The automatic execution of the update scripts is especially interesting when you build smaller applications that don't run in a highly regulated enterprise environment. In these situations, it's often not possible to run the update process yourself and there might be no operations team which executes the SQL scripts. So, you need to run the database update automatically when you start your application.

Run Liquibase as part of a Java SE application

Before you can use the Liquibase API, you need to add the required dependencies to your application. The following maven coordinates add the Liquibase core component in version 3.5.3 to your project.

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>3.5.3</version>
</dependency>
```

Integrate Liquibase into Hibernate's bootstrapping process

You first need to build a *StandardServiceRegistry* and use it to instantiate a *MetadataSources* object. I don't provide any configuration when calling these methods and Hibernate reads the *hibernate.cfg.xml* file from the classpath.

```
// Prepare the Hibernate configuration
StandardServiceRegistry reg =
new StandardServiceRegistryBuilder().configure().build();
MetadataSources metaDataSrc =
        new MetadataSources(reg);
```

In the next step, you can use the *MetadataSources* object to get an instance of a *ConnectionProvider* service and retrieve a *java.sql.Connection*. Hibernate created this connection based on the configuration data in the *hibernate.cfg.xml* file. You can then use the *Connection* object to create a Liquibase-specific *JdbcConnection*.

```
// Get a database connection
Connection con = metaDataSrc.getServiceRegistry()
        .getService(ConnectionProvider.class).getConnection();
JdbcConnection jdbcCon = new JdbcConnection(con);
```

Now you've everything you need to initialize Liquibase. You first need to create a *Database* object and provide the name of the *changelog* file, a *ClassLoaderResourceAccessor* instance and the *Database* object. Then, you can call the update method with a reference to the context you want to use for your database update.

```
// Initialize Liquibase and run the update
Database database = DatabaseFactory.getInstance()
        .findCorrectDatabaseImplementation(jdbcCon);
Liquibase liquibase = new Liquibase("db.changelog.xml",
        new ClassLoaderResourceAccessor(), database);
liquibase.update("test");
```

After you've updated your database, you can follow Hibernate's standard bootstrapping process. You therefore use the *MetadataSources* object to build your *MetaData* and build a *SessionFactory*.

```
// Create Hibernate SessionFactory
sf = metaDataSrc.addAnnotatedClass(Author.class)
        .addAnnotatedClass(Book.class).buildMetadata()
        .buildSessionFactory();
```

Run Liquibase with Spring Boot

You just need to add Liquibase Core to your classpath to integrate it into your Spring Boot application.

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>3.5.3</version>
</dependency>
```

The Liquibase integration will automatically load the master changelog file from *db/changelog/db.changelog-master.yaml*.

Run Liquibase with a CDI container

You need to add the following 2 dependencies to your project and include them in your deployment to run Liquibase in a CDI container.

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>3.5.3</version>
</dependency>
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-cdi</artifactId>
    <version>3.5.3</version>
</dependency>
```

You then need to implement a CDI bean that produces a *CDILiquibaseConfig*, a *DataSource* and a *ResourceAccessor*.

```
@Dependent
public class LiquibaseCdiIntegration {


    @Resource
    private DataSource myDataSource;


    @Produces
    @LiquibaseType
    public CDILiquibaseConfig createConfig() {
        CDILiquibaseConfig config = new CDILiquibaseConfig();
        config.setChangeLog("//c:/tmp/db.changelog.xml");
        return config;
    }


    @Produces
    @LiquibaseType
    public DataSource createDataSource() throws SQLException {
        return myDataSource;
    }


    @Produces
    @LiquibaseType
    public ResourceAccessor create() {
        return new
ClassLoaderResourceAccessor(getClass().getClassLoader());
    }
}
```