
SchemaSpy Documentation

Release 6.0.0

SchemaSpy Contributors

Sep 22, 2019

1	Overview	3
2	What's new	5
2.1	6.0.0	5
2.2	6.1.0	6
3	Features	7
4	Sample documentation	9
5	SchemaSpy GUI	11
6	Installation	13
6.1	Requirements	13
6.2	Optional	13
6.3	SchemaSpy	14
7	Get Started	15
7.1	Configuration	15
7.2	Running SchemaSpy	15
8	Advanced Usage	17
8.1	Supply Connection-properties	17
8.2	Create your own DB type	17
8.3	Supply or override database type	18
8.4	Create you own DB type super advanced	18
8.5	Add markdown comments using additional metadata	19
9	Single Sign-On	21
9.1	mssql	21
9.2	mssql-jtds	21
10	Command-Line Arguments	23
10.1	General	23
10.2	Database related	23
10.3	Html report related	24
11	DatabaseType	27

11.1	Selection	27
11.2	Layout	27
11.3	ConnectionSpec	28
11.4	Other Properties	29
11.5	Sql query instead of DatabaseMetaData	29
12	SchemaMeta	31
12.1	Add comments/remarks	31
12.2	Add relationships	32
12.3	Add remote tables	32
12.4	Add columns	32
12.5	Exclude columns from implied relationships	33
12.6	Exclude columns from diagrams	33
13	Frequent Asked Questions	35
13.1	General	35
13.2	OSX	35

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

Overview

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.

SchemaSpy comes with ABSOLUTELY NO WARRANTY. SchemaSpy is free software licensed and distributed under LGPL version 3 or later. SchemaSpy can be redistributed under the conditions of LGPL version 3 or later. <http://www.gnu.org/licenses/>

If you like SchemaSpy, don't forget to give us a star on .

SchemaSpy produces dot-file and uses either the dot executable from [Graphviz](#) or embedded [viz.js](#) 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. Graphviz is not required to view the output generated by SchemaSpy, but the dot program should be in your PATH (not CLASSPATH) or use the `-gv` or `-vizjs` arguments, else none of the entity relationship diagrams will be rendered.

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.

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

2.1 6.0.0

- **Html report**
 - Now uses mustache
 - DataTables for data
 - Markdown rendering of comments
- **DatabaseTypes:**
 - sqlite-xerial
 - redshift
 - orathin-service
 - netezza
 - mysql-socket
 - mssql08
 - mssql08-jtds
 - mssql08-jtds-instance
 - impala
 - hive
 - hive-kerberos-driverwrapper
 - hive-kerberos-driverwrapper-zookeeper

2.2 6.1.0

- **Diagrams**
 - Now has option to use embedded viz.js (no need for Graphviz) `-vizjs`
 - Limit the degree of separation in table diagrams. Allowed values are 1 or 2.
- **XML**
 - Now includes routines
- **Html report**
 - Column page loads faster
 - Table page contains check constraints

CHAPTER 3

Features

- Supports most JDBC compliant dbms (support missing? you can add your own)
- Generates for foreign keys
- Generates for implied relationships (name, type) of a column matches a primary key
- Generates for relationships based on rails naming conventions
- Shows column relationship and actions
- Shows routines (Functions/Stored procedures)
- Shows views and definitions
- Will render present in comments
- Allows for supplying additional metadata, see *SchemaMeta*
- Present a set of found anomalies

CHAPTER 4

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 5

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](#).

6.1 Requirements

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

6.1.1 Java 8

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

6.2 Optional

Since version 6.1.0 Graphviz is no longer required. There is an embedded viz.js that can be used by adding command line argument `viz.js`

6.2.1 Graphviz

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

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/>.

6.3 SchemaSpy

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

7.1 Configuration

Parameters can be specified in the command 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
```

7.2 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
```

7.2.1 Parameters priority

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

7.2.2 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. Supports a directory as argument, which will add directory and all content to classpath, will recurse. Supports multiple paths separated by OS dependent path separator

[-hq] or [-lq] Generate higher or lower-quality diagrams. Various installations of Graphviz (depending on OS and/or version) will default to generate `/img` 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.

[-imageformat outputImageFormat] The format of the image that gets generated. Supported formats are `svg` and `png`. Defaults to `png`. E.g. `-imageformat svg`

For a comprehensive listing see [Command-Line Arguments](#)

8.1 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 omitted by adding 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 `\` it will use it as "useSSL=false". If key or value contains `/` it needs to be escaped with a single `\`. Multiple pairs can be separated by `;`. If using linux shell or git bash in windows the `\` also needs to be escaped with a `\` so for example it would look like `useSSL\\=false`.

8.2 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.properties`:

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

3. Now you can run `schemaspy` with `-t mysql-nossl`

If you want to have a closer look at the db-types you can find them at [github](#)

8.3 Supply or override database type

1. Create a new file in same directory as the schemaspj-jar, let's call it myDbType.properties
2. Start by extending the database type you want to supply or override sql statements for as an example we will extends postgresSQL:

```
extends=pgsql
```

3. Queries you can supply or override can be found at *Sql query instead of DatabaseMetaData* we will override routines:

```
extends=pgsql
selectRoutinesSql=select r.routine_name, case p.prokind when 'f' then 'FUNCTION'
↳when 'p' then 'PROCEDURE' when 'a' then 'AGGREGATE' when 'w' then 'WINDOW' else
↳'UNKNOWN' end as routine_type, case when p.prorotset then 'SETOF ' else '' end
↳|| case when r.data_type = 'USER-DEFINED' then r.type_udt_name else r.data_type
↳end as dtd_identifier, r.external_language as routine_body, r.routine_
↳definition, r.sql_data_access, r.security_type, r.is_deterministic, d.
↳description as routine_comment from information_schema.routines r left join pg_
↳namespace ns on r.routine_schema = ns.nspname left join pg_proc p on ns.oid = p.
↳pronamespace and r.routine_name = p.proname left join pg_description d on d.
↳objoid = p.oid where r.routine_schema = :schema
```

4. We also want to add materialized views to view as document at *Other Properties*:

```
extends=pgsql
selectRoutinesSql=select r.routine_name, case p.prokind when 'f' then 'FUNCTION'
↳when 'p' then 'PROCEDURE' when 'a' then 'AGGREGATE' when 'w' then 'WINDOW' else
↳'UNKNOWN' end as routine_type, case when p.prorotset then 'SETOF ' else '' end
↳|| case when r.data_type = 'USER-DEFINED' then r.type_udt_name else r.data_type
↳end as dtd_identifier, r.external_language as routine_body, r.routine_
↳definition, r.sql_data_access, r.security_type, r.is_deterministic, d.
↳description as routine_comment from information_schema.routines r left join pg_
↳namespace ns on r.routine_schema = ns.nspname left join pg_proc p on ns.oid = p.
↳pronamespace and r.routine_name = p.proname left join pg_description d on d.
↳objoid = p.oid where r.routine_schema = :schema
viewTypes=VIEW,MATERIALIZED VIEW
```

5. Now run schemaspj with your own database type -t myDbType

8.4 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 schemaspj-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-mysql-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.

8.5 Add markdown comments using additional metadata

Schemaspy supports markdown in comments Not all dbms supports comments or long enough comments or comments might just be missing.

Schemaspy also supports supplying additional metadata *SchemaMeta* More precise the ability to add/replace comments. *Add comments/remarks*

```
1 <schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
↳xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
2   <comments>Database comment</comments>
3   <tables>
4     <table name="ACCOUNT" comments="I've added comment that links using markdown
↳to markdown documentation [markdown] (https://daringfireball.net/projects/markdown/)
↳" >
5       <column name="accountId" comments="And now the schemaspy avatar !
↳[avatar] (https://avatars3.githubusercontent.com/u/20635098?s=20&v=4 "SchemaSpy")" />
6     </table>
7   </tables>
8 </schemaMeta>
```

Now just run with `-meta [path-to-above-xml]`

9.1 mssql

When running SchemaSpy **before** `-jar SchemaSpy-[version].jar add -Djava.library.path=[path-to-dir-containing-sqljdbc_auth.dll]` **after** `-jar SchemaSpy-[version].jar add -sso` When using cmd `add -connprops integratedSecurity\=true` When using git bash in windows `add -connprops integratedSecurity\\=true`

9.2 mssql-jtds

When running SchemaSpy **before** `-jar SchemaSpy-[version].jar add -Djava.library.path=[path-to-dir-containing-ntlmauth.dll]` **after** `-jar SchemaSpy-[version].jar add -sso`

Command-Line Arguments

Most of the command-line arguments can be specified in a properties file either with the default name `schemaspy.properties` or in a file specified using `-configFile` the command-line arguments should be prefixed with `schemaspy.` As an example `-sso` would be `schemaspy.sso` and `-u username` would be `schemaspy.u=username`.

10.1 General

[-h] Print help message

[-dbhelp] Print databaseType required arguments

[-configFile filePath] Path to configFile to be used, default is to look for `schemaspy.properties`

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

10.2 Database related

10.2.1 Connecting

[-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.

[-host hostName] Hostname/ip to connect to, if required by databaseType.

[-port portNumber] Port that dbms listens to, if required by databaseType.

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

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

[-sso] Single sign-on, used when `-u` and `-p` should be ignored. See also *Single Sign-On*

[-pfp] Prompt for password, if you don't want to have password in command history.

[-connprops filePathOrKeyValue] Either a properties-file with additional properties or a key/value list, pairs separated by `;` and key and value separated by `\=` example `-connprops key1\=value1;key2\=value2` see also *Supply Connection-properties*. In linux shells or git bash on windows the `\` also needs to be escaped so it should look like `\\=`. ConnectionProperties will always be populated with `-u` and `-p` if they exist.

[-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. Multiple jars can be specified using os-specific path separator.

[-loadjars] Load siblings to jar specified in `-dp`, only works for single jar entry in `-dp`

10.2.2 Processing

[-cat catalog] Filter using a specific catalog this is usually the root of the database and contains schemas.

[-s schema] Database schema. This is optional if it's the same as user or isn't supported by your database.

[-schemas listOfSchemas] List of schemas to analyze, separated by space or `,` or `'` or `"`

[-all] Try to analyze all schemas in database, schemas can be excluded with `-schemSpec` which as defaults set by `databaseType`

[-schemaspec schemaRegEx] Schemas to analyze, default to all, might be specified by `databaseType`.

[-dbthreads number] Specify how many threads/connections should be used when reading data from database, defaults to 15 or as specified by `databaseType`

[-norows] Skip fetching number of rows in tables.

[-noviews] Skip processing of views.

[-i includeTableRegexp] Include table(s) in analysis, defaults to match everything

[-I excludeTableRegex] Exclude table(s) from analysis, defaults to exclude tables containing `$`, can be overridden with `-I ""`

10.2.3 Additional data

[-meta fileOrPath] Single schema analysis file path to `SchemaMeta.xml`, when running `-all` or `-schemas` path to directory containing `SchemaMeta.xml`s with pattern `(DatabaseName|Schema).meta.xml`

10.3 Html report related

[-nohtml] Skip generation of html report.

[-noimplied] Don't look for implied relationships.

[-nopages] Just list data as one long list instead of pages.

[-rails] Use Rails-based naming convention to find relationships between logical foreign keys and primary keys.

[-template path] Path to custom mustache template/css directory, needs to contain full set of templates. Bundled templates can be found in jar `'/layout'` and can be extracted with jar tool or any zip capable tool.

[-maxdet number] Limit for when tables should be shown with details.

[-css fileName] Use a custom stylesheet. Bundled stylesheet can be extracted from jar(using zip capable tool), path `'/layout/schemaSpy.css'`

[-desc description] Add a description to the index page.

10.3.1 Diagram related

[-gv directoryPath] Path to directory containing graphviz executable(dot).

[-renderer :rendererName] Specify which renderer to use should be prefixed with ':' example `-renderer :cairo`

[-hq] or [-lq] Generate higher or lower-quality diagrams. Various installations of Graphviz (depending on OS and/or version) will default to generating 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.

[-imageformat outputImageFormat] The format of the image that gets generated. Supported formats are svg and png. Defaults to png. E.g. `-imageformat svg`

[-maxdet number] Limit for when tables shouldn't be detailed. Evaluated against total number of tables in schema. Defaults to 300.

[-font fontName] Change font used in diagrams, defaults to 'Helvetica'

[-fontsize number] Change font size in large diagrams, defaults to 11

[-rankdirbug] Switch diagram direction from 'top to bottom' to 'right to left'

[-X excludeColumnRegex] Exclude column(s), regular expression to exclude column(s) from diagrams, defaults to nothing.

[-x excludeIndirectColumnsRegex] Exclude column(s) from diagrams where column(s) aren't directly referenced by focal table, defaults to nothing.

[-vizjs] Use embedded viz.js instead of Graphviz. Useful when graphviz isn't installed. Memory is set to 64 MB, if you receive the error "Cannot enlarge memory arrays" please report this to us.

[-degree 1 or 2] Limit the degree of separation (1 shows less, 2 is default), 1 is a good option for large databases with lots of relationships.

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

11.1 Selection

On the commandline you specify the databaseType using the option `-t`. The option can be specified with either `[name].properties` or just `[name]` the `.properties` will be added if missing. So if you create one, be sure to have `.properties` extension.

Example: `-t mysql`

or `-t mysql.properties`

The search order is:

1. `user.dir/`
2. Classpath
3. Classpath in schemaspy supplied location

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

It will look for:

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

11.2 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 it 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`

11.3 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`

11.4 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`)

tableTypes= Which types should be considered tables, default is TABLE

viewTypes= Which types should be considered views, default is VIEW

11.5 Sql query instead of DatabaseMetaData

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 overrides:

selectSchemasSql= *Fetch comments for a schema, expected columns:* **schema_comment**

selectCatalogsSql= *Fetch comments for a catalog, expected columns:* **catalog_comment**

selectTablesSql= *Fetch tables, expected columns:* **table_name, table_catalog, table_schema, table_comment, table_rows**

selectViewsSql= *Fetch views, expected columns:* **view_name, view_catalog, view_schema, view_comment, view_definition**

selectIndexesSql= *Fetch indexes, expected columns:* **INDEX_NAME, TYPE, NON_UNIQUE, COLUMN_NAME, ASC_OR_DESC**

selectPrimaryKeysSql= *Fetch table PKs, expected columns:* **TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, KEY_SEQ, PK_NAME**

selectRowCountSql= *Fetch row count for a table, expected columns:* **row_count**

selectColumnTypesSql= *Fetch column type for all columns, expected columns:* **table_name, column_name, column_type, short_column_type**

selectRoutinesSql= *Fetch routines, expected columns:* **routine_name, routine_type, dtd_identifier, routine_body, routine_definition, sql_data_access, security_type, is_deterministic, routine_comment**

selectRoutineParametersSql= *Fetch parameters for routines, expected columns:* **specific_name, parameter_name, dtd_identifier, parameter_mode**

selectViewSql= *Fetch definition for a view, expected columns:* **view_definition, text** (text has been deprecated)

selectCheckConstraintsSql= *Fetch check constraints for all tables, expected columns: **table_name, constraint_name***

selectTableIdsSql= *Fetch ids for all tables, expected columns: **table_name, table_id***

selectIndexIdsSql= *Fetch ids for all indexes, expected columns: **table_name, index_name, index_id***

selectTableCommentsSql= *Fetch comments for all tables, expected columns: **table_name, comments***

selectColumnCommentsSql= *Fetch comments for all columns, expected columns: **table_name, column_name, comments***

Is a way to modify input that will affect 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.

12.1 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. Supports markdown, example see *Add markdown comments using additional metadata*

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

12.2 Add relationships

```

1 <schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
2   <tables>
3     <table name="AGENT">
4       <column name="acId" type="INT">
5         <foreignKey table="ACCOUNT" column="accountId" />
6       </column>
7       <column name="coId" type="INT">
8         <foreignKey table="COMPANY" column="companyId" />
9       </column>
10    </table>
11  </tables>
12 </schemaMeta>

```

12.3 Add remote tables

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

```

1 <schemaMeta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪xsi:noNamespaceSchemaLocation="http://schemaspy.org/xsd/6/schemameta.xsd" >
2   <tables>
3     <table name="CONTRACT" remoteCatalog="other" remoteSchema="other">
4       <column name="contractId" autoUpdated="true" primaryKey="true" type="INT"/
  ↪>
5       <column name="accountId" type="INT">
6         <foreignKey table="ACCOUNT" column="accountId"/>
7       </column>
8       <column name="agentId" type="INT">
9         <foreignKey table="AGENT" column="aId"/>
10      </column>
11    </table>
12  </tables>
13 </schemaMeta>

```

12.4 Add columns

```

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

```

12.5 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

```

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

```

12.6 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

```

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

```

Frequent Asked Questions

13.1 General

13.1.1 Schema or Catalog name can't be null

This means that Schema or Catalog information could not be extracted from connection. In this case you need to add options `-s [schemaName]` or `-cat [catalogName]` In most cases for catalog you can use `-cat %` In mysql you can use same as `-db`

13.1.2 “Cannot enlarge memory arrays” when using viz.js

According to viz.js documentation the memory is default 16MB this should be enough. We have increased this to 64 MB if you receive this error, please report this to us.

13.1.3 I just receive a cryptic error like “ERROR - null”

The code has previously avoided to log stacktraces, we now log them but only when `-debug` is used. So any cryptic error can be enhanced with stacktrace by running SchemaSpy with the argument `-debug`

13.2 OSX

13.2.1 Graphviz

There have been lots of issue with graphviz and OSX So install using brew `brew install graphviz --with-librsvg --with-pango` Depending on OSX version *Older than High Sierra*, add `-renderer :quartz` to the commandline *High Sierra or newer*, add `-renderer :cairo` to the commandline