Accesssing External Databases

From ILE RPG (with help from Java)

Presented by

Scott Klement

http://www.scottklement.com

"There are 10 types of people in the world.
Those who understand binary, and those who don't."

Objectives Of This Session

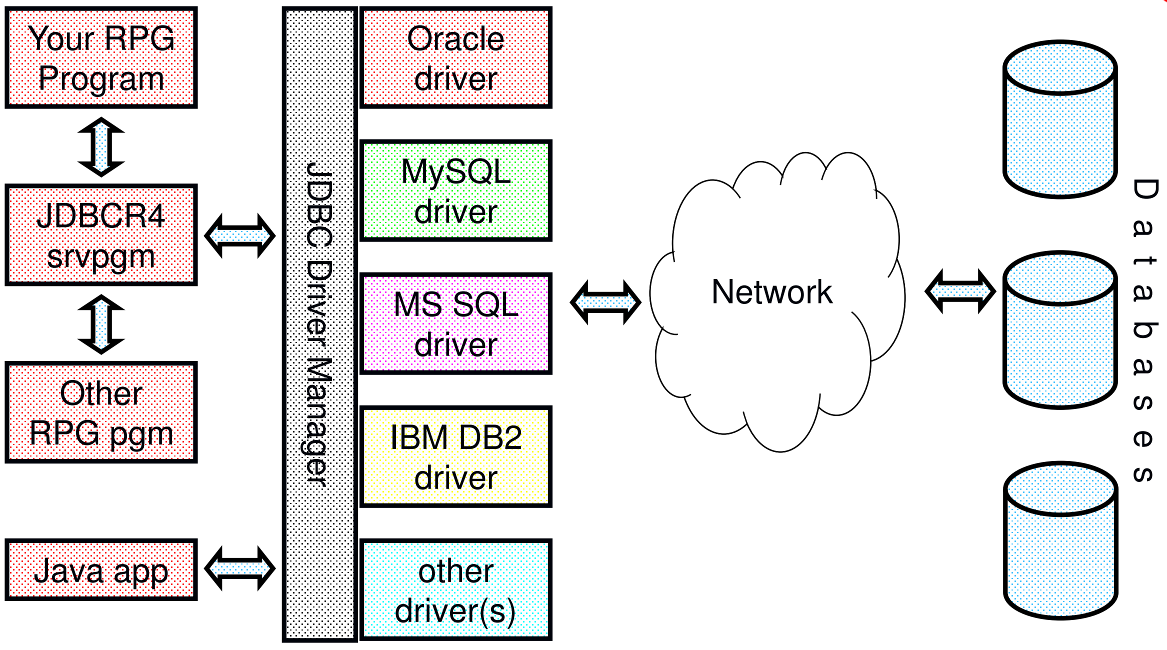
- Understand why you should use JDBC drivers from RPG.
- Understand how to install the JDBC drivers
- Understand how to use the JDBC drivers from RPG
- Learn where to find more information

Note: JDBC provides SQL access to a database. If you don't have at least a basic understanding of SQL, this session might be hard to follow.

A Solution to a Problem

- Virtually all businesses today use more than one computer platform. Unlike 20 years ago, no company is "AS/400 only".
- A Windows programmer can access any database on any platform
  - Database manufacturer provides a "driver"
  - Install driver into ODBC framework to enable programs to access the database.
- Few database manufacturers make drivers for IBM i
  - Market is too small?
  - RPG programmers haven't expressed enough interest?
- Manufacturers make drivers for Java – and Java runs on any platform!
- RPG can call Java methods directly on V5R1 and later.
  - So RPG can use Java's drivers – enabling access to just about any database on the market!

Component Overview



JDBC Drivers Provide

- JDBC = Java Data Base Connectivity
- Provide a means for Java (and RPG!) code to access a database
- Access is done through SQL statements
- SQL statements can do most anything:
    - Read data bases (SELECT statement)
    - Update (UPDATE statement)
    - Add new records (INSERT statement)
    - Create new databases, tables, indexes, views (CREATE statements)
    - Etc.
- This is done through calls to the JDBC drivers (not via RPG's normal "embedded SQL preprocessor").
- Scott has provided JDBCR4, an RPG wrapper to simplify calling JDBC.

Drivers You Can Use

- Driver must be "type 4", which means it's pure Java
    - Other drivers work by calling a Windows DLL, which will not work.
    - Type 4 is pure Java – so will run on all platforms.

Oracle refers to their Type 4 driver as a "thin" driver.

MySQL refers to theirs as "Connector/J".

Note: Although this presentation gives examples for MS SQL Server, Oracle, MySQL and IBM DB2, it should work with any database, as long as you can find a type 4 driver and figure out the correct connection string.

Install Into Your System i

- JDBC type 4 drivers are Java classes. They are almost always packaged in a JAR file.
- The vendor often puts the JAR file inside a ZIP or EXE file along with other stuff such as documentation, the license agreement, etc.

1. Download/Unzip/Install the vendor's package on your PC.
2. Upload the JAR file (or files) to the IFS on your System i.
   - MySQL: mysql-connector-j-8.0.33-bin.jar
   - MariaDB: mariadb-java-client-3.1.4.jar
   - Oracle: ojdbc8.jar
   - SQL Server: jtds-1.3.1.jar
   - IBM DB2 for i: jt400.jar
3. Add the JAR file (using the full IFS path name) to your CLASSPATH.
4. When RPG calls Java, the Java database manager will use the CLASSPATH to find the driver.

Example of Installing JDBC driver

- Create an IFS folder to store my JDBC drivers.

```
CRTDIR DIR('/java') DTAAUT(*RX) OBJAUT(*NONE)
CRTDIR DIR('/java/jdbc') DTAAUT(*RX) OBJAUT(*NONE)
```

- Download the SQL server driver for jTDS (highly recommended over Microsoft's own driver -- download links are at the end of the presentation)
- Under Windows double-click the .ZIP to unzip it. Tell it to unzip to the C:\JTDS folder.
- Use FTP in BINARY mode to copy the jtds-1.3.1.jar file from the `C:\JTDS` folder to the `/java/jdbc` folder in my IFS.
- Set my CLASSPATH as follows:

```
ADDENVVAR ENVVAR(CLASSPATH) VALUE('/java/jdbc/jdts-1.3.1.jar')
```

  - ***CLASSPATH must be set before JVM is loaded.***
  - ***Do it after a fresh sign off/on.***

Needed Information

Information You'll Need:

- Fully-qualified Java class name of driver.

```
SQL Server:  net.sourceforge.jtds.jdbc.Driver
Oracle:      oracle.jdbc.OracleDriver
MySQL:       com.mysql.jdbc.Driver
MariaDB:     org.mariadb.jdbc.Driver
DB2 for i:   com.ibm.as400.access.AS400JDBCDriver
```

- Connection String

```
SQL Server:  jdbc:jtds:sqlserver://myserver.example.com:1433
Oracle:      jdbc:oracle:thin:@myserver.example.com:1521:myDataBase
MySQL:       jdbc:mysql://myserver.example.com/myDataBase
MariaDB:     jdbc:mariadb://myserver.example.com/myDataBase
DB2 for i:   jdbc:as400://myserver.example.com
```

- Any needed properties
  - Usually a username & password.
  - Sometimes other attributes (*SYS vs *SQL, How errors are reported, etc.)

Getting Needed Info for Other Drivers

If you need to use a different JDBC driver than the ones I've listed here, how do you know what the class name and connection string should be?

The easiest solution is to look at sample Java code that uses the driver. This'll be included in the driver's documentation, or else by searching the web.

Class name can be found in a statement like one of these:

```
Class.forName("com.ibm.as400.access.AS400JDBCDriver")
  -- OR --
DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
```

The connection string will be in a DriverManager.getConnection call:

```
conn = DriverManager.getConnection("jdbc:db2://example.com:50000/phonedb" . . .
```

Introducing JDBCR4

- JDBCR4 is an RPG service program that Scott wrote to simplify the task of calling JDBC from RPG.
- It was originally written for articles about JDBC in the System iNetwork Programming Tips newsletter.
- Links to the articles where you can download the code (for free) are provided at the end of this presentation.
- The RPG sample code in this article will use this service program.
- You could certainly call the Java methods without using this service program (but why??)
  - Write your own prototypes
  - Write your own routines to convert between Java & RPG data types.

RPG Connect w/Properties

```
/copy JDBC_H

dcl-S userid  char(50);
dcl-S passwrd char(50);
dcl-S conn    like(Connection);
dcl-S prop    like(Properties);

userid  = 'scott';
passwrd = 'bigboy';

prop = JDBC_Properties();
JDBC_setProp(prop: 'user'           : %trim(userid) );
JDBC_setProp(prop: 'password'       : %trim(passwrd));
JDBC_setProp(prop: 'connectTimeout': '60'          );

conn = JDBC_ConnProp( 'org.mariadb.jdbc.Driver'
                    : 'jdbc:mariadb://www.scottklement.com/mysql'
                    : prop);
JDBC_freeProp(prop);

if (conn = *NULL);
  snd-msg *escape 'Unable to connect to MYSQL database!';
endif;

JDBC_close(conn);
```

Another Properties Example

```
/copy JDBC_H

dcl-S userid  char(50);
dcl-S passwrd char(50);
dcl-S conn    like(Connection);
dcl-S prop    like(Properties);

prop = JDBC_Properties();
JDBC_setProp(prop: 'user'     : 'scott'     );
JDBC_setProp(prop: 'password': 'bigboy'    );
JDBC_setProp(prop: 'prompt'   : 'false'     );
JDBC_setProp(prop: 'errors'   : 'full'      );
JDBC_setProp(prop: 'naming'   : 'system'    );
JDBC_setProp(prop: 'libraries':'*LIBL,ISNMAG');

conn = JDBC_ConnProp( 'com.ibm.as400.access.AS400JDBCDriver'
                    : 'jdbc:as400://localhost'
                    : prop );
```

Types of Java SQL Statements

- Immediate Statements
  SQL string is interpreted by database, and then run immediately.
- Prepared Statements
  SQL string is "compiled" by database.
  Statement can then be run multiple times without re-compiling.
  You can fill-in placeholders with values before statement is run.
- Callable statements
  Very much like a prepared statement, except that it calls a stored procedure.

---- *types of statements used with the above methods* ----

- "Query" Statements
  Statements that return a "Result Set" (very much like a cursor, except it contains meta-information about columns in the result set.)
- "Update" Statements
  Statements that do not return a result set. Name is not quite accurate, you can use this for anything that doesn't return a result set, including DDL, INSERT, UPDATE, etc.
- "Call" Statements

Routines For Immediate Statements

- JDBC_ExecUpd(connection : sql statement string)
  Run an "update" statement (one that does not return a result set).
  Wrapper for the Java executeUpdate() method.

  Returns the number of rows affected
  or 0 for statements that don't affect rows (such as "create table")
  or -1 if an error occurs.

- JDBC_ExecQry(connection : sql statement string)
  Run a "query" statement (one that returns a result set).
  Wrapper for the Java executeQuery() method.

  Returns a ResultSet object. (like a cursor – used to retrieve results of statement)
  Or *NULL if an error occurs.

Immediate "Update" Example

```
rc = JDBC_ExecUpd( conn : 'Create Table Item_Information'
                        + '('
                        + '  ItemNo      Dec(5,0) Not Null, '
                        + '  Count       Int Not Null, '
                        + '  LastChg     Timestamp '
                        + '       Default CURRENT_TIMESTAMP, '
                        + '  LastSold    Date Default Null, '
                        + '  TimeTest    Time Default Null, '
                        + '  Price       Dec(7,2) Not Null, '
                        + '  Description VarChar(25) not Null '
                        + ')' );
if (rc < 0);
  snd-msg *escape 'Unable to CREATE table';
endif;
```

NOTE: SQL Statements should be in the syntax of the target database. This lets you take advantage of any extensions they have to the SQL standard. JDBC does have a tool called "escaping" that can help make your statements database-neutral. See links at the end of this talk for more about escaping.

Working With a Result Set

- JDBC_ExecQry(connection : sql statement string)
  Run a "query" statement (one that returns a result set).
  Wrapper for the Java executeQuery() method.
  Returns a result set, or *NULL upon error.

--- when working with result sets, use the following ---

JDBC_nextRow(ResultSet)
- advances to the next available row in a result set.
- Returns *ON if successful, *OFF if you've reached the end of the result set.

JDBC_getCol(ResultSet : ColNo)
- Returns the value of a column in the result set.
- Column is identified by ordinal number. (first col returned is number 1, second is number 2, etc.)

JDBC_getColByName(ResultSet : ColumnName)
- Returns the value of a column in the result set using the column name.

JDBC_freeResult(ResultSet)
- Closes the ResultSet (like closing a cursor in embedded SQL)
- Frees up the memory used by the result set.

Immediate "Query" Example

```
dcl-s ResSet  like(ResultSet);

      ... connect as before ...

ResSet = JDBC_ExecQry( conn : 'Select Department, +
                                       Employee_No, +
                                       Employee_Name +
                                 from Employee_Master +
                                 order by Department' );
if (ResSet = *null);
  snd-msg 'Error running SELECT statement';
endif;

dow JDBC_nextRow(ResSet);
  Dept   = JDBC_getCol(ResSet: 1);
  EmpNo  = %int(JDBC_getCol(ResSet: 2));
  Name   = JDBC_getCol(ResSet: 3);
    ... do something with Dept, EmpNo & Name (print them?) ...
enddo;

JDBC_freeResult(ResSet);
```

Java will convert columns to character, as needed. You can use RPG to convert it back, as needed.

Prepared Statements

- JDBC_PrepStmt(connection : SQL statement string)
  - Returns a PreparedStatement Java object for the given SQL statement.
  - It "prepares" the SQL statement.
  - I like to think of this as "compiling" the statement, so that the code in the statement can be run again and again quickly.
  - Placeholders (called "parameter markers") can represent variable values, letting you re-run a statement without having to prepare a new statement.
  - The "parameter markers" also help you avoid "SQL Injection Attacks"

JDBC_ExecPrepUpd(PreparedStatement)

- Runs a prepared statement that does not return a result set.

JDBC_ExecPrepQry(PreparedStatement)

- Runs a prepared statement that returns a result set

JDBC_FreePrepStmt(PreparedStatement)

- Frees up the memory used by a Prepared Statement

Prepared Statement Query

```
dcl-s stmt    like(PreparedStatement);
dcl-s ResSet  like(ResultSet);

          . . .  Connect as before . . .

stmt = JDBC_PrepStmt( conn : 'Select Department, +
                                     Employee_No, +
                                     Employee_Name +
                                from Employee_Master +
                              order by Department' );
if ( stmt = *null );
  snd-msg 'Unable to create prepared statement';
endif;

ResSet = JDBC_ExecPrepQry( stmt );
if (ResSet = *null);
  snd-msg 'Unable to execute prepared statement';
endif;

         . . .   Read the Result Set The Same Way You Did with
                 an immediate statement . . .

JDBC_freeResult(ResSet);
JDBC_freePrepStmt(stmt);
```

Parameter Markers

You place a ? where you want data inserted.   Then you number the markers from left to right, and set them by number by calling the following routines:

- JDBC_setString(stmt : parameter number : 'String Value');
- JDBC_setInt(stmt : parameter number : integer value);
- JDBC_setDouble(stmt : parameter number : floating point value);
- JDBC_setDecimal(stmt : parameter number : decimal number);
- JDBC_setDate(stmt : parameter number : date field);
- JDBC_setTime(stmt : parameter number : time field);
- JDBC_setTimestamp(stmt : parameter number : timestamp field);

Parameter Marker Example

```
stmt = JDBC_PrepStmt( conn : 'Select Department, +
                                     Employee_Name +
                                from Employee_Master +
                               where Employee_No=?' );
if ( stmt = *null );
  snd-msg 'Unable to prepare statement';
endif;

EmpNo = 1004;
JDBC_SetInt( stmt: 1: EmpNo );

ResSet = JDBC_ExecPrepQry( Stmt );
if (ResSet = *null);
  snd-msg 'Unable to execute statement';
endif;

          . . .   Read the Result Set The Same Way You Did with
                  an immediate statement . . .

JDBC_freeResult(ResSet);
JDBC_freePrepStmt(stmt);
```

Prepared Statement Insert

```
stmt = JDBC_PrepStmt( conn : 'Insert Into Employee_Master +
                                (Employee_No, +
                                 Employee_Name, +
                                 Department ) +
                                 Values (?, ?, ?)' );
if ( stmt = *null );
  snd-msg 'Prepare statement failed';
endif;

JDBC_setInt   ( stmt: 1: 4321 );
JDBC_setString( stmt: 2: 'Klement, Scott C.');
JDBC_setString( stmt: 3: 'IT' );
if JDBC_execPrepUpd( stmt ) < 0;
  snd-msg 'Execute for 4321 failed';
endif;

EmpNo = 4322;
Name  = 'John Q. Public';
Dept  = 'AP';
JDBC_setInt( stmt: 1: EmpNo );
JDBC_setString( stmt: 2: Name );
JDBC_setString( stmt: 3: Dept );
JDBC_execPrepUpd( stmt );
```

You can use literals, constants or values

Wrapped In a Procedure

```
WriteRec( stmt: 1234: 'Alex Aguilera': 'SH' );
WriteRec( stmt: 1012: 'Jerry Berry'  : 'PK' );
WriteRec( stmt: 2001: 'Paul Smith'   : 'SA' );
JDBC_FreePrepStmt( stmt );
```

```
dcl-proc WriteRec;
  dcl-pi *n ind;
    stmt                 like(PreparedStatement);
    EmpNo packed(4: 0) value;
    Name  char(30)       const;
    Dept  char(2)        const;
  end-pi;

  JDBC_setInt   ( stmt: 1: EmpNo );
  JDBC_setString( stmt: 2: Name  );
  JDBC_SetString( stmt: 3: Dept  );

  if JDBC_ExecPrepUpd( stmt ) < 0;
    return *OFF;
  else;
    return *ON;
  endif;
end-proc;
```

I often like to wrap my inserts or updates into subprocedures so I can call them in a manner that's more like RPG's native I/O.

Running Callable Statements

Callable statements are very much like Prepared Statements that return result sets. The only real difference is that they call routines instead of accessing databases.

Like all SQL executed through JDBC, the syntax of the SQL statements varies from one platform to the next.

JDBC_PrepCall(Connection : Call Statement String)

Prepares a callable statement.

JDBC_RegisterOutParameter(CallableStatement: Parm No: DataType)

Notifies JDBC that one of the parameters will be used to return data from the stored procedure. By default, all parameters are input only.

JDBC_ExecCall(CallableStatement)

Execute a callable statement

JDBC_FreeCallStmt(CallableStatement)

Free up the memory used by a callable statement.

Results from Callable Statements

Stored procedures can return an update count (like an "update" statement) or they can return one or more result sets. The return value from JDBC_ExecCall() will be *ON if a result set is returned, or *OFF otherwise.

JDBC_getUpdateCount(CallableStatement)

When no result set is returned, this gives you a count of the number of affected rows (ala executeUpdate).

JDBC_getResultSet(CallableStatement)

When one (or more) result sets are returned, this gives you the result set. It returns a ResultSet object (ala executeQuery).

JDBC_getMoreResults(CallableStatement)

Advances to the next result set if more than one were returned. Returns *ON if another result set is found, *OFF otherwise – also closes ResultSet.

JDBC_getString(), JDBC_getInt(), JDBC_getShort(), JDBC_getBoolean()

Get the values of output parameters passed from the stored procedure.

Stored Procedure Example

```
dcl-s stmt        like(CallableStatement);
dcl-s rs          like(ResultSet);
dcl-s IsResultSet ind;

     ... Connect as before, etc....

stmt = JDBC_PrepCall( conn
                    : 'call order_new(012001)' );

IsResultSet = JDBC_execCall( stmt );

dow IsResultSet;
  rs = JDBC_getResultSet( stmt );

  dow JDBC_nextRow(rs);
    field1 = JDBC_getCol(rs: 1);
    field2 = JDBC_getColByName(rs: 'SHIPNAME');
     ... do something with the data...
  enddo;

  IsResultSet = JDBC_getMoreResults( stmt );
enddo;

JDBC_FreeCallStmt(stmt);
JDBC_Close(conn);
```

Result Set Meta-Data

This neat feature of JDBC lets you get information about the result set that was returned from an SQL statement, such as:

- *Number of columns*
- *Name, Data Type, Size, Decimal Positions of each column*

Useful for writing "dynamic" applications where the data that's returned might not be the same each time.

- Generic report program (user feeds an SQL statement, and you print a report.)
- Stored procedures that can return different result sets

TIP: WORKS GREAT FOR TESTING THE EXTERNAL STORED PROCEDURES YOU'VE WRITTEN IN RPG!

Meta-Data Routines

JDBC_getMetaData(ResultSet)
    Retrieve the meta data from a result set. This meta data object is used with the following routines.

JDBC_getColCount(MetaData)
    Returns the number of columns in the result set.

JDBC_getColName(MetaData : Col No)
    Returns the name of one of the columns in the result set.

JDBC_getColDspSize(MetaData: Col No)
    Returns the size of a column (intended for display) in the result set.

JDBC_getColTypName(MetaData: Col No)
    Returns the data type of a column in the result set.

Meta-Data Example (1 of 4)

```
**free
ctl-opt dftactgrp(*no) bnddir('JDBC') option(*srcstmt);

/copy JDBC_H

dcl-S conn        like(Connection);
dcl-S prop        like(Properties);
dcl-s rs          like(ResultSet);
dcl-s msg         varchar(100);
dcl-s stmt        like(CallableStatement);
dcl-s rsmd        like(ResultSetMetaData);
dcl-s IsResultSet ind;
dcl-s x           int(10);


prop = JDBC_Properties();
JDBC_setProp(prop: 'user'     : 'scott'    );
JDBC_setProp(prop: 'password': 'bigboy'    );
JDBC_setProp(prop: 'prompt'   : 'false'    );
JDBC_setProp(prop: 'errors'   : 'full'     );
JDBC_setProp(prop: 'naming'   : 'system'   );
JDBC_setProp(prop: 'libraries':'*LIBL,ISNMAG');
```

Meta-Data Example (2 of 4)

```
conn = JDBC_ConnProp( 'com.ibm.as400.access.AS400JDBCDriver'
                    : 'jdbc:as400://localhost'
                    : prop );
JDBC_freeProp(prop);

if (conn = *NULL);
  snd-msg *escape 'Unable to connect to local database!';
endif;

stmt = JDBC_PrepCall( conn
                    : 'call order_new(012001)' );
if ( stmt = *null );
  snd-msg 'Prepare statement failed';
endif;

IsResultSet = JDBC_execCall( stmt );
```

NOTE: If you replace JDBC_PrepCall with JDBC_PrepStmt, you could use the same logic to run a regular SQL statement – such as a SELECT.

Meta-Data Example (3 of 4)

```
dow IsResultSet;

  rs = JDBC_getResultSet( stmt );
  rsmd = JDBC_getMetaData(rs);

  dow JDBC_nextRow(rs);
    for x = 1 to JDBC_getColCount(rsmd);
      msg = JDBC_getColName(rsmd: x)
          + '='
          + JDBC_getCol(rs: x);
      snd-msg msg;
    endfor;
  enddo;

  IsResultSet = JDBC_getMoreResults( stmt );

enddo;

JDBC_FreeCallStmt(stmt);
JDBC_Close(conn);
*inlr = *on;
```

Meta-Data Example (4 of 4)

This sample program just outputs the column names & their values with SND-MSG, so they'll appear in my job log, as follows:

```
ORDERNO=A0000026
CUSTNO=12001
SCAC=UPSN
SHIPNAME=Scott Klement
SHIPADDR1=System iNEWS Magazine
SHIPADDR2=321 Sesame St.
SHIPADDR3=Franklin, WI  53132
BILLNAME=Wayne Madden
BILLADDR1=Penton Technology Media
BILLADDR2=123 29th St.
BILLADDR3=Loveland, CO.
SHIPDATE=2023-05-18
MSGID=
MSG=
```

Miscellaneous

Support for Nulls

- To support null fields in databases, the JDBC_getCol() and JDBC_setXXX procedures have an optional parameter of type "named indicator".
- If you pass this parameter when reading a column, it'll be turned on if the field is set to null in the databaase.
- If you pass this parameter when setting the value for a column, then the field will be marked as null if the indicator is on, not-null, otherwise.

Commitment Control

- In addition to the other procedures mentioned, JDBCR4 has procedures for commitment control.
- **JDBC_Commit()** commits transactions to disk.
- **JDBC_Rollback()** rolls the values back.

Miscellaneous

All of the previous examples in this presentation assumed you were using EBCDIC values. However, there are also corresponding routines to work with Unicode values, which allow data from other character sets aside from your job CCSID.

They work the same as the corresponding EBCDIC statements, only difference is the data is in Unicode (UTF-16)

Support for Unicode
- **JDBC_getColC** = Get column value (by column number)
- **JDBC_getColByNameC** = Get column value (by column name)
- **JDBC_setStringC** = set parameter marker to Unicode string
- **JDBC_PrepStmtC** = prepare statement from Unicode string
- **JDBC_PrepCallC** = prepare callable statement from Unicode string

Links to JDBC Drivers

Oracle Driver
https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html

MariaDB Driver:
https://mariadb.com/kb/en/installing-mariadb-connectorj/
https://mariadb.com/kb/en/about-mariadb-connector-j/

jTDS Open Source driver for MS SQL Server
http://jtds.sourceforge.net
(Microsoft also makes a driver, but it is not recommended. Far too many people have reported problems with it.)

MySQL Connector/J Driver:
http://www.mysql.com/products/connector/j/

IBM DB2 Driver for IBM i (JTOpen)
http://jt400.sourceforge.net

More Information

Scott's web page contains the JDBCR4 RPG interface, sample code, and the handout for this talk.
http://www.scottklement.com/jdbc/

Note: Scott wrote many articles related to these tools on iProDeveloper.com. Unfortunately, they removed these articles when they decided to exit the IBM i market. The titles are listed on Scott's site, so you can search for them in the Internet Wayback Machine (Internet Archive) if you need them.

This Presentation

You can download a PDF copy of this presentation from:

http://www.scottklement.com/presentations/

Thank you!