

# Database – Selecting Rows

## Overview

In this tutorial we'll cover using SQLXML to query a database table for rows. This tutorial expands on basic Console topography and configuration, so users should be familiar with those concepts.

## Steps

We'll start by preparing a database for this and other SQLXML-related exercises. You can use any database that is accessible via JDBC drivers, though we'll be using "H2" since it is extremely simple to get up and running and requires no configuration. If you do not have H2 installed, you can find it here: <http://www.h2database.com/html/main.html>.

H2 is included with your distribution of Console. To launch it run the following command (where <install\_folder> should be C:\Program Files (x86)\PilotFish\eiConsole):

```
java -jar <install_folder>/runtime/lib/h2*.jar
```

You can also just execute the JAR file directly. Either way you should see the following icon in your task tray:



A browser window should pop open which looks like this:

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:tcp://localhost/~f

User Name: sa

Password:

Connect Test Connection

We can provide values for the user name and password if we wish to change them, but for this tutorial we'll leave them at their defaults (“sa” and blank). The one field we'll need to modify is the “JDBC URL,” which is the unique URL and name for our database instance. We'll create one called tutorial:

jdbc:h2:tcp://localhost/~/tutorial

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:tcp://localhost/~tutorial

User Name: sa

Password:

Connect Test Connection

Click “Connect” to launch the main H2 screen:

Auto commit Max rows: 1000 Auto complete Normal

jdbc:h2:tcp://localhost/~tutorial

Run (Ctrl+Enter) Clear SQL statement:

INFORMATION\_SCHEMA

Users

H2 1.3.167 (2012-05-23)

**Important Commands**

- Displays this Help Page
- Shows the Command History
- Executes the current SQL statement
- Disconnects from the database

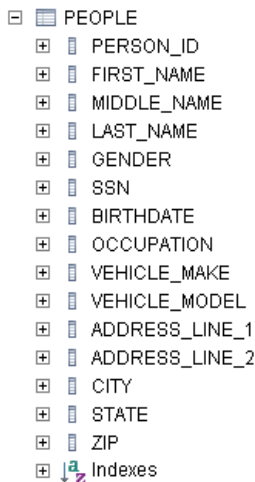
**Sample SQL Script**

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

We can enter SQL commands into the text area and then click the “Run (Ctrl + Enter)” button to execute them. To start, we'll use the content of the “create\_people.sql” file to create a “PEOPLE” table:

```
CREATE TABLE PEOPLE(  
    PERSON_ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    FIRST_NAME VARCHAR(100),  
    MIDDLE_NAME VARCHAR(100),  
    LAST_NAME VARCHAR(100),  
    GENDER VARCHAR(20),  
    SSN VARCHAR(15),  
    BIRTHDATE VARCHAR(15),  
    OCCUPATION VARCHAR(100),  
    VEHICLE_MAKE VARCHAR(20),  
    VEHICLE_MODEL VARCHAR(20),  
    ADDRESS_LINE_1 VARCHAR(100),  
    ADDRESS_LINE_2 VARCHAR(100),  
    CITY VARCHAR(40),  
    STATE VARCHAR(4),  
    ZIP VARCHAR(10)  
);
```

Once executed, we should have a table definition visible:



The screenshot shows a database tool interface. On the left, there is a tree view with a folder icon and the text 'PEOPLE'. To the right of this, a list of table columns is displayed, each preceded by a small icon consisting of a plus sign and a vertical bar. The columns are: PERSON\_ID, FIRST\_NAME, MIDDLE\_NAME, LAST\_NAME, GENDER, SSN, BIRTHDATE, OCCUPATION, VEHICLE\_MAKE, VEHICLE\_MODEL, ADDRESS\_LINE\_1, ADDRESS\_LINE\_2, CITY, STATE, and ZIP. At the bottom of the list, there is an icon with a plus sign and the text 'Indexes'.

Now we'll wish to populate our table with PEOPLE entries. The file "insert\_people.sql" contains 100 such entries. Copy its contents to the text area and execute it. Afterward, run this SQL command to verify the output:

```
SELECT * FROM PEOPLE
```

You should get ~100 rows back:

SELECT * FROM PEOPLE;									
PERSON_ID	FIRST_NAME	MIDDLE_NAME	LAST_NAME	GENDER	SSN	BIRTHDATE	OCCUPATION	VEHICLE_MAKE	VEHICLE
1	Cinderella	B	Jacquiline	Female	408-13-7134	1-22-1964	Long distance operator	BUICK	PARK AV
2	Conrad	P	Freddie	Male	754-66-6371	3-2-1946	De-icer element winder	ISUZU	TROOPE
3	Anita	R	Hedy	Female	536-22-0745	9-17-1934	Sales, automobile leasing	JEEP	COMMAN
4	Marcelino	W	Johnathon	Male	523-25-6908	1-25-1979	Supervisor, fabric coating	FORD	FIESTA
5	Robby	E	Justin	Male	539-65-1181	11-26-1979	Design analyst	DODGE	LANCER
6	Kristopher	U	Whitney	Male	573-32-	3-17-1977	Cooker cleaner	MINI	COOPEF

We'll now want to copy a few pieces of information for subsequent use: the URL, driver, username, and password. Copy these to a simple text editor for reference. Here are our values:

Username: sa

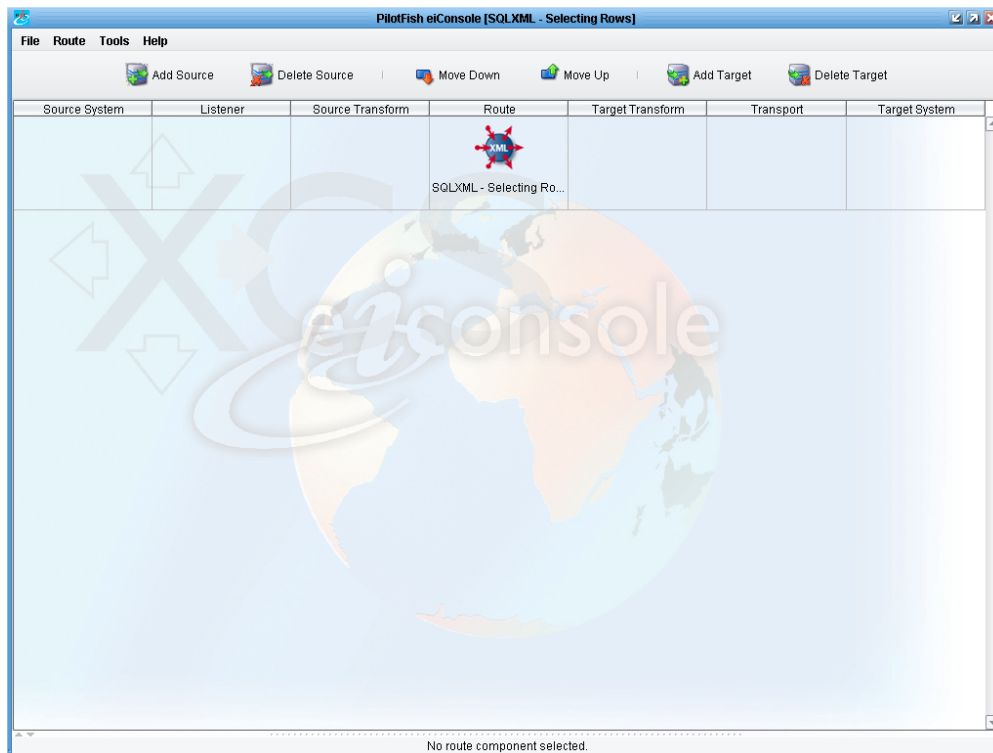
Password:

Driver: org.h2.Driver

URL: jdbc:h2:tcp://localhost/~/tutorial

You can find the URL at the top left of the H2 browser.

We'll now create a new Route in the Console called "Database – Selecting Rows":



To interact with our database, we have a few standard components available to us: the “Database Polling SQL Listener,” the “Database SQL Transformation Module,” and the “Database SQL Transport.” Each of these uses a PilotFish language called “SQLXML,” which can be summarized as an XML-based, database agnostic set of SQL instructions.

Each module can be configured using an “input file” referencing a file containing SQLXML or, if this is not provided, by assuming that the contents of any transaction reaching that module is SQLXML. In either case, the SQLXML received will be interpreted and executed against a configured database connection, which can be a “Data Source” or JDBC-driven.

We'll start by configuring a basic “Database Polling SQL Listener.” Add a new Source and select that Listener from the Listener Type drop-down:

Listener Configuration Processor Configuration

**Listener Configuration**

Listener Name:

Listener Type:

Listener Description:

Connection Scheduling JDBC Props

☒ Basic ☐ Advanced

Polling interval:  seconds

Input file:

Start by changing to the Connection tab and providing our previously copied values:

Listener Configuration

Listener Name:

Listener Type:

Listener Description:

Basic Advanced

Connection Scheduling JDBC Props

User name:

Password:

Type: ☐ DataSource ☒ JDBC Connection

DataSource name:

JDBC driver:

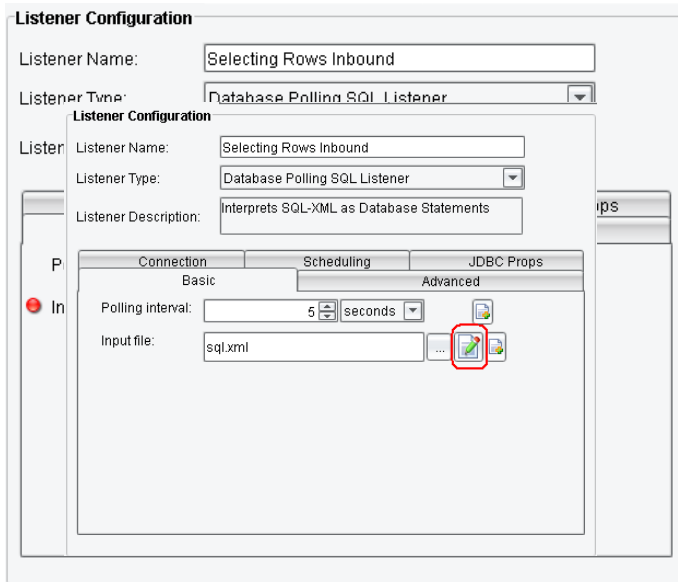
JDBC URL:

Click the “Test Connection” to ensure that our values are correct and that our database is functioning:

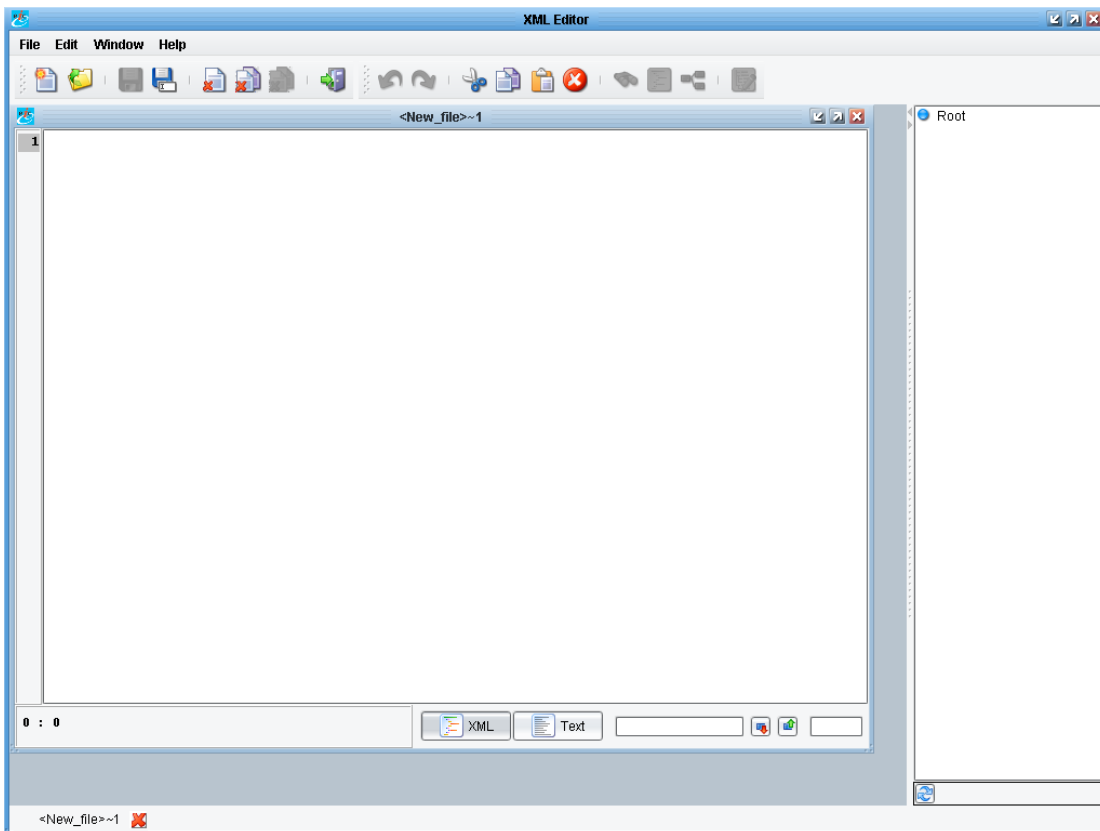


Back on the “Basic” tab, we’ll need to first provide a “Polling Interval,” which is how often our SQLXML is executed against the provided database connection. Since this is a tutorial, we’ll assume 5 seconds is sufficient:

Next we'll need to create our SQLXML file. Type the new filename “sql.xml” into the Input File field, then click the “Edit” button and select “In XML Editor”:



The image shows the 'Listener Configuration' dialog box. The 'Listener Name' is 'Selecting Rows Inbound' and the 'Listener Type' is 'Database Polling SQL Listener'. The 'Listener Description' is 'Interprets SQL-XML as Database Statements'. The 'Input File' field is 'sql.xml'. A red circle highlights the 'Edit' button (a green icon with a pencil) next to the 'Input File' field.



This will open the “XML Editor”. Here we can write out the body of our SQLXML.

As this is an XML file, we'll start by providing the basic XML header:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Our next line will be the root element, “SQLXML.” There's a particular namespace associated with SQLXML, so we'll define that using the “xmlns” attribute:

```
<SQLXML xmlns="http://pilotfish.sqlxml">
```

```
</SQLXML>
```

We can now provide a number of different SQLXML instructions inside the SQLXML body, including Execute, Select, Insert, Update, and Delete. We'll start with Select. The Select element has a required attribute called “into” which will specify the name of a variable for the selected rows to be placed in. We'll call ours “records”:

```
<Select into="records">
```

```
</Select>
```

Our code should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<SQLXML xmlns="http://pilotfish.sqlxml">
  <Select into="records">
  </Select>
</SQLXML>
```

The structure of the Select instruction is pretty simple; it expects a single child element sharing names with the table to select from. Please note that different databases may impose different restrictions on case sensitivity. Underneath the table name element, we'll provide empty tags matching the names of columns we're interesting in retrieving. We'll provide all of them:

```
<PEOPLE>
  <PERSON_ID />
  <FIRST_NAME />
  <MIDDLE_NAME />
  <LAST_NAME />
  <GENDER />
  <SSN />
  <BIRTHDATE />
  <OCCUPATION />
  <VEHICLE_MAKE />
  <VEHICLE_MODEL />
  <ADDRESS_LINE_1 />
  <ADDRESS_LINE_2 />
  <CITY />
```



```
<STATE />
<ZIP />
</PEOPLE>
```

The Select instruction should now create a SQL query fetching the various provided columns from the PEOPLE table. The results will be stored in a variable called “records.” Now we simply need to output this variable to XML. To do so, we'll add another instruction next to Select called “XMLOut” with an attribute called “var” specifying our “records” variable:

```
<XMLOut var="records"/>
```

The final file should look like the output below, now to save our file; Click the “save” button and navigate to our Working Directory, then the “routes” folder, and then finally the folder matching the name of our current Route (“Database – Selecting Rows”). Save the file there as “sql.xml” then close the XML Editor.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SQLXML xmlns="http://pilotfish.sqlxml">
  <Select into="records">
    <PEOPLE>
      <FIRST_NAME />
      <MIDDLE_NAME />
      <LAST_NAME />
      <GENDER />
      <SSN />
      <BIRTHDATE />
      <OCCUPATION />
      <VEHICLE_MAKE />
      <VEHICLE_MODEL />
      <ADDRESS_LINE_1 />
      <ADDRESS_LINE_2 />
      <CITY />
      <STATE />
      <ZIP />
    </PEOPLE>
  </Select>
  <XMLOut var="records" />
</SQLXML>
```

There's one more configuration value we'll change, located under the “Advanced” tab. Enable the “Use Single Output Stream” option:

**Listener Configuration**

Listener Name:

Listener Type:

Listener Description:

Connection	Scheduling	JDBC Props
Basic		
Advanced		
Initialize on trigger only:	<input type="checkbox"/>	
Restart on listening error:	<input type="checkbox"/>	
Is Batch Sensitive?:	<input type="checkbox"/>	
Timeout for Queries:	<input type="text" value="10"/> minutes	
Restrict Metadata To Catalog:	<input type="text" value=""/>	
Restrict Metadata To Schema:	<input type="text" value=""/>	
Restrict Metadata To Table(s):	<input type="text" value="%"/>	
Use Single Output Stream:	<input checked="" type="checkbox"/>	

This will cause all selected rows to come back in a single transaction. Unchecked, and by default, the Listener will produce a separate transaction for each row selected.

Stage Output Viewer

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <EIPData>
3   <RECORDS>
4     <DATA>
5       <STATE>NH</STATE>
6       <BIRTHDATE>1-22-1964</BIRTHDATE>
7       <OCCUPATION>Long distance operator</OCCUPATION>
8       <SSN>408-13-7134</SSN>
9       <VEHICLEMODEL>PARK AVENUE</VEHICLEMODEL>
10      <ADDRESSLINE2 />
11      <LASTNAME>Jacqueline</LASTNAME>
12      <ADDRESSLINE1>2256 E Aurelio St</ADDRESSLINE1>
13      <ZIP>87005</ZIP>
14      <FIRSTNAME>Cinderella</FIRSTNAME>
15      <MIDDLENAME>B</MIDDLENAME>
16      <GENDER>Female</GENDER>
17      <VEHICLEMAKE>BUICK</VEHICLEMAKE>
18      <CITY>Bluewater</CITY>
19    </DATA>
20    <DATA>
21      <STATE>OK</STATE>
22      <BIRTHDATE>3-2-1946</BIRTHDATE>
23      <OCCUPATION>De-icer element winder</OCCUPATION>
24      <SSN>754-66-6371</SSN>
25      <VEHICLEMODEL>TROOPER II</VEHICLEMODEL>
26      <ADDRESSLINE2 />
27      <LASTNAME>Freddie</LASTNAME>
28      <ADDRESSLINE1>6788 E Kevin Blvd</ADDRESSLINE1>
29      <ZIP>73487</ZIP>
30      <FIRSTNAME>Conrad</FIRSTNAME>
  
```

1:1

XML Text

Switch to the Console's Testing Mode and execute the test at the Listener stage, then view the output:

The Select instruction created a root element called "EIPData." We then have a "RECORDS" tag, created from our variable name ("records"). Each row then becomes a "DATA" element with the corresponding columns underneath.

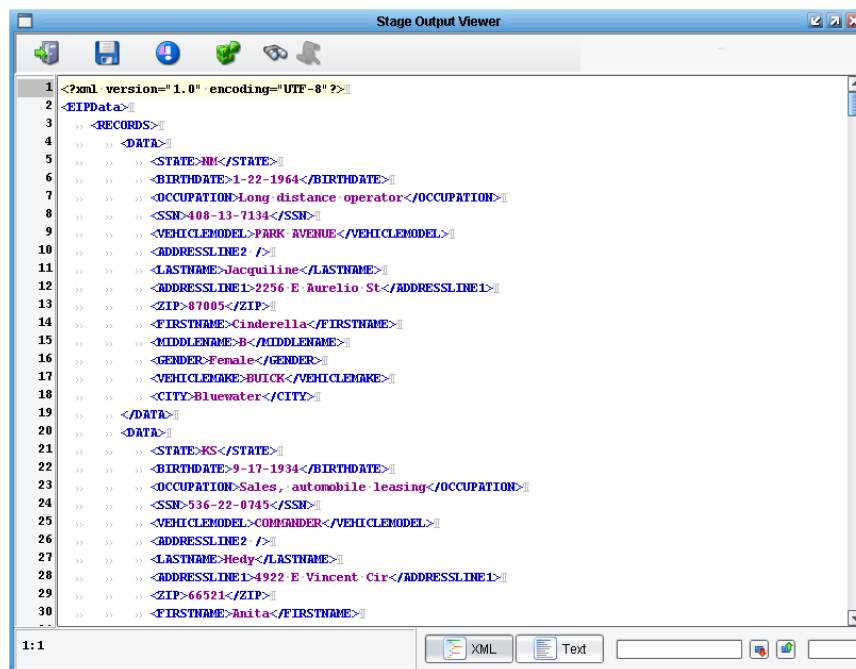
Suppose we need to specify the equivalent of a SQL "where" clause. That is, we need to select only particular rows. Return to your "sql.xml" file in the XML Editor. To use a column in such a way, you'll

need to provide a “key” attribute set to “true,” as well as the value to check against as the column text. Modifying the query to grab only those PEOPLE rows with Gender = “Female”:

```
<GENDER key="true">Female</GENDER>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<SQLXML xmlns="http://pilotfish.sqlxml">
  <Select into="records">
    <PEOPLE>
      <FIRST_NAME />
      <MIDDLE_NAME />
      <LAST_NAME />
      <GENDER key="true">Female</GENDER>
      <SSN />
      <BIRTHDATE />
      <OCCUPATION />
      <VEHICLE_MAKE />
      <VEHICLE_MODEL />
      <ADDRESS_LINE_1 />
      <ADDRESS_LINE_2 />
      <CITY />
      <STATE />
      <ZIP />
    </PEOPLE>
  </Select>
  <XMLOut var="records" />
</SQLXML>
```

Once again, execute this as a test and review the output:



If you review the results, you'll see all entries have the “GENDER” value as “Female.”

\*\*\* Bonus \*\*\*

Try creating a transaction per record by unchecking the “Use Single Output Stream” property:

The image shows a 'Listener Configuration' dialog box. At the top, there are three fields: 'Listener Name' with the value 'Selecting Rows Inbound', 'Listener Type' with a dropdown menu showing 'Database Polling SQL Listener', and 'Listener Description' with the text 'Interprets SQL/XML as Database Statements'. Below these fields are three tabs: 'Connection', 'Scheduling', and 'JDBC Props'. The 'JDBC Props' tab is selected, and it is further divided into 'Basic' and 'Advanced' sub-tabs. The 'Basic' sub-tab is active, showing several properties with checkboxes and input fields. The 'Use Single Output Stream' property at the bottom is unchecked, and its checkbox is highlighted with a red square. Other properties include 'Initialize on trigger only', 'Restart on listening error', 'Is Batch Sensitive?', 'Timeout for Queries' (set to 10 minutes), 'Restrict Metadata To Catalog', 'Restrict Metadata To Schema', and 'Restrict Metadata To Table(s)' (set to %).

Property	Value
Listener Name	Selecting Rows Inbound
Listener Type	Database Polling SQL Listener
Listener Description	Interprets SQL/XML as Database Statements
Initialize on trigger only	<input type="checkbox"/>
Restart on listening error	<input type="checkbox"/>
Is Batch Sensitive?	<input type="checkbox"/>
Timeout for Queries	10 minutes
Restrict Metadata To Catalog	
Restrict Metadata To Schema	
Restrict Metadata To Table(s)	%
Use Single Output Stream	<input type="checkbox"/>