Tools for a dependable, predictable Java enterprise environment
May 2004

DB2® Information Management Software

Managing Enterprise Java Applications for DB2

By Curt Cotner,
Maryela Weihrauch,
Heather Lamb

Introduction:

Our large business customers deploying Java™ Enterprise Database Applications tell us they need to be able to count on consistent and dependable application performance day after day and week after week. They need to count on their Java applications to perform dependably in mission critical systems. Their businesses depend on consistent performance. They need tools that make it easy to monitor and manage performance, and quickly get to the root of problems that arise. IBM has responded to these needs by developing a suite of DB2® software offerings that provide large business customers a superior, predictable, dependable and diagnosable Java enterprise environment. No competitor comes close to DB2's unique offerings for manageability and dependability of Java applications.

This paper will help customers who want more control over their Java applications understand the full extent of what is offered to them by DB2. Not only can Java applications be as dependable and manageable as enterprise applications in Cobol, but control and monitoring Java applications is simpler for the developer. This paper describes the features developers should use to maximize the manageability of their applications. They will learn how integrated monitoring for WebSphere® software from IBM and DB2, the Universal Java Client, and static SQL support via SQLJ (embedded SQL for Java) are the essential tools needed to create the dependable and manageable Java Enterprise system they need.

Dependable and manageable Java Enterprise software has become a major focus for zSeries customers with the introduction of the new zSeries optional feature - the zAAP Processor. The new IBM @server® zSeries® Application Assist Processor (zAAP), available on the IBM eServer zSeries 990 (z990) and zSeries 890 (z890) servers, is an attractively priced specialized processing unit that provides a strategic z/OS Java execution environment for customers who desire the powerful integration advantages and traditional Qualities of Service of the zSeries platform.

zAAPs may enable customers to:

- Integrate and run e-business Java workloads on the same server as your database, helping to simplify and reduce the infrastructure required for Web applications
- Maximize the value of their zSeries investment through increased system productivity by reducing the demands and capacity requirements on general purpose processors which may then be available for reallocation to other zSeries workloads.

Java workload is transparently executed on the zAAP processors. You don't have to change your applications at all. IBM does not impose software charges on zAAP capacity. When configured with general purpose processors within logical partitions running z/OS (or z/OS.e), zAAPs may help increase general purpose processor productivity and may contribute to lowering the overall cost of computing for z/OS Java technology-based applications.

To learn more about zAAPs, the Java Predictor Tool, or the zSeries platform, please contact your local IBM Sales Representative, IBM Business Partner or visit: **ibm.com**/servers/eserver/zseries/zaap/

The Power of Simple Application Management

Easy troubleshooting and simple, easy to use performance monitoring is important for large enterprises. In Java as in any other language, developers need to be able to quickly pinpoint and prevent potential problems in either the application or the application environment. A large scale enterprise system is complex, and the number of application components could number in the hundreds or thousands. DB2 makes it easier to tell which application component is experiencing a problem, and to track program behavior across all layers of the system architecture.

The cornerstone for this solution is the DB2 Universal Java Client. It is a single JDBC (type2/type4) and SQLJ client for DB2 available in both workstation and z/OS hardware configurations. It replaces individual clients under each configuration with a single product, making application behavior more portable across DB2 platforms. It improves DB2Connect consistency and performance by unifying many of the code paths for specific hardware configurations. And, for the enterprise application spread across many different hardware configurations, it is a single point of unification that the developer can count on no matter what environment he or she is working in.

Error handling, tracing, and instrumentation features in the Universal Java Client bring the power of application management straight to the developer. Often the first contact a developer has with a Java application problem is through the Java SQLException that is thrown. The Universal Java Client has made it easier to retrieve DB2 error information from the SQLException, using the new DB2Diagnosable class. Complete sqlca information (SQLCode, SQLErrmc, SQLErrp, SQLErrd, SQLState, SQLWarn, and SQLErrorMessage) can be retrieved from DB2 via both SQLJ and JDBC whenever an exception is thrown by the server. In addition, native error messages can be retrieved from the DB2 server (including DB2 for OS/390 v6 and v7), with the installation of supporting stored procedures. To ensure maximum performance, the native message is only returned when explicitly requested.

The application trace is another primary method of problem investigation for application developers. The Universal Java Client trace facility has several enhancements aimed at ease of use and information integration. The trace has also been integrated with WebSphere trace. It can be activated through an external API of the DB2Connection class and, importantly, it can be dynamically turned on and off. Finally, the developer can configure the trace to display a variety of different layers of detail to help focus on the key level being diagnosed, be it analyzing method flows or examining the contents of a DRDA® buffer. Example 1 shows how the trace level can be set to show only method calls, making tracing through an application flow much simpler and removing trace information that would be extraneous to this activity.

Example 1

Trace fragment showing method calls for the execution of a query:

```
...
[ibm][db2][jcc][Thread:main][Connection@50b9ee8a]setAutoCommit(false) called
[ibm][db2][jcc][Thread:main][Connection@50b9ee8a]prepareStatement(SELECT
   FKEY FROM WRKTB01 WHERE (FKEY >= ?) OPTIMIZE FOR 1 ROW ) called
[ibm][db2][jcc][Thread:main][Connection@50b9ee8a]prepareStatement () returned
   PreparedStatement@ee32e8a
[ibm][db2][jcc][Thread:main][PreparedStatement@ee32e8a]setShort (1, 400) calle
[ibm][db2][jcc][Thread:main][PreparedStatement@ee32e8a]executeQuery () called
[ibm][db2][jcc][Thread:main][ResultSetMetaData@2b98ae8a]BEGIN
   TRACE_RESULT_SET_META_DATA
[ibm][db2][jcc][Thread:main][ResultSetMetaData@2b98ae8a]Result set meta data for
   statement Statement@136bee8a
[ibm][db2][jcc][Thread:main][ResultSetMetaData@2b98ae8a]Number of result set
   columns: 12
...
```

Once an application is deployed in a production environment, the ability to monitor performance and the ability to easily analyze runtime behavior rise in concern. The Universal Java Client supports easy to use monitoring with powerful payoff in terms of enhanced monitoring capability. It allows Java applications to tightly integrate with existing monitoring tools that applications in other languages have used in DB2 for a long time. This enhanced Java application monitoring is available through client instrumentation API's for JDBC or SQLJ, through client tracing of DB2 correlation IDs, and through client application monitoring. Client instrumentation consists of adding four small lines of code to the application after the getConnection() call, as this example shows:

```
 con.setClientUser("maryela1");
 con.setClientWorkStation("9.30.11.123");
 con.setClientApplicationInformation("payment");
 con.setClientAccountingInformation(String);
```

This application-unique information with then be displayed in several ways by server side monitoring tools, including the DISPLAY THREAD DETAIL command, and tools like DB2PM. In z/OS the client userid, client workstation, and client application name shown here will all be included in the standard header that is written for every IFC record, such as accounting records, SQL trace, and lock trace. This makes it much easier to distinguish different client applications coming from the same application server, even when using connection pooling to maximize transaction throughput.

Example 2

DISPLAY THREAD without the client accounting API:

```
-D71B DIS THREAD(*)

NAME   ST A REQ ID        AUTHID   PLAN     ASID  TOKEN
SERVER  RA *  952 db2jccThread USRT001 DISTSERV 004A  11
 V445-G91E81C5.G49D.00F330EEE11F=122 ACCESSING DATA FOR 9.30.129.197
SERVER  RA *  112 db2jccThread USRT001 DISTSERV 004A  123
 V445-G91E81C5.G49D.00F330DF1111=222 ACCESSING DATA FOR 9.30.129.197
SERVER  RA *  432 db2jccThread USRT001 DISTSERV 004A  432
 V445-G91E81C5.G49D.00F330DF736F=432 ACCESSING DATA FOR 9.30.129.197
SERVER  RA *  772 db2jccThread USRT001 DISTSERV 004A  21
 V445-G91E81C5.G49D.00F330DF736F=382 ACCESSING DATA FOR 9.30.129.197
```

DISPLAY THREAD using the client accounting API:

```
-D71B DIS THREAD(*)

NAME    ST A REQ ID          AUTHID  PLAN     ASID  TOKEN
SERVER  RA *  952 db2jccThread USRT001 DISTSERV 004A  432
 V437-WORKSTATION=9.30.129.202, USERID=SALLY,
   APPLICATION NAME=payment
 V445-G91E81C5.G49D.00F330DF7111=222 ACCESSING DATA FOR 9.30.129.197
SERVER  RA *  952 db2jccThread USRT001 DISTSERV 004A  432
 V437-WORKSTATION=9.30.129.214, USERID=JOE,
   APPLICATION NAME=accounts_payable
 V445-G91E81C5.G49D.00F330DF7222=332 ACCESSING DATA FOR 9.30.129.197
SERVER  RA *  952 db2jccThread USRT001 DISTSERV 004A  432
 V437-WORKSTATION=9.30.129.111, USERID=SAM,
   APPLICATION NAME=hr_appl1

 V445-G91E81C5.G49D.00F330DF753F=442 ACCESSING DATA FOR 9.30.129.197
```

Example 3:

DB2PM without the client accounting API

```
1      LOCATION: SYPEC15A                              DB2 PERFORMANCE MONITOR (V7)
          GROUP: N/P                                    ACCOUNTING REPORT - SHORT
      SUBSYSTEM: V71A                                      ORDER: TRANSACT
    DB2 VERSION: V7                                        SCOPE: MEMBER

TRANSACT      #OCCURS  #ROLLBK  SELECTS  INSERTS  UPDATES  DELETES  CLASS1  EL.TIME   CLASS2  EL.TIME ...
               #DISTRS  #COMMIT  FETCHES    OPENS   CLOSES  PREPARE  CLASS1  CPUTIME   CLASS2  CPUTIME ...

db2jccmain          40        0     0.00     2.00     0.00     2.50          0.153494          0.006284 ...
                    40       40     0.00     0.00     0.00     1.00          0.002617          0.001833 ...

      PROGRAM NAME   TYPE     #OCCURS   SQLSTMT   CL7  ELAP.TIME   CL7  CPU TIME   CL8  SUSP.TIME   CL8  SUSP
      SYSLH200       PACKAGE       40      8.00         0.006282        0.001831         0.004182        1.05

      REQUESTER      METH      #DDFS     TRANS     #ROLLBK     #COMMIT     SQLRECV     ROWSENT     CONVI
      9.30.112.192   DRDA         40      0.00           0          40        6.52        0.00      0.00
```

DB2PM with the client accounting API

```
1      LOCATION: SYPEC15A                              DB2 PERFORMANCE MONITOR (V7)
          GROUP: N/P                                    ACCOUNTING REPORT - SHORT
      SUBSYSTEM: V71A                                      ORDER: TRANSACT
    DB2 VERSION: V7                                        SCOPE: MEMBER

              #OCCURS  #ROLLBK  SELECTS  INSERTS  UPDATES  DELETES  CLASS1  EL.TIME   CLASS2  EL.TIME ...
TRANSACT       #DISTRS  #COMMIT  FETCHES    OPENS   CLOSES  PREPARE  CLASS1  CPUTIME   CLASS2  CPUTIME ...

accounts_payable    20        0     0.00     0.00     0.00     5.00          0.031649          0.007198 ...
                    20       20     0.00     0.00     0.00     1.00          0.002452          0.001624 ...

      PROGRAM NAME   TYPE     #OCCURS   SQLSTMT   CL7  ELAP.TIME   CL7  CPU TIME   CL8  SUSP.TIME   CL8  SUSP
      SYSLH200       PACKAGE       20      8.00         0.007118        0.001564         0.005431        1.00

      REQUESTER      METH      #DDFS     TRANS     #ROLLBK     #COMMIT     SQLRECV     ROWSENT     CONVI
       l9.30.112.192 DRDA         20      0.00           0          20        8.00        0.00      0.00

hr_apll             20        0     0.00     4.00     0.00     0.00          0.276760          0.005323 ...
                    20       20     0.00     0.00     0.00     1.00          0.002953          0.002047 ...

      PROGRAM NAME   TYPE     #OCCURS   SQLSTMT   CL7  ELAP.TIME   CL7  CPU TIME   CL8  SUSP.TIME   CL8  SUSP
      SYSLH200       PACKAGE       20      8.00         0.005238        0.001978         0.003026        1.10

      REQUESTER      METH      #DDFS     TRANS     #ROLLBK     #COMMIT     SQLRECV     ROWSENT     CONVI
      9.30.112.192   DRDA         20      0.00           0          20        7.00        0.00      0.00

GRANDTOTAL          40        0     0.00     2.00     0.00     2.50          0.154204          0.006261 ...
                    40       40     0.00     0.00     0.00     1.00          0.002702          0.001835 ...

      PROGRAM NAME   TYPE     #OCCURS   SQLSTMT   CL7  ELAP.TIME   CL7  CPU TIME   CL8  SUSP.TIME   CL8  SUSP
      ALL PROG       PACKAGE       40      8.00         0.006178        0.001771         0.004228        1.05

      REQUESTER      METH      #DDFS     TRANS     #ROLLBK     #COMMIT     SQLRECV     ROWSENT     CONVI
      9.30.112.192   DRDA         40      0.00           0          40        7.50        0.00      0.00
```

Example 4:

IFC record without the client accounting API:

```
 1   LOCATION: SYPEC15A          DB2 PERFORMANCE MONITOR (V7)
GROUP: N/P                       LOCKING TRACE - DETAIL          REQUESTED FROM:       MEMBER:
N/P                                                                                    TO:
SUBSYSTEM: V71A                                                                        ACTUAL FROM:
DB2 VERSION: V7                                      SCOPE: MEMBER                     PAGE DATE:
0PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE       EVENT TIMESTAMP     --- L O C K  R E S O U R C E ---
PLANNAME CONNECT                 RELATED TIMESTAMP   EVENT     TYPE      NAME          EVENT SPECIFIC DATA

USRT001 db2jccma DRDA            00:13:57.05142568   LOCK      DATAPAGE  DB =WRKDB01   DURATION=COMMIT
USRT001 in                       00F7FA24C76F        REQUEST             OB =3         RSN CODE= 0
DISTSERV SERVER                                                          PAGE=X'000003' HASH =X'00043403'
REQLOC :9.30.112.192
ENDUSER :USRT001
WSNAME :v8ec192
TRANSACT:db2jccmain
                                 00:13:57.05298958   LOCK      DATAPAGE  DB =WRKDB01   DURATION=COMMIT
                                                     REQUEST             OB =3         RSN CODE=X'10'
                                                                         PAGE=X'000003' HASH =X'00043403'
```

IFC record with the client accounting API:

```
 1   LOCATION: SYPEC15A          DB2 PERFORMANCE MONITOR (V7)                          PAGE:
GROUP: N/P                       LOCKING TRACE - DETAIL                                REQUESTED FROM:
MEMBER: N/P                                                                            TO:
SUBSYSTEM: V71A                                                                        ACTUAL FROM:
DB2 VERSION: V7                                      SCOPE: MEMBER                     PAGE DATE:
0PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE       EVENT TIMESTAMP     --- L O C K  R E S O U R C E ---
PLANNAME CONNECT                 RELATED TIMESTAMP   EVENT     TYPE      NAME          EVENT SPECIFIC DATA

USRT001 db2jccma DRDA            00:15:36.56897636   LOCK      TABLE     DB =269       DURATION=COMMIT
USRT001 in                       00F7FA264D12        REQUEST             OB =3         RSN CODE= 0
DISTSERV SERVER                                                                        HASH =X'0001030D'
REQLOC :9.30.112.192
ENDUSER :maryela
WSNAME :9.30.129.202
TRANSACT:hr_appl1
                                 00:15:36.56913120   LOCK      DATAPAGE  B =269        DURATION=COMMIT
                                                     REQUEST             OB =3         RSN CODE= 0
                                                                         PAGE=X'000002' HASH =X'00047402'
```

Integration of the Java and DB2 application monitoring tools extends in the other direction as well, from the DB2 server to the Universal Client. DB2 Correlation IDs are created by the server and appear in the server's recovery log, the server's error message log, the server trace records, and Universal Java Client trace. The DB2 correlator is used to tie activity monitoring across DB2 and WebSphere, so that customers can easily correlate trace records, error logs, and log records across the two products.

Example 5

Correlation ID in server side main log (db2diag.log):

```
2003-04-29-12.27.43.791070  Instance:db2inst1  Node:000
PID:2706(db2agent (ICMNLSDB))  TID:8192  Appid:G916625D.NA8C.068149162729

access plan manager sqlra_sqlC_dump Probe:25  Database:ICMNLSDB
```

Correlation ID in DB2 server side trace:

```
3571   mbt_scb DB2 common communication sqlccgetapplid cei (3.3.43.10.2.1)
        pid 1188018 tid 1 cpid -1 node 0 sec 0 nsec 16431127 probe 10
        marker name: PD_SQLT_MARK_APPID
        Description: Correlator identifier (TCP/IP connection, JDBC type 4) bytes 26
        appID: G916625D.NA8C.068149162729
```

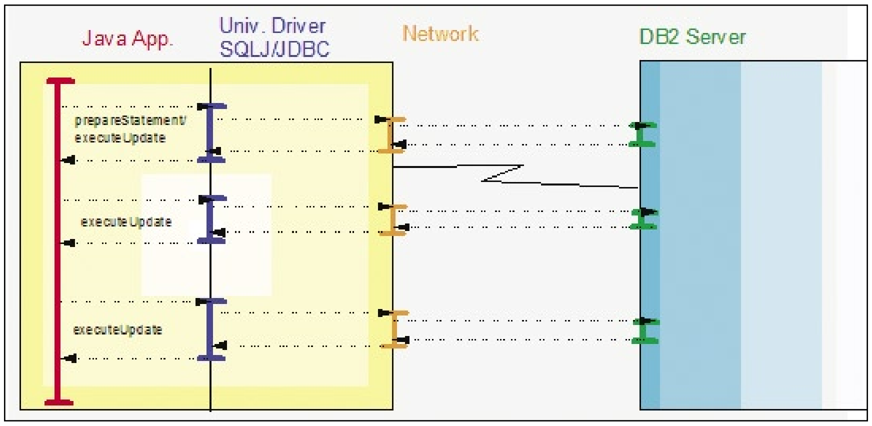
Correlation ID in Universal Driver trace:

```
[ibm][db2][jcc][time:1050540951783][thread:main][Connection@8385e3] Database product
  version: SQL08012
[ibm][db2][jcc][time:1050540951783][thread:main][Connection@8385e3] Driver name: IBM DB2
  JDBC Universal Driver
[ibm][db2][jcc][time:1050540951783][thread:main][Connection@8385e3] Driver version: 1.3.7
  Test Build
[ibm][db2][jcc][time:1050540951783][thread:main][Connection@8385e3] DB2 Correlator: G9166
  25D.NA8C.068149162729
[ibm][db2][jcc][time:1050540951783][thread:main][Connection@8385e3] END TRACE_CONNECTS
```

Finally, the DB2SystemMonitor class is a unique Java based tool for elapsed time analysis. An application programmer can use this class to measure server time, network IO time, driver time, and application time. Elapsed time analysis used to require hours, and the involvement of several specialists, for example a DBA, network system programmer, and host system programmer. The DB2SystemMonitor class allows elapsed time analysis to be performed on demand by the Java application programmer in a matter of minutes. Analysis can be done without involvement from the DBA or the system programmer, and without impacting CPU cost at the database server since most of this information is already collected by DB2 for the class(1) and class(2) accounting trace.

Example 6

```
DB2SystemMonitor monitor=
((DB2Connection)conn).getDB2SystemMonitor();
monitor.enable(true);
monitor.start(com.ibm.db2.jcc.DB2SystemMonitor.RESET_TIMES);
monitor.stop();
apptime = monitor.getApplicationTimeMillis(); // time to right of red band
drivertime = monitor.getCoreDriverTimeMicros(); // time to right of blue bands
nettime = monitor.getNetworkIOTimeMicros(); // time to right of orange bands
servertime = monitor.getServerTimeMicros(); // time to right of green bands
```



Together, these monitoring tools provide a full set of integrated methods for monitoring deployed enterprise applications . Using the Universal Client's SQLJ support can bring even more to the table.

The Importance of Consistent Performance

Once the monitoring tools have been used to build an instrumented application, the goal is to make sure the application will run consistently and reliably day after day. Coding and deploying Java Enterprise Applications in SQLJ makes this easier.

A deployed SQLJ application will run much more dependably than dynamic SQL applications, where database access paths could potentially change from day to day. An access path change could mean an outage if it happens at the wrong time to a critical SQL statement in a deployed enterprise application. SQLJ prevents this from happening because SQLJ statement access paths can be explicitly locked in and controlled through normal customer change control processes. During deployment of an SQLJ program, each SQL statement is recorded in the DB2 catalog and access paths are bound. These paths remain constant until the program package is rebound. EXPLAIN data can be saved during SQLJ program deployment to verify ahead of time which access path will be used. All of this means runtime surprises are minimized. How does SQLJ make all this possible? SQLJ allows Java programs to take advantage of DB2's support for static SQL.

All static SQL programs require more deployment steps than dynamic programs in order to achieve the advantages of static SQL. For example, a specific package must be created at the server for each program. SQLJ is no exception. To simplify the development and deployment of static SQL Java applications, the Universal Client introduces a streamlined SQLJ deployment process that creates fully portable DB2 SQLJ applications. Platform specific files such as DBRM's and .bnd files have been eliminated from the SQLJ deployment process since all necessary bind information is stored in the serialized profile. DRDA protocols are used for customization and bind steps, using the same set of pure Java tools against all platforms. The use of DRDA protocols and the Universal Java Client makes it easy to build, test, and deploy SQLJ applications across multiple platforms. For example, a developer may wish to initially build and test a new application in a workstation environment. She will then want to deploy the application remotely from her workstation to a z/OS configuration. Finally, she will want to transition smoothly to the production platform. The Universal Client will support all these steps being executed directly from her workstation if she wishes.

Using SQLJ for Enterprise Java applications not only improves reliability characteristics, it also allows the system to take advantage of static SQL security features. Static SQL allows database table privileges to be associated with individual programs, as part of a package or plan. Privileges are thereby granted to the program's author rather than to the program's user. Contrast this security mechanism to dynamic SQL, where end users must be granted table privileges. In a large enterprise, a dynamic security model could have the potential to cause security exposures. But with static SQL, users do not need general table privileges to run, they only need privileges to execute a specific program. The system has much tighter control over what users are allowed to do.

Finally, some of DB2's monitoring tools discussed earlier in this paper are even easier to use with SQLJ. SQLJ package names are distinct for each program, so programs can be identified by their package name using online monitoring tools such as DB2PM and Omegamon. With dynamic SQL, the package name is usually the same across applications, so the package name can't be used to distinguish between applications. Having a distinct package for each program means that with SQLJ it is easier to use existing DB2 tools to identify programs involved in deadlocking or time-out issues, identify the SQL statements issued by a given application program, monitor SQL activity by program, and measure program level CPU time, I/O operations, getpages and other performance characteristics that are associated with each package.

Example 7

Identification of excessive CPU consumption

PLANNAME	#OCCURS	#ROLLBK	SELECTS	INSERTS	UPDATES	DELETES	CLASS1	EL.TIME	CLASS2	EL.TIME
	#DISTRS	#COMMIT	FETCHES	OPENS	CLOSES	PREPARE	CLASS1	CPUTIME	CLASS2	CPUTIME
db2jccm	2204	56	0.03	0.07	0.01	0.03		1.748419		0.056311
	2204	2153	4.10	1.22	0.92	1.17		0.005872		0.005565

PROGRAM NAME	TYPE	#OCCURS	SQLSTMT	CL7 ELAP.TIME	CL7 CPU TIME	CL8 SUSP.TIME	CL8 SUSPI
SQLLC200	PACKAGE	136	42.47	0.250418	0.033727	0.090457	8.10
SQLLH200	PACKAGE	2014	5.98	0.035617	0.003522	0.008152	1.34
TEST402	PACKAGE	2	21.00	0.007536	0.001540	0.003806	1.50
TEST620	PACKAGE	1	78.00	2.007186	0.012467	1.629740	35.00
TEST621	**PACKAGE**	**2**	**44.50**	**1.920644**	**0.167270**	**1.226005**	**408.50**
TEST622	PACKAGE	2	73.00	0.228051	0.012666	0.125299	36.50
TEST025	PACKAGE	4	53.75	0.835342	0.009109	0.781471	12.25
TEST021	PACKAGE	4	34.75	0.025548	0.004915	0.005629	3.75
TEST051	PACKAGE	2	247.00	1.407256	0.027396	0.757469	87.50
TEST670	PACKAGE	5	15.80	0.077186	0.005786	0.033538	3.40

Other Performance Enhancements and Features: Don't Miss Out!

- The Universal Client takes advantage of several significant DRDA improvements, including support for long SQL names and statements, DRDA query block sizes up to 2 megabytes, internal DRDA performance improvements, and server supplied stored procedures for SQL error messages and database meta data. Because the Universal Java Client uses DRDA, it also makes the traditional DB2 client configuration optional, potentially saving steps in environment configuration and management. The use of DRDA protocols also greatly improves the Universal client's ability to tolerate different maintenance levels of client and DB2 server software, eliminating the prior restrictions on the client being within one release level of the server.

- Support for JDBC and SQLJ 3.0 specifications by the Universal Java Client includes savepoint support, new meta data for PreparedStatements, the return of auto generated keys, support for multiple open result sets for a single stored procedure, WITH HOLD cursors, and improved BLOB/CLOB support.
- SQLJ support has been enhanced to allow combining multiple source files into a single package on DB2. This reduces the potential proliferation of packages for SQLJ enterprise applications. Taking advantage of this and the rest of the newly reworked SQLJ in the Universal Java Client will require changes in existing procedures for SQLJ users, but thesimplified deployment process and future cross platform portability of theapplications are well worth it.
- WSAD 5.1 has complete tooling for SQLJ, and can now be used to generate SQLJ CMP beans. Static singleton selects will be generated for improved performance where possible. Also, SQLJ programs are fully supported in the WSAD workbench. WSAD has a built-in SQLJ profile translation and customization tool, SQLJ editor, and SQLJ debugger, and the workbench treats .sqlj and .ser files as first class objects.
- New SQL features are being designed and implemented specifically to enhance persistence performance for Java enterprise applications running with DB2 and WebSphere. For example, when appropriate, a FOR READ ONLY KEEP UPDATE LOCKS clause will be automatically generated by WebSphere for queries that will use pessimistic locking for searched updates against z/OS. This support allows SQLJ and JDBC to minimize network traffic and CPU cost for WebSphere persistence. The FOR READ ONLY clause allows the open, fetch and close to be performed with a single network message, but locks are still held for later entity bean updates in the same unit of work.

Summary:

Here we summarize the features available to the Java system developer who wants to use all the tools DB2 provides to create a dependably performing, manageable enterprise system.

Above all, the system developer wants to focus on creating a system that will perform consistently. For a Java application running on DB2, this need is best addressed by DB2's support for static SQL through SQLJ. Statically bound packages make it easy to validate and ensure consistent application performance in a production environment. SQLJ support in the Universal Driver has been specifically redesigned to simplify cross platform development and deployment of SQLJ applications. Support is integrated with WebSphere application development tools to put these advantages within easy reach of developers.

Just as important to the developer is the ability to easily and accurately monitor and diagnose application behavior. The Universal Java Client provides tracing facilities that simplify problem determination and make it easy to correlate activity on the server and client. The Universal Java Client, WAS, and DB2 share instrumentation and accounting information among themselves that can be tracked either by the application programmer or the system administrator. A variety of tools and options are available to simplify application monitoring that give more control in oversight of the deployed application.

Finally, a developer will want to take advantage of the various new performance features in DB2 that maximize the performance of commonly used functions in Java enterprise applications.

Conclusion:

The features discussed in this paper are for customers for whom reliability and consistent application performance are key requirements. DB2 offers an array of options that enhance the capability to create and manage dependable Java database applications. With a little knowledge and planning during development and deployment, customers writing Java applications can easily use the many diagnosis and monitoring tools available from DB2 and Websphere, and deploy a Java system with unmatched performance and reliability.

e business software

G507-1457-00