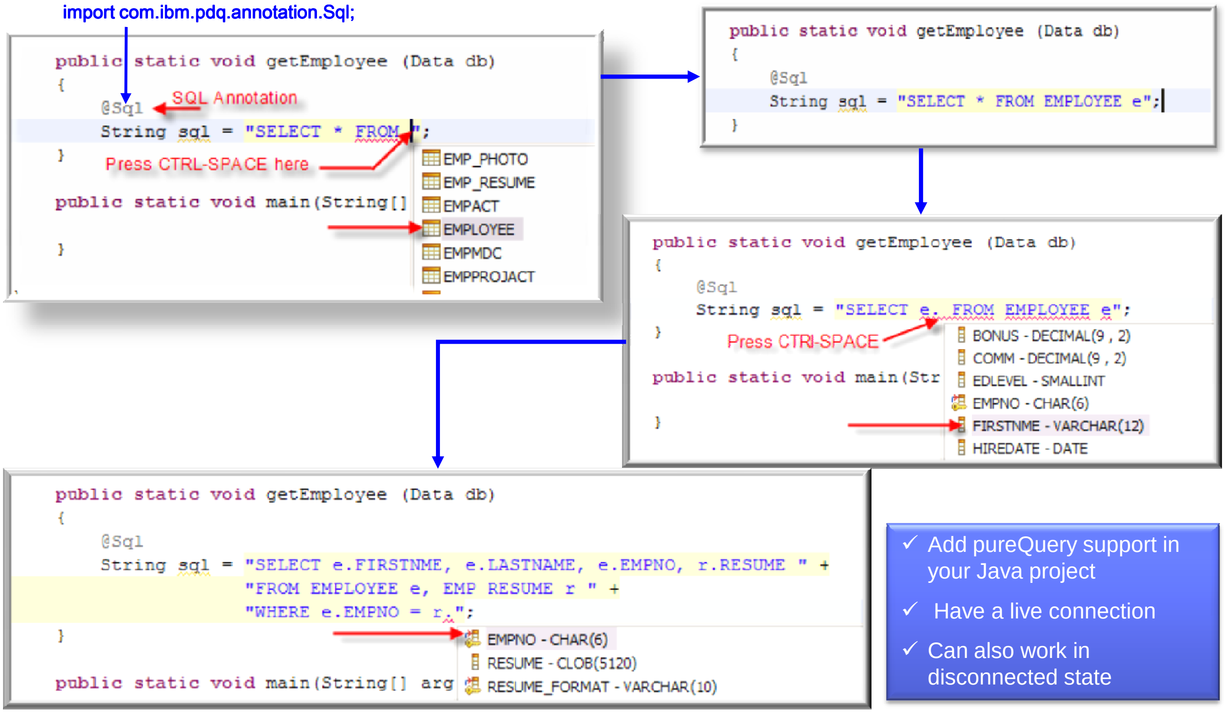


```
public class InlineStyle

    private Data db;
```

---

pureQuery Tools

SQL Assist



import com.ibm.pdq.annotation.Sql;

```
public static void getEmployee (Data db)
{
    @Sql
    String sql = "SELECT * FROM ";
}

public static void main(String[]

}
```

```
public static void getEmployee (Data db)
{
    @Sql
    String sql = "SELECT * FROM EMPLOYEE e";
}
```

```
public static void getEmployee (Data db)
{
    @Sql
    String sql = "SELECT e. FROM EMPLOYEE e";
}

public static void main(Str

}
```

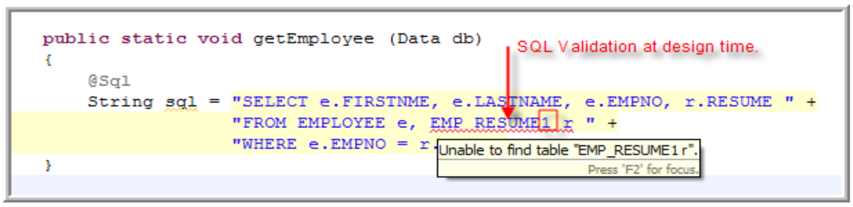
```
public static void getEmployee (Data db)
{
    @Sql
    String sql = "SELECT e.FIRSTNME, e.LASTNAME, e.EMPNO, r.RESUME " +
                 "FROM EMPLOYEE e, EMP_RESUME r " +
                 "WHERE e.EMPNO = r.";
}

public static void main(String[] arg
```

- ✓ Add pureQuery support in your Java project
- ✓ Have a live connection
- ✓ Can also work in disconnected state

pureQuery Tools
SQL Validation

SQL Validation

- Avoids runtime SQL syntax or missing table name errors
- Checks syntax of SQL as you code your program
- @Sql annotation required for validation and context help
- Customize SQL validation severity settings

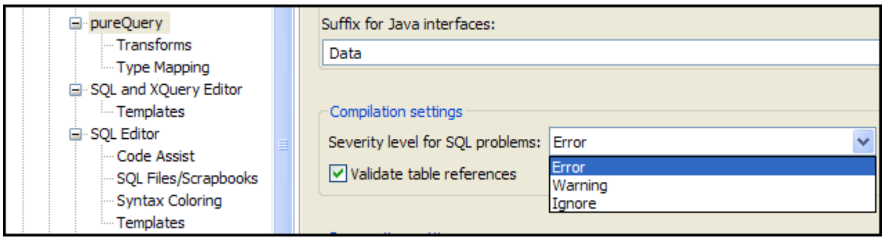


```
public static void getEmployee (Data db)
{
    @Sql
    String sql = "SELECT e.FIRSTNME, e.LASTNAME, e.EMPNO, r.RESUME " +
                 "FROM EMPLOYEE e, EMP_RESUME1 r " +
                 "WHERE e.EMPNO = r.
}
```

SQL Validation at design time.

Unable to find table "EMP_RESUME1 r".
Press 'F2' for focus.

- ✓ Syntactic validation
- ✓ Semantic validation
- ✓ Host variable validation



Suffix for Java interfaces: Data

Compilation settings

Severity level for SQL problems: Error

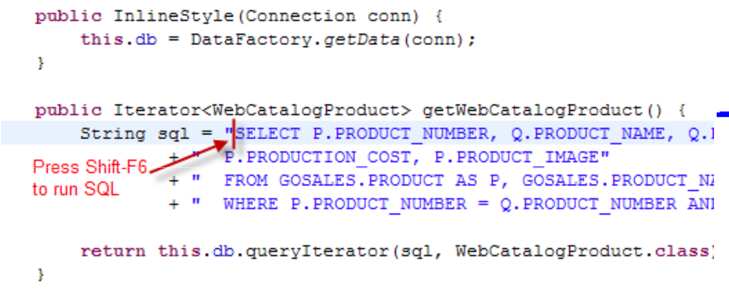
☑ Validate table references

---

pureQuery Tools
SQL Execution from within Java code

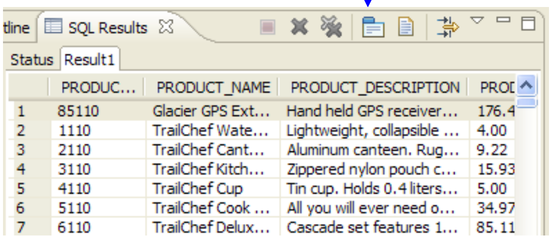
SQL Execution

- Press SHIFT-F6 within a SQL statement and supply values for any variables, if any.



```
public InlineStyle(Connection conn) {
    this.db = DataFactory.getData(conn);
}

public Iterator<WebCatalogProduct> getWebCatalogProduct() {
    String sql = "SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.I
            + "  P.PRODUCTION_COST, P.PRODUCT_IMAGE"
            + "  FROM GOSALES.PRODUCT AS P, GOSALES.PRODUCT_NA
            + "  WHERE P.PRODUCT_NUMBER = Q.PRODUCT_NUMBER ANI

    return this.db.queryIterator(sql, WebCatalogProduct.class)
}
```

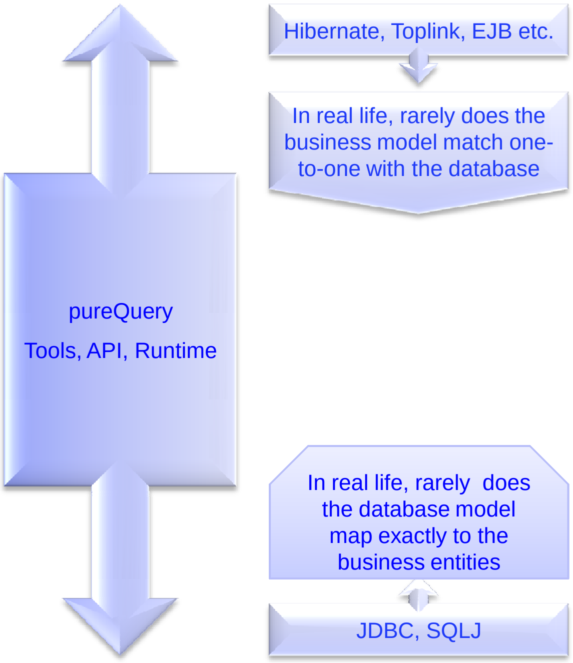
Press Shift-F6 to run SQL



	PRODUC...	PRODUCT_NAME	PRODUCT_DESCRIPTION	PROD
1	85110	Glacier GPS Ext...	Hand held GPS receiver...	176.4
2	1110	TrailChef Wate...	Lightweight, collapsible ...	4.00
3	2110	TrailChef Cant...	Aluminum canteen. Rug...	9.22
4	3110	TrailChef Kitch...	Zippered nylon pouch c...	15.93
5	4110	TrailChef Cup	Tin cup. Holds 0.4 liters...	5.00
6	5110	TrailChef Cook ...	All you will ever need o...	34.97
7	6110	TrailChef Delux...	Cascade set features 1...	85.11

Application Evolution

Meet-in-the-middle



Hibernate, Toplink, EJB etc.

In real life, rarely does the business model match one-to-one with the database

pureQuery

Tools, API, Runtime

In real life, rarely does the database model map exactly to the business entities

JDBC, SQLJ

Top down application evolution

- ❑ A purist application development approach from object oriented philosophy
- ❑ System is designed as top-down from object domain model
- ↓ Have beans and want to map them to the database
- ↓ Object model already exists

Meet-in-the middle approach

- ↑ Database already exists
- ↑ Have tables / SQL and want to map them to the beans
- ❑ System is designed bottom-up based upon created database model
- ❑ A purist database model up to 3rd degree of normalization

Bottom-up application evolution

Rapid Application Development

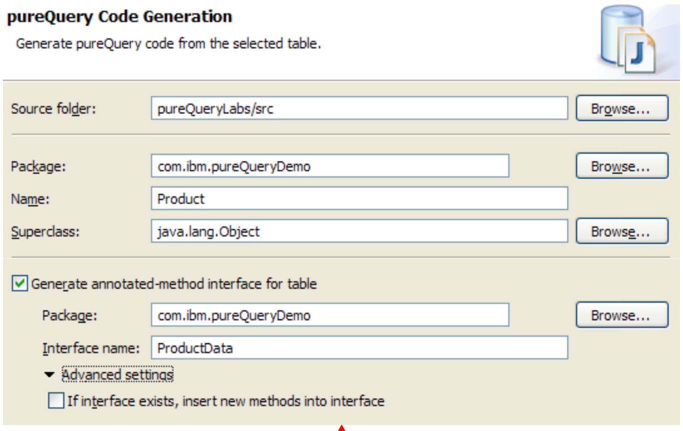
Scenarios for pureQuery

Top down - Beans	• Have mapped beans • Generate database schema
SQL in XML	• Have all bean annotations • Create SQL in XML file
Static SQL	• Need better performance
Meet in the middle	• Map beans to database • Generate data access code
Query over collections	• Work with in-memory and relational data
Bottom up - Beans	• Mapped beans available • Generate data access code
Bottom up - Database	• Database available • Generate data access code
Bottom up - SQL	• Reuse SQL • Generate data access code

Rapid Application Development

Generate code from table, view

- Increase developer activity
- Integrated Java and SQL editor
- pureQuery tools generate simple data access layer
- Flexible data access
  - ✓ Annotated method
  - ✓ In-line method
- Ability to generate test class or JUnit test cases



pureQuery Code Generation

Generate pureQuery code from the selected table.

Source folder: pureQueryLabs/src

Package: com.ibm.pureQueryDemo

Name: Product

Superclass: java.lang.Object

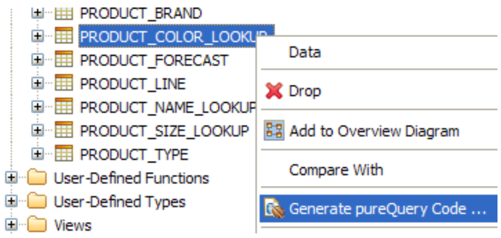
☑ Generate annotated-method interface for table

Package: com.ibm.pureQueryDemo

Interface name: ProductData

▾ Advanced settings

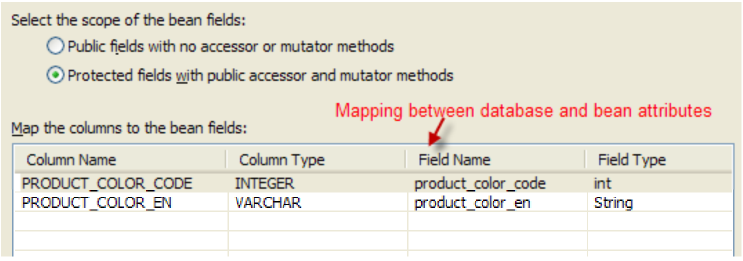
☐ If interface exists, insert new methods into interface



Rapid Application Development

Generate code from table, view (Continued)

- Ability to map columns names to bean name
- Choose different methods for SQL CRUD (Create, Read, Update, Delete) generation



Select the scope of the bean fields:

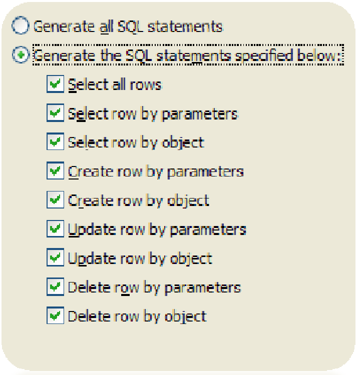
○ Public fields with no accessor or mutator methods

● Protected fields with public accessor and mutator methods

Map the columns to the bean fields:

Mapping between database and bean attributes

Column Name	Column Type	Field Name	Field Type
PRODUCT_COLOR_CODE	INTEGER	product_color_code	int
PRODUCT_COLOR_EN	VARCHAR	product_color_en	String



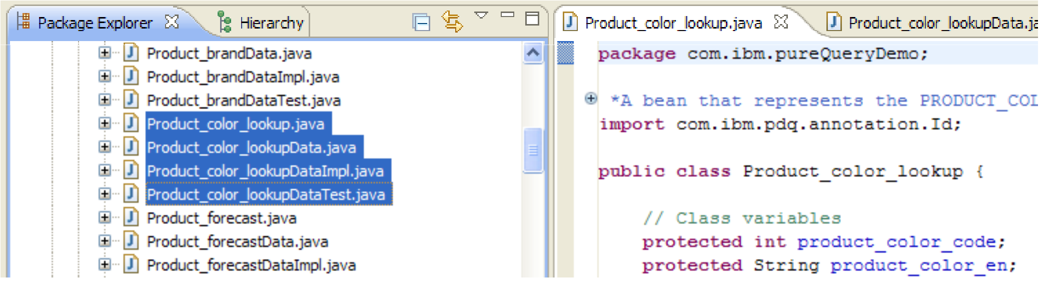
○ Generate all SQL statements

● Generate the SQL statements specified below:

☑ Select all rows
☑ Select row by parameters
☑ Select row by object
☑ Create row by parameters
☑ Create row by object
☑ Update row by parameters
☑ Update row by object
☑ Delete row by parameters
☑ Delete row by object

Rapid Application Development
Generate code from table, view (Continued)

- The code is generated for
  - ✓ Bean
  - ✓ Interface giving a method for SQL
  - ✓ Test class or JUnit test case.

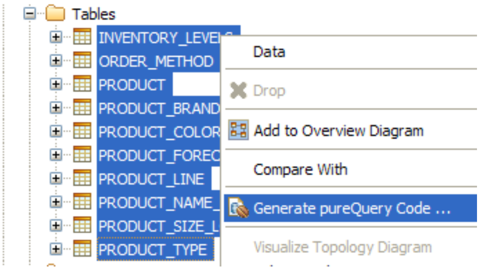


Product_brandData.java
Product_brandDataImpl.java
Product_brandDataTest.java
Product_color_lookup.java
Product_color_lookupData.java
Product_color_lookupDataImpl.java
Product_color_lookupDataTest.java
Product_forecast.java
Product_forecastData.java
Product_forecastDataImpl.java

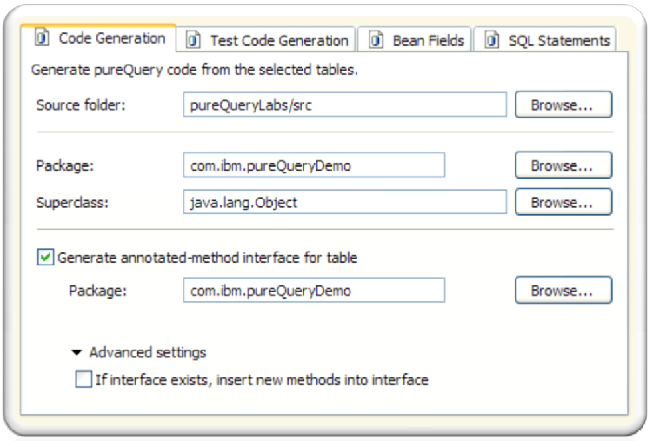
```
package com.ibm.pureQueryDemo;

 *A bean that represents the PRODUCT_COI
import com.ibm.pdq.annotation.Id;

public class Product_color_lookup {

    // Class variables
    protected int product_color_code;
    protected String product_color_en;
```

Rapid Application Development
Generate code from multiple tables

Step-1: Select multiple tables



Step-2: Specify global settings for all tables



Generate pureQuery code from the selected tables.

Source folder: pureQueryLabs/src

Package: com.ibm.pureQueryDemo

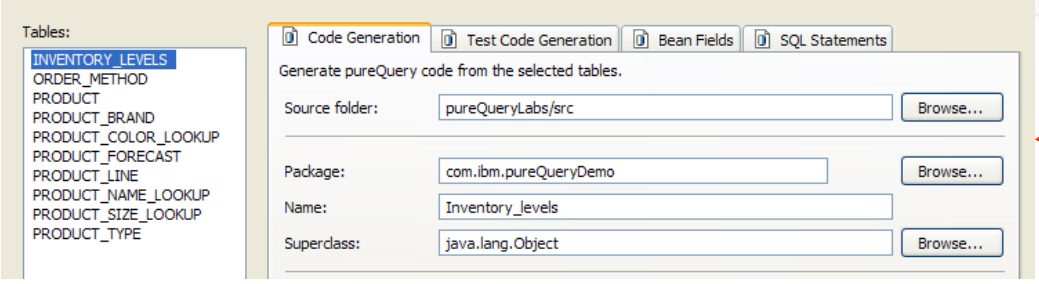
Superclass: java.lang.Object

☑ Generate annotated-method interface for table

Package: com.ibm.pureQueryDemo

☐ If interface exists, insert new methods into interface

Step-3: Customize settings at the table level



Tables:
INVENTORY_LEVELS
ORDER_METHOD
PRODUCT
PRODUCT_BRAND
PRODUCT_COLOR_LOOKUP
PRODUCT_FORECAST
PRODUCT_LINE
PRODUCT_NAME_LOOKUP
PRODUCT_SIZE_LOOKUP
PRODUCT_TYPE

Generate pureQuery code from the selected tables.

Source folder: pureQueryLabs/src

Package: com.ibm.pureQueryDemo

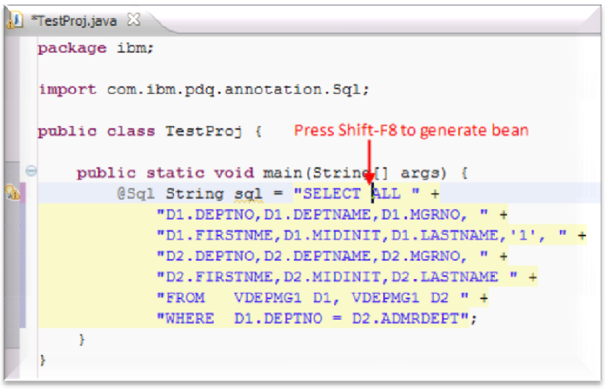
Name: Inventory_levels

Superclass: java.lang.Object

Rapid Application Development

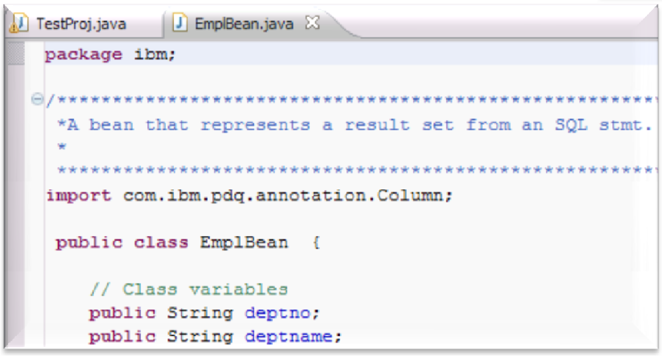
Generate bean from SQL

- After pressing Shift-F8 in SQL in Java program
- Specify package name for bean (ibm)
- Specify bean name (EmplBean)
- Specify type of bean
  - ✓ With public fields with no accessor or mutator methods, or,
  - ✓ Protected fields with public accessor and mutator methods
- Map SQL columns name to bean fields
- Check generated bean from SQL



```
package ibm;

import com.ibm.pdq.annotation.Sql;

public class TestProj {

    public static void main(String[] args) {
        @Sql String sql = "SELECT ALL " +
            "D1.DEPTNO,D1.DEPTNAME,D1.MGRNO, " +
            "D1.FIRSTNME,D1.MIDINIT,D1.LASTNAME,'1', " +
            "D2.DEPTNO,D2.DEPTNAME,D2.MGRNO, " +
            "D2.FIRSTNME,D2.MIDINIT,D2.LASTNAME " +
            "FROM   VDEPMG1 D1, VDEPMG1 D2 " +
            "WHERE  D1.DEPTNO = D2.ADMRDEPT";
    }
}
```

Press Shift-F8 to generate bean



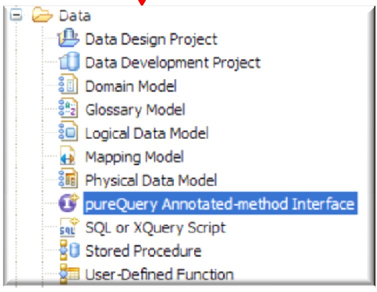
```
package ibm;

/*********************************************************
 *A bean that represents a result set from an SQL stmt.
 *
 *********************************************************
import com.ibm.pdq.annotation.Column;

public class EmplBean  {

    // Class variables
    public String deptno;
    public String deptname;
```

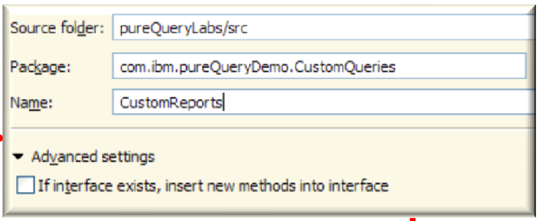
- Reverse engineer bean to SQL
- Click Window > pureQuery > Generate DDL

Rapid Application Development

Generate code from SQL Scripts

File>New>Other





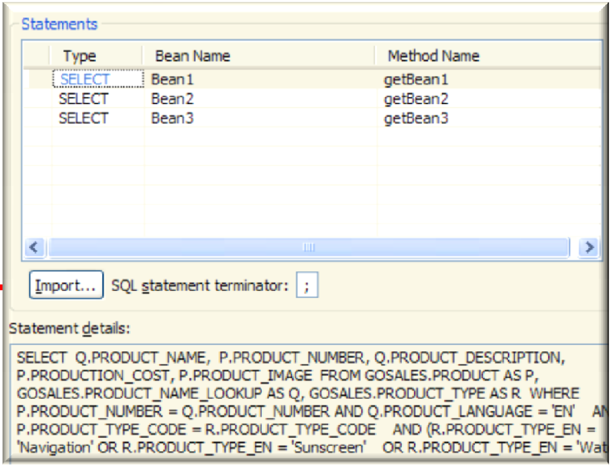
Source folder: pureQueryLabs/src

Package: com.ibm.pureQueryDemo.CustomQueries

Name: CustomReports

▾ Advanced settings

☐ If interface exists, insert new methods into interface



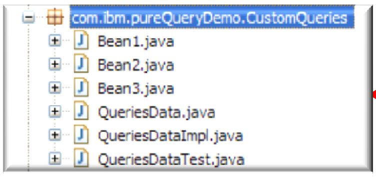
Statements

Type	Bean Name	Method Name
SELECT	Bean1	getBean1
SELECT	Bean2	getBean2
SELECT	Bean3	getBean3

Import... SQL statement terminator: ;

Statement details:

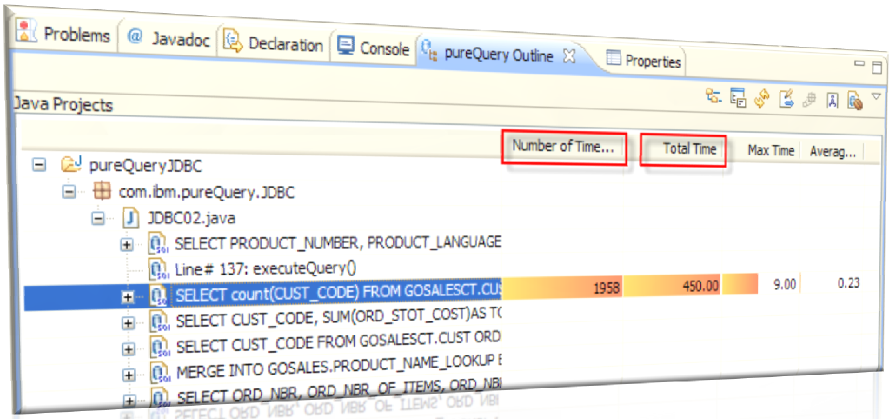
```
SELECT  Q.PRODUCT_NAME,  P.PRODUCT_NUMBER, Q.PRODUCT_DESCRIPTION,
P.PRODUCTION_COST, P.PRODUCT_IMAGE  FROM GOSALES.PRODUCT AS P,
GOSALES.PRODUCT_NAME_LOOKUP AS Q, GOSALES.PRODUCT_TYPE AS R  WHERE
P.PRODUCT_NUMBER = Q.PRODUCT_NUMBER AND Q.PRODUCT_LANGUAGE = 'EN'    AN
P.PRODUCT_TYPE_CODE = R.PRODUCT_TYPE_CODE    AND (R.PRODUCT_TYPE_EN =
'Navigation' OR R.PRODUCT_TYPE_EN = 'Sunscreen'    OR R.PRODUCT_TYPE_EN = 'Wat
```



pureQuery Outline View

Speed-up isolation of problems

- Capture application-SQL-data object correlation (with or without the source code)
- Trace SQL statements to Java source code for faster problem isolation
- Enhance impact analysis by identifying specific application code impacted by a database changes
- Answer “Where used” questions like “Where is this column used within the application?”
- Use with modern Java frameworks e.g. Hibernate, Spring, iBatis, OpenJPA



pureQueryJDBC
com.ibm.pureQuery.JDBC
JDBC02.java
SELECT PRODUCT_NUMBER, PRODUCT_LANGUAGE
Line# 137: executeQuery()
SELECT count(CUST_CODE) FROM GOSALESCT.CU 1958 450.00 9.00 0.23
SELECT CUST_CODE, SUM(ORD_STOT_COST)AS T
SELECT CUST_CODE FROM GOSALESCT.CUST ORD
MERGE INTO GOSALES.PRODUCT_NAME_LOOKUP
SELECT ORD_NBR, ORD_NBR_OF_ITEMS, ORD_NB

---

pureQuery API

Lab 04 Explore pureQuery API

Tools
API
Runtime

An IBM Proof of Technology

ON DEMAND BUSINESS™

14 slides

pureQuery API

Comparison

In-Line API

- Simplified and intelligent direct SQL access

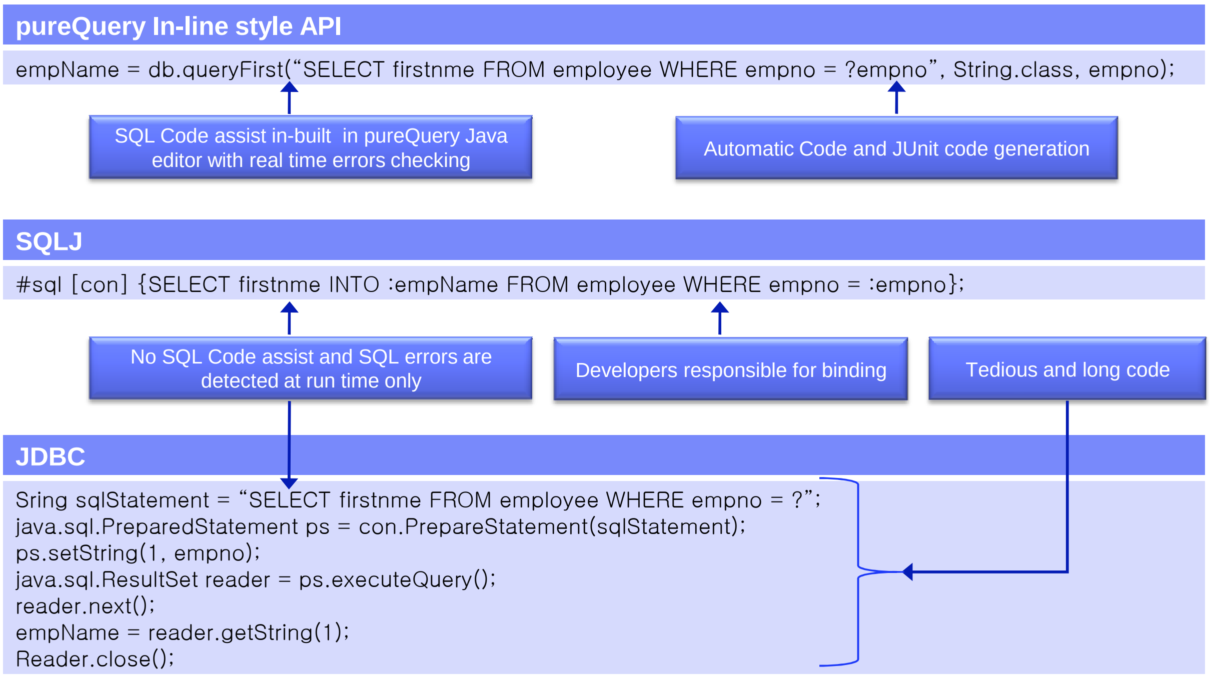
Annotation Method

- SQL defined in Java interface files
- Encapsulate SQL in Java methods
- Named Query – encapsulate SQL in JPA XML files **(Extension of annotation method)**

Feature	In-line API	Annotation method API
pureQuery static SQL	**No**	**Yes**
pureQuery XML scenario	**No**	**Yes**
Code generation merge new	**No**	**Yes**
Result set handlers	**Yes**	**No**
All other features	**Yes**	**Yes**

In-line pureQuery API

Retrieve single row comparison with JDBC / SQLJ



pureQuery In-line style API

```
empName = db.queryFirst(“SELECT firstnme FROM employee WHERE empno = ?empno”, String.class, empno);
```

SQL Code assist in-built in pureQuery Java editor with real time errors checking

Automatic Code and JUnit code generation

SQLJ

```
#sql [con] {SELECT firstnme INTO :empName FROM employee WHERE empno = :empno};
```

No SQL Code assist and SQL errors are detected at run time only

Developers responsible for binding

Tedious and long code

JDBC

```
Sring sqlStatement = “SELECT firstnme FROM employee WHERE empno = ?”;
java.sql.PreparedStatement ps = con.PrepareStatement(sqlStatement);
ps.setString(1, empno);
java.sql.ResultSet reader = ps.executeQuery();
reader.next();
empName = reader.getString(1);
Reader.close();
```

In-line pureQuery API

How it works?

pureQuery In-line style API

```
empName = db.queryFirst("SELECT firstnme FROM employee WHERE empno = ?empno", String.class, empno);
```

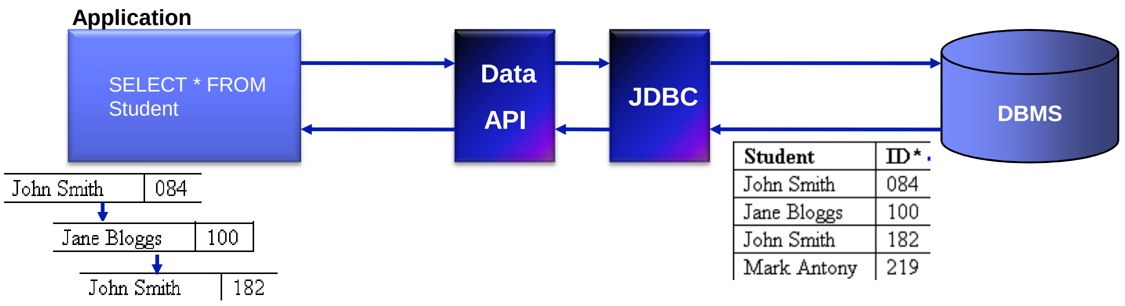
In-line style API

- Uses JDBC `PreparedStatement` to prepare and examine the SQL.
- Created default `statementDescriptor` object and the necessary default handlers.
- Invokes an internal pureQuery method to execute the statement and handle input and output parameters
  - The internal API is similar to what the pureQuery generated code for annotated method style will use.

In-line pureQuery API

Data API

- **Import** `com.ibm.pdq.runtime`
- **Data –** The Java class containing methods for accessing database.
- **Simple queries executed right out of the box**
- **Return result set in any of the common Java collection types**
  - ✓ **Arrays**
  - ✓ **List**
  - ✓ **Maps**
  - ✓ **Iterators**
- **Easily extensible to more complex mapping**



Student	ID*
John Smith	084
Jane Bloggs	100
John Smith	182
Mark Antony	219

John Smith 084
Jane Bloggs 100
John Smith 182

In-line pureQuery API

SQL Statements in code

- ❑ **SQL statements coded directly in the Java code with code-assist.**
- ❑ **SQL can be fully declared and embedded or constructed at execution time.**
- ❑ **All standard SQL supported including DML, DDL**

In-line style pureQuery API

```
import com.acme.Customer;
Connection con = DriverManager.getConnection(…);
Data db = DataFactory.getData(con);
Customer c;
int region = 123;
Iterator<Customer> customers =
   db.queryIterator("SELECT custId, name " +
                    "FROM Customer WHERE region=?1",
                     Customer.class, region);
while (customers.hasNext())
{
    c = customers.next();
    System.out.println(c.custId+" "+c.name);
}
((ResultIterator) customers).close();
```

Customer Class (Customer.java)

```
public Customer
{
    @Id public Integer custId;
    public String name;
    public String addressL1;
    public String city;
    @Column(name="REGION")
    public Integer storeRegion;
 ...
}
```

Return result sets in

- ❑ Iterators
- ❑ Lists
- ❑ Maps
- ❑ Arrays and others

In-line pureQuery API

Query parameters alternative

- ❑ **Standard parameter markers** (e.g. ?)
- ❑ **Numbered parameter markers** (e.g. ?1, ?2)
- ❑ **Named parameters** (e.g. :name, :address)
  - ✓ **Parameters can be passed as a bean**

```
Employee employee = db.queryFirst(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, Employee.class, empNo);
```

  - ✓ **Parameters can be passed as a map**

```
Map parms = (new HashMap()).put(“empNo”, “000120”);
Employee employee = db.queryFirst(“SELECT * FROM employee “ +
                                  “WHERE empno = :empNo”, Employee.class, parms);
```

In-line pureQuery API

Query - Examples

- ❑ **BEANS**

```
Employee employee = db.queryFirst(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, Employee.class, empNo);
```

- ❑ **LIST**

```
List<Employee> employees = db.queryList(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, Employee.class, empNo);
```

- ❑ **ARRAYS**

```
Employee[] employees = db.queryArray(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, Employee.class, empNo);
```

- ❑ **ITERATORS**

```
Iterator<Employee> employees = db.queryIterator(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, Employee.class, empNo);
```

- ❑ **Query result can be returned as a MAP**

```
Map<String,Object> employee = db.queryFirst(“SELECT * FROM employee “ +
                                  “WHERE empno = ?”, empNo);

List<Map<String, Object>> employees = db.queryList(“SELECT * FROM employee “ +
                                  “WHERE LastName LIKE ?”, “Vi%”);
```

- ❑ **In the result map**
  - ✓ The column names become String keys
  - ✓ The column values become Object values

In-line pureQuery API

Insert, Update and Delete

- ❑ **INSERT, UPDATE and DELETE are performed via *update* method**
  - ✓ INSERT

```
int rowsAffected = db.update(“INSERT INTO employee (empno, firstname, lastname) “ +
                             “VALUES (:empno, :firstname, :lastname)”, employee);
```

  - ✓ UPDATE

```
int rowsAffected = db.update(“UPDATE employee SET firstname = :firstname, lastname =  “ +
                             “:lastname WHERE empno = :empno)”, employee);
```

  - ✓ DELETE

```
int rowsAffected = db.update(“DELETE FROM employee WHERE empno = :empno “, empno);
```

  - ✓ Many INSERTS or UPDATES – (Automatically batches)

```
int rowsAffected = db.updateMany(“INSERT INTO employee (empno, firstname, lastname) “ +
                             “VALUES (:empno, :firstname, :lastname)”, employee);
```

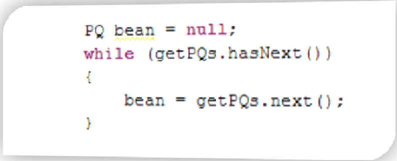
Annotated Method pureQuery API

For static SQL at runtime

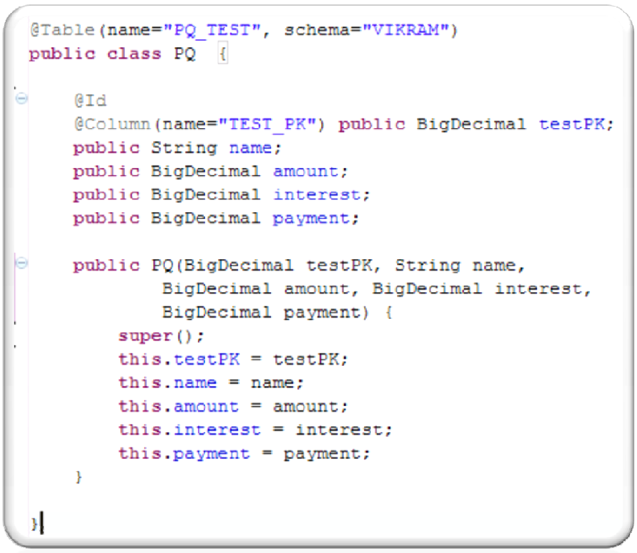
pureQuery annotated method style API

@Select(sql="select * from PQ_TEST") ⟵ In interface file
Iterator<PQ> getPQs = data.getPQs(); ⟵ Call getPQs() from business layer

- ✓ SQL can be in annotation or in XML file
- ✓ Source does not have dependencies on API
- ✓ Automatic Code and JUnit code generation
- ✓ Manage SQL separately, uses JPA XML format



```
PQ bean = null;
while (getPQs.hasNext())
{
    bean = getPQs.next();
}
```



```
@Table(name="PQ_TEST", schema="VIKRAM")
public class PQ  {

    @Id
    @Column(name="TEST_PK") public BigDecimal testPK;
    public String name;
    public BigDecimal amount;
    public BigDecimal interest;
    public BigDecimal payment;

    public PQ(BigDecimal testPK, String name,
            BigDecimal amount, BigDecimal interest,
            BigDecimal payment) {
        super();
        this.testPK = testPK;
        this.name = name;
        this.amount = amount;
        this.interest = interest;
        this.payment = payment;
    }

}
```

Annotated Method pureQuery API

How it works?

Annotated Method pureQuery API

@Select(sql="select * from PQ_TEST") Iterator<PQ> getPQs = data.getPQs();

Annotated method API

- SQL string is defined as Java annotation in an interface file
- @SELECT, @UPDATE and @CALL annotations are placed on user defined method declarations
- A code generator pre-processes the interface and generates code for each declared annotated method
- Generated code has methods to execute SQL statement defined in annotation
- Pre-defining SQL string is essential for static execution support for this coding style

Annotated Method pureQuery API

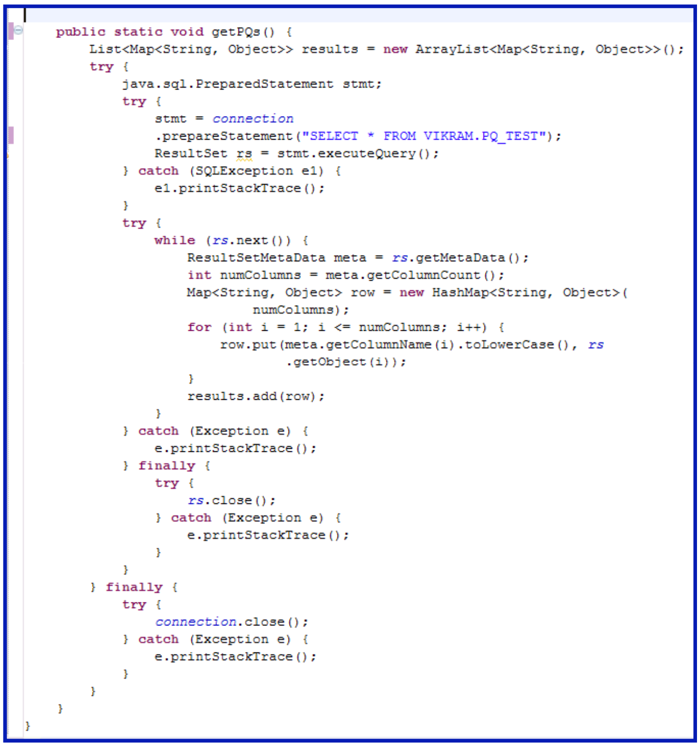
Advantages and compare

Advantages of annotated method

- Allows for organizing / isolating SQL accessor methods into separate interface files
- Administrator can create libraries of SQL methods
- Developer accesses pre-canned Java objects and access methods
- Easy deployment of static SQL
- Application metadata is gathered, stored and registered.

Compare pureQuery vs. JDBC

- Simple vs. long
- Auto gen. vs. manually written
- Productive vs. long time



```
    public static void getPQs() {
        List<Map<String, Object>> results = new ArrayList<Map<String, Object>>();
        try {
            java.sql.PreparedStatement stmt;
            try {
                stmt = connection
                .prepareStatement("SELECT * FROM VIKRAM.PQ_TEST");
                ResultSet rs = stmt.executeQuery();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            try {
                while (rs.next()) {
                    ResultSetMetaData meta = rs.getMetaData();
                    int numColumns = meta.getColumnCount();
                    Map<String, Object> row = new HashMap<String, Object>(
                            numColumns);
                    for (int i = 1; i <= numColumns; i++) {
                        row.put(meta.getColumnName(i).toLowerCase(), rs
                                .getObject(i));
                    }
                    results.add(row);
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    rs.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        } finally {
            try {
                connection.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

---

pureQuery Prerequisites

Recommended settings

- ☐ **IBM JCC Driver 3.52.xx or higher**
  1. db2jcc.jar - JDBC driver for Type 2 and 4 (JDBC 3 Standard)
  2. db2jcc4.jar - JDBC driver for Type 2 and 4 (JDBC 4 Standard)

- ☐ **JDK level 1.5 or higher (JDBC 3.0)**

- ☐ **Set these JCC properties**
  - retrieveMessagesFromServerOnGetMessage=true
  - emulateParameterMetaDataForZCalls=1

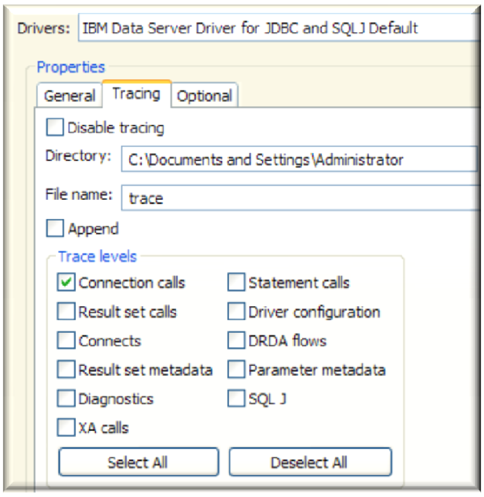
pureQuery Logs

JCC Driver Settings

- **pureQuery sits on top of JDBC driver, using JCC logging solves the problem**
- **Can be set at connection level by modifying tracing options**
- **Can be set for execution of a pureQuery application by adding properties to the connection URL.**

```
jdbc:db2://localhost:50000/SAMPLE:traceFile=c:/jdbc4.log;TraceLevel=TRACE_ALL;
```

- **Can be set to multiple directories if using multi-threading or connection pooling**



Drivers: IBM Data Server Driver for JDBC and SQLJ Default

☐ Disable tracing

Directory: C:\Documents and Settings\Administrator

File name: trace

☐ Append

☑ Connection calls
☐ Result set calls
☐ Connects
☐ Result set metadata
☐ Diagnostics
☐ XA calls
☐ Statement calls
☐ Driver configuration
☐ DRDA flows
☐ Parameter metadata
☐ SQL J

Use tracing through a property file = `DB2JccConfiguraion.properties`

```
db2.jcc.override.traceFile=C:/temp/jdbc.trace
db2.jcc.override.traceFileAppend=true
db2.jcc.override.TraceLevel=TRACE_ALL
db2.jcc.override.currentSchema=VIKRAM
```

End of Core Topics – Part 2

Presentation By:

Vikram Khatri

Certified I/T Specialist, DB2 Migrations
IBM Advanced Technical Expert DB2 for Clusters
IBM Certified Database Administrator for DB2 UDB V8.1 for Linux, UNIX & Windows
IBM Certified Database Administrator for DB2 UDB V9 for Linux, UNIX & Windows
IBM Certified Solutions Expert for DB2 UDB V8.1 Family Application Development
Project Management Professional Certified
vikram.khatri@us.ibm.com

Burt Vialpando

Certified I/T Specialist, DB2 Migrations, I/T Specialist Certification Board
IBM Certified Advanced Database Administrator for DB2 UDB V8.1 for Linux, UNIX & Windows
IBM Certified Solutions Expert for DB2 UDB V8.1 Family Application Development
IBM Certified Solutions Designer for DB2 Business Intelligence V8
IBM Certified Solutions Expert for DB2 UDB V7.1 Database Administration for OS/390
Oracle 9i Database Administrator Certified Associate
burt.vialpando@us.ibm.com

IBM Software Group

IBM Data Studio pureQuery
for DBAs and Application Developers on z/OS

Part 2

An IBM Proof of Technology

ON DEMAND BUSINESS™

Version 2.2 May 7, 2009

Vikram S Khatri and Burt Vialpando

PoT Overview
Agenda – Core Topics – Part 2

Topics part 1	**Data Studio Environment Overview**	**- Lab 01**
	pureQuery Environment Overview	**- Lab 02**
	pureQuery Tools	**- Lab 03**
	pureQuery API	**- Lab 04**
Topics part 2	**pureQuery Runtime**	**- Lab 05**
	pureQuery Explain	**- Lab 06**
	pureQuery for JDBC Applications	**- Lab 07**
	pureQuery Advanced Concepts	**- Lab 08**

IBM Software Group

pureQuery Runtime

Tools
API
Runtime

Lab 05 Explore pureQuery Runtime

An IBM Proof of Technology

ON DEMAND BUSINESS™

22 slides

---

pureQuery Runtime

Capabilities

pureQuery Runtime

Control performance

- Decide at deployment time how the SQL is executed
- Understand and lock down the access plan for SQL
- Replace suboptimal SQL without changing the application

Control security

- Prevent SQL injection
- Prevent execution of unauthorized SQL
- Better manage database security

See inside applications that are driving your database

- Understand where SQL comes from
- Understand when frameworks and ORM's are getting in the way

Simplify problem determination and troubleshooting

- Correlate problem SQL with applications, ORM's and frameworks

DB2 Packages

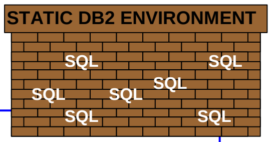
What and Why

- **DB2 packages: required for all SQL executing in the database**
  - **Packages control the SQL execution environment**
    - **eg. Isolation level, blocking, parallelism, etc.**
  - **Packages contain the compiler-determined access path**
    - **eg. What steps will have an index scan, table scan, merge, sort, filter, etc.**
  - **Packages come in two types:**
    - **Dynamic - Compiled and bound at runtime ("on the fly")**
    - **Static - Compiled and bound during preparation & stored in the database ***
- **Either package type is placed in the package cache during execution**
  - **This allows, but does not guarantee, repeated use**
  - **Any relational database (RDBMS) product has the same exposure: a finite sized package (SQL) cache**
    - **In high transaction system, old SQL cache will be flushed at some point requiring a recompile of dynamic SQL**
- **The DB2 package with pureQuery advantage:**
  - **DB2 has <u>STATIC SQL</u> packages – which can avoid a SQL compile at runtime**
  - **pureQuery can leverage this static SQL for your Java applications**

*** See the DB2 catalog view: syscat.packages**

DB2 "True" Static SQL

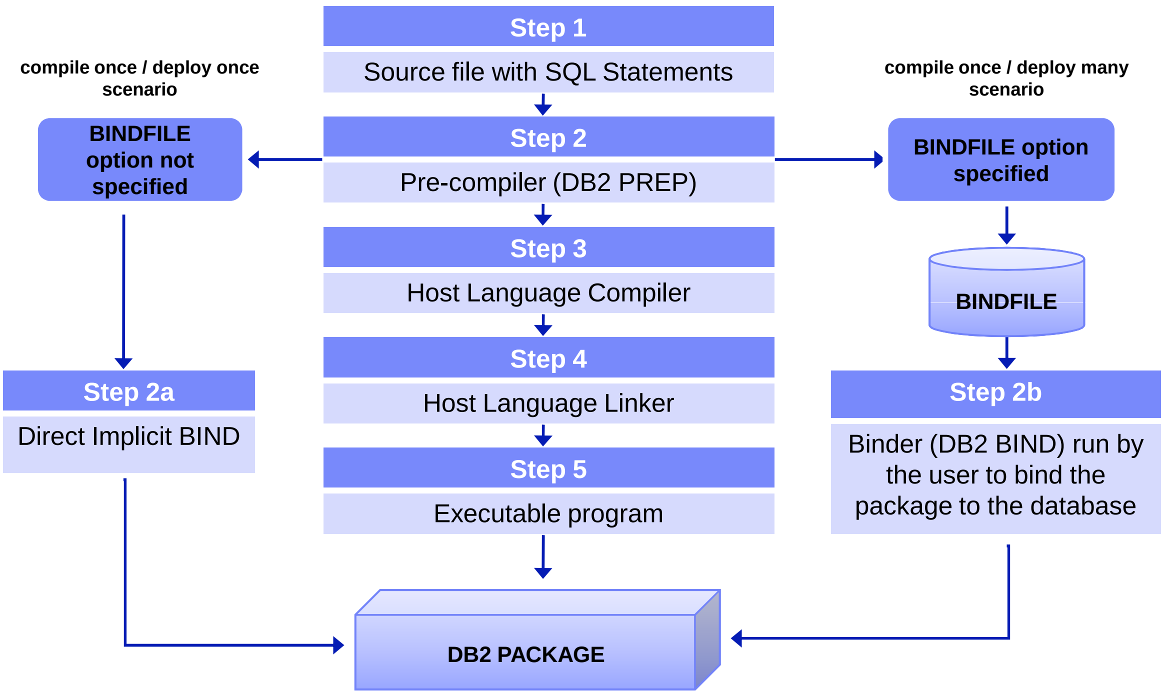
Clarifying The Definition



- 🚫 **DB2 static SQL is not the use of:**
  - **SQL parameter markers or literals - or any other SQL writing technique**
  - **The package (SQL) cache itself**
  - **CURSOR_SHARING=FORCED - or any other SQL cache improvement technique**
- **The above only potentially improves the reusability of SQL**
  - **DBAs & developers may think this is the meaning of "static SQL"**
  - **To DB2, these techniques apply to both static and dynamic SQL in the cache**
    - **To DB2, if the SQL is flushed and has to be recompiled all over again, it's dynamic**
    - **Flushed SQL happens quite often in high transaction systems – no way to prevent it as you can only set your package (SQL) cache so big**
- 👍 **True static SQL in DB2:**
  - **Is SQL with an access path & execution environment determined at preparation time**
    - **NOT at runtime**
  - **Is SQL in DB2 packages - stored as objects in the database itself**
    - **The packages are placed in the package cache and are immediately executable**
    - **Avoids a SQL recompile even if flushed from the package cache**
  - **Is SQL that does not require an authorization check for every object in it**
    - **Only one check on the package for all the objects of all the SQL in that package**

DB2 Package Creation – Traditional Sources

Overview Using Embedded SQLJ, C/C++, COBOL, etc. *



Step 1
Source file with SQL Statements

Step 2
Pre-compiler (DB2 PREP)

Step 3
Host Language Compiler

Step 4
Host Language Linker

Step 5
Executable program

compile once / deploy once scenario

BINDFILE option not specified

Step 2a
Direct Implicit BIND

compile once / deploy many scenario

BINDFILE option specified

BINDFILE

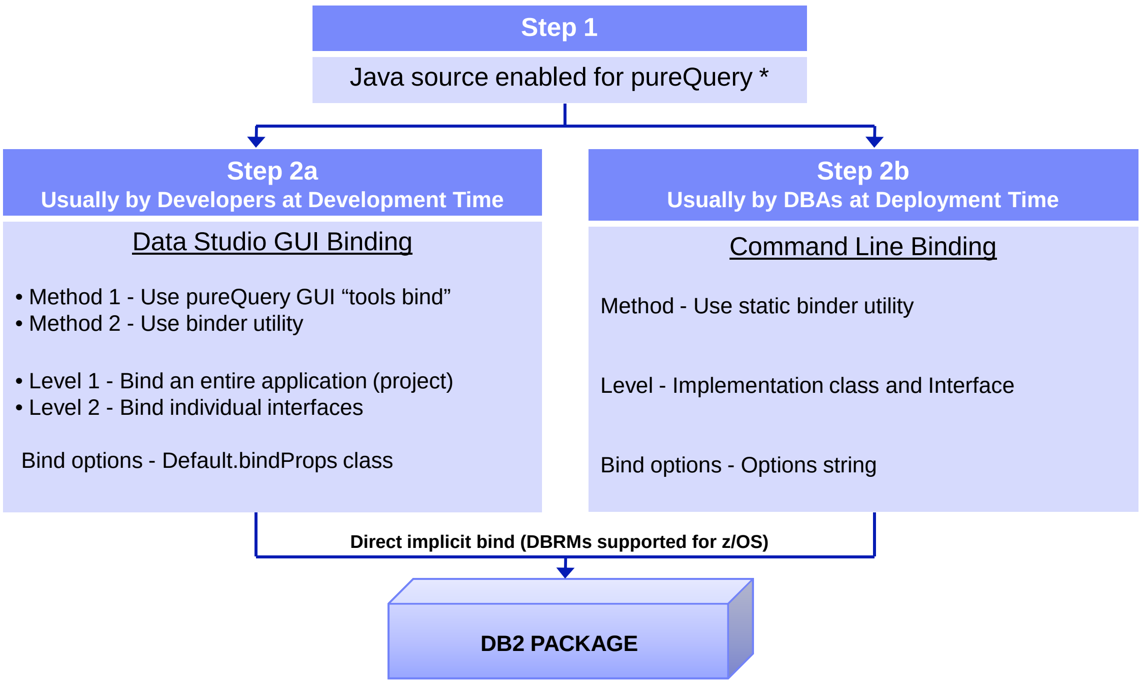
Step 2b
Binder (DB2 BIND) run by the user to bind the package to the database

DB2 PACKAGE

*** DB2 stored procedure creation also produces a DB2 package, but doesn't have these steps and always has a direct implicit bind**

DB2 Package Creation – Java Sources

Overview Using pureQuery for Java Applications



Step 1
Java source enabled for pureQuery *

Step 2a
Usually by Developers at Development Time

Data Studio GUI Binding

- Method 1 - Use pureQuery GUI "tools bind"
- Method 2 - Use binder utility

- Level 1 - Bind an entire application (project)
- Level 2 - Bind individual interfaces

Bind options - Default.bindProps class

Step 2b
Usually by DBAs at Deployment Time

Command Line Binding

Method - Use static binder utility

Level - Implementation class and Interface

Bind options - Options string

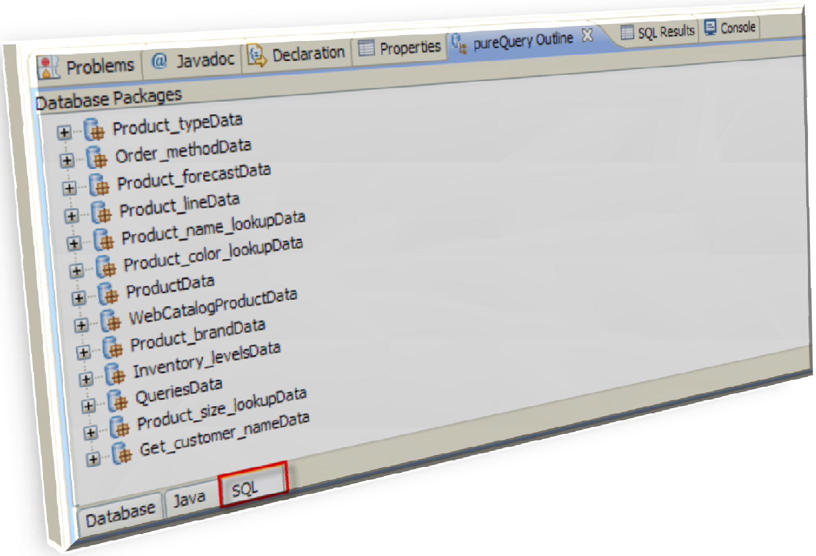
Direct implicit bind (DBRMs supported for z/OS)

DB2 PACKAGE

*** Ask about the PoT: "IBM Data Studio pureQuery for the Application Developer" for more details on this**

pureQuery Runtime

Future DB2 Packages in pureQuery Outline



Database Packages
- Product_typeData
- Order_methodData
- Product_forecastData
- Product_lineData
- Product_name_lookupData
- Product_color_lookupData
- ProductData
- WebCatalogProductData
- Product_brandData
- Inventory_levelsData
- QueriesData
- Product_size_lookupData
- Get_customer_nameData

---

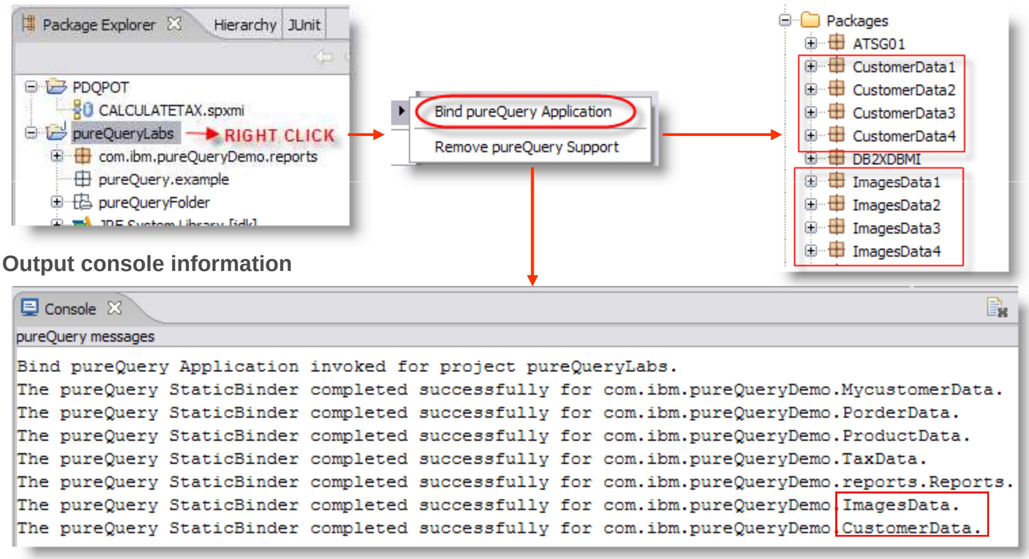
pureQuery Binding

Data Studio GUI Example

- **Use Data Studio to create & bind packages at development time**
  - **4 packages will be created for isolation levels UR,CS,RS & RR - if ISOLATION level not specified**

Data Studio project option

DB2 packages created



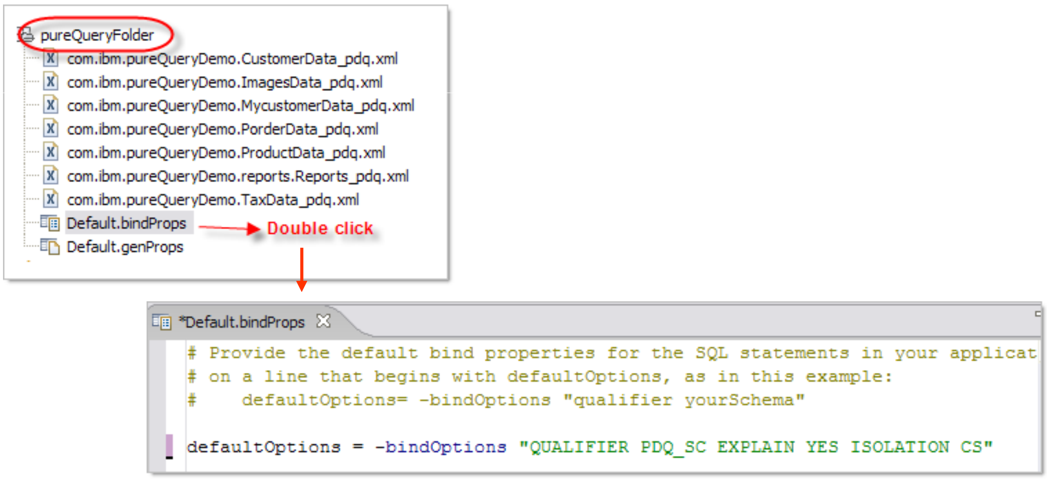
Output console information

```
Bind pureQuery Application invoked for project pureQueryLabs.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.MycustomerData.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.PorderData.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.ProductData.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.TaxData.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.reports.Reports.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.ImagesData.
The pureQuery StaticBinder completed successfully for com.ibm.pureQueryDemo.CustomerData.
```

pureQuery Binding

Setting Data Studio GUI Bind Options

- **Setting Bind Options in Data Studio**
  - **Under your project expand: pureQueryFolder**
  - **Double click file: Default.bindProps**
    - **Enter: `defaultOptions = -bindOptions “option string”`**
    - **Example below shows a qualifier * schema name, explain output saved and isolation set to CS**


pureQueryFolder
com.ibm.pureQueryDemo.CustomerData_pdq.xml
com.ibm.pureQueryDemo.ImagesData_pdq.xml
com.ibm.pureQueryDemo.MycustomerData_pdq.xml
com.ibm.pureQueryDemo.PorderData_pdq.xml
com.ibm.pureQueryDemo.ProductData_pdq.xml
com.ibm.pureQueryDemo.reports.Reports_pdq.xml
com.ibm.pureQueryDemo.TaxData_pdq.xml
Default.bindProps
Default.genProps
Double click
*Default.bindProps

```
# Provide the default bind properties for the SQL statements in your applicat
# on a line that begins with defaultOptions, as in this example:
#    defaultOptions= -bindOptions "qualifier yourSchema"

defaultOptions = -bindOptions "QUALIFIER PDQ_SC EXPLAIN YES ISOLATION CS"
```

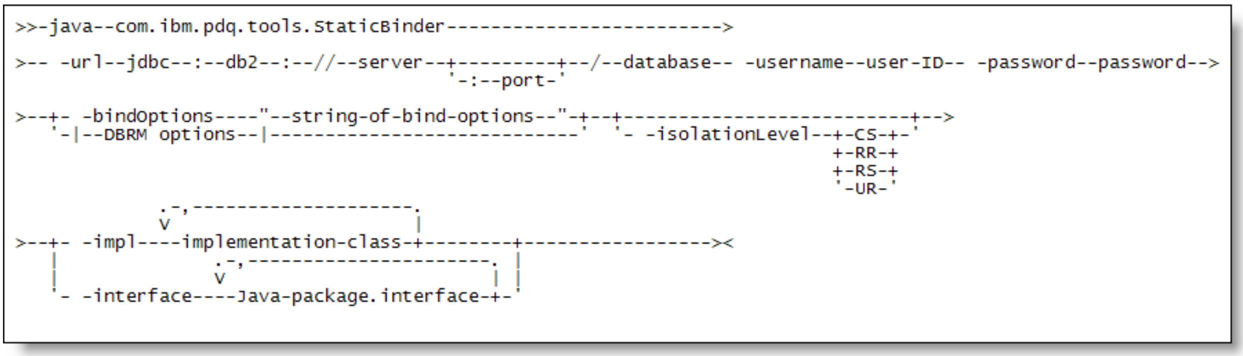
*** “Qualifier” bind option allows SQL to be schema neutral so the schema is defined at bind time.**

---

pureQuery Binding

Static Binder Utility

- **Static Binder Command Line Parameters**
  - **`-url` (Type 4 JCC driver)**
    - **server**
    - **port**
    - **database (Location name for z/OS: `VALUES CURRENT SERVER`)**
  - **`-user`**
  - **`-password`**
  - **`-bindOptions`**
  - **`-impl` (implementation class file name must end with .impl)**



```
>>-java--com.ibm.pdq.tools.StaticBinder------------------------->

>-- -url--jdbc--:--db2--:--//--server--+---------+--/--database-- -username--user-ID-- -password--password-->
                                       '-:--port-'

>--+- -bindOptions----"--string-of-bind-options--"-+--+--------------------------+-->
   '-|--DBRM options--|---------------------------'  '- -isolationLevel--+-CS-+-'
                                                                         +-RR-+
                                                                         +-RS-+
                                                                         '-UR-'

               .-,-----------------.
               V                   |
>--+- -impl----implementation-class-+--------+-----------------><
   |                .-,-------------------.  |
   |                V                     |  |
   '- -interface----Java-package.interface-+-'
```

pureQuery Binding

Bind Options

DB2 LUW		DB2 z/OS	
Parameter	**Values**	**Parameter**	**Values**
BLOCKING	UNAMBIG, ALL, NO	ACTION	REPLACE (REPLVER), ADD
DEGREE	ANY	DBPROTOCOL	DRDA, PRIVATE
EXPLAIN	NO, YES	DEGREE	1, ANY
EXPLSNAP	NO, ALL, YES	EXPLAIN	NO, YES
FEDERATED	NO, YES	IMMEDWRITE	NO, PH1, YES
FUNCPATH	SCHEMA-NAME	ISOLATION	RR, RS, CS, UR
OWNER	AUTH-ID	NOREOPT	VARS
QUALIFIER	QUALIFIER-NAME	REOPT	VARS
INSERT	BUF, DEF	OPTHINT	HINT-ID
ISOLATION	CS, RR, RS, UR	OWNER	AUTH-ID
QUERYOPT	LEVEL	PATH	USER, SCHEMA-NAME
SQLERROR	NPPACKAGE, CONTINUE	QUALIFIER	QUALIFIER-NAME
SQLWARN	YES, NO	RELEASE	COMMIT, DEALLOCATE
STATICREADONLY	NO, YES	SQLERROR	NOPACKAGE, CONTINUE
VALIDATE	RUN, BIND	VALIDATE	RUN, BIND

---

pureQuery Binding

Package Binding Details

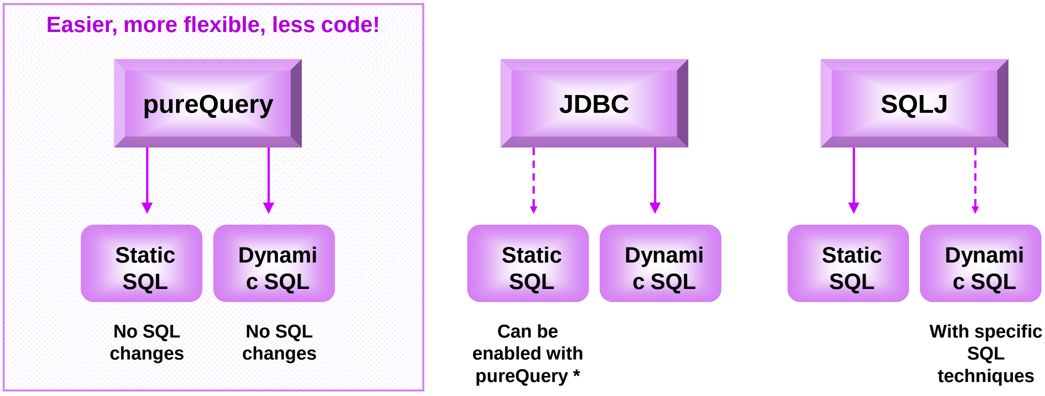
- **Meta Data**
  - **Class file - contains collection, program name, version information, etc.**

- **Packages**
  - **Long package names are used**
  - **Package and Java interface uses a simple one-to-one mapping**
  - **LUW specifics**
    - **Bind file cannot be used (always direct implicit bind)**
    - **Full interface names can be used in the package name**
  - **z/OS specifics**
    - **DBRMs can be used**
    - **Known issues with job control language (JCL), data source name (DSN) are addressed by client tools enhancements**
    - **BIND, REBIND, DROP, FREE commands available from client invocations**

pureQuery SQL Execution Modes

pureQuery vs. JDBC & SQLJ

- **pureQuery runs SQL in static or dynamic modes**
  - **Globally at the JVM™ level or individually at the interface level**
  - **Flexibility exists without changing the SQL coding**
- **JDBC is traditionally for dynamic SQL**
  - **Although pureQuery can support static SQL for JDBC ***
- **SQLJ is traditionally for static SQL**
  - **Although certain SQL coding techniques can force dynamic execution**



Easier, more flexible, less code!

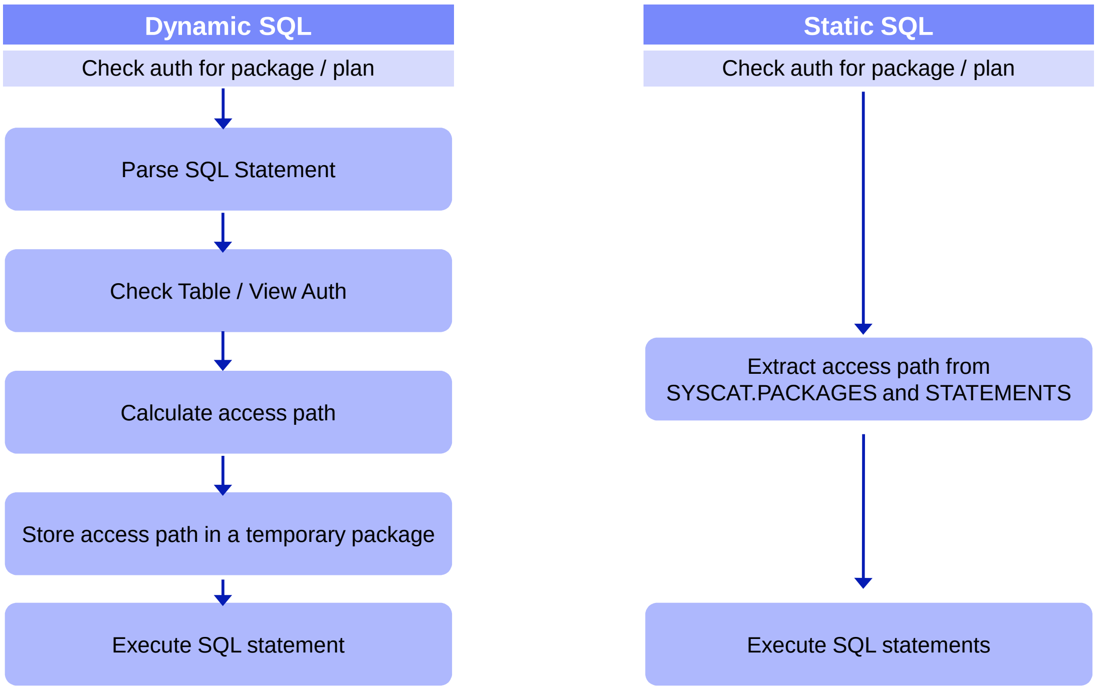
DB2 Static SQL

Comparison vs. Dynamic SQL

Feature	Dynamic SQL (pureQuery, JDBC)	Static SQL (pureQuery, SQLJ)
Performance	Can approach static SQL performance with help from dynamic SQL cache. Cache misses are costly	All SQL parsing, catalog access, done at BIND time. Fully optimized during execution.
Access path reliability	Unpredictable – Any prepare can get a new access path as statistics or host variables change	Guaranteed – locked in at BIND time All SQL available ahead of time for analysis by EXPLAIN.
Authorization	Privileges handled at object level. All users or groups must have direct table privileges – Security exposure, and administrative burden exists.	Privileges are package based. Only administrator needs table access. Users/Groups have execute authority. Prevents non-authorized SQL execution.
Monitoring, Problem determination	Database view is of the JDBC or call level interface (CLI) package – No easy distinction of where any SQL statement came from.	Package view of applications makes it simple to track back to the SQL statement location in the application.
Capacity planning, Forecasting	Difficult to summarize performance data at program level.	Package Level Accounting gives program view of workload to aid accurate forecasting.
Tracking dependent objects	No record of which objects are referenced by a compiled SQL statement	Object dependencies registered in database catalog.

SQL Execution

Dynamic vs. Static



Dynamic SQL

Check auth for package / plan

Parse SQL Statement

Check Table / View Auth

Calculate access path

Store access path in a temporary package

Execute SQL statement

Static SQL

Check auth for package / plan

Extract access path from SYSCAT.PACKAGES and STATEMENTS

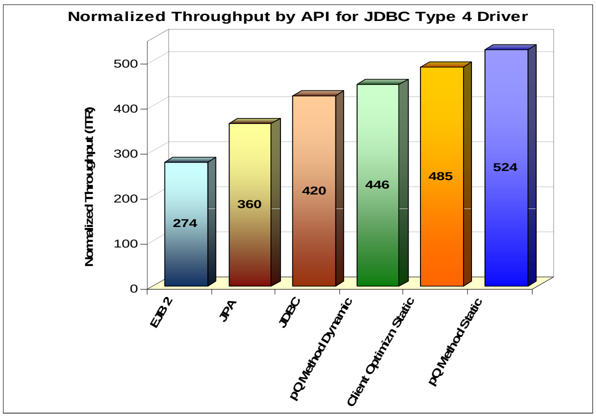
Execute SQL statements

---

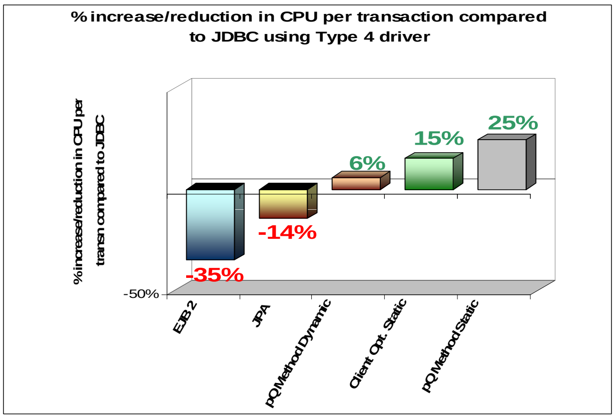
pureQuery for z/OS

Dynamic vs Static (Type-4)

- **In-house testing shows double-digit reduction in CPU costs over dynamic JDBC**



Normalized Throughput by API for JDBC Type 4 Driver



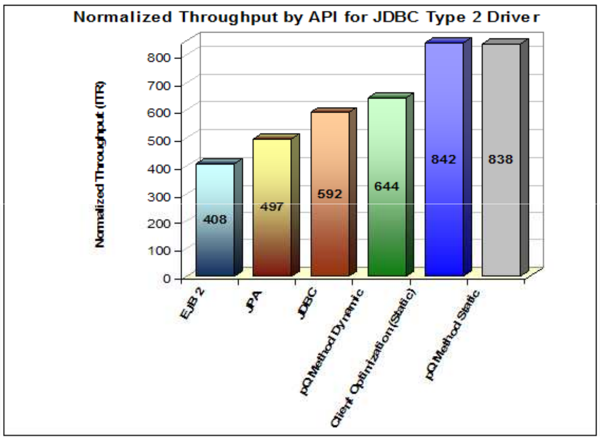
% increase/reduction in CPU per transaction compared to JDBC using Type 4 driver

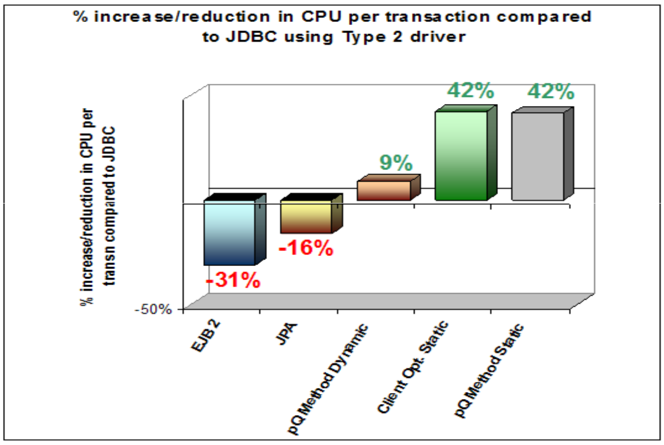
- ❑ IBM Relational Workhouse Workload (IRWW) – an OLTP workload, Type 4 driver
- ❑ Cache hit ratio between 70 and 85%
- ❑ 15% - 25% reduction on CPU per transaction over dynamic JDBC

pureQuery for z/OS

Dynamic vs Static (Type – 2)

- **In-house testing shows double-digit reduction in CPU costs over dynamic JDBC**



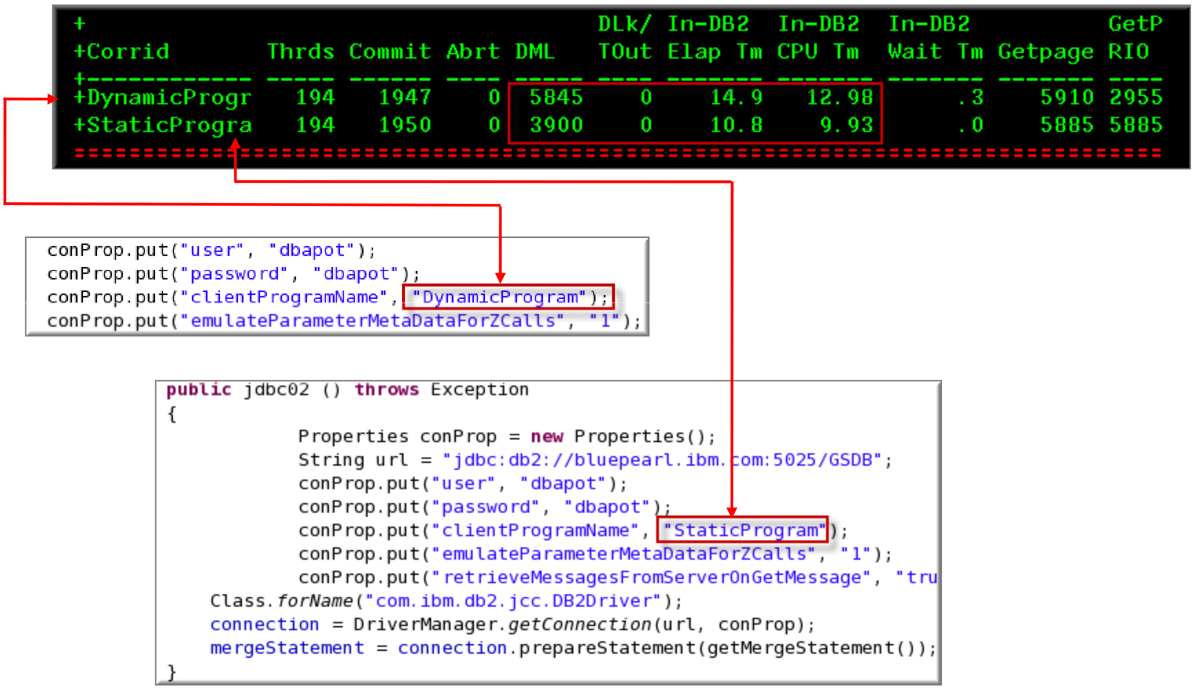


- ❑ IRWW – an OLTP workload, Type 2 driver (Local call)
- ❑ Cache hit ratio between 70 and 85%
- ❑ 42% reduction on CPU per transaction over dynamic JDBC

---

pureQuery for z/OS

Dynamic vs Static – CPU Utilization



```
+                                     DLk/ In-DB2  In-DB2  In-DB2          GetP
+Corrid       Thrds Commit Abrt DML   TOut Elap Tm CPU Tm  Wait Tm Getpage RIO
+------------ ----- ------ ---- ----- ---- ------- ------- ------- ------- ----
+DynamicProgr   194   1947    0  5845    0    14.9   12.98      .3    5910 2955
+StaticProgra   194   1950    0  3900    0    10.8    9.93      .0    5885 5885
===============================================================================
```

```
conProp.put("user", "dbapot");
conProp.put("password", "dbapot");
conProp.put("clientProgramName", "DynamicProgram");
conProp.put("emulateParameterMetaDataForZCalls", "1");
```

```
public jdbc02 () throws Exception
{
            Properties conProp = new Properties();
            String url = "jdbc:db2://bluepearl.ibm.com:5025/GSDB";
            conProp.put("user", "dbapot");
            conProp.put("password", "dbapot");
            conProp.put("clientProgramName", "StaticProgram");
            conProp.put("emulateParameterMetaDataForZCalls", "1");
            conProp.put("retrieveMessagesFromServerOnGetMessage", "tru
    Class.forName("com.ibm.db2.jcc.DB2Driver");
    connection = DriverManager.getConnection(url, conProp);
    mergeStatement = connection.prepareStatement(getMergeStatement());
}
```

Table Privilege Example

Dynamic vs. Static SQL

Dynamic SQL

- ❑ Table privileges must be granted directly to the user, groups or role.

GRANT SELECT ON TABLE EMPLOYEE TO GROUP HR;

EMPNO	FIRSTNME	MIDINIT	LASTNAME
000010	CHRISTNE	I	HAAS
000020	MICHAEL	L	THOMPSON
000030	SALLY	A	KWAN
000040	JOHN	B	GEYER

Static SQL

- ❑ Users require no specific table privileges

GRANT EXECUTE ON PACKAGE EMP_PKG TO GROUP HR;

PKGSCHEMA	PKGNAME	BOUNDBY	BOUNDBYTYPE
VIKRAM	EMP_PKG	BIND_ADM	U
NULLID	SYSSH101	VIKRAM	U

GRANT SELECT ON TABLE EMPLOYEE TO USER BIND_ADM;

EMPNO	FIRSTNME	MIDINIT	LASTNAME
000010	CHRISTNE	I	HAAS
000020	MICHAEL	L	THOMPSON
000030	SALLY	A	KWAN
000040	JOHN	B	GEYER

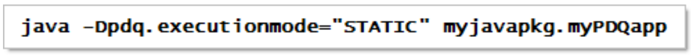
---

pureQuery SQL Execution Modes

Enabling Static SQL

- ❑ **At global (JVM) level**

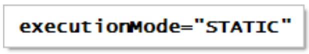
1. Set JVM Property when invoking JRE



```
java -Dpdq.executionmode="STATIC" myjavapkg.myPDQapp
```

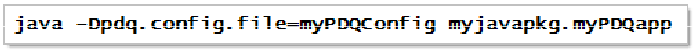
OR

2. Create a configuration file containing executionMode parameter - put it in the classpath



```
executionMode="STATIC"
```

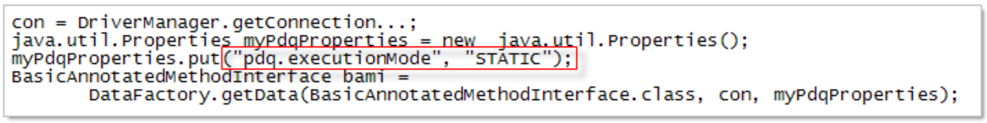
Specify configuration file through JVM property when invoking JRE



```
java -Dpdq.config.file=myPDQConfig myjavapkg.myPDQapp
```

- ❑ **At Interface level**

Pass a property object to the DataFactory when creating an implementation of the interface - specify executionMode="STATIC"



```
con = DriverManager.getConnection...;
java.util.Properties myPdqProperties = new  java.util.Properties();
myPdqProperties.put("pdq.executionMode", "STATIC");
BasicAnnotatedMethodInterface bami =
        DataFactory.getData(BasicAnnotatedMethodInterface.class, con, myPdqProperties);
```

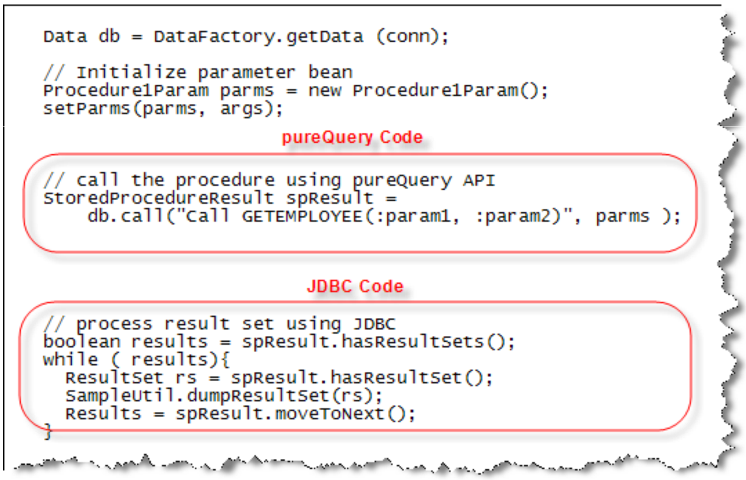
- ❑ **PDQ.JAR, PDQMGMT.JAR and License JAR files**
  - ➢ **Required at runtime for APIs by the application server**
  - ➢ **Required at runtime for APIs by the stand-alone program**

pureQuery Runtime API

With JDBC Code

- **pureQuery runtime can work with any database that has a JDBC driver**
- **Mix JDBC code with pureQuery API**

Sample JAVA program



```
Data db = DataFactory.getData (conn);

// Initialize parameter bean
Procedure1Param parms = new Procedure1Param();
setParms(parms, args);
```

pureQuery Code

```
// call the procedure using pureQuery API
StoredProcedureResult spResult =
    db.call("call GETEMPLOYEE(:param1, :param2)", parms );
```

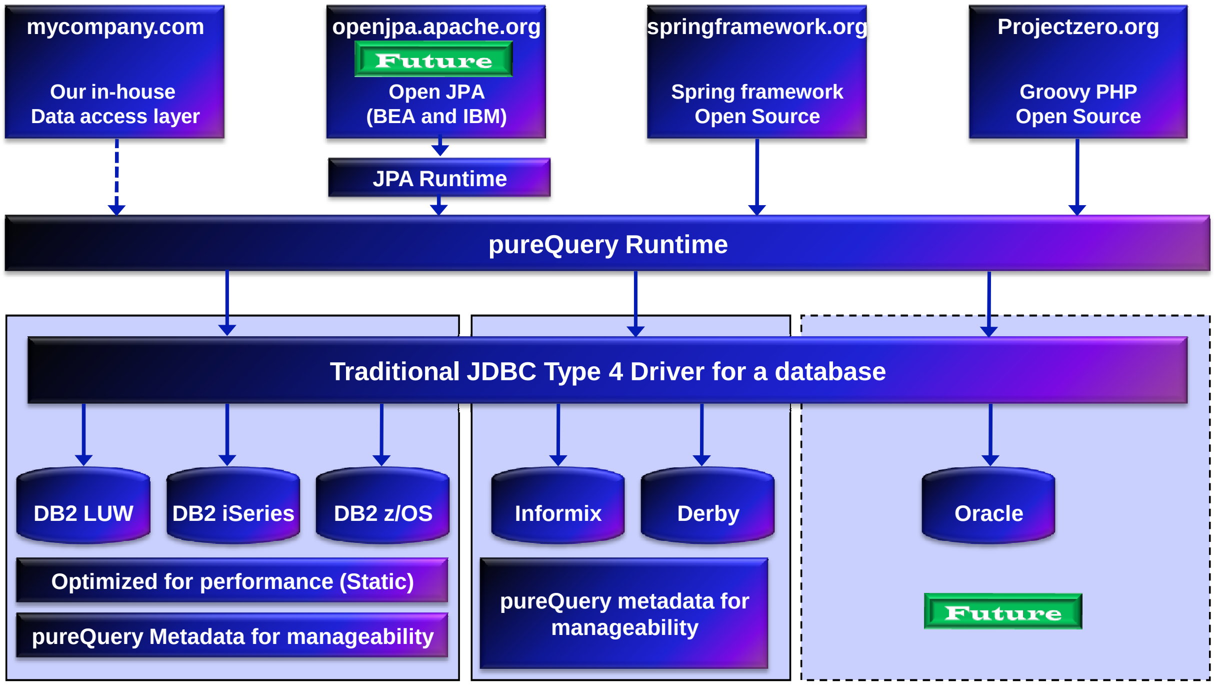
JDBC Code

```
// process result set using JDBC
boolean results = spResult.hasResultSets();
while ( results){
  ResultSet rs = spResult.hasResultSet();
  SampleUtil.dumpResultSet(rs);
  Results = spResult.moveToNext();
}
```

---

pureQuery Runtime Capabilities

Database Agnostic Ability



mycompany.com
Our in-house
Data access layer

openjpa.apache.org
Future
Open JPA
(BEA and IBM)

JPA Runtime

springframework.org
Spring framework
Open Source

Projectzero.org
Groovy PHP
Open Source

pureQuery Runtime

Traditional JDBC Type 4 Driver for a database

DB2 LUW

DB2 iSeries

DB2 z/OS

Optimized for performance (Static)

pureQuery Metadata for manageability

Informix

Derby

pureQuery metadata for manageability

Oracle

Future

pureQuery Explain

Lab 06 pureQuery Explain

An IBM Proof of Technology

ON DEMAND BUSINESS™

9 slides

Explain Facilities

GUI and Command Line

- **Explain Tables**
  - GUI tools like Data Studio or Control Center create them for you
  - You can create them by running EXPLAIN.DDL ,**or**
  - Run **`CALL SYSPROC.SYSINSTALLOBJECTS('EXPLAIN','C',NULL,CURRENT SCHEMA)`**
- **GUI Tools**
  - Visual Explain for a SQL from Data Studio or from Control Center or Command Editor
  - Visual Explain for all SQLs in a Stored Procedure from Control Center
- **Command Line Tools (Mostly for DBAs)**
  - db2expln
  - db2exfmt

```
C:\> db2expln -d SAMPLE -c %USERNAME% -p % -s 0 -graph -i -o explain_output.txt

C:\> db2 delete from EXPLAIN_INSTANCE
C:\> db2 rebind package ‘package_name’
C:\> db2exfmt -D SAMPLE -O explain.fmt -1

C:\> db2 delete from EXPLAIN_INSTANCE
C:\> db2 explain plan with snapshot for <SQL Query>
C:\> db2exfmt –D SAMPLE –O explain.fmt -1
```

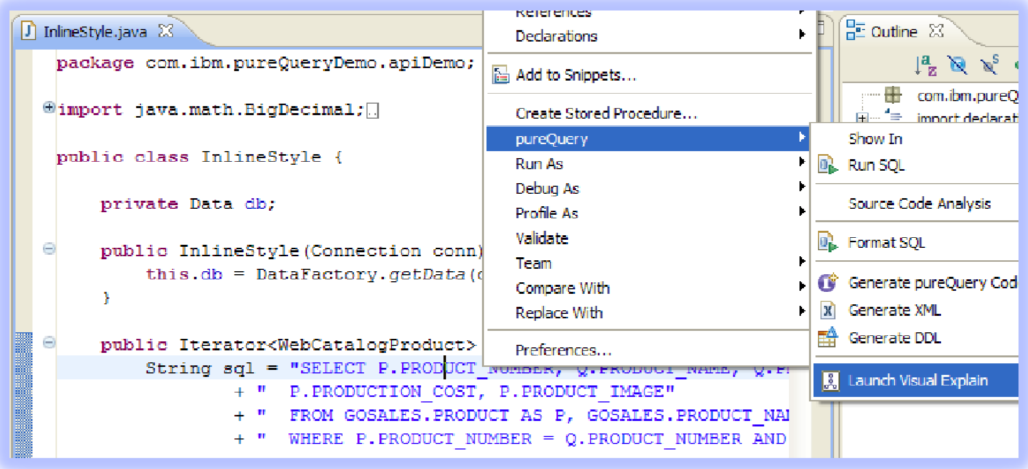
After rebind of package, run explain plan using db2exfmt

Example for creating explain plan for a dynamic SQL using db2exfmt

Explain Facilities

Data Studio

- **Check explain plan as you develop**
- **Right click on SQL in Java program**
  - Select `pureQuery>Launch Visual Explain`

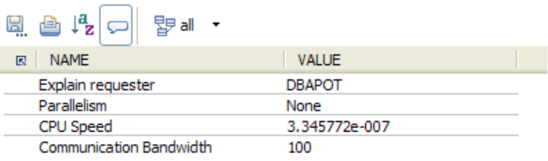


```
package com.ibm.pureQueryDemo.apiDemo;

import java.math.BigDecimal;

public class InlineStyle {

    private Data db;

    public InlineStyle(Connection conn)
        this.db = DataFactory.getData(c
    }

    public Iterator<WebCatalogProduct>
        String sql = "SELECT P.PRODUCT_NUMBER, Q.PRODUCT_NAME, Q.P
                + "  P.PRODUCTION_COST, P.PRODUCT_IMAGE"
                + "  FROM GOSALES.PRODUCT AS P, GOSALES.PRODUCT_NAI
                + "  WHERE P.PRODUCT_NUMBER = Q.PRODUCT_NUMBER AND
```

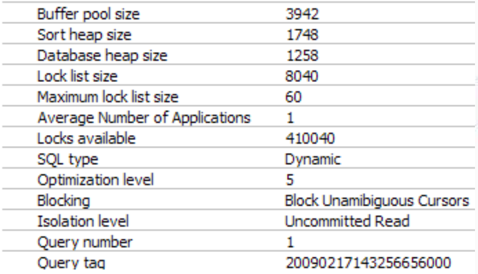
Explain Facilities

Data Studio

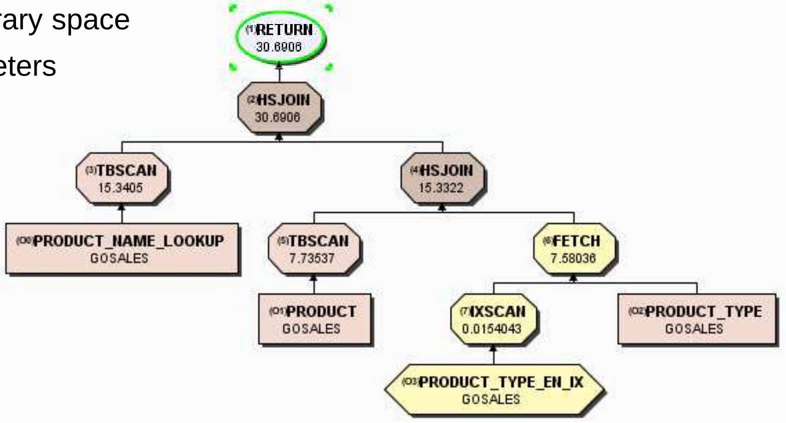
- Analyze explain plan
  - Identify bottleneck by looking high timerons on nodes
  - Check for full table scans
  - Different sort mechanisms
  - Sort spilling to temporary space
- Check optimization parameters



NAME	VALUE
Explain requester	DBAPOT
Parallelism	None
CPU Speed	3.345772e-007
Communication Bandwidth	100



Buffer pool size	3942
Sort heap size	1748
Database heap size	1258
Lock list size	8040
Maximum lock list size	60
Average Number of Applications	1
Locks available	410040
SQL type	Dynamic
Optimization level	5
Blocking	Block Unamibiguous Cursors
Isolation level	Uncommitted Read
Query number	1
Query tag	20090217143256656000



Explain Facilities

Use of Parameter markers - Dilemma

Use literals in SQL	Use parameter markers in SQL
👍 Get an optimum plan	👍 May get an optimum or sub-optimal plan
👍 Selection of right indexes based on actual value of literals.	👍 May use indexes when it is not required or may use FT scan when index can be used
☒ Pay cost of compiling similar SQL statements if literals are not same	👍 Compile SQL only once and use again and again
☒ May prove to be heavy burden in long run	👍 May not be very effective to meet individual query goals
👍 **Fortunately, there is a meet-in-the middle effective approach in DB2 by using REOPT parameter**	
👍 Use parameter markers to avoid huge cost of compilation	
👍 REOPT allows DB2 to peek at actual value of parameter, special registers or host variables at run time and evaluate access path	

Explain Facilities

Use of Parameter markers – Use REOPT

- **Create NULLIDR1 and NULLIDRA collections**
  - `db2 bind db2clipk.bnd collection NULLIDR1`
  - `db2 bind db2clipk.bnd collection NULLIDRA`
- **Set proper property in pureQuery for dynamic SQL**
  - Set **`jdbcCollection=NULLIDR1`** for REOPT=ONCE
  - Set **`jdbcCollection=NULLIDRA`** for REOPT=ALWAYS
  - **`REOPT=None`** (Default)
  - **`REOPT=ONCE`** – Get access path only once from actual parameters
  - **`REOPT=ALWAYS`** – Get access path every time using actual parameters
- **Set proper binding property in pureQuery for static SQL**
  - **`bindOptions=REOPT ONCE or`**
  - **`bindOptions=REOPT ALWAYS`**

Static SQL – use REOPT in bindOptions

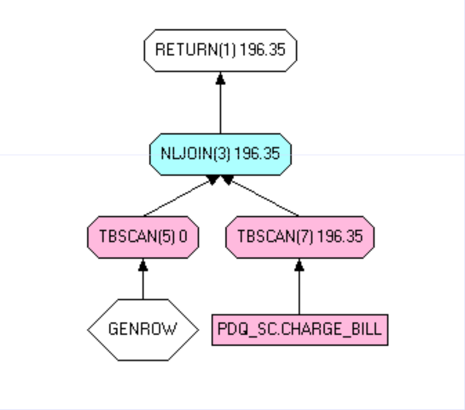
Dynamic SQL – use jdbcCollection connection property to set NULLIDR1 or NULLIDRA

Explain Facilities

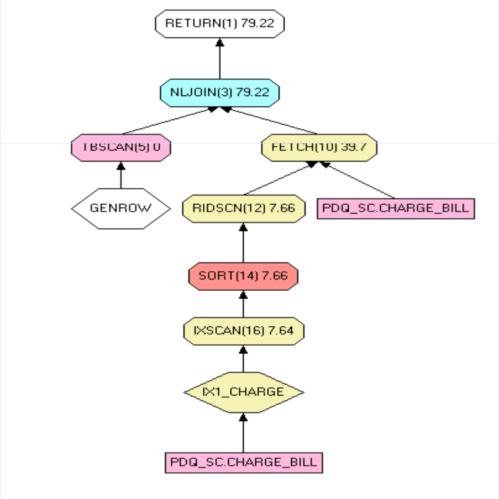
Parameter markers

`SELECT BILL_GK, BILL_CHARGE_IND FROM PDQ_SC.CHARGE_BILL WHERE RATE_PRICE BETWEEN ? AND ?`

Plan - no index on rate_price



Plan - index on rate_price

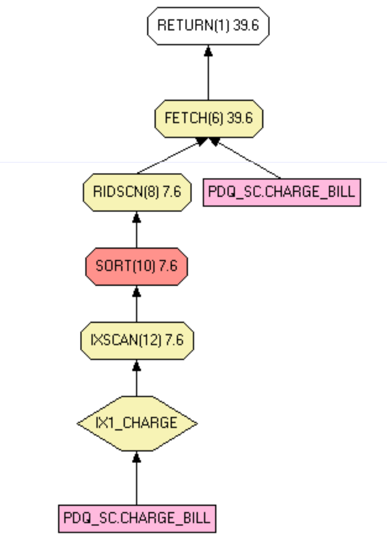


Explain Facilities

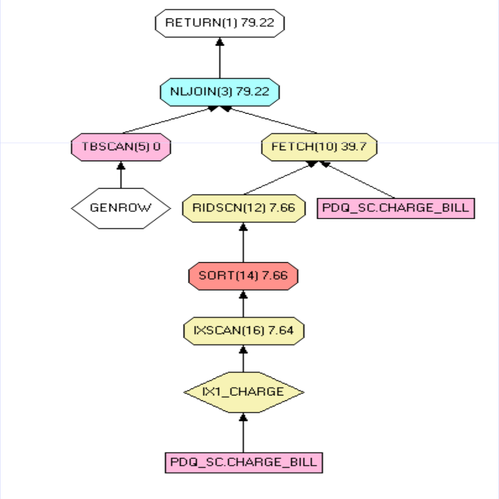
Literals – Parameter Markers

`SELECT BILL_GK, BILL_CHARGE_IND FROM PDQ_SC.CHARGE_BILL WHERE RATE_PRICE BETWEEN DOUBLE(100) AND DOUBLE(200)`

Literals – Explain Plan



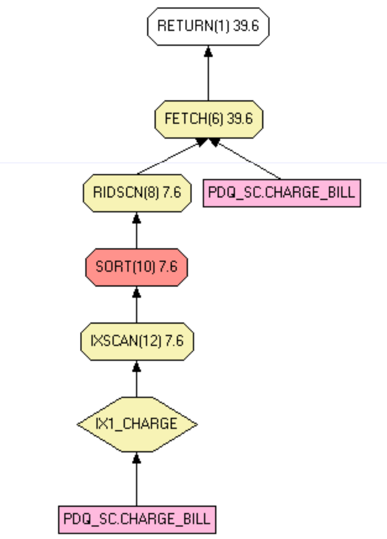
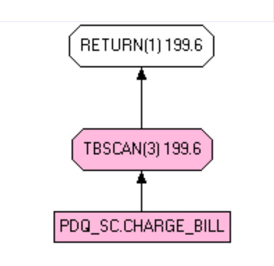
Parameter markers – Explain Plan



Explain Facilities

Literals – Different values

`SELECT BILL_GK, BILL_CHARGE_IND FROM PDQ_SC.CHARGE_BILL WHERE RATE_PRICE BETWEEN DOUBLE(100) AND DOUBLE(200)`

Literals – Between 100 and 200	**Literals – Between 2000 and 90000**
	

IBM Software Group

Optimize JDBC Applications with pureQuery

Lab 07 pureQuery for JDBC Applications

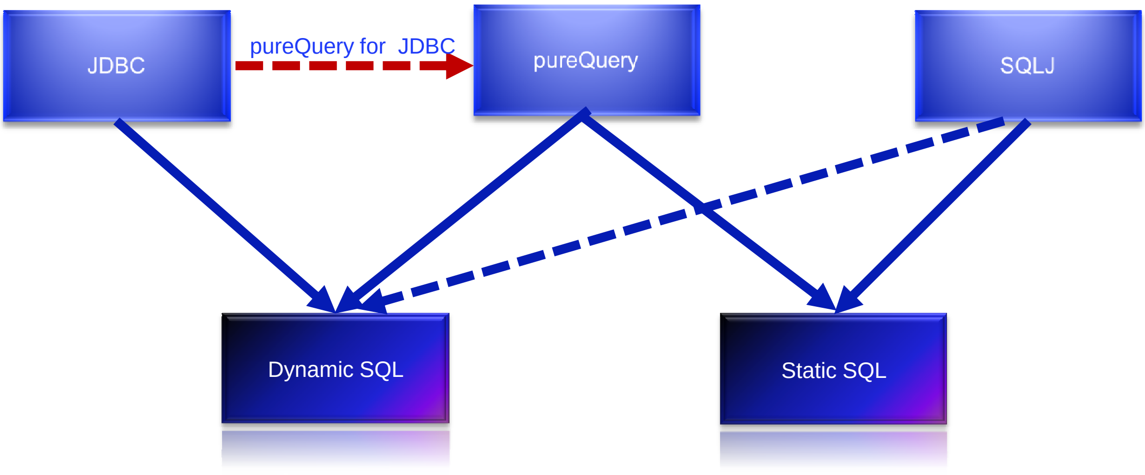
An IBM Proof of Technology

ON DEMAND BUSINESS™

16 slides

pureQuery Runtime
JDBC and SQLJ compared with pureQuery

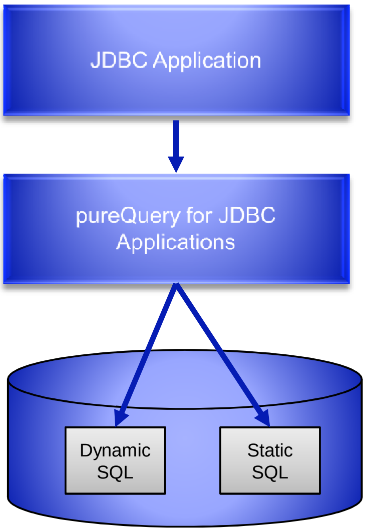
- Flexibility with pureQuery



---

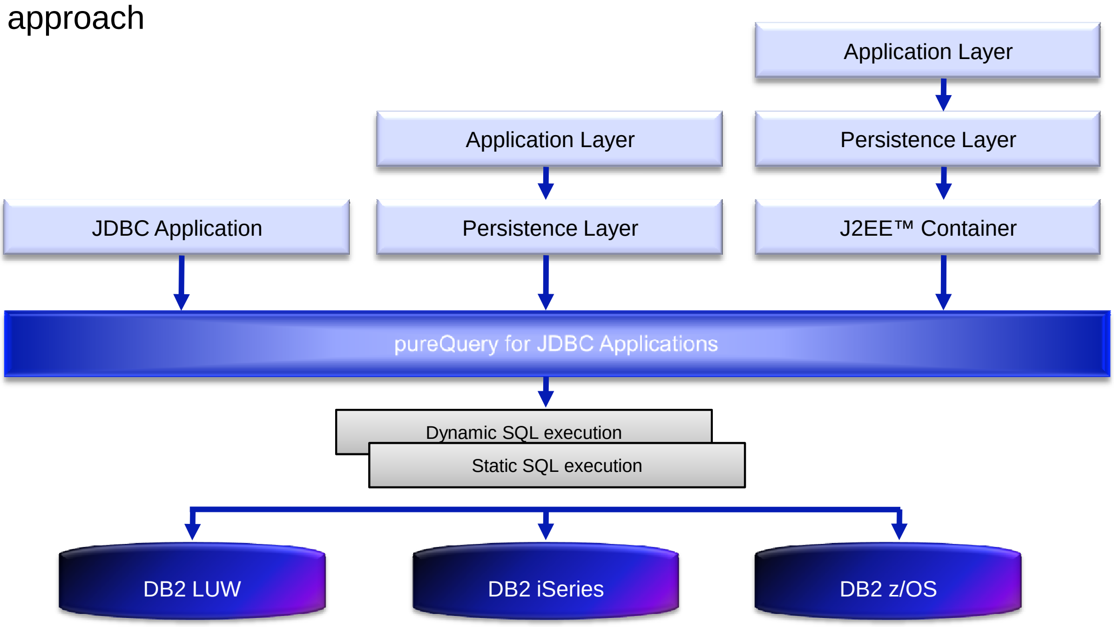
pureQuery for JDBC
Overview

- pureQuery for JDBC application enables static execution of SQL without any code change



pureQuery for JDBC
Overview (continued …)

- pureQuery for JDBC application can be used regardless of persistence approach

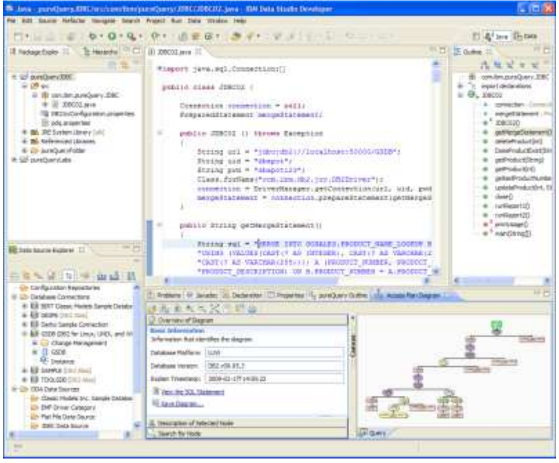

Application Layer
Persistence Layer
J2EE™ Container
Application Layer
Persistence Layer
JDBC Application
pureQuery for JDBC Applications
Dynamic SQL execution
Static SQL execution
DB2 LUW
DB2 iSeries
DB2 z/OS

---

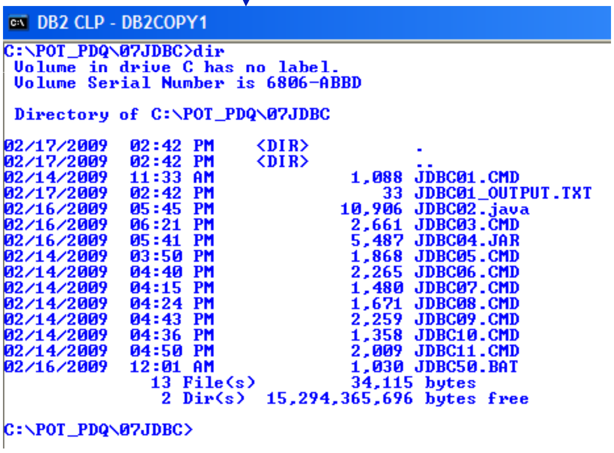
pureQuery for JDBC
Components

- IBM Data Studio Developer edition for applications having source
- IBM pureQuery Runtime for applications with no source
- IBM JCC Driver 3.52+

Application with source



Application with no source

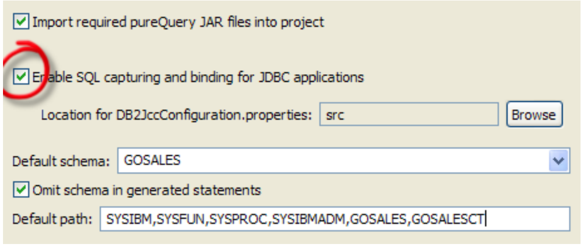


```
C:\POT_PDQ\07JDBC>dir
 Volume in drive C has no label.
 Volume Serial Number is 6806-ABBD

 Directory of C:\POT_PDQ\07JDBC

02/17/2009  02:42 PM    <DIR>          .
02/17/2009  02:42 PM    <DIR>          ..
02/14/2009  11:33 AM             1,088 JDBC01.CMD
02/17/2009  02:42 PM                33 JDBC01_OUTPUT.TXT
02/16/2009  05:45 PM            10,906 JDBC02.java
02/16/2009  06:21 PM             2,661 JDBC03.CMD
02/16/2009  05:41 PM             5,487 JDBC04.JAR
02/14/2009  03:50 PM             1,868 JDBC05.CMD
02/14/2009  04:40 PM             2,265 JDBC06.CMD
02/14/2009  04:15 PM             1,480 JDBC07.CMD
02/14/2009  04:24 PM             1,671 JDBC08.CMD
02/14/2009  04:43 PM             2,259 JDBC09.CMD
02/14/2009  04:36 PM             1,358 JDBC10.CMD
02/14/2009  04:50 PM             2,009 JDBC11.CMD
02/16/2009  12:01 AM             1,030 JDBC50.BAT
              13 File(s)         34,115 bytes
               2 Dir(s)  15,294,365,696 bytes free

C:\POT_PDQ\07JDBC>
```

pureQuery for JDBC

Process from dynamic to static

- **Capture**
  - Record information about successfully executed SQL statements
- **Configure**
  - Enrich the captured information in preparation of the bind utility
- **Bind**
  - Generate DBRM (DB2 on z/OS) or Packages (DB2 LUW)
- **Execute**
  - Driver determines the execution mode for each SQL statement and, if possible runs in that mode



☑ Import required pureQuery JAR files into project

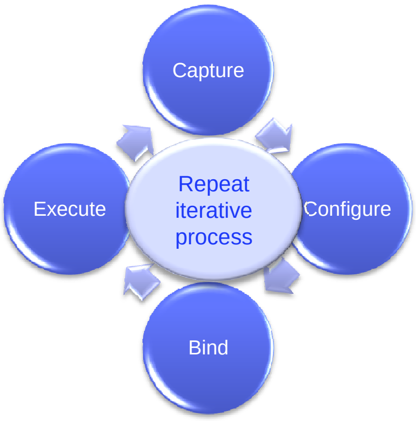
☑ Enable SQL capturing and binding for JDBC applications

Location for DB2JccConfiguration.properties: src

Default schema: GOSALES

☑ Omit schema in generated statements

Default path: SYSIBM,SYSFUN,SYSPROC,SYSIBMADM,GOSALES,GOSALESCT



---

pureQuery for JDBC

Capture

- **Enable JCC driver's capture mode in** `pdq.properties`

```
pdq.captureMode=ON
pdq.executionMode=DYNAMIC
pdq.pureQueryXml=pureQueryFolder/capture.pdqxml
pdq.stackTraceDepth=-1
```

- **Capture mode can be defined at**
  - Through `DB2JccConfiguration.properties` **or** `pdq.properties`
  - Connection
  - JVM
  - Data Source
- **Incremental process**
  - Performed as a part of the test cycle
- **Inspect capture log**
  - To detect potential problems

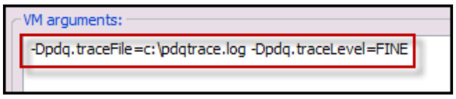
pureQuery for JDBC

Capture (continued …)

- **Capture settings**

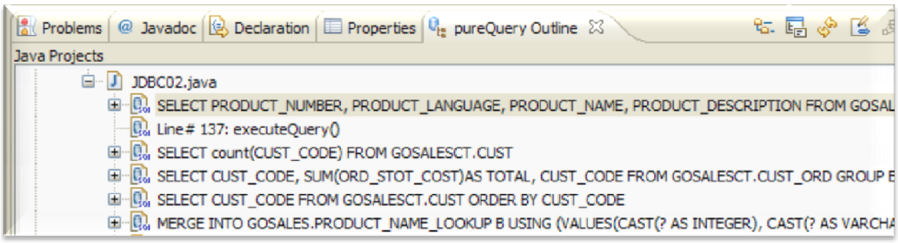
Enable capture	`captureMode(ON)`
Disable capture	`captureMode(OFF)`
Execute SQL dynamic	`executionMode(DYNAMIC)`
Execute SQL static	`executionMode(STATIC)`
Metafile	`pureQueryXML(path/to/metadata)`

- **Enable pureQuery logging**



```
-Dpdq.traceFile=c:\pdqtrace.log -Dpdq.traceLevel=FINE
```

- **Browse captured metadata – pureQuery outlive view**



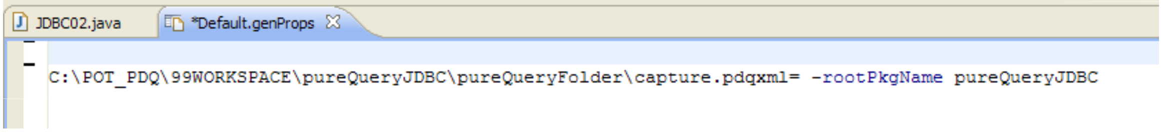
```
SELECT PRODUCT_NUMBER, PRODUCT_LANGUAGE, PRODUCT_NAME, PRODUCT_DESCRIPTION FROM GOSAL
Line# 137: executeQuery()
SELECT count(CUST_CODE) FROM GOSALESCT.CUST
SELECT CUST_CODE, SUM(ORD_STOT_COST)AS TOTAL, CUST_CODE FROM GOSALESCT.CUST_ORD GROUP E
SELECT CUST_CODE FROM GOSALESCT.CUST ORDER BY CUST_CODE
MERGE INTO GOSALES.PRODUCT_NAME_LOOKUP B USING (VALUES(CAST(? AS INTEGER), CAST(? AS VARCHA
```

---

pureQuery for JDBC

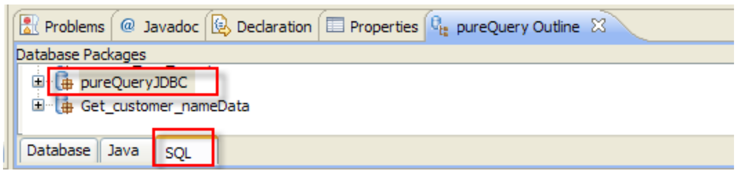
Configure

- **Enrich capture metadata in preparation of bind**
  - Collection name
  - Package base name
  - Maximum statements per package
- **Data Studio - specify parameters in `Default.genProps`**



```
C:\POT_PDQ\99WORKSPACE\pureQueryJDBC\pureQueryFolder\capture.pdqxml= -rootPkgName pureQueryJDBC
```

- **pureQuery outline view – Package preview**



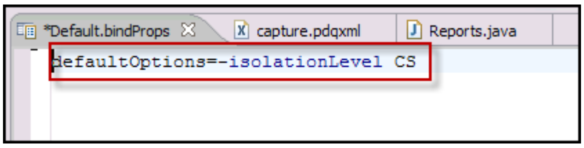
- **Command line – use `Configure` command**

```
"%JAVA_HOME%\bin\java" com.ibm.pdq.tools.Configure ^
                       -pureQueryXml %CD%\capture.pdqxml ^
                       -rootPkgName pureQueryJDBC -collection PDQCOL
```

pureQuery for JDBC

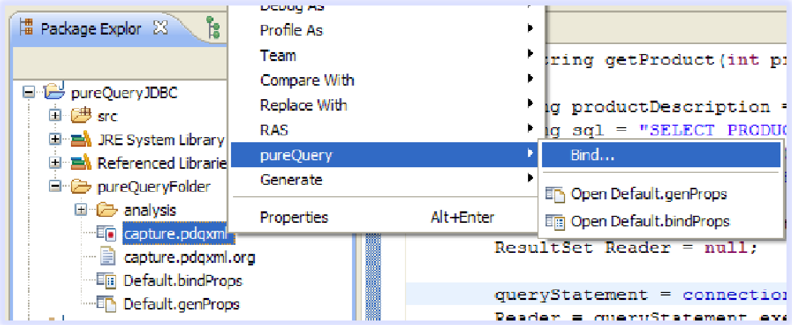
Bind – Data Studio

- **Specify BIND options**



```
defaultOptions=-isolationLevel CS
```

- **Bind – Captured SQL**



---

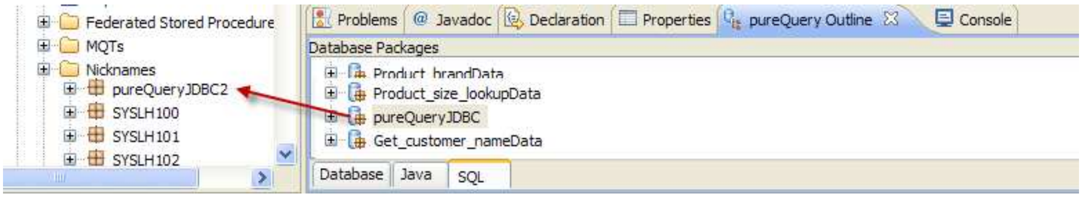
pureQuery for JDBC

StaticBinder

- **BIND through command line**

```
"%JAVA_HOME%\bin\java" com.ibm.pdq.tools.StaticBinder
                       -pureQueryXml %CD%\capture.pdqxml
                       -user DBAPOT
                       -password dbapot123
                       -url "jdbc:db2://localhost:50000/GSDB"
                       -isolationLevel CS
```

- Use Data Source Explorer to verify packages



pureQuery for JDBC

Execution

- **Test**
  - **Dynamic execution and capture SQL incrementally**

Incremental capture	`captureMode(ON)`
Execute SQL	`executionMode(DYNAMIC)`
Allow dynamic SQL	`allowDynamicSQL(TRUE)`

  - **Static execution – mixed mode**

Incremental capture	`captureMode(ON)`
Execute SQL	`executionMode(STATIC)`
Allow dynamic SQL	`allowDynamicSQL(TRUE)`

  - **Static execution – exception mode**

Incremental capture	`captureMode(ON)`
Execute SQL	`executionMode(STATIC)`
Allow dynamic SQL	`allowDynamicSQL(FALSE)`

- **Production**
  - **Static execution – mixed mode**

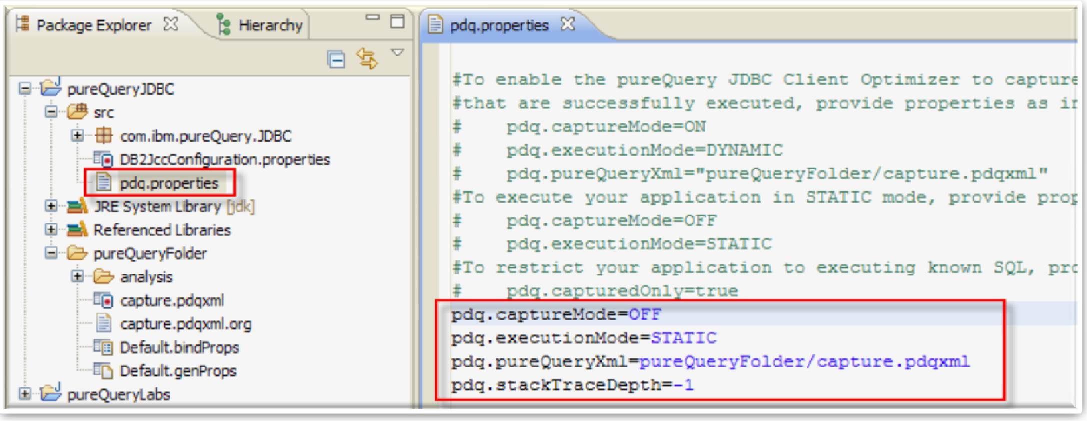
Disable capture	`captureMode(OFF)`
Execute SQL static	`executionMode(STATIC)`

---

pureQuery for JDBC

Execution

- **Data Studio**



```
#To enable the pureQuery JDBC Client Optimizer to capture
#that are successfully executed, provide properties as i
#     pdq.captureMode=ON
#     pdq.executionMode=DYNAMIC
#     pdq.pureQueryXml="pureQueryFolder/capture.pdqxml"
#To execute your application in STATIC mode, provide prop
#     pdq.captureMode=OFF
#     pdq.executionMode=STATIC
#To restrict your application to executing known SQL, pro
#     pdq.capturedOnly=true
pdq.captureMode=OFF
pdq.executionMode=STATIC
pdq.pureQueryXml=pureQueryFolder/capture.pdqxml
pdq.stackTraceDepth=-1
```

- **Command line**

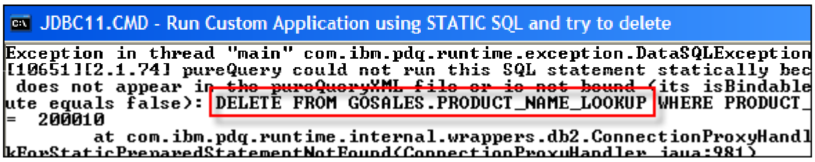
```
SET CAPTURE=-Ddb2.jcc.pdqProperties=captureMode(ON),
SET CAPTURE=%CAPTURE%executionMode(DYNAMIC),
SET CAPTURE=%CAPTURE%pureQueryXml(%CD%\capture.pdqxml),
SET CAPTURE=%CAPTURE%allowDynamicSQL(true)
```

pureQuery for JDBC

Execution – Exception

- **Exception for not yet captured SQL statements**
  - **Dynamic SQL are not allowed**

Disable capture	`captureMode(OFF)`
Execute SQL	`executionMode(STATIC)`
Allow dynamic SQL	`allowDynamicSQL(FALSE)`



JDBC11.CMD - Run Custom Application using STATIC SQL and try to delete

```
Exception in thread "main" com.ibm.pdq.runtime.exception.DataSQLException
[10651][2.1.74] pureQuery could not run this SQL statement statically bec
 does not appear in the pureQueryXML file or is not bound (its isBindable
ute equals false): DELETE FROM GOSALES.PRODUCT_NAME_LOOKUP WHERE PRODUCT_
=  200010
        at com.ibm.pdq.runtime.internal.wrappers.db2.ConnectionProxyHandl
kForStaticPreparedStatementNotFound(ConnectionProxyHandler.java:981)
```

- **For no exception**

Allow dynamic SQL	`allowDynamicSQL(TRUE`

---

pureQuery for JDBC

Enablement for Hibernate Applications

- **Use proper options in *each* data source**

```
<xa-datasource>
    <jndi-name>DSIDataSource</jndi-name>
    <track-connection-by-tx/>
    <isSameRM-override-value>false</isSameRM-override-value>
    <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-datasource-class>
    <xa-datasource-property name="ServerName">172.22.200.32</xa-datasource-property>
    <xa-datasource-property name="User">db2inst1</xa-datasource-property>
    <xa-datasource-property name="Password">sles10</xa-datasource-property>
    <xa-datasource-property name="DatabaseName">NRADC</xa-datasource-property>
    <xa-datasource-property name="PortNumber">50001</xa-datasource-property>
    <xa-datasource-property name="DriverType">4</xa-datasource-property>
    <xa-datasource-property name="CurrentSchema">NRADC_PROD</xa-datasource-property>
    <xa-datasource-property
name="PdqProperties">captureMode(OFF),executionMode(STATIC),pureQueryXml(nradc_prod_1.pdqxml),
allowDynamicSQL(true)</xa-datasource-property>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.DB2ExceptionSorter</exception-sorter-class-name>
    <no-tx-separate-pools/>
      <metadata>
         <type-mapping>DB2</type-mapping>
      </metadata>
  </xa-datasource>
.
.
</datasources>
```

- **Use proper dialect**

```
<attribute name="Dialect">net.sf.hibernate.dialect.DB2Dialect</attribute>
```

pureQuery for JDBC
Enablement for Hibernate Applications

- **Follow capture, configure, bind and Run**
  - **Capture – Modify properties in driver source XML file**

```
captureMode(ON),executionMode(DYNAMIC),pureQueryXml(nradc_prod_1.pdqxml),
allowDynamicSQL(true)
```

  - **Configure**

```
"%JAVA_HOME%\bin\java" com.ibm.pdq.tools.Configure ^
                       -pureQueryXml %CAPTXML%\nradc_prod_1.pdqxml ^
                       -rootPkgName OMSAPP1 -collection NRADC_PROD_PKG
```

  - **Bind**

```
"%JAVA_HOME%\bin\java" com.ibm.pdq.tools.StaticBinder ^
                       -pureQueryXml %CAPTXML%\nradc_prod_1.pdqxml ^
                       -user db2inst1 ^
                       -password sles10 ^
                       -url "jdbc:db2://172.22.200.32:50001/NRADC" ^
                       -isolationLevel CS ^
                       -bindOptions "QUALIFIER NRADC_PROD"
```

  - **Run - Modify properties in driver source XML file**

```
captureMode(OFF),executionMode(STATIC),pureQueryXml(nradc_prod_1.pdqxml),
allowDynamicSQL(true)
```

IBM Software Group

pureQuery Advanced Concepts

Lab 08 pureQuery Advanced Concepts

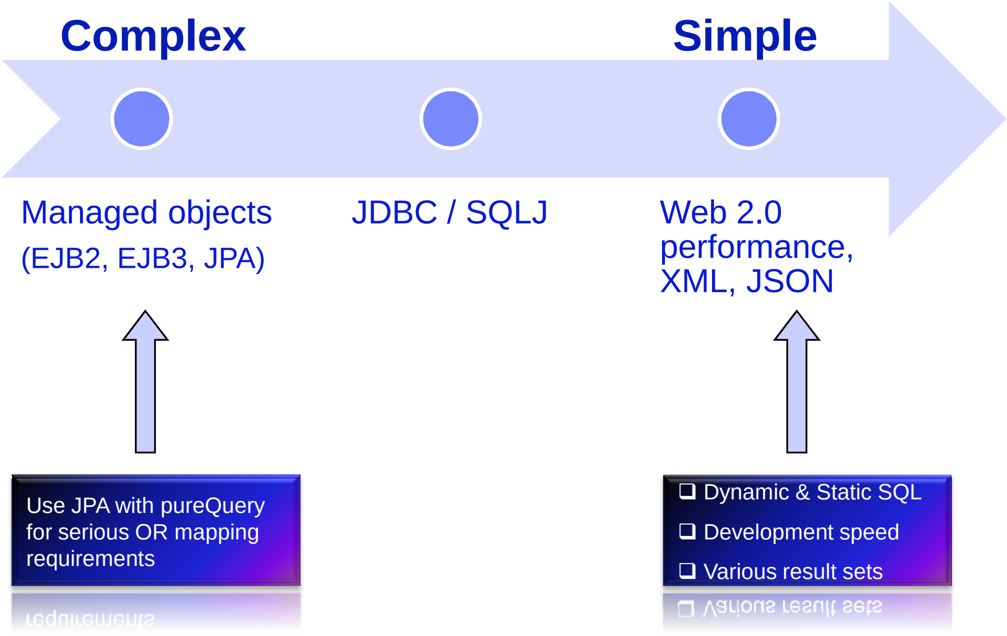
An IBM Proof of Technology

ON DEMAND BUSINESS™

16 slides

Java Data API Space

Simple to Complex – pureQuery



Complex

Simple

Managed objects
(EJB2, EJB3, JPA)

JDBC / SQLJ

Web 2.0
performance,
XML, JSON

Use JPA with pureQuery for serious OR mapping requirements

❑ Dynamic & Static SQL
❑ Development speed
❑ Various result sets

Java Persistence API (JPA) and pureQuery

Together

- Enterprise JavaBeans™ Query Language (EJB QL) 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 with 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
- Hibernate users should use the JPA standard APIs and migrate to the openJPA implementation.

pureQuery API

Java Packages

- **pureQuery Java Packages**

Java Packages	Description
com.ibm.pdq.annotation	Methods for annotations like @Sql, @Update, @Select, @Table, @Column
com.ibm.pdq.runtime	Provides interfaces for Data, ResultIterator and StoredProcedureResult
com.ibm.pdq.runtime.data.handlers	Provides result handlers such as JSON, XML, Paging, Iterator, Call etc
com.ibm.pdq.runtime.exception	DataRuntimeException and UpdateManyExcption
com.ibm.pdq.runtime.factory	DataFactory is used to create and run instances of implementations of Data interface (com.ibm.pdq.runtime)
com.ibm.pdq.runtime.handlers	Interfaces to process results of Object of Type T using handlers for stored procedures call, parameters, result and rows.
com.ibm.pdq.runtime.statement	Hook interface provides ability to provide pre and post execution methods for bracketing any Data API or annotated method

---

Annotation method API

Nuts and Bolts

- **The pureQuery Interface**
  - Example: CustomerData.java
- **Annotated method declarations**
  - @Select, @update, @call
  - One SQL element for each annotation

```
package com.ibm.db2.pureQuery;

import java.util.Iterator;
import com.ibm.pdq.annotation.Select;

public interface CustomerData {
// Select PDQ_SC.CUSTOMER by parameters
@Select(sql="select CID, NAME, COUNTRY, STREET, CITY,
PROVINCE, ZIP, PHONE, INFO from PDQ_SC.CUSTOMER where
CID = ?")  Customer getCustomer(int cid);
  ...
}
```

```
>>-+-@Call---+--(--sql--=--"--SQL-statement--"--)--------------->
   +-@Select-+
   '-@Update-'
>--+---------------------------------------------------------------------------------------------+-->
   '-@Handler--(--+-rowHandler--=--class-name--+--------------------------------+----------------+--)-'
                  |                            '-,--parameterHandler--=--class-name-'                |
                  +-callHandlerWithParameters--=--class-name--+------------------------------------+-+
                  |                                           '-,--parameterHandler--=--class-name-' |
                  '-parameterHandler--=--class-name--------------------------------------------------'

>--modifiers--return-type--method-name--(--+------------------------------------+--);-><
                                           | .-,------------------------------. |
                                           | V                                | |
                                           '---parameter-type--parameter-name-+-'
```

Annotation method API

Nuts and Bolts (continued …)

- **Declared return types**
  - Indicates in what object format the results of SQL will be returned
  - Results can also be processed by the pureQuery engine into a collection of user defined pureQuery beans
  - The generated code's RowHandler and ResultHandler for each method are responsible for carrying out the mappings before annotated method returns the results to the caller
- **Declared parameter types**
  - A repeating argument pair (type and name) expected at method invocation time
  - These parameters are matched with the parameter markers specified in SQL statements
  - The parameters can be scalar types (?), bean classes (:name) or map objects (:name)
- **Hooks**
  - Is a callback mechanism to with specialized processing

```
return-type:

>>-+-Iterator<T>-----------+---------------><
   +-java.sql.ResultSet----+
   +-List<T>---------------+
   +-Map<String,Object>----+
   +-StoredProcedureResult-+
   +-T---------------------+
   +-T[]-------------------+
   '-void------------------'
```

Parameter marker	Meaning
?	Scalar – cannot mix with others
?n	Match a method's parameter by position. Can use same ordinal marker more than once in same SQL statement
:name	Shortened version of ?1.name
:n.name	These markers refer to method parameters ?*n*, which must refer to java.util.Map<String> objects or pureQuery beans. *name* must refer to a property within a java.util.Map<String> object or a pureQuery bean.

Parameter Markers

Examples – Annotated Method

- **Annotated Method Style**
  - Interface defining a method to insert a record

```
import com.company.Employee;
public interface HumanResources
{
 @Update(sql=
  "INSERT INTO HRDept.Employee(EMPNO, FIRSTNME, " +
 "MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)" +
  "VALUES( :employeeId, :firstName, :middleInitial, " +
:lastName, :departmentId, :extension, :hireDate)",
  int newEmployee( Employee newHire );
}
```

```
public Employee {
    public String employeeId;
    public String firstName;
    public String middleInitial;
    public String lastName;
    public String departmentId;
    public String extension;
    public Date hireDate;
 }
```

  - Interface defines a method newEmployee () returning an int representing SQL statement's update count and making use of Employee bean as newHire
  - Use above method in an application shown below:

```
Connection con = DriverManager.getConnection(...);
HumanResources hr = DataFactory.getData( HumanResources.class, con );

Employee newCollegeHire = new Employee("000010", "CHRISTINE", "I",
"HAAS", "A00", "3978", new java.sql.Date(System.currentTimeMillis()));

int oneCount = hr.newEmployee(newCollegeHire);
```

Parameter Markers

Examples – Inline Method

- **Inline Method Style**
  - Insert statement to be visible in application source
  - Use Update method defined in an implementation of data interface
  - Create an instance of Data interface
  - Create a new Employee object
  - Call db.update() method

```
public Employee {
    public String employeeId;
    public String firstName;
    public String middleInitial;
    public String lastName;
    public String departmentId;
    public String extension;
    public Date hireDate;
 }
```

```
Connection con = DriverManager.getConnection(...);
Data db = DataFactory.getData(con);

Employee newCollegeHire =
   new Employee("000010", "CHRISTINE", "I", "HAAS", "A00",
      "3978", new java.sql.Date(System.currentTimeMillis()));

int oneCount = db.update("INSERT INTO HRDept.Employee(EMPNO, FIRSTNME, MIDINIT, LASTNAME, " +
 "WORKDEPT, PHONENO, HIREDATE) VALUES(:employeeId, :firstName, :middleInitial, :lastName, " +
 ":departmentId, :extension, :hireDate )", newCollegeHire );
```

---

pureQuery Beans

Requirements

- **pureQuery supports JavaBeans convention**
  - But, pureQuery works well with objects that do not follow convention strictly. For example: public properties for variables
- **pureQuery bean**
  - Must have no-argument constructor available
  - Fields must be unique to case in-sensitive search. For example: name and NAME not allowed
  - Getter and setter methods must come in pair
  - Can have public properties but then no getter or setter
  - Name of properties must be unique to case in-sensitive search
  - getter method is named getName(), the corresponding property must be NAME.

Annotation method API

Implementation Generator

- **Implementation generator executes as a Eclipse plug-in**
  - **Can be run from Data Studio**
    - ✓ Happens automatically during Build Project or whenever you save annotated method java file
  - **Can be run from command line**
    - ✓ Runs through headless Eclipse
    - ✓ Initializes and invokes Eclipse for each interface
    - ✓ Command invocation takes several parameters

- Uses JDBC connection to get ResultSetMetaData and ParameterMetaData for each SQL
- Examines all input and output beans to gather data types for each field
- Creates ParameterHandler method to map all parameters from input objects to SQL statement parameter markers
- Creates RowHandler method to map all parameters from input objects to SQL statement parameter markers
- Creates method that invoke internal pureQuery APIs to use the handlers to perform all required operations

---

Annotated Method – Named Query

SQL and OR mapping in XML File

ORM.XML

XML document is JPA compatible

```
<?xml version="1.0" encoding="UTF-8"?>
<orm:entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
  <orm:named-native-query
      name="ibm.EmployeeInterface#getEmployeeBean(java.lang.String)">
      <orm:query>
        <![CDATA[select EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,
                      PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE,
                      SALARY, BONUS, COMM from EMPLOYEE where EMPNO =
      ?]]>
        </orm:query>
  </orm:named-native-query>
  <orm:entity class="ibm.EmployeeBean">
      <orm:attributes>
         <orm:id name="empno">
            <orm:column name="EMPNO" />
              <orm:generated-value />
         </orm:id>
         <basic name="empno">
             <orm:column name="EMPNO" />
         </basic>
      </orm:attributes>
  </orm:entity>
</orm:entity-mappings>
```

SQL Method

Java object bean

Control bean property name

pureQuery Result Set Mapping

JSON, XML

Custom mapping using handlers

- **Row handler**
  - Allows to do custom mapping between a data row and java object
  - Return single bean
  - Example:
    - Map a single row to an XML element
    - Map a single row to more than one java object (example: nested beans)
- **Result-set handler**
  - Implements the result set iteration strategy
  - Returns sets of objects or complex objects (e.g. List)
  - Example:
    - Map multiple rows to a single structure. e.g. XML, JSON, HTML and complex things that you can think of.

---

pureQuery Result Set Mapping

Result set example

```
public static void queryProductInfo(Data db)
{
    String strSQL = "SELECT NAME, CATEGORY, PRICE " +
          "FROM PDQ_SC.PRODUCT WHERE PID = '100-101-01'";
    String prodInfo = db.query(strSQL, new MyHandler());
    System.out.println(prodInfo);
}
```

- **queryProductInfo** is called from your business layer
- pureQuery API method *query* executes SQL but hooks your custom code **MyHandler** to format the return result set the way you want it.
- Opportunities are limitless to format return result set using custom handler

```
public class MyHandler implements ResultHandler<String>
{
   public String handle(ResultSet rs){
   StringBuffer result = new StringBuffer();
   try {
     ResultSetMetaData m = rs.getMetaData();

     while(rs.next()){
       for(int col = 1; col<=m.getColumnCount(); col++){
          result.append("<"+m.getColumnName(col)+">");
          result.append(rs.getObject(col));
          result.append("</"+m.getColumnName(col)+">");
          result.append("\n");
        }
      }
   }
   catch(SQLException ex){
    throw new RuntimeException ("Unable to access the result " +
       ex.getMessage (), ex);
   }
   return result.toString();
  }
}
```

In-memory collections

Join with relational data

- getEmployees holds data from Employee table from database
- departments holds array of Department bean built (in memory data)
- Join getEmployees with departments through pureQuery API
- Arguments to the queryList
    1. SQL like construct to join in-memory collections.
    2. Result set is returned in EmpName bean
    3. getEmployees - SQL data from previous step
    4. In-memory data in departments
    5. Empno

```
...

Iterator<Employee> getEmployees = db.queryIterator (
   "SELECT * FROM EMPLOYEE", Employee.class);

Department[] departments = ...;

String empno = 'A0010';

List<EmpName> empNames = db.queryList (
  "select c.firstnme, c.lastname from ? c, ? d " +
  "where c.empno=? and c.deptid = d.deptid",
   EmpName.class, getEmployees, departments, empno);

for(EmpName a : empNames)
{
     System.out.println(a.firstnme+", "+ a.lastname);
}
```

Pluggable Control Point

Hook callback

- Hooks are the easiest way to conduct specialized processing in in the midst of a generated code.
- Allows you to receive control in an exit before and / or after each pureQuery database access
- Exit is defined ahead of time and associated with the connection
- Possible uses
    1. Like a database trigger
    2. Add performance monitor hook
    3. Customized error checking and handling after execution

```
public static class TrackingHook implements Hook {

   public void pre(String methodName, Data
objectInstance,
                  SqlStatementType sqlStatementType,
Object... parameters) {
    System.out.println(methodName + "**Customer data has
been accessed**");
   }

   public void post(String methodName, Data
objectInstance,
       Object returnValue, SqlStatementType
sqlStatementType,
          Object... parameters) {
      // do nothing
   }
}
```

Pluggable Control Point

Hook callback

- Example of pre and post hook methods

```
public static class TrackingHook implements Hook {

   public void pre(String methodName, Data objectInstance,
                  SqlStatementType sqlStatementType, Object... parameters) {
    System.out.println(methodName + "**Customer data has been accessed**");
   }

   public void post(String methodName, Data objectInstance,
       Object returnValue, SqlStatementType sqlStatementType,
          Object... parameters) {
      // do nothing
   }
}
```

- Registering a hook

```
...
Connection con = ...;

// use the DataFactory to instantiate the interface and provide an instance of Hook

CustomerData cd = DataFactory.getData(CustomerData.class, con, new TrackingHook());

// execute the sql for getCustomers() and get the results,
// the pre() and post() methods are automatically called

Iterator<Customer> cust = cd.getCustomers();

// the application now consumes the Iterator of Customer beans
...
```

End of Core Topics – Part 2

Presentation By:

Vikram Khatri

Certified I/T Specialist, DB2 Migrations
IBM Advanced Technical Expert DB2 for Clusters
IBM Certified Database Administrator for DB2 UDB V8.1 for Linux, UNIX & Windows
IBM Certified Database Administrator for DB2 UDB V9 for Linux, UNIX & Windows
IBM Certified Solutions Expert for DB2 UDB V8.1 Family Application Development
Project Management Professional Certified
vikram.khatri@us.ibm.com

Burt Vialpando

Certified I/T Specialist, DB2 Migrations, I/T Specialist Certification Board
IBM Certified Advanced Database Administrator for DB2 UDB V8.1 for Linux, UNIX & Windows
IBM Certified Solutions Expert for DB2 UDB V8.1 Family Application Development
IBM Certified Solutions Designer for DB2 Business Intelligence V8
IBM Certified Solutions Expert for DB2 UDB V7.1 Database Administration for OS/390
Oracle 9i Database Administrator Certified Associate
burt.vialpando@us.ibm.com