EvolveDB - Documentation for Developer

Author:	Torben Eckwert, M.Sc.
E-Mail:	torben.eckwert@zdh.thm.de
Subject Area:	Computer Science
Supervisors:	Prof. Dr. rer. nat. Michael Guckert Prof. Dr. ing. Gabriele Taentzer

December 4, 2024 , Wetzlar

FORSCHUNGSCAMPUS MITTELHESSEN

Contents

1 Introduction

EvolveDB is an Eclipse-based framework designed to facilitate the evolution of relational database schemas. Users can specify schema evolution steps by editing a reverse-engineered database model. The framework automatically analyzes the differences between the current schema and the modified model, generating a corresponding data migration script to ensure consistency.

This user manual provides comprehensive guidance, including step-by-step installation instructions and introductory tutorials tailored to help end users navigate and utilize EvolveDB effectively.

2 Prerequisites and Installation

EvolveDB is a plug-in for recent versions of the Eclipse Modeling Tools (last tested with version 2022-06). It is an open-source project licensed under the Apache License 2.0. The complete source code is freely accessible on our GitHub repository.

2.1 Installation via Eclipse Installer

Setting up a development environment can be a time consuming task. Such a setup can include:

- the installation of certain plug-ins
- the configuration of certain preferences
- cloning certain Git repositories
- importing projects from these Git repositories

Manually performing these steps is not only tedious but also prone to errors. Fortunately, the Eclipse Installer simplifies this process. As a standalone application, it is designed to install Eclipse products and keep them up-to-date. The underlying technology, Oomph, extends these capabilities to automate the provisioning of a pre-configured and customized Eclipse IDE. Oomph allows developers to automate their setup process using a setup profile. A setup profile is a configuration file that contains the tasks required to configure your Eclipse IDE precisely as needed. To streamline your workflow, our GitHub repository includes an Oomph setup file, located in the documentation folder.

2.1.1 Getting Started with the Oomph Installer

Download the Eclipse Installer from the official Eclipse website. Select the installer package appropriate for your platform and extract it. To use a setup file, launch the installer and switch to Advanced Mode. From there, you can load the provided setup file to automate the configuration of your Eclipse environment, saving time and reducing the likelihood of errors.

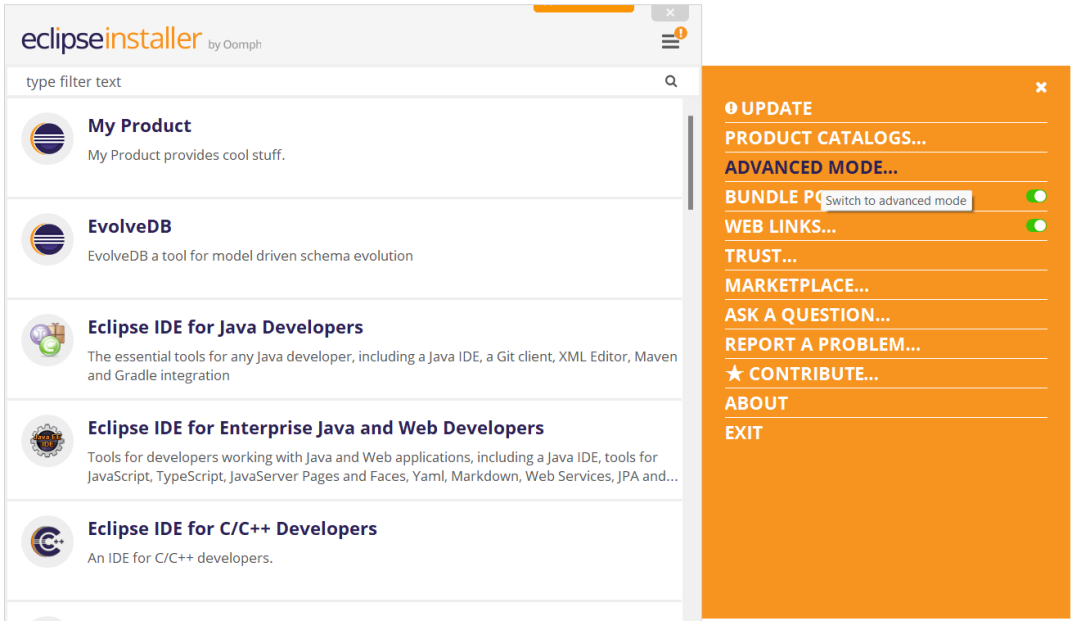

Figure 1: Eclipse installer: Advanced Mode

Out of the box the installer provides certain default categories containing different profiles, e.g., for Eclipse Projects. Additional user setups can be added by using the green plus button besides the filter text. After adding the custom EvolveDB setup file, the Eclipse installer creates an Eclipse environment with EvolveDB and all necessary plugins installed.

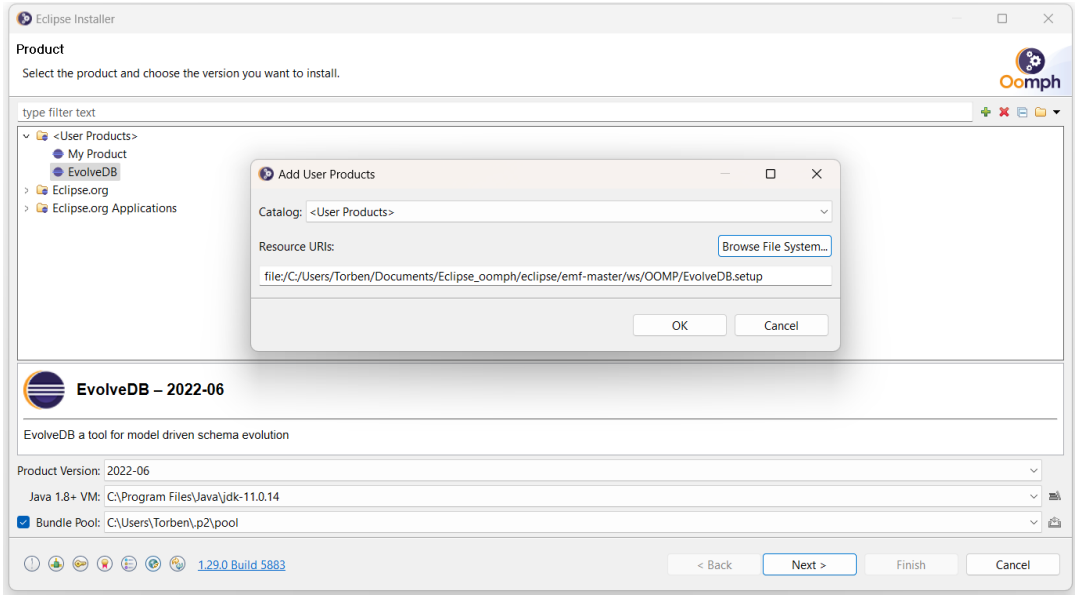

Figure 2: Eclipse installer: Additional user setup

The Eclipse Installer comes with several default categories, each containing various profiles tailored for Eclipse projects. These profiles provide a convenient starting point for setting up

development environments. To extend the available options, additional user-defined setup files can be added by clicking the green + button next to the filter text field. Once you add the custom EvolveDB setup file from our repository, the Eclipse Installer automatically creates a fully configured Eclipse environment. This setup includes EvolveDB and all required plug-ins, ensuring you are ready to start working immediately.

2.2 Manual installation

Before downloading the source code of EvolveDB, we need to install some additional Eclipse plugins. The following list contains the mandatory plugins.

- Eclipse OCL is available via the eclipse marketplace.
- The Henshin project provides a state-of-the-art model transformation language for the Eclipse Modeling Framework.
- ATL 3.5 (ATL Transformation Language) is a model transformation language and toolkit.
- ATL/EMFTVM 4.2.1 The EMF Transformation Virtual Machine (EMFTVM) is a run-time engine for the ATL Transformation Language (ATL).
- Xtend is a statically-typed programming languages for Java developers.

SiLift is a generic model comparison environment for EMF-based models. The SiLift update site is no longer available, but SiLift is included in the release version of EvolveDB. **Help ⇒ Install New Software...** (fig. 3).

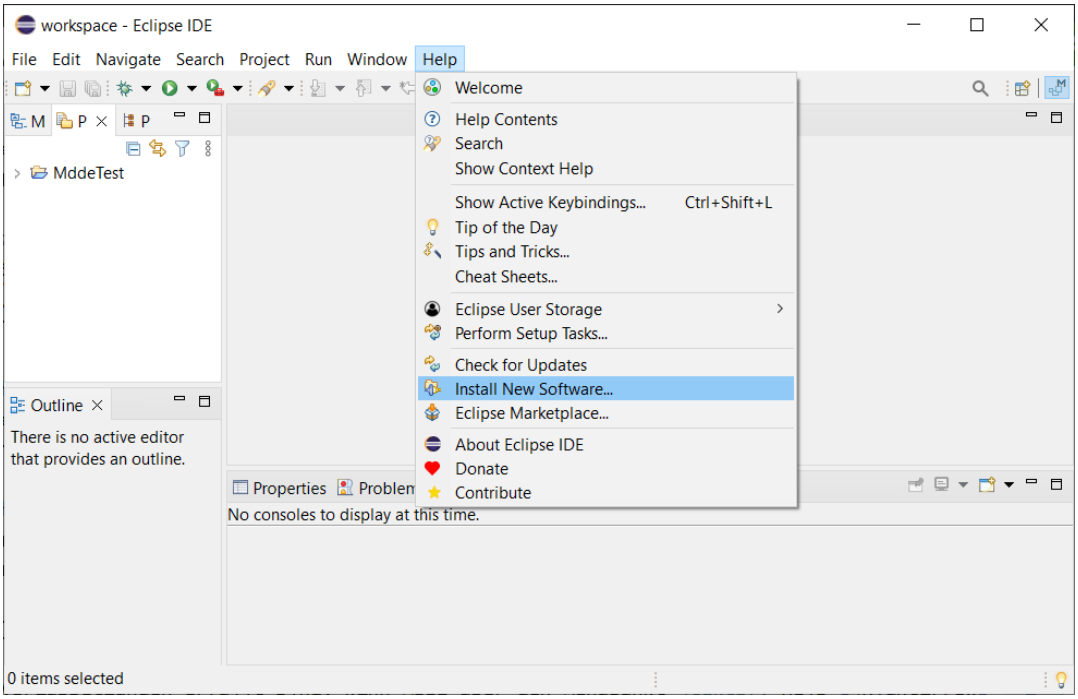

Figure 3: Eclipse: Install New Software...

The repository contains multiple categories (fig. 4). For installing SiLift, select *SiLift* and *SiLift Matcher*.

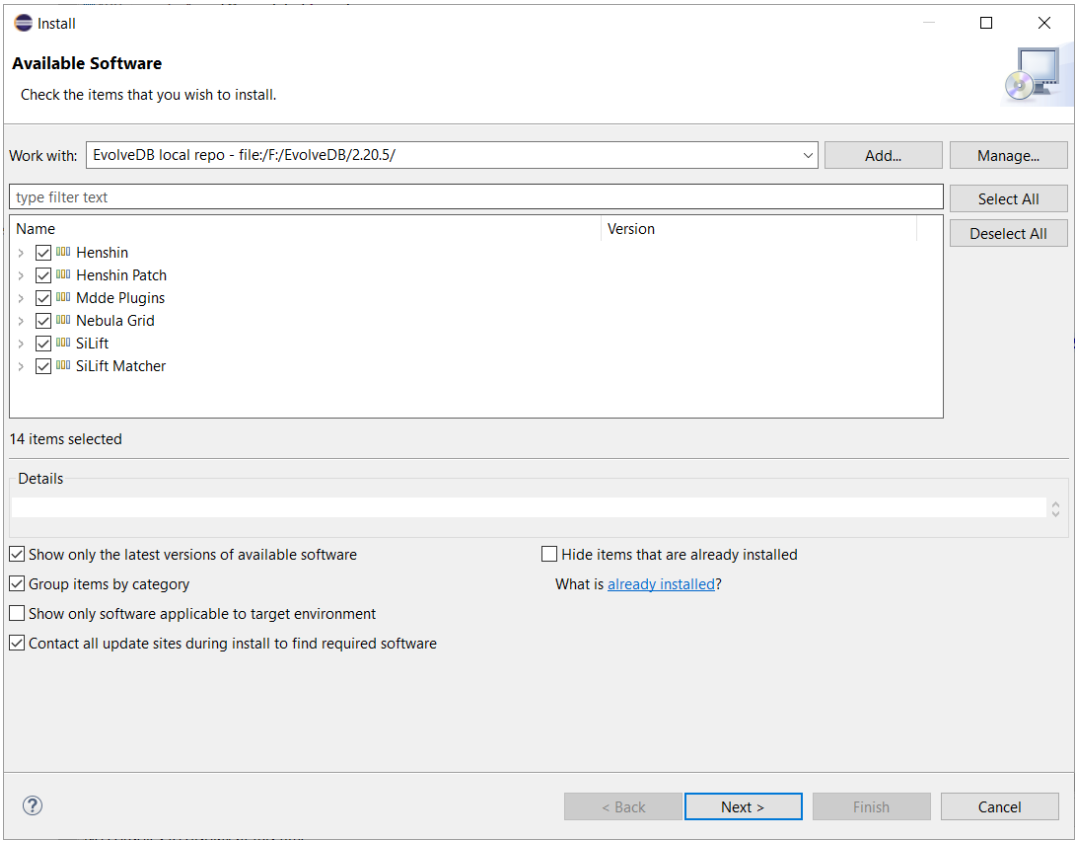

Figure 4: Eclipse: EvolveDB local repository

If all requirements are fulfilled, the source code of EvolveDB can be cloned and imported.

3 EvolveDB

EvolveDB employs a model-driven reengineering process for schema evolution, which consists of three key phases: reverse engineering, restructuring, and forward engineering.

- **Reverse Engineering:** The process begins with the creation of a model representing the existing database schema. This model is derived through reverse engineering, providing a structured view of the database.
- **Restructuring:**In this phase, the extracted model is edited to define the desired changes, resulting in a new version of the schema model. EvolveDB then analyzes the differences between the original schema and the evolved model.
- **Forward Engineering:** Using the calculated delta between the two model versions, EvolveDB generates SQL migration scripts. These scripts facilitate the migration of both the schema and the associated data, ensuring the database remains consistent.

EvolveDB includes:
A data source and a migration script generator designed to support MySQL databases. Two extension points that enable the integration of custom data sources or generators for other database systems. By contributing to these extension points, users can easily adapt EvolveDB to their specific requirements, making it a flexible and extensible solution for schema evolution.

3.1 Create a new datasource

EvolveDB supports the integration of custom drivers, allowing users to extend its functionality to suit specific requirements. To integrate a custom driver, follow these steps: Navigate to **File** ⇒ **New** ⇒ **Other** ⇒ **Plug-in Development** ⇒ **Plug-in Project** and open the MANIFEST.MF file. Switch to the *Dependencies* tab and add the required dependencies as shown in Figure 6. Ensure you include all necessary libraries and packages to ensure compatibility with EvolveDB.

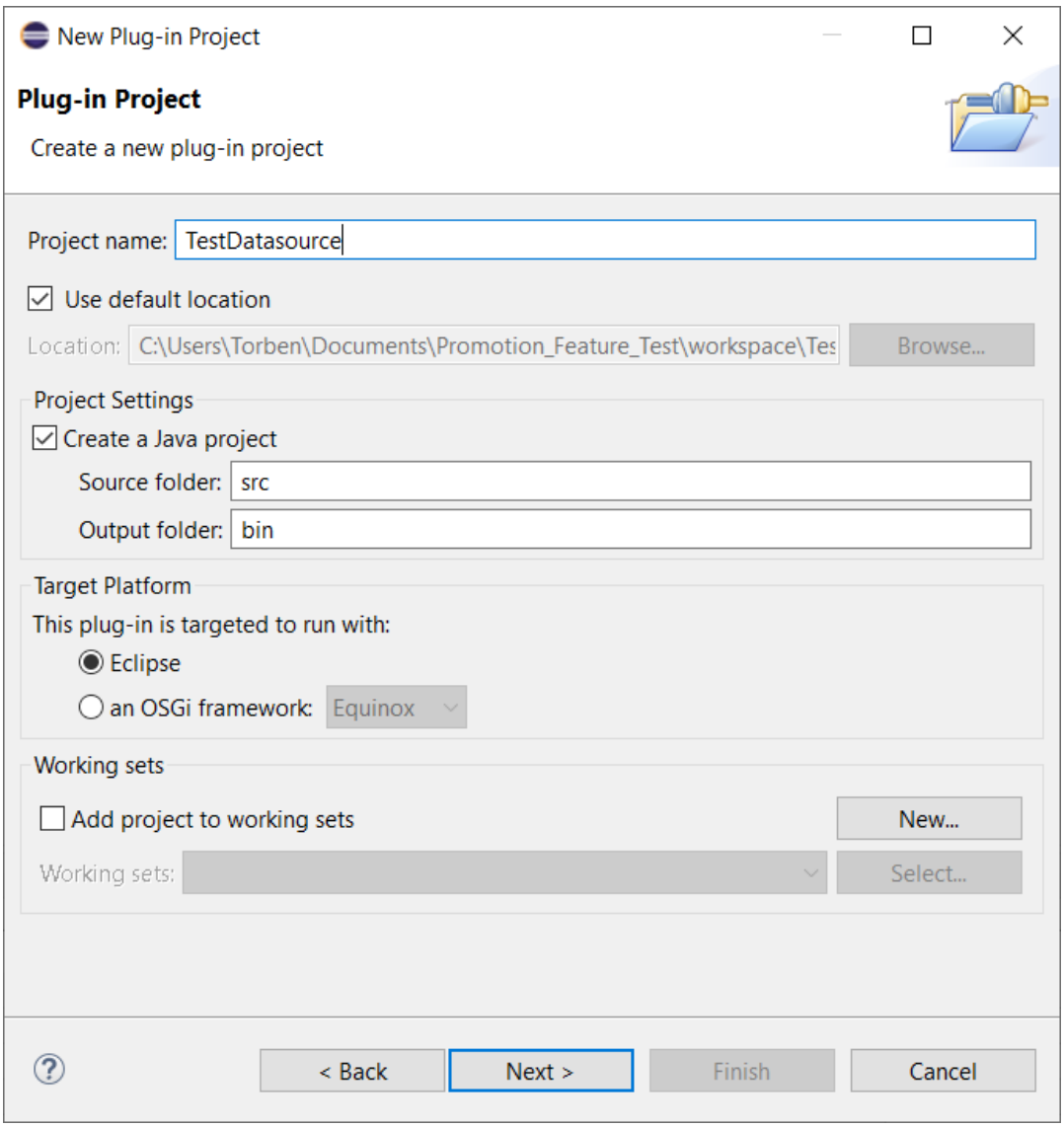

Figure 5: Eclipse: Create New Plug-in Project

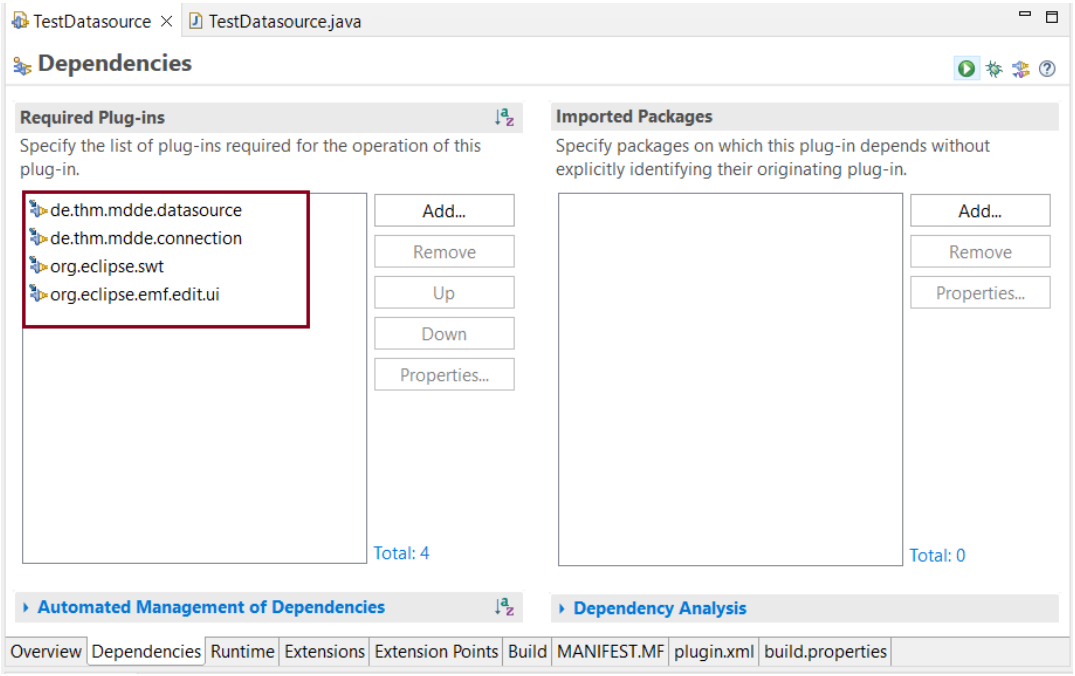

Figure 6: Plug-in Project Manifest.mf

Next, we need to create a class that implements the *EDBDataSource* interface (as shown in Figure 7). This interface defines the required methods that any custom data source must implement to integrate with EvolveDB. Figure 8 illustrates an example class, TestDatasource, which implements the EDBDataSource interface. This class serves as a blueprint for creating custom data sources. The implementation involves overriding the methods defined in the interface to provide the necessary functionality specific to your data source.

```java
import org.eclipse.emf.ecore.EObject;

public interface EDBDataSource {

    /**
     * Datasource name
     * @return the name of the datasource
     */
    String getName();


    /**
     * Connection icon
     * @return Database Icon
     */
    Image getImage();


    /**
     * DBPDriver class
     * @return Driver class
     */
    DBPDriver getDriver();



    /**
     * DBPDriverDependencies
     * @return the corresponding driver dependencies
     */
    DBPDriverDependencies getDriverDependencies();


    /**
     * The UI for entering the connection details.
     * @return
     */
    void openConnectionUi();


    /**
     * Returns the root element of the newly created model;
     * @return
     */
    EObject getRootObject();


    /**
     * Returns true if the user left the wizard via the cancel button.
     * @return
     */
    boolean isCanceled();


}
```

Figure 7: EDBDataSource Interface

TestDatasource × TestDatasource.java ×

```java
package de.thm.mdde.datasource;

import org.eclipse.emf.ecore.EObject;
import org.eclipse.swt.graphics.Image;

import de.thm.mdde.connection.model.DBPDriver;
import de.thm.mdde.connection.model.DBPDriverDependencies;

public class TestDatasource implements EDBDataSource{

    @Override
    public DBPDriver getDriver() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public DBPDriverDependencies getDriverDependencies() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Image getImage() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return "TestDriver";
    }

    @Override
    public EObject getRootObject() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean isCanceled() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void openConnectionUi() {
        // TODO Auto-generated method stub

    }

}
```

Figure 8: Class TestDatasource

After implementing the new class, the next step is to register it as an extension for EvolveDB. To achieve this, open the *MANIFEST.MF* file in your plug-in project and navigate to the Extensions tab. In this tab, click on the Add button to create a new extension. From the list of available extension points, select *de.thm.mdde.datasource* (as shown in Figure 9). This step

ensures that your custom class is recognized and integrated into EvolveDB as a valid data source.

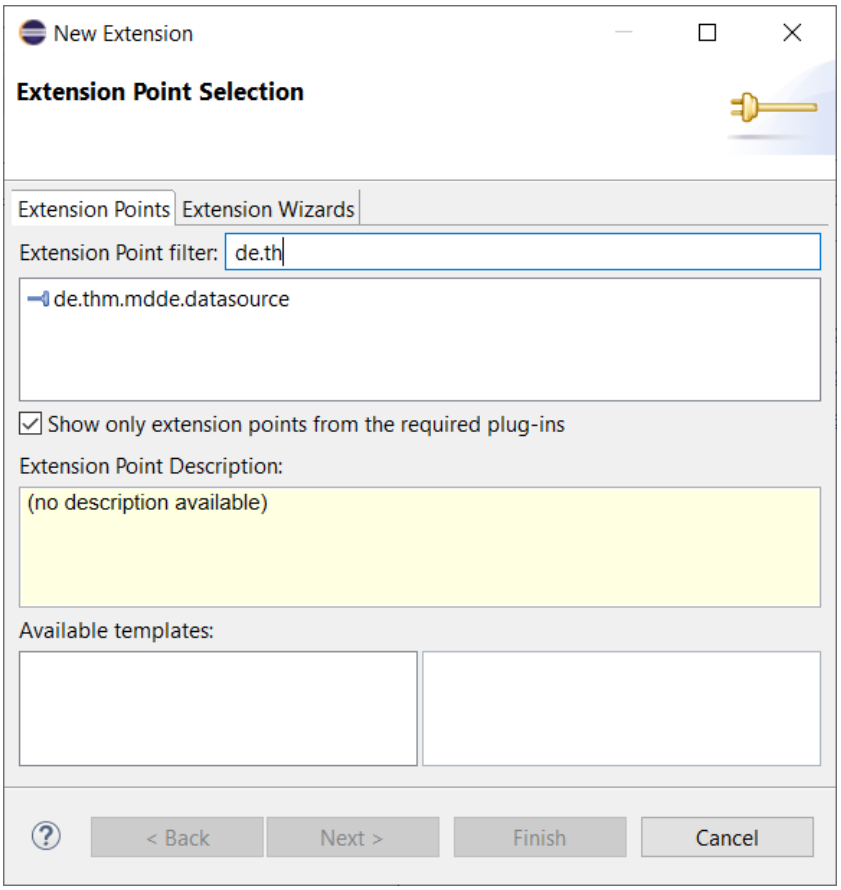

Figure 9: de.thm.mdde.datasource extension point

Select the *plugin.xml* and add the path to the class we created. Figure 10 shows the edited *plugin.xml*.

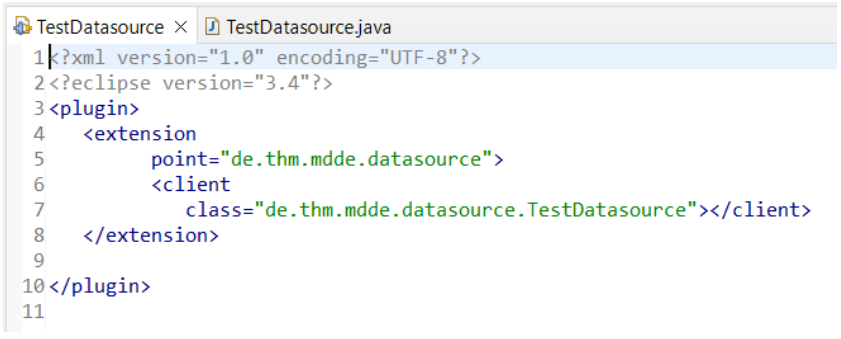

Figure 10: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer

and open the context menu with a right-click. In the context menu we choose export (11).

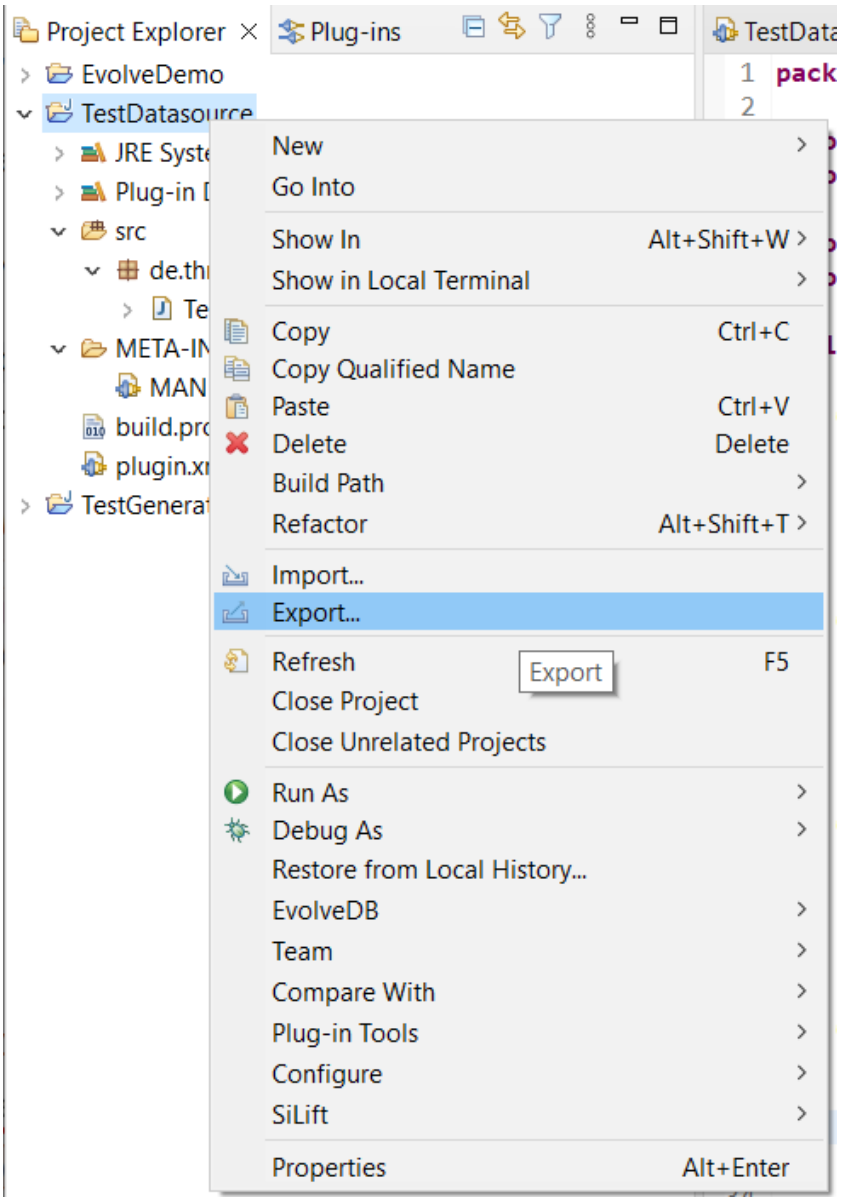

Figure 11: Export a plugin

A dialog box will appear. In this dialog, select **Plug-in Development** ⇒ **Deployable plug-ins and fragements** and then click Next (see Figure 12).

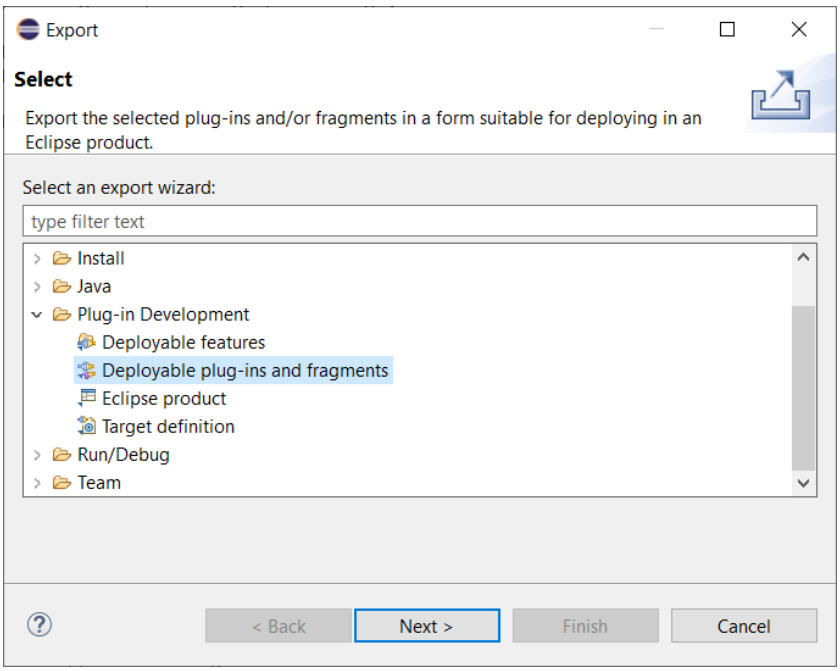

Figure 12: Export Wizard: page 1

In the next step, select Install into Host Repository and specify the path to the plugin folder of your Eclipse installation (see Figure 13). Once the path is set, click Finish to complete the installation. After the installation is successful, restart Eclipse. The newly created data source will now be available for use.

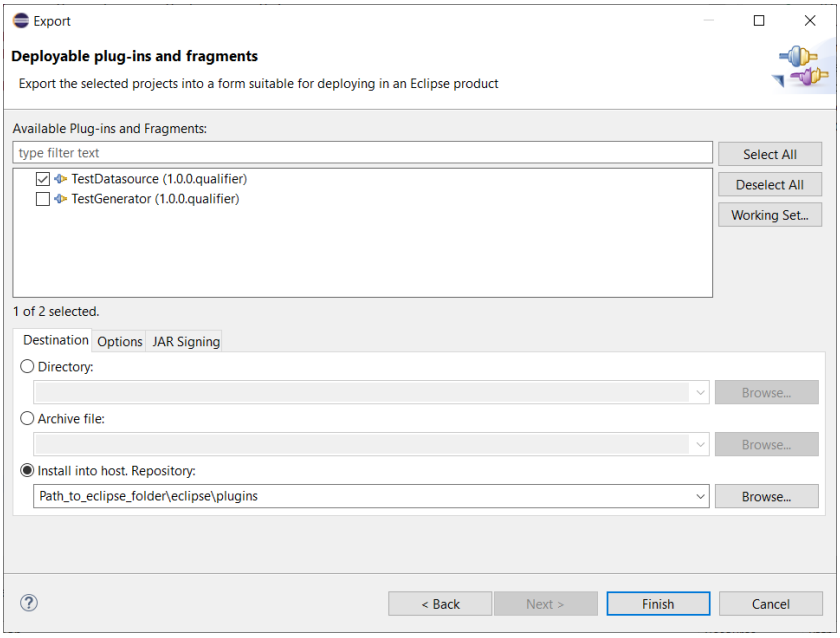

Figure 13: Export wizard: page 2

3.2 Create a new migration script generator

The default installation of EvolveDB includes a migration script generator compatible with MySQL version 5.7 or higher. However, it is possible to add custom generators by leveraging an extension point. To achieve this, start by creating a new plug-in project. Navigate to **File** ⇒ **New** ⇒ **Other** ⇒ **Plug-in Development** ⇒ **Plug-in Project** and complete the setup. Once the plug-in project is created, open the MANIFEST.MF file and switch to the Dependencies tab. Add the required dependencies as shown in Figure 15. This step ensures that your custom generator integrates seamlessly with EvolveDB.

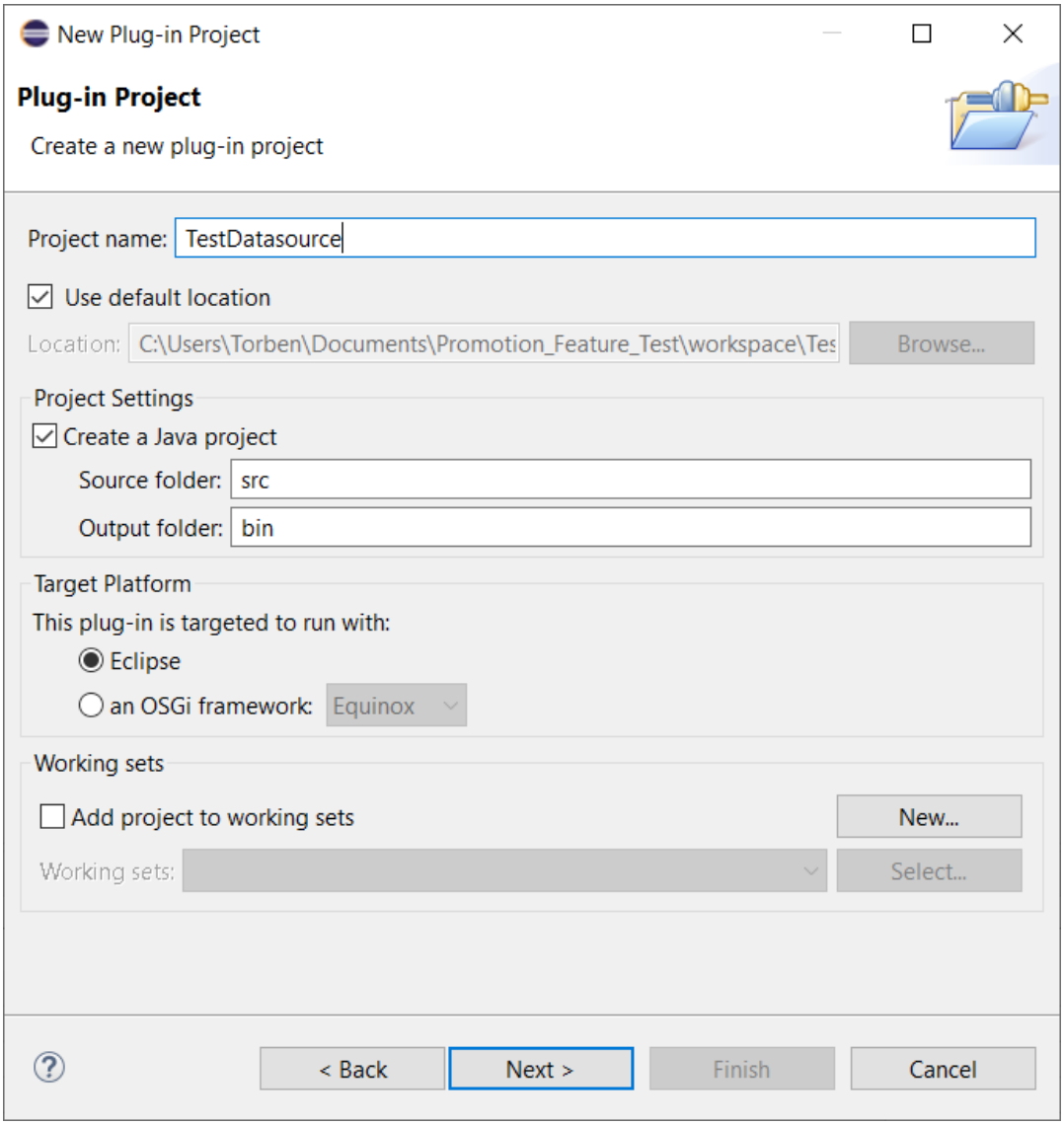

Figure 14: Eclipse: Create New Plug-in Project

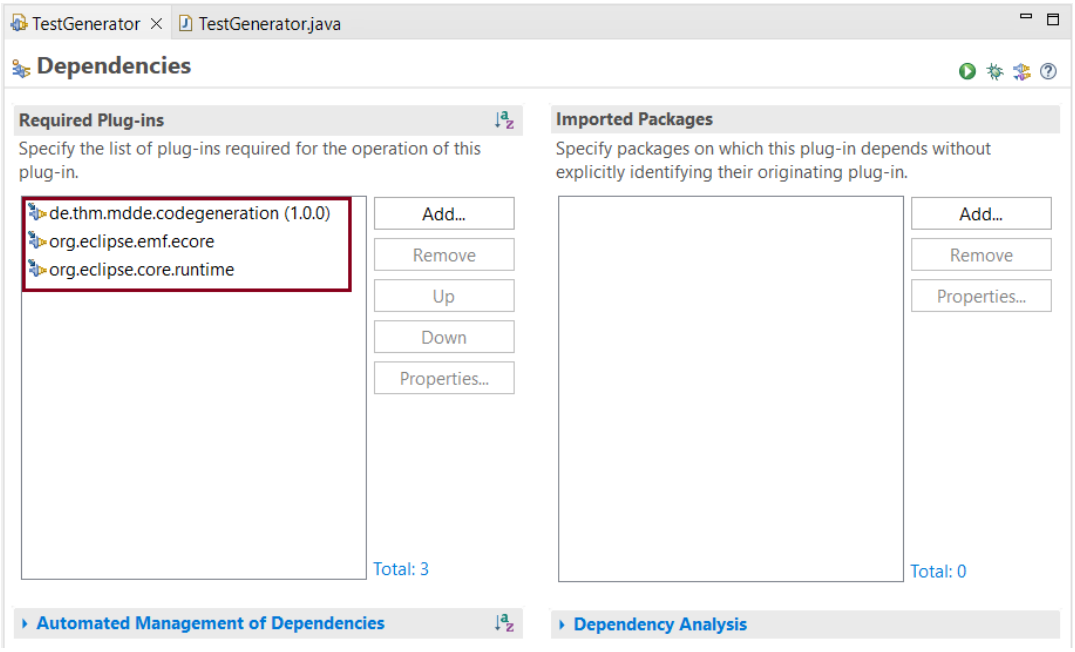

Figure 15: Plug-in Project Manifest.mf

Next, we need to create a class that implements the *ISQLGenerator* interface (as shown in Figure 16). This interface defines the required methods that any custom SQL generator must implement to integrate with EvolveDB. Figure 17 illustrates an example class, TestGenerator, which implements the *ISQLGenerator* interface.

```java
import org.eclipse.core.runtime.IProgressMonitor;

public interface ISQLGenerator {

    /**
     * Returns the display name of the extension.
     * @return
     */
    String getDisplayName();


    /**
     * Generate the migrations.
     * @param resEcoreFile --> The matching model.
     * @param project --> The currently selected project.
     * @param generator --> The generator chosen by the user.
     * @param monitor --> ProgressMonitor
     */
    void generate(Resource resEcoreFile, IProgressMonitor monitor);



    /**
     * Returns the content for the migration script.
     * @return
     */
    String getContent();

}
```

Figure 16: ISQLGenerator Interface

TestGenerator | TestGenerator.java

```java
package de.thm.mdde.generator;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.emf.ecore.resource.Resource;

import de.thm.extensionpoint.interfaces.ISQLGenerator;

public class TestGenerator implements ISQLGenerator{

    @Override
    public void generate(Resource arg0, IProgressMonitor arg1) {
        // TODO Auto-generated method stub

    }

    @Override
    public String getContent() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getDisplayName() {
        return "TestGenerator";
    }

```

Figure 17: Class TestGenerator

After implementing the new class, the next step is to register it as an extension for EvolveDB. To achieve this, open the *MANIFEST.MF* file in your plug-in project and navigate to the Exten-

sions tab. In this tab, click on the Add button to create a new extension. From the list of available extension points, select *de.thm.mdde.extensionpoint.SQLGenerator* (as shown in Figure 18). This step ensures that your custom class is recognized and integrated into EvolveDB as a valid data source.

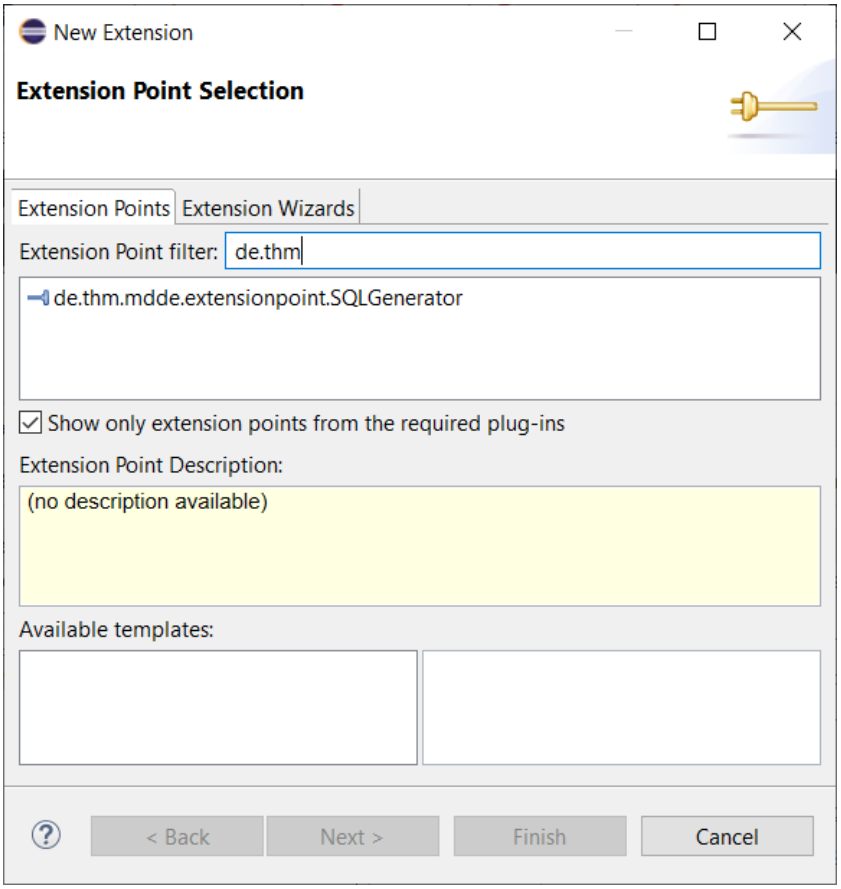

Figure 18: de.thm.mdde.extensionpoint.SQLGenerator extension point

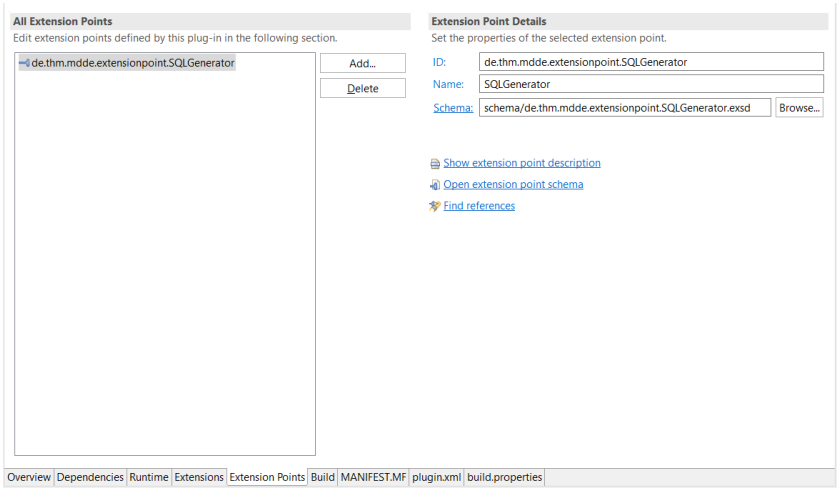

Figure 19: de.thm.mdde.extensionpoint.SQLGenerator extension point

Select the *plugin.xml* and add the path to the class we created. Figure 20 shows the edited *plugin.xml*.

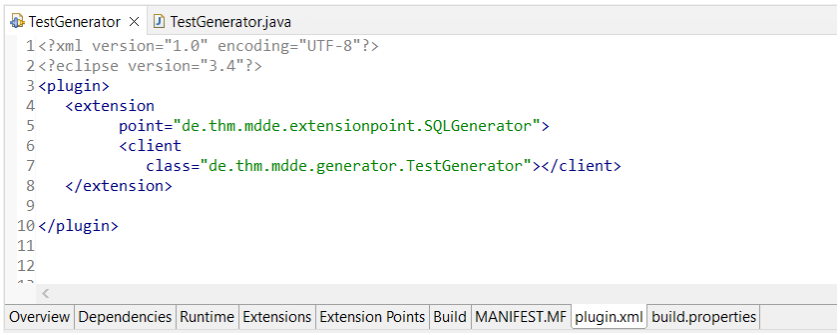

Figure 20: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer and open the context menu with a right-click. In the context menu, we choose export (21).

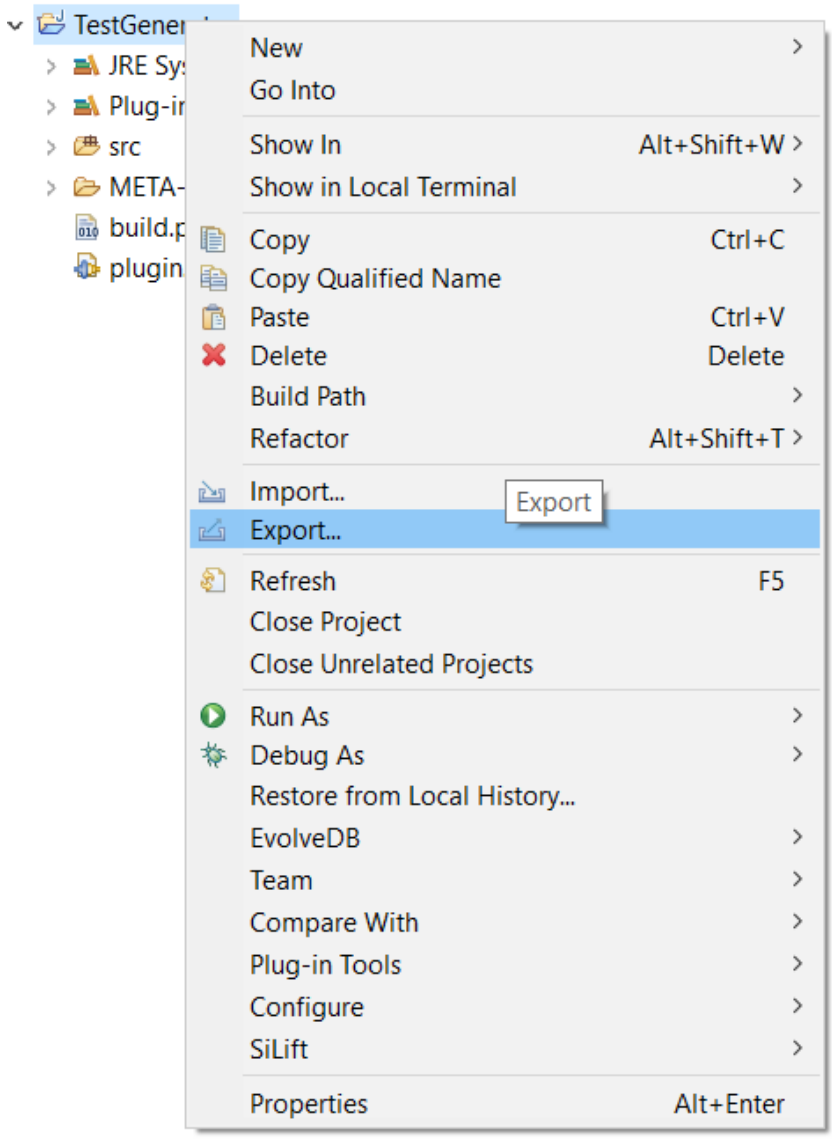

Figure 21: Export a plugin

A dialog box will appear. In this dialog, select **Plug-in Development** ⇒ **Deployable plug-ins and fragements** and then click Next (fig. 22).

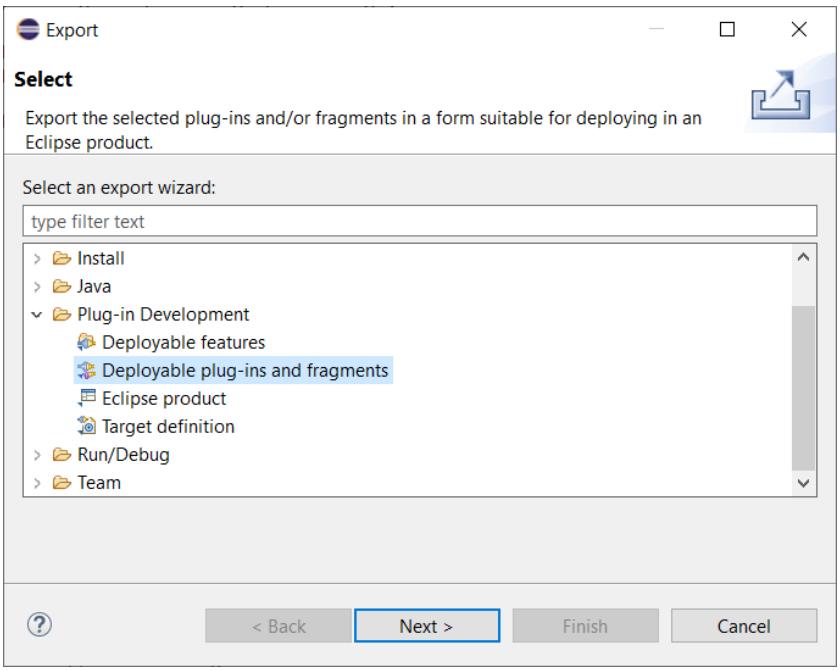

Figure 22: Export Wizard: page 1

In the next step, select Install into Host Repository and specify the path to the plugin folder of your Eclipse installation (see Figure 23). Once the path is set, click Finish to complete the installation. After the installation is successful, restart Eclipse. The newly created data source will now be available for use.

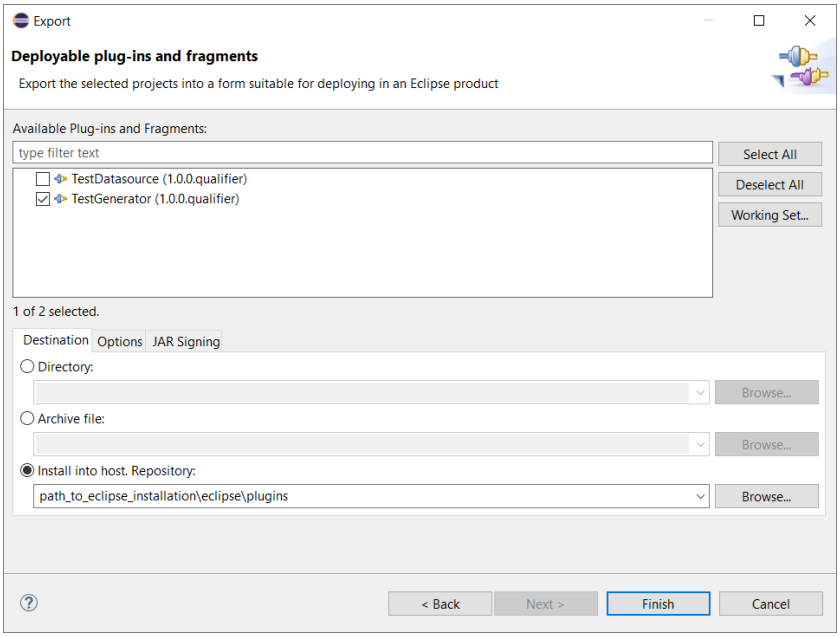

Figure 23: Export wizard: page 2

4 Metamodel

EvolveDB uses EMF for the database (MDDE model) and the migration metamodel. This section gives a short overview of both metamodels.

4.1 MDDE metamodel

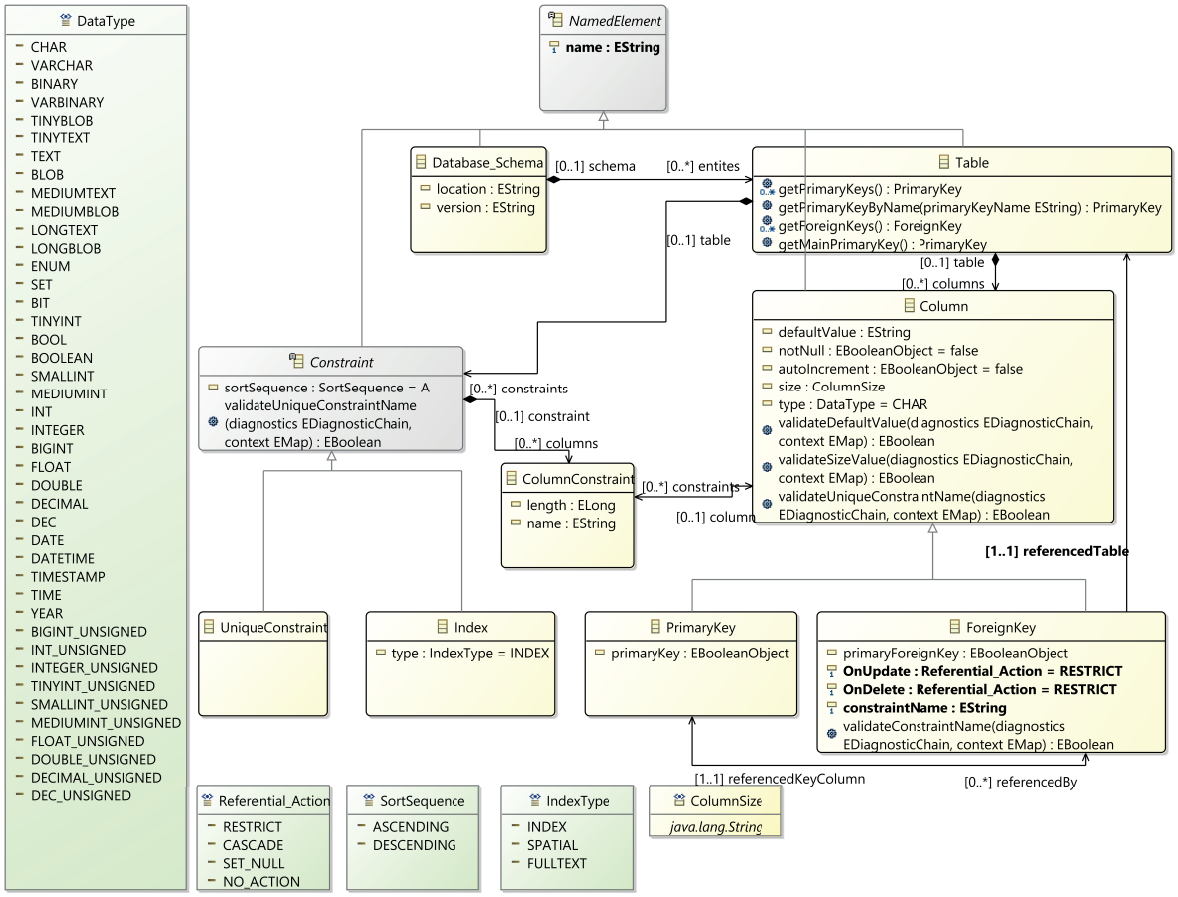

Figure 24: MDDE Database meta-model

The abstract syntax of the MDDE metamodel (see Figure 24) closely resembles the relational model. When implementing a new data source, the getRootObject() method from the EDBDataSource interface must return the root object of an instance of this metamodel. Typically, this root object is an instance of the *Database_Schema* class, which serves as the entry point for the model's structure and data.

4.2 Migration metamodel

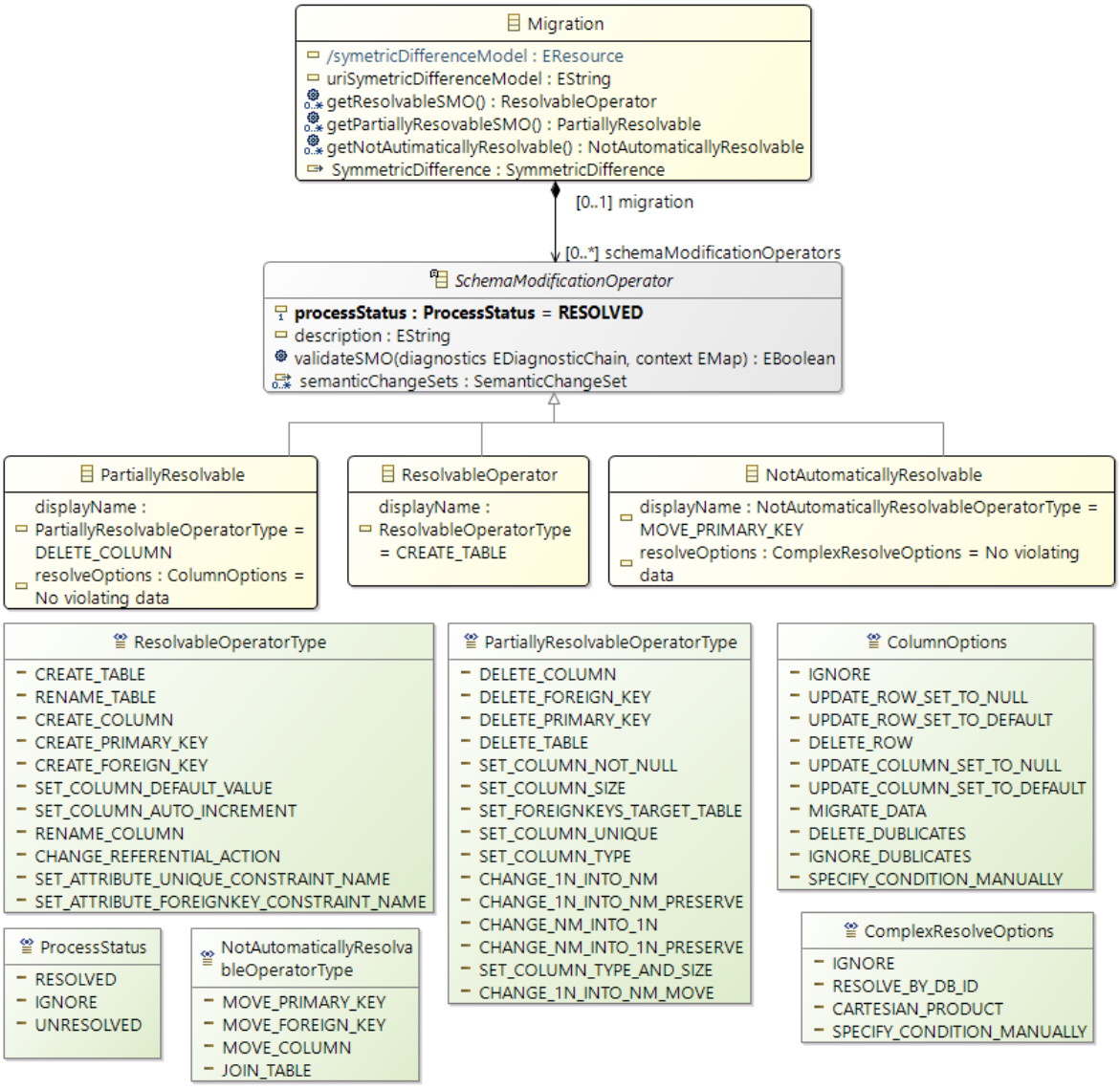

Figure 25: Migration metamodel

The symmetric difference model, generated by SiLift during the model matching phase, is used solely to represent the differences between the two model versions. However, this model is not designed to accommodate the additional information necessary for migration. To address this limitation, the difference.symmetric model is transformed into a migration model. The migration model serves as an intermediary, referencing the symmetric difference model as well as both MDDE models. Figure 25 illustrates the metamodel for the migration model, highlighting its role in bridging the gap between the symmetric differences and the migration requirements.