Java-Programme in der Oracle-Datenbank? Na klar!

Matthias Schulz, Schulz IT Services GmbH

Stored Procedures bieten eine Reihe bekannter Vorteile, aber warum sollte man Java dafür einsetzen? Der Artikel zeigt, welche Eigenschaften Java Stored Procedures haben und wie Java-Programme in die Oracle-Datenbank kommen.

Stored Procedures bieten bei datennahen Prozessen eine bessere Kapselung, höhere Datensicherheit, Trennung von Data- und Business-Logik, kürzere Programmausführungszeiten, geringere Netzwerklast und vieles mehr. In Oracle-Datenbanken sind sie meistens in der Sprache PL/SQL geschrieben.

Java Stored Procedures, also in der Datenbank gespeicherte und ausführbare Java-Programme, eignen sich zur Lösung vielfältigster Aufgaben. Das Spektrum reicht von einfachen Einzel-Klassen (wie Filehandling, Mailversand, Datei-Zipper etc.) bis hin zu großen komplexen Systemen (wie ETL-System, Workflow, Datamart-Beladung etc.).

Der Einsatz von Java zum Erstellen von Stored Procedures ermöglicht die Nutzung der unzähligen Bibliotheken und Frameworks, die für Java verfügbar sind. So bieten bereits die Bordmittel von Java einen deutlich besseren Zugriff auf Filesystem und andere Server, als dies mit PL/SQL alleine möglich wäre. Sollen weitere Frameworks und Bibliotheken verwendet werden, so lassen sich deren „.jar"-Files mit dem Tool „loadjava" (siehe unten) direkt in die Oracle-Datenbank laden.

Nicht zuletzt ermöglichen Java Stored Procedures die Wiederverwendung von bereites vorhandenen Programmen sowie Know-how und erlauben es, Programme zu erstellen, die im Gegensatz zu PL/

SQL unabhängig vom Datenbank-Hersteller sind und auch in anderen Datenbank-Systemen als Oracle eingesetzt werden können.

Sicherheit

Wie sicher und leistungsfähig sind Java Stored Procedures? Die nationalen Behörden der USA und Großbritanniens verwenden diese zur Abwicklung der Einreisekontrollen. Da ein Ausfall dieser Systeme jegliche Ein- und Ausreise zum Erliegen brächte, wurde eine Technologie gewählt, die Sicherheit, Stabilität, hohe Performance und langfristige Produktverfügbarkeit gewährleistet.

Ein einfaches Beispiel

Java-Klassen können mit dem Befehl „CREATE JAVA SOURCE" direkt als Source-Code in die Oracle-Datenbank geladen werden, ähnlich wie ein PL/SQL-Source. Die Option „AND RESOLVE" kompiliert den Source-Code und erstellt ein Java Class Object. Es kann nahezu jede Java-SE-Klasse erstellt werden, die Verwendung von Grafik- oder UI-Methoden ist jedoch nicht möglich *(siehe Listing 1)*.

Ein PL/SQL-Wrapper ermöglicht den Zugriff auf die Methoden von Java-Klassen. Jede „static"-Java-Methode kann über eine passende PL/SQL-Funktion oder Prozedur aufgerufen werden, deren Rückgabewert, Parameter-Anzahl und -Datentyp kompatibel sind *(siehe Listing 2)*.

Um die Konsolenausgabe des Beispiels sichtbar zu machen, müssen sowohl die Konsolenausgabe aktiviert als auch die Ausgabe der Java-Konsole umgeleitet werden *(siehe Listing 3)*. *Listing 4* zeigt den Aufruf der Java-Klasse mit dem PL/SQL-Wrapper. Mit „DROP JAVA SOURCE "TestClass";" löscht man das Java-Source-Objekt und die zugehörige Java-Klasse wieder aus der Datenbank.

Das „loadjava"-Tool

Mit „loadjava" *(siehe „https://docs.oracle.com/database/122/JJDEV/loadjava-tool.htm")* werden einzelne Files (Java-Source, Java-Class, Resource-File etc.) und ganze „.jar"-Files in die Oracle-Datenbank geladen. Beim „.jar"-File wird der komplette Inhalt als Objekte des aktuellen Users angelegt, wobei Package-Strukturen erhalten bleiben. Das Tool ist ein Bestandteil des Oracle-Clients, nicht jedoch des Instant-Clients. Es ist zu beachten, dass Pfad-Angaben vor dem zu ladenden Datei-Namen als Package-Informationen mit in der Datenbank abgelegt werden *(siehe Listing 5)*.

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE
NAMED"TestClass" AS
public class TestClass {
  public static void hello (String name) {
     System.out.println ("Hello " + name + "!");
  }
}
/
```

Listing 1

```
CREATE OR REPLACE PROCEDURE test_java_hello(name VAR-
CHAR2)
AS LANGUAGE JAVA NAME 'TestClass.hello(java.lang.
String)';
/
```

Listing 2

Die verwendeten Parameter haben folgende Bedeutung:

- *example.jar*
  Name des „.jar"-Files
- *thin*
  JDBC-Thin-Client verwenden
- *force*
  Vorhandene Klassen überschreiben
- *resolve*
  Sourcen sofort kompilieren
- *verbose*
  Detaillierte Ausgaben
- *user*
  Datenbank-Verbindung: Username/Passwort@Datenbank

Die Performance

Sind Java Stored Procedures schneller als Java-Programme, die außerhalb der Datenbank laufen? Ein einfaches Testprogramm, das sowohl stand-alone als auch in der Datenbank läuft, beantwortet diese Frage. Es soll folgende Funktionen ausführen:

1. Datenbank-Verbindung aufbauen
2. Tabelle „BASIC_LOB_TABLE", falls vorhanden, entfernen und erstellen
3. Einfügen von zwei Datensätzen
4. Selektieren der Datensätze und Abarbeiten des Cursors in einer Schleife
5. Lesen der Clob- und Blob-Felder der Tabelle und Ermitteln der Größen
6. Truncate auf das Clob- und das Blob-Objekt und Ermitteln der Größen

Tabelle 1 und *Abbildung 1* zeigen das Ergebnis. Betrachtet man nur die Laufzeit der DML-Operationen (Schritte 3 bis 6), so ergibt sich bei den hier verwendeten Systemen eine um den Faktor 12 schnellere Programmausführung bei der Java Stored Procedure.

Ein primärer Vorteil von Java Stored Procedures und Stored Proce-

```
-- Konsolenausgabe aktivieren:
set serveroutput on size 1000000 ;

-- Java-Ausgaben auf die Konsole umleiten:
exec dbms_java.set_output(1000000) ;
```

Listing 3

```
BEGIN
  test_java_hello('DOAG');
END;
/
```

Listing 4

```
D:\Oracle\product\12.1.0\client_1\bin\loadjava.bat
example.jar -thin -force -resolve -verbose -user MY_
USER/my_password@MYDB
```

Listing 5

Lauf	Stand-alone	JServer	schneller
1	80 ms	10 ms	8 mal
2	94 ms	8 ms	12 mal
3	93 ms	8 ms	12 mal
4	93 ms	8 ms	12 mal
5	93 ms	7 ms	13 mal

Tabelle 1: Vergleich der Laufzeiten – DML-Operationen (Schritte 3 bis 6)

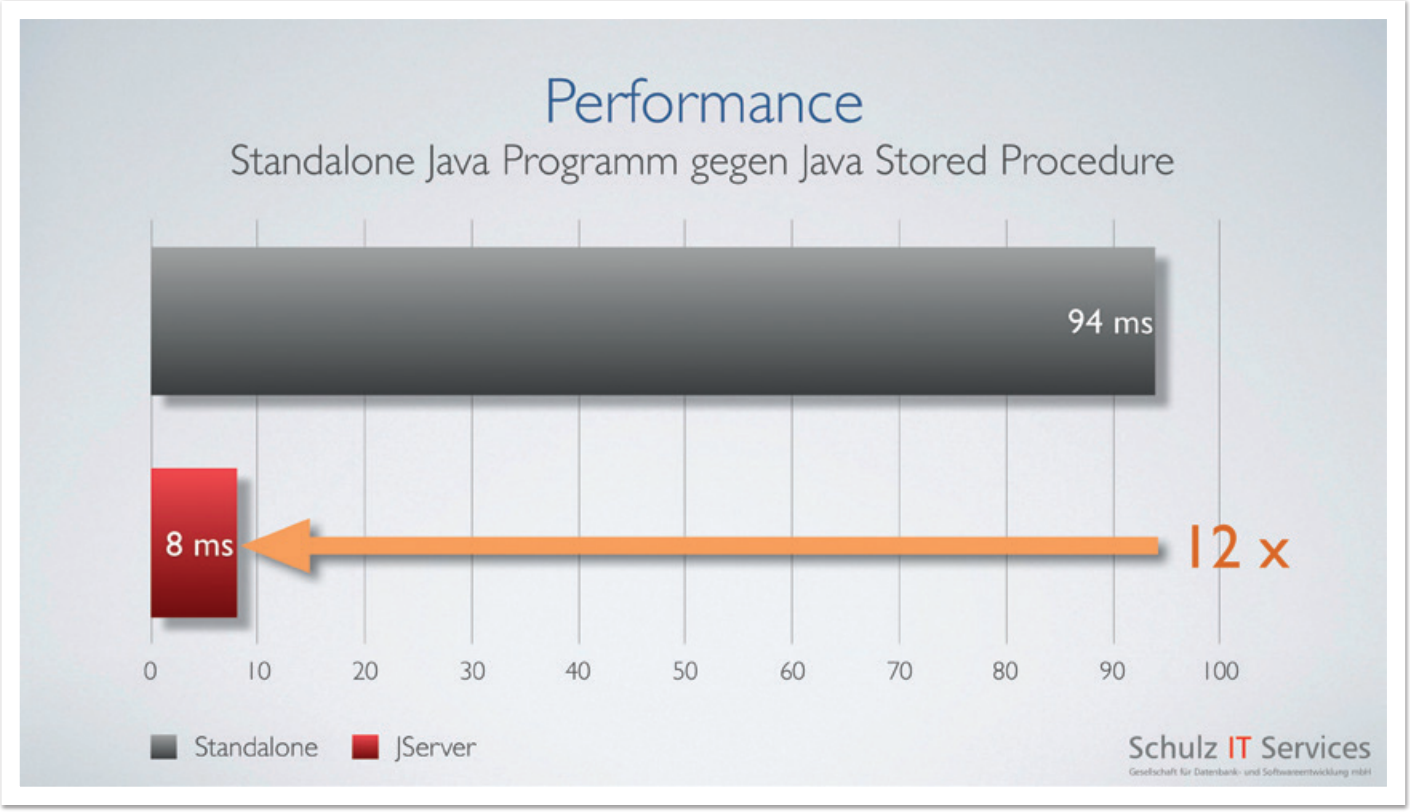

Abbildung 1: Performance – Java-Stand-alone-Programm gegen Java Stored Procedure

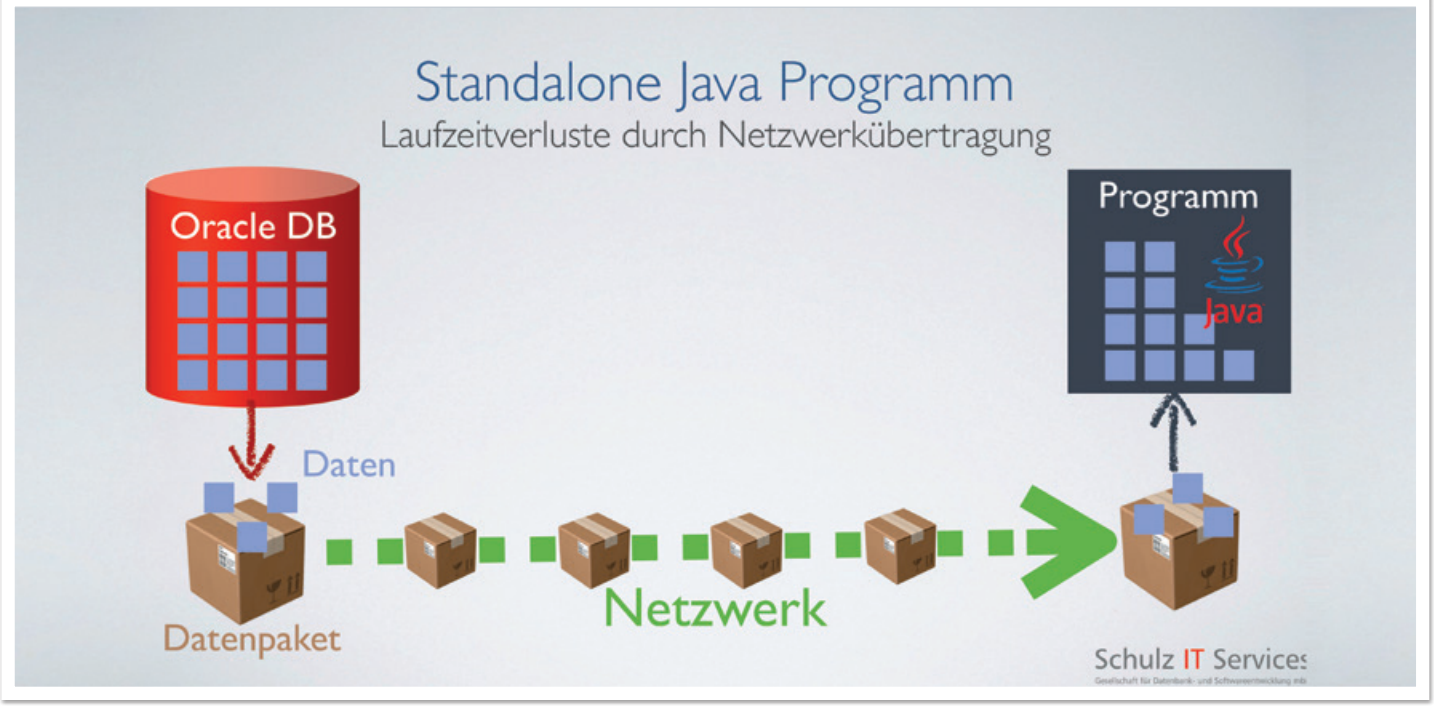

Abbildung 2: Laufzeit-Verluste durch Netzwerk-Übertragung

dures im Allgemeinen ist der Wegfall des Umwegs der Daten über das Netzwerk. Einpacken, Übertragen und Wiederauspacken der Daten erzeugt einen beachtlichen Zeitverlust, der hier einfach wegfällt *(siehe Abbildung 2)*. Der Source-Code für das Testprogramm steht unter *„https://oracledeli.wordpress.com/2017/12/23/java-stored-procedures-performance-test".*

Java-Versionen in der Oracle-Datenbank

Oracle liefert seit der Datenbank-Version 8.1.5 eine integrierte Java Virtual Machine (JVM) namens „JServer" mit aus *(siehe Tabelle 2)*. Die View „SELECT * FROM all_registry_banners;" gibt an, ob und wenn ja, welche Version des JVM installiert ist. Die installierte Java-Version erhält man mit „SELECT dbms_java.get_ojvm_property(propstring=>'java.version') AS java_version FROM DUAL;".

Für den Zugriff auf die eigene Datenbank existiert eine immer geöffnete JDBC-Verbindung „Connection con = DriverManager.getConnection(" jdbc:default:connection:");". Diese sollte nicht geschlossen werden. Auf andere Datenbanken kann, wie gewohnt, via JDBC zugegriffen werden.

Stand-alone oder JServer?

Um festzustellen, ob das Programm innerhalb einer Oracle-Datenbank (JServer) läuft, lässt sich mit „System.getProperty(„oracle.jserver.version")" eine System-Property abfragen. Liefert die Methode einen Wert zurück, läuft das Programm innerhalb einer Oracle-Datenbank, ansonsten wird „null" zurückgegeben *(siehe Listing 6)*.

Session und Multi-Threading

Alle Java Stored Procedures laufen innerhalb einer Oracle-Datenbank-Session und sind völlig voneinander isoliert, als ob jedes Programm eine eigene separate JVM und einen eigenen Garbage Collector hätte. Um Daten zwischen Programmen auszutauschen, sollten daher Alternativen wie Tabellen oder Messages (Oracle Advanced Queuing) verwendet werden. Multi-Threading-

Oracle-Datenbank Version	Java-Version Standard	Java-Version unterstützt
11g	JRE 1.5	-
11g, ab 11.2.0.4	JDK 6	-
12c	JDK 6	JDK 7
12c, ab 12.2.0.1	Java 8	Nashorn (JavaScript Engine)

Tabelle 2: Java-Versionen in der Oracle-Datenbank

```
if (System.getProperty("oracle.jserver.version") != null){
  /* program runs inside the Oracle database */
}
else{
  /* program runs as standalone */
}
```

Listing 6

```
SELECT dbms_java.longname(o.object_name) as long_ob-
ject_name,
        o.*
    FROM USER_OBJECTS o
  WHERE o.object_type LIKE 'JAVA%'
  ORDER BY dbms_java.longname(o.object_name);
```

Listing 7

```
SELECT dbms_java.longname(e.name) as long_object_name,
        e.*
    FROM USER_ERRORS e
  WHERE e.TYPE LIKE 'JAVA%'
  ORDER BY dbms_java.longname(e.name);
```

Listing 8

View	Description
USER_JAVA_ARGUMENTS	argument information of stored java class owned by the user
USER_JAVA_CLASSES	class level information of stored java class owned by the user
USER_JAVA_COMPILER_OPTIONS	native compiler options provided by the user
USER_JAVA_DERIVATIONS	this view maps java source objects and their derived java class objects and java resource objects for the java class owned by user
USER_JAVA_FIELDS	field information of stored java class owned by the user
USER_JAVA_IMPLEMENTS	interfaces implemented by the stored java class owned by user
USER_JAVA_INNERS	list of inner classes refered by the stored java class owned by user
USER_JAVA_ARGUMENTS	argument information of stored java class owned by the user
USER_JAVA_CLASSES	class level information of stored java class owned by the user
USER_JAVA_COMPILER_OPTIONS	native compiler options provided by the user
USER_JAVA_DERIVATIONS	this view maps java source objects and their derived java class objects and java resource objects for the java class owned by user
USER_JAVA_FIELDS	field information of stored java class owned by the user
USER_JAVA_IMPLEMENTS	interfaces implemented by the stored java class owned by user
USER_JAVA_INNERS	list of inner classes refered by the stored java class owned by user

Tabelle 3: Views für Java-Objekte im Data Dictionary

Programme sind zwar lauffähig, verwenden aber nur eine CPU. Sind mehrere Threads erforderlich, lässt sich dies durch Oracle-Scheduler-Jobs realisieren.

Data Dictionary

Wie bei Oracle üblich, finden sich auch sämtliche Java-Objekte im Data Dictionary *(siehe Tabelle 3)*. Die View „USER_OBJECTS“ ent-

```
SET SERVEROUTPUT ON ;

DECLARE
     p_resourcefile_name VARCHAR2(4000 CHAR) := 'messagetransmitter.properties';
     l_content           CLOB;
     l_max_chars         INTEGER := 32767;
     l_amount            INTEGER := l_max_chars;
     l_text              VARCHAR2(32767 CHAR);
BEGIN
     DBMS_LOB.createtemporary(l_content, FALSE);
     DBMS_JAVA.EXPORT_RESOURCE(p_resourcefile_name, l_content);
     DBMS_LOB.READ(l_content, l_amount, 1, l_text);
     DBMS_OUTPUT.put_line(l_text);
     IF l_amount >= l_max_chars THEN
        DBMS_OUTPUT.put_line('[...]');
     END IF;
END;
/
```

Listing 9

```
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED  "SocketTest " AS
import java.net.Socket;
public final class SocketTest
{
  public static void connectTest(final String pHost, final int pPort)
  {
     Socket socket;
     System.out.println( "testSourceSocketConnection: host:  " + pHost
               +  ", port: " + pPort);
     try {
        socket = new Socket(pHost, pPort);

        if (socket.isConnected()) {
           System.out.println( "socket.isConnected() = true\n ");
        }
        else {
           System.out.println( "socket.isConnected() = false !!!\n ");
        }
        socket.close();
     }
     catch (Exception e) {e.printStackTrace();}
  }
}
/
```

Listing 10

```
CREATE OR REPLACE PROCEDURE
  SOCKET_TEST(p_host VARCHAR2, p_port NUMBER)
AS LANGUAGE JAVA
  NAME 'SocketTest.connectTest(java.lang.String, int)';
/
```

Listing 11

```
SET SERVEROUTPUT ON ;
BEGIN
  DBMS_JAVA.set_output(1000000);
  SOCKET_TEST('myserver.mycompany.com', 61616);
END;
/
```

Listing 12

```
testSourceSocketConnection: host: myserver.mycompany.com, port:123
java.security.AccessControlException: the Permission (java.net.SocketPermission myserver.mycompany.com resolve) has
not been granted to DISTRIBUTOR_PROCESSING.
The PL/SQL to grant this is
dbms_java.grant_permission( 'DISTRIBUTOR_PROCESSING', 'SYS:java.net.SocketPermission', 'myserver.mycompany.com',
'resolve' )
  at java.security.AccessControlContext...
```

Listing 13

```
dbms_java.grant_permission( 'DISTRIBUTOR_PROCESSING', 'SYS:java.net.SocketPermission', 'myserver.mycompany.com',
'resolve' )
```

Listing 14

Thema	PL/SQL	Java	Vorteil
SQL-Befehle	•		Keine zusätzliche JDBC-Schicht
SQL-Datentypen	•		Keine Umwandlungen
Berechnungen		•	Schneller
Moderne Sprachelemente		•	Vererbung, Closures etc.
Filehandling		•	Mehr Möglichkeiten/Frameworks
Mailversand		•	Mehr Möglichkeiten/Frameworks
Zugriff auf Web-Server		•	Mehr Möglichkeiten/Frameworks
Zugriff auf andere Systeme		•	JDBC, REST etc.

Tabelle 4: Java oder PL/SQL?

hält auch alle Java-Objekte, erkennbar an einem mit „JAVA" beginnenden Object Type. Die Namen der Java-Objekte werden verkürzt im Data Dictionary abgelegt, jedoch kann mit der Funktion „dbms_java.longname" wieder der volle Name angezeigt werden *(siehe Listing 7)*. Fehlerhafte Java-Objekte finden sich in der View „USER_ERRORS" *(siehe Listing 8)*.

Um den Inhalt eines Java-Objekts, etwa den eines Resource-Files, auszugeben, dient das Skript in *Listing 9*, hier für das File „message-transmitter.properties".

Berechtigungen

Oracle sichert Java Stored Procedures sorgsam ab. Das bringt in der Praxis zunächst einige Beschränkungen mit sich und erfordert beispielsweise für Zugriffe auf das Dateisystem oder einen fremden Server die Vergabe expliziter Rechte. Dazu ein Beispiel: „SocketTest" ist ein einfaches Programm, um zu testen, ob eine Verbindung zu einem bestimmten Server mit einem bestimmten Port aufgebaut werden kann *(siehe Listing 10)*. *Listing 11* zeigt den PL/SQL-Wrapper. Wir testen nun den Zugriff auf einen Server. Die Werte für Host *(„myserver.mycompany.com")* und Port (61616) sollten durch Daten eines realen Servers ersetzt werden *(siehe Listing 12)*. *Listing 13* zeigt das Ergebnis. Wir erhalten eine Fehlermeldung, weil das Programm (noch) keine Java-Permission besitzt, um auf den gewünschten Server (hier *„myserver.mycompany.com")* zuzugreifen. Die Vergabe der fehlenden Rechte gestaltet sich bei Java Stored Procedures jedoch recht einfach, da mit der Fehlermeldung bereits ein fertiger Befehl zur Vergabe der fehlenden Rechte mitgeliefert wird *(siehe Listing 14)*.

Java oder PL/SQL?

Wann sich welche der beiden Sprachen besser eignet, hängt von verschiedenen Faktoren ab und bedarf einer individuellen Abwägung – je nach Aufgabe eignet sich entweder PL/SQL oder Java besser *(siehe Tabelle 4)*. Letztendlich sollte in die Entscheidung auch das Know-how der Entwickler, die Verfügbarkeit von entsprechenden Entwicklern im Markt sowie die Wiederverwendbarkeit der Programme eine Berücksichtigung finden. Bleibt abschließend nur noch die Frage, ob man nun Java oder PL/SQL zum Erstellen von Stored Procedures einsetzen sollte. Die beste Antwort lautet: PL/SQL und Java, je nach Aufgabe.

Weiterführende Links

- Performance Test Source Code: *https://oracledeli.wordpress.com/2017/12/23/java-stored-procedures-performance-test*
- Database Java Developer's Guide: *https://docs.oracle.com/database/122/JJDEV/toc.htm*
- The loadjava Tool: *https://docs.oracle.com/database/122/JJDEV/loadjava-tool.htm*
- DBMS_JAVA Package: *https://docs.oracle.com/database/122/JJDEV/DBMS-JAVA-package.htm*

Matthias Schulz
schulz@schulz-it-services.de

Matthias Schulz ist seit dem Jahr 2002 Geschäftsführer der Schulz IT Services GmbH und dort als Consultant, Trainer und Konferenzsprecher im Datenbank-Umfeld tätig. Seine Schwerpunkte sind Data Vault, High-End-Systeme (Exadata, Exasol) und die Oracle-Datenbank (Entwicklung, Tuning, Architektur, ETL, Testing und Java in der Datenbank). Zudem ist er Dozent für Informatik (Datenbanken) an der Dualen Hochschule Baden-Württemberg.