Help: Database query Action Plug-in

Version

0.1.5

Overview

This plug-in provides the ability to execute an SQL query on a database and format its result.

Description

Saved Parameters Description

- **Name**: The name used to identify a saved DatabaseQuery configured instance.
- **Comments**: Some comments about this saved DatabaseQuery configured instance.
- **Database Type**: The type of the external Database to connect to for the execution of the query. Supported RDBMS types are MySQL, Oracle and all ODBC compatible RDBMS.
- **Database host**: In the case of ODBC and Oracle, the DB host where the RDBMS is executing is speficied in the datasource. In the case of MySQL it needs to be specified explicity if the DB is not hosted on localhost
- **Database port**: The port to connect to the database. As for the host, the port to be used is specified in the datasource configuration for ODBC and Oracle, hence this parameter only makes sense for MySQL
- **Database name (or source name)**: For MySQL, the database name needs to be specified, for ODBC and Oracle, the data source name is used instead.
- **Database user**: The user name for the external database being queried
- **Database password**: The password associate to the database user
- **Database query**: The SQL query to execute. Stored procedure can be used. Dynamic attributes are supported. For example '**select name from users where login = "<%=user_login%>"**' would expand an input variable name 'user_login' and return the matching database rows.
- **Row formatting code**: This code will be executed for each row of the result set. Each row is represented in a variable name 'row' as a hash indexed by the database column name (as a symbol), for example **{:name=>'Maurice',:login=>'momo'}**. The content of the row variable can be modified to alter formatting. For example, **row = "#{row[:login]} (#{row[:name]})"**, would change the content of row to be a string, here '**momo (Maurice)**'.
- **Results formatting code**: This code will be executed after each row has been post-processed with the code specified in the 'row formatting code' box. A variable 'results' containing the array of the formatted rows can be manipulated to modify the action result. For example, **results = results.join(", ")** combined with the previous row formatting code, would mean results (and hence the output of the action) would be a string. Here '**momo (Maurice), baba (Bill)**'
- **Result Type**: The type of the action output. By default, the action returns an array of hash. With each matching row found represented by a hash, indexed by the database column name (as a symbol), for example **[{:name=>'Maurice',:login=>'momo'},{:name=>'Bill',:login=>'baba'}]**. The type chose should match the formatting code specified. No automatic type translation will be done, so all transformation needs to be explicit in the row and results formatting code boxes. In the example above, the output type should be set to 'string'

Inputs description

The list of inputs depends on the configuration of the Database Query action template as the query and formatting fields support dynamic inputs.

Outputs description

- *Query_results*: The formatted results of the query. If no formatting is provided, the results is an array of hash, otherwise the type of this outputs will depends on the specified output type.
- *Inserted Id*: Only if the query is an insert operation, the ID of the new record is returned.

Supported Actions

None

Dependencies

None

Operating Instructions

Notes: in order to be able to use Oracle or ODBC, some additional installation steps are required.

For Oracle: The Oracle client needs to be installed on the Orchestrator server.
The datasource to be accessed must be set up in the tnsnames.ora file. Before attempting a connection through Orchestrator, ensure that connectivity can be achieved and queries can be executed from the command line.
Once the oracle client is installed, the ruby-oci8 gem needs to be installed.

```
        cd /opt/aspera/orchestrator         gem install
vendor/gems_linux-gnu/ruby-oci8-2.1.4.gem
```

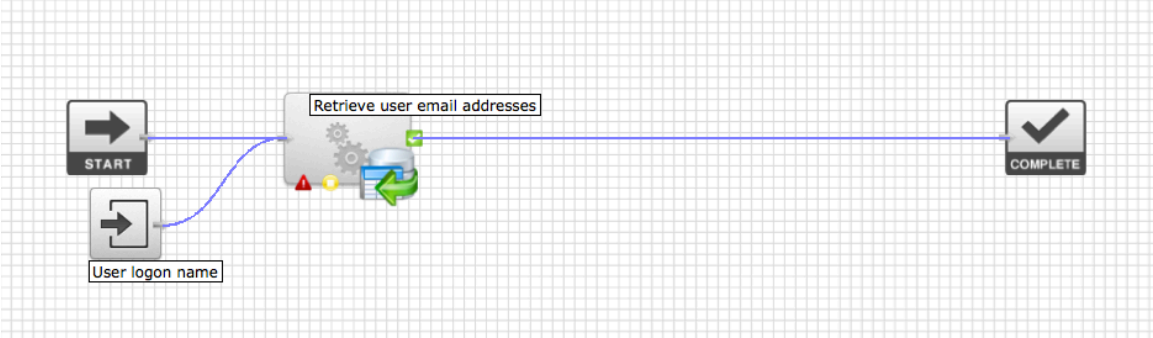
For ODBC: The ODBC driver needs to be installed on the Orchestrator server (FreeTDS can be used on Linux).
The ODBC source to be accessed must be set up. Before attempting a connection through Orchestrator, ensure that connectivity can be achieved and queries can be executed from the command line.
Once the ODBC drivers are installed, the ruby-odbc and dbd-odbc gems need to be installed.

```
        cd /opt/aspera/orchestrator         gem install
vendor/gems_linux-gnu/ruby-odbc-0.99994.gem         gem install
vendor/gems_linux-gnu/dbd-odbc-0.2.5.gem
```

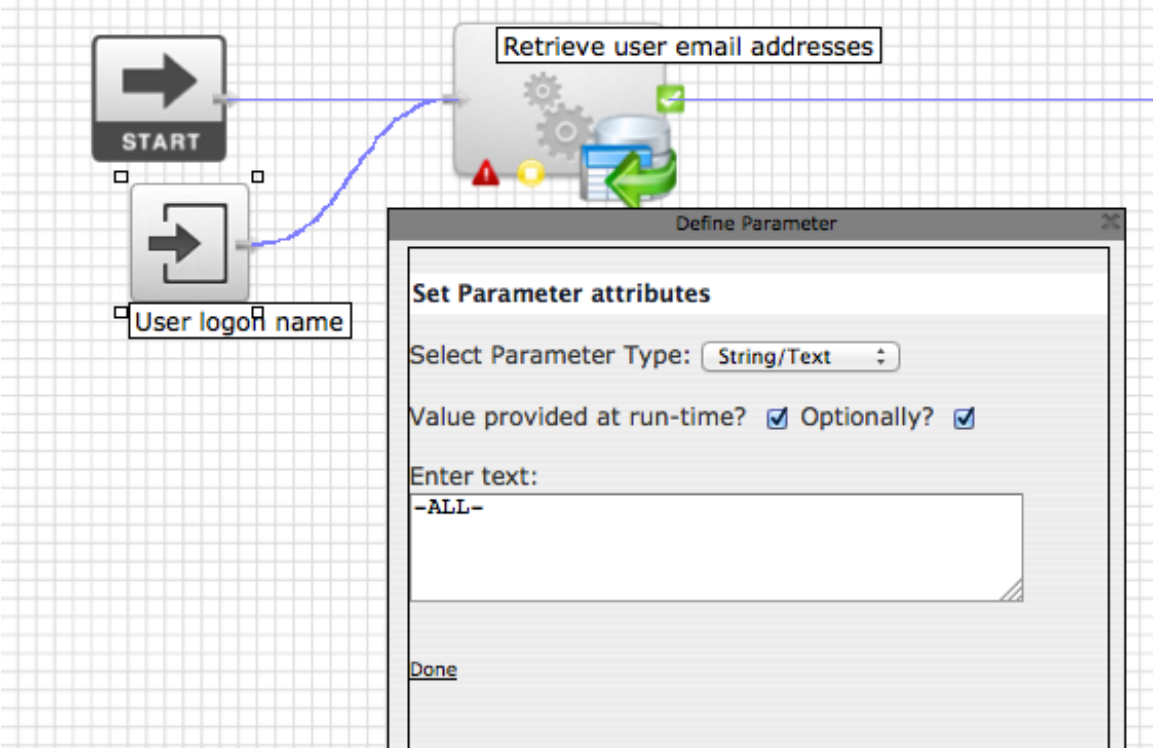
Sample workflows

We will create a small workflow whose goal is to retrieve information about active user from a database. For simplicity here we will use the Orchestrator database.

The workflow just as a single step for now:



The parameter is a runtime string and is defaulted to a value indicating no filtering is required.



If a value is provided, the goal is to only provide information for the specified user.

The database query action configuration is set as follow:

Database query configuration

Name Retrieve user email addresses

Comments

Database type Mysql

Database host (mysql only) localhost

Database port (mysql only) 3306

Database name (or odbc source) orchestrator

Database user root

Database password ••••••

re-enter ••••••

Database query

```
SELECT firstName, lastName, email FROM users WHERE
```

Toggle editor

Row formatting code

Toggle editor

The string used can contain static text and variable expanded at run-time. Variables are indicated using the syntax <%=variable_name%> embedded into static text. For example '/tmp/<%=basename(full_name)%>'
The row is represented as a variable 'row' and is a hash with the column name as a symbol key (i.e.: row[:first_name]=>'Maurice')

Results formatting code

Toggle editor

As above, the string used can contain static text and variable expanded at run-time. The resuls are represented in an array name 'results' of which each line is by default a hash, unless the formatting as been changed above.

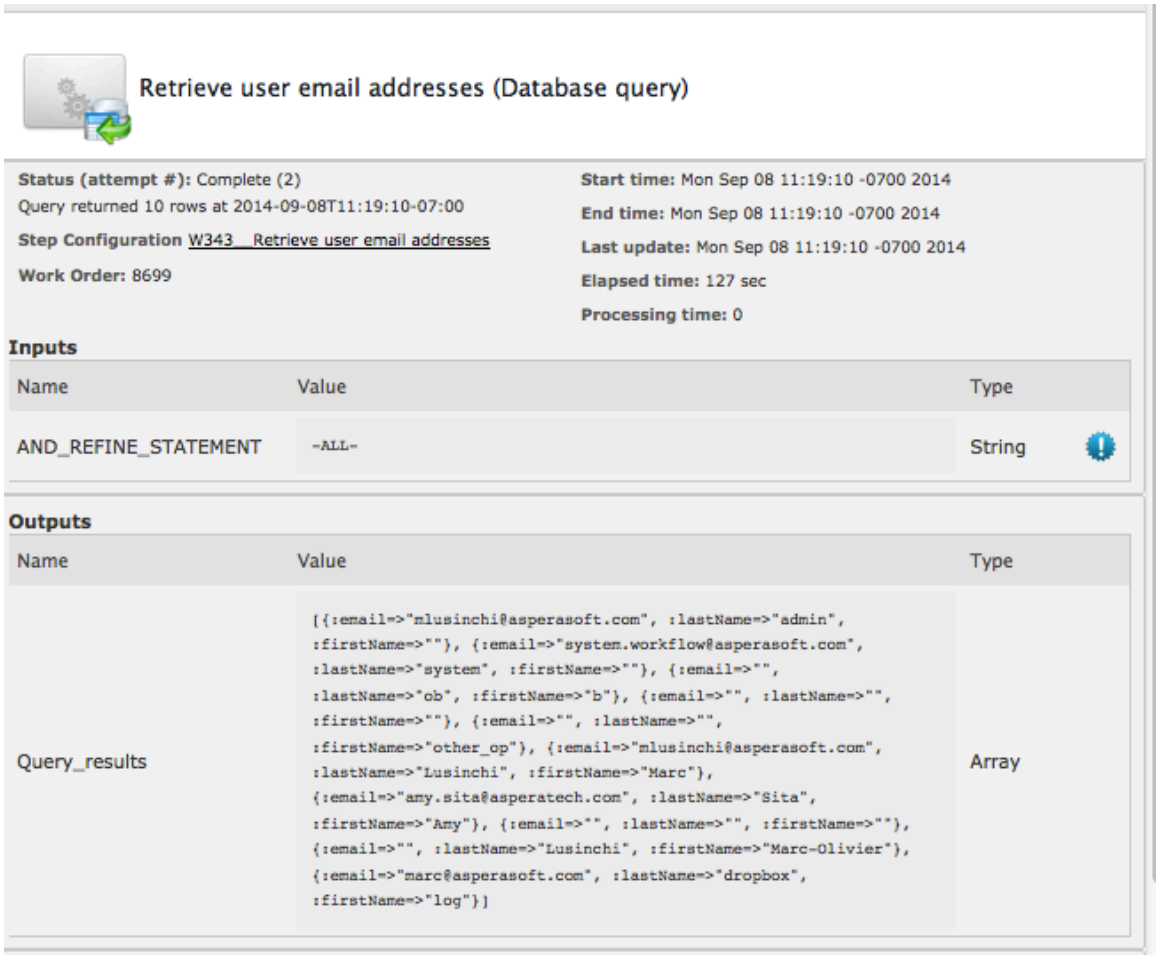
Result type array

The full query is:

> **SELECT firstName, lastName, email FROM users WHERE isActive = true <%= AND_REFINE_STATEMENT %>**

Note the dynamic attribute, if it is blank, the query will return the data for all active user. To narrow the scope to a specific user login, for example '**admin**', **AND_REFINE_STATEMENT** would need to be set to "**AND login = 'admin'**"

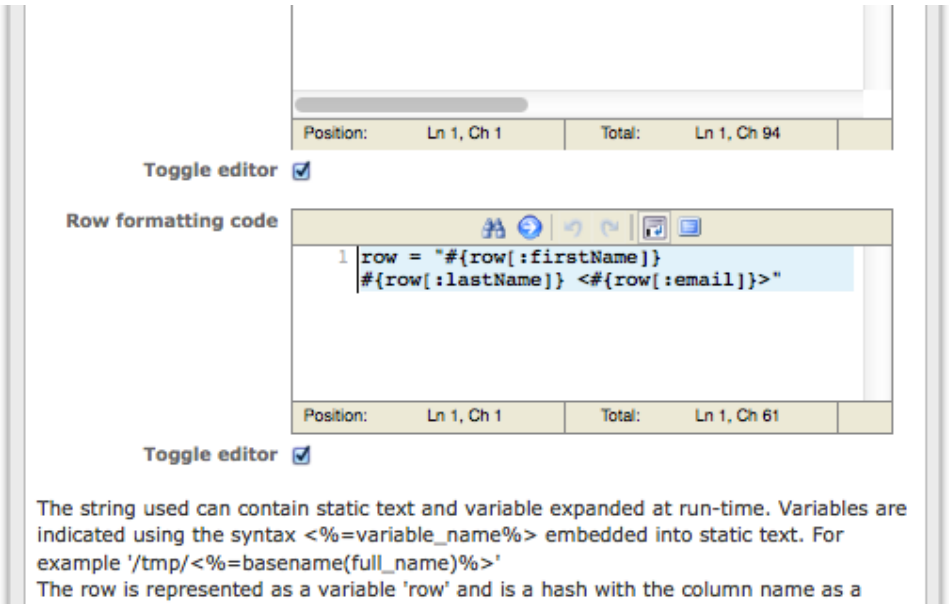
If we run this workflow with the default runtime value, we get the following outputs for that step:



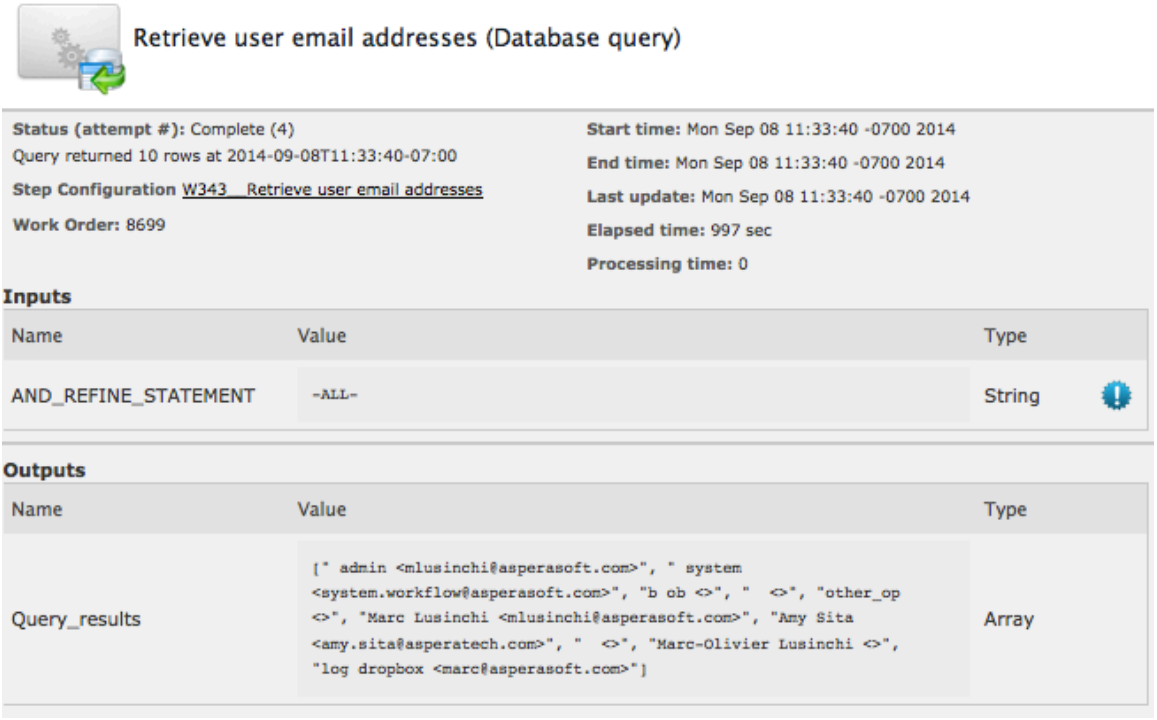
Note that the result is an array of hash, with each hash representing the data for one row. Symbols are used as key, not string. So to get a value of a row, the syntax would be **output_value[0][:email]** (value here would be the string "mlusinchi@asperasoft.com" . And **output_value[0]['email']** would be nil.

Now let's format the row. Let's say we want the email address in rich format: **FirstName LastName <email@address>**

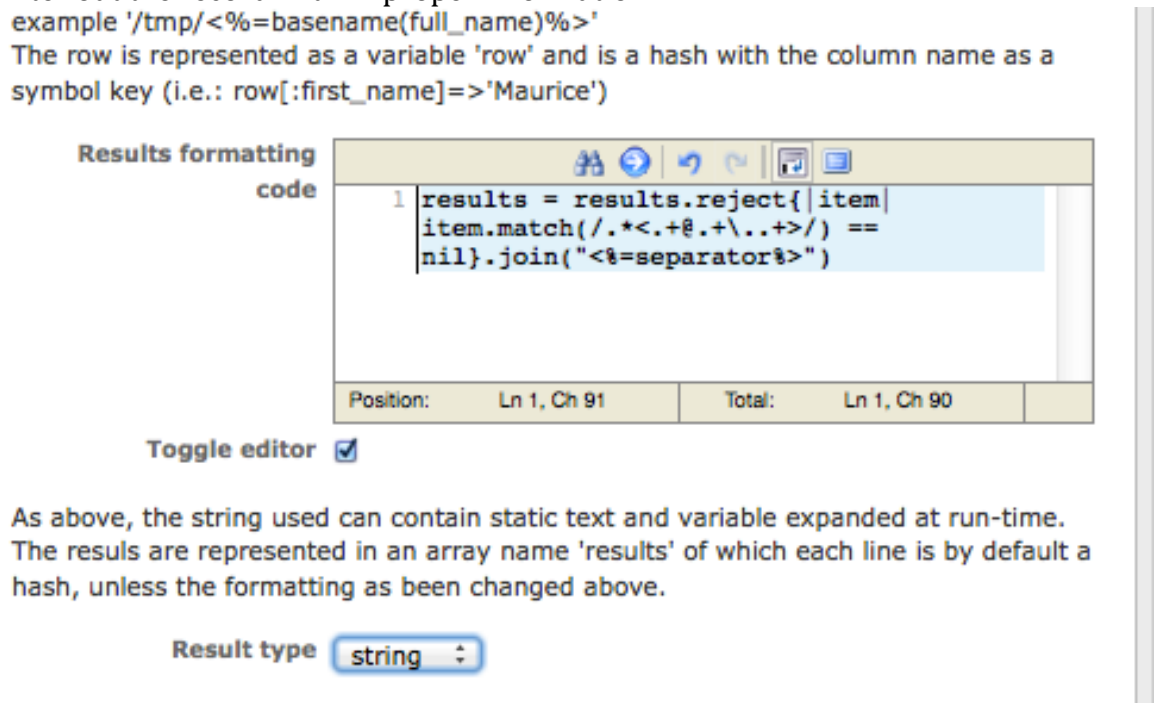
This is done by modifying the **row formatting** code field of the action configuration:



The result is now



Now let's assume we want to get the result as an aggregated string and we want to filter out the record with improper information.

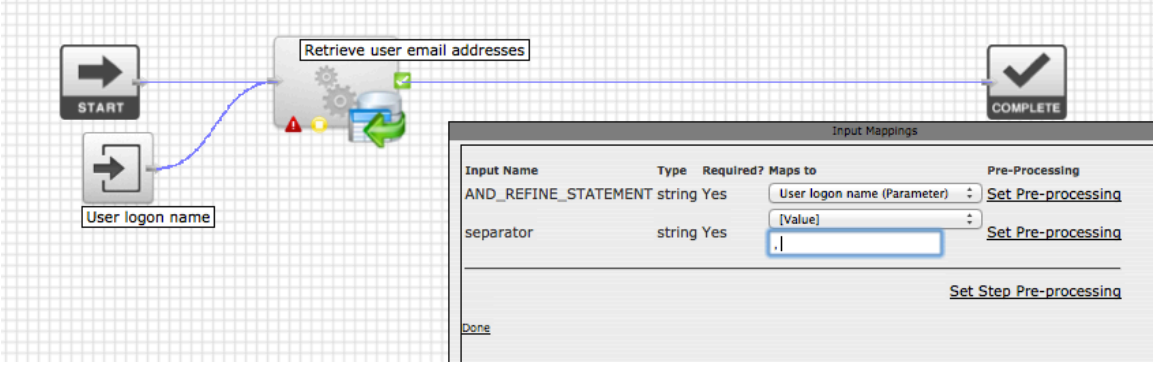


The reject statement will cause all the entries not matching the provided format to be skipped. The join statement converts the array to a delimitated string. The

separator to be used was made a dynamic attribute and would thus be whatever is specified as a string input “separator”.

Note that **the result type** was set to **string** to match what the **results formatting code** is returning.

Now since we have an extra input, we need to map it



Here the mapping is done to inline value of “**,** “ (comma then space)

If we relaunch, this time we get the following results: