Esper JDBC Driver

Version 5.4.0

by *EsperTech Inc.* [http://www.espertech.com]

Preface

This document describes the Esper JDBC driver. The document assumes that the reader has prior knowledge of Esper.

If you are new to Esper, please study some of the tutorials and case studies available on the public web site at `http://www.espertech.com/esper`, and skim over the reference documentation.

The *Chapter 1, Overview* chapter is the best place to start.

Chapter 1. Overview

The Esper JDBC driver provides programmatic access to the information computed by Esper through the standardized JDBC API.

One goal of the JDBC driver is to allow reporting and charting software to present information derived by EPL continues queries. A second goal is to provide a familiar API to programmers.

Via the JDBC driver your favorite JDBC-capable tool and your application can perform the following functions:

1. Query an existing Esper EPL statement to obtain the current results of the statement and metadata.
2. Dynamically execute non-continuous, fire-and-forget queries against named windows held by an Esper engine instance.

In the next section, *Chapter 2, Concepts*, we explain Esper EPL statements in the terminology of JDBC, assuming the reader has prior knowledge of Esper.

The JDBC driver is a type IV driver that supports both *local* (embedded) and *remote* (network) operation.

In *local* operation, your application, the JDBC driver and the Esper engine instance(s) reside in the same Java virtual machine on the same host. This mode is further described in *Chapter 3, Local Driver*.

Via *remote* operation, your application and the JDBC driver can reside on a different Java VM and host then the Esper engine instance(s). Please consult *Chapter 4, Remote Server and Driver* for use of the remote network server and remote JDBC driver.

The section *Chapter 5, Driver Parameters* explains common driver parameters.

Chapter 6, Examples creates a report using Eclipse BIRT for reporting and charting, Esper for computing continuous query results, and both communicating via JDBC.

The last chapter is a reference of compliance to JDBC API standards; See *Chapter 7, Compliance, Features and Limitations*. The JDBC driver is compliant to the JDBC 3 API standard with exceptions. Please make sure to review driver limitations.

Chapter 2. Concepts

2.1. Catalogs

As documented for Esper, multiple Esper engine instances can co-exist within the same JVM. A provider URI uniquely identifies each Esper provider (engine) instance. The JDBC driver represents each available (initialized) Esper engine instance URI as a catalog name, and the Esper default provider instance (no provider URI) as the catalog name `"default"`.

At connection time, your application can specify a provider URI via a driver URL parameter or property on the data source. Setting the catalog at the time you obtain a connection is equivalent to setting the catalog name on an existing connection.

2.2. Stored Procedures

Your application creates EPL statements within an Esper engine instance typically via the Esper administrative API. Each started EPL statement is visible as a stored procedure through JDBC.

The query result of a stored procedure is the events returned by the `safeIterator` method on `EPStatement`.

Any type of EPL statement can be invoked as a stored procedure via the JDBC driver. EPL statements that return query results include `select` and `insert into` clauses as well as `create variable` and `create window` clauses.

To retrieve results of started Esper EPL statements (stored procedures in JDBC terminology), your application may use either of the following syntax:

- The shortcut notation:

  ```
  [provider_URI.]<esper_statement_name>
  ```

  .
- The standard escape syntax :

  ```
  {call [provider_URI.]<esper_statement_name>}
  ```

  .

The *provider_URI* is equivalent to the JDBC catalog name and may optionally prefix the statement name.

For example, suppose your application creates an EPL statement:

```
String epl = "select symbol, avg(price) as avp from Tick group by symbol";
String name = "avgPriceBySymbol";
```

```
epAdministrator.createEPL(epl, name);
```

The code for the JDBC call via `PreparedStatement` may look like this:

```
String eplStmtName = "avgPriceBySymbol";
PreparedStatement stmt = connection.prepareStatement(eplStmtName);
ResultSet = stmt.executeQuery();
//... code to interrogate the result set
//... rs.next(); rs.getDouble("avp");
```

The equivalent call using the JDBC escape syntax is show here, and this example specifies a provider URI:

```
String query = "{call default.avgPriceBySymbol}";
PreparedStatement stmt = connection.prepareStatement(query);
```

2.3. Tables

Your application may create named windows via the `create window` clause. Such named windows are listed as tables in JDBC metadata. Your application can prepare and execute dynamic (non-continuous, fire-and-forget) EPL against named windows.

To demonstrate, assume your application creates a named window in an Esper engine instance:

```
String epl = "create window TickSummary.win:time(60 sec) as select symbol, price
 from Tick";
epAdministrator.createEPL(epl);
// ... more statements here to populate the named window from a stream
```

Via the JDBC driver the name window can be queried using a `java.sql.Statement` or `java.sql.PreparedStatement` as shown here:

```
String query = "select * from TickSummary";
PreparedStatement stmt = connection.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
//... code to interrogate the result set
//... rs.next(); rs.getDouble("price");
```

The dynamic EPL query can contain all elements of the Esper EPL query language, including joins.

The following limitations apply to dynamic EPL queries:

- There is no support for subqueries.
- Views are not allowed within dynamic EPL queries.
- The `prev` and `prior` operators are not allowed.
- The output rate limiting clause and insert-into clause are not honored by dynamic queries.

Chapter 3. Local Driver

The local JDBC driver requires that your application JDBC code, Esper engine instances and the driver reside in the same Java VM.

The following table outlines the connection information for the local driver:

Table 3.1. Local Driver

Item	Description
Jar file	esper-jdbcserver-*version*.jar
Jar dependencies	Same as Esper dependencies
`java.sql.Driver` class	com.espertech.esper.jdbc.local.EPLLocalJdbcDriver
URL	`jdbc:esper:local[/Esper_provider_URI] [&parameter=value[&...]]`
`javax.sql.DataSource` class	com.espertech.esper.jdbc.local.EPLLocalDataSource

The driver connection URL may specify a provider URI as part of the connection URL. If your application does not specify a provider URI as part of the URL, the driver connects to the Esper default provider instance. The literal `default` can also be specified to connect to the default provider instance.

Note that the JDBC driver does not allocate a new Esper engine instance, it only returns existing, already-allocated engine instances. Therefore, if the Esper engine instance by the name has not been initialized beforehand, the driver returns an error upon connecting.

The next code snippet connects to the default Esper engine instance using the `DriverManager` class:

```
EPServiceProvider service = EPServiceProviderManager.getDefaultProvider();
// ... create statements here ...
Class.forName("com.espertech.esper.jdbc.local.EPLLocalJdbcDriver");
String url = "jdbc:esper:local";
Connection connection = DriverManager.getConnection(url);
```

This example specifies the Esper engine instance named "primary" in the URL:

```
EPServiceProvider service = EPServiceProviderManager.getProvider("primary");
// ... create statements here ...
Class.forName("com.espertech.esper.jdbc.local.EPLLocalJdbcDriver");
String url = "jdbc:esper:local/primary";
```

```
Connection connection = DriverManager.getConnection(url);
```

An example that connects via `DataSource` to obtain a connection is:

```
EPLLocalDataSource localDataSource = new EPLLocalDataSource();
localDataSource.setProviderURI("primary");
Connection connection = localDataSource.getConnection();
```

Chapter 4. Remote Server and Driver

The JDBC remote network server and the JDBC remote driver allow your application JDBC code and driver to reside in a different Java VM and host as Esper engine instance(s). As part of the remote JDBC driver package, a remote server class acts as a socket server for incoming JDBC connections. Your application must first start a remote network server before connecting via the remote driver.

4.1. Remote Network Server

The remote network server requires the following dependent libraries in the classpath:

1. Apache MINA (mina-core-2.0.0-M1.jar) and its dependencies (slf4j-api-1.5.0.jar and slf4j-log4j12-1.5.0.jar).
2. The driver `esper-jdbc-driver-`*version*`.jar` and server `esper-jdbc-server-`*version*`.jar` jar files.

There are two options to start the EsperJDBC network server endpoint:

1. As a JDBC endpoint via the EsperJDBC endpoint API:

   Allows JDBC to be added to an already running Esper engine instance and provides more options for JDBC connectivity.

2. As an Esper plug-in:

   This requires your application to add EsperJDBC to the Esper configuration (XML or API). EsperJDBC is then started automatically as part of Esper engine initialization.

The above options are mutually exclusive.

4.1.1. Endpoint API

The class that provides an endpoint for JDBC connections thus acting as a remote network server for JDBC, is `com.espertech.esper.jdbc.server.remote.JDBCEndpoint`.

Your application can start a JDBC endpoint to listen on a given port, without any additional configuration, as follows:

```
JDBCEndpoint endpoint = new JDBCEndpoint(8450); // using port 8450
```

```
endpoint.start(); // this is a non-blocking call
...
endpoint.destroy();  // stop accepting connections
```

The endpoint can be configured further using the `JDBCEndpointConfiguration` class.

```
JDBCEndpointConfiguration config = new JDBCEndpointConfiguration();
config.setListenPort(8450);
config.setSessionIdleTimeout(600); // 10-miinute idle timeout

JDBCEndpoint endpoint = new JDBCEndpoint(config);
endpoint.start();
```

The next section outlines each remote network server configuration option:

Table 4.1. JDBC Endpoint Configuration

Option	Default	Description
Listen port	8450	The port the remote network server listens to for client connections.
Processor Count	1	The number of threads dedicated to handling client connections and query processing.
Session Idle Timeout	600 seconds (10 minutes)	The number of seconds that a client connection can be idle before the remote network server closes the connection and flushes the cache for the connection.
Filters	none	Filters can be registered with the remote network server to implement custom logic pertaining to client connections, such as authentication.

As part of the JDBC endpoint configuration, a filter implementation may be registered that is applied to new connections by the endpoint connection handler. The filter interface is `JDBCEndpointConnectionFilter`. By returning a value of `true` an incoming connection is accepted:

```
JDBCEndpointConfiguration config = new JDBCEndpointConfiguration();
config.setNewConnectionFilter(new JDBCEndpointConnectionFilter() {
  public boolean filter(ConnectionInformation connectionInformation) {
      String user = connectionInformation.getUser();
      String pwd = connectionInformation.getPassword();
      if (user.equals("myuser")) && (pwd.equals("mypassword"))) {
        return true;
      }
```

```
      return false;
  }
}
```

Additional filters that implement the Apache MINA interface `org.apache.mina.common.IOFilter` can be registered through the endpoint configuration, please check the Apache MINA documentation for available filters.

4.1.2. Plugin Configuration

In alternative to the endpoint API as described above, you may use EsperJDBC as a plug-in adapter to an Esper configuration. The Esper engine initialization then also initializes and starts the EsperJDBC endpoint.

Under this option EsperJDBC is initialized when your application first obtains an `EPServiceProvider` instance for a given URI or when your application calls the `initialize` method on an `EPServiceProvider`. EsperJDBC is destroyed when your application calls the `destroy` method on an `EPServiceProvider` instance or when it calls the `initialize` method on `EPServiceProvider` instance that had the adapter in its configuration.

This option configures the JDBC endpoint via properties passed in XML or `Properties` object.

4.1.2.1. Via Esper Configuration API

This section shows how to register EsperJDBC as part of Esper configuration using the Esper configuration API.

Your application must populate a `Properties` object to set EsperJDBC configuration options.

The sample code here sets the properties to use an JDBC server port 8450:

```
Configuration configuration = new Configuration();      // Esper Configuration
 class

// Properties are used to pass EsperJMX configuration
Properties properties = new Properties();
properties.put("port", "8450");
properties.put("processorCount", "1");
properties.put("sessionIdleTimeout", "600");

configuration.addPluginLoader(
  "EsperJDBC",
  "com.espertech.esper.jdbc.server.remote.JDBCServerPlugin",
  properties);
```

Note that is is not possible to add filters via the properties object configuration as above.

4.1.2.2. Via Esper Configuration XML

The XML as below configures an engine instance with EsperJDBC. It specifies the same configuration options as outlined above.

```
<esper-configuration>
  <plugin-loader name="EsperJDBC"
  class-name="com.espertech.esper.jdbc.server.remote.JDBCServerPlugin">
  <init-arg name="port" value="8450" />
  <init-arg name="processorCount" value="1" />
  <init-arg name="sessionIdleTimeout" value="600" />
   </plugin-loader>
</esper-configuration>
```

4.2. JDBC Remote Client

The following table outlines the connection information for the remote network driver:

Table 4.2. Remote Driver

Item	Description
Jar file	esper-jdbcdriver-*version*.jar
Jar dependencies	none
`java.sql.Driver` class	com.espertech.esper.jdbc.remote.EPLRemoteJdbcDriver
URL	`jdbc:esper:remote:`*`host`*`:`*`port`*`[/Esper_provider_URI][&parameter=value[&...]]`
`javax.sql.DataSource` class	com.espertech.esper.jdbc.remote.EPLRemoteDataSource

Similar to the local driver, the remote driver connection URL may specify a provider URI as part of the URL.

The next code snippet connects using the `DriverManager` class:

```
Class.forName("com.espertech.esper.jdbc.remote.EPLRemoteJdbcDriver");
// Using localhost (127.0.0.1) as hostname and default port
String url = "jdbc:esper:remote:127.0.01:8450";
Connection connection = DriverManager.getConnection(url);
```

An example that connects via `DataSource` to obtain a connection is:

```
EPLRemoteDataSource remoteDataSource = new EPLRemoteDataSource();
```

```
remoteDataSource.setServer("myhostname");
remoteDataSource.setPortNumber(8450);
Connection connection = remoteDataSource.getConnection();
```

4.2.1. JDBC Remote Driver Parameters

This section outlines optional parameters specific to the remote JDBC driver only. Common JDBC driver parameters applicable to both remote and local JDBC drivers are listed in *Chapter 5, Driver Parameters*.

Table 4.3. Remote Driver Parameters

Parameter	Description
socketTimeout	An integer-type parameter to define the socket timeout, default is 0 (zero). The amount of time to wait (in seconds) for network activity before timing out. Use with care! A zero value is an infinite timeout. If a non-zero value is supplied this must be greater than the maximum time that the server will take to answer any query. Once the timeout value is exceeded the network connection will be closed. This parameter may be useful for detecting dead network connections in a pooled environment.
user	An string-type parameter to specify a user name for the connection. For use with the `JDBCEndpointConnectionFilter` that may perform authentication and accept or reject incoming connections.
password	An string-type parameter to specify a password for the connection. For use with the `JDBCEndpointConnectionFilter`.

This sample code connects via `DriverManager` and specifies a new socket timeout of 60 seconds:

```
String url = "jdbc:esper:remote:myhostname:8450&socketTimeout=60";
Connection connection = DriverManager.getConnection(url);
```

The next code snippet connects via `DataSource` and also specifies a new socket timeout:

```
EPLRemoteDataSource remoteDataSource = new EPLRemoteDataSource();
remoteDataSource.setServer("myhostname");
remoteDataSource.setPortNumber(8450);
remoteDataSource.setSocketTimeout(60);
```

```
Connection connection = remoteDataSource.getConnection();
```

Chapter 5. Driver Parameters

This section documents driver parameters that are common to both the local and the remote JDBC drivers.

Table 5.1. Driver Parameters

Parameter	Description
log.debug	By default this setting is `false`. Set to `true` to enable driver debug-level logging, and set the `log.file` setting to a writable URL or file name.
log.file	If `log.debug` is set to `true`, this setting is the URL of the log file, or the log file name, to use for logging messages (remote driver only). For example, set `"&log.debug=true&log.file=/tmp/logs/epljdbc.log"` or `"&log.debug=true&log.file=c://tmp//logs/epljdbc.log"`. The log directory must exist for logging to take place, the driver appends log entries to given file. For the local driver, the logging destination is supplied by the Log4J configuration file instead.

Chapter 6. Examples

In order to compile and run the samples please follow the below instructions:

1. Make sure Java 1.5 or greater is installed and the JAVA_HOME environment variable is set.
2. Copy the Esper distribution jar file and its runtime dependencies to `jdbcserver/lib`.
3. Open a console window and change directory to examples/etc.
4. Run "setenv.bat" (Windows) or "setenv.sh" (Unix) to verify your environment settings.
5. Run "compile.bat" (Windows) or "compile.sh" (Unix) to compile the examples.
6. Now you are ready to run the examples. Further information to running each example can be found in the "examples/etc" folder in file "readme.txt".
7. Modify the logger logging level in the "log4j.xml" configuration file changing DEBUG to INFO on a class or package level to reduce the volume of text output.

6.1. SLA Reporting with the Eclipse BIRT Business Intelligence and Reporting Tools

This example generates reports and charts illustrating real-time information computed by Esper for monitoring a database search service.

The reporting framework used in this example is Eclipse BIRT, the Business Intelligence and Reporting Tool of the Eclipse project. The Esper JDBC remote driver allows BIRT to query Esper EPL metadata and statement results.

The example requires prior knowledge of Esper and EPL. Please consult the introductory examples on the website and within the Esper distribution and documentation.

To run the complete example, a separate download of the Eclipse BIRT reporting software is required, available from `http://www.eclipse.org/birt`. The BIRT Report Designer *RCP Designer* suffices. The example has been tested with BIRT RCP Report Designer version 2.2.2.

The example first introduces the use case requirements, and then outlines how to run the simulator that also starts a JDBC remote server. After the simulator is running on your system, you are ready to follow the instructions to create a report using the BIRT Report Designer.

The example previews reports in the BIRT designer studio. It does not include a BIRT runtime, server or deployment environment for continuous reporting.

In general, for scheduling reports to run at given intervals, consider defining an Esper EPL pattern that uses the `timer:at` or `timer:interval` observers. Please consult the documentation for more information on patterns.

6.1.1. Use Case Overview

The example is taken out of the domain of service level monitoring. The system under monitoring processes incoming client requests that are searches for documents in a large document database.

Each search request is placed into a logical queue for processing by one of the search services. Each search service may have multiple service instances performing searches. A service instance takes a search request from the service's logical input queue and processes the search. When done, the service instance takes the next search request from the queue or waits if there are no further requests.

For monitoring, the system has been instrumented to send events to Esper as follows:

1. Every search request submitted by a user results in one event representing the request, by name `SubmittedWorkEvent`.
2. At the time a service instance takes a search request from its queue and starts to process the search request, the service instance generates a `StartWorkEvent` event.
3. When a service instance completes processing a search request then the service instance generates a `CompletionEvent` event.
4. Every 1 second interval we generate of event for each logical queue, with the queue name and the current queue depth of the queue. This event is termed `QueueReportEvent`.

6.1.2. Simulator and JDBC Remote Network Server

The example includes a simulator for search request generation, logical service queues and service instances.

Run "run_slareportserver.bat" (Windows) or "run_slareportserver.sh" (Unix) in the `examples/etc` folder to start the SLA traffic simulator. The classes for this example live in package `com.espertech.esper.jdbc.examples.slareport`.

Upon startup, the simulator performs the following steps:

1. Acquires an Esper engine instance and creates EPL statements to analyze the event stream that the search service simulator produces.
2. Starts the search service simulator.
3. Starts the JDBC remote network server endpoint on the port configured in the property file `slareportserver_config.properties`, by default port 8450.

The search service simulator creates 3 service queues (`java.util.concurrent.BlockingQueue`). For each of the 3 service queues the simulator starts

2 service instances (`java.lang.Thread`). A request generator thread starts producing requests at the speed of 10 search requests per second and places each search request into one of the queues (random selection).

6.1.3. Creating a Report with Eclipse BIRT

This section is a step-by-step walkthrough to generating a simple report of real-time, search service monitoring data.

Prerequisite to completing the steps is that a Eclipse BIRT designer environment is installed and started. The steps outline how to create a simple listing of average processing time by service and service instance.

Table 6.1. Steps

Step	Description
1. Create a New Report	In BIRT, create a new report and choose the Simple Listing report template.
2. Create a Data Source	1. In the Data Explorer view, right-click on Data Sources and create a new Data Source. 2. Select JDBC Data Source and enter the name `EsperEPLDataSource` in the field Data Source Name, click Next. 3. Click on Manage Drivers, and Add to add the Esper JDBC remote driver jar file from the distribution, by name `esper-jdbcdriver-`*version*`.jar`. Click on ok. 4. As the Driver Class, enter `com.espertech.esper.jdbc.remote.EPLRemoteJdbcDriver` 5. As the Database URL, enter `jdbc:esper:remote:127.0.0.1:8450`, replacing within the URL the host (e.g. 127.0.0.1) and port (e.g. 8450 by default) as necessary. 6. Click on Test Connection. You should get a confirmation that the connection was successful. 7. Click on Finish to complete adding the Data Source.
3. Create a Data Set from an existing Esper EPL Statement	1. In the Data Explorer view, right-click on Data Sets and create a new Data Set. 2. Enter the Data Set Name as `EsperEPLDataSet` and make sure the Data Source `EsperEPLDataSource` is selected. Change the Data Set Type to SQL Stored Procedure Query and click on Next. 3. BIRT now presents in the Available Items section of the dialog all stored procedures (EPL statements) and tables (named windows).

Step	Description
	4. In the right hand text editor, clear the text and enter `default.CompletedPerInstance`, or double-click the stored procedure by name `CompletedPerInstance`. Click on Finish to add the Data Set. 5. The BIRT designer now presents the Edit Data Set dialog. Click on Preview Results on the left-hand side menu to view current query results. 6. Click on OK to complete adding a Data Set.
4. Bind Data Set columns to columns in the Report	BIRT presents the report layout in the Layout tab of the main workbench editor view. Columns of a Data Set can be dragged and dropped onto the report layout. 1. In the Data Explorer view, expand the `EsperEPLDataSet` Data Set. BIRT presents a list of columns of the EPL statement. 2. Click on the `serviceName` column of the `EsperEPLDataSet` Data Set and drag the column onto the report layout in the Layout tab of the main workbench window, into the first column of the row marked as Detail Row in the Layout. 3. Perform the step above for the `count(*)` column, dragging the column into the Layout tab to the same row and the next column to the right. 4. Click on the Preview tab of the main workbench editor view. BIRT fetches the current data from the continuously computing Esper engine via the JDBC remote driver, and generates the report.

The steps as outlined above walked through the design of a basic report. BIRT allows the new report design to be saved to disk as an XML document, for execution in the BIRT runtime environment (not included in example).

6.1.4. Dynamic EPL Query Example

While the previous section obtained statement results of an existing EPL statement that the simulator had created upon startup, this sections creates a dynamic (non-continuous, fire-and-forget) EPL query that is not stored within the Esper engine, and that contains filter and grouping criteria that are provided through BIRT.

The next walkthrough creates a further Data Set in BIRT that performs a dynamic query against a named window which holds the last 1 minute of search criteria objects. The section does not create a further report or bind the new Data Set columns to report table columns. It simply creates the Data Set and previews the Data Set query results.

Table 6.2. Steps

Step	Description
Create a Data Set specifying a dynamic EPL Query	1. Make sure the Layout tab is selected in the workbench editor. 2. In the Data Explorer view, right-click on Data Sets and create a new Data Set. 3. Enter the Data Set Name as `EsperEPLDataSetDyn` and make sure the Data Source `EsperEPLDataSource` is selected. Change the Data Set Type to SQL Select Query and click on Next. 4. On the right-hand side of the Query dialog for the Data Set, enter a query such as: `select * from SearchRequests where searchCriteria like '%science%'` and click on Finish to finish adding the Data Set. This sample query selects search criteria that contain the text "science" submitted in the last 60 seconds. 5. The BIRT designer now presents the Edit Data Set dialog. Click on Preview Results on the left-hand side menu to view current query results. 6. Click on OK to complete adding the Data Set.

6.1.5. Conclusion

The Eclipse BIRT report and chart designer and also the BIRT runtime interact via the JDBC API with one or more Esper engine instances.

Existing Esper EPL statements show in BIRT as stored procedures. Dynamic EPL statements can query named windows.

Metadata provided by the JDBC driver and Esper is used by the designer to allow interactive design of queries as well as reports, while the BIRT preview capability fetches current computation results from Esper.

Chapter 7. Compliance, Features and Limitations

7.1. Compliance

The local and remote JDBC drivers are JDBC 3.0 API compliant by providing all the required interface implementations.

The following exceptions apply to full JDBC compliance:

1. The driver does not support SQL92 commands for data definition, including `create table`, `drop table`, `create index` and `drop index`
2. The driver supports only a subset of SQL92 commands for data manipulation; It does not support the SQL `UPDATE` statement and the SQL `DELETE` statement. The driver does also not support `executeUpdate`.

7.2. Features

Other notable features supported by the driver are:

1. The driver supports SQL `SELECT` and `INSERT INTO` statements, and all aspects of Esper EPL with limitations for dynamic EPL queries as listed in *Section 2.3, "Tables"*.
2. The driver supports escape syntax for stored procedures to retrieve results of Esper EPL statements.
3. `ResultSet` returned by the driver are always scrollable and positionable.
4. Query execution is multithread-safe, i.e. an Esper engine instance can process events while queries are being executed.

7.3. Limitations

Features not supported by the driver or other known limitations are noted below:

1. All transactions are read-only; Methods to affect transaction boundaries such as `commit` and `rollback` do not take effect. Distributed transactions are not supported.
2. Stored procedure calls and prepated statement EPL queries do not allow parameterization via IN, OUT and INOUT parameters.
3. Result sets cannot be updated via any of the `update` methods.
4. There is no support for batching via `addBatch` and `executeBatch`.
5. There is no support for authentication or authorization by the JDBC driver and endpoint.
6. The driver package does not provide implementations for `ConnectionPoolDataSource` and `XADataSource`.

7. The drivers do not support escape syntax other then for stored procedure calls.
8. The drivers do not support the rowset API, namely not support packages `javax.sql.rowset`, `javax.sql.rowset.serial` and `javax.sql.rowset.spi`.
9. The drivers do not support advanced data types.