Optimize Enterprise Generation Language (EGL) applications using pureQuery

Venkatesh Gopal, Data Studio Development Team
Kathryn Zeidenstein, Data Studio Enablement team
About EGL and pureQuery

Enterprise Generation Language (EGL) is a modern programming language specifically designed to help the business-oriented developers quickly write full-function applications and services based on Java™ and modern Web technologies. Business-oriented developers write their business logic in EGL source code using the powerful development facilities of Rational® Business Developer Extension (RBD), Rational Developer for System z with EGL, or Rational Developer for i for SOA Construction. From there, the RBDe tools generate Java or COBOL code, along with all the runtime artifacts you need to deploy the application to the desired execution platform.

pureQuery is IBM's, high-performance data access platform focused on simplifying, developing, securing, managing, and optimizing applications that access data. pureQuery provides access to data in databases and in-memory Java objects via its tools and APIs (delivered in IBM Data Studio Developer), and runtime environment (IBM Data Studio pureQuery Runtime). pureQuery also provides monitoring services, which enable end-to-end database monitoring for Java applications, as delivered in DB2 Performance Expert Extended Insight Feature.

Data Studio Developer provides key database development capabilities to help you develop and test database routines, generate and deploy data-centric Web services, create and run SQL and XQuery queries, and develop and optimize Java applications. Designed for use with IBM Data Studio pureQuery Runtime, together these offerings can improve application performance, security, and manageability. You can get a trial download and find out more information about Data Studio Developer here. We strongly recommend that you also install Fix Pack 1, which you can find here. (A development version of pureQuery Runtime is included in the Data Studio Developer.)

pureQuery provides an API, but you don't have to use it to take advantage of pureQuery capabilities. You can use pureQuery to optimize existing Java database applications that access DB2 on any platform, including Java applications generated from EGL. This capability is important because as more and more business-critical applications move to Java and Web 2.0 technologies, the expectations for performance, security and reliability increase as well. Optimizing with pureQuery helps these data-driven applications meet those expectations.

How does pureQuery “optimize” database access? The secret sauce to pureQuery optimization is embodied in its ability to capture SQL and associated metadata from running Java applications. After SQL is captured, the development tools and the runtime can take advantage of that captured SQL in a couple of crucial ways.

- By capturing the SQL, it can now be bound into database packages to enable static execution. Static execution mode means the access path to the database is locked down and is less likely to vary widely as can happen with dynamic execution. It also provides a more controlled security model and makes it easier to do monitoring, because SQL is bound into unique package names that you provide the names for. See this article for information about why static SQL provides better performance, security, and predictability. And if your database of choice is DB2 for z/OS, you have the potential to see significant reductions in CPU cost. See this article for more details on performance results against DB2® for z/OS®.

- The metadata from the captured SQL powers the development tool (Data Studio Developer) with capabilities that include the ability to correlate generated SQL with
associated lines of Java source code to make it easier to isolate problems and do impact analysis. In addition, you can see elapsed time profiles of the statements.

**Eclipse Shell Sharing**

Both Data Studio Developer and Rational Business Developer are based on open source Eclipse. Compatible Eclipse-based products can be installed in a "shell-shared" mode. The concept of shell sharing between Eclipse-based products means that you can share the core Eclipse components so that they are not duplicated among each Eclipse-based product. Shell sharing eliminates the need to install several Eclipse platforms for each product, thus saving disk space and eliminating duplication of components. The following URL provides a list of compatible levels products that can shell share


**Installation Requirements**

For our tutorial here, we have installed Rational Business Developer v7.5.1 ([trial download](http://www-01.ibm.com/support/docview.wss?rs=0&uid=swg21279139)) and Data Studio Developer 2.1 ([trial download](http://www-01.ibm.com/support/docview.wss?rs=0&uid=swg21279139)) in shell sharing mode. So, when you launch the product, you will get the capabilities of both Data Studio Developer and RBD in a single IDE. Also, this tutorial example uses DB2 for Linux, UNIX, and Windows v 9.5.3 and uses the employee table in the SAMPLE database.

**Objectives**

This tutorial shows you how to use the capabilities of pureQuery in Java applications that were generated by EGL, for accessing DB2 for Linux, UNIX, and Windows and z/OS. The capabilities of pureQuery that can be used are

- **Client optimization** wherein an existing Java application’s SQL statements and associated metadata can be captured, and these captured statements can be made to run in static mode. Client optimization is the foundation to enable many additional benefits in Data Studio Developer including elapsed time for SQL statements and the ability to associate particular SQL statements with particular lines of Java source code.
- **SQL insight** which provides a list of SQL statements originating from your EGL application with details on number of times executed and execution times.
- **SQL replacement**, which lets you replace the SQL statements with better performing ones without having to change the application.
- **Prevention of SQL injection** – by allowing only SQL statements that have been captured and approved to run against the database.

This tutorial covers the details in two sections:

- The first section deals with creation of a simple EGL application and then generating the Java for the application.
- The second section shows how to use pureQuery client optimization with the generated Java to realize the benefits of static SQL and the additional capabilities described above.
Section 1 - Creating a simple EGL application

Step 1 - Launching the product and getting into the EGL perspective

Launch IBM Rational Business Developer. For my installation, the screenshot is shown below. Note that when the product launches, we see that both Rational Business Developer and Data Studio are present in the installation (because we installed them in a shell-sharing mode).

Figure 1. RBD and Data Studio Developer are shell sharing

Figure 2. Launch Rational Business Developer

Make sure that the EGL perspective shows up. If the EGL perspective does not show up, go to Window->Open Perspective->Other and then click on EGL.
Step 2 - Create an EGL Project

Create an EGL Project using the following steps:

1. File->New->EGL Project
   Call the project EGLDSIntegration (highlighted in red in Figure 3) and pick the General Project.

   ![New EGL Project dialog](image)

   **Figure 3. Creating a new EGL project**

   2. Click Next, and leave everything to defaults.
   3. Click on Finish.

Step 3 - Create an EGL Package and a source file

1. Click on the Project, go to the EGLSource folder and click on New Package
2. Enter package1 for the Package name text area, and then click on Finish (Figure 4).
3. Add a new source file by clicking on Package1 from the Project Explorer and then choose **New->Source File** (Figure 5). Name the file simplefetch in the wizard.
Step 4 - Setting build properties

1. Click on the eglbld file (EGLDSIntegration.eglbd), and then pick “Java target System(Basic)” from the build option filter and uncheck show only specified options.
2. After that enter the values of your DB2 connection by updating the values for dbms, sqlJDBCClass, sqlDB, sqlID and SQLPassword. In this tutorial, we set the values of sqlID and sqlPassword, and everything is left at defaults (for connecting to our local DB2 for LUW instance) (Figure 6).
3. Save the eglbld file by clicking on **Save** in the main menu.
Figure 6. Building the Java source

Step 5 - Writing your test application

Click on the empty source file that you created earlier, simplefetch.egl, and cut and paste the following code:

```java
package package1;

// Put EGL Source File Contents Here

// basic program
//
program simplefetch type BasicProgram {

    // Use Declarations
    // use usePartReference;

    empno String;
    firstname String;
    lastname String;
```
```java
function main()

    prepare stmt1 from "select empno,firstname,lastname,hiredate from employee"
    open cur1 with stmt1;
    while (SQLLIB.sqlData.sqlcode == 0)
        get next from cur1 into empno,firstname,lastname, hiredate;
        writeStdOut("Emp Details -> Number: " + empno + " Name : " + firstname + " " + lastname + " Hire Date : " + hiredate);
    end;
    close cur1;

    get with #sql{select firstname,lastname from employee where empno='200140'} into firstname,lastname;
    writeStdOut("Singleton fetch Name: " + firstname + " " + lastname);

    get with #sql{select firstname,lastname from employee where empno='000340'} into firstname,lastname;
    writeStdOut("Singleton fetch Name: " + firstname + " " + lastname);

end

Step 6 - Generate the Project

1. Right click on the Project and then click on Generate. This generates the Java code for the simplefetch application.
```
2. You will need to add the location of the db2jcc jar file. In order to do this, right click on the **Project -> Properties** and then Java Build path. Pick **Libraries** and add the location to your db2jcc.jar file by clicking on the **Add External JARS** button and providing the full path to the file. The jar file will show up as shown in the diagram (Figure 8).
Step 7 - Running the EGL Java application

Go to the javasource Folder under the project and then right click on simplefetch.java and then pick Run As ->Java Application. You will see the results in the bottom right of the screen in the “Console” view.
Figure 9. Run the simplefetch application

Output -

Emp Details -> Number: 000010 Name : CHRISTINE HAAS Hire Date : 1/1/95
Emp Details -> Number: 000020 Name : MICHAEL THOMPSON Hire Date : 10/10/03
Emp Details -> Number: 000030 Name : SALLY KWAN Hire Date : 4/5/05
Emp Details -> Number: 000050 Name : JOHN GEYER Hire Date : 8/17/79
Emp Details -> Number: 000060 Name : IRVING STERN Hire Date : 9/14/03
Emp Details -> Number: 000070 Name : EVA PULASKI Hire Date : 9/30/05

Emp Details -> Number: 200280 Name : EILEEN SCHWARTZ Hire Date : 3/24/97
Emp Details -> Number: 200310 Name : MICHELLE SPRINGER Hire Date : 9/12/94
Emp Details -> Number: 200330 Name : HELENA WONG Hire Date : 2/23/06
Emp Details -> Number: 200340 Name : ROY ALONZO Hire Date : 7/5/97
Emp Details -> Number: 200340 Name : ROY ALONZO Hire Date : 7/5/97
Singleton fetch Name: KIM NATZ
Singleton fetch Name: JASON GOUNOT
Section 2 - Using pureQuery with the EGL generated Java code

Now that we have a simple EGL application running, let's use client optimization to capture SQL and associated metadata from the running Java application.

Once we’ve done that, you’ll see how to bind the captured SQL into DB2 packages to lock down the DB2 access path for more consistent performance and improved security. We'll also touch on some other key benefits of using Data Studio Developer to leverage the captured SQL and metadata for performance tuning, problem isolation, and more.

Step 1 - Switch to the Java perspective

You can switch to the Java Perspective by clicking on Window->Open Perspective->Other->Java and then clicking on OK.

Step 2 - Enable the project for PureQuery support
1. Enable PureQuery support for the project by right clicking on the Project EGLDSIntegration and then selecting pureQuery->Add pureQuery Support...

Figure 10. Add pureQuery to your Java project

2. Pick SAMPLE from the connection on the wizard that appears, then click Next.
3. On the next screen, enable the capturing and binding for JDBC applications.
**Figure 11. Enable capture**

**Note:** When enabling pureQuery support, you may get an error for multiple jar files. Remove the db2jcc.jar file that you added earlier (in Figure 8) by highlighting the JCC file and clicking on **Remove** and your classpath will have only the jcc driver that pureQuery includes.

The EGL application should run without any issues with the db2jcc file that the pureQuery enablement part included. Switch the perspective back to EGL and use the **Run As** option as before to ensure that the application works without any issues.
Step 3 - Create a new PureQuery Run Configuration

1. Create a new PureQuery run configuration. Right click on the Project and select Run As- > Run Configurations.

2. Go to PureQuery and pick New_configuration and give the configuration the name EGLDSIntegrationPQ as shown in Figure 13.

3. Enter package1.simplefetch in the main class field.

4. Click on Apply and then Run.
Step 4 - Capture/bind and run in static mode

1. Click on the db2jccConfiguration.properties file (found in the JavaSource folder in the project) and add the following properties to enable capture and run in dynamic mode:

   ```
   db2.jcc.pdqProperties=captureMode(ON),executionMode(DYNAMIC)
   ```

2. When you run the application, since you turned capture mode on, all the SQL statements in the application are captured in a file called capture.pdqxml. This file can be found in the PureQueryFolder in the project. Double clicking on the file opens up a customized editor with the SQL statements that have been captured. For the above EGL/Java application, Figure 14 shows the list of SQL statements.
3. The next step is to bind the captured statements to a database package to lock in the access plan. For this, you need to right click on the project and select PureQuery -> Bind Application.

A database package (not to be confused with a Java package) is the object that contains all the information that is required for DB2 to run the SQL statements included in that package. The process to create packages is called a bind, and that is when DB2 does such things as determine the access paths for each SQL statement associated with a particular package.
If prompted for the database connection, pick SAMPLE from the list and click on **Finish**. The default schema that gets used is NULLID and also the package names etc are preset. See the [detailed documentation](#) on pureQuery bind options to see how to change these to desired.
values. To see if the statements are indeed bound, you can switch to the Data Perspective, explore into your database as shown in below in Figure 16.

Figure 16. Exploring inside the database package

4. After the bind process is done, run the application in static mode first by modifying the `db2jcc.configuration` properties to the following:

   ```properties
db2.jcc.pdqProperties=captureMode(OFF),executionMode(STATIC)
   ```

5. Save the file.

6. Run the application again by clicking on the Run menu and then picking `EGLDSIntegrationPQ`. Now, you will have all the statements running in static mode rather than dynamic mode.

Figure 17. Running your java application (`EGLDSIntegrationPQ`)
Using other pureQuery capabilities

Now that you’ve done that, try these other cool features in Data Studio Developer
Here are some cool things you can do with Java applications in Data Studio Developer. For more details on any of these capabilities, including how to use them, see this article.

pureQuery SQL outline

After you capture the SQL, the pureQuery outline provides key insights into the SQL statements that were generated from your Java application. The Java view on the outline will show the Java classes from which the SQL originated. (This capability is available without having to capture the SQL first, but it is more accurate with captured SQL.)

Figure 18. SQL statements and from which Java packages they originated

You can also see a profile of the statements that were run including the elapsed time for each statement and how many times each statement was executed.
For viewing execution details on the SQL, click on the “Toggle profile” button highlighted in the upper right of Figure 19 below and expand the view to look at the SQL details.
This is not something you would do in a development environment when you have access to source code. But if there were a situation in which the source code was inaccessible or a DBA would need to change SQL to fix an emergency performance problem, this capability can be useful.

To do this, you can use the capture.pdqxml editor to replace an existing SQL statement with a statement that has been tuned. For more information on replacing SQL statements, see this article or the associated video series.

**Reduce SQL injection risk**

SQL injection refers to a (sometimes) malicious intent to leverage vulnerabilities in dynamic SQL to pass in commands to the database. There are two ways to avoid this by using pureQuery.

1. Run your application only using static execution, which means that all SQL executed against the database is known because it is in a package that you have created. This option may not be possible for all applications.
2. Run the application to ensure that only SQL that was captured can be executed. You do this by setting a driver property before running the application. Any SQL that doesn't match what is in the capture file will not be executed.

**Summary**

Both Rational Business Developer and Data Studio Developer are key tools to help organizations implement a strategy toward enterprise modernization. Use Rational Business Developer to create modern, web-enabled applications, but make the database access secure and high-performing using the capabilities in Data Studio Developer and pureQuery Runtime.