May 7–11, 2006

Tampa Convention Center

Tampa, Florida, USA

IDUG® 2006

North America

L02

Java UDR's: Pushing Their Limits In IDS 10

Hal Maner

M Systems International, Inc.

8 May 2006 • 01:00 p.m. – 02:10 p.m.

Platform: Informix

GoFurther

Presentation Overview

- Java in the Informix engine - introduction
- Why do this? What are the advantages?
- Java setup and configuration in IDS 10
- Demo application
- Application troubleshooting
- Application best practices
- Real world example
- Summary and conclusion

This presentation is about how to create and deploy Java applications running inside the database server. After a brief coverage of the Java environment setup in IDS 10, we will review a full Java server side application that goes far beyond a simple static function typically found in most literature. What I mean by that is we will see how to put together a UDR that is made up of multiple Java classes.

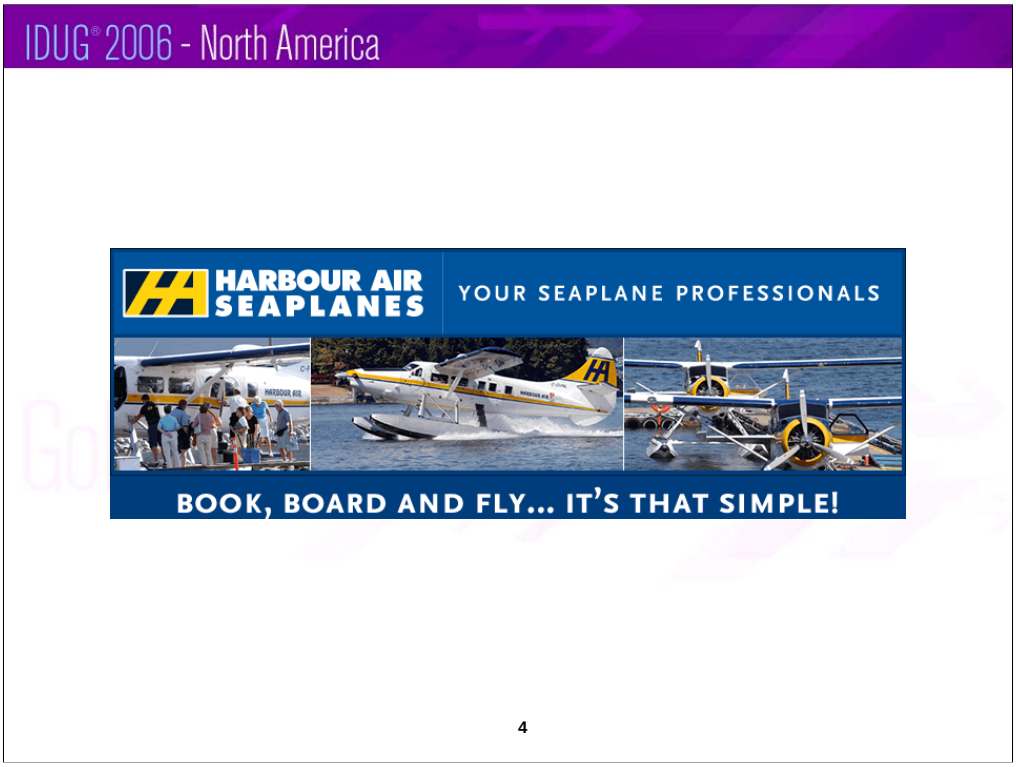
We will talk about the advantages of this approach, troubleshooting techniques, application best practices, and we will also briefly talk about an enterprise application component built this way.

Java in IDS Introduction

- Original capability came from Illustra, acquired by Informix Corporation in 1996
- Extensibility features of Illustra added to IDS 7 gave us IDS 9 (called Informix Universal Server back then)
- Now comes with IDS 9 and 10 – it is called J/Foundation
- Larry Ellison, Chairman Of Oracle, about the idea of database engine extensibility and Informix-Illustra merger in 1996: "it's like trying to integrate a boat and a plane; there is no way in hell this can be done…" (Source: "The Real Story Of Informix Software And Phil White" by Steve W. Martin)

3

Informix Dynamic Server (IDS), like in many other areas, has been the innovator and technology leader in embedding a Java Virtual Machine (JVM) technology in a database server. This powerful feature, called J/Foundation, has been part of IDS since version 9 (called Informix-Universal Server back then) was released following the 1996 acquisition of Illustra. Informix-Universal server was the industry's first "fully extensible" database. Larry Ellison, the chairman of Oracle, at the time said this is like putting together a boat and a plane – it can never be done.



Other example of boat and plane integration…

Advantages of UDR's

- Reduced load on network bandwidth
- Better application performance compared with same functionality in a thick client
- Ease of deployment
- Rich, fully object oriented, mainstream programming language
- Application layer flexibility
- Ease of function centralization when multiple applications access the same database

5

First, let's define it: A User Defined Routine (UDR) is a routine that you create and register in the system catalog tables and then invoke within an SQL statement. A UDR can be either a function (can accept arguments and can return values) or a procedure (can accept arguments but does not return values).

Some of the advantages of UDR's are:

Application speed/reduced bandwidth load: The primary reason to deploy application functionality in the form of a UDR should be to keep this functionality where it belongs, i.e. in the server. The increasing use of the client/server and especially web applications bring with them the need to minimize the network traffic between the client and the server. This is true no matter how fast the connection in between may be – the less traffic there is, the faster the application will run (and will also positively impact the performance of other applications that depend on the same "pipe" because of the reduced load on the bandwidth). Server disk I/O intensive applications are the perfect candidate. Those of you who developed or supported a thick client application that makes heavy use of cursors over a Wide Area Network (WAN) connection know how important it is to have a high speed network bandwidth and how much network traffic such an application creates. A UDR allows almost all of that to take place inside the database server, without slowing down due to network bandwidth capacity. I should mention here that I am not advocating Java UDR's as the best way of writing applications – they have a specific purpose, they are a good tool and technology to use in certain application situations (mainly, where your application development need is somewhere on the fence between writing a stored procedure and writing a client based or server based function/executable.

Ease of deployment: Server side deployment is fast and much easier when compared to client side installation. Within a few minutes and from a central location, new application functionality can be added in the form of a Java archive (jar) file.

Rich, standard full programming language: Instead of being limited to SQL or a proprietary stored procedure language, we can take advantage of a modern object-oriented standard programming language for server side routines. The code is nearly 100% portable because it is written in an industry standard programming language, Java, that work with most industry leading databases and all industry leading operating system platforms.

Application layer flexibility refers to the fact that, again, thanks to the use of an industry standard programming language, if, for whatever reason, you decide to run your code outside the server, you can do so with minimal work. This is not possible to do with a proprietary stored procedure language.

Another advantage is what I will call "function centralization". In environments where multiple applications, perhaps each written in a different language, frequently have to perform a certain task in a common way – for example roll-up (calculate) the total cost of a certain assembly. By implementing this common function as a UDR, you can enable all of these applications to use the same routine with an SQL call.

Java Setup and Configuration in IDS 10

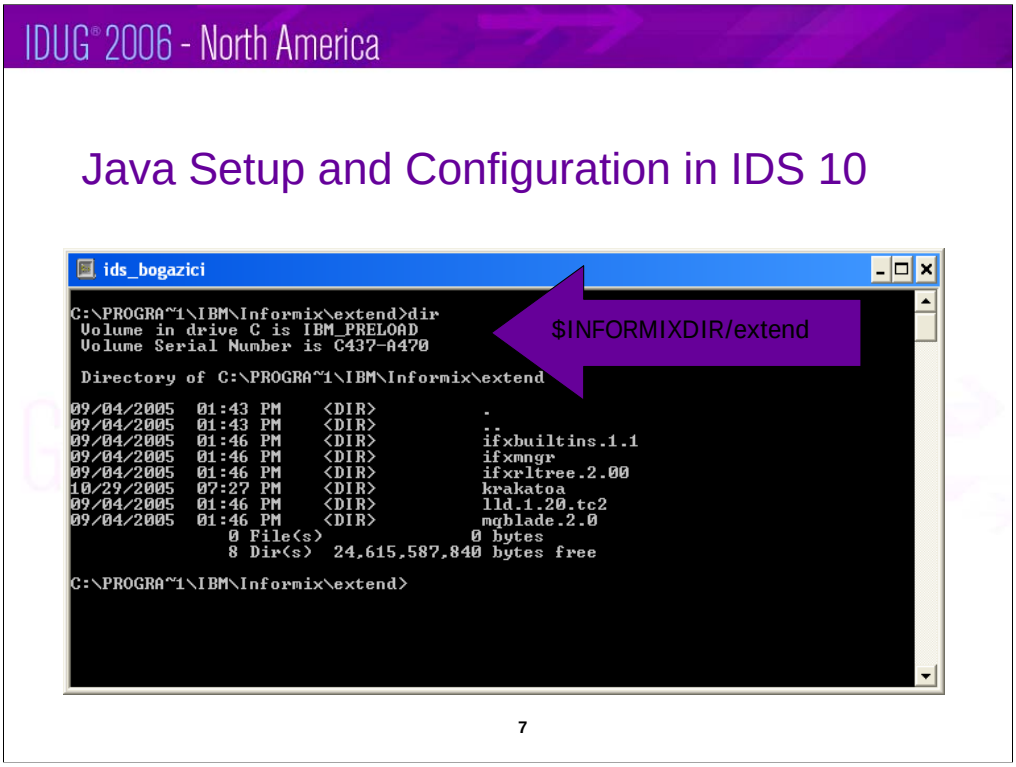
- IDS 10 supports JDK version 1.4 and JDBC version 3.0.
- JDK/JRE can be downloaded from java.sun.com
- Install the appropriate JDK or JRE on the server, as needed. IDS 10 comes with JRE version 1.4.2 (in the directory $INFORMIXDIR/extend/krakatoa/jre)

Now we will cover the configuration of Java in IDS 10.

IDS 10 supports JDK/JRE version 1.4 and JDBC 3.0. IDS 9.40 supports JDK/JRE version 1.3.

When you install IDS 10, a 1.4.2 version JRE is automatically installed for you in the extend/krakatoa/jre directory under $INFORMIXDIR. This extend/krakatoa directory is where most of the J/Foundation files are – we will need to become quite familiar with this directory and its subdirectories.

It should be noted that you can download the right JDK/JRE for you from java.sun.com if you do not want to use the JRE included with IDS. We will see how to configure this a little later in the presentation.



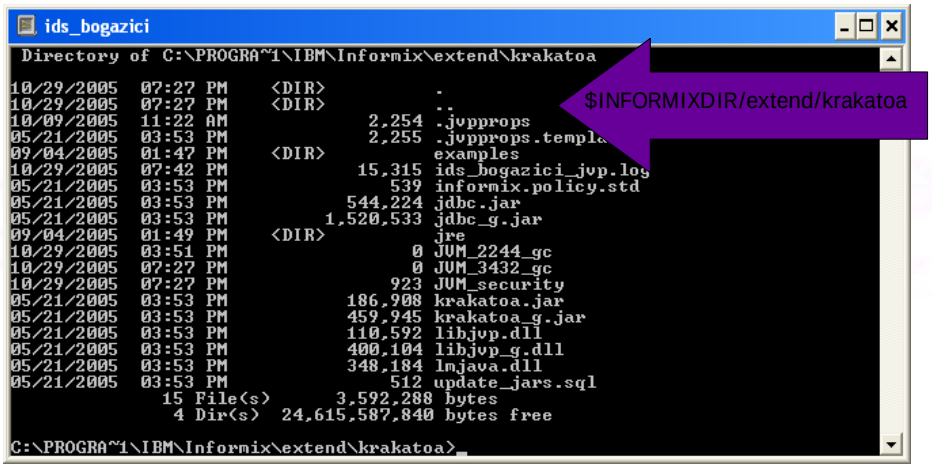
$INFORMIXDIR/extend directory.

The directory to point out here is krakatoa.

krakatoa was the code name of the product now called J/Foundation when it was first being developed in the late nineties – this name is still used in the directories, as you see.

I should mention that I normally work with this under a Sun Solaris environment – I built this demo application and its environment under Windows specifically for this presentation and I typically left things at default values, so this is why you see a long INFORMIXDIR value and my onconfig file is pretty much out of the box with the exception of J/Foundation related parameters.

Java Setup and Configuration in IDS 10



8

The jar files you see in this directory make up the heart of the J/Foundation product: jdbc.jar and krakatoa.jar. These contain the classes that allow us to run java routines in the database server. The _g versions are the debug versions of these files. The examples directory has some useful examples to help you get started. Also worth pointing out are a few text files here that we will talk about more later: .jvpprops and the …jvp.log file.

Jre is the java runtime edition that comes with the database server. When you check the java –version in the jre/bin directory here, you see that the JRE version is 1.4.2:

C:\Program Files\IBM\Informix\extend\krakatoa\jre\bin>java -version

java version "1.4.2"

Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)

Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142sr1a-20050209 (JIT enabled: jitc))

Java Setup and Configuration in IDS 10

Onconfig parameter:

SBSPACENAME sbspace1

The name of the default system sbspace.

Database server stores your java UDR jar files here.

Database server also stores your smart BLOB's here if a specific dbspace was not provided for them.

9

In our production server setups where we use UDR jar files to store here (no smart BLOBS), 50,000 pages has been an adequate size for this dbspace.

Java Setup and Configuration in IDS 10

Onconfig parameter:

VPCLASS jvp,num=1

The number of java virtual processors that the database server should start.

This is a CPU VP…

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPJAVAHOME
C:\PROGRA~1\IBM\Informix\extend\krakatoa\jre

Directory where the jre for the database server is installed.
The default is $INFORMIXDIR/extend/krakatoa/jre.

You can set this to a value for your jre. It does not have to be under $INFORMIXDIR/extend/krakatoa.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPHOME C:\PROGRA~1\IBM\Informix\extend\krakatoa

Directory where the J/Foundation classes, including JDBC classes are installed.

Default is $INFORMIXDIR/extend/krakatoa.

This is where the jdbc.jar and krakatoa.jar are installed.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPLOGFILE
C:\PROGRA~1\IBM\Informix\extend\krakatoa\jvp.log

Java trace output and stack dumps are written to this file by the database server.

You can choose the name of this file and you can also decide where it will reside. During development, depending on the tracing level you chose, this file can get quite large.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPPROPFILE
C:\PROGRA~1\IBM\Informix\extend\krakatoa\.jvpprops

This is an optional parameter. When set, it points to the path of the java virtual processor properties file.

Trace level settings, trace verbosity settings, monitor port, and a few other properties are set in this file.

JVP.monitor.port provides some additional debugging capabilities. You can query the state of the individual java vp's using a built-in function called jvpcontrol. This allows you to, for example, find out how much memory is being used by the jvp. It is recommended that you create this file – the properties you do not need can stay commented out.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JDKVERSION 1.4

This is the major JDK version supported by the database server. For IDS 10, the valid values are 1.4, 1.3, and 1.2. Unless you have a specific reason to use an older version, use 1.4.

IDS 10.00.TC3 version for Windows comes with JRE version 1.4.2.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPJAVALIB \bin\

This is the path to the JVM libraries relative to JVPJAVAHOME.

This is almost always \bin\ - leave it at default.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPJAVAVM jsig;dbgmalloc;hpi;jvm;java;net;zip;jpeg

List of the JVM libraries that the database server will load.

Unless you have very specific requirements, leave the default in the onconfig.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPCLASSPATH
C:\PROGRA~1\IBM\Informix\extend\krakatoa\krakatoa.jar;
C:\PROGRA~1\IBM\Informix\extend\krakatoa\jdbc.jar

This sets the CLASSPATH for the J/Foundation jar files (krakatoa.jar and jdbc.jar) for the database server to use during startup.

You can use the _g versions if you will be debugging, however the tracing facilities still work with the regular versions – so far we have not needed to use the _g versions. The directory path above, again, corresponds to $INFORMIXDIR/extend/krakatoa.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPARGS -Xms128m;-Xmx128m

VERY IMPORTANT!!!

This parameter is not set for you and it may not even be in the onconfig.std file, yet it is a very important parameter.

This is where you set the options for the jvm, including the amount of memory it can use.

This parameter allows you to set options for the JVM. An important option you can set here is the amount of memory to allocate to the JVM. This needs to be sufficiently large – the default is 16 MB and this is not enough for "pushing the limits". Informix tech support recommends (at least, they did at one point) a value not higher than 256 MB. You should know that the UDR will primarily consume the virtual segment of the memory.

Java Setup and Configuration in IDS 10

Onconfig parameter:

JVPARGS -Xms128m;-Xmx128m

VERY IMPORTANT!!!

The onconfig line shown above allocates 128 MB to the jvm.

If you do not set this parameter you only get 16 MB by default.

This parameter allows you to set options for the JVM. An important option you can set here is the amount of memory to allocate to the JVM. This needs to be sufficiently large – the default is 16 MB and this is not enough for “pushing the limits”. Informix tech support recommends (at least, they did at one point) a value not higher than 256 MB. You should know that the UDR will primarily consume the virtual segment of the memory.

Let's take a moment to take a peek at my onconfig file and see some of the parameters I have been talking about.

Java Setup and Configuration in IDS 10

Environment variable JAR_TEMP_PATH

Set it if you want temporary jar files to be stored somewhere other than C:\tmp (Windows) or /tmp (Unix).

If this environment variable is not set and the directories mentioned above do not exist, you will get an error when you try to install your jar file.

Under Unix you will probably not have to worry about this, but I am including it here because under Windows it sometimes causes a problem:

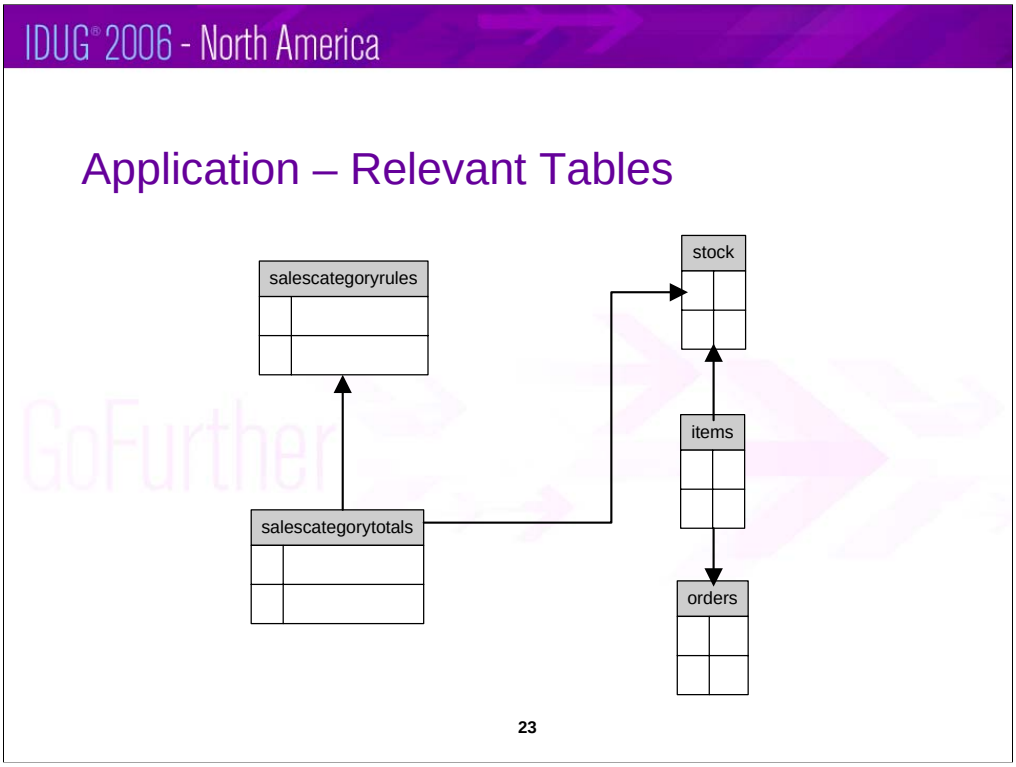
If this environment variable is not set, J/Foundation will look for C:\tmp folder under windows and /tmp under Unix. /tmp normally exists under Unix but C:\tmp does not always get created under Windows. You either need to create C:\tmp or set this environment variable.

Application - Description

- Demo application using the superstores_demo database.
- Following some database driven sales tracking rules, it checks the orders (orders and items tables) for a given date range and calculates sales metrics, writing these to a table called salescategorytotals.

A demo application will be introduced and run live here.

The business requirement is to check the sales orders for a given date range and calculate sales money totals, plus calculate a certain "sales point" totals for various categories of items. These categories and their respective points are defined in what I call a "rule table", salescategoryrules. The output is written to a different table, called salescategorytotals. The assumption here is that another application will make use of the salescategorytotals table.



The tables that are relevant to our demo are shown here.

The stock table is where the stocked items are defined. It is a part master table.

The orders and items tables are the order header and order detail tables, respectively. One orders row may have one or more items rows.

Salescategory rules table stores the… sales category rules… For example, SWIMMING is a sales category and there are entries in the salescategoryrules table for each stock number that is classified as a SWIMMING related part and a corresponding number of points.

Application – stock table

DBAccess

```
INFO - stock:_ Columns  Indexes  Privileges  References  Status  ...
Display column names and data types for a table.

-------- superstores_demo@ids_bogazici --------- Press CTRL-W for Help --------

Column name          Type                                  Nulls

stock_num            smallint                              yes
manu_code            char(3)                               yes
unit                 char(4)                               yes
description          varchar(15,0)                         yes
unit_price           money(6,2)                            yes
min_reord_qty        smallint                              yes
min_inv_qty          smallint                              yes
manu_item_num        char(20)                              yes
unit_cost            money(6,2)                            yes
status               char(1)                               yes
bin_num              integer                               yes
qty_on_hand          smallint                              yes
bigger_unit          char(4)                               yes
per_bigger_unit      smallint                              yes
```


Here is a look at the stock table – a part master table. Stock_num is the column we are interested in here.

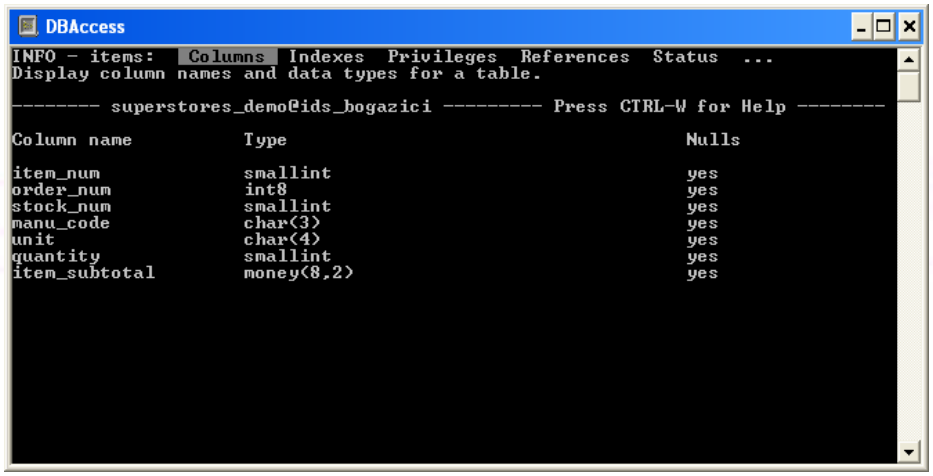
Application – orders table

DBAccess

```
INFO - orders:_ Columns  Indexes  Privileges  References  Status  ...
Display column names and data types for a table.

-------- superstores_demo@ids_bogazici --------- Press CTRL-W for Help --------

Column name          Type                                   Nulls

order_num            serial8                                no
order_date           date                                   yes
customer_num         integer                                yes
shipping             ship_t                                 yes
backlog              boolean                                yes
po_num               char(10)                               yes
paid_date            date                                   yes
```

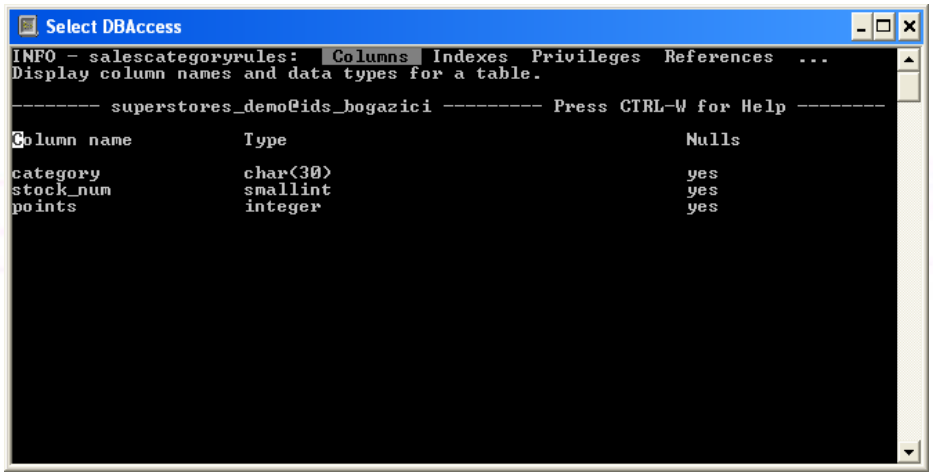

Orders table keeps order header information. The order_date is the field that will be checked against the date range in our UDR.

Application – items table



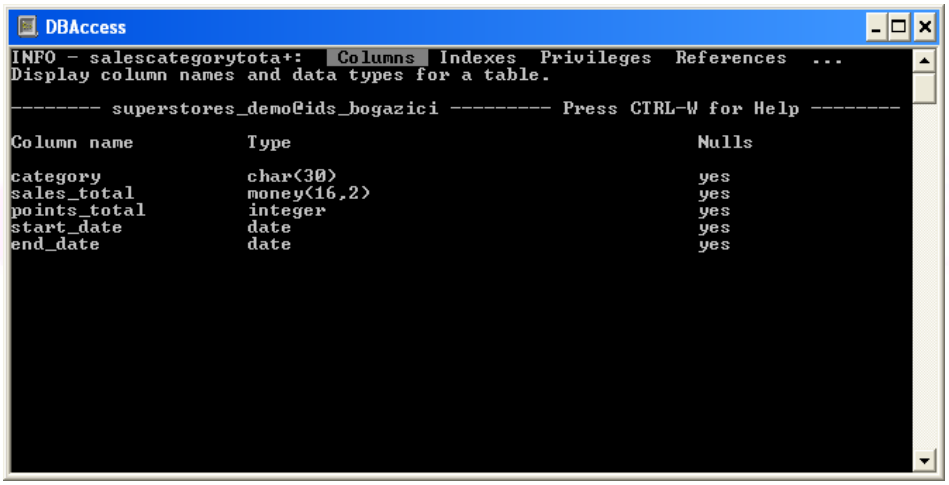
Items table is tied to the orders table by the order_num column. Along with item_num field, this makes it a unique identifier. Each items row has a stock_num column, its quantity ordered, and an item_subtotal column that is the money total for the line item.

Application – salescategoryrules table



This is the simple definition of a "rule": for the given category and stock_num, a number of points are assigned. The UDR finds every order line item this stock_num appears in that falls within the specified date range and multiplies the line item quantity by the points value to calculate the points for that line item.

Application – salescategorytotals table



The UDR writes its results into this table.

Application – Steps To Compile and Install

- Design and create your UDR (.java) classes
- Compile your classes (to .class files)
- Build your jar file to be installed in the database
- Install your jar file in the database
- You are now ready to make use of your UDR

Here are the steps: design your java classes, compile them, build a jar file with them, install the jar file in the database.

Application – UDR (.java) classes

CategoryTotalCalculator.java

```
public static int calculateCategoryTotals(String startDate, String endDate)
throws SQLException
   {
      …
      try
      {
         SuperStoresConnection superStoresConnection = new
SuperStoresConnection();
         udrConn = superStoresConnection.connectToDatabase();
         CategoryTotalGeneric categoryTotalGeneric = new
CategoryTotalGeneric(udrConn);
         traceMessage = "CategoryTotalCalculator.calculateCategoryTotals
connected" ;
         categoryTotalGeneric.printTraceMessage(traceZone,3,traceMessage);
         categoryTotalGeneric.truncateSalesCategoryTotals();
         resultCode = categoryTotalGeneric.driveCategoryTotalCalculation();
      }
      …
```

Application – UDR (.java) classes

SuperStoresConnection.java

```
/**
 * Performs the actual connection to the database
 * @return Connection object
 */
   public static Connection connectToDatabase() throws SQLException
   {
      try
      {
         url = "jdbc:informix-direct:";
         databaseName = "superstores_demo";
         connectionParams = "";
         driverToUse = "com.informix.jdbc.IfxDriver";
         Class.forName(driverToUse);
         udrConn =
DriverManager.getConnection(url+databaseName+connectionParams);
         udrConn.setAutoCommit(false);
      }
     …
```

Application – Compiling

comp_java.bat

```
erase *.class
javac -source 1.4 -target 1.4 CategoryTotalCalculator.java
```

Application – Building Your jar File

buildcatgcalcjar.bat

```
erase catgcalc.jar
jar cvf catgcalc.jar *.class
```

Application – Installing Your jar File

execinstjar.sql

```
BEGIN WORK;
EXECUTE PROCEDURE sqlj.install_jar
('file:\MyDocuments\JavaUDRPaper\UDR\catgcalc.jar',
'catgcalc_jar',0);
CREATE FUNCTION calculateCategoryTotals(
   startDate CHAR(10),endDate CHAR(10)
   )
   RETURNS INTEGER
   EXTERNAL NAME
'catgcalc_jar:CategoryTotalCalculator.calculateCategoryTot
als'
   LANGUAGE JAVA;
COMMIT WORK;
```

Application – Running

runudr.sql

```
EXECUTE FUNCTION calculateCategoryTotals('01/01/2005','12/31/2005');
```

Application - Demo

Application demo:

Application Troubleshooting

Tracing

- UDR tracing can generate messages written to the JVP log file specified by the JVPLOGFILE onconfig parameter.
- com.informix.udr.UDRTraceable is the name of the java interface for tracing.
- UDR Tracing uses “trace zones” - this is a “logical” concept where you can name multiple zones in one UDR or one zone for all your UDR’s.
- Within each zone, you can have six devels of detail, from zero (off) to five (superfine i.e. most detailed).

Application Troubleshooting

Tracing …continued…

In order to implement tracing in your code:

Import the com.informix.udr package and set your trace zones:

```
import com.informix.udr.*;
…
private static String traceZone = "DemoZone";
…
```


The com.informix.udr package provides you with the necessary classes for tracing, among other things. The trace zones can be named according to your application tracing needs. You can setup different tracing zones as your application requires, or simply set one trace zone.

Application Troubleshooting

Tracing …continued…

In order to implement tracing in your code:

Add code in your UDR to set and send messages.

```
traceMessage = “This message should show in the trace log" ;
traceZoneNumber = 3; // will only print if level is 3 or higher
printTraceMessage(traceZoneName,traceZoneNumber,traceMessage);
```

Application Troubleshooting

Tracing …continued…

In order to implement tracing in your code:

Write a generic method to write the messages.

```
private UDRTraceable traceable;
…
public void printTraceMessage(String traceZone, int traceLevel,
String traceMessage)
{
   traceable.tracePrint(traceZone,traceLevel,traceMessage);
}
```


This method can be called from anywhere in your application (as long as you have access to the object, that is) and can send a trace message to the requested zone and level.

Application Best Practices

Most important: DO NOT FORGET TO CLOSE YOUR ResultSet AND Statement OBJECTS!!!

… otherwise, you will run into problems and you may even cause IDS to crash…

Closing the result set and statement objects are very important in Java in general, but especially important with the UDR's. The increased risk with the UDR's is that you can bring the whole database server to its knees. To do this, next slide…

Application Best Practices

Use the close() method for your ResultSet and Statement objects.

```
public void closeSalesCategoriesResultSet() throws SQLException
{
   try
   {
      salesCategoriesResultSet.close();
      salesCategoriesStatement.close();
   }
   catch (SQLException ex)
   {
      ex.printStackTrace();
      throw ex;
   }
}
```


ResultSet and Statement classes in java have a close() method. This is what you need to use, as soon as you are done with that result set or statement in your code, for example:

Application Best Practices

If you are calling your UDR from Informix-4GL, be sure to FREE the statement, otherwise the virtual memory segment will not clear properly – for example, use:

```
LET udr_statement_string = "EXECUTE FUNCTION calculateCost(?)"
PREPARE udr_statement FROM udr_statement_string
EXECUTE udr_statement USING order_number INTO udr_result_code
FREE udr_statement
```


As you see in this slide, not only should you be very careful about cleaning up after yourself (i.e. closing result sets and statement objects) but also should you be careful about freeing your statements if you are calling UDR routines with them. Now – you may have seen in the related literature or in other presentations that Java is "safe" – in fact some of them may sound like all memory issues of C have been cured in Java… The reality is that most of this is true and Java is a great choice as a programming language but nothing should be taken for granted. It is a known fact that database related objects, when left open instead of being closed at the appropriate time, have the potential to cause memory problems. This is true not only with Java UDR's but regular Java applications as well. The UDR's, as I mentioned before, have the potential to do more damage though, since the whole engine is vulnerable…

Real World Example of Java UDR's

- Rules based configuration engines running as UDR's.
- Global Fortune 500 company.
- Serving a geographically dispersed user base of approximately 100 in North America over a WAN connection.
- Servers currently running Solaris 10 and IDS 10 on Sun Fire V240 with two 1 Ghz processors and 2 GB of RAM.

All this sounds good but does it really work? Has it been proven, you may ask…

An industrial company in the Global Fortune 500 had a need for product (engineering) configurators. The user base was in all regions of the United States, accessing one server in the East Coast and one in Midwest. The connections are across a wide area network, so the network bandwidth is an important parameter. Without completely re-architecting the solution, adding extensilibity to the application and encapsulating the complicated, database intensive business rules and logic on the server is made possible with the UDR technology.

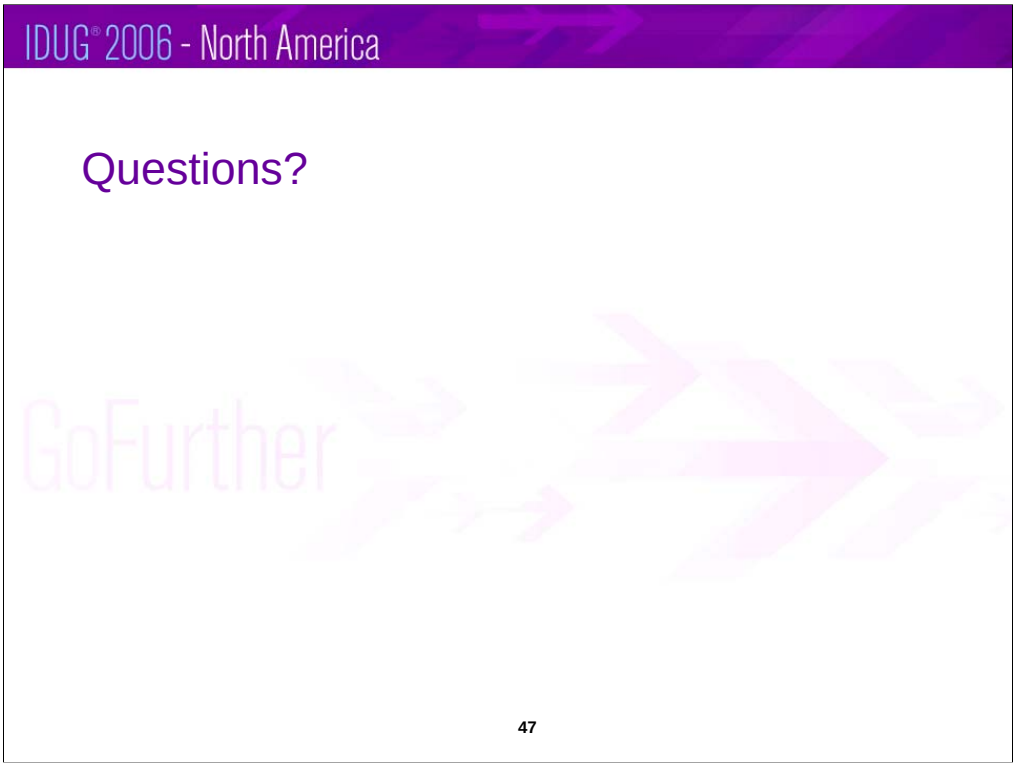
Possibilities are endless – you can write a UDR that sends an e-mail, updates a website, or do anything on the server with a single SQL command from the client.

Summary and Conclusion

- Java UDR's are a good option to consider when you have to create server based, database intensive applications.
- Reduced network traffic, use of a portable, fully object oriented language are some of the advantages of Java UDR's.
- Setting up J/Foundation in IDS requires the availability of a Java virtual machine and the configuration of several onconfig parameters.

Summary and Conclusion

- It is possible to create simple, single class UDR's as well as more complicated ones, consisting of multiple classes and an object-oriented application.
- J/Foundation provides zone and level based tracing capabilities useful for debugging during development and troubleshooting during production.
- You need to be careful and meticulous about closing and freeing your statements and result sets in Java UDR's in order to avoid memory problems.



As mentioned before, slides will be provided on the Forum website.

Also, to those who are interested, I will be happy to e-mail you a zip file with all the scripts and code that were covered in the presentation. Give me a business card with your e-mail address on it, please.

Session L02

Java UDR's: Pushing Their Limits In IDS 10

Hal Maner

M Systems International, Inc.

hmaner@msystemsintl.com