JDBC/ODBC Driver Implementation for WSO2 Data Analytics Server

JDBC Driver V. 1.0.0

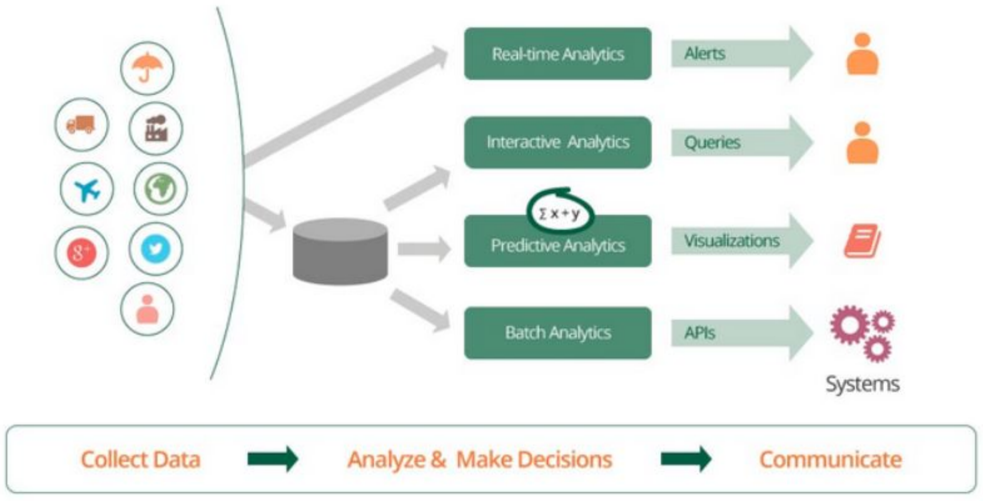
By Anupama Pathirage

Table of Content

1. **Problem Background**

As a part of WSO2's analytics platform, WSO2 DAS introduces a single solution with the ability to build systems and applications that collect and analyze both realtime and persisted, data and communicate the results.WSO2 DAS workflow consists of the following three main phases.

- Collecting Data
- Analyzing Data
- Communicating Results



The final step in business activity monitoring is to present the analyzed information. It enables decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns.To query and present the analyzed information, currently the following methods are supported by the WSO2 DAS [1].

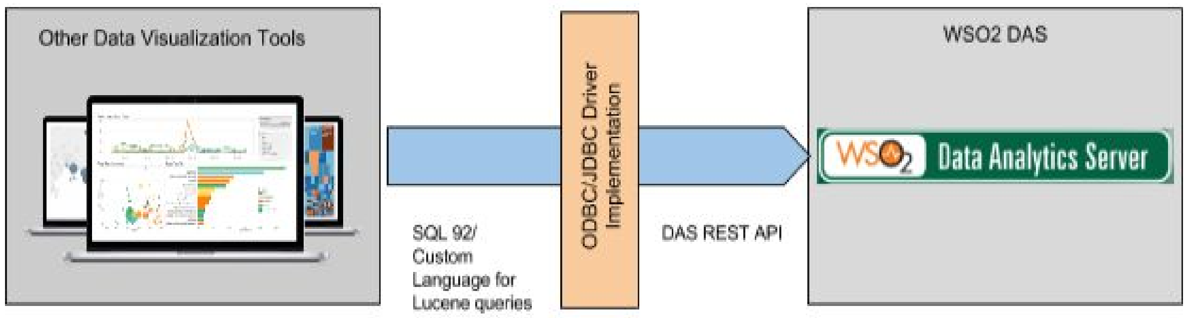
- Visualizing Results via Analytics Dashboard
- Creating Alerts
- Communicating Results Through REST API
- Analytics JavaScript (JS) API

Problem:

Although WSO2 DAS comes with Analytics Dashboard and REST/JS API, currently the ability to connect with other external data visualization/charting tools is limited. Since the DAS has its own data abstraction layer, it is not possible to directly go to the native data source and read it. Today's data visualization tools are highly sophisticated and there's a rash of new tools because visual discovery is in big demand. So having an interface which can interact with outside data visualization tools will add more value to WSO2 DAS.

2. Proposed Solution

Most of the data visualization products include the ability to connect to a wide variety of data sources. Many of these data sources are implemented as native connections, which mean they have implemented techniques, capabilities and optimizations specific to these data sources. Additionally, data visualization products have implemented connectors to use the general-purpose ODBC or JDBC standard to access data sources beyond the list of named options available when creating a new connection. Even though DAS has the REST API to read data, it is required to have a JDBC/ODBC compatible interface to integrate with existing tools.


Other Data Visualization Tools
SQL 92/ Custom Language for Lucene queries
ODBC/JDBC Driver Implementation
DAS REST API
WSO2 DAS
WSO2 Data Analytics Server

So following two driver support is proposed for the WSO2 DAS.

1. ODBC Driver - Allows ODBC compliant application to use WSO2 DAS as a data source.

2. JDBC Driver - Allows Java application to use WSO2 DAS as the data source.

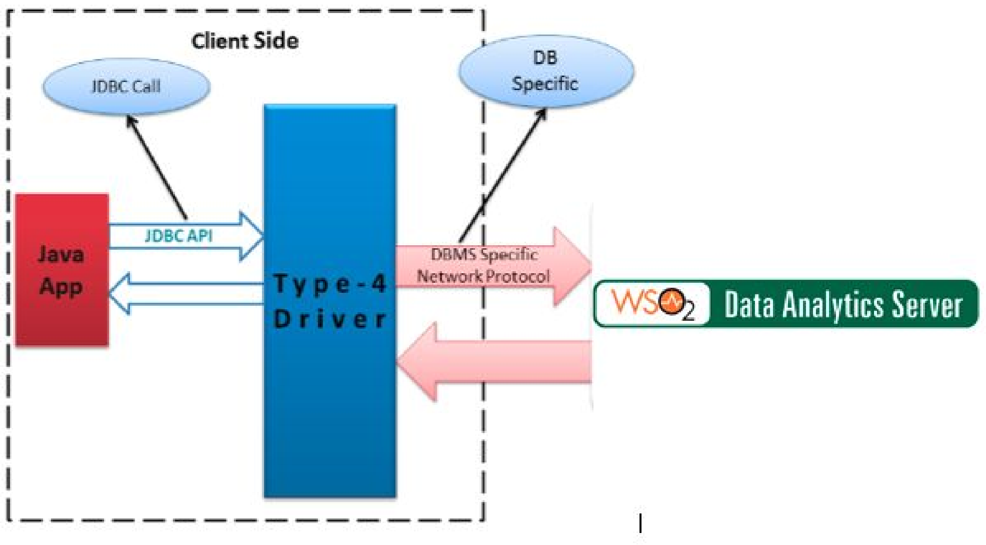
3. Example Usages of the Proposed Solution

Following are some data visualization products which provide JDBC/ODBC connection support. There are lot of other data visualization products/tools which gives support for them.

- Tableau - A Data Visualization platform based on BI. Other than the list of named data source options, it provide support for custom connections via standard ODBC drivers. [2]
- QlikView - A Business Intelligence platform which brings a whole new level of analysis, insight, and value to existing data stores with user interfaces that are clean, simple, and straightforward. It support custom connections for data sources via JDBC drivers.[3]
- Power BI- Provide report/dashboard generation support for data sources connected via ODBC [4]
- Microstrategy Visual Insight- Data visualization tool which is a customized, interactive display that can be used to explore business data. Provide support for ODBC drivers.

4. High Level Architecture

JDBC provides a programming-level interface for uniformly communicating with a database or a data source. To use the JDBC API with a particular data source , a JDBC driver is required to mediate between JDBC technology and the database.



JDBC drivers divide into four types or levels. Each type defines a JDBC driver implementation with increasingly higher levels of platform independence, performance, and deployment administration. The four types are:

- Type 1: JDBC-ODBC (Open Database Connectivity) Bridge
- Type 2: Native-API, partly Java driver
- Type 3: Network-protocol, all-Java driver
- Type 4: Native-protocol, all-Java driver

This DAS JDBC driver is implemented in Java and directly speaks to DAS service using its native protocol. It belongs to the Type 4 driver category and it includes all database call in one JAR file, which makes it very easy to use. Because of light weight, this is also known as thin JDBC driver. Since this driver is also written in pure Java, it is portable across all platform.

5. SQL functionalities

Following SQL Functionalities are supported by the DAS JDBC Driver.

Database and Table Metadata Retrieval :

- Get Database/Table Metadata
- Get Column Metadata
- Get Primary Keys of a given table
- Get Index columns of a given table

Select Queries :

- Select *

  SELECT * FROM overused_devices
- Select with column names

  SELECT house_id,state,metro_ area,device_id,timestamp FROM overused_devices
- Select with where clause (supports =, <>/!=, <, >, <=, >=)

  SELECT * FROM overused_devices WHERE house_id = 15

  SELECT * FROM overused_devices WHERE timestamp > 1455777700195
- And operation

  SELECT * FROM overused_devices WHERE house_id = 15 AND metro_area ='San Francisco'
- Or Operation

  SELECT * FROM overused_devices WHERE house_id = 15 OR metro_area='San Francisco'

Prepared Statement Support:

SELECT * FROM OVERUSED_DEVICES WHERE HOUSE_ID = ? AND METRO_AREA = ?

Aggregate Functions :

- MIN - returns the smallest value in a given column
- MAX-returns the largest value in a given column
- SUM-returns the sum of the numeric values in a given column
- COUNT- returns the total number of values in a given column
- COUNT(*) - returns the number of rows in a table

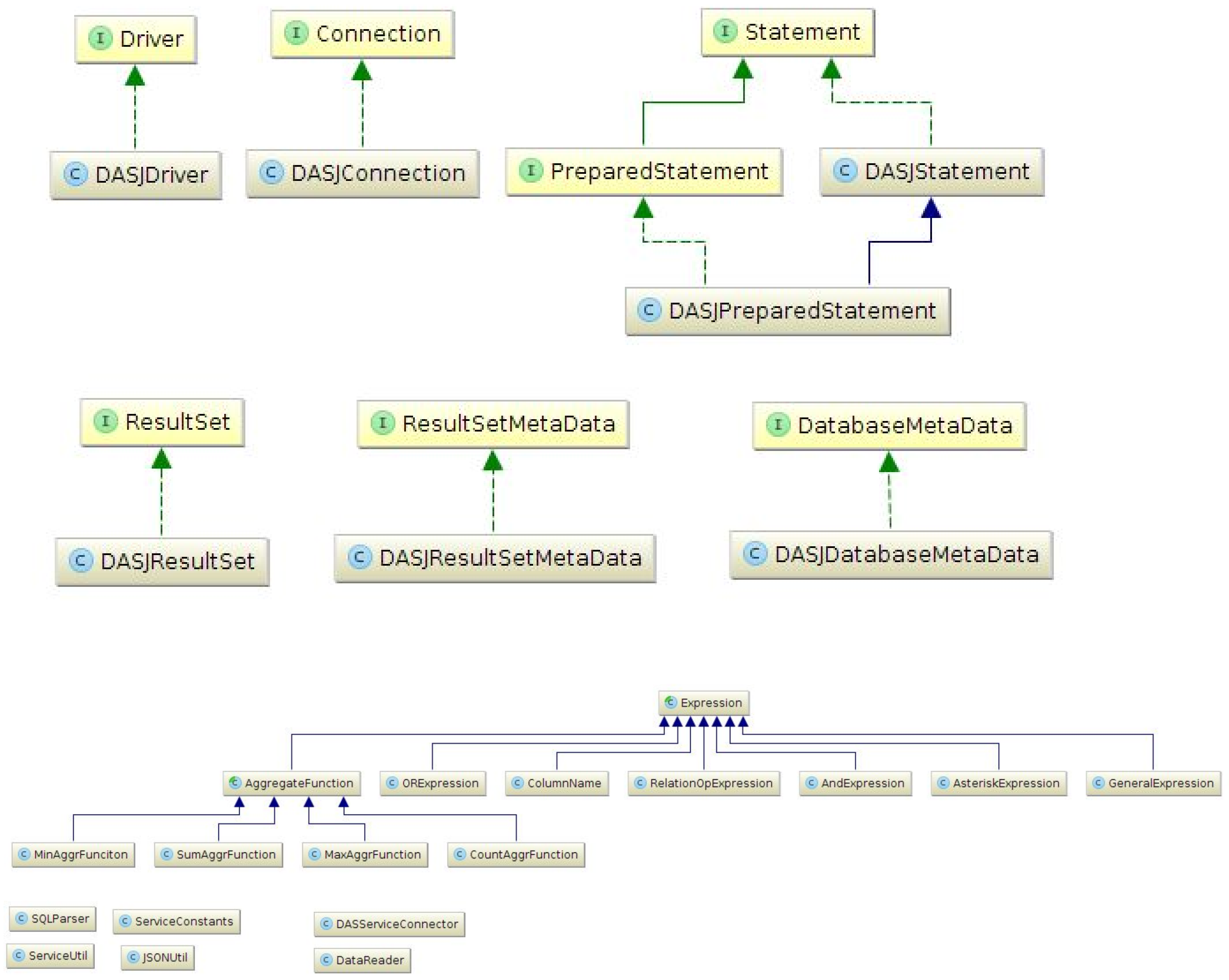
Default Connection Parameters:

Jdbc url : https://localhost:9443/analytics/

User Name : admin
Password: admn

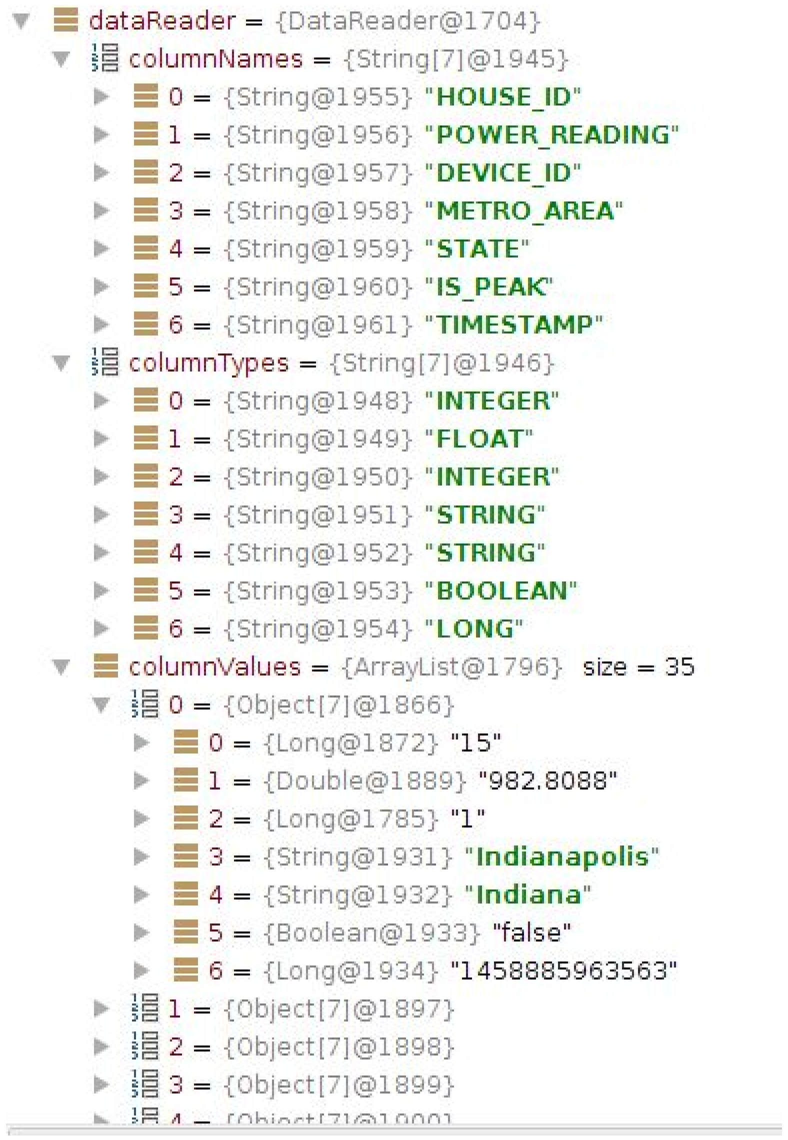
6. Implementation Details

The class diagram for the JDBC driver is as follows.



Some of the important Data Structures used in JDBC driver are as follows.

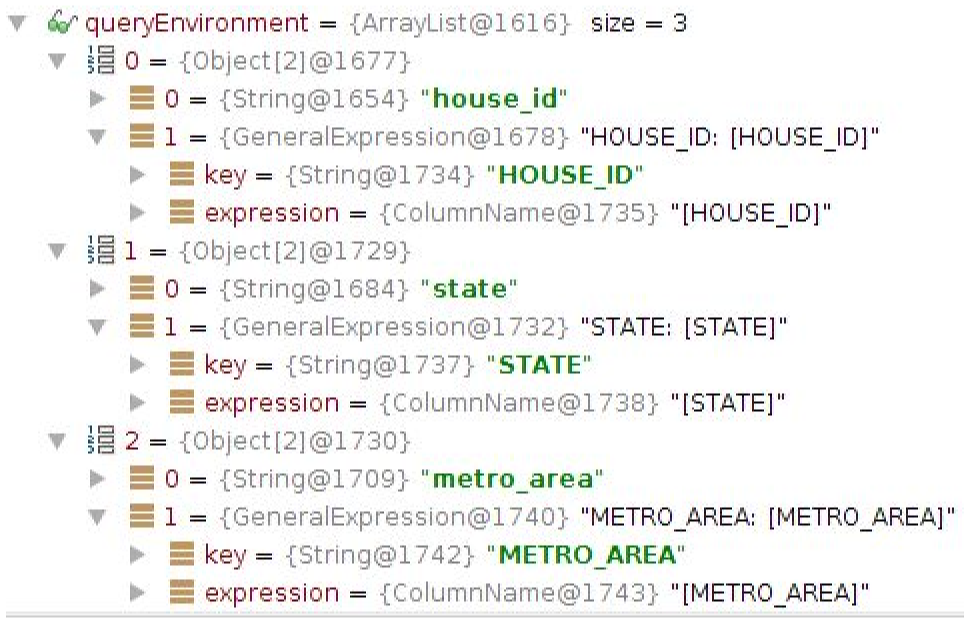
- DataReader - Contains the table data and column details for a particular table in a query



```
▼ dataReader = {DataReader@1704}
  ▼ columnNames = {String[7]@1945}
    ▶ 0 = {String@1955} "HOUSE_ID"
    ▶ 1 = {String@1956} "POWER_READING"
    ▶ 2 = {String@1957} "DEVICE_ID"
    ▶ 3 = {String@1958} "METRO_AREA"
    ▶ 4 = {String@1959} "STATE"
    ▶ 5 = {String@1960} "IS_PEAK"
    ▶ 6 = {String@1961} "TIMESTAMP"
  ▼ columnTypes = {String[7]@1946}
    ▶ 0 = {String@1948} "INTEGER"
    ▶ 1 = {String@1949} "FLOAT"
    ▶ 2 = {String@1950} "INTEGER"
    ▶ 3 = {String@1951} "STRING"
    ▶ 4 = {String@1952} "STRING"
    ▶ 5 = {String@1953} "BOOLEAN"
    ▶ 6 = {String@1954} "LONG"
  ▼ columnValues = {ArrayList@1796}  size = 35
    ▼ 0 = {Object[7]@1866}
      ▶ 0 = {Long@1872} "15"
      ▶ 1 = {Double@1889} "982.8088"
      ▶ 2 = {Long@1785} "1"
      ▶ 3 = {String@1931} "Indianapolis"
      ▶ 4 = {String@1932} "Indiana"
      ▶ 5 = {Boolean@1933} "false"
      ▶ 6 = {Long@1934} "1458885963563"
    ▶ 1 = {Object[7]@1897}
    ▶ 2 = {Object[7]@1898}
    ▶ 3 = {Object[7]@1899}
```

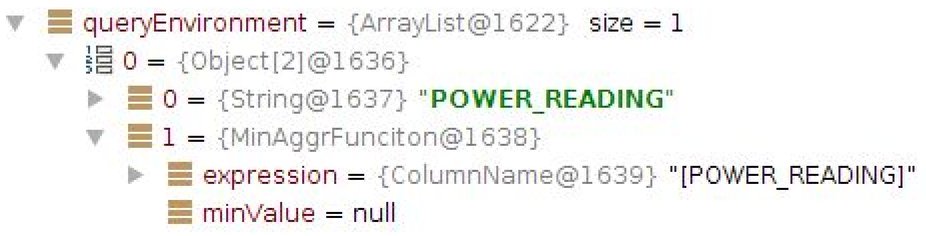
- QueryEnvironment = Contains the column name list given in the select query.

Ex: **SELECT house_id, state, metro_area FROM overused_devices**



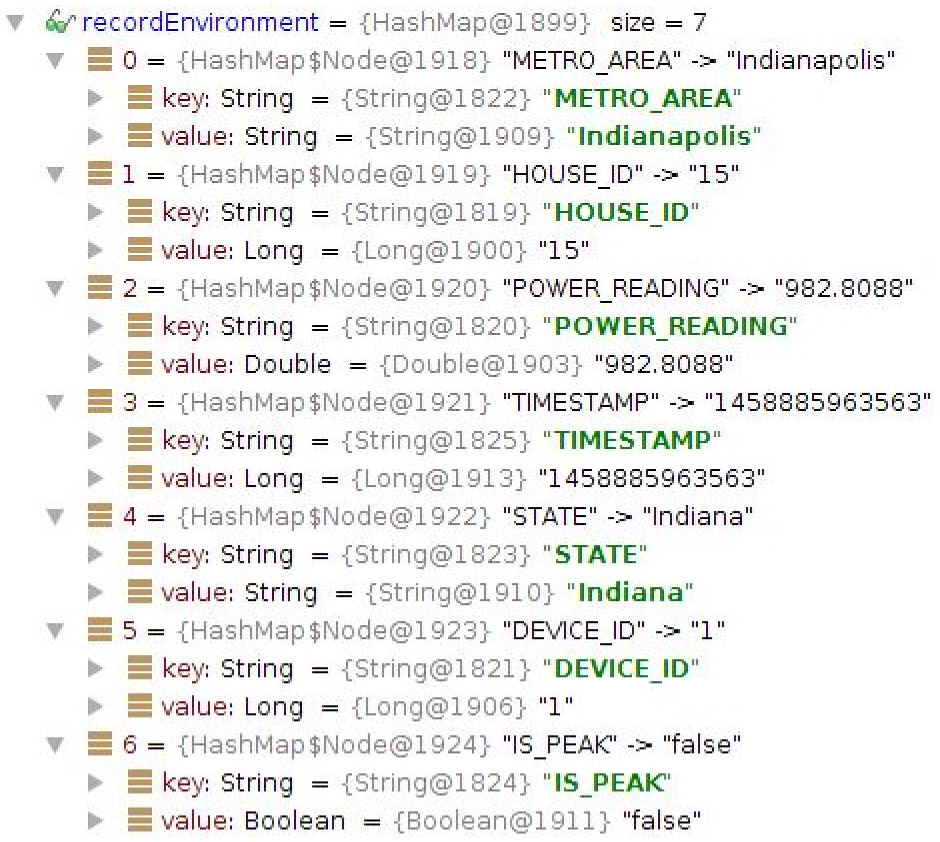
```
▼ queryEnvironment = {ArrayList@1616}  size = 3
  ▼ 0 = {Object[2]@1677}
    ▶ 0 = {String@1654} "house_id"
    ▼ 1 = {GeneralExpression@1678} "HOUSE_ID: [HOUSE_ID]"
      ▶ key = {String@1734} "HOUSE_ID"
      ▶ expression = {ColumnName@1735} "[HOUSE_ID]"
  ▼ 1 = {Object[2]@1729}
    ▶ 0 = {String@1684} "state"
    ▼ 1 = {GeneralExpression@1732} "STATE: [STATE]"
      ▶ key = {String@1737} "STATE"
      ▶ expression = {ColumnName@1738} "[STATE]"
  ▼ 2 = {Object[2]@1730}
    ▶ 0 = {String@1709} "metro_area"
    ▼ 1 = {GeneralExpression@1740} "METRO_AREA: [METRO_AREA]"
      ▶ key = {String@1742} "METRO_AREA"
      ▶ expression = {ColumnName@1743} "[METRO_AREA]"
```

Ex: Select **SELECT MIN(power_reading) FROM overused_devices**



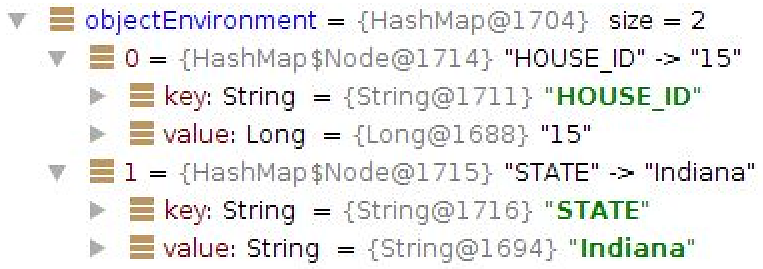
```
▼ queryEnvironment = {ArrayList@1622}  size = 1
  ▼ 0 = {Object[2]@1636}
    ▶ 0 = {String@1637} "POWER_READING"
    ▼ 1 = {MinAggrFunciton@1638}
      ▶ expression = {ColumnName@1639} "[POWER_READING]"
        minValue = null
```

- RecordEnvironment = Contains the values of the current record retrived by next function.



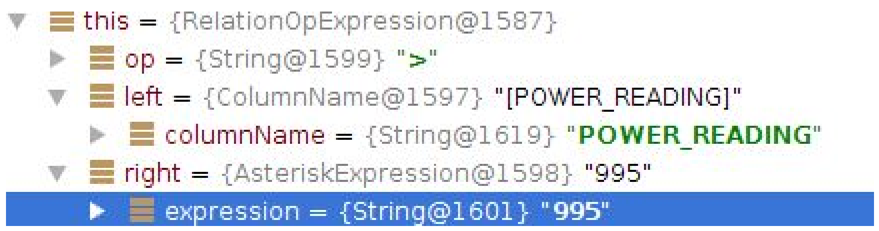
```
▼ recordEnvironment = {HashMap@1899}  size = 7
  ▼ 0 = {HashMap$Node@1918} "METRO_AREA" -> "Indianapolis"
    ▶ key: String  = {String@1822} "METRO_AREA"
    ▶ value: String  = {String@1909} "Indianapolis"
  ▼ 1 = {HashMap$Node@1919} "HOUSE_ID" -> "15"
    ▶ key: String  = {String@1819} "HOUSE_ID"
    ▶ value: Long  = {Long@1900} "15"
  ▼ 2 = {HashMap$Node@1920} "POWER_READING" -> "982.8088"
    ▶ key: String  = {String@1820} "POWER_READING"
    ▶ value: Double  = {Double@1903} "982.8088"
  ▼ 3 = {HashMap$Node@1921} "TIMESTAMP" -> "1458885963563"
    ▶ key: String  = {String@1825} "TIMESTAMP"
    ▶ value: Long  = {Long@1913} "1458885963563"
  ▼ 4 = {HashMap$Node@1922} "STATE" -> "Indiana"
    ▶ key: String  = {String@1823} "STATE"
    ▶ value: String  = {String@1910} "Indiana"
  ▼ 5 = {HashMap$Node@1923} "DEVICE_ID" -> "1"
    ▶ key: String  = {String@1821} "DEVICE_ID"
    ▶ value: Long  = {Long@1906} "1"
  ▼ 6 = {HashMap$Node@1924} "IS_PEAK" -> "false"
    ▶ key: String  = {String@1824} "IS_PEAK"
    ▶ value: Boolean  = {Boolean@1911} "false"
```

- ObjectEnvironment - Contains the values of the current record for the requested column name list + column names in the where clause.



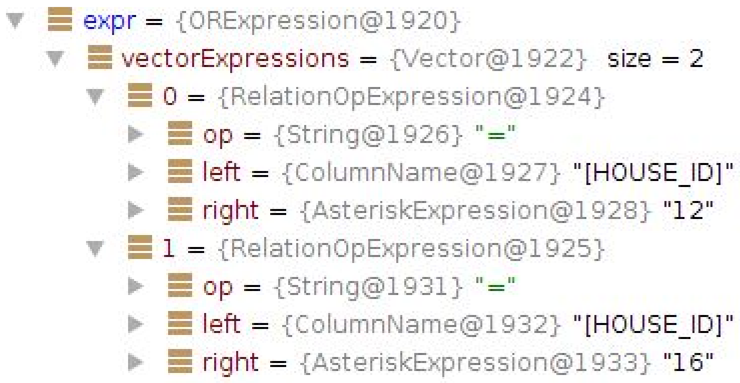
```
▼ objectEnvironment = {HashMap@1704}  size = 2
  ▼ 0 = {HashMap$Node@1714} "HOUSE_ID" -> "15"
    ▶ key: String  = {String@1711} "HOUSE_ID"
    ▶ value: Long  = {Long@1688} "15"
  ▼ 1 = {HashMap$Node@1715} "STATE" -> "Indiana"
    ▶ key: String  = {String@1716} "STATE"
    ▶ value: String  = {String@1694} "Indiana"
```

- RelationOperation Expression



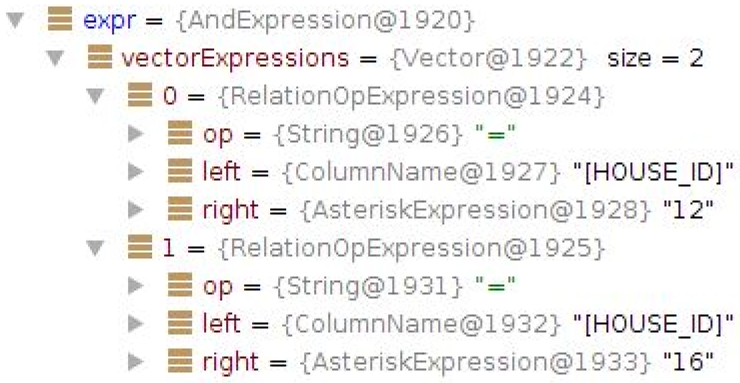
```
▼ this = {RelationOpExpression@1587}
  ▶ op = {String@1599} ">"
  ▼ left = {ColumnName@1597} "[POWER_READING]"
    ▶ columnName = {String@1619} "POWER_READING"
  ▼ right = {AsteriskExpression@1598} "995"
    ▶ expression = {String@1601} "995"
```

- OR Expression



```
▼ expr = {ORExpression@1920}
  ▼ vectorExpressions = {Vector@1922}  size = 2
    ▼ 0 = {RelationOpExpression@1924}
      ▶ op = {String@1926} "="
      ▶ left = {ColumnName@1927} "[HOUSE_ID]"
      ▶ right = {AsteriskExpression@1928} "12"
    ▼ 1 = {RelationOpExpression@1925}
      ▶ op = {String@1931} "="
      ▶ left = {ColumnName@1932} "[HOUSE_ID]"
      ▶ right = {AsteriskExpression@1933} "16"
```

- And Expression



```
▼ expr = {AndExpression@1920}
  ▼ vectorExpressions = {Vector@1922}  size = 2
    ▼ 0 = {RelationOpExpression@1924}
      ▶ op = {String@1926} "="
      ▶ left = {ColumnName@1927} "[HOUSE_ID]"
      ▶ right = {AsteriskExpression@1928} "12"
    ▼ 1 = {RelationOpExpression@1925}
      ▶ op = {String@1931} "="
      ▶ left = {ColumnName@1932} "[HOUSE_ID]"
      ▶ right = {AsteriskExpression@1933} "16"
```

7. Future Work

In the current version of DAS JDBC Driver following features are not supported.

- Improve the where clause filtering to use DAS primary key support and Lucene indexing.
- Add record limit support for select query.

8. References:

[1] https://docs.wso2.com/display/DAS301/WSO2+Data+Analytics+Server+Documentation
[2] http://kb.tableau.com/articles/knowledgebase/tableau-and-odbc
[3]http://market.qlik.com/qlikview-jdbc-connector.html
[4] https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-data-sources/
[5] http://www2.microstrategy.com/producthelp/9.3/WebUser/WebHelp/Lang_1033/creating_an_analysis.htm
[6] https://docs.wso2.com/display/DAS301/Analytics+REST+API+Guide