Progress®
Corticon®

Corticon Tutorial

Connecting a Progress Corticon Decision Service to a Database using EDC

Copyright

Last updated: Corticon 7.2

Updated: 2025/06/02

Table of Contents

1

Tutorial - Connecting a Progress Corticon Decision Service to a Database using EDC

Corticon Decision Services can connect to a database to perform read, write, and delete operations on it. If your Decision Service needs to access a database, your role as a Corticon integration developer is to enable database access. There are three techniques for Corticon database integration.

- Enterprise Data Connector (EDC) where data loads are moderate and tight integration with rules in an intuitive and friendly way is preferred.
- Advanced Data Connectors (ADC) that enable connection to multiple databases to access stored SQL queries through service call-outs.
- Batch Processing that let you use stored SQL queries to chunk large data flows and persist responses in a database.

This tutorial uses EDC. To get started, you need to perform two tasks:

- Map a Vocabulary to a database in Corticon Studio in one of two ways:
  - Create a database schema from an existing Vocabulary
  - Map an existing Vocabulary to an existing database schema
- Configure database access properties during deployment in Corticon Server.

In this tutorial, you will learn how to perform these tasks.

This tutorial is designed for hands-on use. We recommend that you install Corticon Studio, and then follow the instructions and illustrations in the tutorial. This tutorial is based on Corticon 7.1.

Note: You will install Corticon Server later as part of the tutorial.

2

How database elements map to Vocabulary elements

For rules in a Decision Service to read or write to a database, the Vocabulary elements used in the rules must be mapped to elements in the database. In the Basic Rule Modeling tutorial, you learned about Vocabulary elements—entities, attributes, and associations. Let's take a look at how these Vocabulary elements correspond to elements in a database.

As you know, a relational database is a set of tables. Each table in the database corresponds to an entity in a Vocabulary. Each record in the table corresponds to an entity instance in a Ruletest. Columns in the table correspond to attributes in the entity. Table joins or table relationships correspond to associations in the Vocabulary. Finally, the database's schema is like the Vocabulary tree.

This table summarizes the relationship between a Vocabulary and a database:

Vocabulary	Database
Entity	Table
Attribute	Table column
Association	Table join
Entity instance	Row or record
Vocabulary	Database schema

As an integration developer, you are responsible for mapping the Vocabulary to the database. You must map each entity used in a rule to a table in the database, each attribute to a column, etc. Once the mapping is configured, rules that use mapped entities and attributes will automatically access the database. You configure all of this mapping in Corticon Studio.

Before you do any mapping, you should gather and identify mapping requirements. Generally, you will have an **existing Vocabulary** and an **existing database schema**. You need to map Vocabulary elements with database elements.

3

Setting up the tutorial

Before you work on each scenario, you need to set up your environment for the tutorial. You must:

- Install Microsoft SQL Server 2019, Developer Edition
- Install Microsoft SQL Server Management Studio (SSMS)
- Configure Microsoft SQL Server 2019, Developer Edition
- Download and import a sample Corticon rule project

Note: EDC can work with a number of relational databases such as Oracle, Progress OpenEdge, IBM DB2, MySQL, and Microsoft SQL Server. This tutorial uses Microsoft SQL Server 2019, Developer Edition.

Step 1: Installing Microsoft SQL Server 2019 Developer and SQL Server Management Studio (SSMS)

Microsoft SQL Server Developer is a version of SQL Server that is free to use and distribute.

Note: There are several videos on the web that walkthrough the **SQL Server 2019 installation** process. They give you a step-by-step look at the process.

Follow these steps to download, install, and configure **SQL Server 2019 Developer** and the recommended **SQL Server Management Studio (SSMS)** software:

1. Browse to https://www.microsoft.com/en-us/evalcenter/evaluate-sql-server-2019 .
2. Download the free **EXE** edition installer.
3. Launch the downloaded package, `SQL2019-SSEI-Eval`.
4. Typically, accept all the default options and locations. Proceed through all the installer panels.

5. On the last panel, choose **Install SSMS**, to open its browser page. There, click **Download SSMS**, and then click **Download SQL Server Management Studio**. Run `SSMS-Setup-ENU.exe`, its installer.
6. When completed, launch `SQL Server Management Studio 20`. Click **Install**.
7. Launch `C:\SQL2019\Evaluation_ENU\SETUP.EXE`
   a. Choose **Installation**.
   b. Choose **New SQL Server standalone installation...**
   c. Proceed through the panels.
      a. On the **Product Key** panel, specify the free edition **Developer**.
      b. On the **Feature Selection** panel, choose **Database Engine Services**, and then click **Next**.
      c. On the **Instance Configuration** panel, choose **Default instance**.
      d. On the **Database Engine Configuration** panel:

   a. Select Mixed Mode (SQL Server authentication and Windows authentication)
   b. In the **Enter Password** and **Confirm Password** fields, enter

      `sqlserver2019`

      This password is for a default administrator user named `sa` as will be used in the sample's connections.

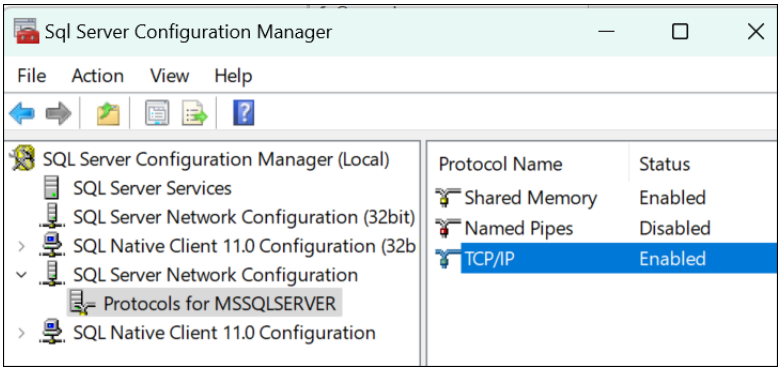
8. Click **Next** on the remaining panels, and then click **Close** to exit the wizard.

You've now installed SQL Server Developer and its tools. The SQL Server Developer database engine starts up automatically.

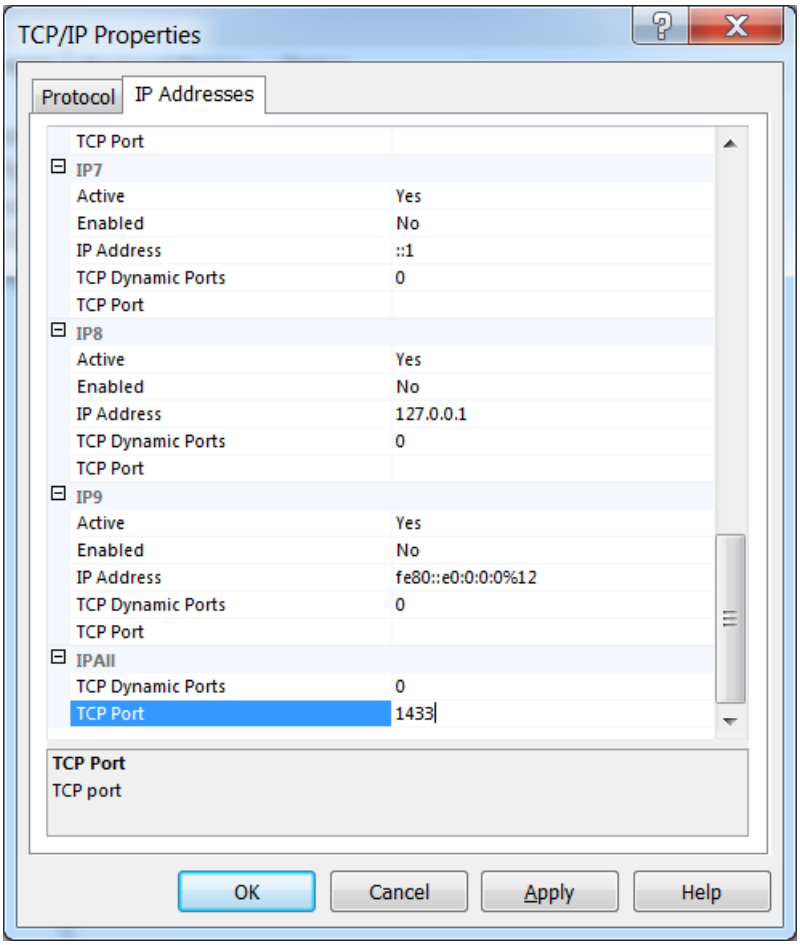
Step 2: Configuring Microsoft SQL Server Developer

Corticon wants to connect to a database through TCP/IP on a designated port. Configure SQL Server for that in these steps:

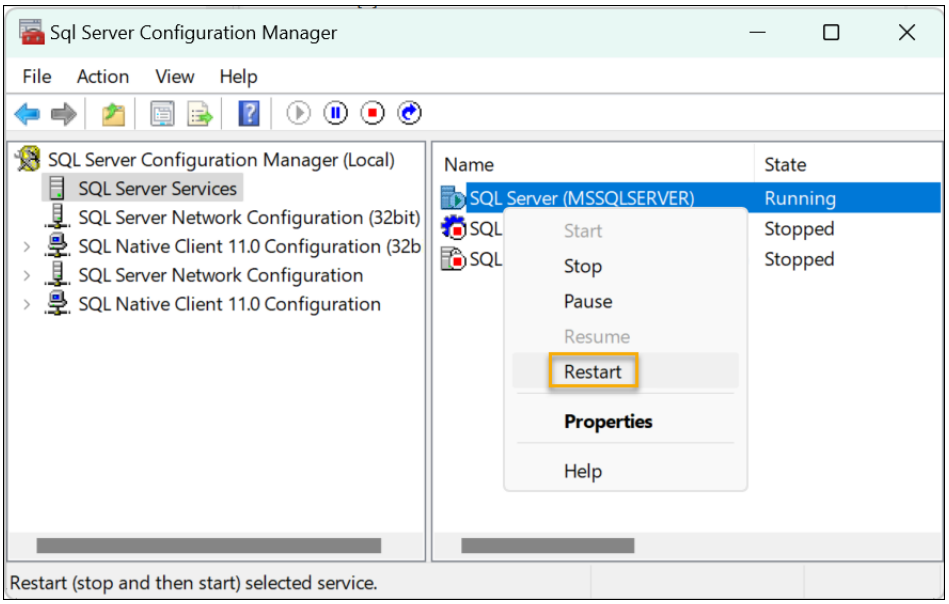
1. Choose **Start > SQL Server 2019 Configuration Manager**.
2. Expand **SQL Server Network Configuration** in the left pane and select **Protocols for SQLEXPRESS**.
3. Right-click **TCP/IP** in the right pane, and then select **Enabled**:



In the TCP/IP Properties window, click the **IP Addresses** tab and scroll to the bottom:



4. In the **IPAll** section's **TCP Port** field, enter **1433**, click **Apply**, and then click **OK**.
5. Restart SQL Server Express by selecting **SQL Server Services** in the left pane, right-clicking **SQL Server (MSSQLSERVER)** on the right, and then choosing **Restart**< as shown:

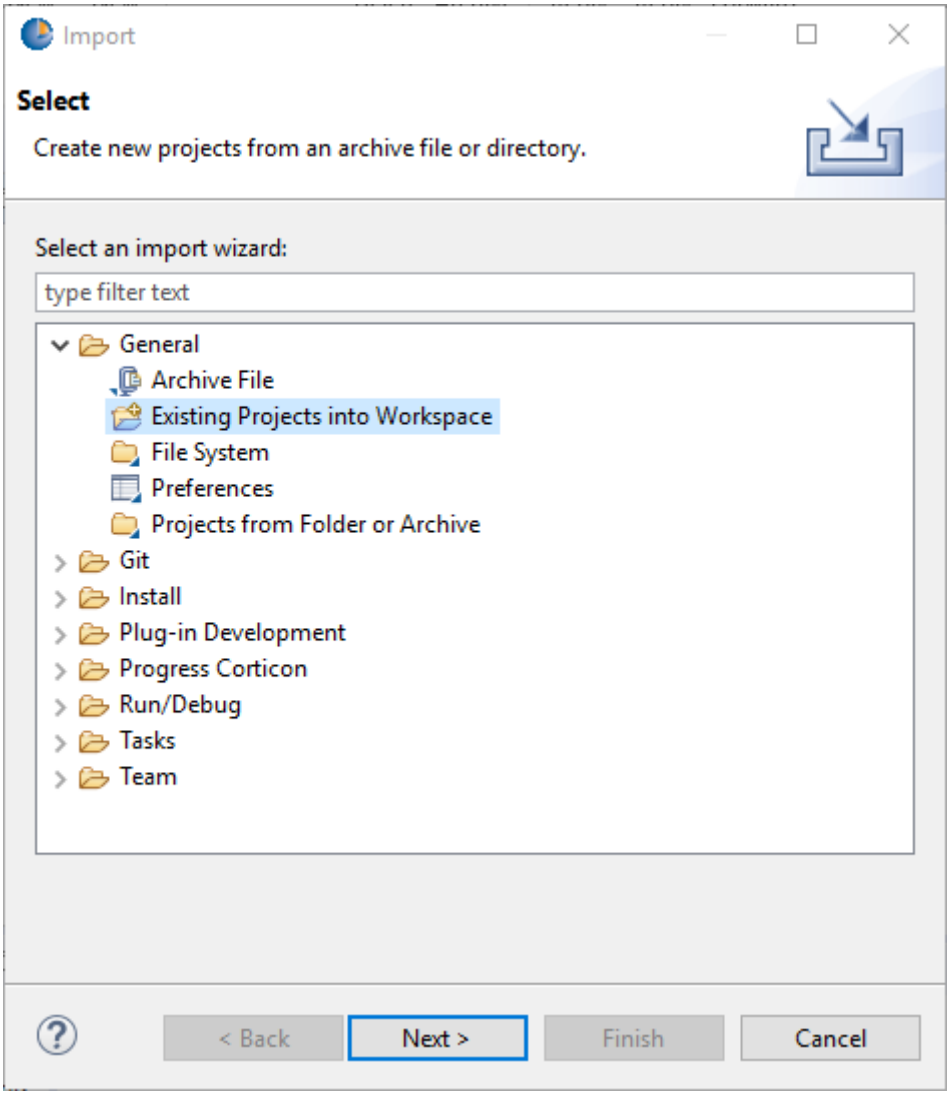


Step 3: Downloading and importing a sample rule project

Next, download and import the sample rule project Connecting_EDC which contains two Vocabularies, several Rulesheets and Ruletests, and a Ruleflow. As part of this tutorial, you will map these Vocabularies to databases in SQL Server. You can then run the Ruletests to verify that the rules in the Rulesheets are able to perform operations on the databases. Later in the tutorial, you will deploy the Ruleflow as a Decision Service using the appropriate database settings.

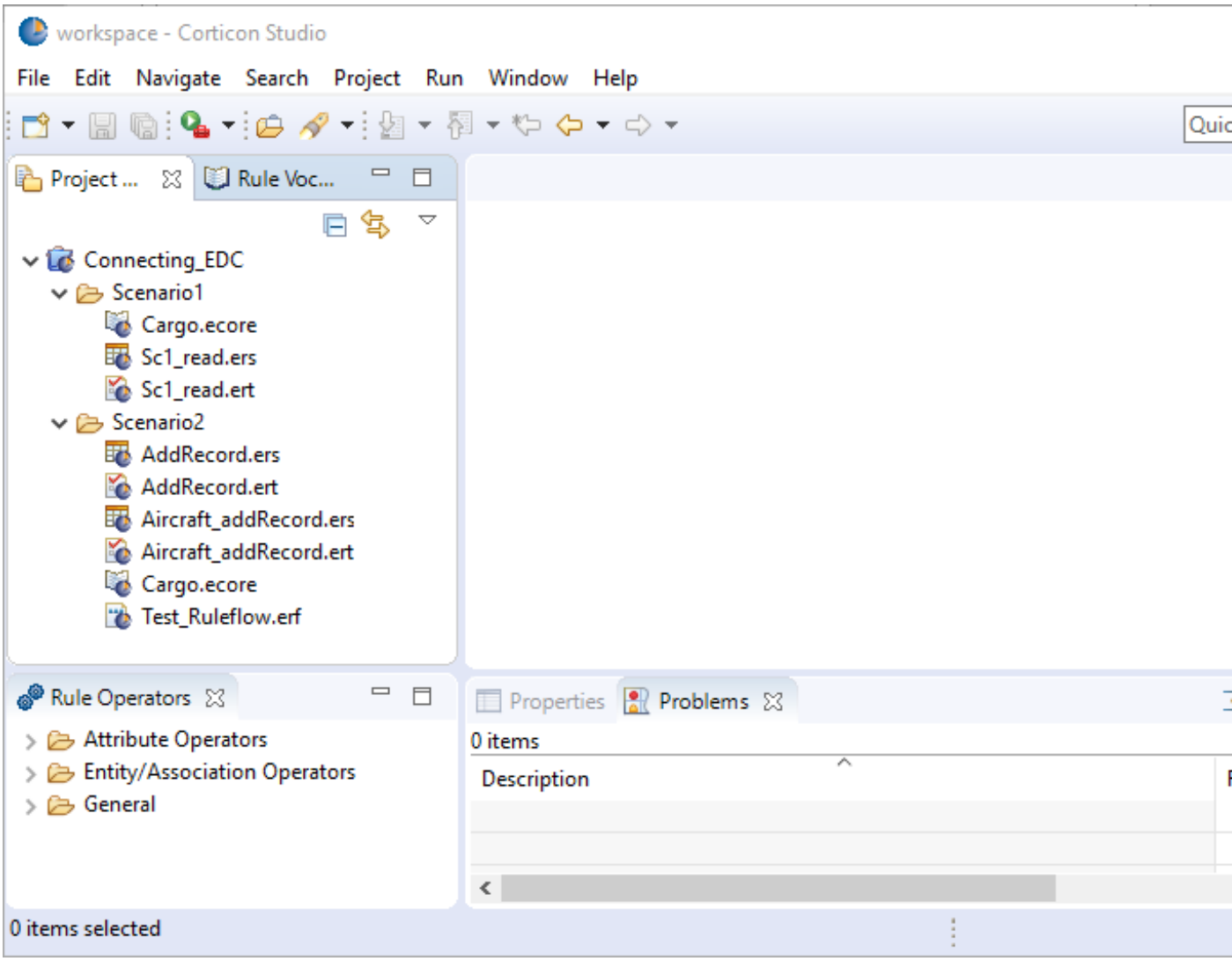
Follow these steps to download and import the sample rule project:

1. Click here to access the sample rule project **Connecting_EDC**.
2. Choose the **Start** menu command **Progress > Corticon Studio**.
3. In Studio, select the menu command **File > Import**.
4. In the **Import** wizard, select **General > Existing Projects into Workspace** and click **Next**.



5. On the Import Projects page:
   a. Click **Select archive file** and specify the location of the **Connecting_EDC.zip** file.
   b. Click **Finish**.

6. Expand the **Connecting_EDC** rule project, and you see two folders: **Scenario1** and **Scenario2**. Each folder contains a Vocabulary and some Rulesheets and Ruletests. The Scenario2 folder also includes the Ruleflow that you will deploy later.



7. Select the menu command **Project > Upgrade Rule Assets**. Click through the dialogs to ensure that the assets in both scenarios are up to date.

Success! Your environment is now ready to use!

You will use the Cargo Vocabulary in Scenario1 to generate a database schema, and you will map the Cargo Vocabulary in Scenario2 to an existing database schema that you will create later. The Rulesheets and Ruletests enable you to do some testing. By running the Ruletests, you can verify that the rules are able to access the database. The Ruleflow in Scenario2 enables you to perform the last task in this tutorial—configuring database settings in Corticon Server during deployment.

4

Scenario 1: Generate a database schema from an existing Vocabulary

Let's look at the scenario where you have an existing Vocabulary and you need to create a database schema based on the Vocabulary. You must perform the following steps:

- Create a database—while the database schema can be generated from the Vocabulary, the database must exist before the schema is generated.
- Set up entities that will be mapped to the database
- Define database connection properties.
- Create the database schema from the Vocabulary.

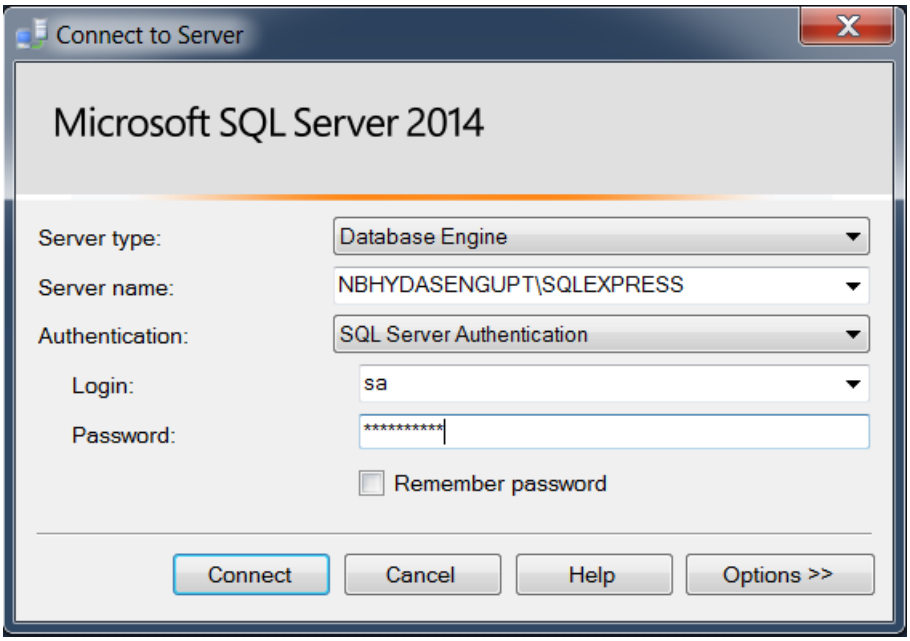
Create a database

Let's create a database named **Scenario1**.

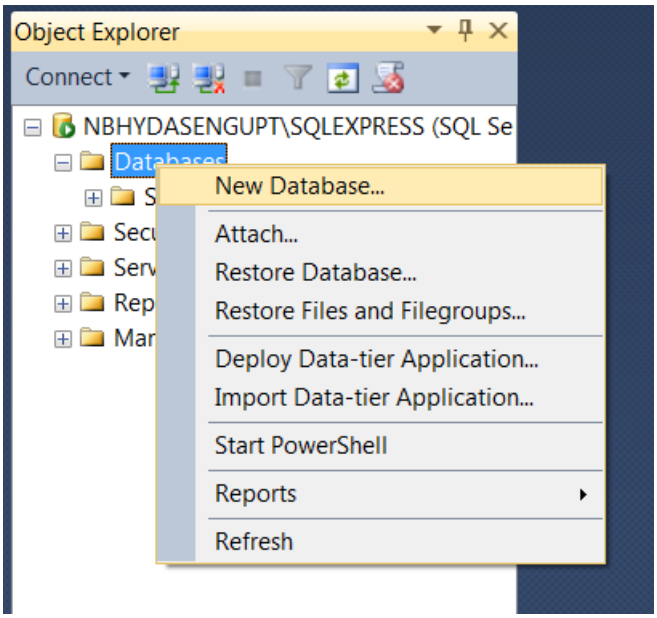
On the **Start** menu, choose **SQL Server Management Studio 20**.

In the Connect Object Explorer window:

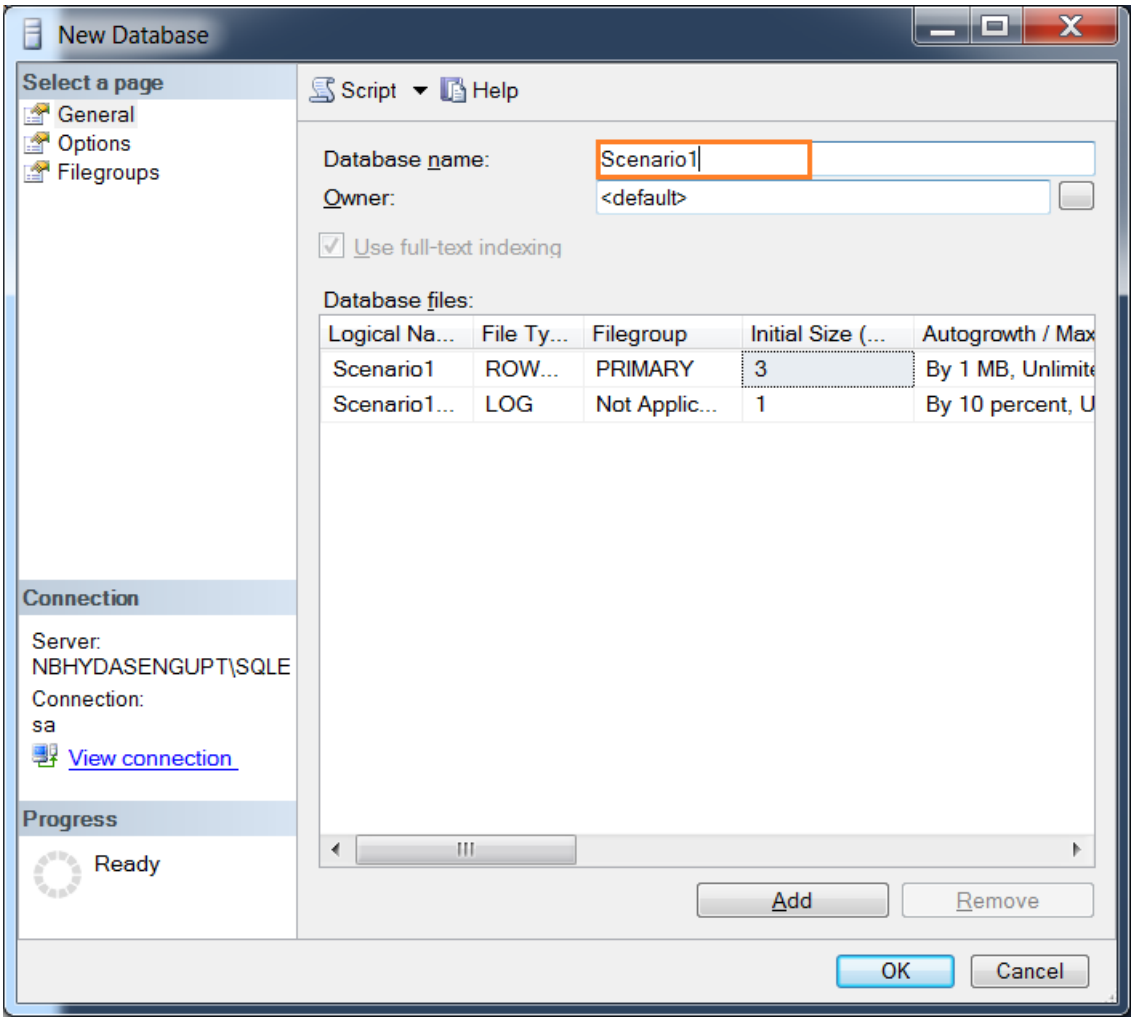
- Select SQL Server Authentication in the Authentication drop-down.
- Enter **sa** in the **Username** field.
- Enter **sqlserver2019** in the **Password** field.
- Click **Connect**.



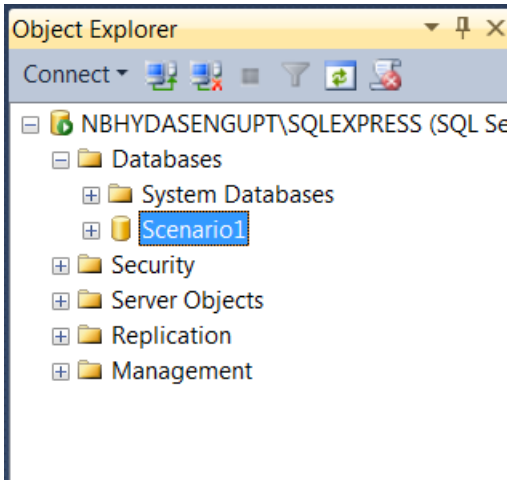
- Right-click Databases and select New Database.



- Enter Scenario1 in the Database name field and click OK.



The database Scenario1 gets created and is displayed in the Databases folder.

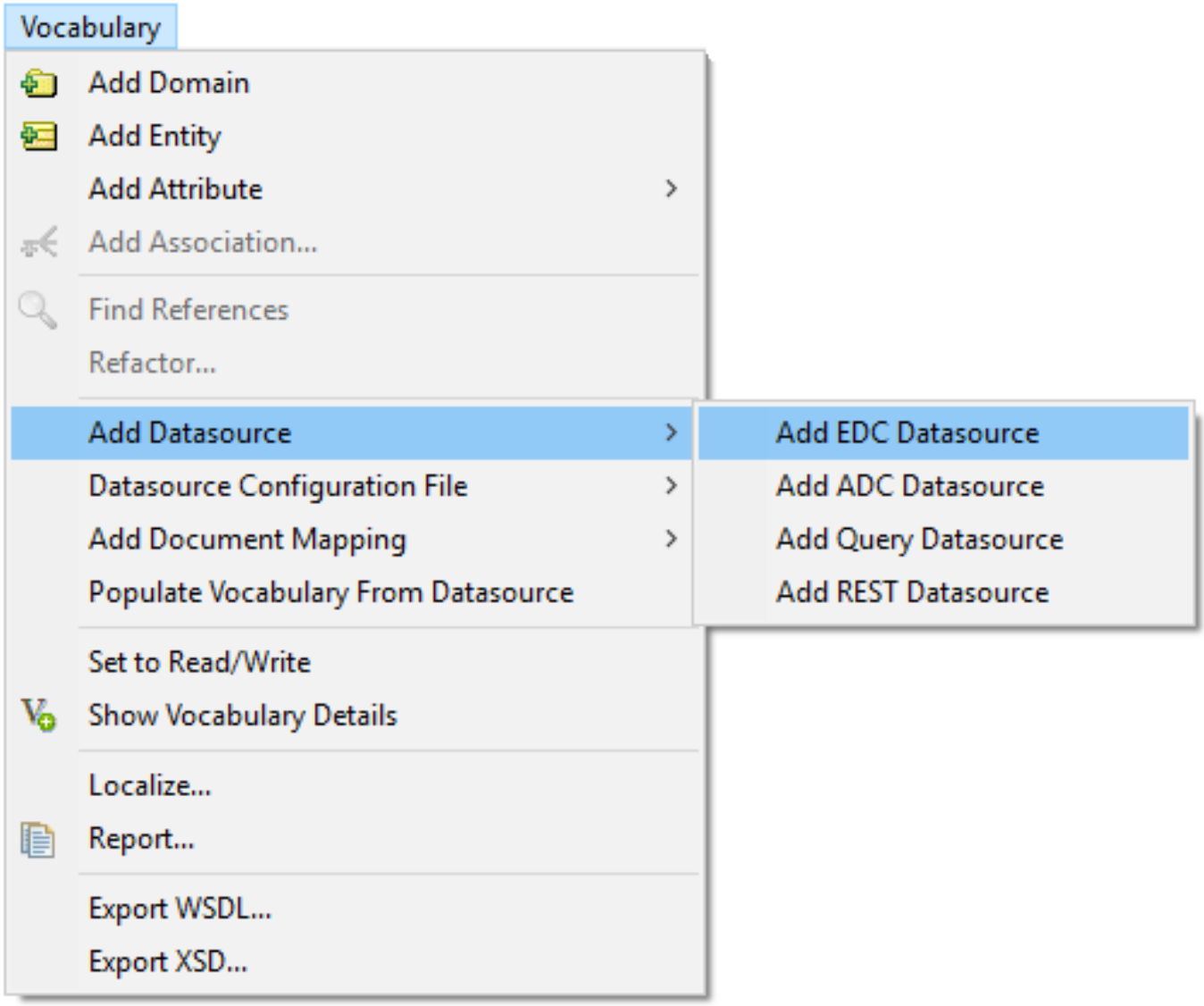


Add the EDC Datasource to the Vocabulary

First, we need to declare that we want to use an EDC Datasource.

In the Studio, open **Cargo.ecore** under **Scenario1** in the **Connecting_EDC** rule project.

Choose the Vocabulary menu command **Add Datasource > Add EDC Datasource**.



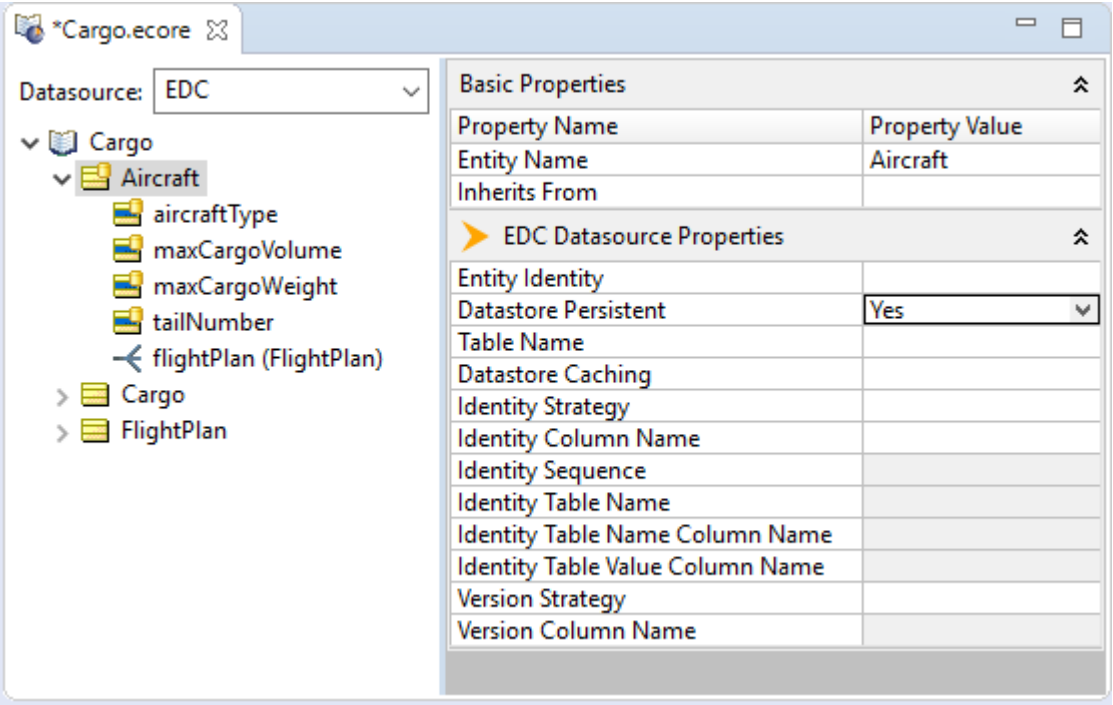
Set up entities for mapping

The next step is to set up the Vocabulary's entities for mapping. To do this, you configure each entity to persist to the database and choose a primary key for each entity.

EDC gives you a number of options for assigning primary keys. For SQL Server, you could choose an Identity strategy. For Oracle, you could choose a Sequence strategy. A more common option is to assign an attribute as the primary key for the entity. This makes sense if the attribute identifies the entity. In this tutorial, you will use this technique.

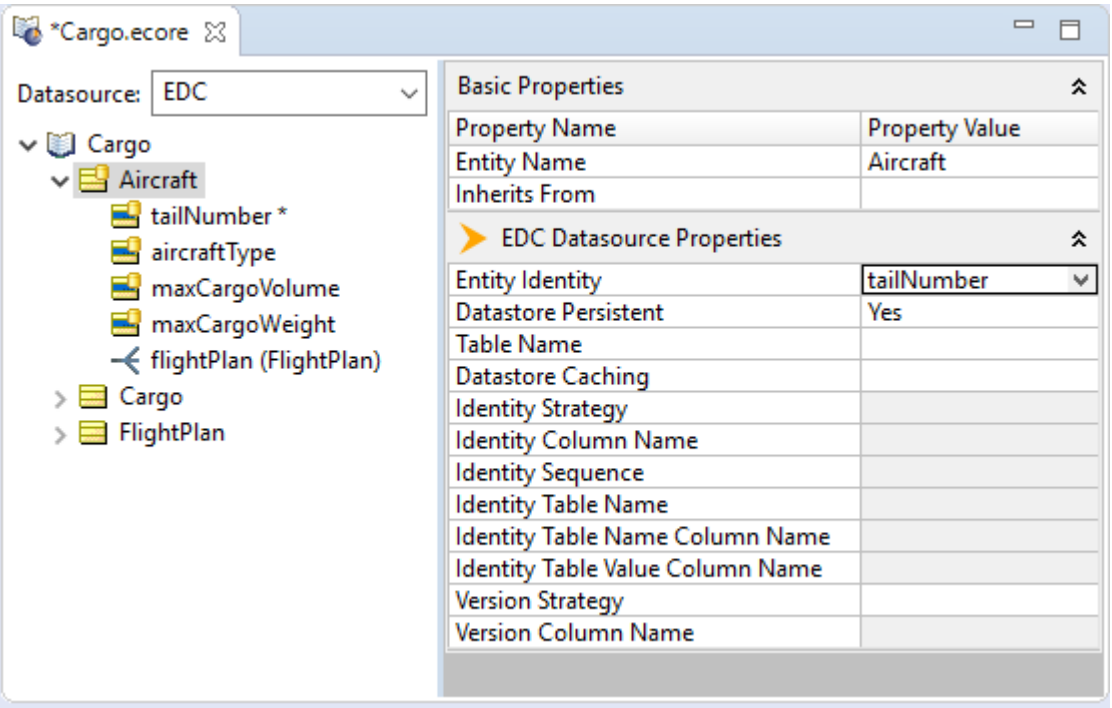
Select the **Aircraft** entity in the Vocabulary. In the **Properties** editor on the right, click the **Datastore Persistent** drop-down, and then select **Yes**.

On the **Datasource** pulldown, choose **EDC**. The icon of the Aircraft entity changes to include a database icon, indicating that the Aircraft entity is now configured for EDC and will be stored in the database together with its attributes and associations.



Let's assign **tailNumber** as the primary key for Aircraft.

Select the **Entity Identity** drop-down and select **tailNumber**.

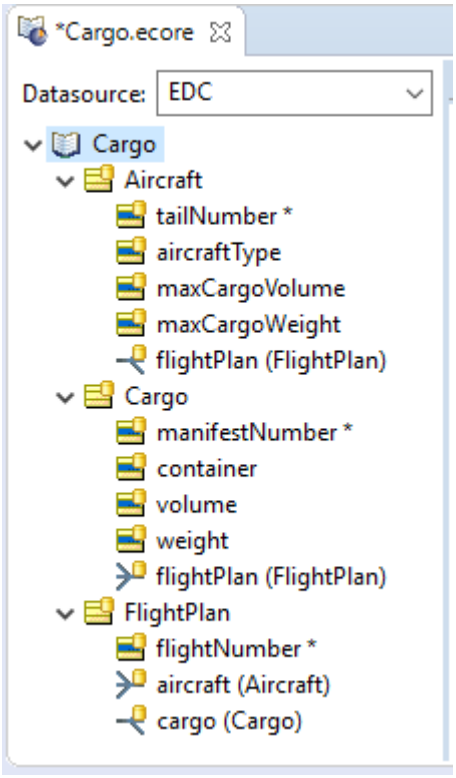


The tailNumber attribute in the Aircraft entity will now be mapped as its primary key, so it moves up to the top of the list of attributes in the entity, and is marked with an asterisk.

Configure the other two entities—Cargo and FlightPlan.

- For Cargo, choose manifestNumber as the Entity Identity.
- For FlightPlan, choose flightNumber as its Entity Identity.

Your Vocabulary will look like this:



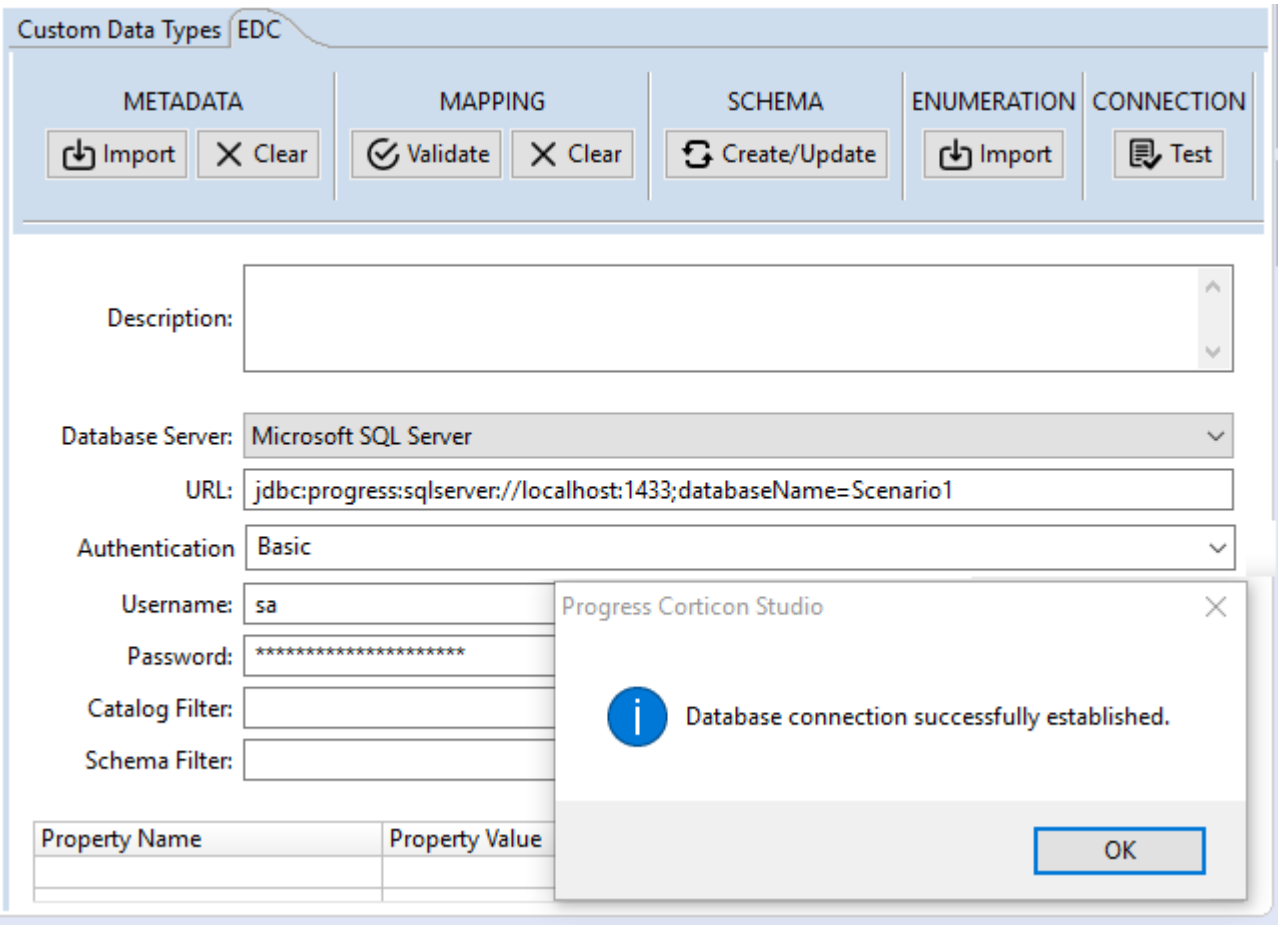
Define database connection properties in the Vocabulary

The next step is to define the database connection properties in the Vocabulary. This enables Corticon to connect to the database and generate the schema from the Vocabulary.

First, choose **File > Save** to store what we have done so far.

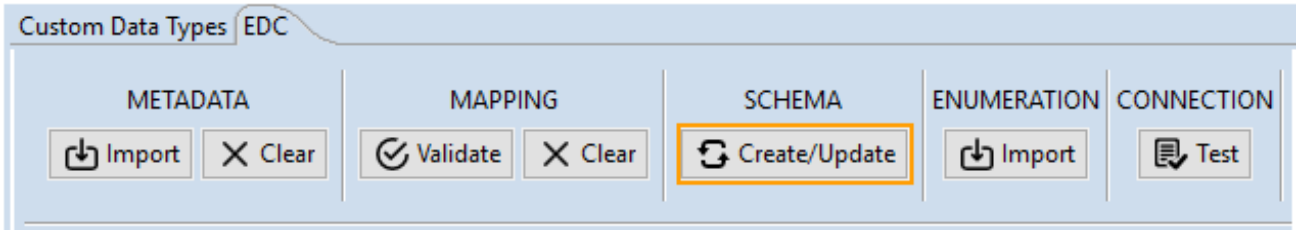
Now, we'll define database connection properties:

- Select the **Cargo** root node in the Vocabulary tree, and then click the **EDC** tab.
- Specify the following database properties:
  - Database Server: **Microsoft SQL Server 2014**
  - Database URL: **jdbc:progress:sqlserver://localhost:1433;databaseName=Scenario1**
  - Username: **sa**
  - Password: **sqlserver2014password**
  - Finally, let's test if the Vocabulary is able to connect to the database. Click CONNECTION **Test**. You see a message indicating that the connection was successful.

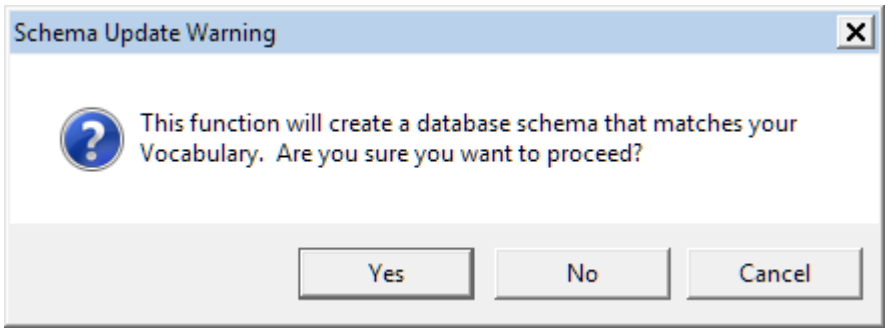


Create a database schema from the Vocabulary

Now that the Vocabulary is connected to the database, you can create a database schema from it. To do this, click Create/Update on the EDC tab.

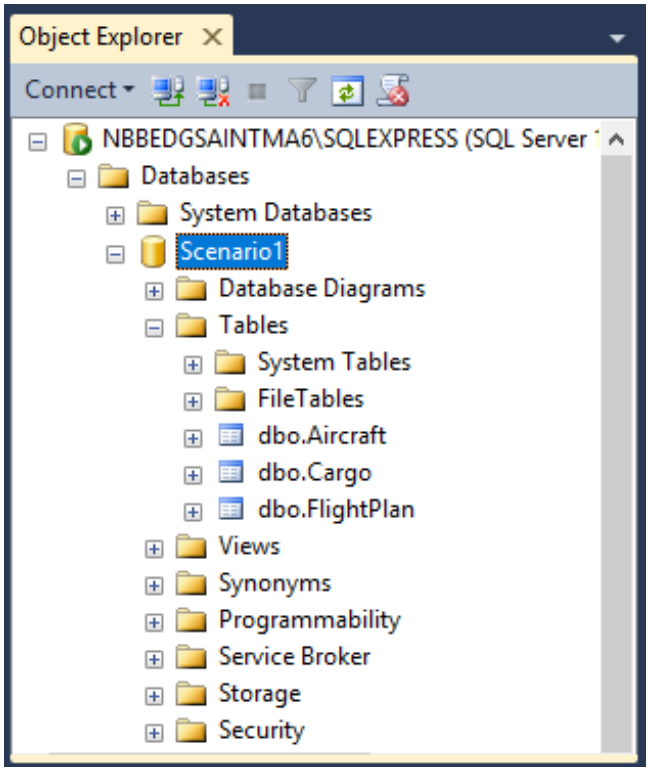


In the Schema Update Warning dialog box, click Yes.



Corticon Studio takes a few seconds to create the database schema; after which you get a message stating that the procedure completed successfully.

Let's verify that a table is created in the database for each entity in the Vocabulary. In SQL Server Management Studio, expand Databases > Scenario1 > Tables folder. You see the tables:



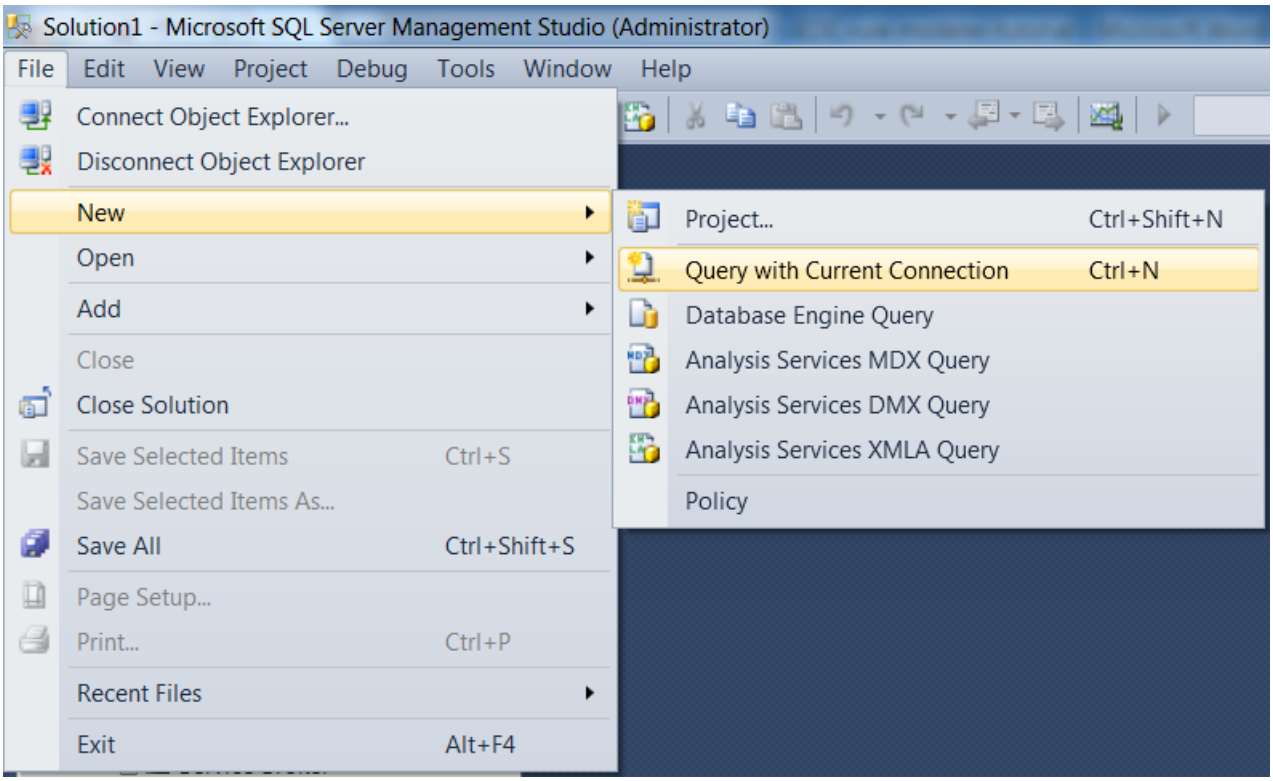
Set up test data

Now that the Vocabulary and database are connected and mapped, let's do some testing. Although this is typically performed by the rule modeler, doing it as part of this tutorial lets you verify that the rules are able to access the database. Let's populate the database with some sample data and then run the Ruletests.

Populating sample data

Follow these steps to populate sample data:

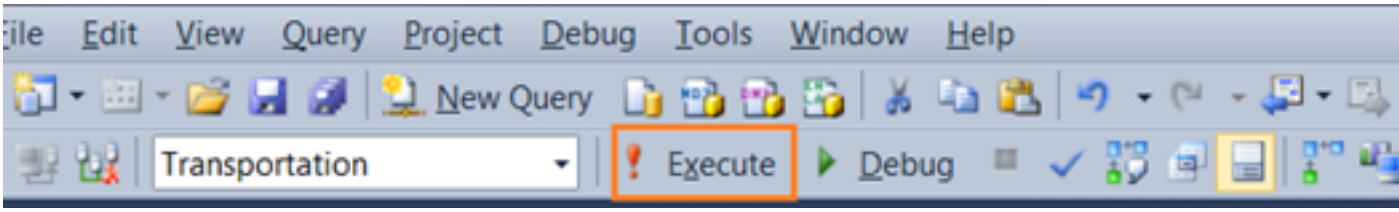
- In SQL Server Management Studio, select File > New > Query with Current Connection. A new SQL Script file opens in a text editor.



- Copy and paste these lines of code into the text editor:

```
INSERT INTO Scenario1.DBO.Aircraft
(aircraftType, maxCargoVolume, maxCargoWeight, tailNumber)
VALUES ('747', 7500, 150000, 'N111A');
INSERT INTO Scenario1.DBO.FlightPlan
(RaircraftAssoc_tailNumber, flightNumber)
VALUES ('N111A', 111);
INSERT INTO Scenario1.DBO.Cargo
(RflightPlanAssoc_flightNumber, container, manifestNumber, volume, weight)
VALUES (111, 'STANDARD', '625A', 3000, 100000);
INSERT INTO Scenario1.DBO.Cargo
(RflightPlanAssoc_flightNumber, container, manifestNumber, volume, weight)
VALUES (111, 'HEAVY', '625B', 5000, 150000);
```

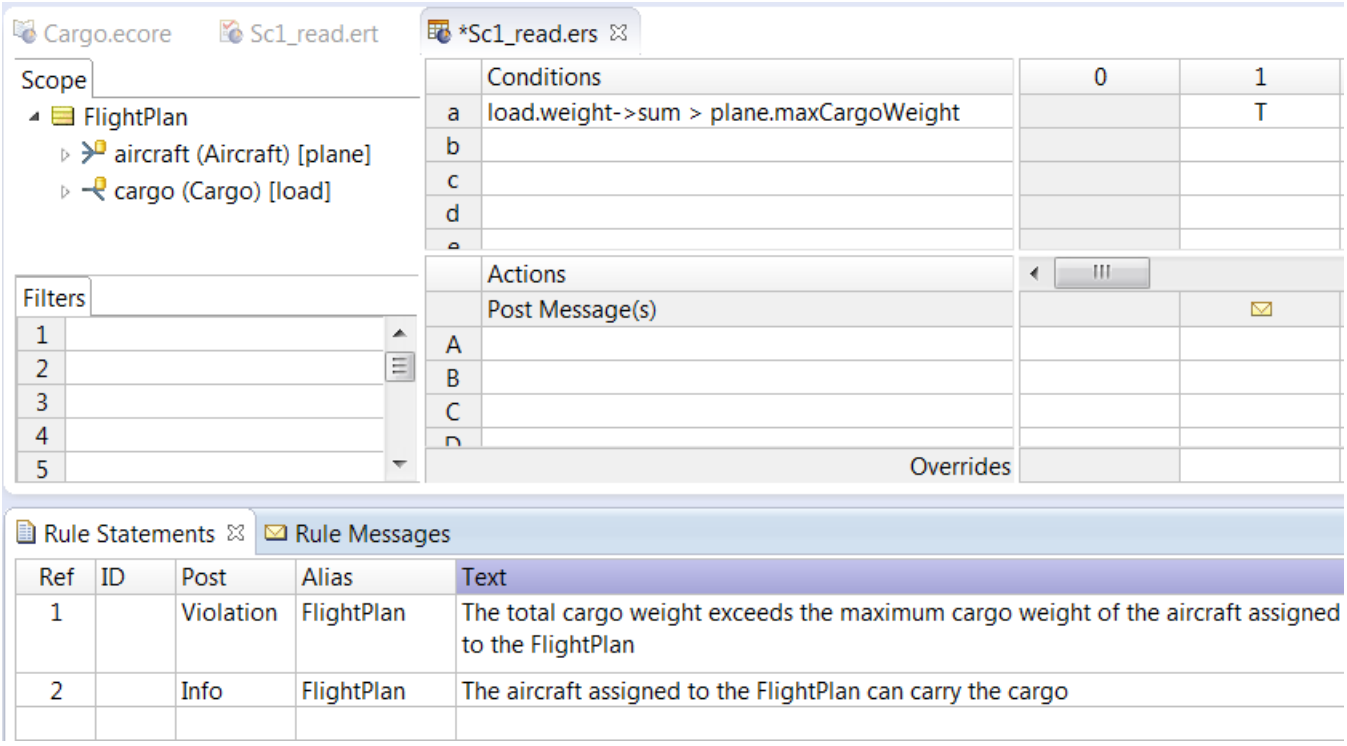
- Click **Execute**.



You see messages indicating that rows have been added to the tables.

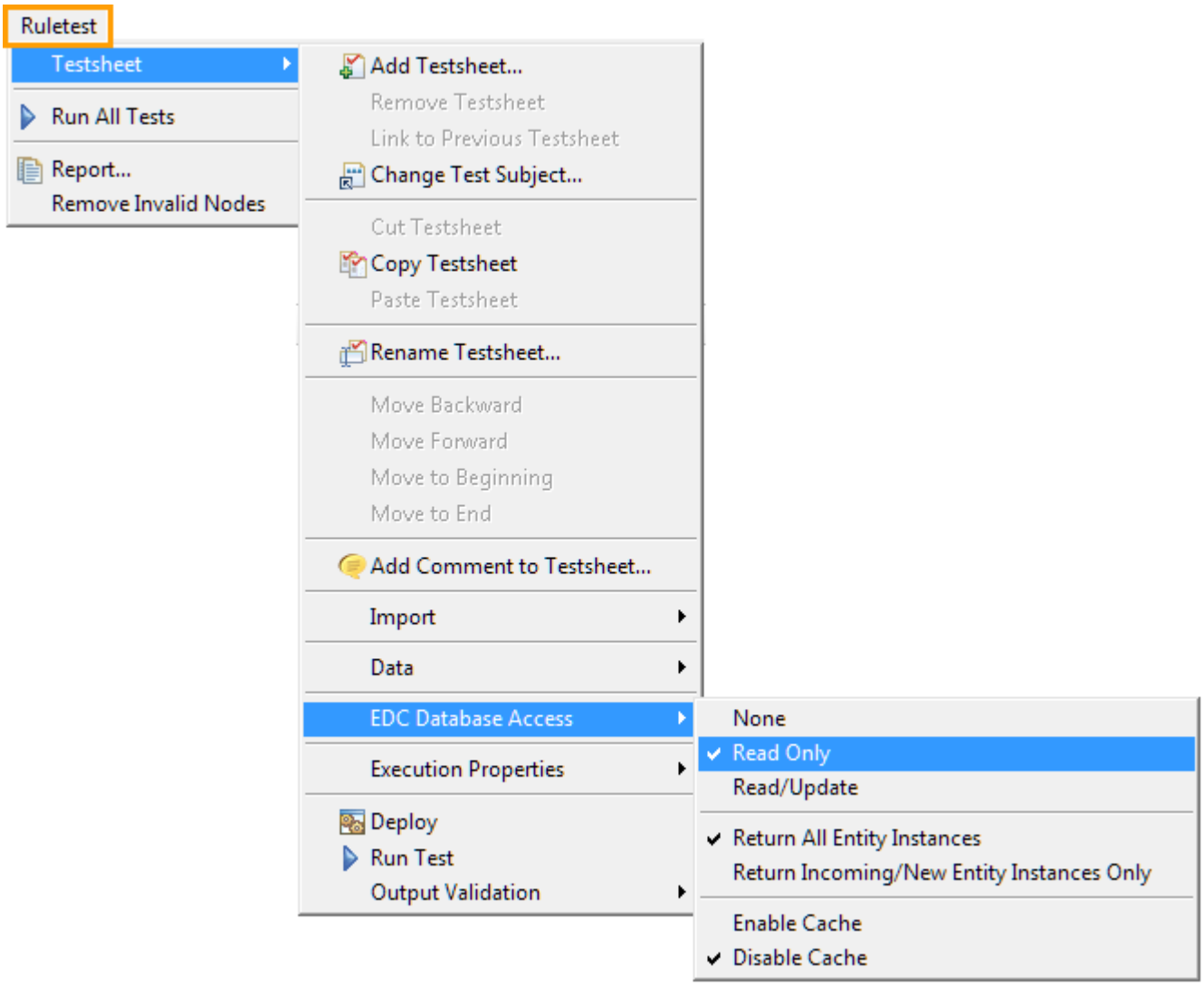
Run the Ruletest

Open the Rulesheet Sc1_read.ers in the Scenario1 subfolder in the sample Rule Project. The rules in this Rulesheet check if the total cargo weight assigned to a FlightPlan is greater than the maximum cargo weight of the aircraft assigned to the same FlightPlan.

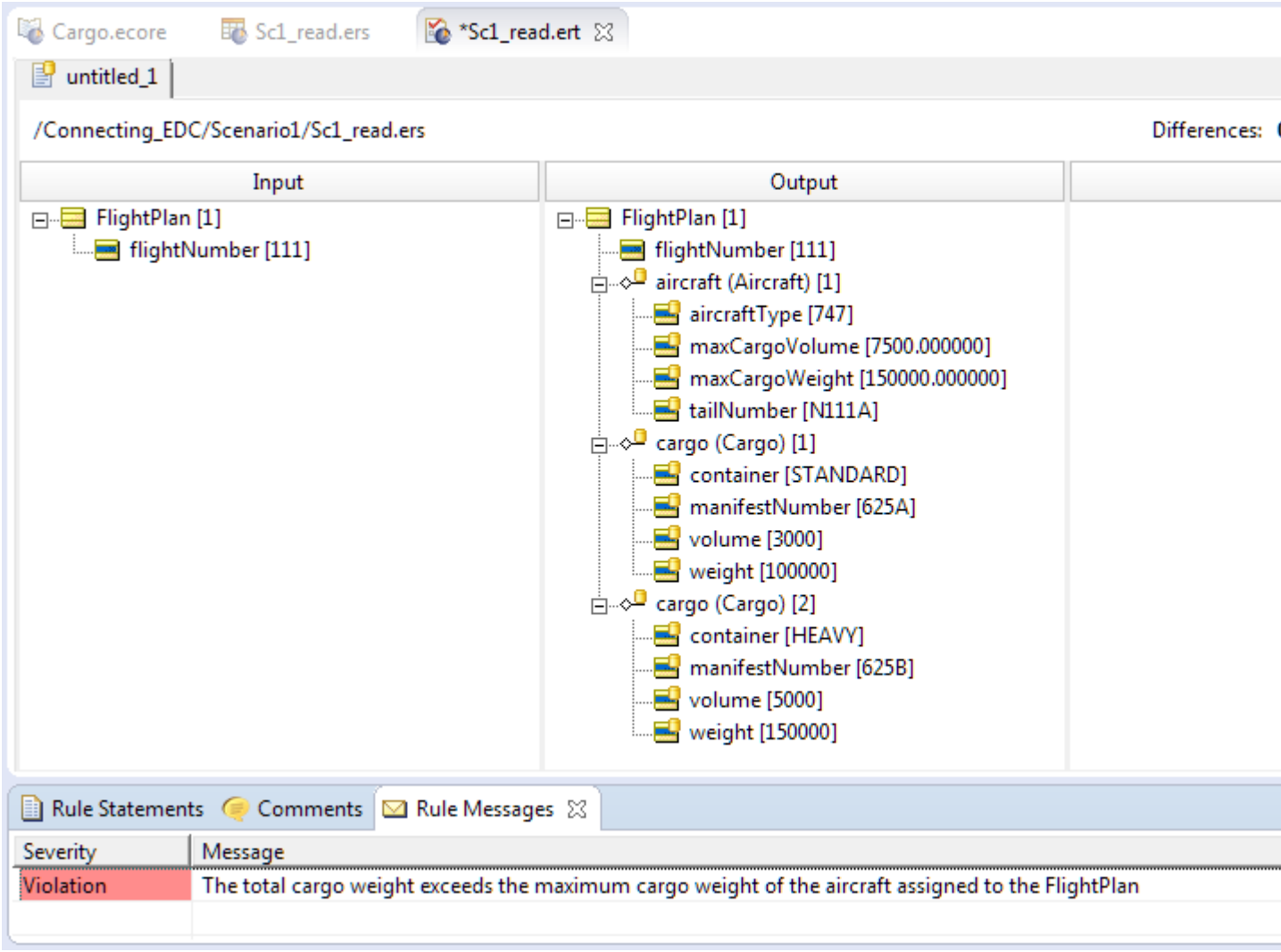


In the previous step, you populated the Scenario1 database with data about Cargo, FlightPlan, and Aircraft. Running the Ruletest will verify that the rule can read those records from the database.

Open the Ruletest named Sc1_read.ert and verify that its database access setting is set to Read Only. Select Ruletest > Testsheet > Database Access and verify that the Read Only option has a checkmark next to it.



Run the test. You see the following results:

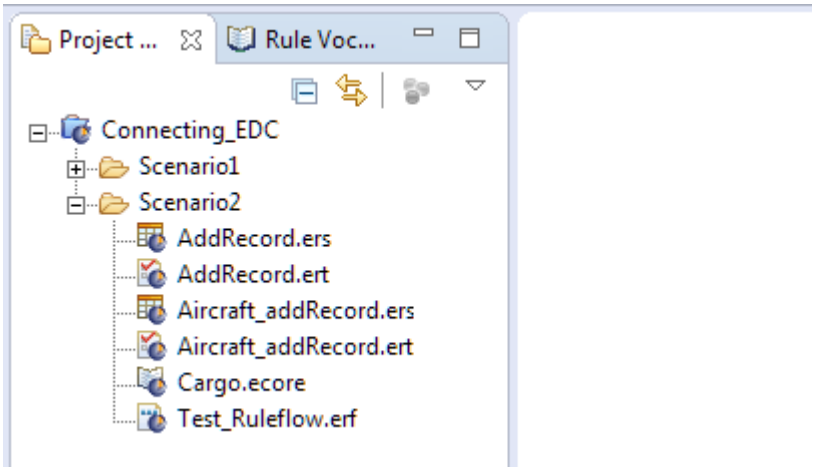


As you can see, based on the flightNumber in input, Corticon retrieved the FlightPlan record and the records of all the instances associated with it, using the record's foreign keys. This data was then processed by the rule and the result displayed in the Ruletest's Output pane.

That completes our look at Scenario1. Let's cleanup to prepare for Scenario2:

Choose File > Save All, and then choose File > Close All

Collapse the Scenario1 folder, and then expand the Scenario2 folder



5

Scenario 2: Mapping an existing Vocabulary with an existing database schema

In this scenario, you have an existing Vocabulary and an existing database and you need to map them. As part of mapping them, you configure the Vocabulary to connect to the database and import the database's metadata. During import, if Corticon finds an element in the database that has a corresponding element with a matching name in the Vocabulary, it automatically maps those elements. For example, if the Vocabulary has a FlightPlan entity and the database has a FlightPlan table, Corticon automatically maps them. You only have to manually map elements that have different names.

In this tutorial's example, you will create a database named Scenario2 that has a schema similar to the Cargo Vocabulary in the Scenario2 folder in the Connecting_EDC rule project. The Cargo Vocabulary and Scenario2 database must be mapped as follows:

Cargo Vocabulary	Scenario2 Database
Entity: Cargo	Table: Shipment
Container	container
manifestNumber*	manifestNumber*
Volume	volume
Weight	weight
Entity: Aircraft	Table: Plane
aircraftType	planeType
maxCargoVolume	maxCargoVolume
maxCargoWeight	maxCargoWeight
tailNumber*	tailNumber*
Entity: FlightPlan	Table: FlightPlan
flightNumber*	flightNumber*

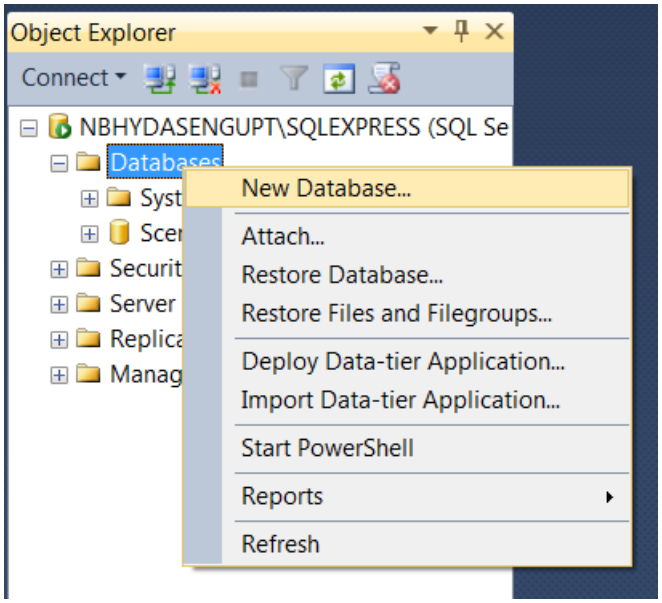
* Primary key

As you can see, most names match, but some elements in the Vocabulary—such as the Cargo entity (Shipment in the database), Aircraft entity (Plane in the database), and the aircraftType attribute (planeType in the database) have differently-named corresponding elements in the database. You need to map these elements manually.

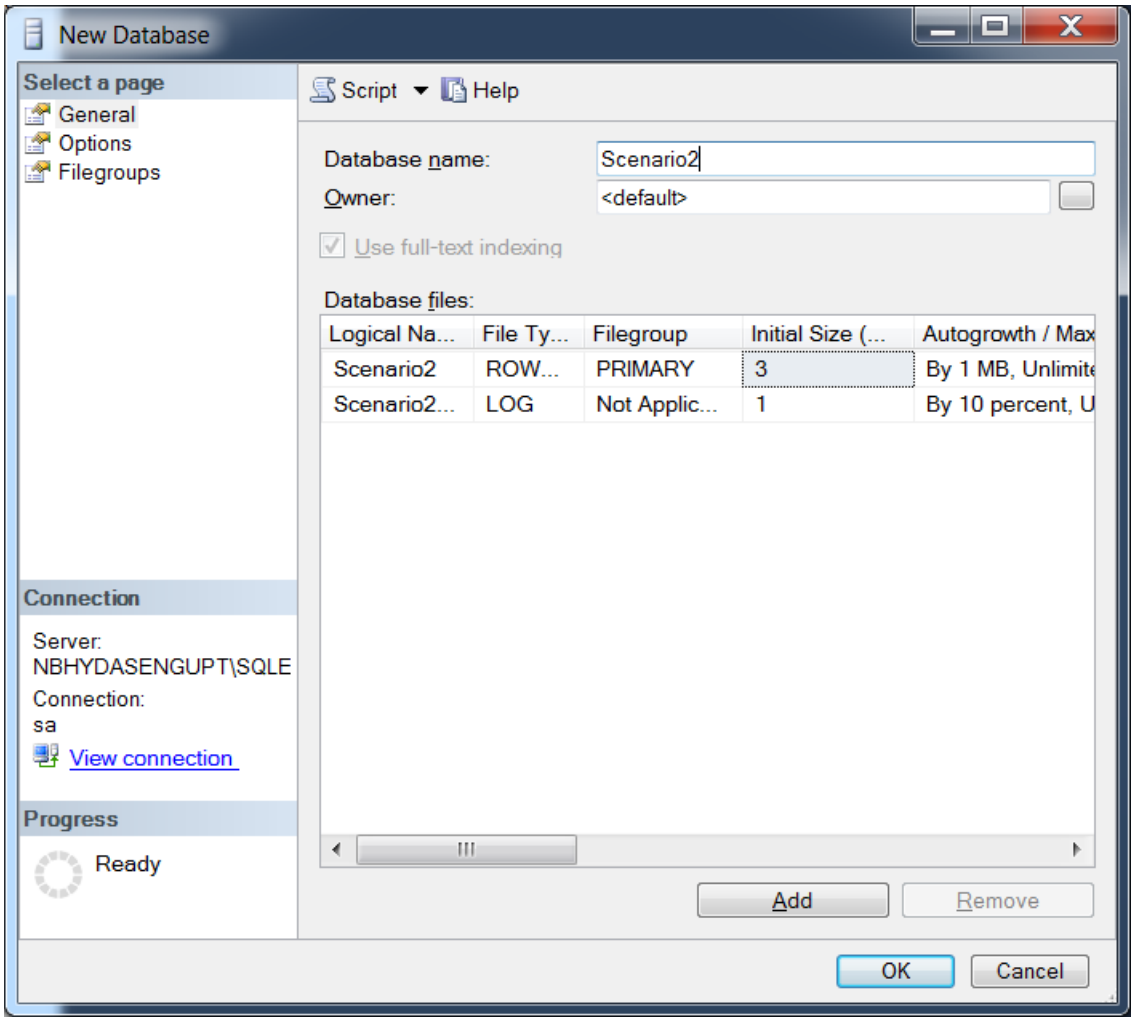
Creating a new database

Let's start by creating a new database and tables that can be mapped to the Cargo.ecore Vocabulary.

In SQL Server Management Studio, right-click Databases and select New Database.

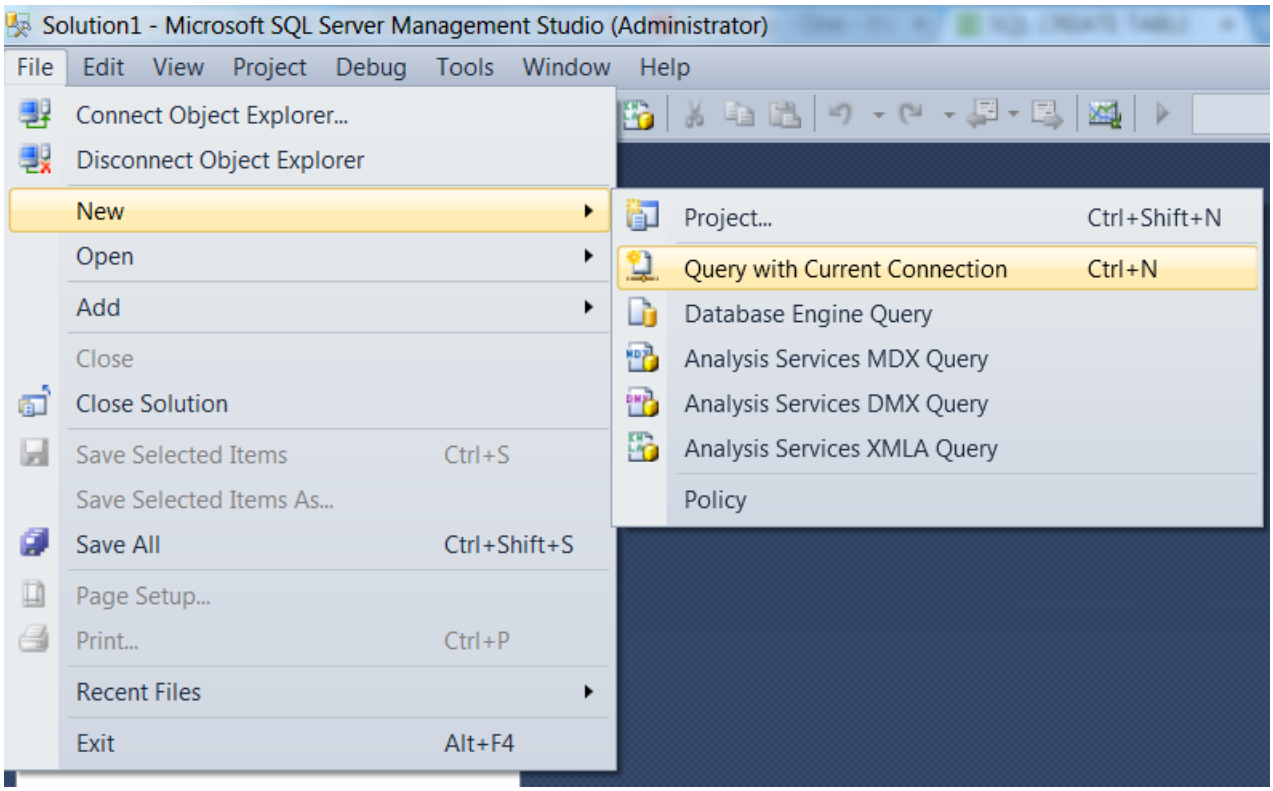


In the New Database window, enter Scenario2 as the Database name and click OK.



Next, let's create tables in the database. We will later map these tables and their columns to entities and attributes in the Cargo.ecore Vocabulary.

In SQL Server Management Studio, select File > New > Query with Current Connection.



In the SQL script editor, copy and paste the following code:

```
CREATE TABLE Scenario2.dbo.Plane
(planeType varchar(50),
maxCargoVolume decimal,
maxCargoWeight decimal,
tailNumber varchar(50) NOT NULL PRIMARY KEY);

CREATE TABLE Scenario2.dbo.FlightPlan
(flightNumber int NOT NULL PRIMARY KEY,
tailNumber varchar(50) FOREIGN KEY REFERENCES Scenario2.dbo.Plane(tailNumber));

CREATE TABLE Scenario2.dbo.Shipment
(container varchar(50),
manifestNumber varchar(50) NOT NULL PRIMARY KEY,
volume decimal,
weight decimal,
flightNumber int FOREIGN KEY REFERENCES Scenario2.dbo.FlightPlan(flightNumber));
```

Click Execute.

You see a message indicating that the command executed successfully.

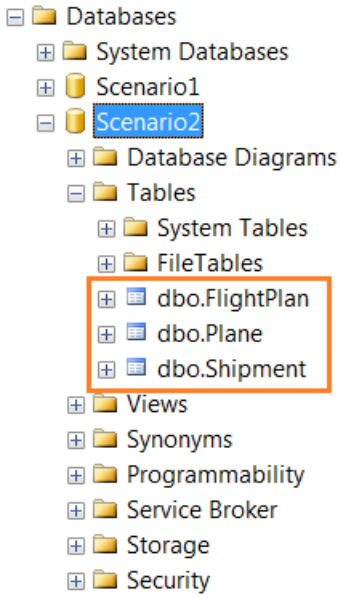
```
manifestID varchar(50) NOT NULL PRIMA
sVolume decimal,
sWeight decimal,
flightNumber int FOREIGN KEY REFERENC
);
```

100 %

Messages

```
Command(s) completed successfully.
```

Expand **Databases > Scenario2 > Tables** in the **Object Explorer** view on the left. You see its three tables—dbo.FlightPlan, dbo.Plane, and dbo.Shipment:

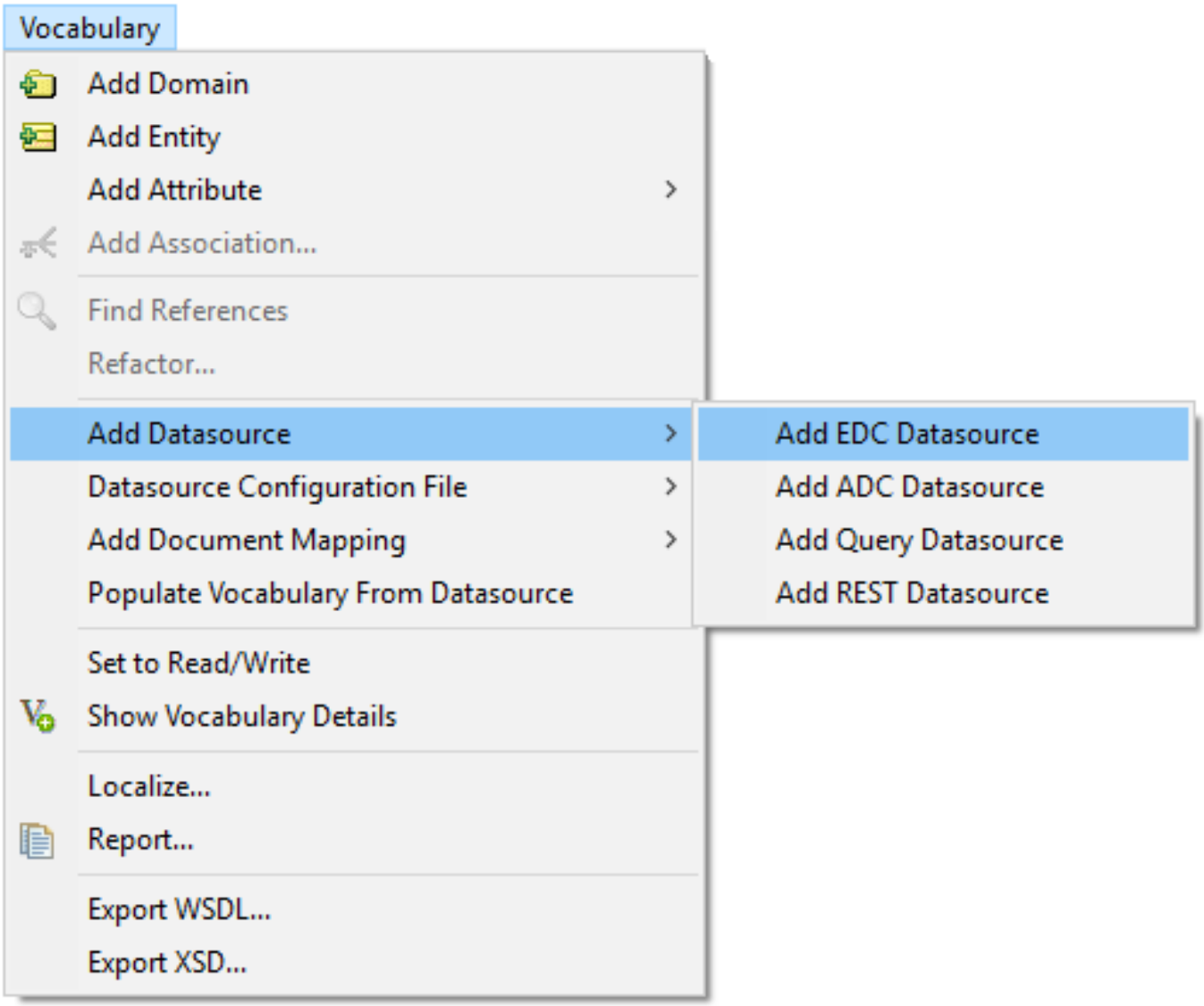


Adding the EDC Datasource to the Vocabulary

As in Scenario 1, we need to declare that we want to use an EDC Datasource.

In the Studio, open Cargo.ecore under Scenario2 in the Connecting_EDC rule project.

Choose the Vocabulary menu command Add Datasource > Add EDC



Mapping the Vocabulary to the database

The next step is to map the Vocabulary to the database. This step consists of six substeps:

- Defining database connection properties
- Setting up entities for mapping
- Importing database metadata
- Manually mapping differently-named entity with tables
- Manually mapping differently-named attributes with columns
- Validating mappings

Defining database connection properties

On the EDC tab, specify the following database connection properties:

- Database Server: Microsoft SQL Server

- Database URL: jdbc:progress:sqlserver://localhost:1433;databaseName=Scenario2
- Username: sa
- Password: sqlserver2014password

Click CONNECTION Test. You see a message indicating that connection was successfully established.

Setting up entities for mapping

As you did in Scenario 1, configure each entity to persist to the database by setting its Datastore Persistent property to Yes. Configure the Entity Identity property for each entity as follows:

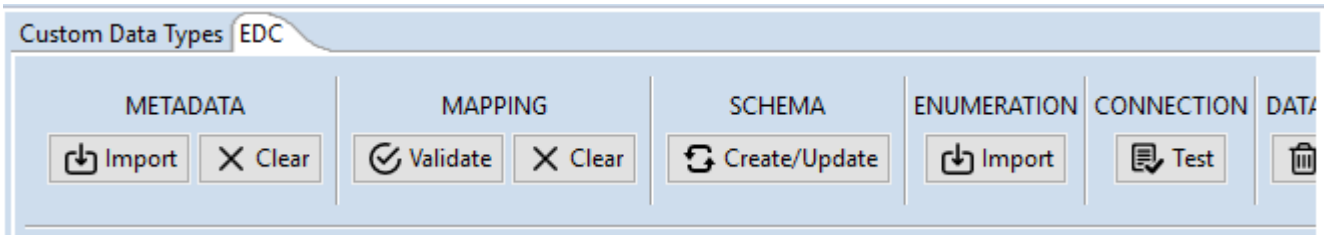
- For Aircraft, choose tailNumber:
- For Cargo, choose manifestNumber as its entity identity.
- For FlightPlan, choose flightNumber as its entity identity.

The icons of the entities change to include a database icon, indicating that they are now configured to be stored in a database together with their attributes and associations.

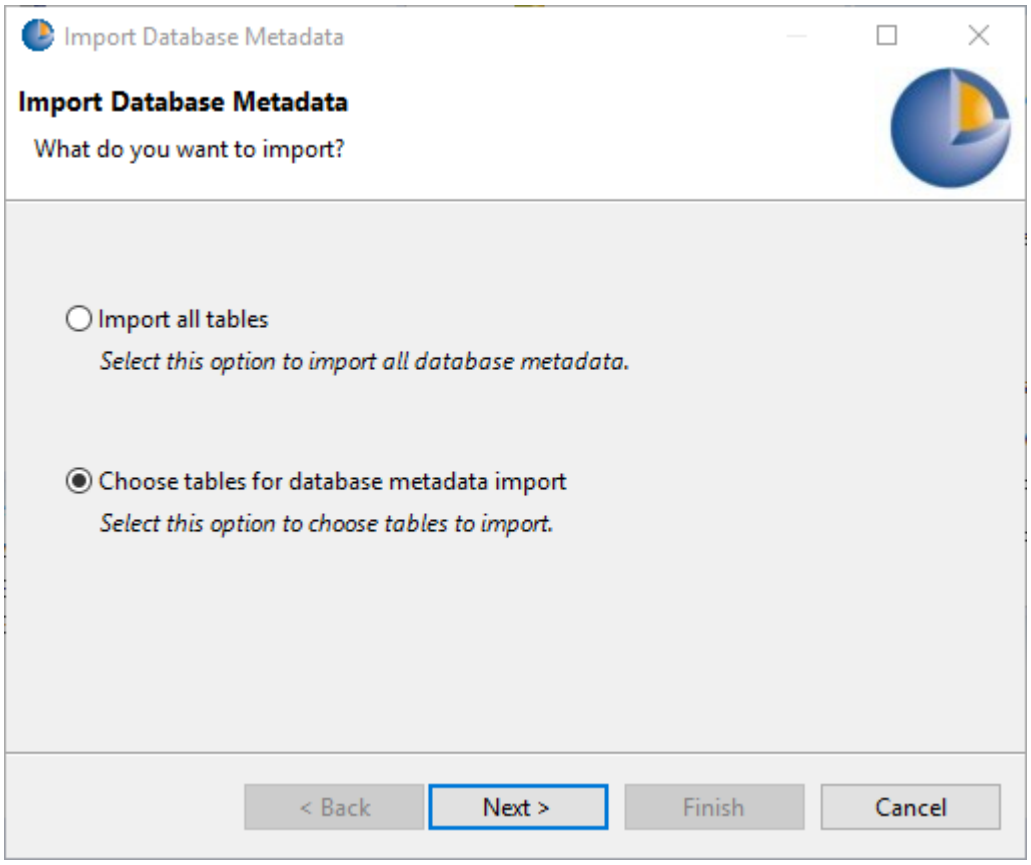
Importing database metadata

After you have configured the entities for database persistence and primary key, you can import the database's metadata. Doing this automatically maps matching entity and table names, and if an entity has been mapped to a table, it also automatically maps matching attribute and column names. Any differently-named entities and attributes will have to be mapped manually.

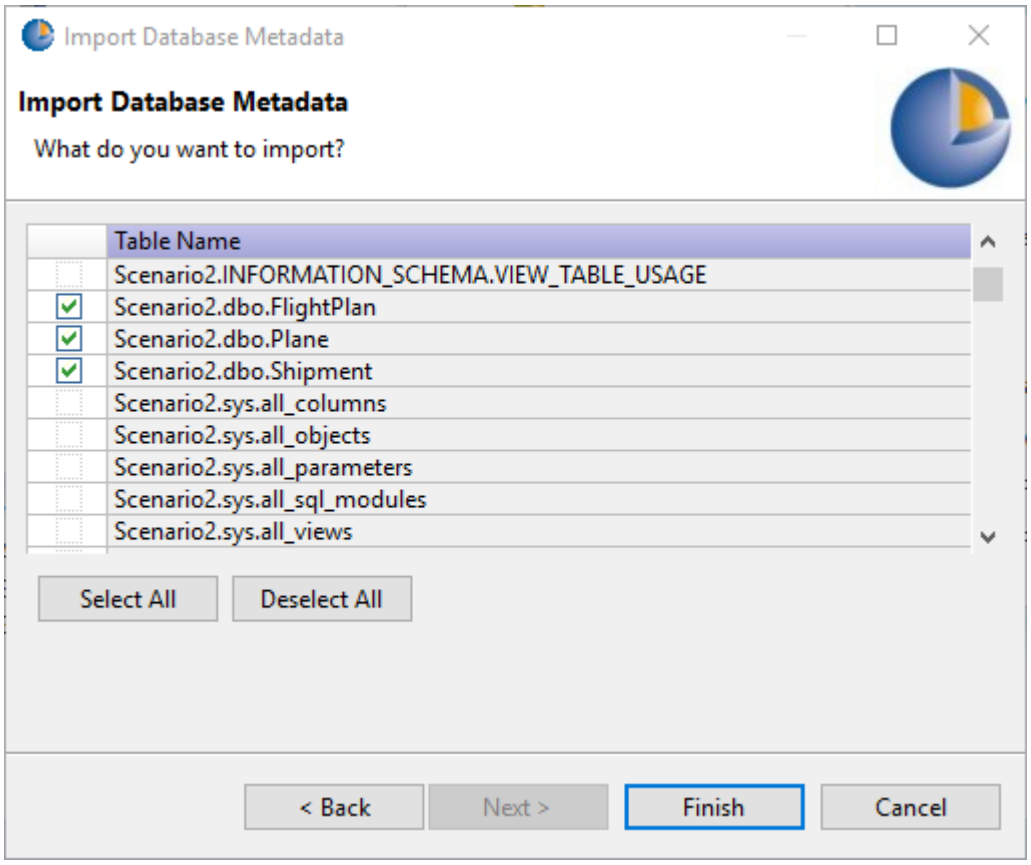
To import database metadata, click the EDC tab METADATA Import.



This opens the Import Database Metadata dialog box. In this dialog box, select Choose tables for database metadata import and click Next.

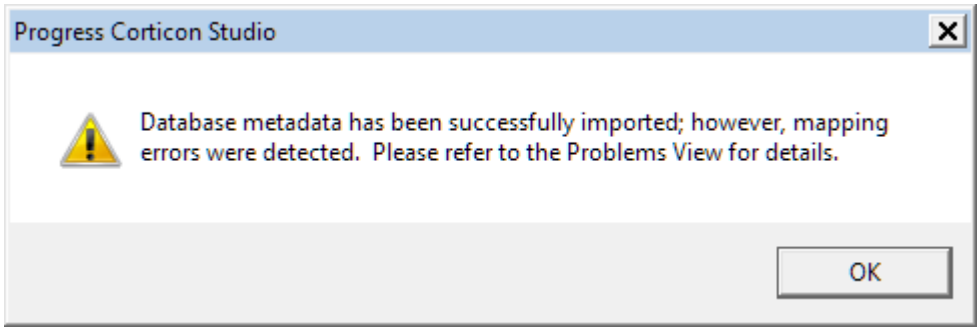


Next, you choose which tables to import. Click Deselect All and then scroll through the list of tables to locate dbo.FlightPlan, dbo.Plane, and dbo.Shipment. Select these tables and click Finish.

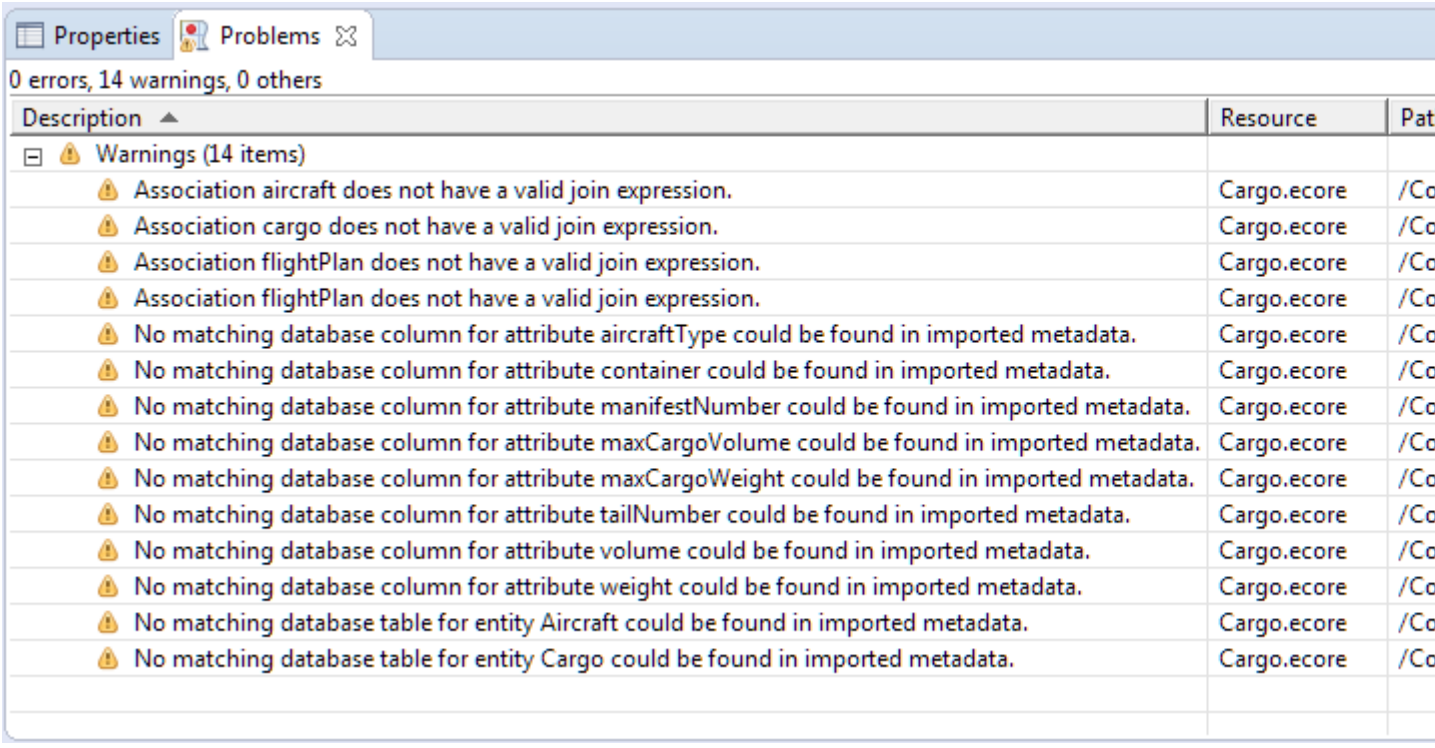


If all entity and attribute names have corresponding elements in the database with matching names, you would see a message indicating that the database metadata has been imported successfully.

However, if there are any differently-named entities (as in our example), the entity and its attributes will not get mapped automatically. You will see the following message:



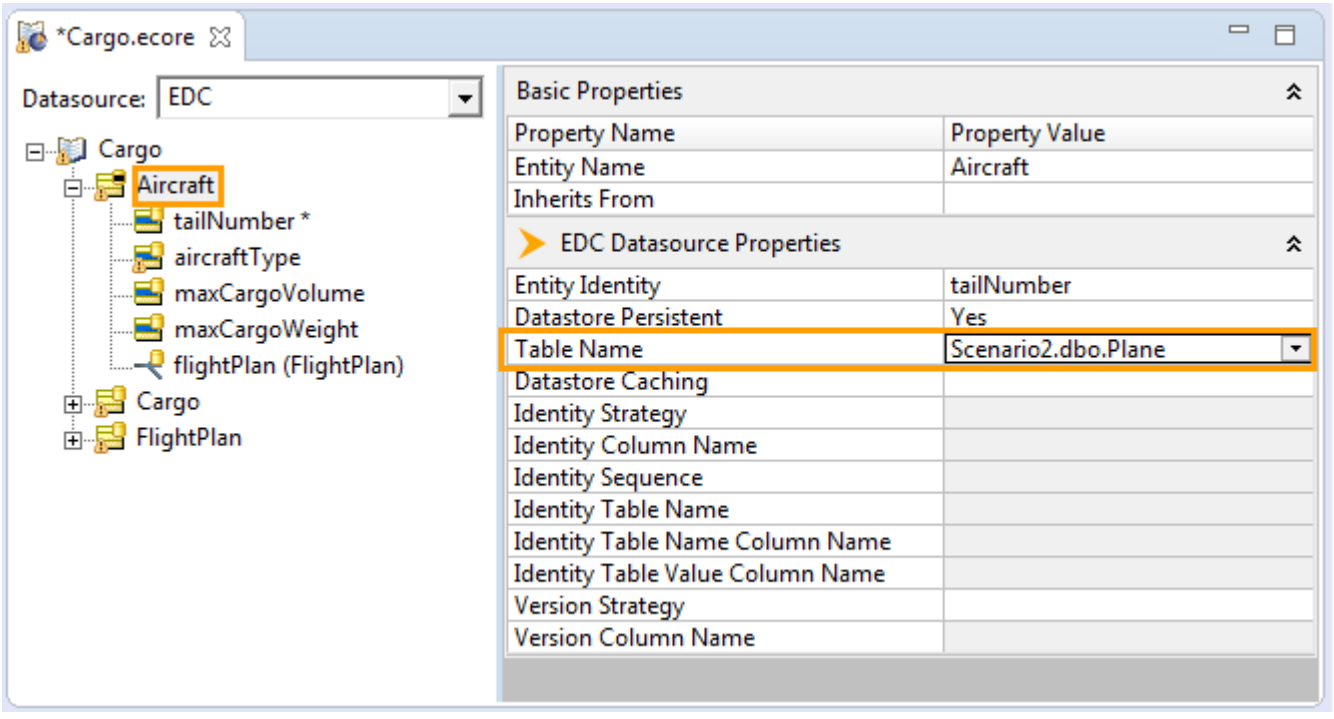
The Problems view shows which elements were not mapped automatically:



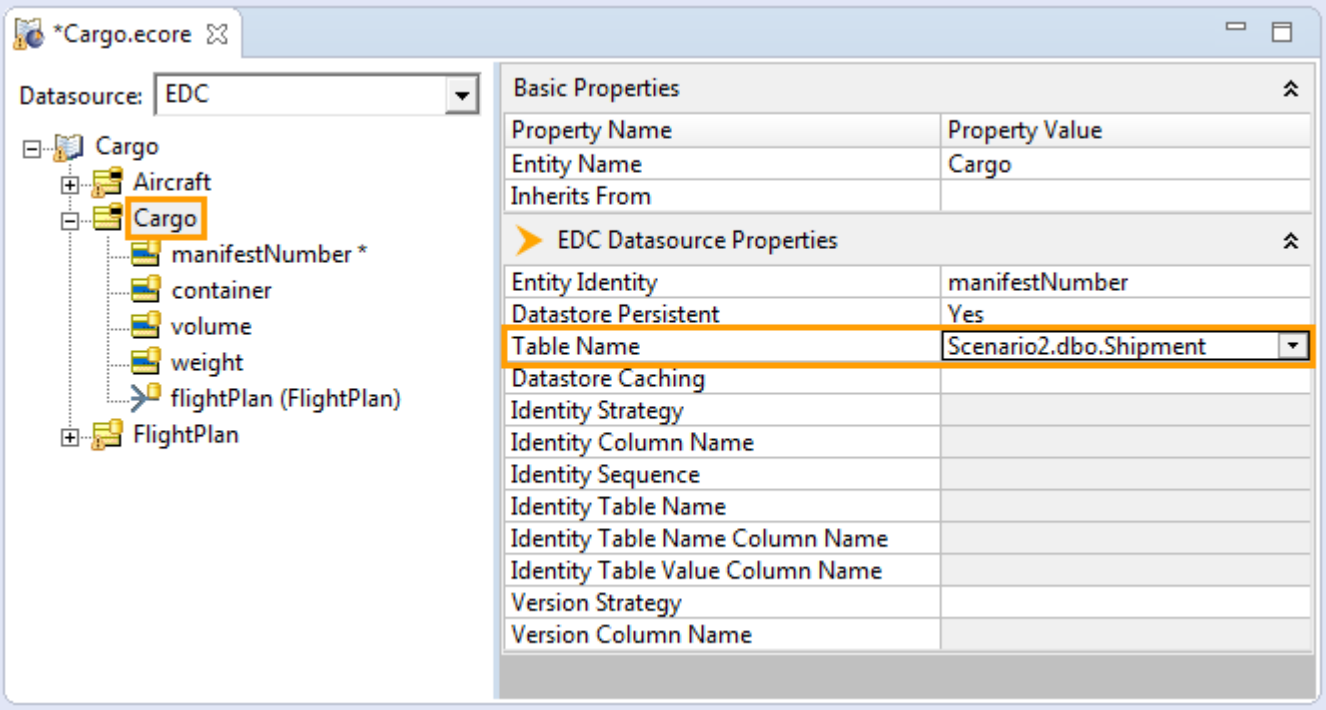
Mapping differently-named entities

Looking at the Problems from bottom to top, you see that no matching database table for two of the entities could be found in the imported metadata. A lot of the other problems derive from those problems, so we will map the mismatched entities manually.

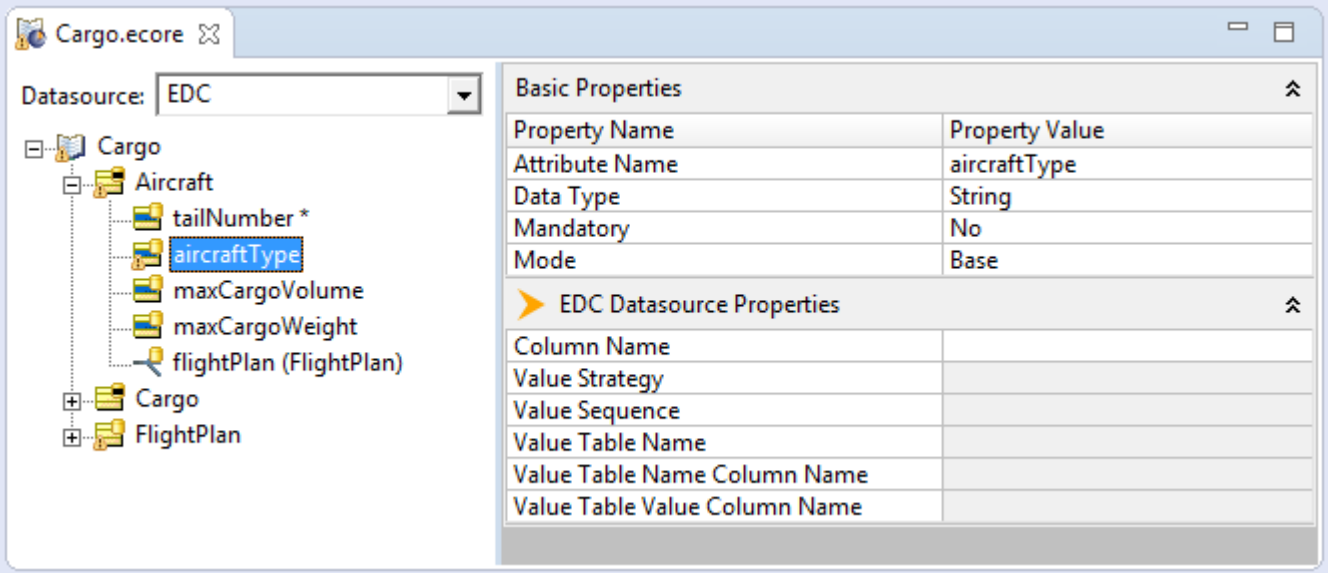
Let's start with the Aircraft entity. Select the **Aircraft** entity and click on the **Table Name** drop-down. In the list of Scenario2 tables, select **Scenario2.dbo.Plane**:



Similarly, click Cargo and then click its Table Name drop-down. Select Scenario2.dbo.Shipment:



Before we correct the one remaining problem, let's take a closer look at the way that Vocabulary "decorations" provide information:



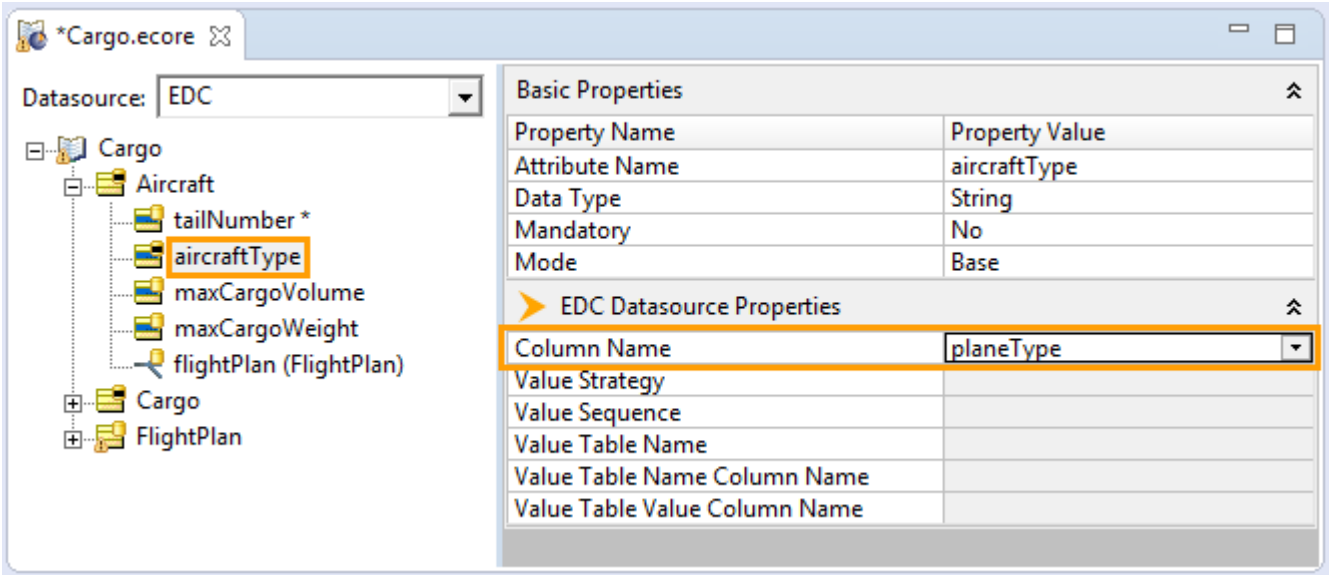
- All the entities and attributes have a database symbol in their upper right.
- The two entities we manually mapped have a black stripe in their database symbol.
- The attribute aircraftType has an alert marker in its lower left.

Save the Vocabulary file so that the problems are re-evaluated.

Mapping differently-named attributes

All attributes that have a corresponding column with the same name get mapped automatically. However, any attribute that does not have a corresponding column with the same name will not get mapped.

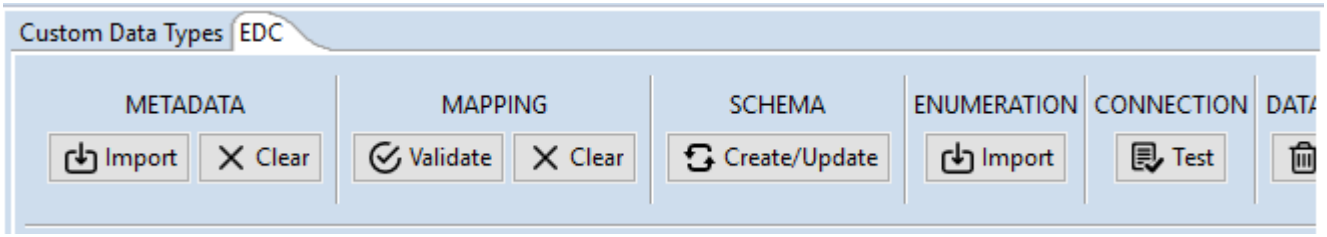
Click aircraftType and then click on the Column Name property. In the list of choices, select planeType:



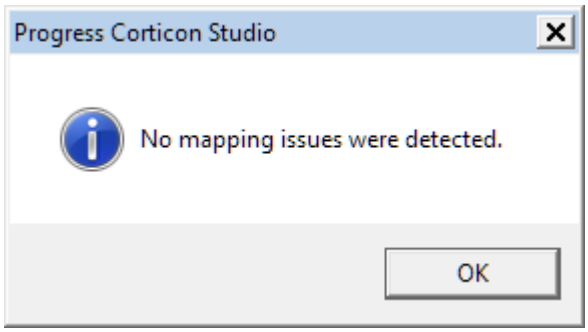
All elements in the Vocabulary are now mapped to the appropriate database element. Save the Vocabulary to show that the Problems view has no entries. Let's verify this by validating the mappings.

Validating mappings

To validate mappings, click the EDC tab MAPPING Validate:



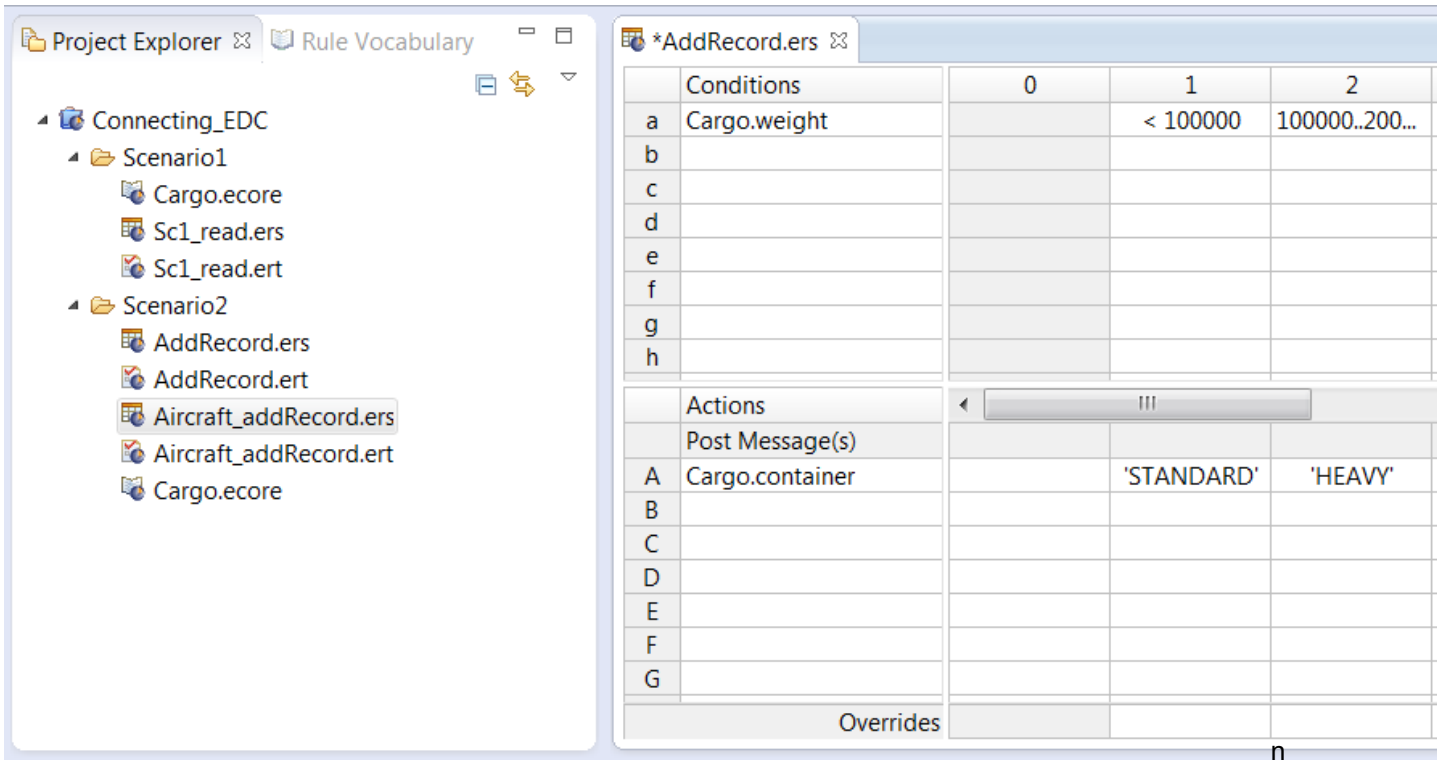
A message indicates that no mapping errors were found:



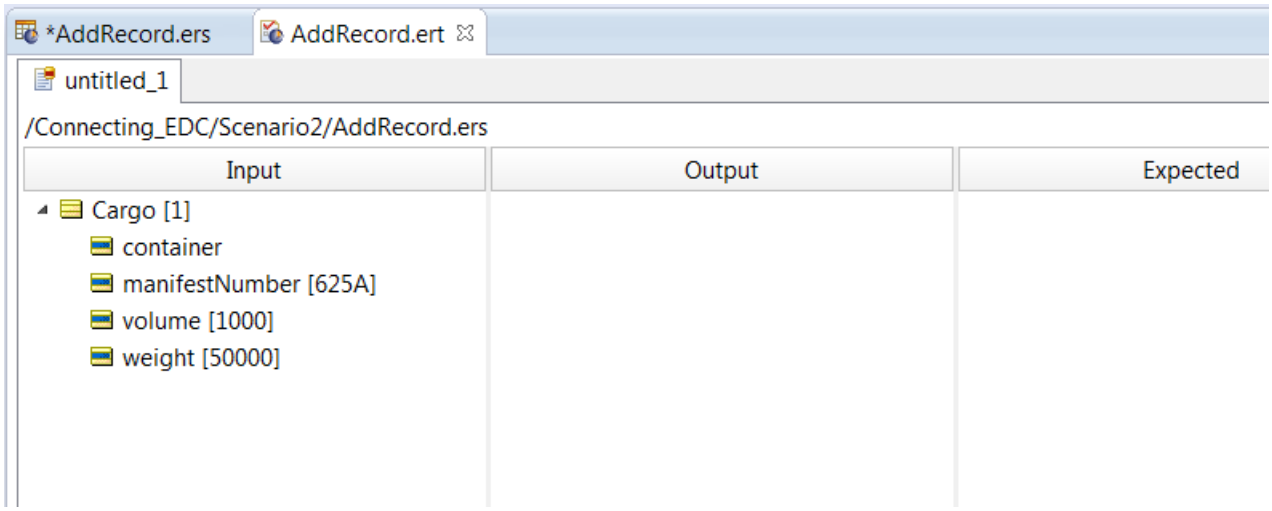
Testing

Let's verify that Corticon rules that use the Cargo.ecore Vocabulary are able to access the Scenario2 database.

Open AddRecord.ers in the Scenario2 folder. The Rulesheet assigns a value to Cargo.container based on the value of Cargo.weight received in input:

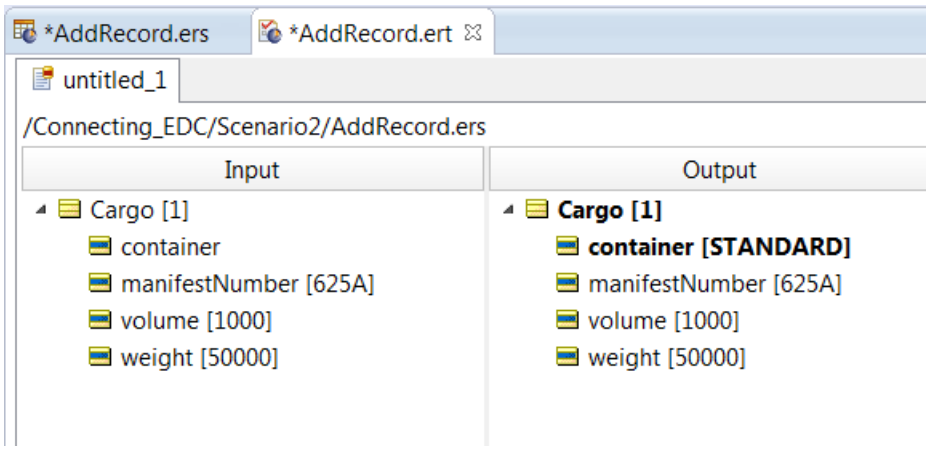


AddRecord.ert in the Scenario2 folder. The Ruletest has one Cargo entity instance in input:

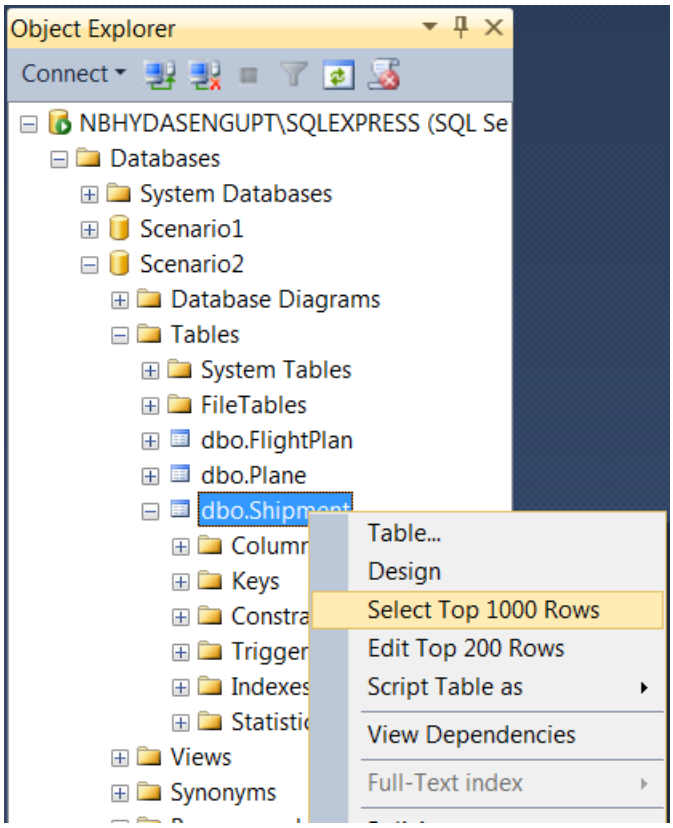


For a Ruletest to perform read/write operations on the database, it must be configured to do so. Verify that the Ruletest's database access setting is set to Read/Update by selecting Ruletest > Testsheet > EDC Database Access. There is a checkmark next to Read/Update.

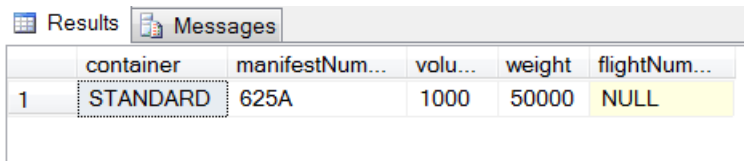
Run the Ruletest. You see the following output:



The value of the container attribute has changed, indicating that the entity instance has been processed by the AddRecord.ers Rulesheet. Let's verify that a record corresponding to the Output has been added to the Shipment table (since Cargo is mapped to Shipment). In SQL Server Management Studio, right-click dbo.Shipment and select Select Top 1000 Rows:

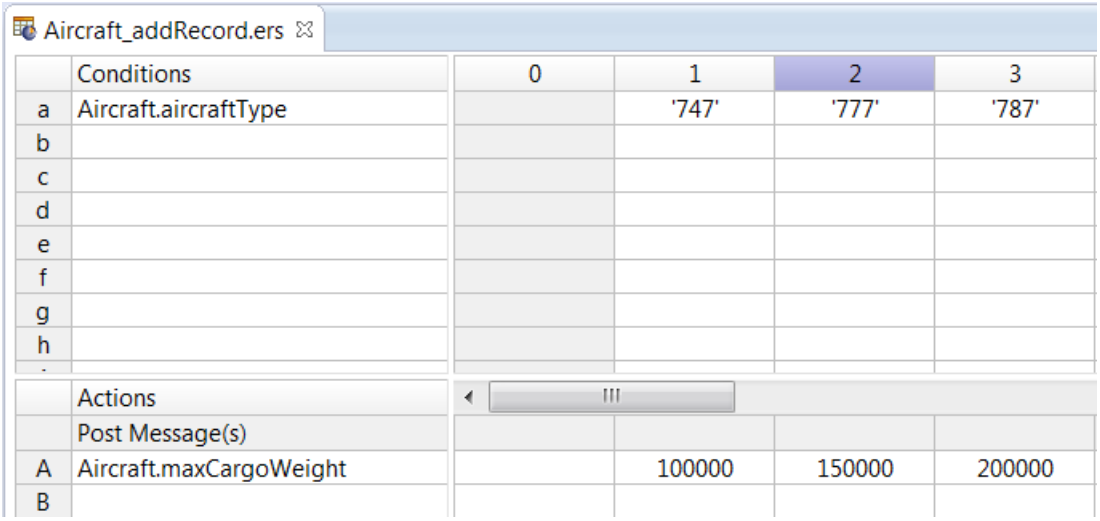


You see one record in the Results that corresponds to the Cargo entity instance in the Ruletest's output:



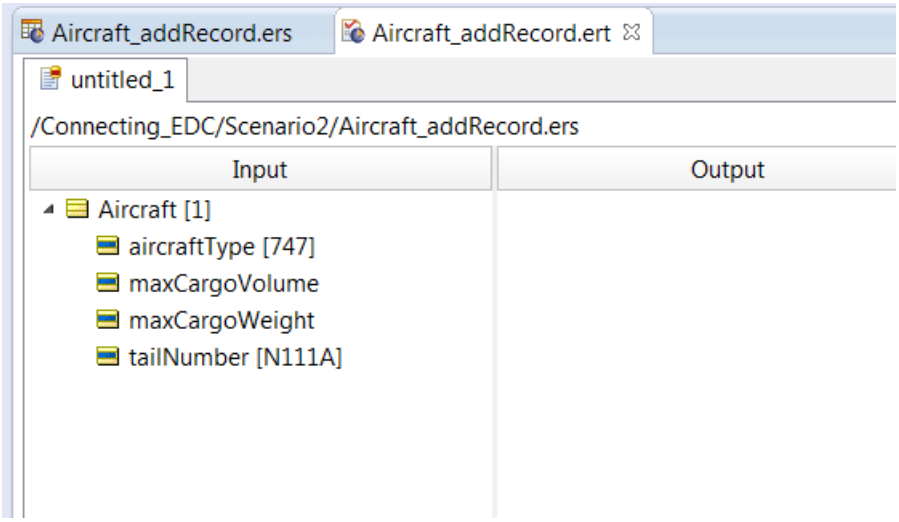
This illustrates how differently named entity/table elements (Cargo vs Shipment) work when mapped.

Let's verify that differently named attributes/columns work just as seamlessly too. Open Aircraft_addRecord.ers in the Scenario2 folder:



The Rulesheet assigns a value to Aircraft.maxCargoWeight based on the value of Aircraft.aircraftType received in input. As you may remember, the Aircraft entity is mapped to the Plane table, while the aircraftType attribute is mapped to a planeType column in the database.

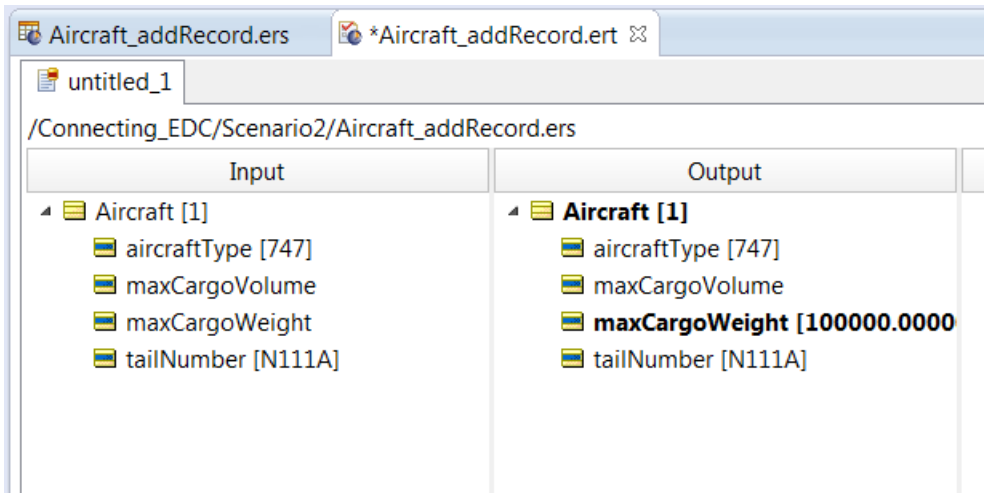
Open Aircraft_addRecord.ert in the Scenario2 folder:



The Ruletest contains one Aircraft entity instance in input. The aircraftType is set to 747.

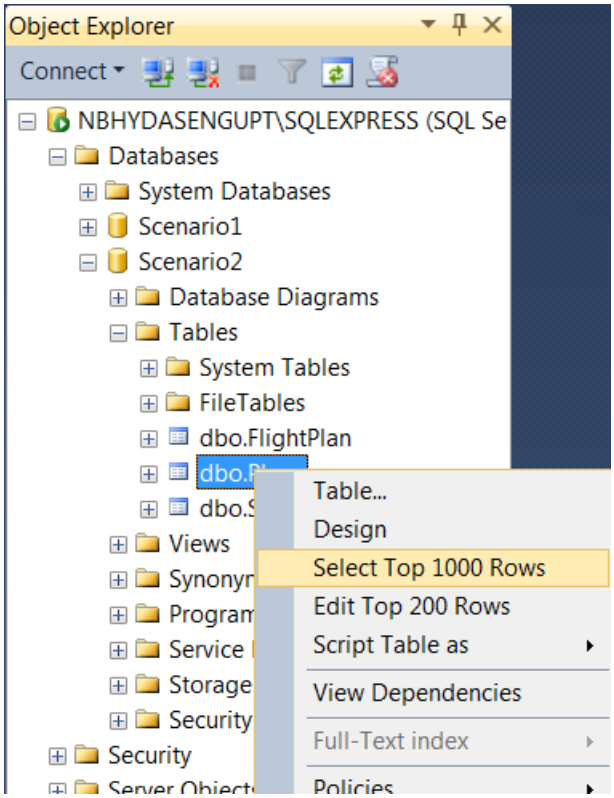
Let's ensure that the Ruletest is configured to perform read/write operations on the database. Select Ruletest > Testsheet > EDC Database Access. There is a checkmark next to Read/Update.

Run the Ruletest. You see the following output:

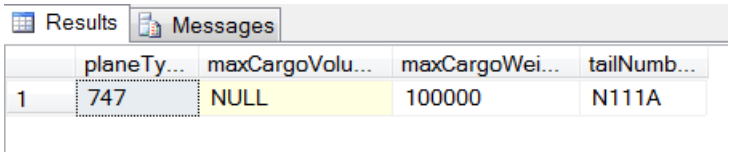


As you can see, the value of the maxCargoWeight attribute has changed. Let's verify that a new record has been added in the Plane table.

In SQL Server Management Studio, right-click dbo.Plane and select Select Top 1000 Rows:



You see one record in the results area corresponding to the Ruletest's output:



The aircraftType attribute is represented through the planeType column in the dbo.Plane table. The values of other columns are also correctly populated based on the Ruletest's output.

You have now successfully mapped a Vocabulary to a database in two ways—by generating a database schema from an existing Vocabulary and by mapping an existing Vocabulary to an existing database schema.

6

Deploying a Decision Service that accesses a database

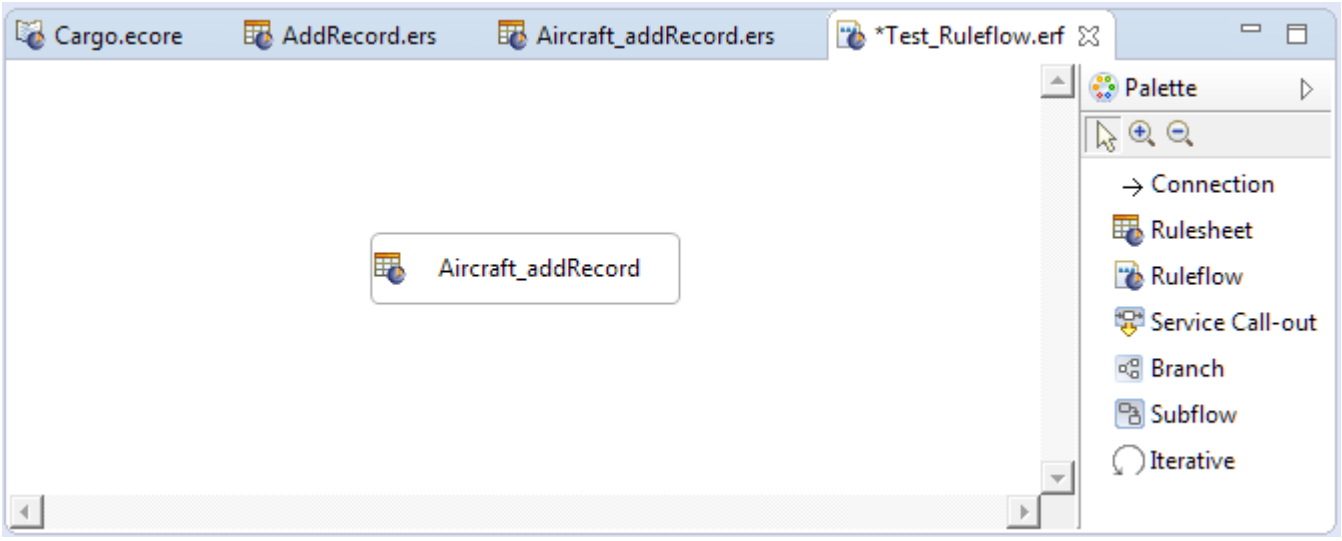
As an integration developer, you are responsible for deploying a Ruleflow as a Decision Service on Corticon Server. During deployment, if your Decision Services will access a database, you need to configure database access settings. Once that is done, the deployed Decision Service will automatically access the database.

In this section of the tutorial, you will deploy the Ruleflow (named Test_Ruleflow) that is part of the Scenario2 folder in the Connecting_EDC rule project.

There are different ways to deploy a Ruleflow as a Decision Service on Corticon Server. The recommended way is to package the Ruleflow into an EDS file and then deploy the EDS file on Corticon Server using the Corticon Web Console. You configure deployment settings (including database access settings) in the Web Console.

Packaging the Ruleflow into an EDS file

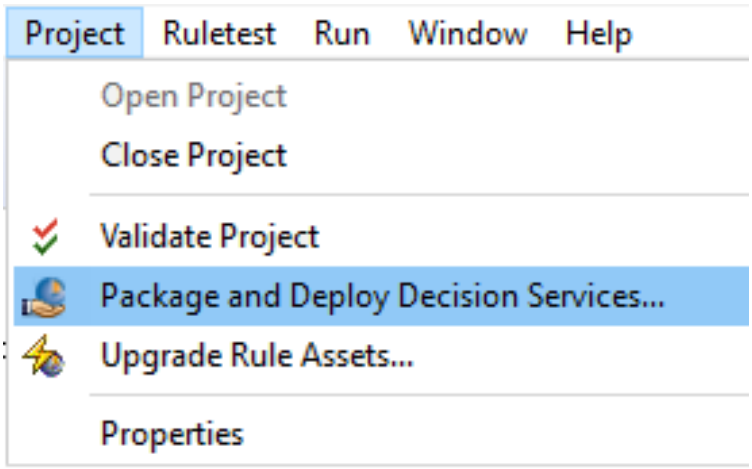
Let's start by packaging the Ruleflow in Corticon Studio. Open the Test_Ruleflow.erf file.



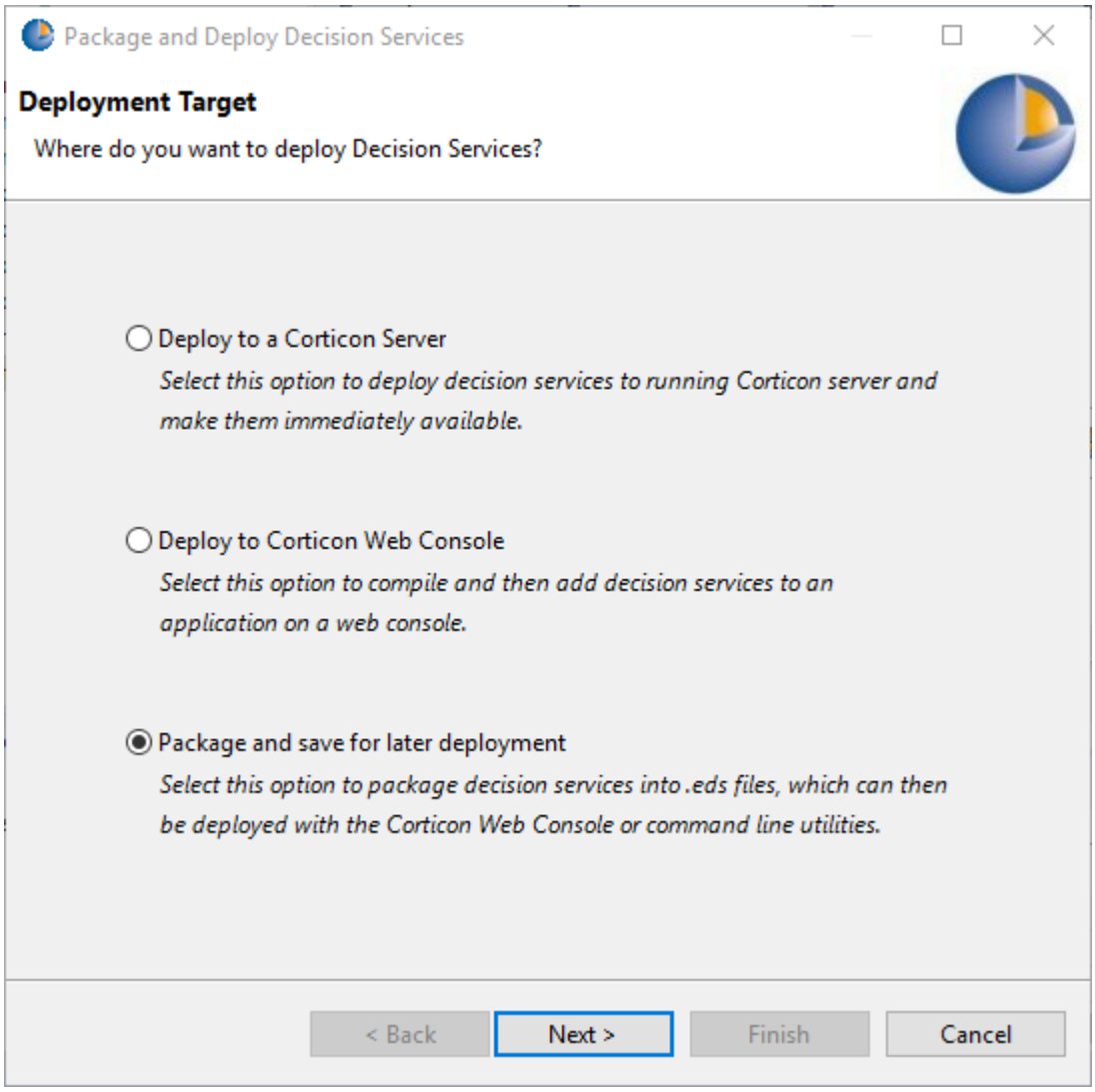
The Ruleflow has just one Rulesheet—Aircraft_addRecord—which you used in the previous section.

Let's package this Ruleflow into an EDS file. When we do this, the Ruleflow, the Rulesheet that it contains, and the Vocabulary all get compiled into a package that can be deployed on Corticon Server.

To package the Ruleflow, select Project > Package and Deploy Decision Services.

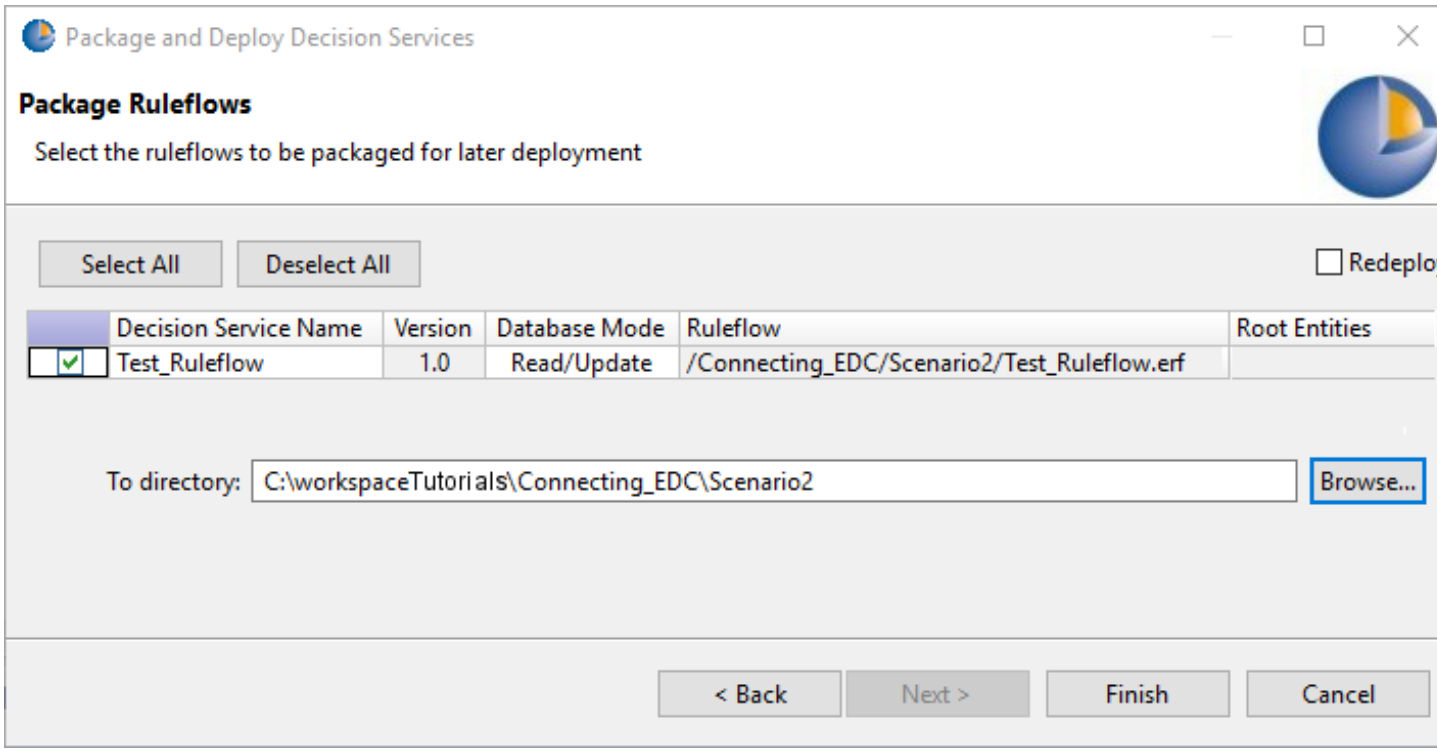


In the Deployment Target window, choose Package and save for later deployment and click Next.

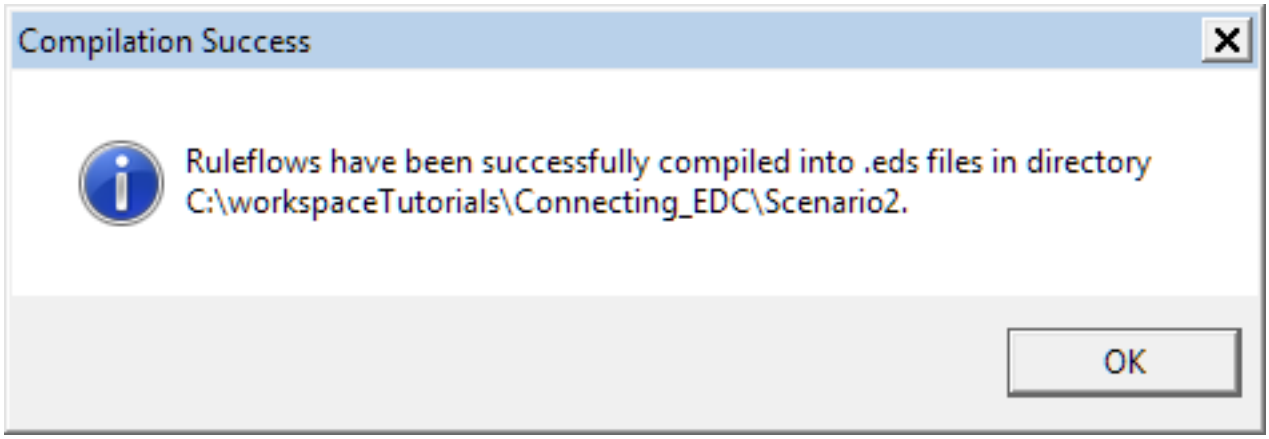


In the Package Ruleflows screen:

- Select Test_Ruleflow.
- Choose **Read/Update** in the **Database Mode** drop-down.
- Choose a location to save the EDS file by clicking the **Browse** button.(It is always good practice to save it in the same folder as the Ruleflow.)
- Click Finish.



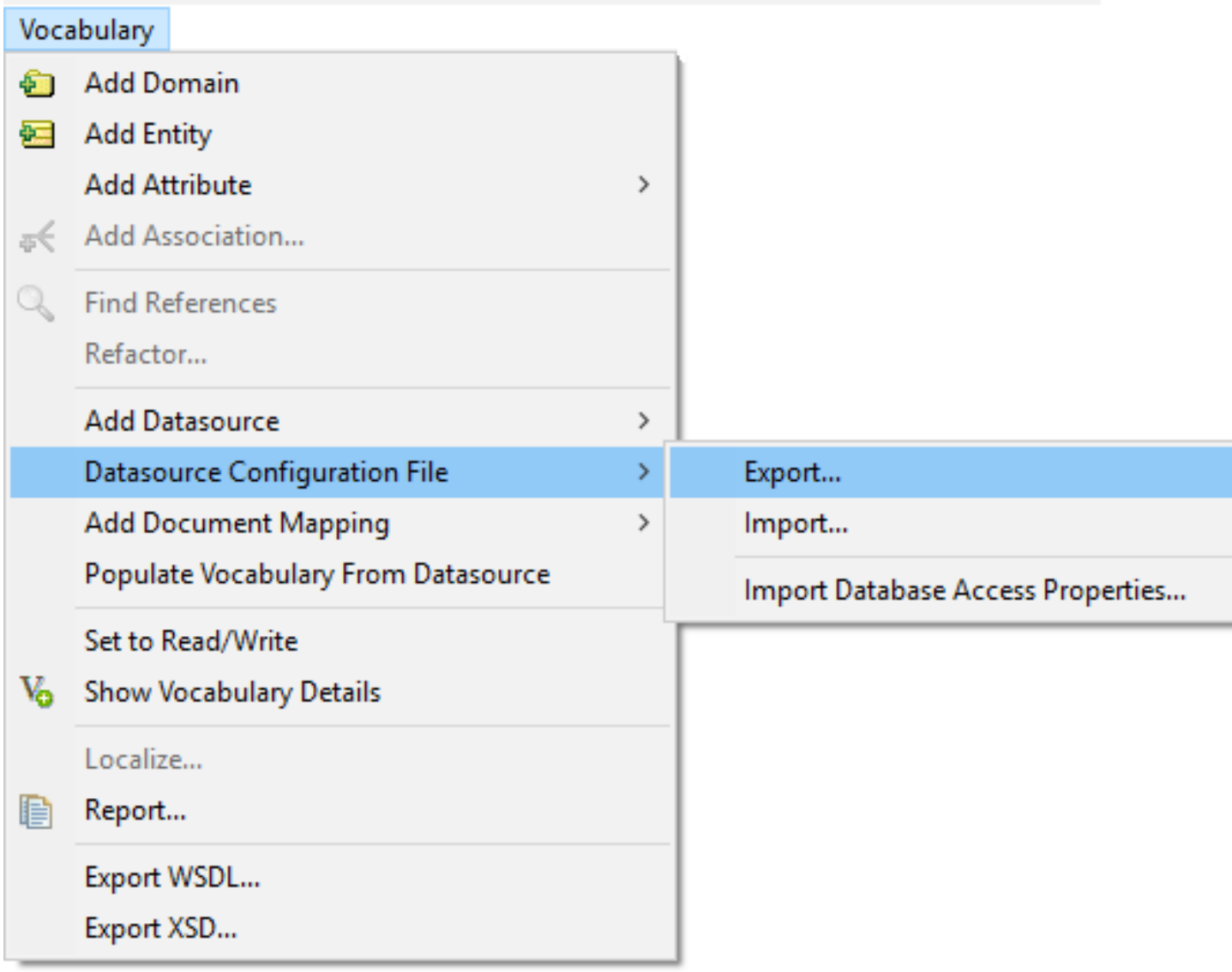
You see a message indicating that the Ruleflow was successfully compiled into an EDS file and saved in the location you specified.



Exporting the Datasource configuration file

As part of deploying the Ruleflow as a Decision Service, you need to supply a configuration file containing information about the database connection. The best way to create the file is to export the properties from Corticon Studio and modify them for your production environment.

Export the database properties by opening the Cargo.ecore Vocabulary in the Scenario2 folder and selecting **Vocabulary > Datasource Configuration File > Export**.



In the **Save As** dialog box, select a location where you want to save the XML file, give it a name such as **Scenario2_Datasource.xml**. (It is always good practice to save it in the same folder as the Decision Service file. Click Save.

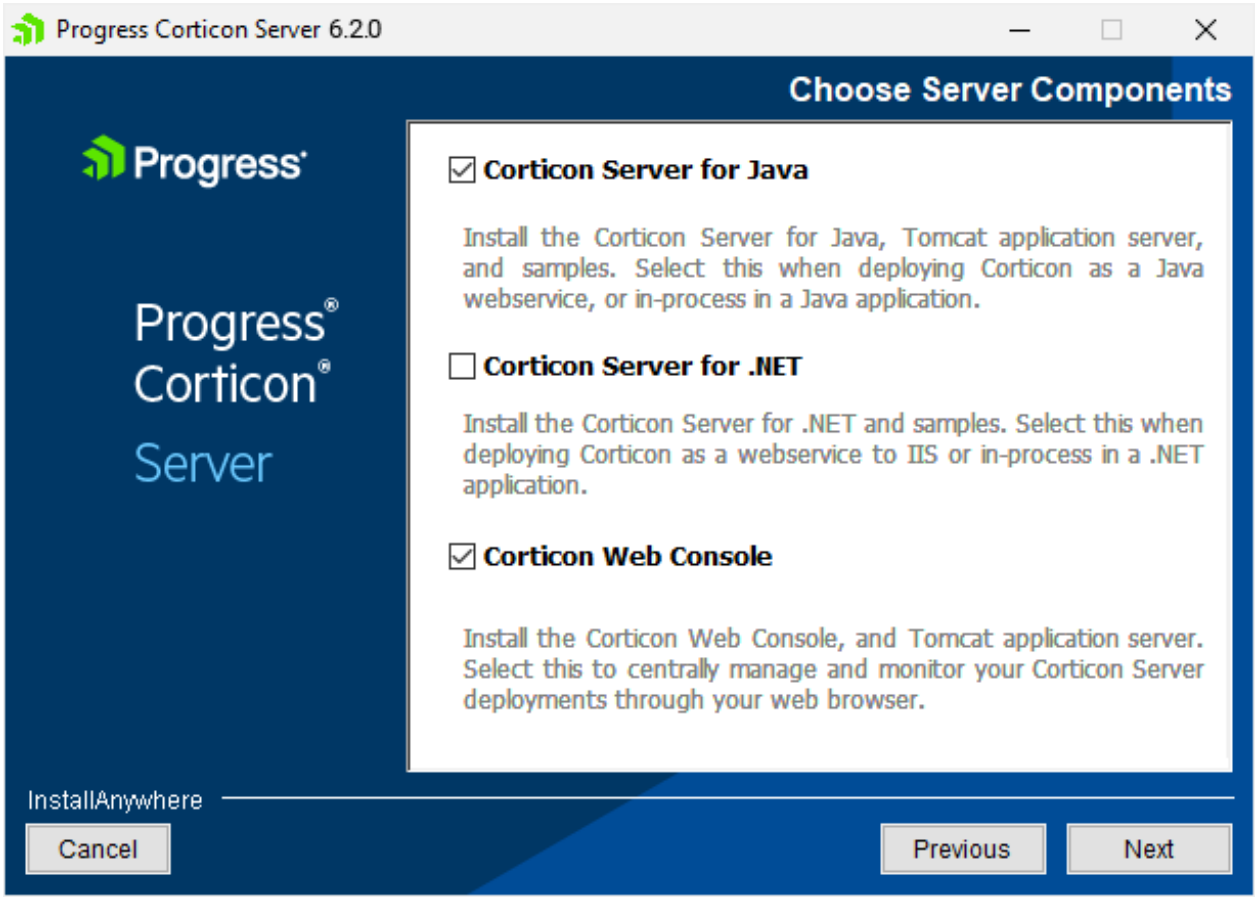
In this tutorial, you will use the same SQL Server database to simulate the production environment, so you do not need to modify the database connection properties in the properties file.

Installing Corticon Server

You'll install Corticon Server for Java so that you can deploy the Decision Service to it. We'll use the Corticon Web Console to do the deployment so you will install that component as well.

1. Download PROGRESS_CORTICON_x.x_SERVER_WIN_64.exe
2. Double-click the installer executable. The installation wizard opens.
3. On the **Introduction** page of the installation wizard, click **Next**.
4. On the **License Agreement** page, read, understand , and agree to the terms, then select **I accept the terms of the License Agreement,** and then click **Next**.
5. On the **Corticon Server Install and Work Directory**, retain the default settings,and then click **Next**

6. On the Choose Server Components page, select Corticon Server for Java and Corticon Web Console, and then click Next

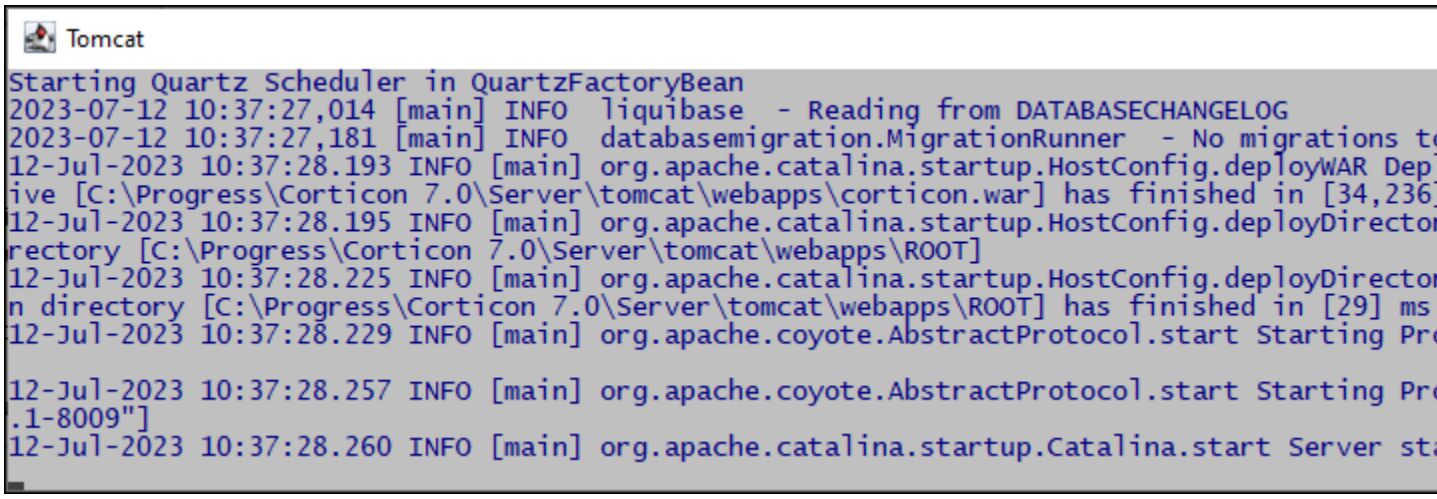


7. On the **Corticon Server Instance and Port Numbers** page, accept the default settings, and then click **Next**.
8. On the Pre-Installation Summary page, click **Install**.
9. Wait for a few minutes for the installation to complete. Then, on the **Install Complete** page, click **Done**.

Starting Corticon Server

On the Start menu, choose **Progress > Start Corticon Server**.

This launches a command console. Corticon Server takes a few minutes to start the first time. When it is running, you are informed of Server startup:

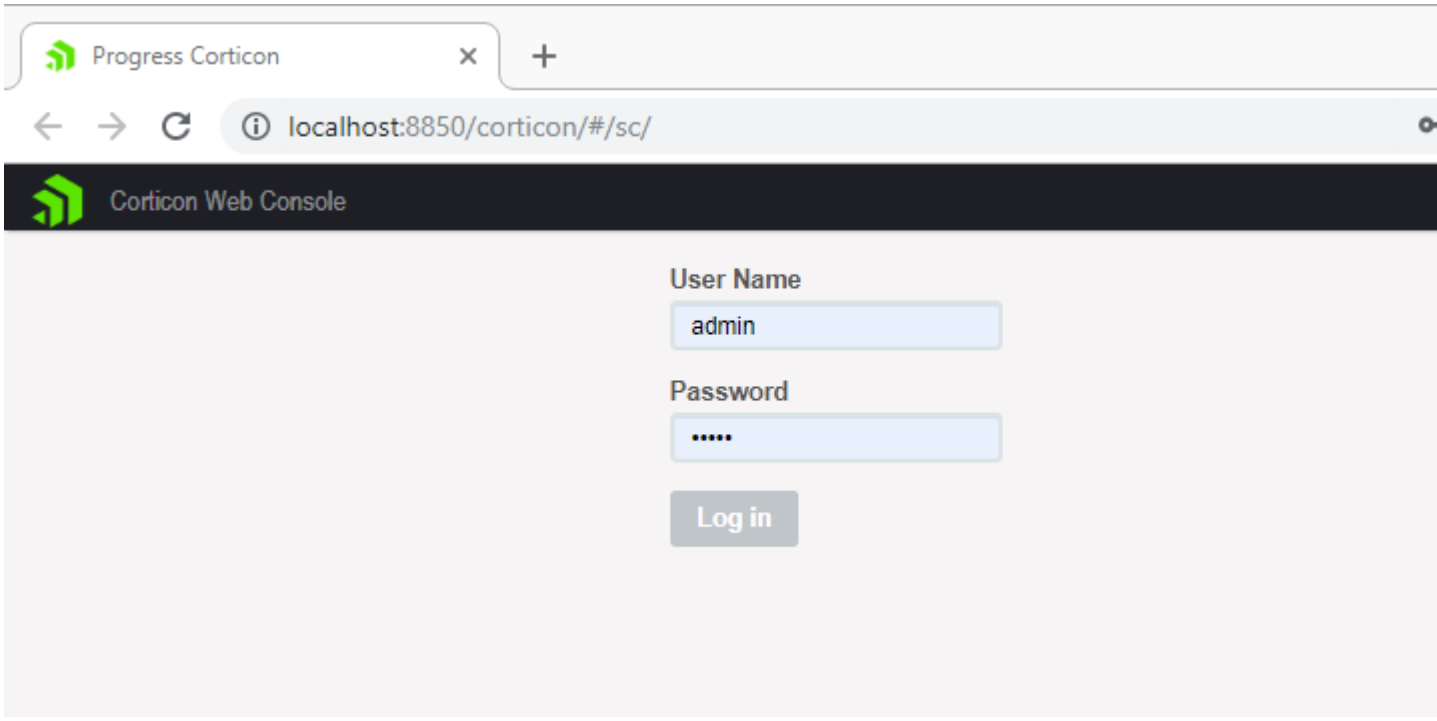


Deploying the Decision Service using the Web Console

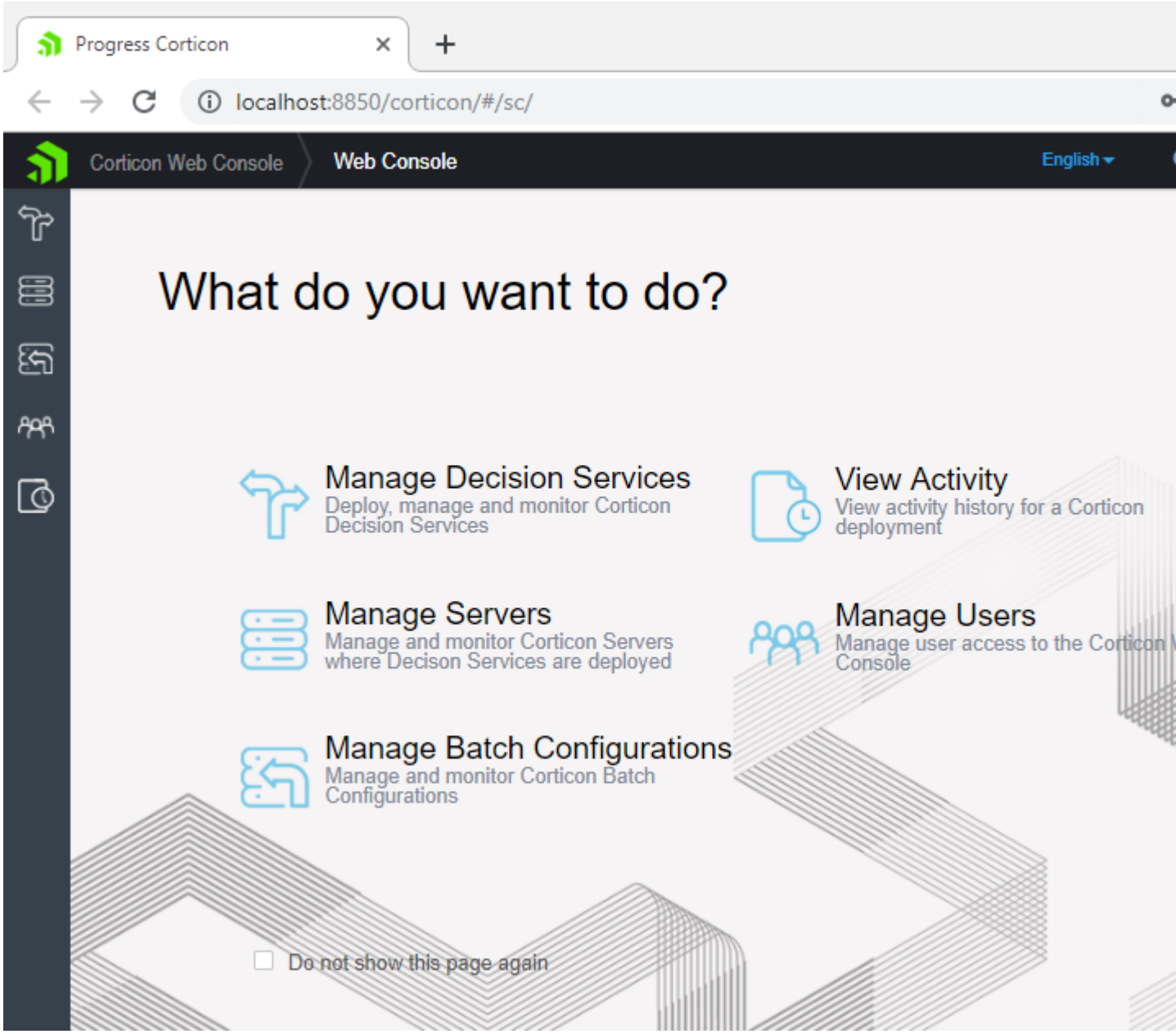
The Corticon Web Console lets you deploy and then monitor your Decision Services.

Launch the Web Console in your browser by entering the URL **http://localhost:8850/corticon.** .

The Web Console login page opens. Enter admin/admin in the User Name/Password fields and click **Log in**.

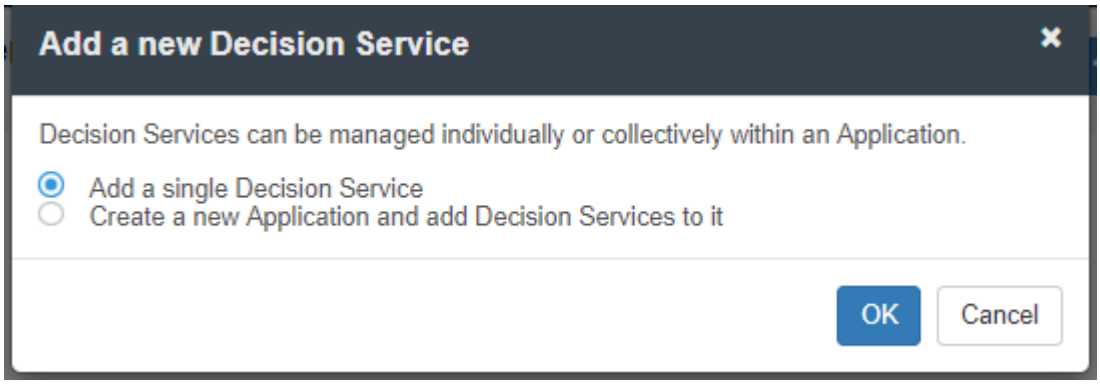


The Web Console Home page opens. Click on **Manage Decision Services**.

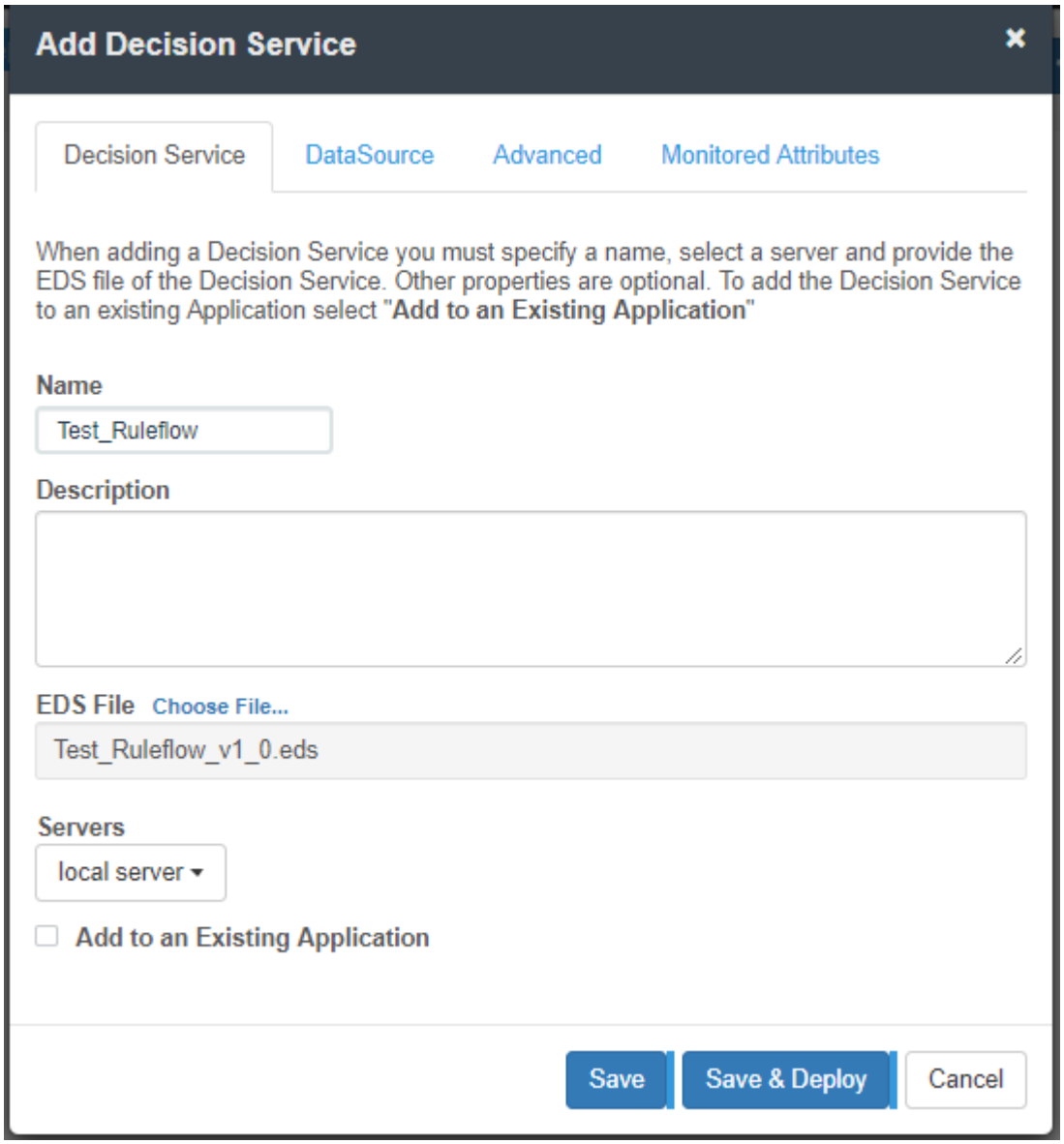


On the Decision Services page, click + Add Decision Service.

In the options panel, choose Add a new Decision Service.



The Add Decision Service dialog box opens on its Decision Service tab:



On the **Add Decision Service** dialog box:

1. In the **Name** field, enter **Test_Ruleflow** as the name for the Decision Service.

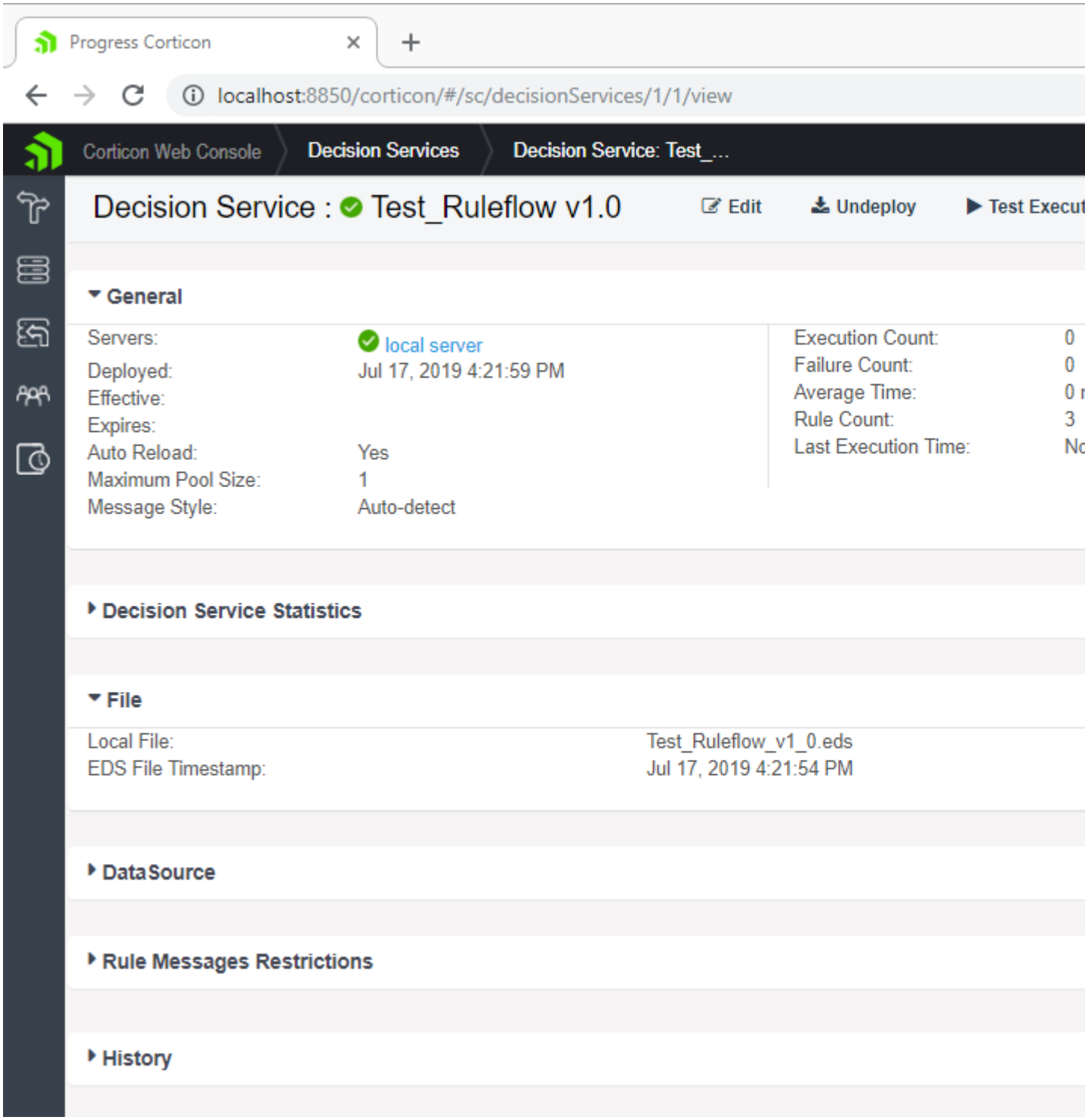
   In the EDS File section, click Choose File. In the window that opens, navigate to the location where you saved the EDS file, Test_Ruleflow_v1.0.eds, and then click Open.

2. On the **Select Servers** pulldown, choose **local server**.
3. Click the **DataSource** tab. Do the following:

   - In the **Datasource Configuration File** section, click **Choose File**.
   - In the window that opens, navigate to the location where you saved the EDS file, **Scenario2_Datasource.xml**, and then click **Open**.
   - In the **EDC Access Mode** section, click **Read/Update**.

4. Click **Save and Deploy**.

The Decision Service is successfully deployed. The green checkmark next to the Application name shows that it properly deployed onto the local server:



Testing

Finally, let's test the deployed Decision Service. One way to test it is to use a Ruletest in Corticon Studio and select the Decision Service as its Test Subject.

Recall that the Decision Service has just one Rulesheet—Aircraft_addRecord.ers—which checks an Aircraft entity instance's aircraftType, assigns a value to the instance's maxCargoWeight attribute, and then adds the instance as a new Aircraft record if a new primary key value is supplied in the tailNumber attribute.

To begin with, let's look at the existing Aircraft records. In SQL Server Management Studio, right-click dbo.Plane in the Scenario2 database and select Select Top 1000 Rows. As you can see there is just one record:

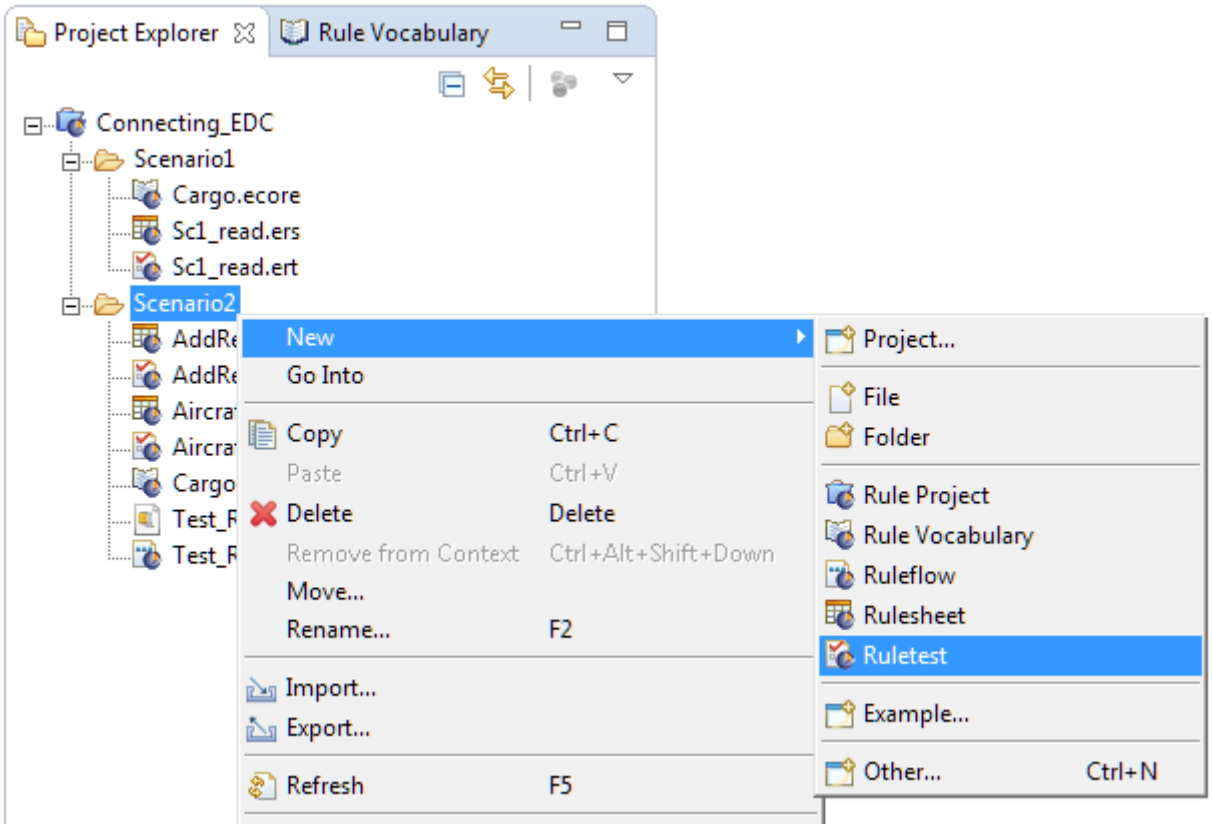
Results | Messages

	planeTy...	maxCargoVolu...	maxCargoWei...	tailNumb...
1	747	NULL	100000	N111A

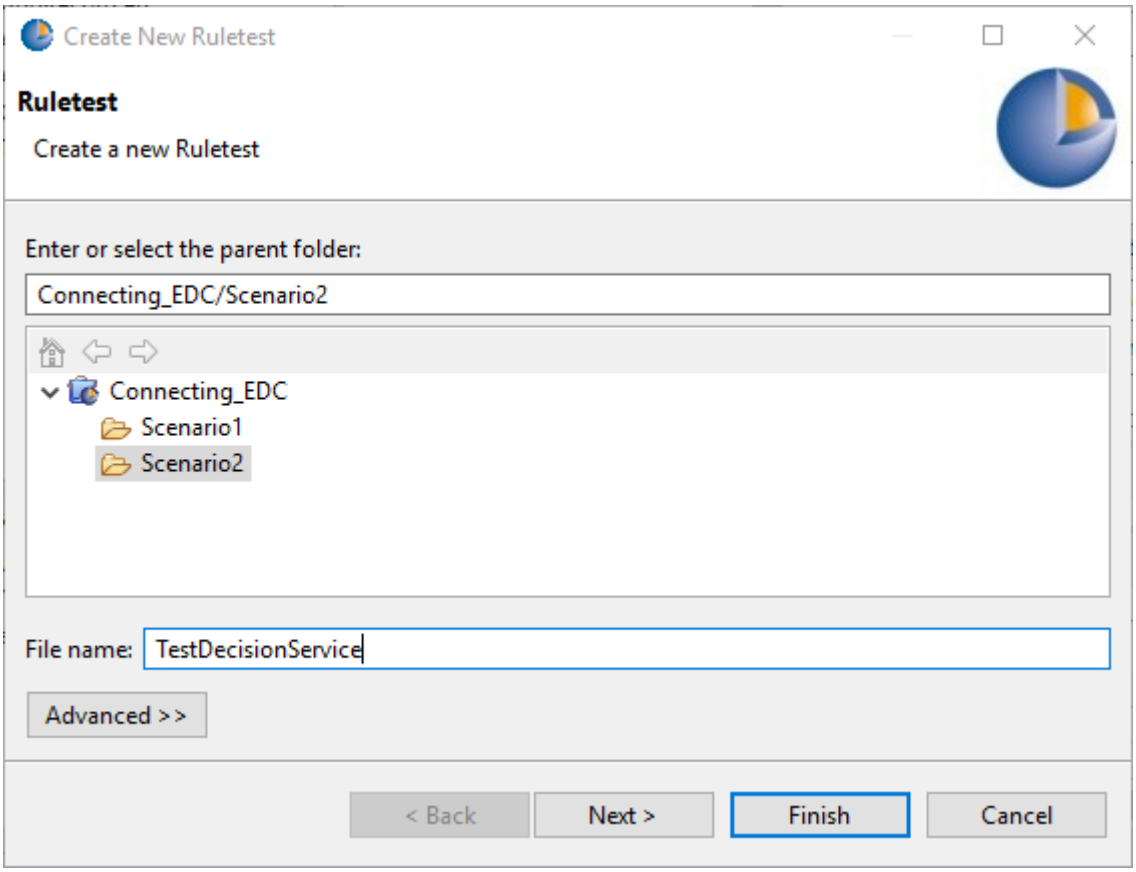
By running the Ruletest, you will verify that when you supply input data with a different primary key value to the Decision Service, a new record is added to this table.

In Corticon Studio, create a new Ruletest named Test_DecisionService as follows:

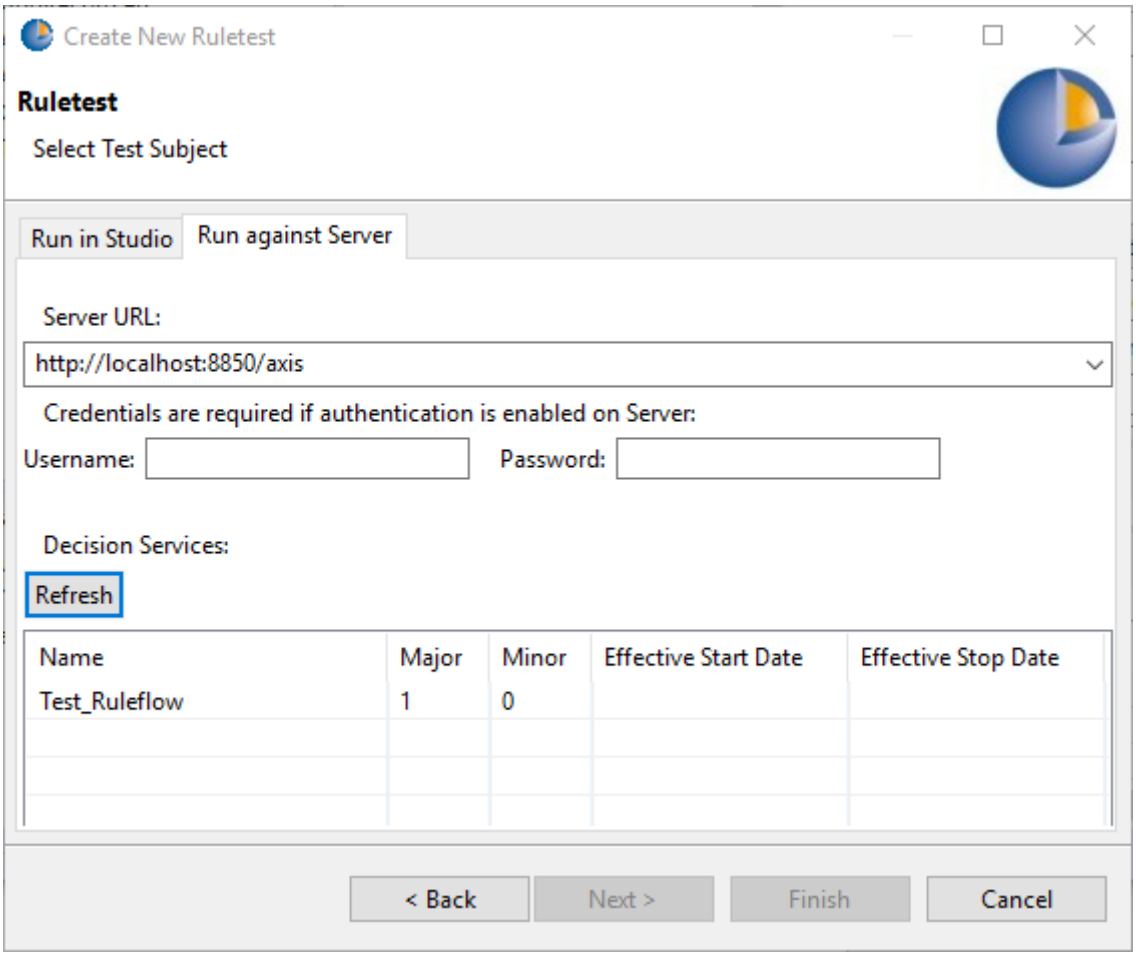
1. Right-click the **Scenario2** folder and select **File > New > Ruletest**:



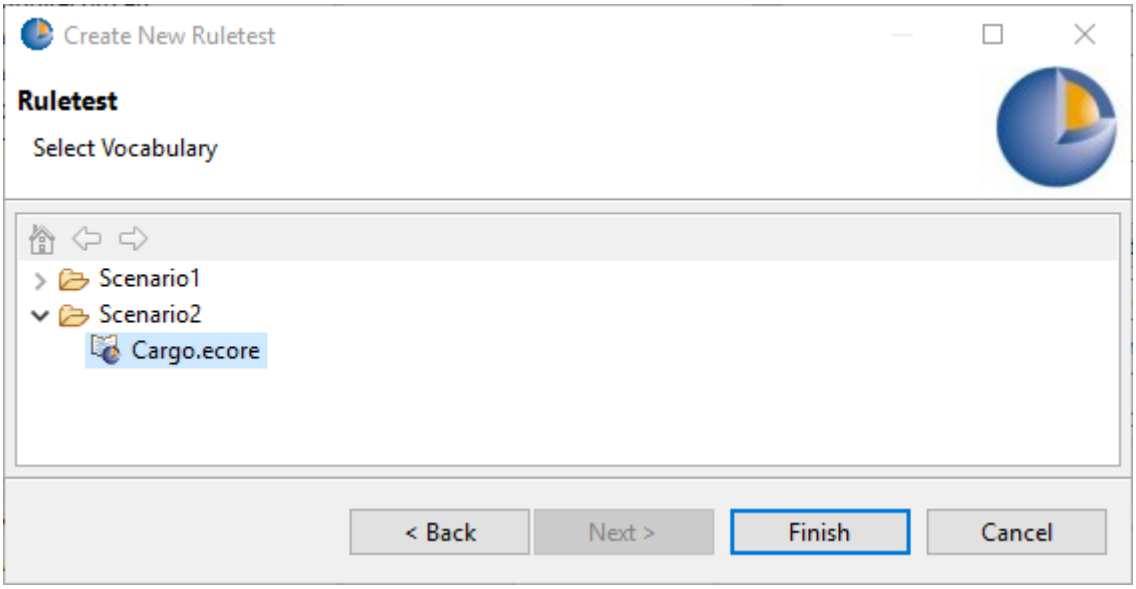
2. In the **Create New Ruletest** wizard, enter **Test_DecisionService** in the **File name** field, and then click **Next**.



3. On the Select Test Subject page:
4. Select the Run against Server tab.
5. Select the Corticon Server URL (http://localhost:8850/axis) from the **Server URL** drop-down.
6. Click the **Refresh** button next to **Decision Services**.
7. Select Test_Ruleflow in the Decision Services section.
8. Click **Next**.



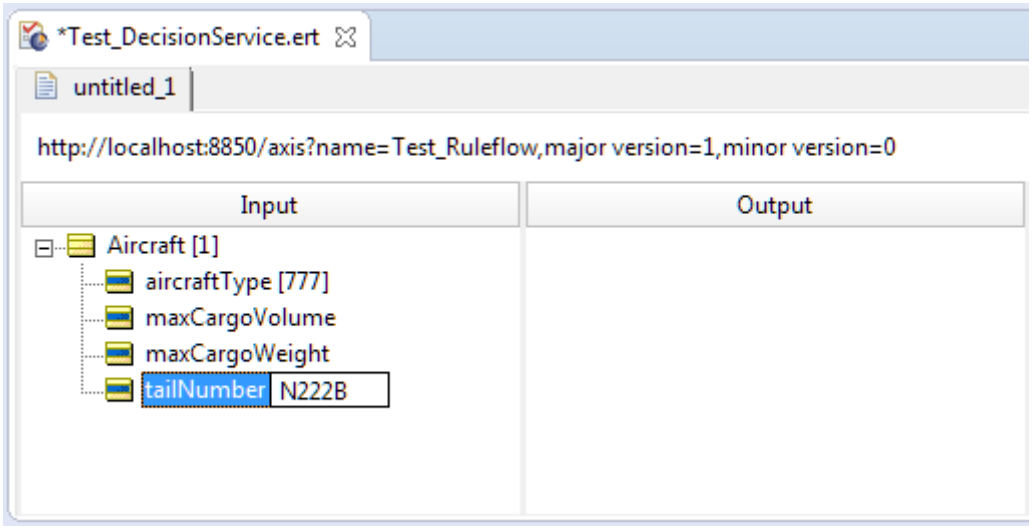
9. On the **Select Vocabulary** page, ensure that **Cargo.ecore** in the **Scenario2** folder is selected,and then click **Finish**.



The Ruletest is now configured. The Test Subject in the Ruletest (highlighted in the image below) should point to the Test_Ruleflow Decision Service. Drag and drop an Aircraft entity instance from the Rule Vocabulary view to the Input pane and define the following values:

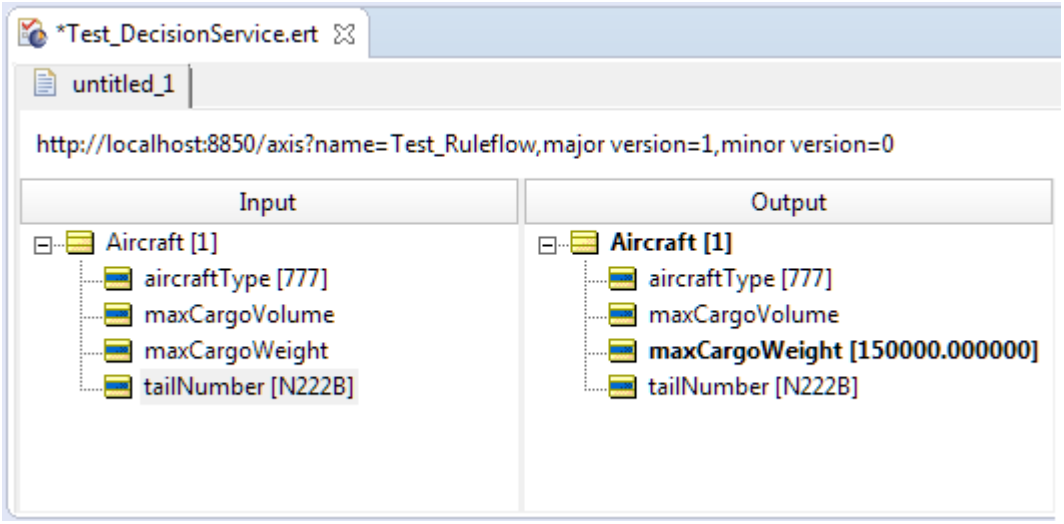
- aircraftType = 777

- tailNumber = N222B (which is unique)

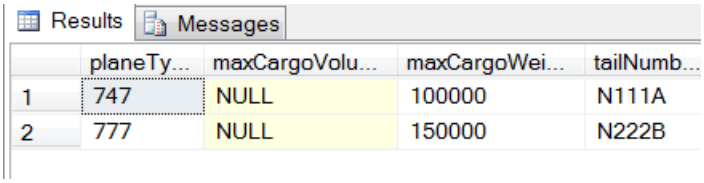


Since this Ruletest invokes a Decision Service on Corticon Server, you do not need to configure its database access setting.

Run the Ruletest. You see the following output:



Let's verify that a new record has been added to the dbo.Plane table based on the Output of this Ruletest. In SQL Server Management Studio, right-click dbo.Plane in the Scenario2 database and select Select Top 1000 Rows. You see the new row was added to the database table:



Our Decision Service running on the Corticon Server accessed and wrote to the database.

Congratulations! You have now mapped a Vocabulary to a database and deployed a Ruleflow as a Decision Service on Corticon Server that can read and write to a database.