

Perceptive Connect Runtime Database Connector

Configuration Guide

Version: 2.x for Perceptive Connect Runtime, version 2.2.x

Written by: Documentation Team, R&D
Date: October 2024

Documentation Notice

Information in this document is subject to change without notice. The software described in this document is furnished only under a separate license agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in the license agreement. This document or accompanying materials may contain certain information which is confidential information of Hyland Software, Inc. and its affiliates, and which may be subject to the confidentiality provisions agreed to by you.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. or one of its affiliates.

Hyland, HXP, OnBase, Alfresco, Nuxeo, and product names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2024 Hyland Software, Inc. and its affiliates.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity's local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity's jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

Table of Contents

Documentation Notice.....	2
Overview	4
Prerequisites.....	4
Installation	4
Download the files.....	4
Install the files	4
Drivers.....	5
Driver wrapping.....	5
Install a driver	5
Pooled data source factories	6
Edit pooled data source factory configuration	6
Delete a pooled data source factory configuration	6
Connection descriptions.....	7
Create a new connection description.....	7
Edit a connection description.....	8
Delete a connection description.....	8
Statement descriptions	8
Statement description parameters	9
Create a new statement description	9
Edit a statement description	10
Delete a statement description	10
REST endpoint execution	11
Unicode support.....	11
Configuration	11
<i>Enable rest endpoint.....</i>	<i>11</i>
<i>Enable CORS.....</i>	<i>11</i>
Statement signature	11
<i>application/json.....</i>	<i>12</i>
<i>application.xml.....</i>	<i>12</i>
Execute statement	13
<i>Execute statement using GET.....</i>	<i>13</i>
<i>Execute statement using POST</i>	<i>13</i>
<i>Execute statement response.....</i>	<i>14</i>

Appendix: Supported types	16
--	-----------

Overview

The Perceptive Connect Runtime (PCR) Database Connector provides a centralized interface for managing how PCR components and developers access your databases. Using the interface provided by the Database Connector, PCR administrators can create statement and connection descriptions that contain the information required to create database connections and execute SQL Statements. After you create the statement and connection descriptions, any developer or component in your PCR environment can access and use these descriptions.

The Database Connector provides features for external access to your configured Statement and connection descriptions through a REST API.

Prerequisites

To use the PCR Database Connector, you must have access to working installations of the following products.

- Perceptive Connect Runtime 2.2

Installation

Download the files

To obtain product installation files, contact the Hyland Software Technical Support group at (440) 788-5600. For a list of Technical Support numbers, go to hyland.com/pswtscontact.

Install the files

To install the PCR Database Connector, complete the following steps.

1. In a browser, go to the **Perceptive Connect Runtime Dashboard** at **`http://{Perceptive Connect Runtime host name}:{port}`**.
2. In the browser dialog box, enter the Connect Runtime user name and password.
Note The default user name and password are admin. However, the administrator can change the defaults during the Connect Runtime installation process.
3. Click **Install a Connector**.
4. On the **Upload New Bundles** page, drag the Database Connector zip file over the box on the right side of the page and then drop it.
5. When the installation completes, press **Accept to accept the installation** or **Roll back to undo the installation**.

Note For more information on installing Connectors please see the Install Connectors section of the [Perceptive Connect Runtime Installation Guide](#).

Drivers

By default, the Database Connector supports the following drivers.

- H2
- MSSQL Microsoft SQL Server 2016 (Preview), Microsoft SQL Server 2014, SQL Server 2012, SQL Server 2008 R2, SQL Server 2008, SQL Server 2005, and SQL Azure
- PostgreSQL 7.2 and newer
- Oracle 12.1 or 12cR1

Note If you need to support a different database version, then you will need to install your database driver. See the [Driver Wrapping](#) section for more information.

Driver wrapping

Before installing a driver, determine if your driver is OSGi compatible. If the driver is compatible, continue to [Install a Driver](#). If the driver is not compatible, you can use the Database Driver Packager utility script to wrap the driver or manually make the driver OSGi-compatible. To use the script, you must have [Python 2.7](#) installed. You can download Python [here](#).

To wrap your driver in an OSGi bundle, complete the following steps.

1. Open your OS command shell.
2. Open your *database-connector.x.x.x.zip* file and extract the contents of *util\DatabaseDriverPackager.zip* file to a temp directory.
3. Navigate to the directory containing the driver package.
4. Type `python package.py` and press **ENTER** to run the script.
5. When prompted, enter the complete file path for the driver.
 - For example, enter `C:/Program Files/MyDrivers/myDriver.jar`
6. When prompted, enter the complete file path for the destination folder.
 - For example, enter `C:/Program File/MyWrappedDrivers`
7. Enter the version of the driver. The script verifies a successful wrapping by displaying the complete file path for the new jar file.
8. The wrapped jar can be installed into the PCR environment by following the instructions below.

Install a driver

Note You only need to explicitly install a driver if you are using a newly wrapped driver. When you install a supported driver, the Database Connector creates two Data Source Factories: the driver's default or provided Data Source Factory, and a Pooled Data Source Factory. Both Data Source Factories appear as options in your **Connection Descriptions' Data Source** drop-down box.

To add your driver to the PCR, perform the following steps.

1. In the **PCR Dashboard**, click **Install a Connector**.
2. Locate your driver's jar file, and drag the jar file to the box on the right side your PCR window.
3. The system displays an installation summary. Use the summary to verify that the installation did not affect any unexpected bundles and then do one of the following sub-steps.

- If any bundles are adversely affected, click **Roll Back** and attempt the Driver Installation steps again.
- If the summary does not indicate any unexpected errors, click **Accept**
 - If the installation failed, use the summary to determine the cause.
 - Fix the cause of the failure and repeat the Driver installation steps.

Pooled data source factories

The Pooled Data Source Factories' configuration supports the following [HikariCP](#) pooled properties: `connectionTimeout`, `idleTimeout`, `maxLifetime`, `maximumPoolSize`, and `poolName`. If you leave a property unassigned, it uses HikariCP's default values.


- **connectionTimeout**. The maximum number of milliseconds a client will wait for a connection.
- **idleTimeout**. The maximum number of milliseconds a connection can sit idle before being retired.
- **maxLifetime**. The maximum lifetime, in milliseconds, of a connection in the pool.
- **maximumPoolSize**. The maximum size of the pool, including both idle and in-use connections.
- **poolName**. The name of the connection pool, used mostly for logging. If you do not name the pool, the Database Connector generates and logs a name for you.

For more details on properties, refer to [HikariCP's homepage](#).

Edit pooled data source factory configuration

To edit a pooled data source factory's configuration, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Under the **Database Connector** section, select the pooled data source factory you want to configure.

Note The pooled data source factory names follow the same convention as data source names in the Connection Descriptions: *Pooled Name (version)*
3. Enter values for the supported properties. Blank properties use HikariCP's default value.
4. Click **Save**.
 - If the save succeeds, then the configuration will have a checkmark under the  column.
 - If you do not want to save the configuration, click **Cancel** or click the **X** in the upper-right corner of the dialog box.

Delete a pooled data source factory configuration

Note Deleting a pooled data source factory configuration does not delete the Pooled Data Source Factory.

To remove a pooled data source factory configuration from the PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Find the pooled data source factory configuration you wish to delete and click **Delete**. The system displays a dialog window at the top of the browser window.
 - Alternatively, click **Edit** to open the configuration window.

- Using the fields located in this window, you can verify that this is the desired configuration.
 - Click **Delete**. The system displays a dialog box at the top of the browser window.
3. Click **OK**.
- If the deletion succeeds, the configuration will lose the checkmark under the ☐ column.
 - If you decide not to delete the configuration, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

Connection descriptions

A connection description contains the information needed to create a database connection using the Database Connector. Each connection description requires the target database's data source, host name, and host port. The description can store a user name and password for the target database, but the user name and password are only required if the target database requires an account for access. Additionally, you can set a network protocol for the connection, and you can add a literal description. Without a connection description, the Database Connector cannot create any statement descriptions, so ensure the required connections exist before creating any statement descriptions. You can only use installed drivers for your descriptions, so ensure all the necessary drivers are in your PCR.

Developers and other PCR components can retrieve connection descriptions created using PCR's user interface. If a developer or component needs a specific connection description, you must either provide the description's name or its persistent identifier (PID). While creating the connection description, you determine the description's name, but the PCR generates its PID. You cannot change the PID.

If you give a connection description a non-unique name, the Database Connector appends a unique identifier to the new description's base name. The unique identifier follows the form of `_n`, where `n` is the number of connection descriptions sharing the base name. If the modified name is unacceptable, you can edit the connection description. If you change a description's name, ensure that you update any PCR components or programs that reference the original name.

Create a new connection description

To add a new connection description to PCR, perform the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Next to the **Connections** subsection of the **Database Connector** section, click the **Plus** button to open the **Connections** window.
3. Enter a name for the connection description and the description's database name, literal description, host, port, network protocol, user name, and password into their respective text fields. If you want to start over with a clean dialog window, click **Reset**.
4. Next to the **Data Source** section, use the drop-down list to select your connection's driver. Each list option displays the name of its corresponding driver. If a driver declares its version, then the driver's version appears in parentheses after the driver's name.
5. Click **Save**. If the save succeeds, you should have a new configuration under the **Connections** subsection with the configured name for the Name column and EDBC for the Bundle column. If you decide not to save the Connection description, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

Edit a connection description

Important If you save an edited connection description, then you cannot undo the changes. To restore an edited description, you must manually re-enter the old values.

To edit an existing connection description in PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Under the **Connections** subsection of the Database Connector, locate the name of the connection description you want to edit.
3. Click **Edit**. The system displays the **Connection** dialog box.
4. Locate and edit the desired fields. If you make a mistake and want to return the window to its initial state, click **Reset**.
5. When you are finished, click **Save**. If you decide not to save your changes, click **Cancel** or click the **X** in the upper, right corner of the window.

Delete a connection description

Important Deleting a connection description cannot be undone. If you need a deleted description, you must manually recreate the description. However, the new connection description will not have the same PID, so you must also update any references to the old PID.

To remove a connection description from PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Under the **Connections** subsection of the Database Connector, locate the name of the connection description you want to delete.
3. Click **Delete**. The system displays a dialog box at the top of the browser window.
 - Alternatively, click **Edit** to open the **Connections** window.
 - Using the fields located in this window, you can verify that this is the desired connection description.
 - Click **Delete**. The system displays a dialog window at the top of the browser window.
4. Click **OK**. If the deletion succeeds, then the configuration will disappear from the **Connections** section. If you decide not to delete the description, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

Statement descriptions

A statement description contains the information needed to execute a SQL statement using the Database Connector. To create a new statement description, you must enter a name for the description and the SQL statement you want to execute. If the statement is parameterized, then you must provide input parameters. If the statement is a SELECT query, then you should provide output parameters. If your components and developers plan to retrieve only the SQL ResultSet of a SELECT statement, then the output parameters are optional, but entering the output parameter is highly recommended. For more information on input and output parameters, refer to [Statement description parameters](#). Before you can create statement descriptions, you must create the related connection descriptions.

As with connection descriptions, the Database Connector provides other PCR components and developers access to statement descriptions. However, unlike connection descriptions, you must

reference statement descriptions by name. PCR still generates a PID, but the Database Connector cannot retrieve statement descriptions by PID.

Like connection descriptions, the Database Connector modifies statement descriptions that have non-unique names. The modifier follows the same form of `_n`, where `n` is the number of statement descriptions sharing the same base name. If a new name is unacceptable, you can edit the description and update any references to the name.

Statement description parameters

Statement descriptions have two complex fields, **input parameters** and **out parameters**. If the description's SQL Statement is parameterized, then the input parameters determine the values for the SQL Statement. Additionally, the order of the input parameters determines the order in which these values are applied to the SQL Statement.

The output parameters determine the values the Database Connector will extract from an executed SQL statement. The order of the output parameters does not matter, but the output parameters' labels must match the output columns from the SQL statement. If the SQL statement selects "*" from a table, then each output parameter must match a column from the SQL statement's target table. If a statement description's SQL statement does not return column data, then the description will not have output parameters.

To add parameters to a statement description, first choose a delimiter. The delimiter separates a parameter's three parts: type, label, and format. *Type* and *label* are required, but *format* is optional. You can use any String for the delimiter, but the delimiter should be short and not used in a parameter's label, type, or format. Using a single special character, such as \$ or # is highly recommended.

Note Some parameter types do not support formatting. For more details, see "ParameterType" in the *Perceptive Connect Runtime Database Connector API Guide*.

After choosing a delimiter, enter the description's input and output parameters. Both input and output parameters follow one of these patterns: `type + delimiter + label` or `type + delimiter + label + delimiter + format`. For example,

- The parameter "**String#FirstName**" represents a parameter with **String** for the type, **#** for the delimiter, and **FirstName** for the label.
- The parameter "**Date#Birthday#MM-dd-YYYY**" represents a parameter with **Date** for the type, **#** for the delimiter, **Birthday** for the label, and **MM-dd-YYYY** for the format.

For more information on formatting, see "ParameterType" in the *Perceptive Connect Runtime Database Connector API Guide* for more details..

The type declaration is not case sensitive. For a complete list of supported types, refer to [Appendix: Supported types](#). To add a parameter, click the **Plus** button located to the right of the desired parameter type. To remove a parameter, click the **Minus** button located to the right of the **Plus** button.

Create a new statement description

To add a new statement description to PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Next to the **Statements** subsection of the **Database Connector** section, click the **Plus** button to open the **Statements** window.
3. Enter a name for the statement description and the description's parameter delimiter, input parameters, output parameters, and SQL Statement into their respective text fields.

- If you want to start over with a clean dialog window, click **Reset**.
 - If your statement does not have input parameters, leave the input parameter section empty.
 - Similarly, if your Statement does not have any output parameters, leave the output parameter section empty.
 - If your Statement does not have any input or output parameters, you may leave the delimiter field blank.
4. Use the **Connection** drop-down field to select the connection to use with this description.
 5. Click **Save**. If the save succeeds, you should have a new configuration under the Statements subsection with the configured name for the Name column and EDBC for the Bundle column. If you decide not to save the statement description, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

Edit a statement description

Important

- If you save an edited statement description, then the edit cannot be undone. To restore an edited description, you must manually re-enter the old values.
- Modifying a statement description's input or output parameters changes how the Database Connector interfaces with the description. You may need to update programs, components, or channels that reference the modified statement description.

To edit an existing statement description in the PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Under the **Statements** subsection of the **Database Connector** section, locate the name of the statement description you want to edit.
3. Click **Edit**. The system displays the **Statement** dialog box.
4. Locate and edit the desired fields. If you make a mistake and want to return the window to its initial state, click **Reset**.
5. When you are finished, click **Save**. If you decide not to save your changes, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

Delete a statement description

Important Deleting a statement description cannot be undone. If you need a deleted description, you must manually recreate the description.

To remove a statement description from the PCR, complete the following steps.

1. In the **PCR Dashboard**, click **Configure**.
2. Under the **Statements** subsection of the **Database Connector** section, locate the name of the statement description you want to delete.
3. Click **Delete**. The system displays a dialog window at the top of the browser window.
 - Alternatively, click **Edit** to open the **Statements** window.
 - Using the fields located in this window, you can verify that this is the desired statement description.

- Click **Delete**. The system displays a dialog window at the top of the browser window.
4. Click **OK**. If the deletion succeeds, then the configuration will disappear from the **Statements** section. If you decide not to delete the description, click **Cancel** or click the **X** in the upper, right corner of the dialog window.

REST endpoint execution

The Database Connector provides a REST endpoint that can execute Statements defined by PCR's configuration UI.

This endpoint is at <http://hostname:port/rs/databaseConnector/>, where hostname and port are your PCR's *hostname* and *port*. You can access the Web Application Description Language (WADL) description for the endpoint at http://hostname:port/rs/databaseConnector?_wadl.

Unicode support

The REST Endpoint supports Unicode. The URL supports Unicode through standard percent-encoding. Additionally, the Endpoint supports supplemental characters.

Note The endpoint does not check grapheme cluster equality. For example, if your Statement name contains an ñ (U+00F1:Latin Small Letter N With Tilde) and you use n (U+006E:Latin Small Letter N) plus the combining tilde (U+0303:Combining tilde) to reference the Statement name, then the names are not equal even though they appear the same visually.

Configuration

You can configure the REST Endpoint through the Database Connector REST Component configuration in the PCR.

Enable rest endpoint

If necessary, you can disable the REST Endpoint. Disabling the endpoint prevents anyone from remotely accessing the Database Connector, so only PCR components and services have access. To disable the REST Endpoint, deselect the Enable option in the configuration.

Enable CORS

The REST Endpoint supports adding CORS headers to the response of all requests to the endpoint. By default, the endpoint adds CORS headers and allows all origins. If you want to disable this behavior, uncheck the Enable CORS option in the configuration.

Statement signature

A Statement signature contains the input parameters and output parameters of a statement description.

To get a statement sSignature, make a GET request to:

<http://hostname:port/rs/databaseConnector/statement/{name}/signature>

Where *name* is a *String* that matches the name of a configured statement in your PCR.

The statement signature endpoint accepts two values for the Accept HTTP Header: *application/json* and *application/xml*.

application/json

The `json` response returns an object with two properties: `inputParameters` and `outputParameters`. Both `inputParameters` and `outputParameters` are arrays. Each element is an object with three properties: `name`, `type`, and `format`. The `name` property contains the parameter's name, the `type` property contains the parameter's class, and the `format` property contains the optional parameter format.

```
{
  "inputParameters": [
    {
      "name": "NAME",
      "type": "TYPE",
      "format": "FORMAT"
    }
  ],
  "outputParameters": [
    {
      "name": "NAME",
      "type": "TYPE",
      "format": "FORMAT"
    }
  ]
}
```

Note If a parameter does not have a format, then the `format` property is an empty string.

application.xml

The `xml` response returns an `xml` object, `<statementSignature>`, with two children: `<inputParameters>` and `<outputParameters>`. Both `<inputParameters>` and `<outputParameters>` can have any number of `<-Parameter>` children. Each `<-Parameter>` child has an input or output prefix to signify its parameter type. Each parameter node has three children: `<name>`, `<type>`, and `<format>`. The `<name>` element contains the parameter's, the `<type>` element contains the parameter's class, and the `<format>` element contains the optional parameter format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statementSignature>
  <inputParameters>
    <inputParameter>
      <name>NAME</name>
      <type>TYPE</type>
      <format>FORMAT</format>
    </inputParameter>
  </inputParameters>
  <outputParameters>
    <outputParameter>
      <name>NAME</name>
      <type>TYPE</type>
      <format>FORMAT</format>
    </outputParameter>
  </outputParameters>
</statementSignature>
```

Note If a parameter does not have a format, then the `<format>` element is empty.

If the endpoint cannot find the requested statement signature, the endpoint returns a 404 Not Found error.

Execute statement

You can execute statements by calling one of the following endpoints.

- GET `http://hostname:port/rs/databaseConnector/statement/{name}`
- POST `http://hostname:port/rs/databaseConnector/statement/{name}`

Where name is the name of a configured statement in your PCR.

Both the GET and POST endpoints execute a statement the same way. Both have their own benefits and calling procedures.

Execute statement using GET

Using GET, you pass input parameters through url-encoded query parameters. The query parameter names are case-sensitive, unique, and match the input parameter names from the statement UI. The GET endpoint does not accept any request body content. Using GET allows external caching client-side or by various networking hardware.

Note The endpoint provides no internal caching.

Example

```
GET http://hostname:port/rs/databaseConnector/statement/
AddBookToLibrary?Title=To%20Kill%20a%20Mockingbird&Author=Harper%20Lee
```

Execute statement using POST

Using POST, you can pass input parameters by using url-encoded query parameters. However, unlike GET, POST also supports passing a request body with a Content-Type of either application/json or application/xml. The request body is optional. You can use any combination of query and request body parameters, but you cannot pass the same parameter in both. If you do not want to supply a request body, ensure either no Content-Type header is set or the Content-Type header is set to text/plain.

Examples

POST call without request body

```
POST
http://hostname:port/rs/databaseConnector/statement/AddBookToLibrary?Title=To%20Kill%20a%20Mockingbird&Author=Harper%20Lee
Content-Type: text/plain
```

POST call with only request body parameters in JSON

```
POST http://hostname:port/rs/databaseConnector/statement/AddBookToLibrary
Content-Type: application/json
{
  "inputParameters": {
    "Title": "To Kill a Mockingbird",
    "Author": "Harper Lee"
  }
}
```

POST call with mixed query and request body parameters in XML

```

POST
http://hostname:port/rs/databaseConnector/statement/AddBookToLibrary?Author=Harper%20Lee
Content-Type: application/xml
<executeStatementRequest>
  <inputParameters>
    <inputParameter>
      <name>Title</name>
      <value>To Kill a Mockingbird</value>
    </inputParameter>
  </inputParameters>
</executeStatementRequest>

```

Execute statement response

The execute statement endpoint produces both `application/json` and `application/xml`. The response contains a results list with result objects that contain `updateCount` and `rows`.

updateCount. The `updateCount` is the number of rows affected by an executed Statement; however, -1 is a reserved value. It signifies that the executed Statement could not change any rows in the database. An UPDATE Statement might not update any rows. In this case, the update count would be 0 instead of -1.

rows. The `rows` property is the list of results from an executed Statement. It has zero to `n` `row` children. Each `row` is a row of the `ResultSet` from the executed Statement. The `rows` property has a value only if the `updateCount` is -1. A Statement cannot update rows in the database and have a result set.

xml. In the xml response, `<results>` will have one `<result>` child that has one `<updateCount>` and one `<rows>` child. The `<rows>` will have zero to `n` `<row>` children. A `<row>` element has a single child, `<columns>`, that contains a list of `<column>`s. There will be a `<column>` for every `outParameter` defined in `StatementInfo`.

json. In the json response, results will have one child that has both an `updateCount` and `rows`. The `rows` will have zero to `n` elements in the array. Each array index contains a map of column names to their values. There will be a column for every `outputParameter` defined in `StatementInfo`.

Examples

GET call to insert a new book into the library that returns `application/xml`

```

GET
http://hostname:port/rs/databaseConnector/statement/AddBookToLibrary?Title=Go%20Set%20a%20Watchman&Author=Harper%20Lee
Accept: application/xml

```

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<executeStatementResponse>
  <results>
    <result>
      <updateCount>1</updateCount>
    </result>
  </results>
</executeStatementResponse>

```

GET call to insert a new book into the library that returns `application/json`

```
GET
http://hostname:port/rs/databaseConnector/statement/AddBookToLibrary?Title=Go%20Set%20
a%20Watchman&Author=Harper%20Lee
Accept: application/json
{
    "updateCount": 1,
    "rows": null
}
```

GET call to query all the books in the library that returns application/xml

```
GET http://hostname:port/rs/databaseConnector/statement/GetAllBooks
Accept: application/xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<executeStatementResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <results>
        <result>
            <updateCount>-1</updateCount>
            <rows>
                <row>
                    <columns>
                        <column>
                            <name>Title</name>
                            <value xsi:type="xs:string">To Kill a Mockingbird</value>
                        </column>
                        <column>
                            <name>Author</name>
                            <value xsi:type="xs:string">Harper Lee</value>
                        </column>
                    </columns>
                </row>
                <row>
                    <columns>
                        <column>
                            <name>Title</name>
                            <value xsi:type="xs:string">Go Set a Watchman</value>
                        </column>
                        <column>
                            <name>Author</name>
                            <value xsi:type="xs:string">Harper Lee</value>
                        </column>
                    </columns>
                </row>
            </rows>
        </result>
    </results>
</executeStatementResponse>
```

GET call to query all the books in the library that returns application/json

```
GET http://hostname:port/rs/databaseConnector/statement/GetAllBooks
Accept: application/json
```

```
{
    "results" : [
        {
            "updateCount": -1,
            "rows": [
```

```
{
  {
    "Title": "To Kill a Mockingbird",
    "Author": "Harper Lee"
  },
  {
    "Title": "Go Set a Watchman",
    "Author": "Harper Lee"
  }
]
}
```

Appendix: Supported types

The supported types are:

- Boolean
- Byte
- **Note** The BYTEA type used by PostgreSQL is not supported at this time.
- Double
- Date
- Float
- Integer
- Long
- Short
- String
- Time
- Timestamp