SchemaSpy Documentation

Release 6.0.0

SchemaSpy Contributors

Dec 20, 2017

Contents

Document your database simply and easily

Do you hate starting on a new project and having to try to figure out someone else's idea of a database? Or are you in QA and the developers expect you to understand all the relationships in their schema? If so then this tool's for you.

CHAPTER 1

What is SchemaSpy ?

SchemaSpy is a Java-based tool (requires Java 8 or higher) that analyzes the metadata of a schema in a database and generates a visual representation of it in a browser-displayable format. It lets you click through the hierarchy of database tables via child and parent table relationships as represented by both HTML links and entity-relationship diagrams. It's also designed to help resolve the obtuse errors that a database sometimes gives related to failures due to constraints. It is free software that is distributed under the terms of the MIT License.

If you like SchemaSpy then please give the start on project.

SchemaSpy uses the dot executable from Graphviz to generate graphical representations of the table/view relationships. This was initially added for people who see things visually. Now the graphical representation of relationships is a fundamental feature of the tool. Graphvis is not required to view the output generated by SchemaSpy, but the dot program should be in your PATH (not CLASSPATH) when running SchemaSpy or none of the entity relationship diagrams will be generated (or use the -gv option).

SchemaSpy uses JDBC's database metadata extraction services to gather the majority of its information, but has to make vendor-specific SQL queries to gather some information such as the SQL associated with a view and the details of check constraints. The differences between vendors have been isolated to configuration files and are extremely limited. Almost all of the vendor-specific SQL is optional.

CHAPTER 2

Sample documentation

Browse some sample documentation generated by SchemaSpy. Note that this was run against an extremely limited schema so it doesn't show the full power of the tool.

CHAPTER 3

SchemaSpy GUI

SchemaSpy is a command line tool. If you're more comfortable with the point-and-click approach then try out Joachim Uhl's SchemaSpyGUI.

SchemaSpy was mentioned in one of th O'Reilly's book

3.1 SchemaSpy

Document your database simply and easily

Do you hate starting on a new project and having to try to figure out someone else's idea of a database? Or are you in QA and the developers expect you to understand all the relationships in their schema? If so then this tool's for you.

3.1.1 What is SchemaSpy ?

SchemaSpy is a Java-based tool (requires Java 8 or higher) that analyzes the metadata of a schema in a database and generates a visual representation of it in a browser-displayable format. It lets you click through the hierarchy of database tables via child and parent table relationships as represented by both HTML links and entity-relationship diagrams. It's also designed to help resolve the obtuse errors that a database sometimes gives related to failures due to constraints. It is free software that is distributed under the terms of the MIT License.

If you like SchemaSpy then please give the start on project.

SchemaSpy uses the dot executable from Graphviz to generate graphical representations of the table/view relationships. This was initially added for people who see things visually. Now the graphical representation of relationships is a fundamental feature of the tool. Graphvis is not required to view the output generated by SchemaSpy, but the dot program should be in your PATH (not CLASSPATH) when running SchemaSpy or none of the entity relationship diagrams will be generated (or use the -gv option).

SchemaSpy uses JDBC's database metadata extraction services to gather the majority of its information, but has to make vendor-specific SQL queries to gather some information such as the SQL associated with a view and the details of check constraints. The differences between vendors have been isolated to configuration files and are extremely limited. Almost all of the vendor-specific SQL is optional.

3.1.2 Sample documentation

Browse some sample documentation generated by SchemaSpy. Note that this was run against an extremely limited schema so it doesn't show the full power of the tool.

3.1.3 SchemaSpy GUI

SchemaSpy is a command line tool. If you're more comfortable with the point-and-click approach then try out Joachim Uhl's SchemaSpyGUI.

SchemaSpy was mentioned in one of th O'Reilly's book

3.2 Installation

3.2.1 Requirements

Before you can start using SchemaSpy you must have installed two things in your system environment.

3.2.1.1 Java 8

Download instructions for all operating systems: https://java.com/en/download/manual.jsp

3.2.1.2 Graphviz

- **Windows** The easiest way to install Graphviz is to download the msi package from http://www.graphviz.org/Download_windows.php

  Warning: Remember to add the folder containing Graphviz's dot.exe application to your system PATH variable, eg.

  ```
  C:\Program Files (x86)\Graphviz2.38\bin
  ```

- **Linux, Mac OS** Please read carefully the detailed instructions on how to install Graphviz on your os version http://www.graphviz.org/Download.php.

3.2.2 SchemaSpy

SchemaSpy is just a single executable jar-file (schemaspy-[version].jar), you can download releases from http://schemaspy.org or the github releases page https://github.com/schemaspy/schemaspy/releases

If you feel adventurous there is a link in the README.md for latest builds.

When you have your jar-file head on over to Get Started

3.3 Get Started

Welcome to SchemaSpy. We will do the best to simplify documentation process of your database.

3.3.1 Configuration

Parameters can be specified in the comand line (described below) or you can predefine configuration in the file. SchemaSpy will search configuration file in `<current-dir>/schemaspy.properties` To use an alternative configuration file run SchemaSpy with parameter: `java -jar schemaspy.jar -configFile path/to/config.file`

Config file example:

```
# type of database. Run with -dbhelp for details
schemaspy.t=mssql
# optional path to alternative jdbc drivers.
schemaspy.dp=path/to/drivers
# database properties: host, port number, name user, password
schemaspy.host=server
schemaspy.port=1433
schemaspy.db=db_name
schemaspy.u=database_user
schemaspy.p=database_password
# output dir to save generated files
schemaspy.o=path/to/output
# db scheme for which generate diagrams
schemaspy.s=dbo
```

3.3.2 Running SchemaSpy

You can easily run SchemaSpy from the command line:

```
java -jar schemaspy.jar -t dbType -dp C:/sqljdbc4-3.0.jar -db dbName -host server -
↪port 1433 [-s schema] -u user [-p password] -o outputDir
```

3.3.2.1 Parameters priority:

It is important to notice, that command-line parameters **override** those configured in schemaspy.properties file.

3.3.2.2 Commonly used parameters:

[-t databaseType] Type of database (e.g. ora, db2, etc.). Use -dbhelp for a list of built-in types. Defaults to ora.

[-db dbName] Name of database to connect to

[-u user] Valid database user id with read access. A user id is required unless -sso is specified.

[-s schema] Database schema. This is optional if it's the same as user or isn't supported by your database. Use -noschema if your database thinks it supports schemas but doesn't (e.g. older versions of Informix).

[-p password] Password associated with that user. Defaults to no password.

[-o outputDirectory] Directory to write the generated HTML/graphs to

[-dp pathToDrivers] Looks for drivers here before looking in driverPath in [databaseType].properties. The drivers are usually contained in .jar or .zip files and are typically provided by your database vendor.

[-hq] or [-lq] Generate higher or lower-quality diagrams. Various installations of Graphviz (depending on OS and/or version) will default to generat /ing either higher or lower quality images. That is, some might not have the "lower quality" libraries and others might not have the "higher quality" libraries. Higher quality output takes

longer to generate and results in significantly larger image files (which take longer to download / display), but the resultant Entity Relationship diagrams generally look better.

3.3.3 Advanced Usage

3.3.3.1 Supply Connection-properties

As an example running mysql with a new driver you'll get warning According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.

This can be omited by addind connection property `useSSL=false`

To add this connection property add following to commandline: `-connprops useSSL\\=false`

`-connprops` can also take a properties file as argument but when escaping the = with double it will use ut as "useSSL=false"

3.3.3.2 Create your own DB type

As an example we will add the connection property from above to the mysql db-type

1. Create a new file in same directory as the schemaspy-jar, let's call it mysql-nossl.properties
2. Add the following content to mysql-nossl.properies:

```
extends=mysql
connectionSpec=jdbc:mysql://<hostOptionalPort>/<db>?useSSL=false
```

3. Now you can run schamaspy with -t mysql-nossl

If you want to have a closer look at the db-types you can find them at github

3.3.3.3 Create you own DB type super advanced

Now we are going to connect to mysql thru unix socket, put on your helmets

1. Download a unix socket library for java and all of it's dependencies, for simplicity put them in a sub-folder called `drivers` in the same folder as the schemaspy-jar:

```
junixsocket-common-2.0.4.jar
junixsocket-mysql-2.0.4.jar
junixsocket-native-2.0.4-x86_64-MacOSX-gpp-jni.nar <- Im on OSX
junixsocket-native-2.0.4.nar
mysql-connector-java-5.1.32.jar
native-lib-loader-2.1.5.jar
slf4j-api-1.7.25.jar
slf4j-simple-1.7.25.jar
```

2. Create your own db-type let's call it my-mysql-socket.properties in same folder as the schemaspy-jar with following content:

```
connectionSpec=jdbc:mysql://<host>/<db>?socketFactory=<socketFactory>&socket=
↪<socket>
socketFactory=ClassName of socket factory which must be in your classpath
socket=Path To Socket
```

3. Now run schemaspy with the following options:

```
java -jar [schemaspy.jar] -t my-mysq-socket \
-dp lib/mysql-connector-java-[version].jar \
-loadjars \
-db [DBName] \
-host localhost \
-port 3306 \
-u [User] \
-socketFactory org.newsclub.net.mysql.AFUNIXDatabaseSocketFactory \
-socket [pathToSocket] \
-o [outputDir]
```

Replace values accordingly.

Yes, you need to specify `-db`, `-host`, `-port`

Yes, the socketFactory could have been written directly into the properties-file, this is just an example, mysql-socket exists as a db-type exactly like this.

And since you might want to use another unix socket library this doesn't close any doors.

3.4 Configuration

3.4.1 DatabaseType

You can create you're own databaseType so lets go through how it works.

3.4.1.1 Selection

On the commandline you specify the databaseType using the option `-t`. The option can be specified with either [name].properties or just [name]

Example: `-t mysql`

or `-t mysql.properties`

The search order is:

1. user.dir/ as supplied
2. user.dir/ as supplied, suffixed with .properties
3. Classpath as supplied
4. Classpath as supplied, suffixed with .properties
5. Classpath in schemaspy supplied location as supplied
6. Classpath in schemaspy supplied location as supplied, suffixed with .properties

This actually means that if you supply `-t my_conf/mydbtype`

It will look for:

1. file: $user.dir/my_conf/mydbtype
2. file: $user.dir/my_conf/mydbtype.properties
3. Classpath: my_conf/mydbtype
4. Classpath: my_conf/mydbtype.properties
5. Classpath: org/schemaspy/types/my_conf/mydbtype
6. Classpath: org/schemaspy/types/my_conf/mydbtype.properties

3.4.1.2 Layout

It can contain wast amount of properties so we will break it down. The Properties-file can contain instructions.

extends `extends` which does what i means, it allows one to override or add properties to an existing databaseType (by specifying a parent/base)

As an example:

```
extends=mysql
```

which you can see in mysql-socket.properties

include `include.[n]` is a bit different it allows one to add a single property from another databaseType. `[n]` is substituted for a number. The value has the form of `[databaseType]::[key]`.

As an example:

```
include.1=mysql::schemaSpec
```

This would have been valid in the mariadb.properties

Then we have required properties:

description= Description for the databaseType (mostly used in logging)

connectionSpec= We will talk more about this one. It's the connectionUrl used, but it supports token replacement

driver= FQDN of the JDBC driver as an example `org.h2.Driver`

3.4.1.3 ConnectionSpec

Let's dive a bit deeper into the connectionSpec.

As an example from mysql-socket:

```
extends=mysql
connectionSpec=jdbc:mysql://<host>/<db>?socketFactory=<socketFactory>&socket=<socket>
socketFactory=ClassName of socket factory which must be in your classpath
socket=Path To Socket
```

We mentioned extends earlier. ConnectionSpec contains the connectionUrl used with the jdbc driver, some might refer to it as the connectionString.

connectionSpec allow token replacement, a token is `<[tokenName]>`. In the above example we have host, db, socketFactory, socket.

This means that when used it expects the following commandline arguments:

```
-h [host] (for host)
-db [dbname] (for db)
-socketFactory [socketFactory class]
-socket [path to socket]
```

host and db are already known, but `-socketFactory` and `-socket` has become a new commandline argument. The presence of the keys in the databaseType properties file is only for description, it's printed when `-dbhelp` is used as a commandline argument. (db and host located in databaseType mysql which is extended)

There is also a synthetic token that can be replaced <hostOptionalPort> which combines host and port if port is supplied. Default separator is `:` but can be changed by specifying another under the key `hostPortSeparator`

3.4.1.4 Other Properties

driverPath= path to classpath resources that will be used when trying to create the jdbc Driver in java same as commandline argument `-dp`

dbThreads= number of threads that can be used to analyze the database

schemaSpec= regular expression used in conjunction with `-all` (and can be command line param `-schemaSpec`)

When metadata in JDBC isn't cutting the mustard. You can replace it with a sql query. They are prepared and supports named parameters as long as they are available. Data is retrieved by column label. So additional columns are ok, but you might need to alias columns so that they are returned correctly to schemaspy.

:dbname DatabaseName `-db`

:schema Schema `-s`

:owner alias for :schema

:table table that the query relates to (think selectRowCountSql)

:view alias for :table

:catalog Catalog `-cat`

Possible Metadata overrides and expected columns in result:

selectSchemasSql= schema_comment

selectCatalogsSql= catalog_comment

selectTablesSql= table_name, table_catalog, table_schema, table_comment, table_rows

selectViewsSql= view_name, view_catalog, view_schema, view_comment, view_definition

selectRowCountSql= row_count

selectColumnTypesSql= table_name, column_name, column_type, short_column_type

selectRoutinesSql= routine_name, routine_type, dtd_identifier, routine_body, routine_definition,sql_data_access, security_type, is_deterministic, routine_comment

selectRoutineParametersSql= specific_name, parameter_name, dtd_identifier, parameter_mode

selectViewSql= view_definition, text

selectCheckConstraintsSql= table_name, constraint_name

selectTableIdsSql= table_name, table_id

selectIndexIdsSql= table_name, index_name, index_id

selectTableCommentsSql= table_name, comments

selectColumnCommentsSql= table_name, column_name, comments

Extend the types of views that exist

viewTypes= default is VIEW

3.4.2 SchemaMeta

Is a way to modify the processing in and output from SchemaSpy.

- *Add comments/remarks*
- *Add relationships*
- *Add remote tables*
- *Add columns*
- *Exclude columns from implied relationships*
- *Exclude columns from diagrams*

All these instructions are defined in xml the schema can be found

Schema contains documentation but lets go through the above mentioned features.

3.4.2.1 Add comments/remarks

The xsd currently allows both comments and remarks. However remarks has been deprecated.

So adding a comment will either add, if missing from database, or replace if comments/remarks exist.

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <comments>Database comment</comments>
    <tables>
        <table name="ACCOUNT" comments="Table comment">
            <column name="accountId" comments="Column comment"/>
        </table>
    </tables>
</schemaMeta>
```

3.4.2.2 Add relationships

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <tables>
        <table name="AGENT">
            <column name="acId" type="INT">
                <foreignKey table="ACCOUNT" column="accountId" />
            </column>
            <column name="coId" type="INT">
                <foreignKey table="COMPANY" column="companyId" />
            </column>
        </table>
    </tables>
</schemaMeta>
```

3.4.2.3 Add remote tables

Specifying the remoteCatalog and remoteSchema attributes on a table makes it a remote table and as such a logical table.

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <tables>
        <table name="CONTRACT" remoteCatalog="other" remoteSchema="other">
            <column name="contractId" autoUpdated="true" primaryKey="true" type="INT"/
↪>
            <column name="accountId" type="INT">
                <foreignKey table="ACCOUNT" column="accountId"/>
            </column>
            <column name="agentId" type="INT">
                <foreignKey table="AGENT" column="aId"/>
            </column>
        </table>
    </tables>
</schemaMeta>
```

3.4.2.4 Add columns

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <tables>
        <table name="ACCOUNT">
            <column name="this_is_new" type="INT" />
        </table>
    </tables>
</schemaMeta>
```

3.4.2.5 Exclude columns from implied relationships

Explicitly disables relationships to or from this column that may be implied by the column's name, type and size.

Available options: to, from, all, none Default: none

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <tables>
        <table name="AGENT">
            <column name="accountId" type="INT" disableImpliedKeys="all"/>
        </table>
    </tables>
</schemaMeta>
```

3.4.2.6 Exclude columns from diagrams

Sometimes the associations displayed on a relationships diagram cause the diagram to become much more cluttered than it needs to be. Enable this setting to not show the relationships between this column and other columns.

Use exceptDirect to disable associations on all diagrams except for the diagrams of tables directly (within one degree of separation) connected to this column.

Available options: all, exceptDirect, none Defaults: none

```
<schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
    <tables>
        <table name="COUNTRY">
            <column name="countryId" type="INT" disableDiagramAssociations="all"/>
        </table>
    </tables>
</schemaMeta>
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search