Database scripts (Flyway)

database

indexmenu_n_50

The application uses the Flyway tool to create and update the database scheme. Each module can contain its own set of scripts, which are applied to the application database scheme at the start of the application (in the phase after initializing the data connection, or rather after initializing the hibernate context) ⇒ **the modules are automatically initializing and updating the database scheme themselves.**

Supported features:

- Flyway migration can be enabled configurationally according to the application profile. For testing purposes, automatic generating of scheme into h2 via hibernate is used (switching would require creating and maintaining two sets of scripts, which will be done in a later phase of development),
- ability to use various database types (jdbc). The sets of scripts must be placed in a folder according to the name of the database used (see the example below) - the jdbc database used is determined dynamically before the Flyway migration itself (IdmFlywayMigrationStrategy).
- multi-module database scripts - each module controls its own part of the database scheme.

Naming convention of the database scripts:

- scripts are located in a folder according to the database used, e.g. **eu/bcvsolutions/idm/core/sql/postgresql** on the classpath of the module
- non-versioned scripts
  - **R__create-view.sql** - the scripts can be run repeatedly (e.g. create or replace view, package, trigger ...)
- versioned scripts
  - can be applied only once in a given sequence
  - **V1_00_000__init.sql** - empty initiation script for the module,
  - **V1_00_001__core-create.sql** - the first functional script for the module,
  - individual numeric series serve as numbers of major-minor-patch versions and are separated by an underscore character
- two underscore characters are followed by a textual description of the script (purpose). Words are separated by a dash.
- the content of the database scripts **must comply with the database design convention**.

An example configuration for the core module

Place the file `flyway-core.properties` on the classpath:

```
## Core Flyway configuration
#
# Whether to automatically call baseline when migrate is executed against a
non-empty schema with no metadata table.
# This schema will then be baselined with the baselineVersion before
executing the migrations.
```

```
# Only migrations above baselineVersion will then be applied.
# This is useful for initial Flyway production deployments on projects with
an existing DB.
# Be careful when enabling this as it removes the safety net that ensures
Flyway does not migrate the wrong database in case of a configuration
mistake!
flyway.core.baselineOnMigrate=true
#
# The name of Flyway's metadata table (default **schema_version**).
# By default (single-schema mode) the metadata table is placed in the
default schema for the connection provided by the datasource.
flyway.core.table=idm_schema_version_core
#
# Comma-separated list of locations to scan recursively for migrations. The
location type is determined by its prefix.
# Unprefixed locations or locations starting with classpath: point to a
package on the classpath and may contain both sql and java-based migrations.
# Locations starting with filesystem: point to a directory on the filesystem
and may only contain sql migrations.
# IdmFlywayMigrationStrategy resolves used jdbc database dynamically -
${dbName} in location could be used.
flyway.core.locations=classpath:eu/bcvsolutions/idm/core/sql/${dbName}
```

We expect to use a **postgresql** or **MS SQL server** database, therefore all the scripts will be found in a concrete folder `eu/bcvsolutions/idm/core/sql/postgresql` or `eu/bcvsolutions/idm/core/sql/sqlserver` on the classpath, where we can insert the empty initiation script `V1\_00\_000\_\_init.sql`.

In java, we will add a configuration to the scanning package. The configuration will use the prepared property file and run the migration:

```
package eu.bcvsolutions.idm.core.config.flyway.impl;

import org.flywaydb.core.Flyway;
import org.springframework.boot.autoconfigure.AutoConfigureAfter;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnExpression;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.boot.autoconfigure.flyway.FlywayProperties;
import org.springframework.boot.context.properties.ConfigurationProperties;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.DependsOn;
import org.springframework.context.annotation.PropertySource;
```

```
import eu.bcvsolutions.idm.core.config.flyway.AbstractFlywayConfiguration;
import eu.bcvsolutions.idm.core.config.flyway.IdmFlywayPostProcessor;
import eu.bcvsolutions.idm.core.config.flyway.IdmFlywayAutoConfiguration;

@Configuration
@ConditionalOnClass(Flyway.class)
@ConditionalOnProperty(prefix = "flyway", name = "enabled", matchIfMissing =
false)
@AutoConfigureAfter(IdmFlywayAutoConfiguration.IdmFlywayConfiguration.class)
@EnableConfigurationProperties(FlywayProperties.class)
@PropertySource("classpath:/flyway-core.properties")
public class FlywayConfigCore extends AbstractFlywayConfiguration {

    private static final org.slf4j.Logger log =
org.slf4j.LoggerFactory.getLogger(FlywayConfigCore.class);

    @Bean
    @ConditionalOnMissingBean(name = "flywayCore")
    @DependsOn(IdmFlywayPostProcessor.NAME)
    @ConditionalOnExpression("${flyway.enabled:true} &&
'${flyway.core.locations}'!=''")
    @ConfigurationProperties(prefix = "flyway.core")
    public Flyway flywayCore() {
        Flyway flyway = super.createFlyway();
        log.info("Starting flyway migration for module core [{}]: ",
flyway.getTable());
        return flyway;
    }
}
```

After running the application, there will be the table `idm\_schema\_version\_core` in the database with a log about running an empty initiation script (« Flyway Baseline »).

An example script

Following scripts can be used if you want to add a new column to some existing table.

Note that the syntax of the script for SQL server is different from the syntax of the script for PostgreSQL.

**src/main/resources/eu/bcvsolutions/idm/core/sql/postgresql/V9_02_001\notification-config-disabled.sql = the first script which will be used in the core version 9.2. <code sql>-- -- CzechIdM 9 Flyway script -- BCV solutions s.r.o. -- -- add disabled column to idm_notification_configuration ALTER TABLE idm_notification_configuration ADD COLUMN disabled boolean NOT NULL DEFAULT false; ALTER TABLE idm_notification_configuration_a ADD COLUMN disabled boolean; ALTER TABLE idm_notification_configuration_a ADD COLUMN disabled_m boolean; </code>
src/main/resources/eu/bcvsolutions/idm/core/sql/sqlserver/V9_02_001\notification-config-disabled.sql**

```
--
-- CzechIdM 9 Flyway script
-- BCV solutions s.r.o.
--
-- add disabled column to idm_notification_configuration

ALTER TABLE idm_notification_configuration ADD disabled bit NOT NULL DEFAULT
0;
ALTER TABLE idm_notification_configuration_a ADD disabled bit;
ALTER TABLE idm_notification_configuration_a ADD disabled_m bit;
```