BlocK-DaQ – Getting Started

# External Prerequisites

In order to facilitate an easy start with BlocK-DaQ and no version conflicts, we provide the files for the three required external tools within the GitHub project.

1. **InfluxDB** (required)

We used InfluxDB to store the CDQM time series data. Since the Java API was not working by the time of the creation, we used the InfluxDB browser version. BlocK-DaQ does not offer an embedded mode (like Derby or GraphDB), but runs InfluxDB outside the Java runtime to persist CDQM results over time.

Run on Linux:

startInflux.sh (for starting InfluxDB server)

startInfluxConsole.sh (for querying InfluxDB using the console)

Run on Windows:

InfluxDB\influxdb-1.7.7-1\_windows\influxd.exe

1. **Grafana** (required)

Grafana is a browser-based dashboard for visualization. After the start, a browser window is opened with the dashboard URL (http://localhost:3000 by default).

Run on Linux:

startGrafana.sh

Run on Windows:

Grafana\grafana-6.2.5\_windows\bin\grafana-server.exe

1. **GraphDB** (optional since embedded version can be used too; this is only for managing the repositories with a GUI and visualize them)

Run on Windows:

GraphDB\ GraphDB\_Free-9.0.0.exe

Open GraphDB Server and Workbench in browser:

<http://localhost:7200/>

Quick start guide for further usage:

<http://graphdb.ontotext.com/free/quick-start-guide.html>

# Use DSD Connectors to Connect to Data Sources

Currently, there are implementations for

* CSV files
* Cassandra DBs
* DSD files
* MySQL DBs
* Oracle DBs (still need to be verified and comprehensively tested)

To each of those data sources, it is possible to create a connection using a DSConnector (only accessing the schema, e.g., sufficient for DSD files), or DSInstanceConnector (for accessing schema + instances of a data source).

ConnectorCSV conn1 = new ConnectorCSV("path/DataCoSupplyChainDataset.csv", ",",

"\n", "SupplyChain");

ConnectorMySQL conn2 = ConnectorMySQL.getInstance("jdbc:mysql://localhost:port",

"dbname", "user", "pw");

In order to transform the schema description of the local data source to virtual DSD elements in the Java runtime environment, it is necessary to load the schema information.

Datasource dsSC = conn1.loadSchema();

Datasource dsCE = conn2.loadSchema();

# Creation of a Knowledge Graph to Manage all Datasources

To conveniently handle multiple DSD data source objects, we developed the class DSDKnowledgeGraph, which holds all data sources and their connections.

DSDKnowledgeGraph kg = new DSDKnowledgeGraph("automotive");

kg.addDatasourceAndConnector(dsSC, conn1);

kg.addDatasourceAndConnector(dsCE, conn2);

The KG class holds an instance of an embedded GraphDB, which persists the triples for later reuse and offers methods for export the KG or to annotate all contained data sources (with their concepts and attributes) with RDPs.

# Creation of a Reference Data Profile

To create and annotate a reference data profile to a DSD element, a corresponding method .annotateProfile(RecordSet)is provided, which needs to be called for each DSD element that should be annotated. In the following example, each attribute of each concept in the Northwind (NW) is annotated with a data profile that uses the first 5,000 records of the respective concept (table).

for (Concept c : ds.getConceptsAndAssociations()) {

RecordList rs = conn.getPartialRecordSet(c, 0, 5000);

for (Attribute a : c.getAttributes()) {

a.annotateProfile(rs);

}

}

Instead of annotating the RDPs for single DSD elements, it is possible to traverse all elements in the KG and to initiate the creation of an RDP for each automatically.

kg.addDataProfile(1000);

The parameter denotes the number of records used to calculate the RDP. Here, we used the first 1000 records.

# Continuously Monitor the Data Quality