Managing pureQuery client optimization in Web application environments, Part 1: Optimize applications on a single application server node

Tips for successful deployment

Skill Level: Intermediate

Charul Kapil (chakapil@in.ibm.com)
Software Engineer
IBM

Jaijeet Chakravorty (jaijeet@us.ibm.com)
Software Engineer
IBM

Manoj Sardana
Software Engineer
IBM

11 Feb 2010

pureQuery client optimization requires the use of properties settings to enable a specific stage of the client optimization process. Settings for these properties vary, depending on the required behavior for your Web application environment. This first article of a two-part series describes property settings for a Web-based application running on a single application server node that uses single or multiple databases shared across multiple applications. The second article will focus on how to set client-optimization properties in more complex Web environments, such as with clustered servers. This article assumes you are familiar with the pureQuery client-optimization process and with setting Web application properties in WebSphere® Application Server or in your chosen application-server environment.

Introduction

One of the main reasons that DB2® provides static SQL execution is to help improve performance. Unfortunately, most Java™ applications (unless using SQLJ) did not have the option to use static execution until the introduction of pureQuery client optimization. Therefore, almost all Java applications that access databases incur the overhead of dynamic execution, including preparing the SQL statements (creating their access plans) at runtime. There are ways to minimize this overhead by using caching. Caching can be used on the database server and also on the client. However, memory usage causes limitations in different scenarios. A discussion of caching is out of scope of this article.

Moving to static SQL can help performance by having the plan created ahead of time (at bind time) rather than at run time. Thus, static statements do not require a PREPARE and DESCRIBE operation before execution.

The client optimization capability of the Optim pureQuery Runtime enables existing Java applications, even those developed using Java frameworks, to use static SQL execution when running against DB2 databases and to get the management and performance benefits of static execution. And you can do this without changing the application code.

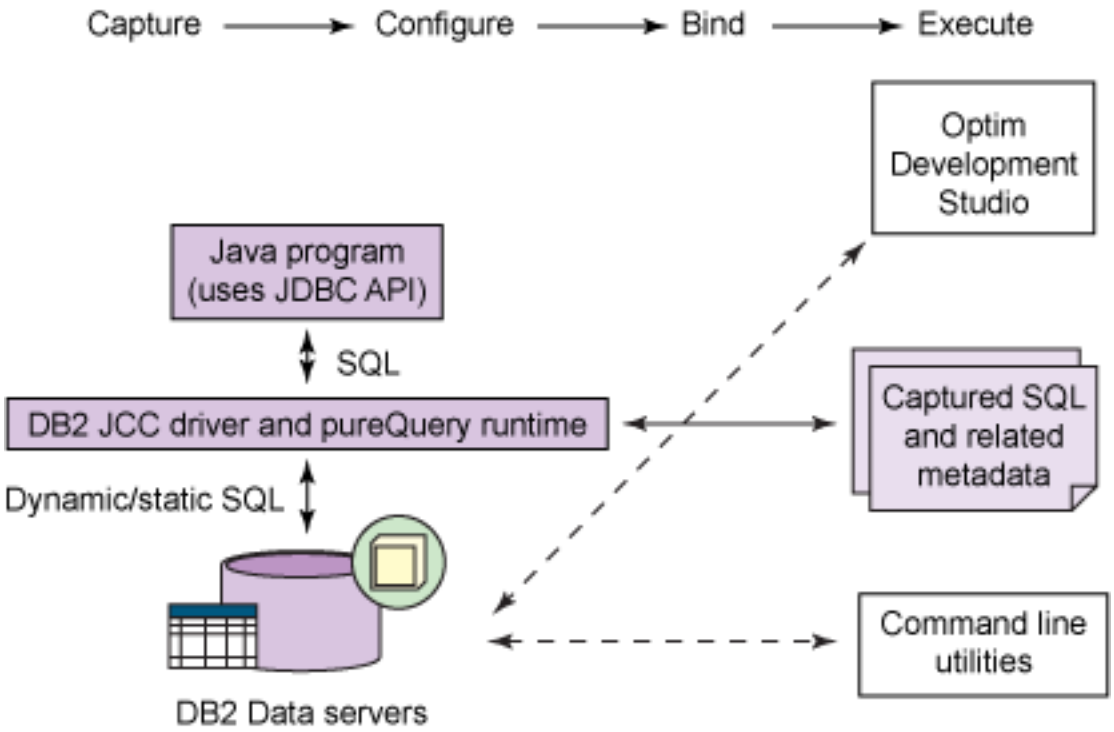
If you are not running DB2, you can take advantage of some other benefits of client optimization. For example, using the development environment in Optim™ Development Studio, you can track back any SQL statement to the originating line of Java source code so you can find and tune SQL more easily, and so you can get a better understanding of the impact of changing a particular SQL statement. You can tune an SQL statement and replace a poorly performing one without having to change the application Java source code. And you can restrict the application to execute a predefined and approved set of SQL statements to avoid SQL injection.

For more detail on the benefits of using static SQL, see the article "'No excuses' database programming for Java" (see Resources). For more information on the capabilities of Optim Development Studio for trackback and for SQL replacement, see the article "What's New in Optim Development Studio 2.2" (see Resources).

Reviewing the client-optimization process

As was originally described in the tutorial "Optimize your existing JDBC applications using pureQuery" (see Resources), the process to convert dynamic SQL to static SQL using pureQuery client optimization requires the following steps, as shown in Figure 1. (If you are not using static, the configure steps and bind steps are not required.)

Figure 1. Client optimization process enables static execution of SQL against DB2 data servers



To use pureQuery, you need the IBM Data Server Driver for JDB and SQLJ, as well as the Optim pureQuery Runtime. Both of these are available for development use on the computer running Optim Development Studio. In Figure 1, this is the box labeled DB2 JCC Driver and pureQuery Runtime. The four major process steps highlighted by arrows in Figure 1 are:

Capture

During this process, you run the application dynamically and enable pureQuery to capture the SQL statements along with the needed information in a pureQueryXml file (capture file). This is the file labeled Captures SQL and related metadata in Figure 1.

Configure

You provide input to map the captured SQL to one or more database package names. The easiest way to do this is using Optim Development Studio.

Bind

You bind the configured capture files into packages or DBRMs to the target server using remote bind. This step is required only for static execution. You can do this using Optim Development Studio, or you can use the Static Binder utility from the command line.

Execute

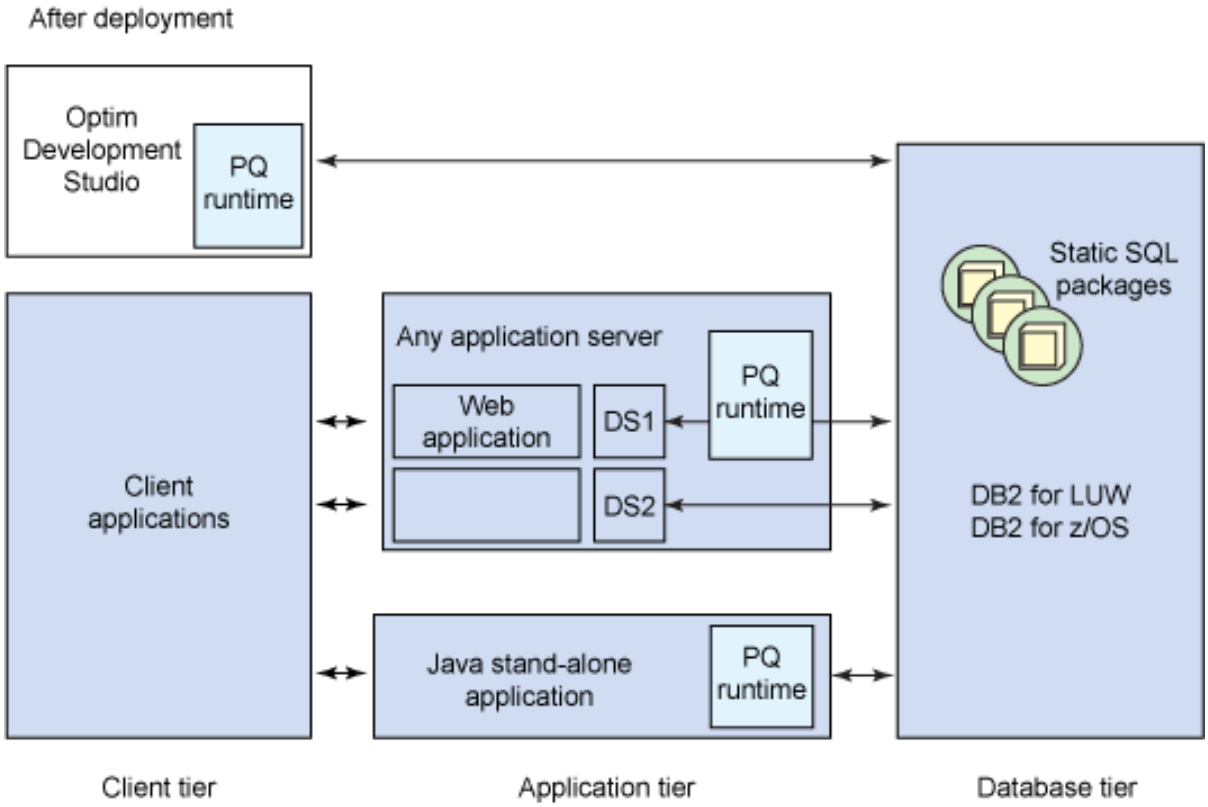
You configure pureQuery to run the captured SQL statically, dynamically, or both, to get the required behavior. You can use Optim Development Studio or

execute from the command line.

Enabling pureQuery Runtime on the application server

Figure 2 shows a typical deployment topology for pureQuery in a three-tiered application environment with client, application, and database tiers. The pureQuery runtime is deployed on the application tier, either where the application server resides, or, if you are not using an application server, where the client drivers are for a standalone application. There is no need to install additional software in the database tier.

Figure 2. Typical web application architecture configured to use pureQuery



Although Figure 2 shows DB2 as the data server, you can also use pureQuery with Informix Dynamic Server and Oracle databases. This article describes how to configure pureQuery when using WebSphere Application Server. If you are using a different application server, the basic tasks described in this article are the same, but the specific steps in your application server environment might be different.

The basic steps are as follows:

1. Install pureQuery Runtime and IBM Data Server Driver for JDBC and

SQLJ.

2. Create a pureQuery-enabled JDBC provider.
3. Create a pureQuery-enabled data source.

Install pureQuery Runtime and IBM Data Server Driver for JDBC and SQLJ

Table 1 lists the Java Runtime Environment (JRE), WebSphere Application Server, and IBM Data Server Driver for JDBC and SQLJ JDBC versions supported by the various releases of the pureQuery Runtime (for Linux®, UNIX®, and Windows® and for z/OS®).

Table 1. Supported versions

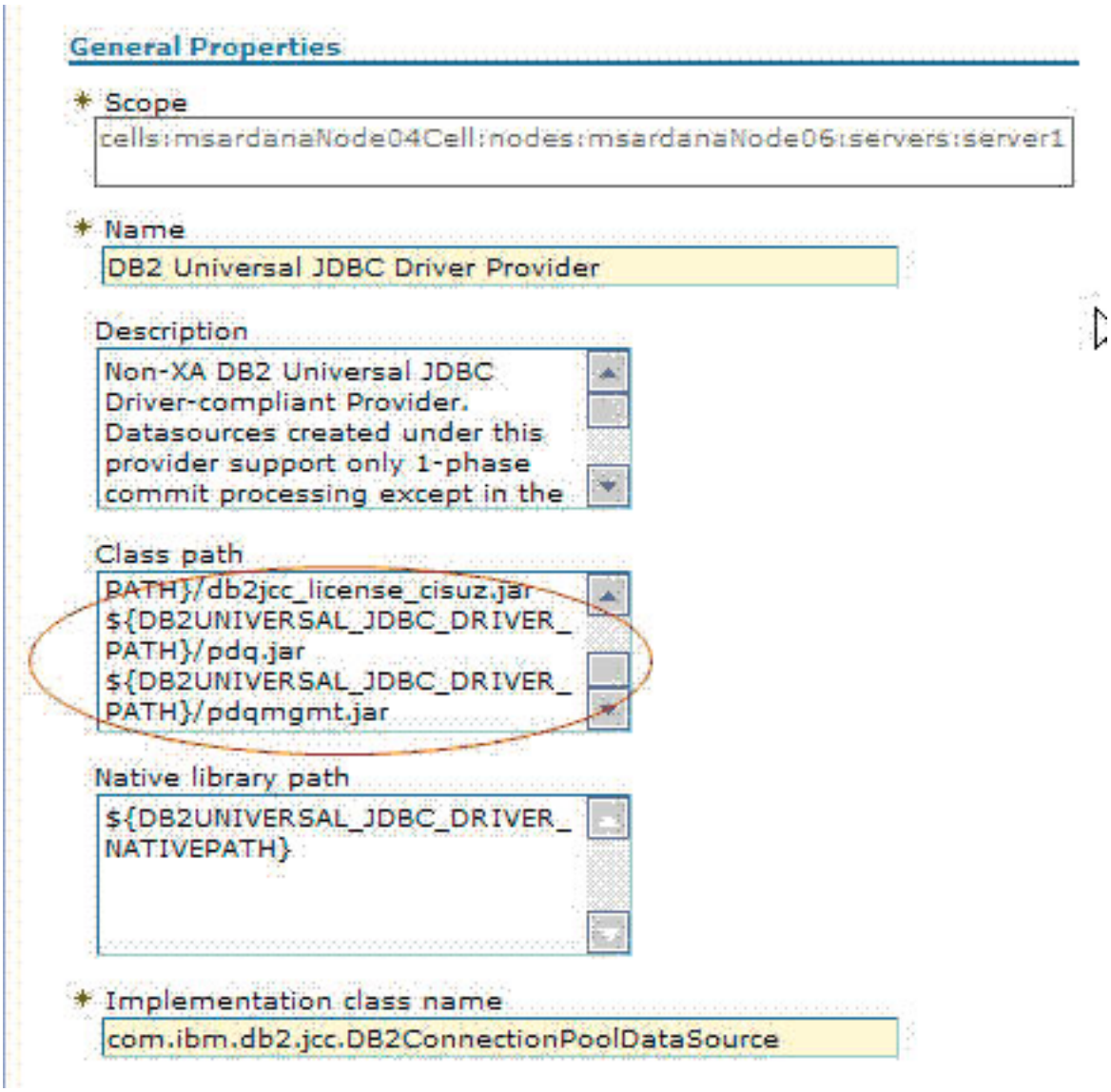
pureQuery Runtime version	IBM JDBC Driver version	JRE version	WebSphere Application Server version
pureQuery 2.1	JCC4.3+ and JCC 3.53+	JRE 1.5 and later	WebSphere Application Server 6.1.0.21, 6.1.0.23, 7.0.0.1
pureQuery 2.2	JCC4.7.89+ and JCC 3.57.86+	JRE 1.5 and later	WebSphere Application Server 6.1.0.21, 6.1.0.23, 7.0.0.1
pureQuery 2.2 and pureQuery 2.2 with Fix Pack 1	JCC4.7.111+ and JCC 3.57.109+	JRE1.5 and later	WebSphere Application Server 6.1.0.21, 6.1.0.23, 7.0.0.1

For more details on the prerequisites, check out the prerequisite links in Resources.

Create a pureQuery-enabled JDBC provider

This section describes how to enable the JDBC provider, using WebSphere Application Server as the application server. It also assumes that the application is using the IBM Data Server Driver for JDBC and SQLJ to connect to the database. This driver is configured as the JDBC provider on WebSphere Application Server. To use pureQuery, you need to add the pureQuery jar files (pdq.jar and pdqmgmt.jar) to the JDBC driver (JDBC provider) classpath. Figure 3 shows the classpath settings in the WebSphere administration console.

Figure 3. Classpath setting for the JDBC provider in WebSphere Application Server administrative console



Create a pureQuery-enabled data source

Now you can create data sources using the created JDBC provider. Follow these steps to create the data source on WebSphere Application Server.

1. In the WebSphere administrative console, click the **Data source** tab (under the **Resource** tab), and click **New**.
2. Provide the data source name and JNDI name used in the application.
3. Select the JDBC provider on which the data source needs to be created, and enter the database details, including database name, server name, port, user ID, and password.

4. Set the value of the pdqProperties data source property to make the application server connection pool aware of pureQuery. The value set for the pdqProperties property can be just a default value, such as `executionMode dynamic`.

pureQuery is enabled for the application server. The application code will use these data sources to create a connection to communicate with the database.

A Web application can use multiple data sources to work on multiple databases. Similarly, multiple Web applications can use a single data source.

Exploring options to configure the pureQuery Runtime environment for client optimization

The client optimization process requires properties to switch from dynamic to static execution and to enable capturing SQL. This section describes three options to set the pureQuery client optimization properties for a Web-based application:

1. Option 1: Configure the pureQuery Runtime options globally
2. Option 2: Configure pureQuery Runtime options for a data source
3. Option 3: Configure pureQuery Runtime options at the application level (new in Version 2.2)

Each option offers benefits and limitations. This section describes the various approaches and considerations to decide which approach to use.

Option 1: Configure the pureQuery Runtime options globally

When should I choose this approach?

This approach enables settings to be applicable for all the data sources using a particular driver. Use this approach when the settings need to be reflected to all the data sources and for all the applications at the same time.

How do I do it?

When using the pdq.properties file to set global properties, put pdq.properties in the driver's CLASSPATH along with the pureQuery jars. The properties set in the driver's CLASSPATH apply to all the applications using the driver. In other words, application and data-source-level granularity is not possible.

What do I need to take into consideration?

While moving from dynamic to static mode, after modifying the pdq.properties file, you need to restart the application server to reflect the changes. Because the properties are applied to all the data sources using the particular JDBC provider, all the SQL will go to the same pureQueryXml file and will run in the same mode.

Option 2: Configure pureQuery Runtime options for a data source

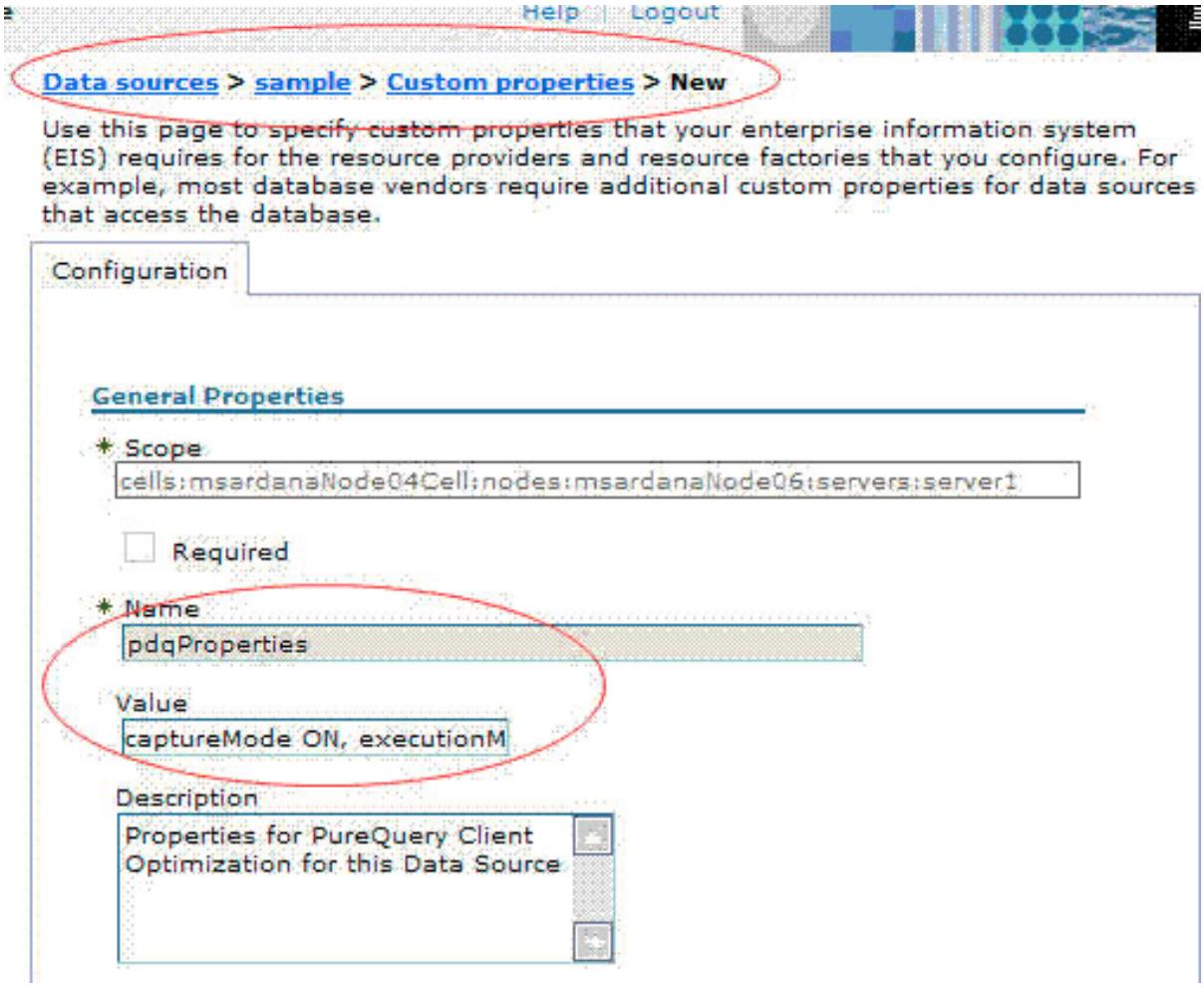
When should I choose this approach?

This approach allows users to set the property for each data source. This is appropriate when the setting for each data source needs to be independent from each other. For example, if your application uses multiple data sources, you might need to run the application dynamically against one data source and statically against another data source.

How do I do it?

To set the pureQuery properties at the data-source level, use the data-source custom property `pdqProperties`. You can set this custom property by going to **DataSources > your datasource name > Custom properties > New** in the WebSphere Application Server administrative console, as shown in Figure 4.

Figure 4. Setting data-source custom properties



What do I need to take into consideration?

As with the global pdq.properties file, each time you change the value of the properties, you need to restart the application server so that the application server can pick up the new values. When the properties are set on the data source, these properties will be applied to all the applications that are using that data source. So when applications share a data source, the captured SQL statements go to a common file for all applications.

Option 3: Configure pureQuery Runtime options at the application level

As described for the other options, properties set in the global pdq.properties file apply to all the applications using that driver and to all data sources used by those applications (because it is placed in the driver CLASSPATH). However, properties set at the data source apply to all applications that are using this data source. To provide a better solution to the limitations of the global and data-source level-setting, Optim pureQuery Runtime Version 2.2 introduced two new properties files, called application-level properties files, for Web applications.

When should I choose this approach?

In an environment where multiple applications use multiple data sources with an n:n relationship.

How do I do it?

By modifying one or both of the following property files:

> **Make the connection pool aware of pureQuery**
> If an application uses only pdq.appwide.properties or pdq.dataSourceName.properties files, set pdqProperty to some value at the data source to make the connection pool aware of pureQuery (see the section Create a pureQuery-enabled data source). The value set for the pdqProperties property can be a default value, such as `executionMode dynamic`). After setting this property the first time, a server restart is required.

pdq.appwide.properties
: This property file specifies the application-specific pureQuery properties. All property values specified in this file will be applied to all the data sources that the application uses.

pdq.*dataSourceName*.properties
: This property file specifies the data-source-specific properties for an application. The properties specified in this file will be applied only to the specific data source that the application uses. The dataSourceName is the string value of dataSourceName custom property for the data source. You can set this custom property using the steps you used to set pdqProperties, shown in Figure 4.

The syntax of setting the properties values is the same as the pdq.properties file, as shown in Listing 1.

Listing 1. Format for pureQuery properties settings

```
pdq.captureMode = ON pdq.executionMode = DYNAMIC
pdq.pureQueryXml = C:/PDQ_2.2/captureFiles/abc_pdq.xml
```

To set some properties for a particular application for only one particular data source, use the pdq.dataSourceName.properties file.

Using these properties files, you can achieve application and data-source-level granularity for the pureQuery client optimization properties.

What do I need to take into consideration?

Using JACL scripts
WebSphere Application Server provides JACL scripts to purge connection pools using the WebSphere Application Server administration tool. To learn more about connection-pooling setting for connection purging, see Resources. To learn more about the WebSphere Application Server administration tool, how you can invoke it, and how the JACL scripts can be written, see Resources.

If an application uses application-specific properties files, such as pdq.appwide.properties or pdq.*dataSourceName*.properties files, you only purge the connection instead of restarting the application server to reflect any changes to the property files.

Configuring application-level pureQuery properties

This section describes more about the application-level pureQuery properties for client optimization, and about how these properties provide application and data-source-level flexibility.

Where to put the application-level properties

To reflect the property values set in the pdq.appwide.properties or in the pdq.dataSourceName.properties properties files, place these files in at least one of the following locations:

- Under WEB-INF/classes for the Web application
- Under WEB-INF/lib directory as a .jar file
- Directly under the installed application directory

The first two options are viable whenever there is a Web module in the application. For EJB applications, you can use any of the three locations.

For example, suppose an application TestClintOptEAR.ear is deployed on the WebSphere Application Server. The application will be installed at the following location: `<WAS_INSTALL_ROOT>/profiles/AppSrv01/installedApps/myNode01Cell/TestClintOptEAR.ear`. AppSrv01 is the profile name, and myNode01Cell is the node name. The application contains a Web module TestClintOpt.war. Here are the options for setting the pureQuery properties files:

Option 1: Place pdq.appwide.properties and/or pdq.*dataSourceName*.properties files in the `TestClintOptEAR.ear/TestClintOpt.war/WEB-INF/classes` directory.
Only Web modules have a `WEB-INF/classes` folder, so this option is only

suitable for an application using the Web module.

Option 2: Create a .jar file, such as `prop.jar`, that contains pdq.appwide.properties or pdq.*dataSourceName*.properties properties files, or both, and place this .jar file in the `TestClintOptEAR.ear/TestClintOpt.war/WEB-INF/lib` directory.

WAR modules can read the properties .jar files directly from the WEB-INF/lib folder, so you do not need to add this .jar information in the classpath. This option is only suitable for an application using the Web module because it has a `WEB-INF/lib` folder after application deployment.

Option 3: Place the prop.jar file, as described in Option 2, in the EAR folder at `<WAS_INSTALL_ROOT>/profiles/AppSrv01/installedApps/myNode01Cell/TestClint`

Add an entry for prop.jar to the MANIFEST.MF of all modules of the application that would use this .jar file to set pureQuery properties. For the example, edit MANIFEST.MF for the TestClintOpt.war module in `<WAS_INSTALL_ROOT>/profiles/AppSrv01/installedApps/myNode01Cell/TestC TestClintOpt.war/META-INF/MANIFEST.MF`. By default MANIFEST.MF file contains only the version details. Also, add this entry to the Class-Path in `prop.jar`. Be sure to add a space before and after the entry, and insert a return at the end, as shown in Listing 2.

Listing 2. MANIFEST.MF content

```
Manifest-Version: 1.0
Class-Path: <path>/prop.jar
```

This way of including the properties files is applicable for all types of applications.

If an application that contains Web and EJB modules needs to specify application-specific properties, a third way of including the prop.jar file is preferred, because you need to maintain only one copy of the properties file. Add the `prop.jar` entry in all the MANIFEST.MF modules of the application. To make any change to the properties values, you need to edit only one copy of the application's properties. If an application has only Web modules, the first option is preferable for easy edit.

Using properties files in an example scenario

Resolution of property values
When a property is set in multiple places, the property value is resolved in the following order:

1. pureQuery looks for pdq.dataSourceName.properties.

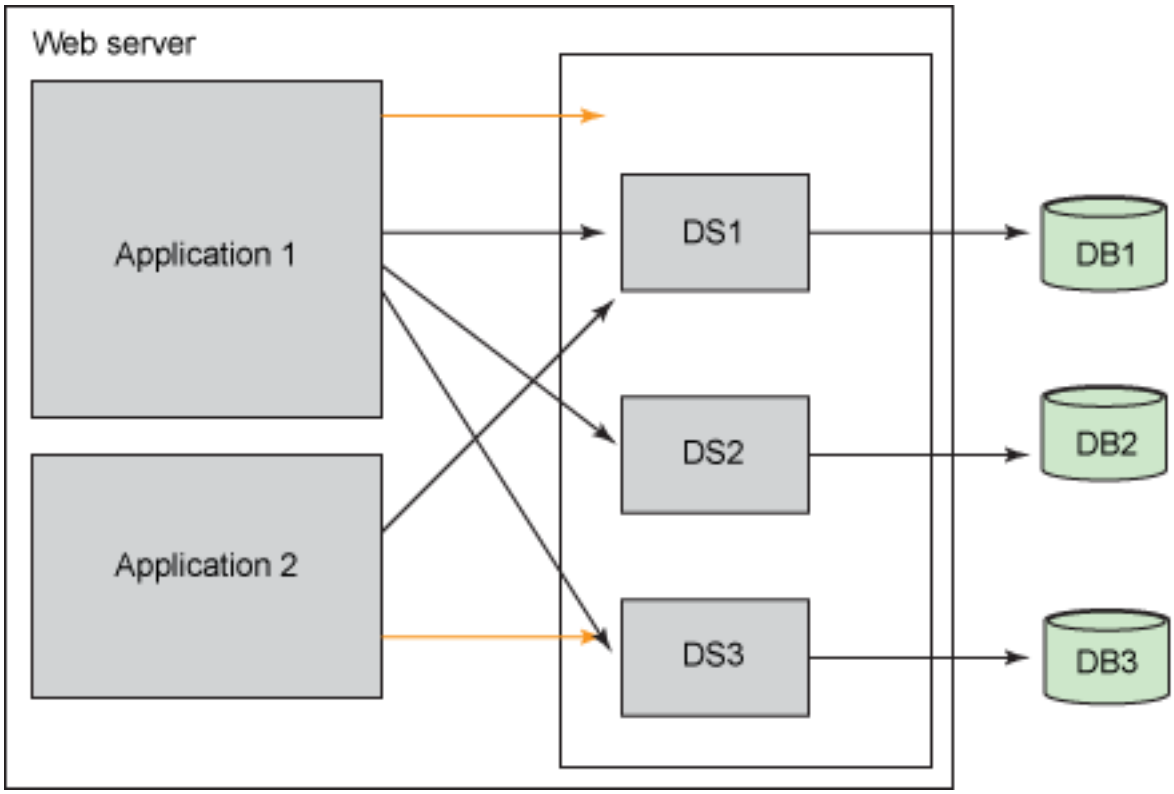
2. If pdq.dataSourceName.properties is not available, pureQuery looks for pdq.appwide.properties.
3. If neither pdq.dataSourceName.properties nor pdq.appwide.properties is available, pureQuery looks for data-source custom properties pdq.properties.
4. If pureQuery is not able to resolve properties with steps 1-3, it will look for pdq.properties at the global level.
5. If a property is still not resolved, pureQuery uses default values.

This section describes how to use properties files to obtain the required behavior.

In complex Web environments with multiple applications or multiple data sources, a different application might need a different mode of execution on any data source. You can get this behavior by setting the properties in both of the application-level properties files. When an application has both pdq.appwide.properties and pdq.dataSourceName.properties files, the application considers both the properties files. However, preference is given to pdq.dataSourceName.properties if the same property is specified in both files.

The following example scenario explains the usage of these properties. The example scenario has two applications using multiple data sources, as shown in Figure 5.

Figure 5. Initial application scenario



Application 1

Application 1 uses three data sources. While DS1 and DS2 are mapped to different DB2 databases, DS3 is a non-DB2 database (Informix or any other DBMS that uses IBM Data Server Driver for JDBC and SQLJ). Initially, an administrator has set up all the databases for capture by setting `captureMode ON` in the pdq.appwide.properties file. Individual data-source-level properties files contain the pureQueryXml files for each data source. Table 2 describes the data sources configuration.

Table 2. Application 1 database configuration

Database name	Configured dataSourceName	Property file name
Database 1 (DB2)	DS1	pdq.ds1.properties
Database 2 (DB2)	DS2	pdq.ds2.properties
Database 3 (non-DB2)	DS3	pdq.ds3.properties

After the administrator captures all the SQL in DS2, he wants to start using static execution against that data source. However, DS1 still needs to run in the capture phase. Meanwhile on DS3, even though static SQL is not possible, the administrator wants to use the option to allow only captured SQL to be executed.

For DS2, against which the administrator is ready to run the application statically, the

administrator can override the captureMode and executionMode properties by providing their values in the pdq.ds1.properties file. The properties would be set to `captureMode OFF` and `executionMode STATIC`. If the administrator wants to disallow any dynamic execution, he can also set `allowDynamicSQL FALSE`. Listing 3 shows these values in the pdq.ds2.properties file.

Listing 3. pdq.ds2.properties settings

```
captureMode=OFF
executionMode=STATIC
pureQueryXml=app1_ds2.pdqxml
```

Because DS1 is still in the capture phase, nothing needs to be overridden. Only the pureQueryXml file name is provided in the pdq.ds1.properties file, such as `pureQueryXml=app1_ds1.pdqxml`.

For DS3, to help prevent SQL injection attacks using dynamic SQL, the administrator must set the `capturedOnly` property to `TRUE` and set the `captureMode` property to `OFF` in the pdq.ds3.properties file, as shown in Listing 4.

Listing 4. pdq.ds3.properties settings

```
captureMode=OFF
capturedOnly=TRUE
pureQueryXml=app1_ds3.pdqxml
```

Application 2

This application uses only one DS1 as its data source. The application was originally set up to run the capture phase. After the capture is complete, the administrator wants to run the application using static execution, but he does not want to change the application and deploy it again. To achieve the required behavior for both applications, the administrator must use the pdq.dataSourceName.properties file to override the existing application-level properties.

To run the application statically on DS1 without changing the application, the administrator can override the captureMode and executionMode properties by providing new values in the pdq.ds1.properties file. Listing 5 shows the value sets in the pdq.ds1.properties file for Application 2.

Listing 5. pdq.ds1.properties settings

```
captureMode=OFF
executionMode=STATIC
pureQueryXml=app2_ds1.pdqxml
```

On DS1, you can set the application-specific properties by using pdq.appwide.properties file of both applications.

The complete solution

For the complete solution, there are two pdq.ds1.properties files and two pdq.appwide.properties files. Because they are in different directories under their respective applications, each file applies to its own specific application.

For the complete example solution, the files and properties in Application 1 are shown in Table 3.

Table 3. Application 1 files and properties

Filename	Property definition	Affects which data source?
pdq.appwide.properties	captureMode = ON executionMode = DYNAMIC	DS1, DS2, DS3
pdq.ds1.properties	pureQueryXml = app1_ds1.pdqxml	DS1
pdq.ds2.properties	captureMode = OFF executionMode = STATIC pureQueryXml = app1_ds2.pdqxml	DS2
pdq.ds3.properties	captureMode = OFF captureOnly = TRUE pureQueryXml = app1_ds3.pdqxml	DS3

The files and properties in Application 2 are shown in Table 4.

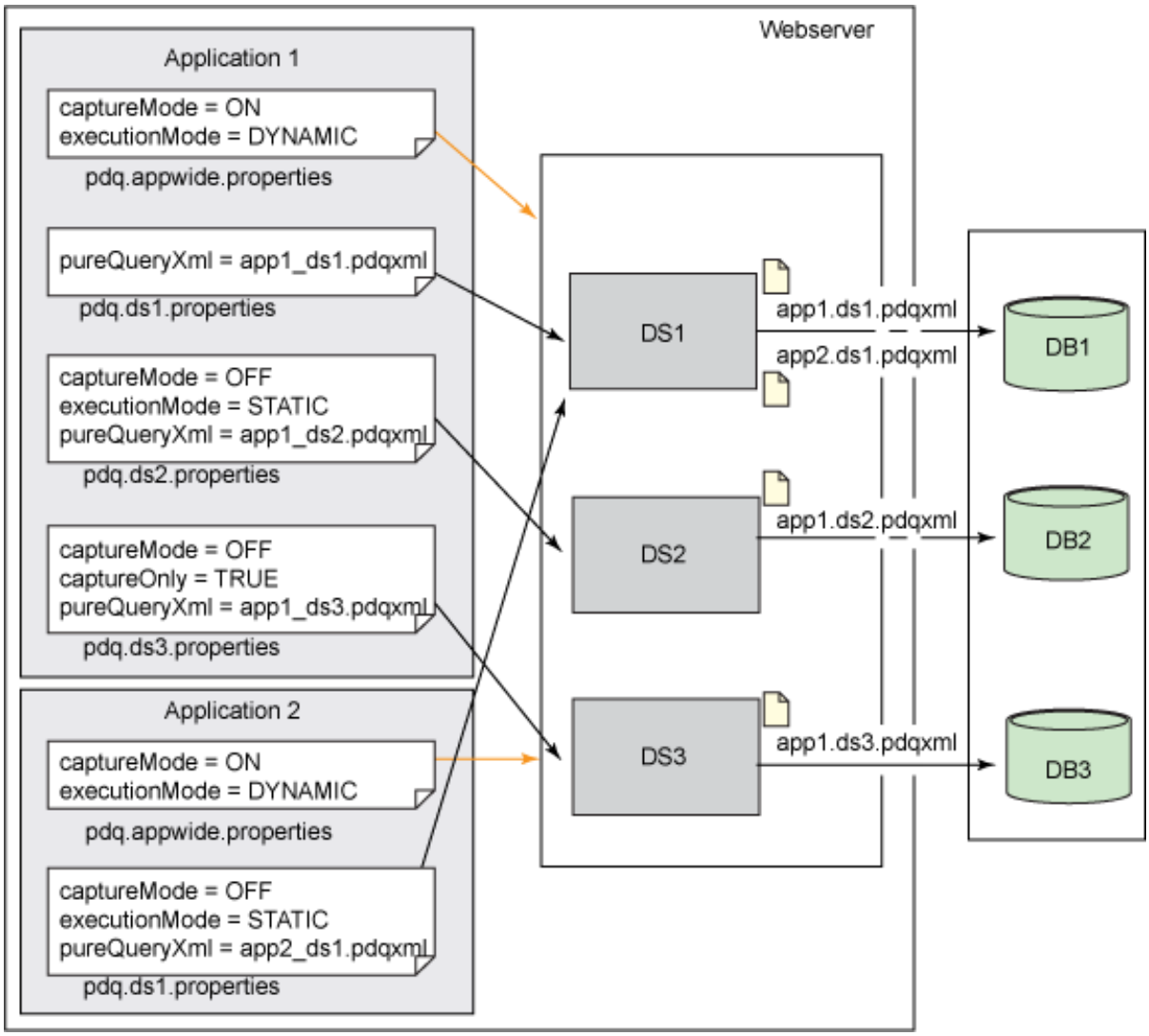
Table 4. Application 2 files and properties

Filename	Property definition	Affects which data source?
pdq.appwide.properties	captureMode = ON executionMode = DYNAMIC	DS1, DS2, DS3
pdq.ds1.properties	captureMode = OFF executionMode = STATIC pureQueryXml = app2_ds1.pdqxml	DS1

Data source DS1 uses files app1.ds1.pdqxml (for Application 1) and app2.ds1.pdqxml (for Application 2) for the DB1 database. Data source DS2 uses file app1.ds2.pdqxml (for Application 1) for the DB2 database. Data source DS3 uses file app1.ds3.pdqxml (for Application 1) for the DB3 database.

The complete example solution is shown in Figure 6.

Figure 6. Final configured solution



Conclusion

This article provided an overview of the client optimization process and the advantages of using it without requiring application code changes, particularly to enable static SQL execution for any existing Java applications that access DB2 databases. You also read about the various levels of configuration properties (global, data source, and application-level) that you can use to support the client optimization process (both capture and execution) in a single node Web-server environment in which multiple applications are sharing multiple data sources.

The second article in the series takes this to the next level of complexity and explains how to use these property files to support applications or data sources that are using multiple or clustered Web servers.

Acknowledgments

We gratefully acknowledge Christopher M Farrar, Kathryn Zeidenstein, and Patrick Titzler for their support in reviewing the content of this article.

Resources

Learn

- Use an RSS feed to request notification for the upcoming articles in this series. (Find out more about RSS feeds of developerWorks content.)
- Read ""No Excuses" Database Programming for Java" to learn more about static SQL and its benefits.
- Learn about the client optimization process using the tutorial "Optimize your existing JDBC applications using pureQuery."
- Find out more about connection pooling setting for connection purging in "Changing connection pool settings with the wsadmin tool."
- Check out "Starting the wsadmin scripting client" to learn more about the WebSphere Application Server administration tool, how you can invoke it, and how to write the JACL scripts.
- Refer to System requirements for pureQuery Runtime for Linux, UNIX, and Windows.
- Find pureQuery on z/OS requirements.
- Go to this technote for prerequisites for pureQuery Runtime for Linux, UNIX, and Windows 2.2.0.1.
- Use this technote for prerequisites for pureQuery Runtime for z/OS 2.2.
- Turn to IBM support for pureQuery installation and planning information.
- See the Integrated Data Management Information Center for product documentation on configuring for Web applications.
- Read "Integrated Data Management: Managing data across its lifecycle" (developerWorks, updated June 2009), which is an article that explains both the vision and reality of Integrated Data Management across roles.
- Watch Demo: Optim solutions for accelerating Java database access to see how one fictional company uses Optim solutions to accelerate problem determination, performance, and development.
- Refer to the developerWorks Optim family page to learn more about Optim solutions. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Learn more about Information Management at the developerWorks Information Management zone. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Link to Data Studio and Optim trial and no-charge software, including a link to Optim Development Studio, which provides you with a 30-day trial for both the development environment and pureQuery Runtime for development use on a single computer.
- Build your next development project with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Check out the Integrated Data Management experts blog and get involved in the Integrated Data Management community space, which has a comprehensive list of resources and downloads.
- Check out the developerWorks blogs and get involved in the developerWorks community.

About the authors

Charul Kapil
Charul Kapil is a software engineer in the IBM India Software Labs, where she has worked for three years. Currently she is working in the OPM-Reporting QA team. Previously, she was a member of the pureQuery Client Optimization QA team.

Jaijeet Chakravorty
Jaijeet Chakravorty is a software engineer with IBM. He is located in the IBM Silicon Valley Lab, San Jose, California. He works with the Java Common Client JDBC driver, SQLJ, and pureQuery client-side products. Jaijeet is involved in product development and providing L3 support to customers worldwide.

Manoj Sardana
Manoj is a staff software engineer who has worked in IBM India software labs for 5 years. He is an IBM-certified advance administrator and application developer for DB2 Linux, UNIX, and Windows. Currently he works on the pureQuery development team. Manoj holds a computer engineering degree from NITK Surathkal. In his free

time, he enjoys listening to music and playing with children.