How to Guide:

Connect Power BI to ArangoDB

In this tutorial we will use Power BI Desktop with the Power BI Web connector for ArangoDB using ArangoDB 3.4.

The Foxx service is not an official Power BI connector, it uses the Web Connector and could be customized to allow fit for individual needs. Power BI is a trademark of Microsoft

ArangoDB

In this tutorial we will use Power BI Desktop with the Power BI Web connector and ArangoDB 3.4.

This tutorial assumes you have access to a running instance of ArangoDB that already has a collection with data you want to export to Power BI. Note that the connector for ArangoDB only supports exporting data from a single collection.

Installing the connector

The Power BI connector can be installed as a Foxx service using the ArangoDB web interface or the Foxx CLI.

To install the service using the web interface:

1. Open the ArangoDB web interface in your browser (e.g. `http://localhost:8529` if ArangoDB is running locally).
2. Enter your ArangoDB credentials and select the database from which you want to export data to Power BI.
3. Select the *Services* tab on the right, press *Add Service* and select the *powerbi-connector* service.
4. Enter a mount point (e.g. `/powerbi`) and press *Install*.

To install the service using the Foxx CLI use the following command (assuming user `root`, database `_system` and mount point `/powerbi`):

```
foxx install -u root -P -H http://localhost:8529 -D _system /powerbi \
https://github.com/arangodb-foxx/powerbi-connector/archive/master.zip
```

Configuring the connector

Before the Power BI connector is ready to use it needs to be configured:

- **Collections**: the names of collections in the current database the connector should have access to (multiple values can be separated by comma but only one collection can be imported at a time).

- **Username** and **password**: credentials that will be used to protect the connector against unauthorized access. Note that these are different from the credentials used to access ArangoDB itself and will only be used by Power BI to authenticate against the connector.

To configure the service from the web interface:

1. Select the *Services* tab and select the mount point where the Power BI connector service was installed.
2. Select the *Settings* tab from the top bar.
3. Fill in the configuration values and press the *Apply* button to save.

To configure the service using the Foxx CLI use the following command (assuming collection `data`, username `powerbi` and password `powerbi123`):

```
foxx config -u root -P -H http://localhost:8529 -D _system /powerbi \
collections=data username=powerbi password=powerbi123
```

Using the connector in Power BI Desktop

We'll start out by connecting a new data source using the *Get Data* button in the toolbar and selecting the *Other* > *Web* data source. You can also find the connector more easily by entering "Web" in the search box. Press the *Connect* button to proceed to the next dialog.

The **URL** should be the full path of the Power BI connector, including the name of the collection you want to load. Note that we can only load one collection at a time and the connector needs to be configured to expose that collection.

For example, if we have a local copy of ArangoDB running on our machine, the Power BI connector was installed at `/powerbi` and the name of the collection we want to import is `data`, our URL looks like this: `http://localhost:8529/_db/_system/powerbi/data`

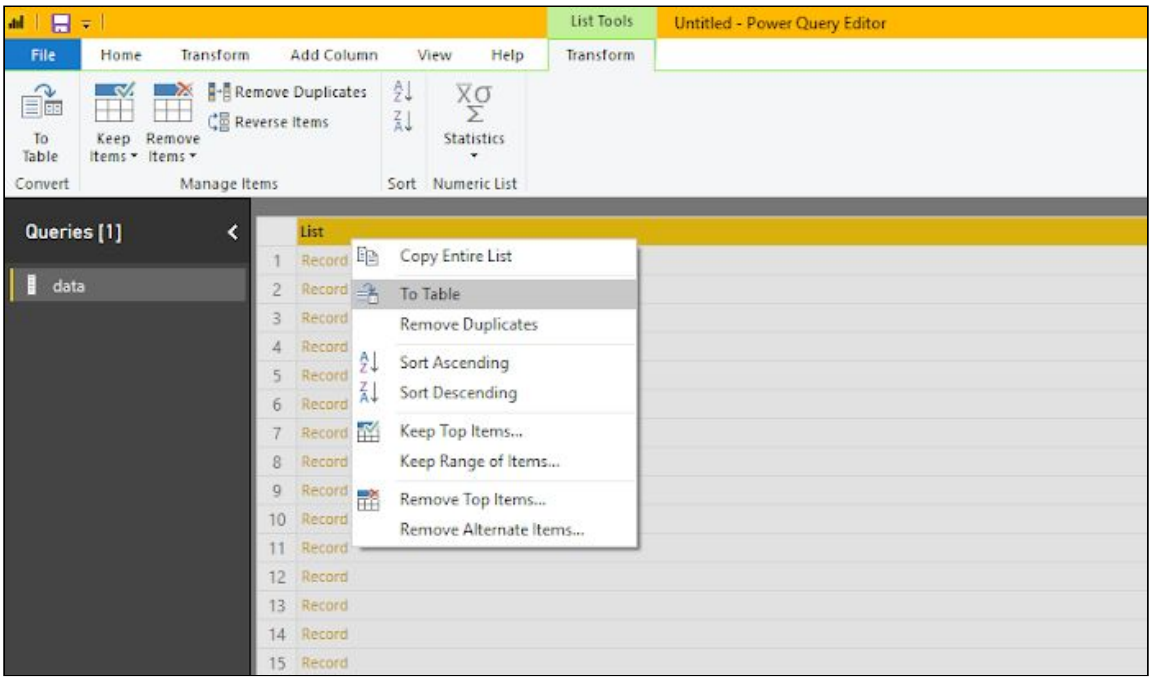
Press the *OK* button to proceed to the next dialog, which will inform you that this URL requires authentication.

Press the **Basic** tab and enter the credentials from the configuration of our Power BI connector. Remember that these are not the credentials used to connect to ArangoDB

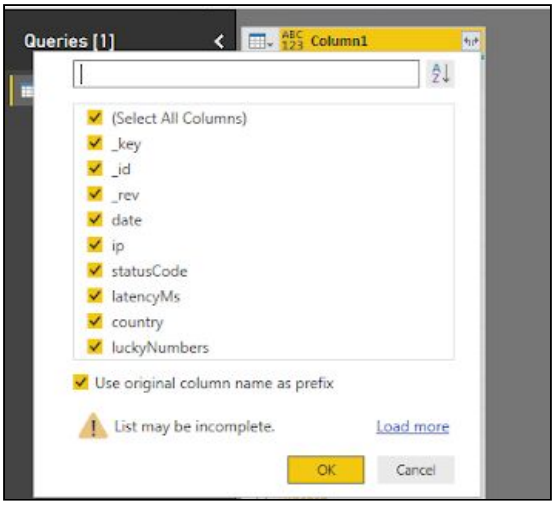
itself. They merely protect the Power BI connector against unauthorized access. Press the *Connect* button to proceed to the next step.

Data selection

If the connection succeeded, you should now see a grid showing page metadata. In the next steps we'll create a function to fetch each page of documents and a Power Query that uses this function to fetch all available pages.



First click on the `List` value in the row with the title **records**, then right-click on the grid's column header and select *To Table* to convert the list to a table. There is no need to change any of the options in the dialog, simply press *OK* to proceed.

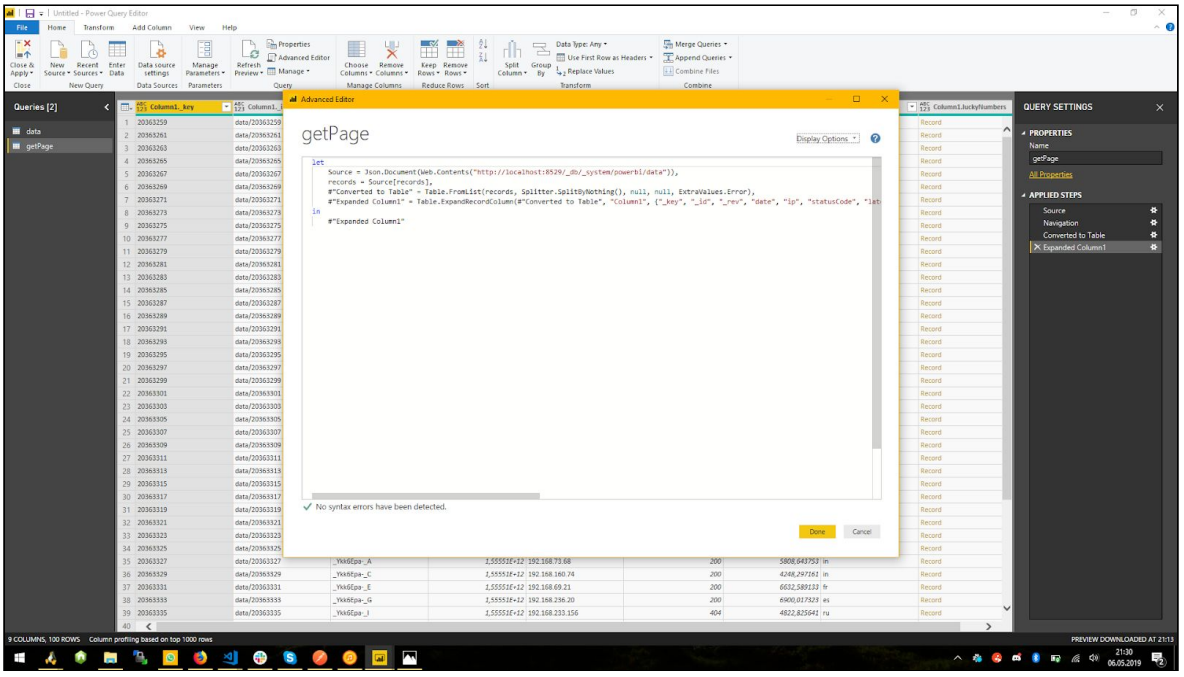


Next we need to expand the records to separate columns. Press the *Expand* icon in the column header and confirm the expansion by pressing the *OK* button. This gives us a table containing all data from the first page of results.

Creating the pagination function

Start by duplicate the `data` query by right-clicking on the query in the right sidebar and selecting *Duplicate*, then rename the duplicate query by right-clicking on it and selecting *Rename*. We'll turn this query into a function, so change the name to `getPage`.

Next right-click again on the `getPage` query and select the *Advanced Editor*.



The first lines should look like this:

```
let
  Source = Json.Document(Web.Contents(
    "http://localhost:8529/_db/_system/powerbi/data"
  )),
  records = Source[records],
```

If the URL you entered during the first steps looks different, the URL in the editor should match yours instead.

We'll turn the query into a function by changing those lines to this:

```
(page as number) => let
  Source = Json.Document(Web.Contents(
    "http://localhost:8529/_db/_system/powerbi/data?page="
    & Text.From(page)
  )),
  records = Source[records],
```

If your URL looked different, make sure to add the `?page=" & Text.From(page)` suffix as above.

The editor should not indicate any syntax errors. Press *Done* to proceed to the next step.

Fetching all pages

Switch back to the `data` query we created initially and in the right sidebar undo all steps after the first one by clicking on the small *Delete* icon that appears when hovering over each step. You should remove these steps in reverse order by starting from the bottom but any errors caused by undoing these steps out of sequence should disappear once you've undone all steps after the first.

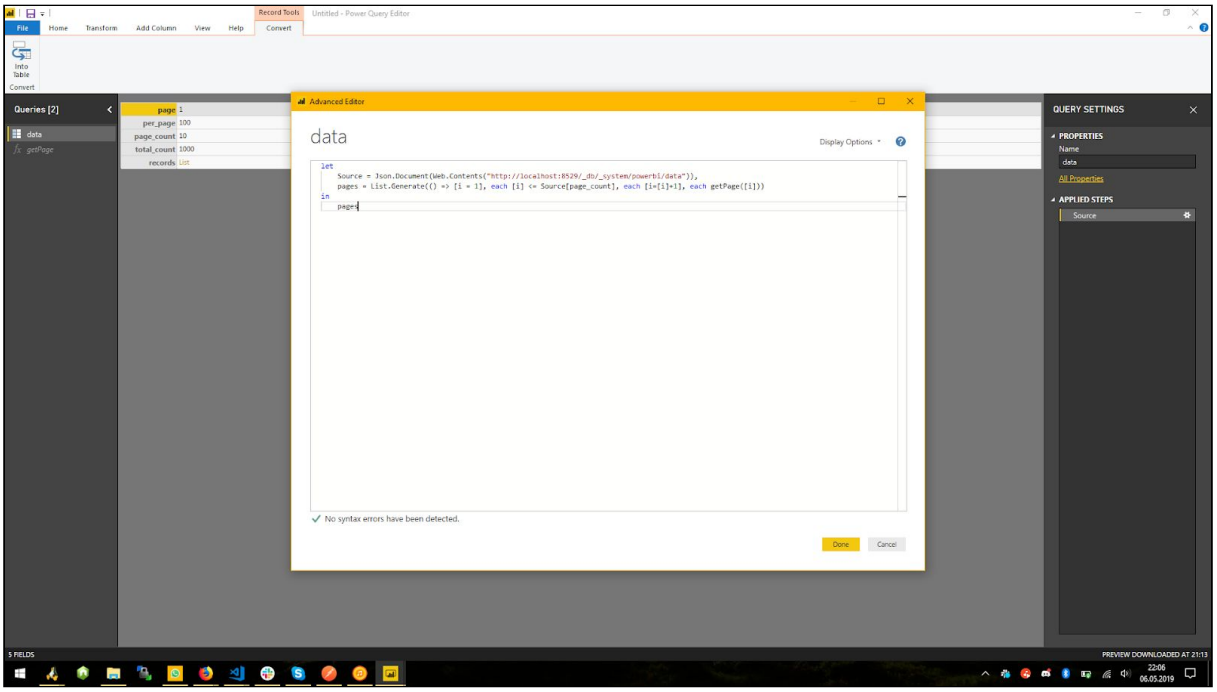
Right-click on the `data` query and select *Advanced Editor* to edit the query.

The query should now look simply like this:

```
let
  Source = Json.Document(Web.Contents(
    "http://localhost:8529/_db/_system/powerbi/data"
  ))
in
  Source
```

Adjust the query so it looks like this:

```
let
  Source = Json.Document(Web.Contents(
    "http://localhost:8529/_db/_system/powerbi/data"
  )),
  pages = List.Generate(() => [i = 1], each [i] <= Source[page_count],
    each [i=[i]+1], each getPage([i]))
in
  pages
```



This will generate a list of results for each page returned by the endpoint. Note that again the URL should match the URL you entered initially.

The editor should again not indicate any syntax errors. Confirm the changes by pressing *Done*.

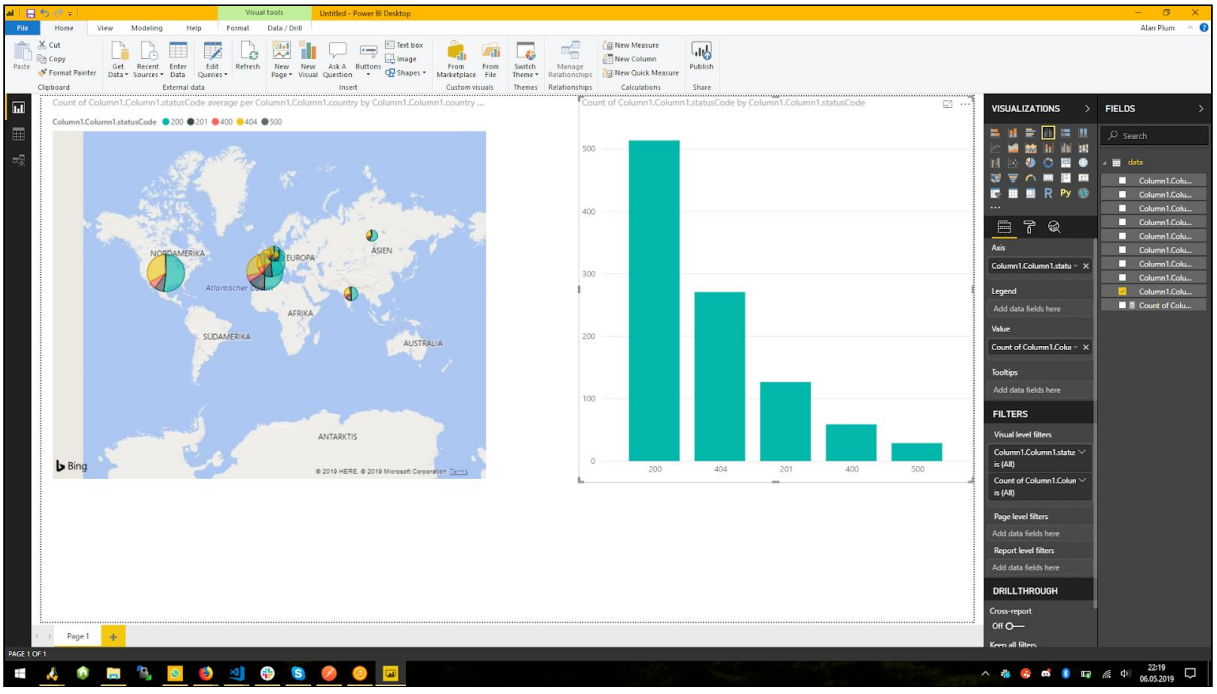
The data grid should now show a list of tables instead. Right-click on the header and select *To Table* to convert the list to a table. You can leave the defaults unchanged and press *OK* to continue.

Next we need to expand the column again by pressing the *Expand* icon in the column header and accepting the defaults by pressing *OK*.

This expands the table to contain all documents from the collection. Press *Close & Apply* in the toolbar to complete the import and proceed.

Creating a worksheet

With all the data imported into Power BI, you can now visualize it by dragging fields onto the worksheet.



Extending the connector with filters

As the Power BI Web connector is a normal Foxx service you can modify the source code to create your own connector. In this section we will extend the existing connector with an option to filter the collection dynamically before handing it over to Power BI.

Using the web interface open the settings tab also used to configure the service earlier and press the download icon in the upper right. This downloads a zip bundle of the service's source code.

Extract the zip archive to any folder on your computer and open the file `index.js` in a code editor.

Defining the operators

There are a lot of useful operators in AQL but for this example we'll stick to the basics. Add the following in the source code before the line starting with `const COLLECTIONS`:

```
const OPERATORS = new Map([
  ["lt", aql.literal("<")],
  ["lte", aql.literal("<=")],
  ["gt", aql.literal(">")],
  ["gte", aql.literal(">=")],
  ["eq", aql.literal("==")],
  ["neq", aql.literal("!=")],
  ["in", aql.literal("in")],
  ["nin", aql.literal("not in")]
]);
```

This gives us a mapping of safe but human-readable names to AQL operators. The `aql.literal` function converts the strings to something we can use in an AQL query template without having to worry about being misinterpreted as a bind variable.

Extending the query parameters

We want to allow users to specify multiple filters. The easiest way to do this with the existing GET route is by adding a query parameter that takes a JSON value.

Find one of the lines starting with `.queryParam(` and add the following immediately before that line:

```
  .queryParam(
    "filters",
    joi
      .array()
      .items(
        joi
          .object()
          .keys({
            fieldName: joi.string().required(),
            operator: joi.only(...OPERATORS.keys()).required(),
            value: joi.any().required()
          })
          .required()
      )
      .optional(),
    "Filter expressions to match the documents against."
  )
```

In plain English this matches an optional JSON array containing objects with three attributes:

- **fieldName**: a string value we will use to decide which field to compare
- **operator**: one of the operator names we defined earlier
- **value**: a value the field will be compared to using the operator

For example, this would limit the results to documents with a `statusCode` field set to either `400` or `500`:

```
[{ "fieldName": "statusCode", "operator": "in", "value": [400, 500] }]
```

Applying the filter

Find the following lines in the source code:

```
const { page, per_page } = req.queryParams;
const start = (page - 1) * per_page;

const { query, bindVars } = aql`
  FOR doc IN ${collection}
  LIMIT ${start}, ${per_page}
  RETURN doc
`;
```

Replace those lines with the following code:

```
const { page, per_page, filters: rawFilters } = req.queryParams;
const start = (page - 1) * per_page;

const filters = rawFilters
  ? rawFilters.map(
      ({ fieldName, operator, value }) =>
        aql`FILTER doc[${fieldName}] ${OPERATORS.get(operator)} ${value}`
    )
  : [];
```

```
const { query, bindVars } = aql`
  FOR doc IN ${collection}
  ${aql.join(filters)}
  LIMIT ${start}, ${per_page}
  RETURN doc
`;
```

Installing the modified connector

To reflect the changes to the source code in the installed service, first create a zip archive of your working copy with the saved changes to the `index.js` file.

To upgrade the service using the web interface:

1. Open the service's *Settings* tab and press the *Replace* button.
2. Open the *Upload* tab and press the *Upload File* button to select the zip file.
3. Press the *Replace* button, don't modify any of the options.
4. Confirm the dialog by pressing the *Replace* button.

To upgrade the service using Foxx CLI (assuming filename `powerbi.zip` and that the file is in the current directory):

```
foxx upgrade -u root -P -H http://localhost:8529 -D _system /powerbi powerbi.zip
```

Using the filter in Power BI Desktop

Press the *Edit Queries* button in the toolbar or follow the instructions to add a new data source using the Power BI Web connector.

We're going to use the following filter expression:

```
[{"fieldName":"statusCode","operator":"in","value":[400,500]}]
```

Edit the `getPage` query to use the following URL:

```
"http://localhost:8529/_db/_system/powerbi/data?filters=[{""fieldName"":"
"statusCode"",""operator"":""in"",""value"":[400,500]}]&page=" &
Text.From(page)
```

Edit the `data` query to use the following URL:

```
"http://localhost:8529/_db/_system/powerbi/data?filters=[{""fieldName"":"
"statusCode"",""operator"":""in"",""value"":[400,500]}]"
```

Note that you can substitute whatever filters make sense for your data instead but quotation marks need to be escaped by using two quotation marks instead of one.

You may need to press the *Refresh Preview* > *Refresh All* button in the toolbar for the changes to take effect.

Once you're satisfied with the results of your filter expression, press the *Close & Apply* button to return to the worksheet.