Database connection requirements for our RDBMSs

**Last Updated:** October 21, 2024

Version at last update: GSRS 3.1.1

This document describes a set of parameters that worked in tests carried out by the GSRS team. Additional combinations of parameters may also work.

GSRS has been tested with 4 database systems: Oracle, PostGreSQL, MariaDB and MySQL. H2 also works in a development environment. GSRS may work with additional database systems, but the team cannot make any recommendations for additional database systems. In each of these cases described, GSRS 3.x configurations described here are intended to be compatible with equivalent databases which may have been created for a GSRS 2.x instance.

Below, we present some strategies for each of the 4 databases. These described strategies have 4 parts:

1. The dependencies necessary for the JDBC driver classes, in the form of lines to insert into your POM.xml file. **Note:** The default pom.xml files found in the gsrs3-main-deployment git repo contain all of these drivers by default.
2. Information for the service-specific configuration file, generally, application.conf. The Hibernate ‘dialect’ is one key piece of information.
3. Additional information configuration settings.
4. Example configuration

**A Few Usage Notes:**

1. The supplied examples are designed around connecting to **existing** databases created for GSRS 2.x and 3.x software. As such the property “spring.jpa.hibernate.ddl-auto” is set to “none”, meaning that new tables will not be generated. If the intention is to generate the schema, this setting can be adjusted. **BE CAREFUL! THIS CAN RESULT IN DATA LOSS! PLEASE BACKUP DATABASES BEFORE ATTEMPTING!**
2. All syntax examples shown below start with “spring” for the configuration settings. This is the default path for the default database. However, for *some* entity services, they are intended to connect to 2 or more datasources. In such cases, by convention, the “spring” prefix is used for the primary core source and another prefix is used for the specific entity datasource. For example the products service will expect settings for both “**spring.datasource.driverClassName**” as well as “**products.datasource.driverClassName**” (and every other datasource/jpa property expected).
3. When creating a new database, indexes listed in <https://github.com/ncats/gsrs-play/tree/GSRS_DEV/conf/sql/post> are generated automatically to improve performance. **NEW in 3.0.2:** If your database was already created, and does not have these indexes, you can set the property “spring.jpa.hibernate.ddl-auto” to “update”, then restart the substances service to add these indexes.
4. To start from zero and create a **completely new** database from scratch, create the new empty database with no tables in your DBMS with a create statement. Make sure to use UTF8 mb4 as the default for character set and collation. Then, in your microservice application.conf file, set, for example, spring.jpa.hibernate.ddl-auto=**create** and spring.jpa.generate-ddl=**true.** Run your microservice for the **first time** with these settings. When your database has been created, go back and set spring.jpa.hibernate.ddl-auto=none|update depending on your needs; and spring.jpa.generate-ddl=false. **If you don’t reset these values, your database will be overwritten next time you run your microservice.**
5. Extra steps may be necessary to configure the RDBMS to accept and properly process UTF8 encoded characters. The steps necessary to do this are beyond the scope of this document, however, for MariaDB and MySql the following SQL commands have been found to be beneficial.

|  |
| --- |
| # When creating the database, set the database character set and collation defaults. CREATE DATABASE db\_name CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;  # Set the character set on these important fields. **SHOULD WE ALSO SPECIFY COLLATION?**  ALTER TABLE ix\_ginas\_name MODIFY name VARCHAR(255) CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_name MODIFY full\_name longtext CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_code MODIFY code VARCHAR(255) CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_amount MODIFY non\_numeric\_value VARCHAR(255) CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_amount MODIFY units VARCHAR(255) CHARACTER SET utf8mb4; ALTER TABLE ix\_ginas\_vocabulary\_term MODIFY value VARCHAR(1000) CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_vocabulary\_term MODIFY display VARCHAR(1000) CHARACTER SET utf8mb4; ALTER TABLE ix\_ginas\_vocabulary\_term MODIFY description VARCHAR(1000) CHARACTER SET utf8mb4; ALTER TABLE ix\_ginas\_reference MODIFY citation longtext CHARACTER SET utf8mb4;  ALTER TABLE ix\_ginas\_note MODIFY note longtext CHARACTER SET utf8mb4;   # Helpful commands for troubleshooting character set issues:  USE db\_name;  SHOW VARIABLES LIKE '%character\_set%';  USE db\_name;  SELECT @@character\_set\_database, @@collation\_database;  SELECT DEFAULT\_CHARACTER\_SET\_NAME, DEFAULT\_COLLATION\_NAME FROM INFORMATION\_SCHEMA.SCHEMATA WHERE SCHEMA\_NAME = 'db\_name';  # The way you connect with a local command line client to the gsrs database may also be important. On the server where you run the Mysql **client**, put this in the Mysql program’s config to make the client use utf8 as a default.  default-character-set=utf8  # Adding UTF-8 to the JDBC connection string may also help. It helped in GSRS 2.x. This is the syntax for 2.x db.default.url="jdbc:mysql://xyz.com:3306/ginas\_tmp?characterEncoding=UTF-8" |

1. There are some common configuration settings for the data source that can be used to fine-tune your application. You can put these settings together with your data source settings like url, username, and password. Update the numbers to fit your application.

|  |
| --- |
| spring.datasource.connectionTimeout=120000 # maximum number of milliseconds that a client will wait for a connection  spring.datasource.maximumPoolSize= 50 |

1. **NOTE from GSRS 3.1**: Database schema changes are introduced in GSRS3.1. Please check the README file and scripts in the ***./substances/database/sql/${your database flavor}/GSRS\_3.1*** folder.
2. Caseand search performance. In database creation, if you want the query to be case-insensitive **GLOBALLY**, you can set the collation to be case-insensitive. Such as in MariaDB, you can create the database with the collation to be utf8mb4\_unicode\_ci. The ci means case insensitive. On the other hand, if you do not want to set it globally, to have a better search performance, you might need to create an index on the query column with UPPER() or LOWER() if there is not one created by JPA. The collation change directly affects the Hibernate issues case insensitive searches. However, in many cases Java is written in a way that pays attention to case. In one important case, you’ll have to set a configuration value so that GSRS code will issue the right type of query. For example, in substances-core.conf of the substances module, see the validatorClass, "ix.ginas.utils.validation.validators.NamesValidator", and the environment variable ${SUBSTANCE\_NAMES\_VALIDATOR\_CASE\_SEARCH\_TYPE}. If you use a case insensitive collation and the value “Implicit” here, this will be speed things up in Mariadb because GSRS then uses a case insensitive search. In other database flavors, that accommodate “functional indexes”, we can use the case-sensitive (explicit) query.
3. GSRS 3.x version up to and including 3.1.1 can either use “sequences” or “auto\_increment” for generating numeric id fields. In the past we noted a spring.jpa.hibernate.use-new-id-generator-mappings=false configuration could facilitate compatibility with GSRS 2.x. Now, starting in 3.1.1, we encourage the GSRS default of true for this setting. This is because Spring-Boot 3.x will no longer have a such a setting. Therefore, sequences will be required in GSRS 3.1.2 due to the planned upgrade of Spring Boot in the next version. While the auto\_increment strategy still works in 3.1.1, its days are numbered.

# Oracle

1. Driver

This goes into the pom.xml file:

|  |
| --- |
| <dependency>  <groupId>com.oracle.database.jdbc</groupId>  <artifactId>ojdbc8</artifactId>  <version>19.10.0.0</version>  </dependency> |

Additional versions of the artifact are available and may work better with your database.

1. Dialect

*spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect*

Or, for example:

*spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect*

Additional versions of the dialect are available and may work better with your database.

1. Additional configuration: *none needed.*
2. Example configuration

|  |
| --- |
| spring.datasource.driverClassName="oracle.jdbc.OracleDriver"  [spring.datasource.url="jdbc:oracle:thin:@//example.com:1532/SUBS](mailto:spring.datasource.url=%22jdbc:oracle:thin:@//example.com:1532/SUBS)"  spring.datasource.username="GSRSXXXXX”  spring.datasource.password="XXXXXXXXX"  spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect spring.jpa.hibernate.ddl-auto=none |

**Note from 3.0.2:** When using GSRS with a newly-created Oracle database generated with GSRS 3.x, you must run the SQL script at

https://github.com/ncats/gsrs3-main-deployment/tree/main/substances/scriptForOracle

before loading data.

# PostGreSQL

1. Driver

This goes into the pom.xml file:

|  |
| --- |
| <dependency>  <groupId>org.postgresql</groupId>  <artifactId>postgresql</artifactId>  <version>42.2.19 </version>  </dependency> |

Additional versions of the artifact are available and may work better with your database.

1. Dialect

|  |
| --- |
| spring.jpa.database-platform = gsrs.repository.sql.dialect.GSRSPostgreSQLDialectCustom |

**Note:** This is a custom dialect provided by GSRS team for backwards compatibility with 2.x.

1. Additional configuration: *none needed.*
2. Example configuration

|  |
| --- |
| spring.datasource.driverClassName=" org.postgresql.Driver"  spring.datasource.url="jdbc: postgresql://example.com:5432/SUBS"  spring.datasource.username="GSRSXXXXX”  spring.datasource.password="XXXXXXXXX"  spring.jpa.database-platform=gsrs.repository.sql.dialect.GSRSPostgreSQLDialectCustom  spring.jpa.hibernate.ddl-auto=none |

**Note from 3.0.2:** The dialect for postgresql that is used by GSRS was modified in 3.0.2 to handle large objects differently. This makes the new dialect compatible with 2.X-based database schemes, but those databases formed from 3.0 and 3.0.1 dialects would need to be adjusted. Some adjustments to some columns may needed for a database formed in 3.0.1 or 3.0.0.

# MariaDB

1. Driver

This goes into the pom.xml file:

|  |
| --- |
| <dependency>  <groupId>org.mariadb.jdbc</groupId>  <artifactId>mariadb-java-client</artifactId>  <version>1.5.7</version>  </dependency> |

Additional versions of the artifact are available and may work better with your database.

1. Dialect: *No specific dialect setting is required for MariaDB*
2. Additional configuration:

An extra line in the database configuration section of application.conf is necessary to be compatible with a GSRS 2.x database:

|  |
| --- |
| # Discouraged starting in 3.1.1 spring.jpa.hibernate.use-new-id-generator-mappings=false |

1. Example configuration

|  |
| --- |
| spring.datasource.driverClassName="org.mariadb.jdbc.Driver"  spring.datasource.url="jdbc:mariadb://example.com:3306/SUBS"  spring.datasource.username="GSRSXXXXX”  spring.datasource.password="XXXXXXXXX"  # The default for mappings, is true. We encourage true starting in GSRS 3.1.1  # The setting will have no effect after 3.1.1 spring.jpa.hibernate.use-new-id-generator-mappings=false  spring.jpa.hibernate.ddl-auto=none |

s

# MySQL

1. Driver

This goes into the pom.xml file:

|  |
| --- |
| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.22</version>  </dependency> |

Note: this is version-specific. (Database version 8.0.22 worked with JDBC driver v. 8.0.22, not with 5.1.33)

1. Dialect

|  |
| --- |
| spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect |

**Note:** This dialect assumes the use of MySQL8. Other dialects may be appropriate based on the specific case.

1. Additional configuration:

An extra line in the database configuration section of application.conf is necessary to be compatible with a GSRS 2.x database:

|  |
| --- |
| # Discouraged starting in 3.1.1 spring.jpa.hibernate.use-new-id-generator-mappings=false |

Also, while not necessary when using a 2.x database, creating a new database with ddl-auto settings may result in failures with this version of MySQL. GSRS developers have found that running the following SQL directly fixes this issue. The main change below from the generated schema is the change in length for description to be 400 instead of 4000.

|  |
| --- |
| create table ix\_ginas\_vocabulary\_term (DTYPE varchar(31) not null,  id bigint not null,  created datetime(6),  deprecated bit not null,  modified datetime(6),  version bigint,  description varchar(400),  display varchar(400),  filters longtext,  hidden bit not null,  origin varchar(255),  regex varchar(400),  selected bit not null,  value varchar(400),  system\_category varchar(255),  fragment\_structure varchar(255),  simplified\_structure varchar(255),  namespace\_id bigint,  owner\_id bigint,  primary key (id)) engine=InnoDB; |

1. Example configuration

|  |
| --- |
| spring.datasource.driverClassName="com.mysql.jdbc.Driver"  spring.datasource.url="jdbc:mysql://example.com:3306/SUBS"  spring.datasource.username="GSRSXXXXX”  spring.datasource.password="XXXXXXXXX"  # The default for mappings, is true. We encourage true starting in GSRS 3.1.1  # The setting will have no effect after 3.1.1 spring.jpa.hibernate.use-new-id-generator-mappings=false  spring.jpa.hibernate.ddl-auto=none |