IBM Software Group

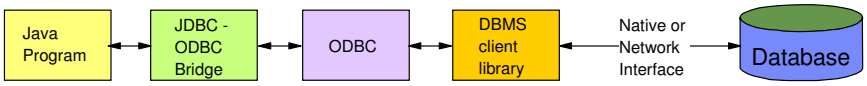
"Universal Java Driver", SQLJ and JDBC enhancements

ON DEMAND BUSINESS

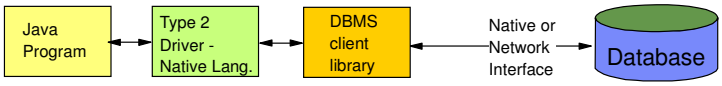
Cécile Benhamou
Technical Sales DB2 z/OS et Tools DB2
cecile_benhamou@fr.ibm.com

JDBC Driver Type

Type 1: JDBC requests are delegated to the ODBC client library

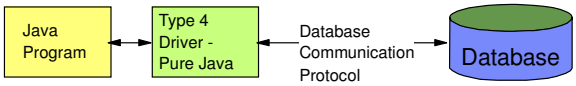


Type 2 (native API): JDBC functionality in Java, built on top of the DB2 client library



Type 3 (native communication protocol): JDBC requests are delegated to a remote JDBC server - 100% Java client library

Type 4 (database protocol): 100 % pure Java client library



JDBC Driver Type

Java defines 4 types of JDBC drivers

- Type 1 -- Implements JDBC API as a mapping to another data access API such as ODBC
  - Drivers of this type are generally dependant on a native library which limits their portability
- Type 2 -- Partly implemented in Java and uses native method calls (JNI) to call a database specific API
  - Drivers are written partly in the Java programming language and partly in native code
  - Limited portability due to native code
- Type 3 -- Uses pure Java client (100%)
  - Communicate with a middleware server using a database independant protocol
  - Middleware server then communicates the client's request to the data source
- Type 4 -- Are pure Java and implement the network protocol for a specific data source
  - The client connects directly to the data source
  - In case of DB2, the DRDA protocol is used

What is the DB2 Universal Java Client?

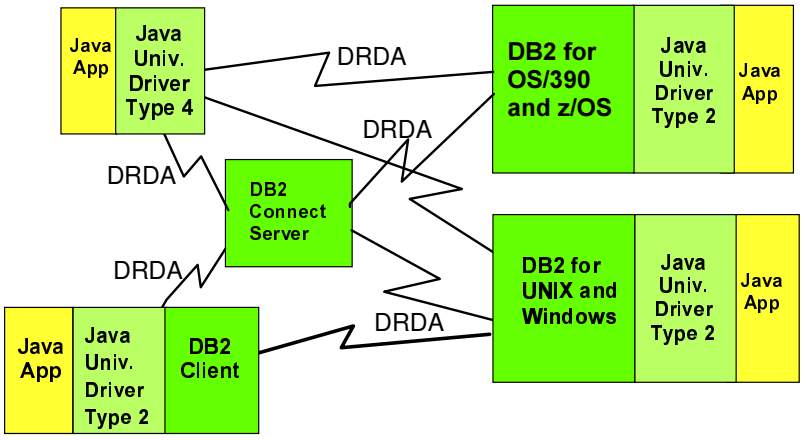
- Significant reengineering of Java support for DB2 Connect, CAE, and DB2 for OS/390 and z/OS client software
- SINGLE Driver for Unix, Windows, Unix and z/OS
- Common Architecture (Type 2 and Type 4)
  - Local (Type 2 connectivity)
  - Distributed (Type 4 connectivity)
- Uses DRDA protocols for all client communication
  - eliminates DB2RA and net driver protocols
  - much better handling of different client/server software levels
- High Performance

Type 2 and Type 4 Driver

- Functionality enhancements for Type 2 and Type 4 drivers
  - **Fully compliant with JDBC/SQLJ 3.0 standard**
    - savepoint support
    - new metadata for PreparedStatements
    - return autogenerated keys
    - multiple open ResultSets for a single stored procedure
    - WITH HOLD cursors
    - improved BLOB/CLOB support
- Provides a full Java application development process for SQLJ

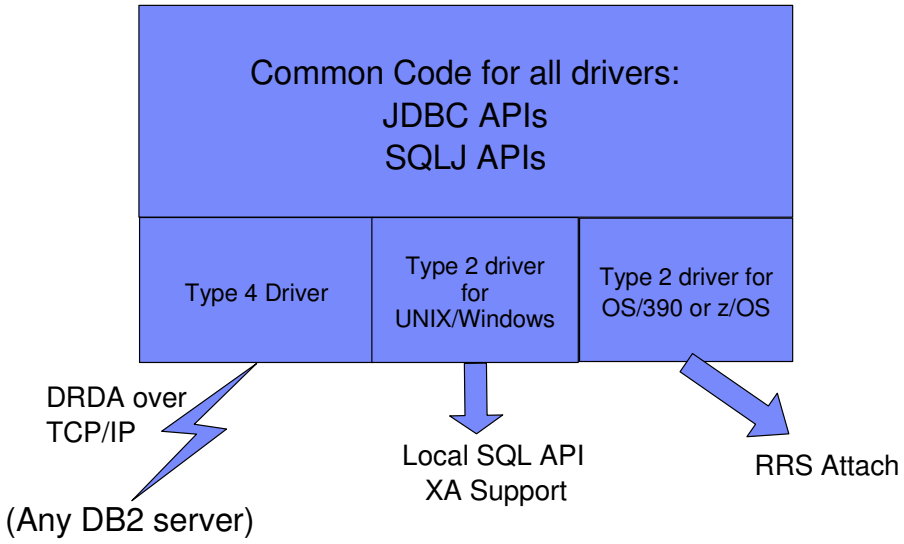
WebSphere. software

Java Universal Driver Connectivity Options

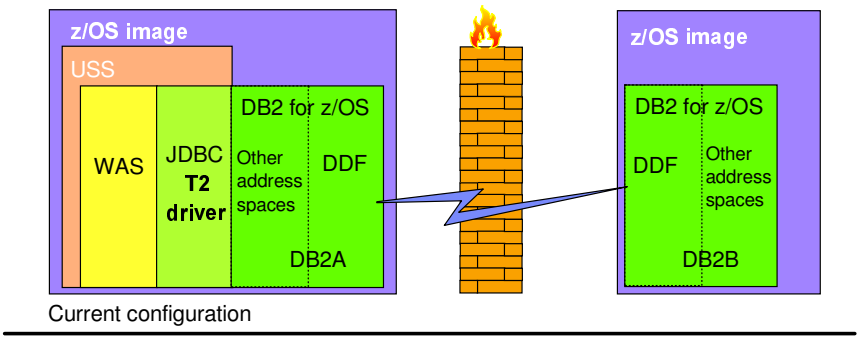


Java App Java application or WebSphere

DB2 Java Universal Client Internal Architecture

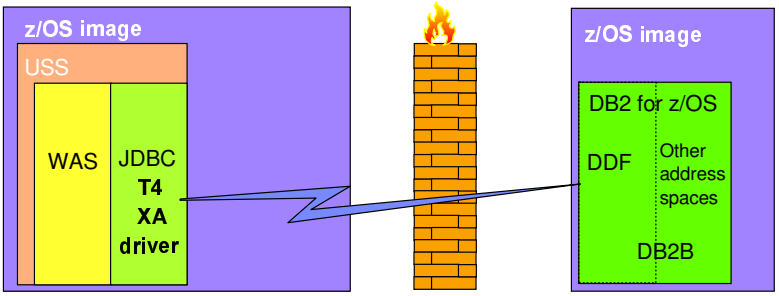


z/OS Application Connectivity to DB2 for z/OS and OS/390



Current configuration

Configuration with IBM z/OS Application Connectivity to DB2 for z/OS and OS/390



Why use SQLJ?

- Productivity
  - less code written by the application programmer
  - resulting code is easier to maintain
- Static SQL authorization model
  - provides Java with a stronger authorization model
- Static SQL performance for Java applications
  - Significant performance advantage over JDBC, even with Global Dynamic Statement Caching turned on, in DB2 Engine to avoid full prepares
- Monitoring/managability
  - static SQL packages for accounting/monitoring
  - static SQL locks in access path, so that access path changes don't occur without a conscious choice

SQLJ Application Development

- 100% Java application process
  - eliminates DBRM files and .bnd files
- New SQLJ serialized profile format
  - fully portable to all platforms -- user can deploy on any server platform without running db2profc on the target system
  - contains information needed for all BIND operations, without having to recustomize on each BIND
  - allow multiple class file to be bound into a single DB2 package
- Simplifies deployment of applications, but does require changes in existing procedures used by SQLJ users

Retrieve a single row from DB2

SQLJ:

```
#sql [con] { SELECT ADDRESS INTO :addr FROM EMP
             WHERE NAME=:name };
```

JDBC:

```
java.sql.PreparedStatement ps = con.prepareStatement(
      "SELECT ADDRESS FROM EMP WHERE NAME=?");
ps.setString(1, name);
java.sql.ResultSet names = ps.executeQuery();
names.next();
addr = names.getString(1);
names.close();
```

-- concise
-- strong typing

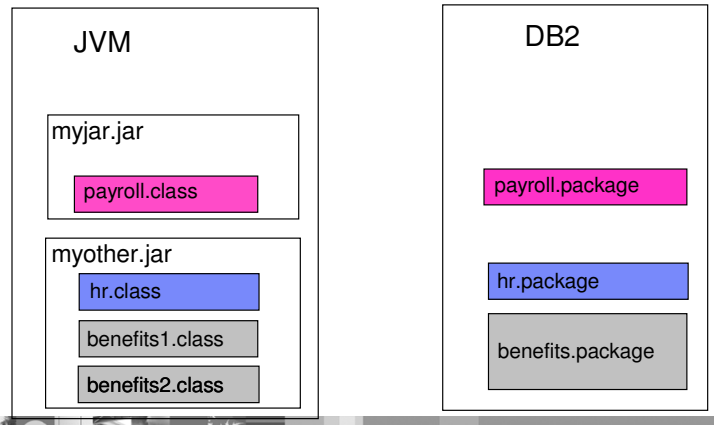
-- portable across platforms and DBMSs
-- compile/bind time schema checking
-- static SQL performance and authorization!!!

Static SQL Authorization

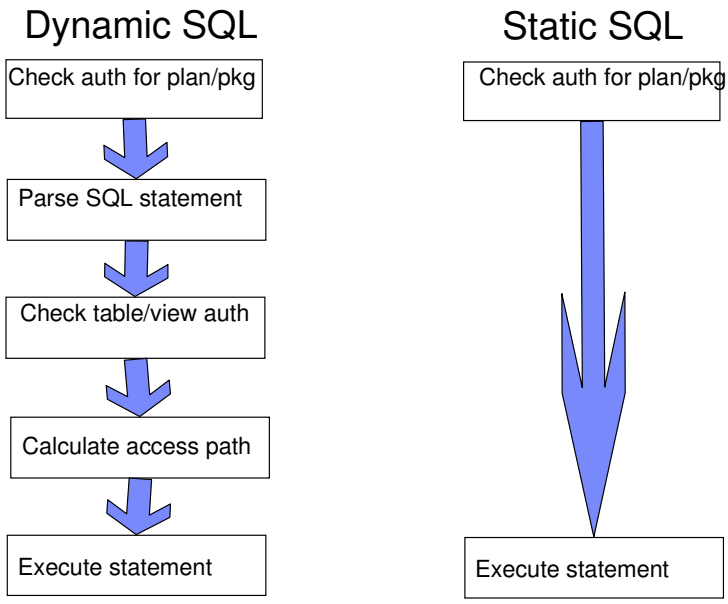
- Static SQL is associated with "program"
  - plans/packages identify "programs" to DB2
  - program author's table privileges are used
  - end users are granted EXECUTE on program
- Dynamic SQL is associated with "user"
  - no notion of "program"
  - end users must have table privileges
  - BIG PROBLEM FOR A LARGE ENTERPRISE!!!

SQLJ packages -- consistent performance

- SQL statements are recorded in the DB2 catalog
- SQL access paths are pre-bound for each SQL statement
- access paths don't change until next REBIND
- package names visible to online monitoring tools (DB2PM, etc.)
- EXPLAIN data can be saved during program deployment



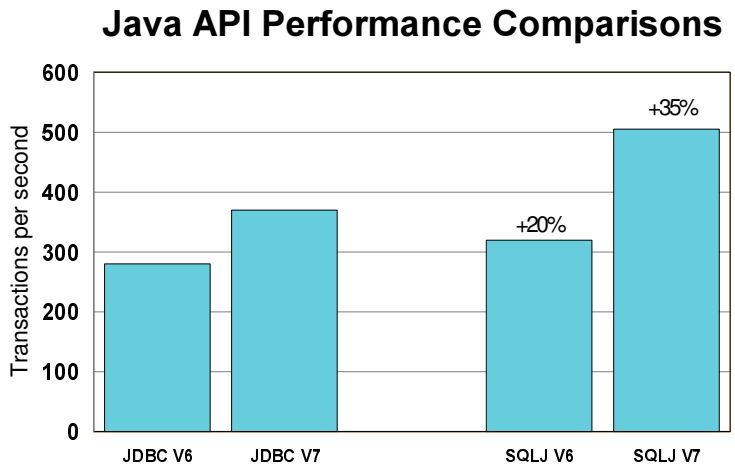
Static SQL is FASTER!!!



SQL API Comparison
Unix Client
Major Financial Firm

Driver	Elapsed Time	CPU Time (Class 1)	Network Messages
Embedded C	0.929	0.092	87
V7 JDBC T2	1.798	0.134	147
V7 SQLJ T2	6.253	0.341	611
V8 JDBC T4	2.004	0.138	147
V8 SQLJ T4	0.790	0.091	87

Java API Performance Comparisons



Normalized throughput for zSeries G7 with 3 engines with 100% cache hit for JDBC.
SQLJ advantage increased from 20% to 35% when Java overhead was reduced.

Acknowledgments

- This presentation is based on :
  - a presentation by John J Campbell
  - the following 'Redbook':
    - DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More (SG24-6079)