Progress®

Corticon Tutorial

Modeling Progress Corticon Rules to Access a Database using EDC

Progress® Corticon®

Copyright

© 2019 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Updated: 2019/11/15

Table of Contents

Tutorial - Modeling Progress Corticon Rules to Access a Database using EDC.....7

Setting up the tutorial.....9

Reading records from a database using a primary key.....25

Modeling the rules.....25

Testing the rules.....30

Reading records from a database without a primary key.....35

Writing to the database 43

Adding a new record to the database.....44

Updating an existing record.....46

Adding a record using a primary key defined in a rule.....47

Deleting records from a database.....51

Deleting a specific record using a primary key.....51

Deleting multiple records based on rule conditions.....53

1

Tutorial - Modeling Progress Corticon Rules to Access a Database using EDC

In the Basic Rule Modeling and the Advanced Rule Modeling tutorials, you learned how to develop and test Corticon rules using a range of Corticon Studio features. You learned how to model rules that processed input data and returned output data.

In many cases, you may want the rules to retrieve data from a database, process it, and/or write data to the database. For example, suppose you wanted to model a rule that receives an aircraft tail number in an incoming request message, and retrieves details about the aircraft type from a database. Or, suppose you wanted to model a rule that writes a record of each flight plan to a database.

Corticon's Enterprise Data Connector (EDC) enables rules to perform read and write operations on data in a database. In this tutorial, you will learn how to use EDC to model rules that:

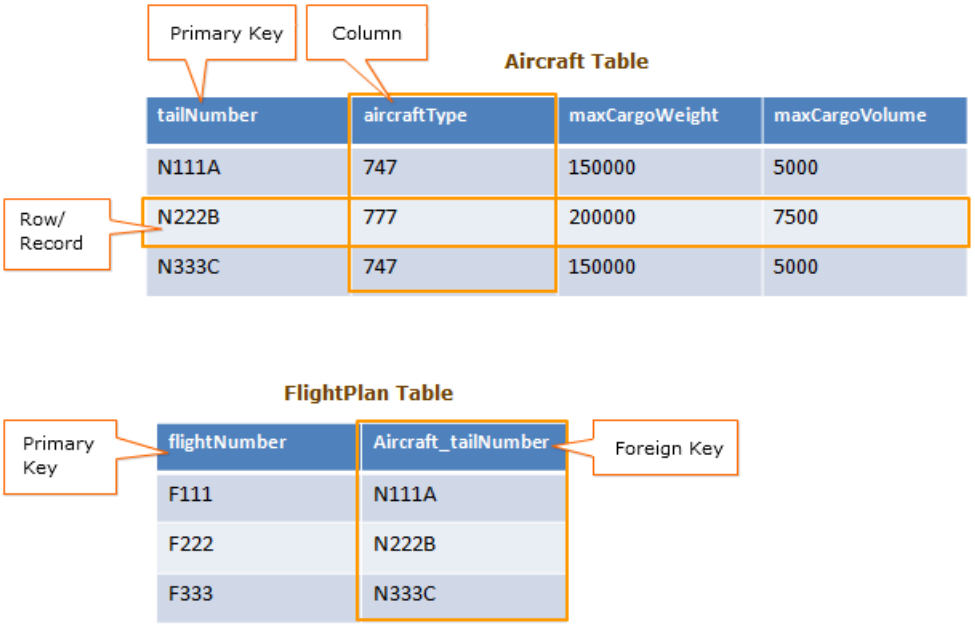
- Read data from a database
- Add or update data to a database
- Delete data from a database

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 6.1.

How database elements map to Vocabulary elements

For rules to read or write to a database, the Vocabulary elements used in the rules must be mapped to elements in the database. The mapping is typically done by an integration developer. However, as a rule modeler, it is useful for you to understand how these elements are mapped.

A relational database is a set of tables. Each table corresponds to an entity in a Vocabulary. A table contains a set of records. Each record is a row in the table. A record is like an entity instance in a Ruletest—it is a set of data values. Columns in the table correspond to attributes in the entity.



Each table has a primary key which uniquely identifies records in the table. The primary key is one of the columns in the table. For example, each **Aircraft** record could be uniquely identified by its **tailNumber**, while each **FlightPlan** record could be identified by its **flightNumber**. Each table can also have a foreign key which associates its records with records in a different table. The foreign key is usually the primary key of the other table. For example, the FlightPlan table could have a foreign key named **Aircraft_tailNumber**, which associates its records with records in an Aircraft table. The relationship between two tables corresponds to an association between two entities in the Vocabulary.

Finally, every database has a schema. A database schema is a blueprint of the database structure. The schema is like the Vocabulary tree. Along with other information, the schema contains the names of tables in the database, the column names, the primary key columns of each table, and information about how the tables are related.

This table summarizes how Vocabulary elements map to database elements:

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

Typically, as a rule modeler you are not responsible for mapping the Vocabulary to the database. This is done by an integration developer. Once the mapping is configured, your rules will automatically access the database. It is important for you to understand how your rules will affect the database. You should coordinate with the integration developer who is responsible for setting up the database and mapping the Vocabulary.

In this tutorial, you will model rules that read and write to a database. Before you can do this, you will need to set up the environment by performing tasks normally performed by an integration developer—such as installing the database and configuring mapping between the Vocabulary and the database. Don't worry, we will walk you through all the steps to set up the environment.

2

Setting up the tutorial

To set up your environment, you need to do the following:

1. Install Microsoft SQL Server Express
2. Configure Microsoft SQL Server Express
3. Create a database
4. Import a sample Corticon Rule Project
5. Map the Vocabulary to the database
6. Populate some sample data

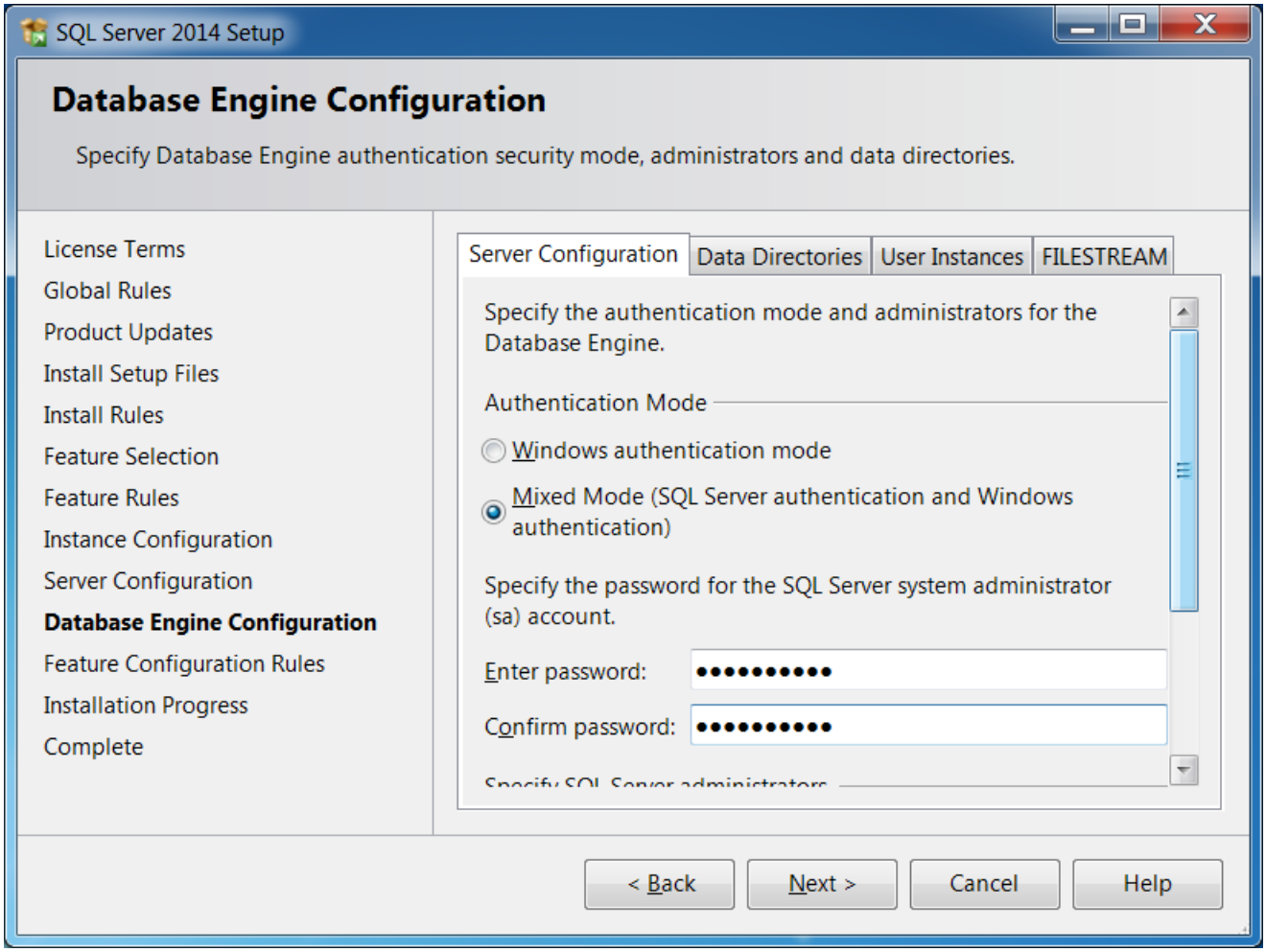
Note: EDC can work with a number of relational databases such as Oracle, Progress OpenEdge, IBM DB2, MySQL, and Microsoft SQL Server. For this tutorial, we will use Microsoft SQL Server Express 2014, which is free to use and distribute. All the instructions in this tutorial will assume you are using SQL Server Express.

Step 1: Installing Microsoft SQL Server Express

Microsoft SQL Server Express and Tools is a version of SQL Server that is free to use and distribute. Follow these steps to download, install, and configure SQL Server Express 2014 software:

1. Download the appropriate installer. On the webpage https://www.microsoft.com/en-us/download/details.aspx?id=42299, click Download.
2. On the Choose the download that you want page, select:ExpressAndTools 64BIT\SQLEXPRWT_x64_ENU.exe
3. Click Next.
4. In your downloads location, unzip its contents.
5. In the SQL Server Express folder, double-click SETUP.

6. In the SQL Server Installation Center wizard that opens, click New SQL Server stand-alone installation or add features to an existing installation.
7. Perform a typical installation, accepting the license agreement and retaining the default settings in each screen, and clicking Next until you reach the Database Engine Configuration panel.
8. IMPORTANT: On the Database Engine Configuration panel's Server Configuration tab:
   a. Select **Mixed Mode (SQL Server and Windows authentication)**
   b. In the **Enter Password** and **Confirm Password** fields, enter **sqlserver2014password**. This password is for a default administrator user named **sa**.



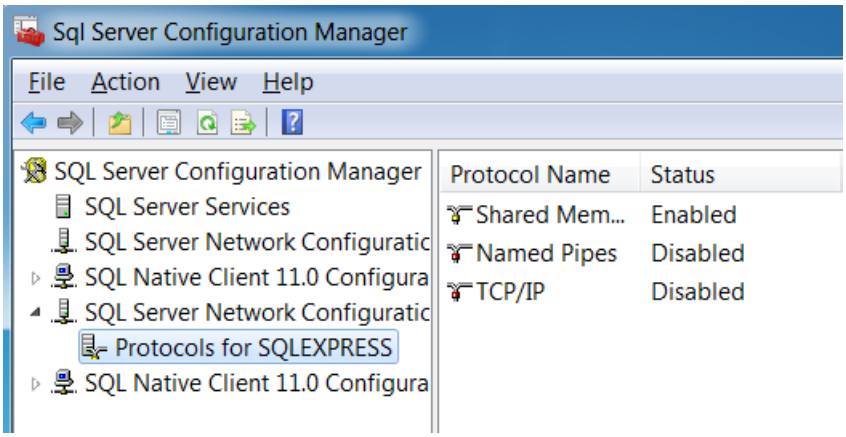
9. Click Next on the remaining panels, and then click Close to exit the wizard.

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

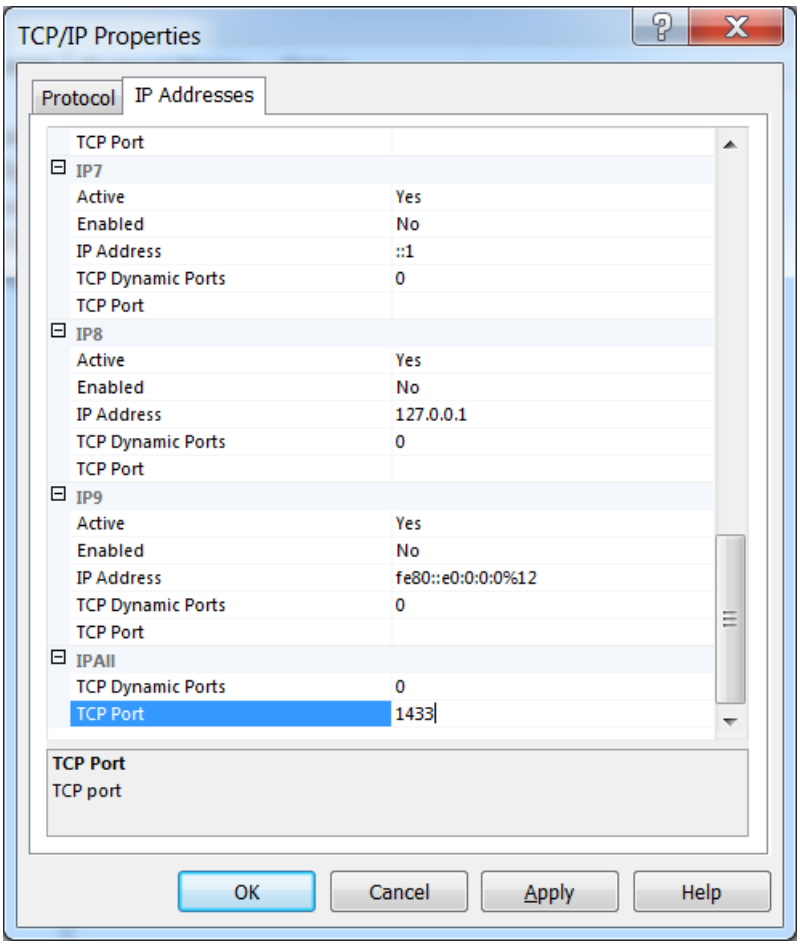
Step 2: Configuring Microsoft SQL Server Express

Corticon wants to connect to a database through TCP/IP on a designated port. We'll configure SQL Server Express for that in these steps:

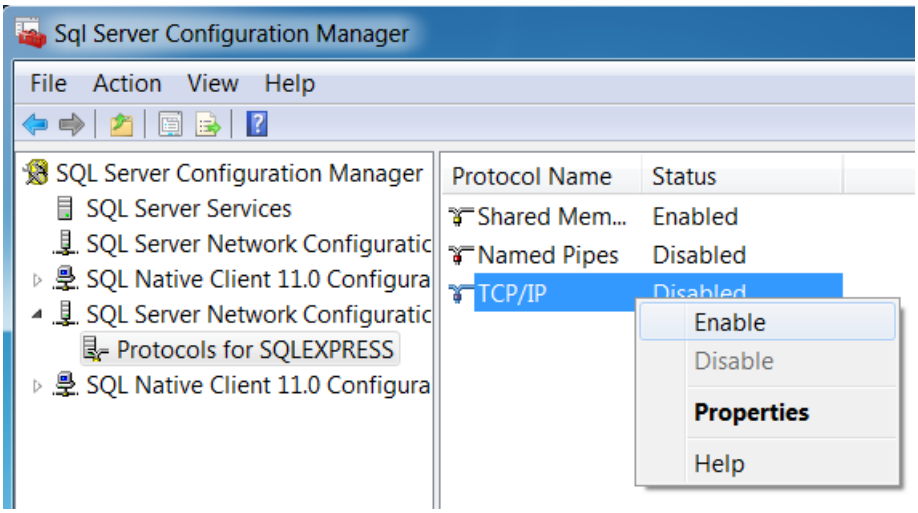
1. Choose **Start > Microsoft SQL Server 2014 > SQL Server 2014 Configuration Manager**.
2. Expand SQL Server Network Configuration in the left pane and select Protocols for SQLEXPRESS.



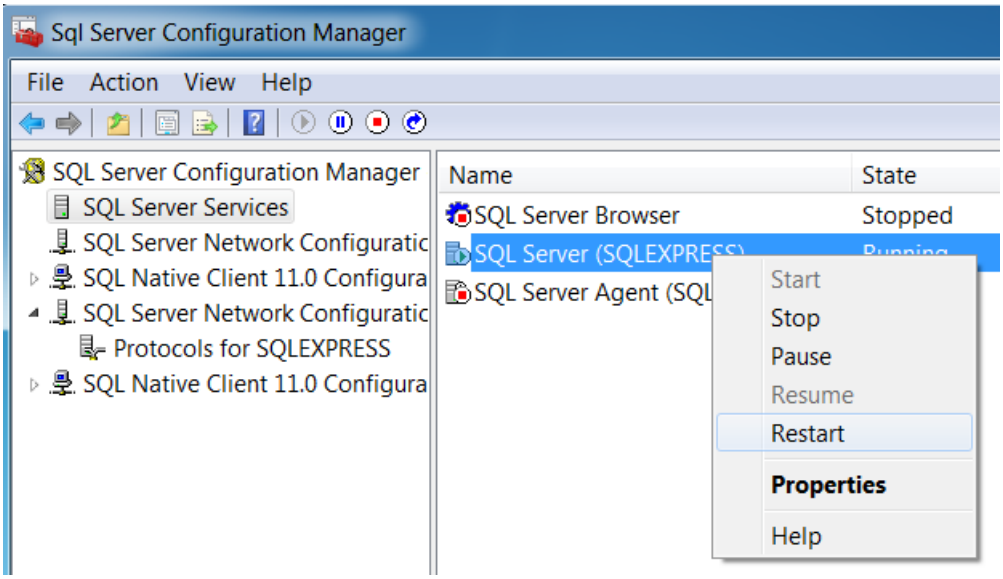
3. Right-click **TCP/IP** in the right pane and select Properties.
4. In the **TCP/IP Properties** window, click the **IP Addresses** tab and scroll to the bottom.
5. In the **IP All** section's **TCP Port** field, enter **1433**, click Apply, and then **OK**:



6. Click **OK** in the **Warning** message box.
7. Right-click **TCP/IP** and click **Enable**. Click **OK** to any warning messages.



8. Restart SQL Server Express by selecting **SQL Server Services** in the left pane, right-clicking **SQL Server (SQLEXPRESS)** on` the right, and then choosing **Restart**.

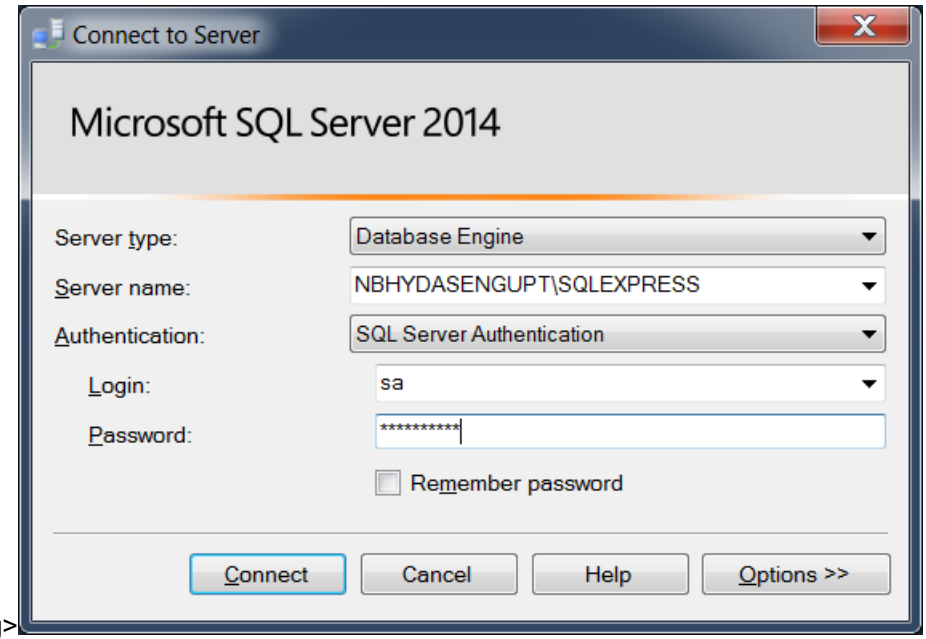


Step 3: Creating a database

Next, you will create a database named Transportation. Later, you will generate a schema for the database from a Corticon Vocabulary.

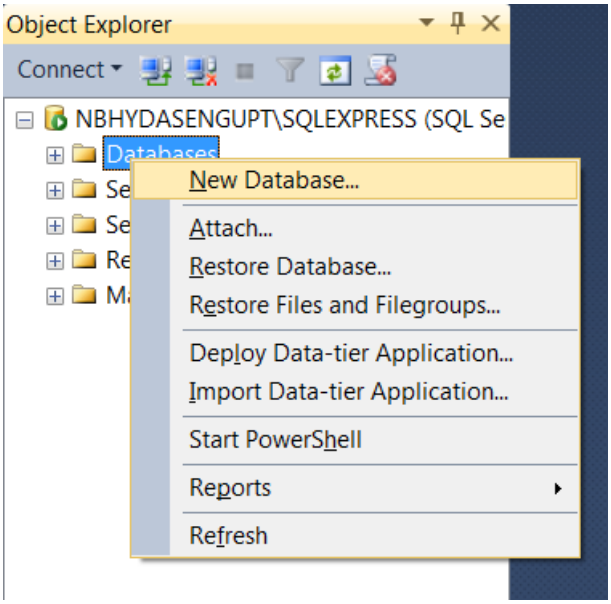
Follow these steps to create the database:

1. Launch SQL Server Management Studio by selecting **Start > All Programs > Microsoft SQL Server 2014 > SQL Server 2014 Management Studio**.
2. In the **Connect to Server** window, select **SQL Server Authentication** in the **Authentication** setting, enter **sa** in the **Login** field and **sqlserver2014password** in the **Password** field, and click **Connect**.

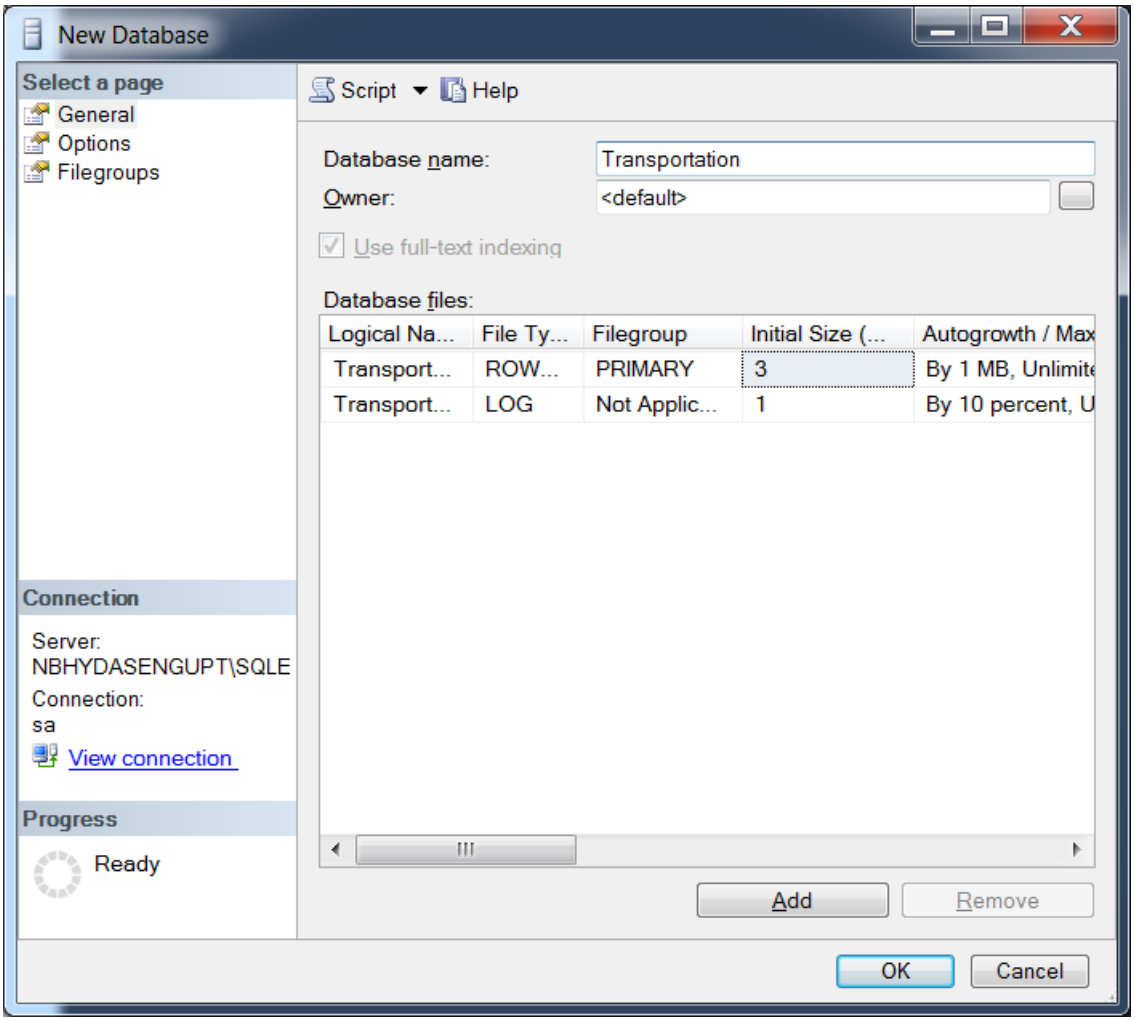


>fig>

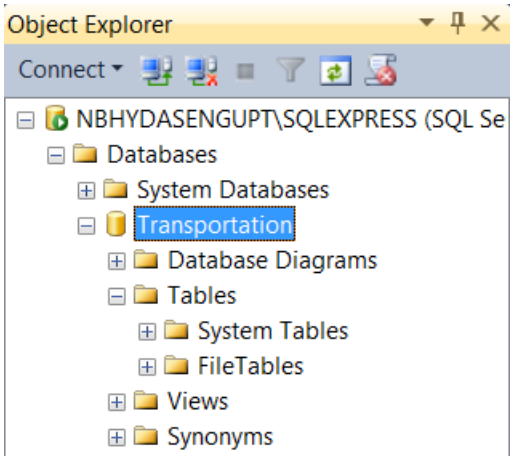
3. Right-click **Databases** and select **New Database**.



4. In the **New Database** window, enter **Transportation** as the Database name and click **OK**.



Your new database named Transportation should now be created. To verify this, expand **Databases**. You should be able to see a new database folder named Transportation. If you expand **Transportation** and then expand its **Tables** subfolder, you will see some default system and file tables, but no user-defined tables.

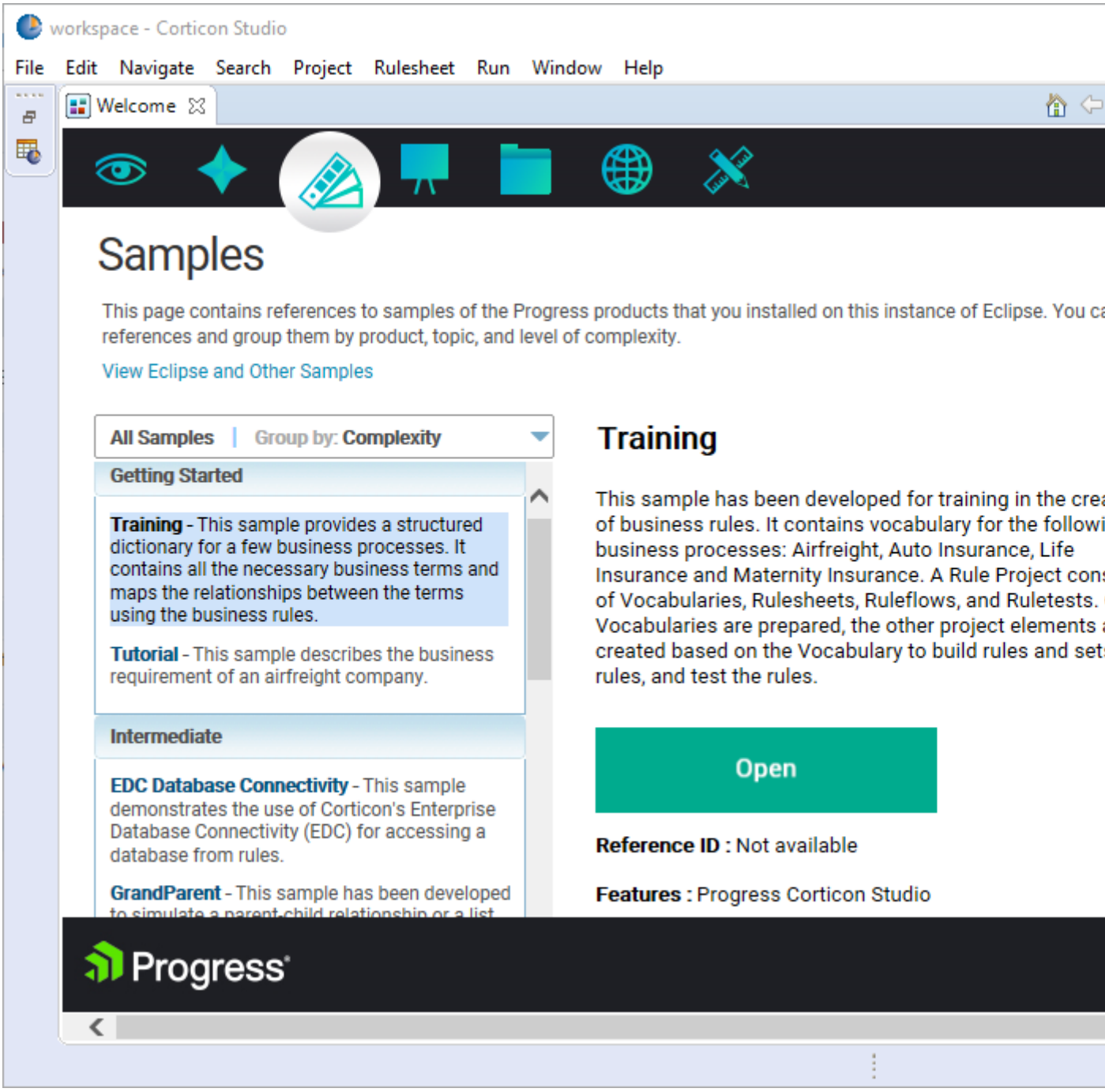


Step 4: Opening a sample rule project

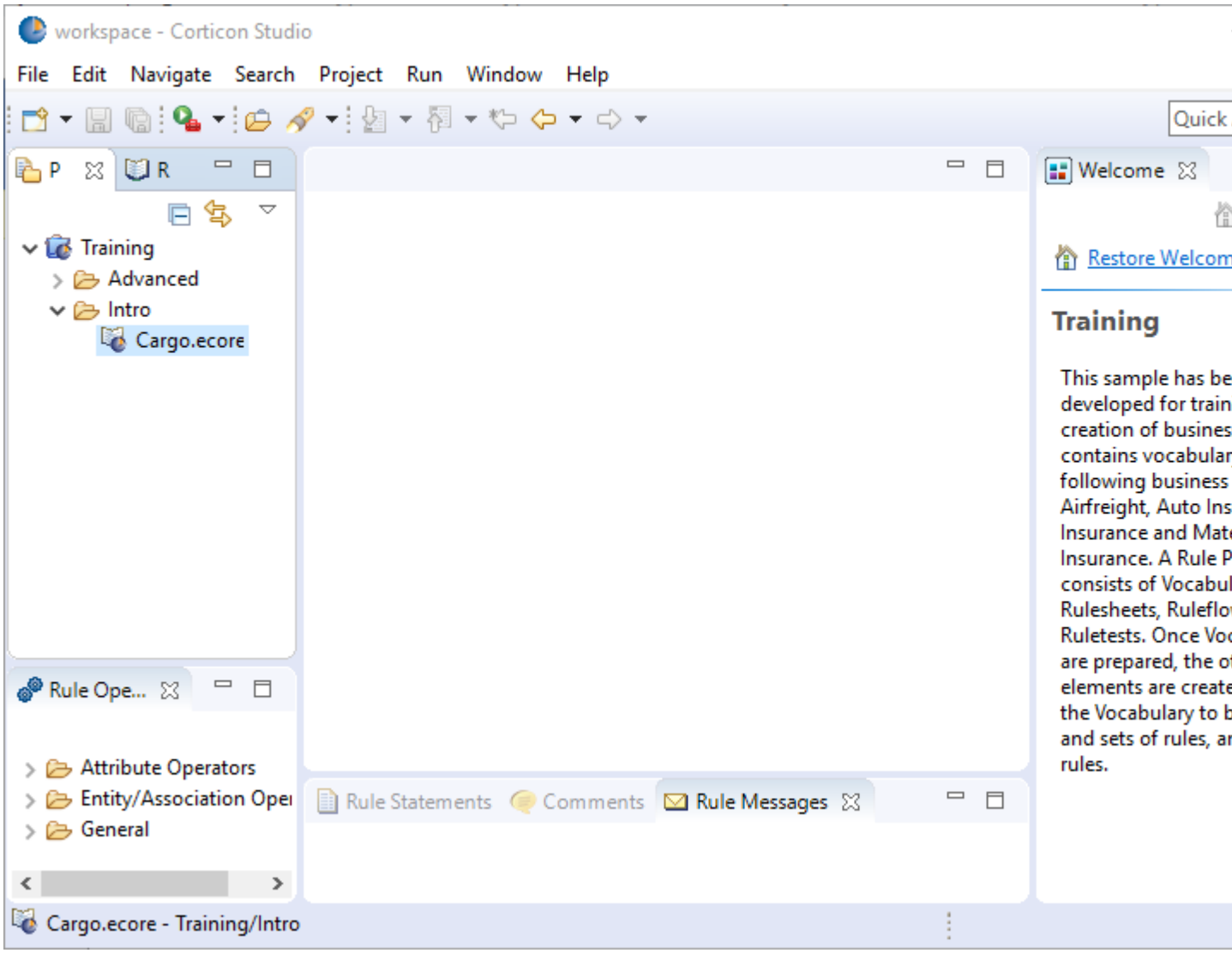
Next, open the sample rule project **Training** which contains the **Cargo** Vocabulary that we will use. As part of this tutorial, you will map this Vocabulary to a database in SQL Server. You will create Rulesheets and Ruletests that read and write to the database.

Follow these steps to open the sample project:

1. Choose the **Start** menu command **Progress > Corticon Studio**.
2. In Studio, select the menu command **Help > Samples.**
3. Choose **Tutorials**:



4. Click **Open**, and then select **Training** and click **OK**.
5. Expand the **Training** project, and you see two folders.
6. Expand the **Intro** folder so you can access the Cargo.ecore located there.



Success! Your environment is now ready to use!

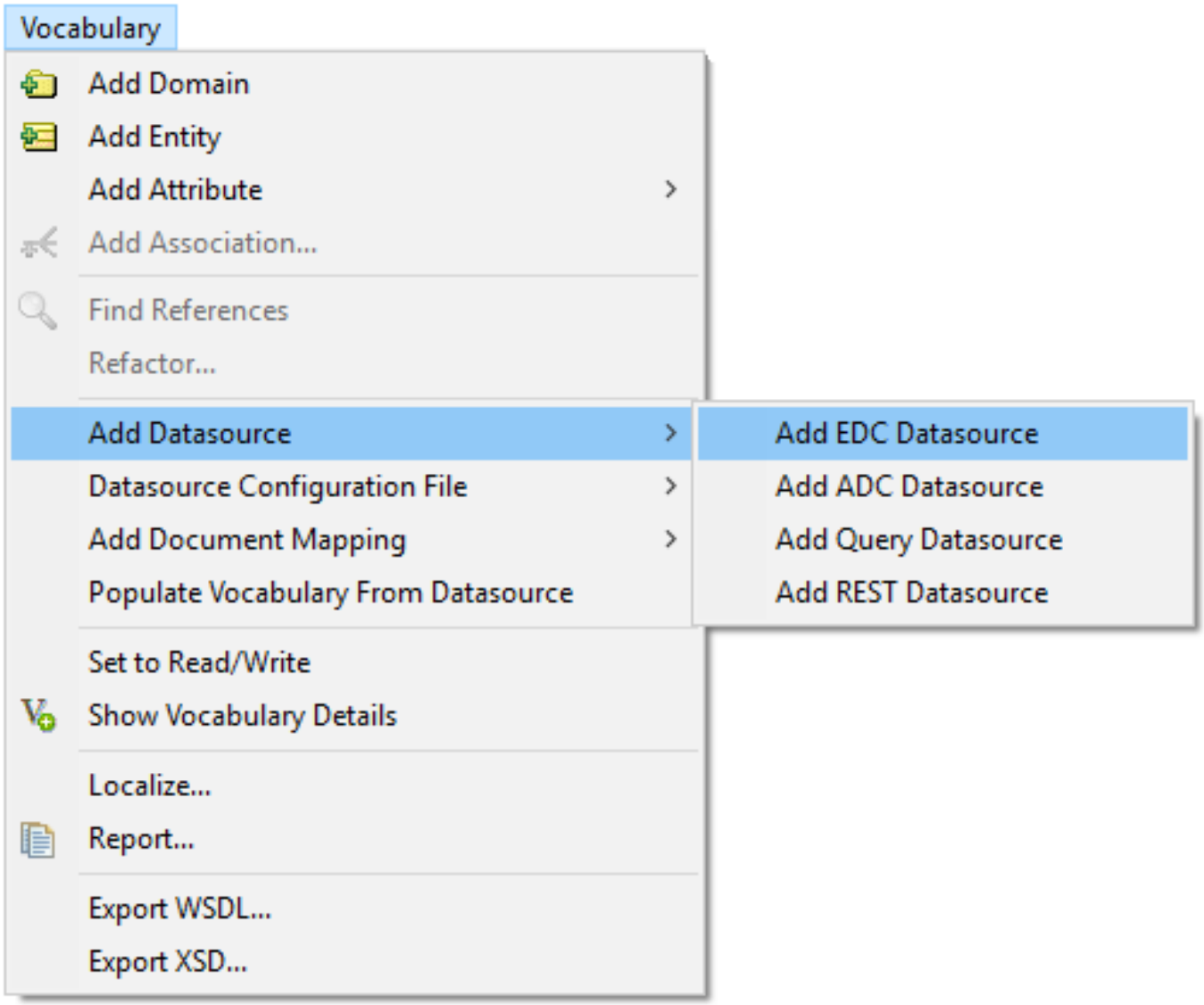
You will use the Cargo Vocabulary to generate a database schema. You will create Rulesheets, and then create Ruletests to do some testing. By running the Ruletests, you can verify that the rules are able to access the database.

Step 5: Adding the EDC Datasource to the Vocabulary

We need to declare that we want to use an EDC Datasource.

In the Studio, open **Cargo.ecore** under **Intro** in the **Training** rule project.

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



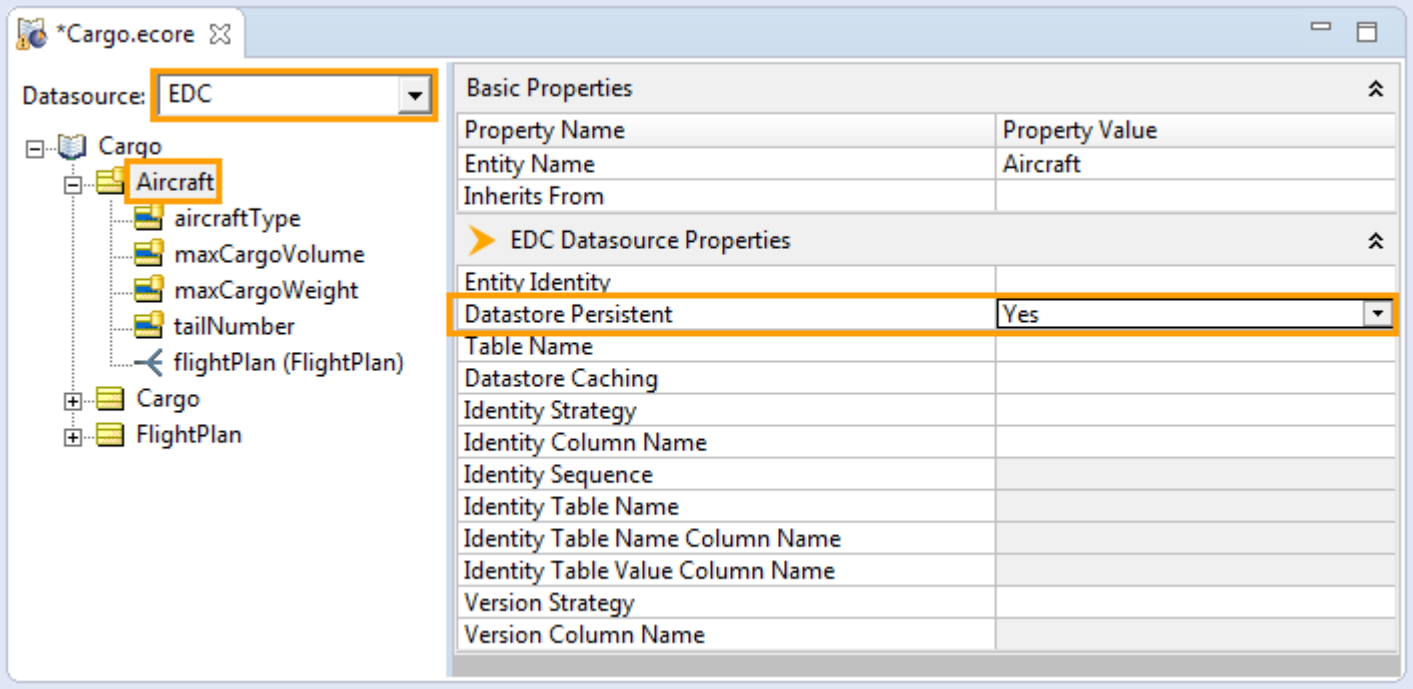
Step 6: Mapping the Vocabulary to the database

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.

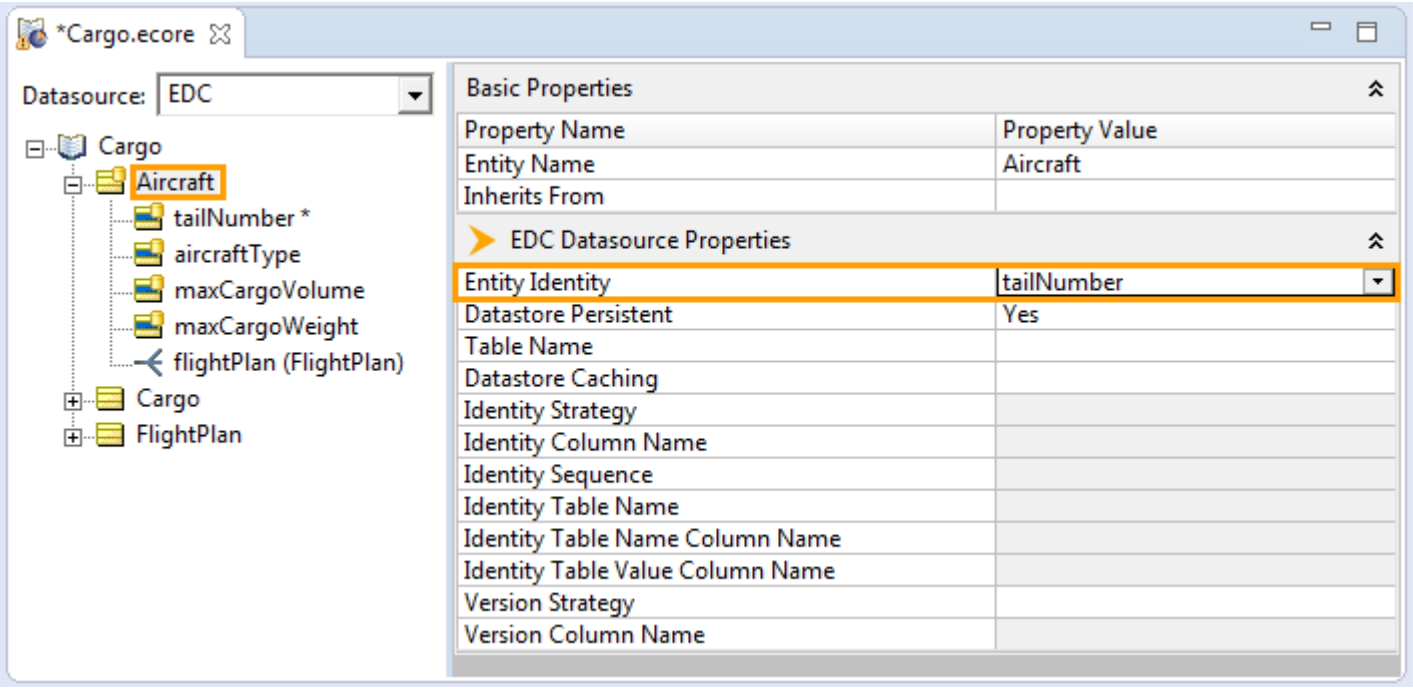
Select the **Aircraft** entity in the Vocabulary. In the **Properties** editor on the right, click the **Datastore Persistent** drop-down, and then select **Yes**.

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 to be stored in a 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**.

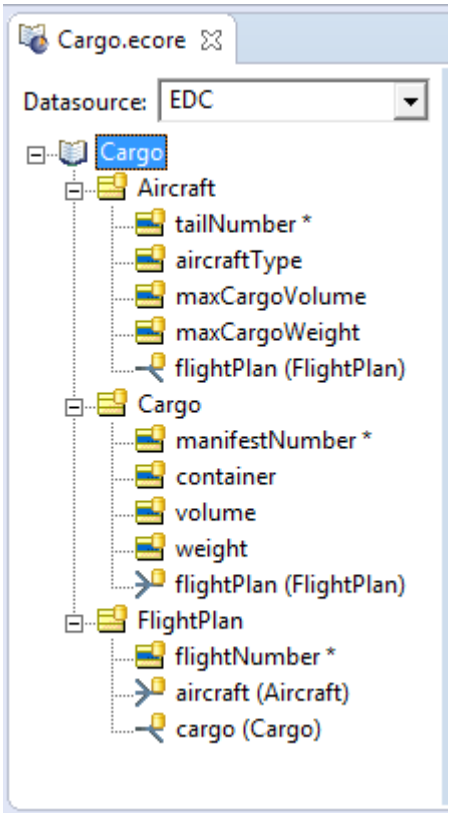


The tailNumber attribute in the Aircraft entity will now be mapped as its primary key, so it moves up to the top of the list of attributes in the entity, and is marked with an asterisk.

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 now looks like this:



Step 7: Establishing the connection to the database

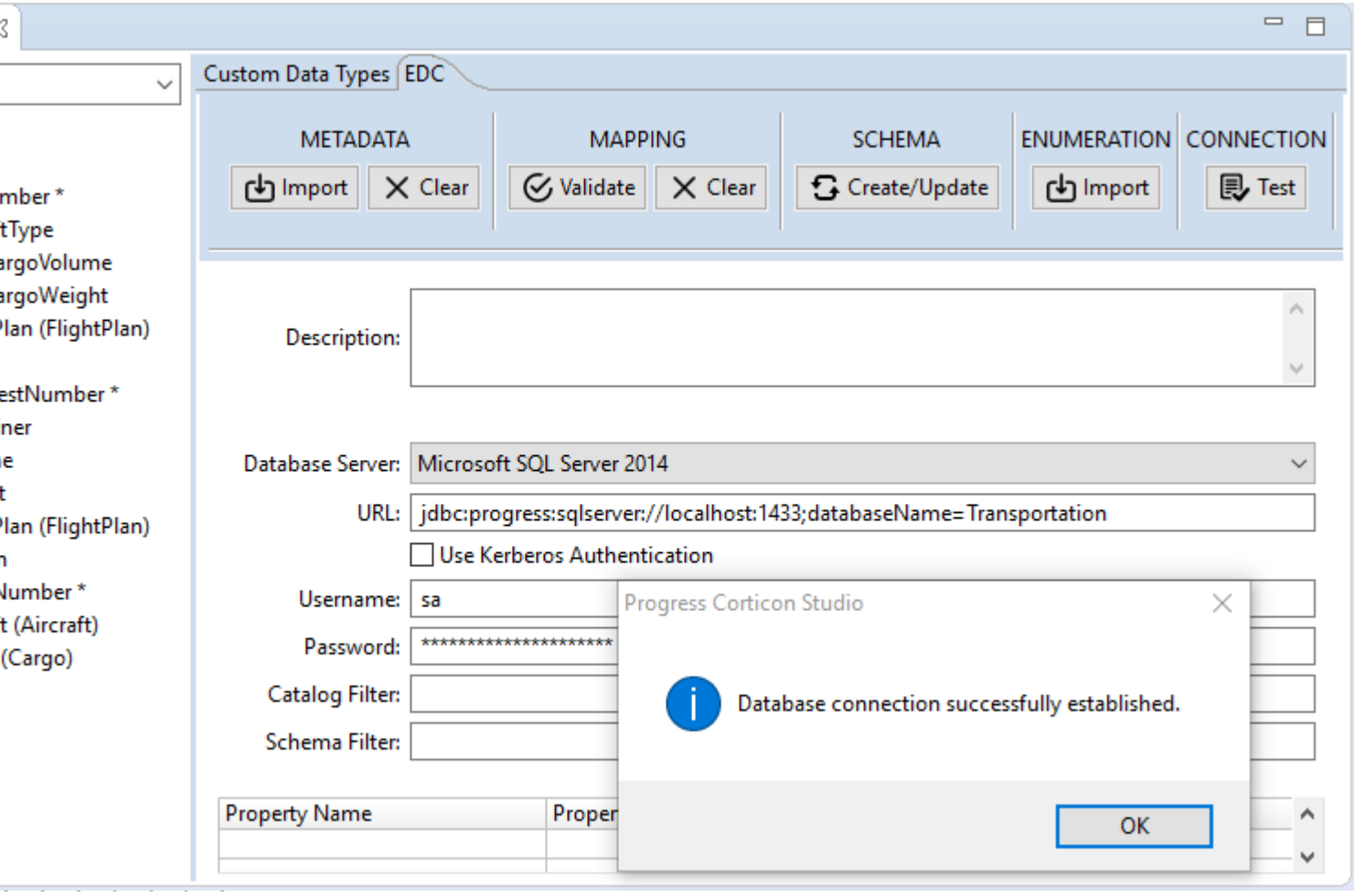
The next step is to define the database connection properties in the Vocabulary. This enables Corticon to connect to the database and generate the schema from the Vocabulary.

First, choose **File > Save** to store what we have done so far. Now, we'll define database connection properties:

1. Select the **Cargo** root node in the Vocabulary tree, and then click the **EDC** tab.
2. Specify the following database properties:
   - Database Server: Microsoft SQL Server 2014
   - Database URL: jdbc:progress:sqlserver://localhost:1433;databaseName=Transportation
   - Username: sa

- Password: **sqlserver2014password**

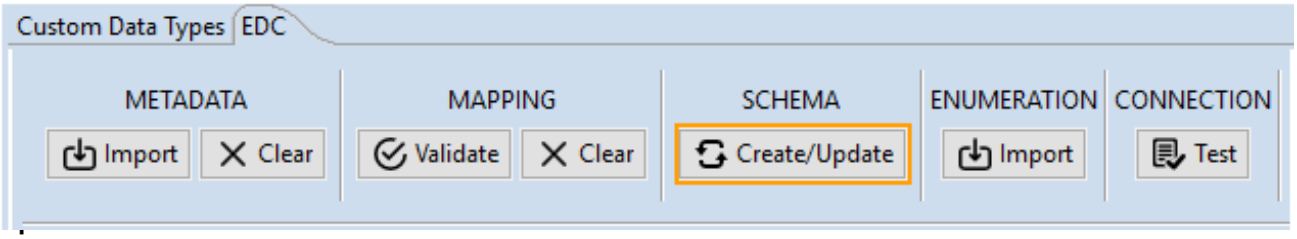
3. Finally, let's test if the Vocabulary is able to connect to the database. Click **Test Connection**. A message indicates that the connection was



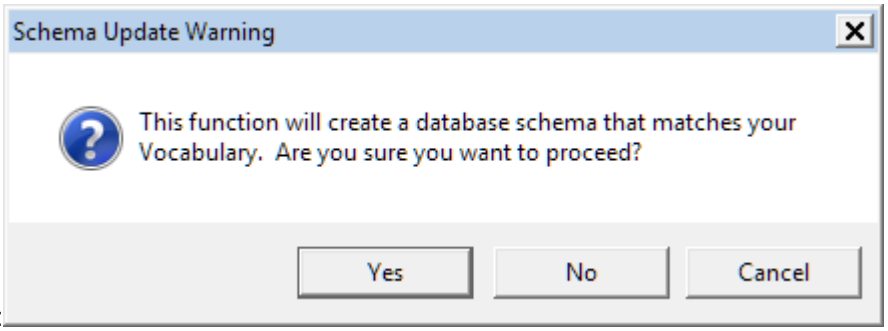
Step 8: Generating the Vocabulary to the database schema

We have a connection, we have defined which entities we want to persist in the database as tables, and which attribute in each entity will be assigned as the primary key. We are ready to generate the database schema.

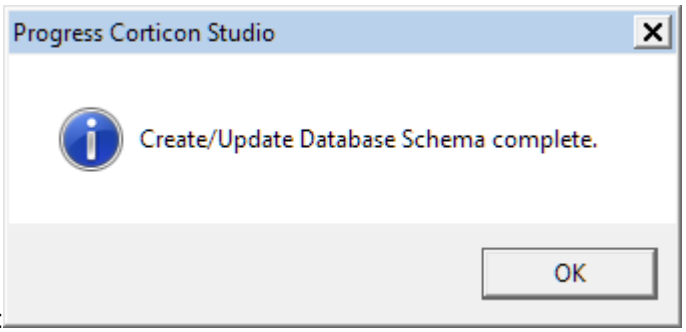
1. On the **EDC** tab, click SCHEMA



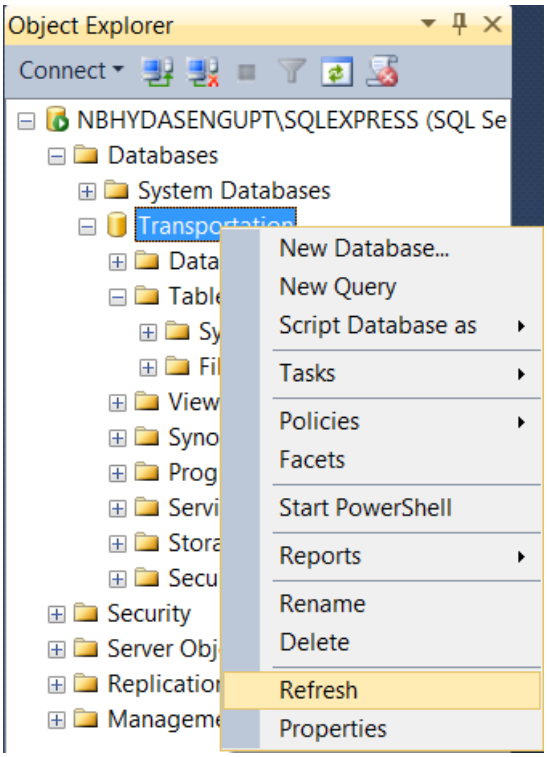
2. Click **Yes** in the **Schema Update Warning** dialog box:



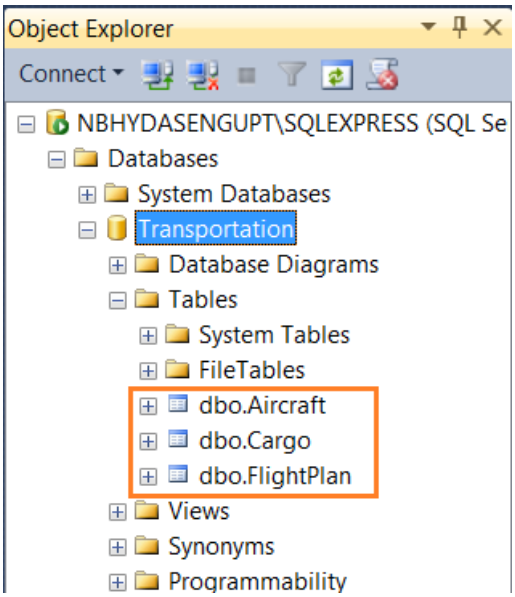
3. When the database schema is created successfully, you see the following message:



4. Verify that a table is created in the database for each entity in the Vocabulary. In **SQL Server Management Studio**, right-click **Transportation** and select **Refresh**.



5. Expand **Transportation > Database**. You see the following tables:

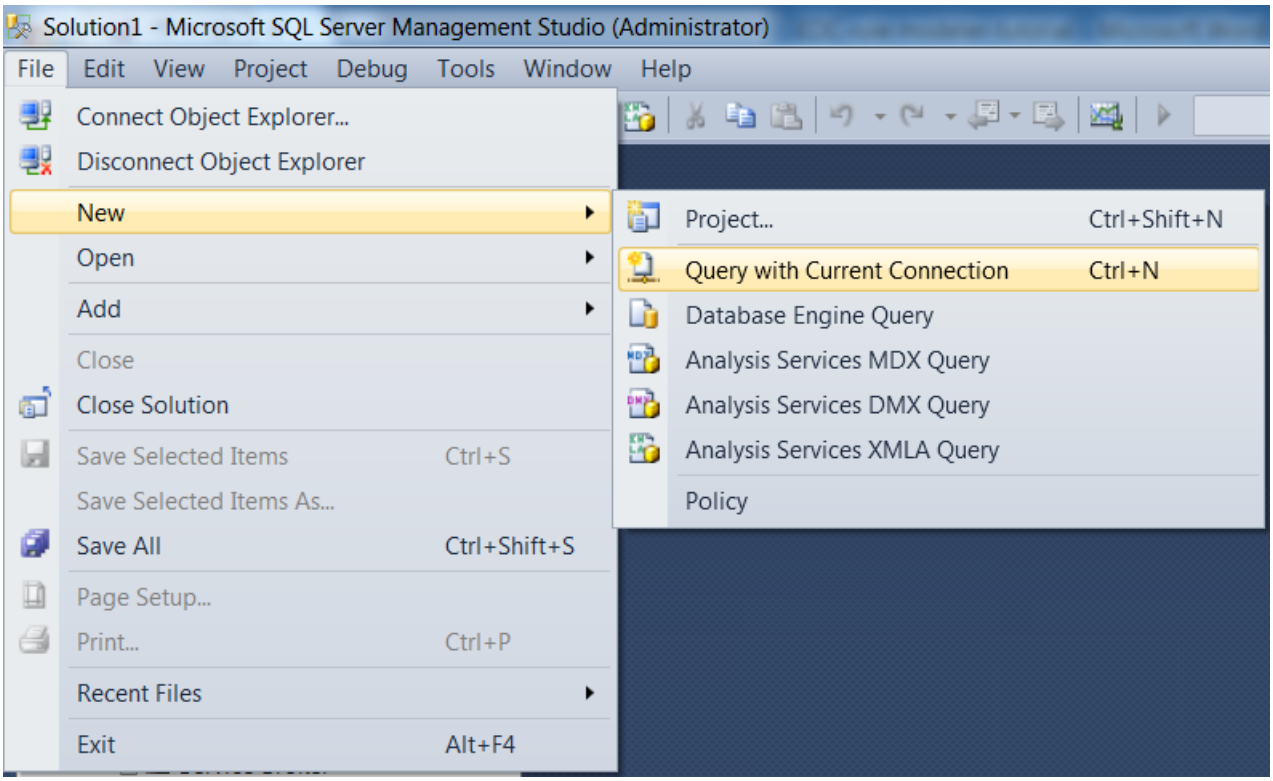
6. 

Step 9: Populating the database with sample data

The last step is to populate some sample data so that when you define rules that read from the **Transportation** database, there are sample records that the rules can process.

Follow these steps to populate sample data:

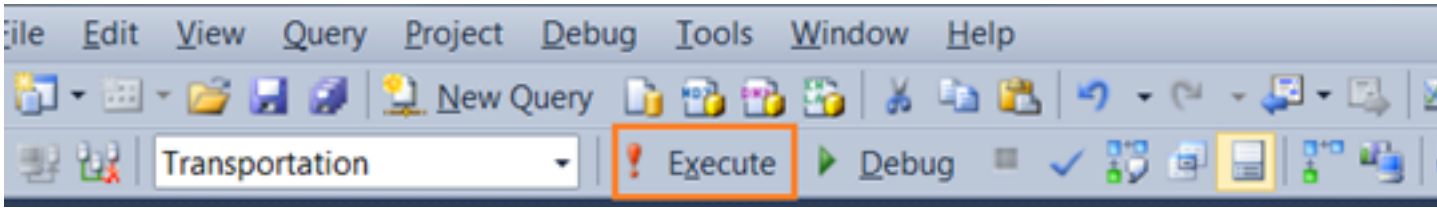
1. In **SQL Server Management Studio**, select **File > New > Query with Current Connection**. A new SQL Script file opens in a text editor.



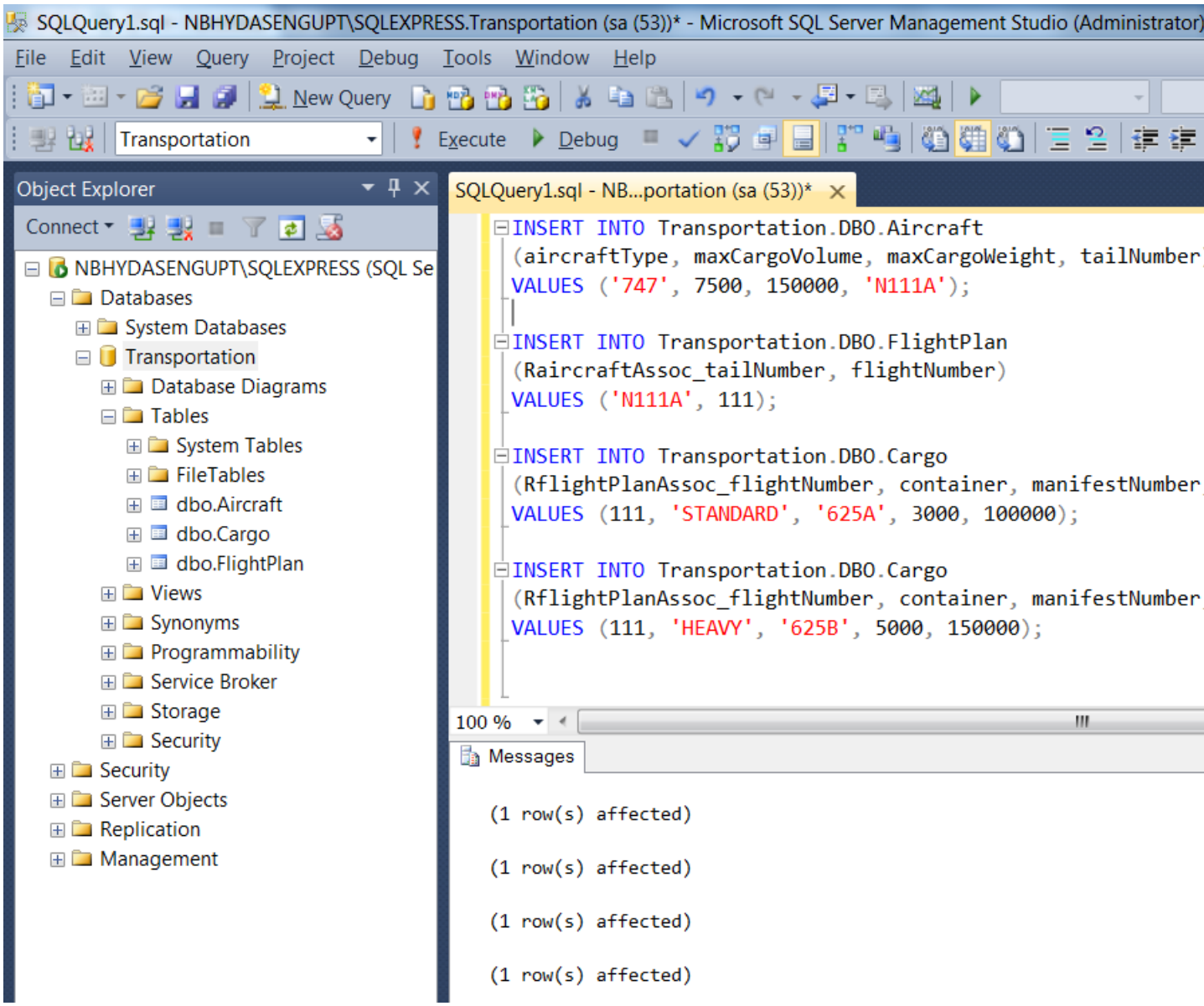
2. Copy and paste these lines of code into the text editor:

```
INSERT INTO Transportation.DBO.Aircraft
(aircraftType, maxCargoVolume, maxCargoWeight, tailNumber)
VALUES ('747', 7500, 150000, 'N111A');

INSERT INTO Transportation.DBO.FlightPlan
(RaircraftAssoc_tailNumber, flightNumber)
VALUES ('N111A', 111);

INSERT INTO Transportation.DBO.Cargo
(RflightPlanAssoc_flightNumber, container, manifestNumber, volume, weight)
VALUES (111, 'STANDARD', '625A', 3000, 100000);

INSERT INTO Transportation.DBO.Cargo
(RflightPlanAssoc_flightNumber, container, manifestNumber, volume, weight)
VALUES (111, 'HEAVY', '625B', 5000, 150000);
```

3. Click **Execute**.



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



Success! Your environment is now ready to use!

3

Reading records from a database using a primary key

Once entities and attributes in a Vocabulary are mapped to a database, any rule that uses those entities or attributes automatically accesses the database's tables.

When it comes to reading records, there are two high-level scenarios:

- When you need to retrieve a specific record. To do this, the rule needs to receive a primary key value in input data and use it to retrieve the record.
- When you need to retrieve all records from multiple tables and compare record values across tables. In this case, you do not need a primary key.

Let's take a look how to model a rule to retrieve a record using a primary key.

For details, see the following topics:

- Modeling the rules
- Testing the rules

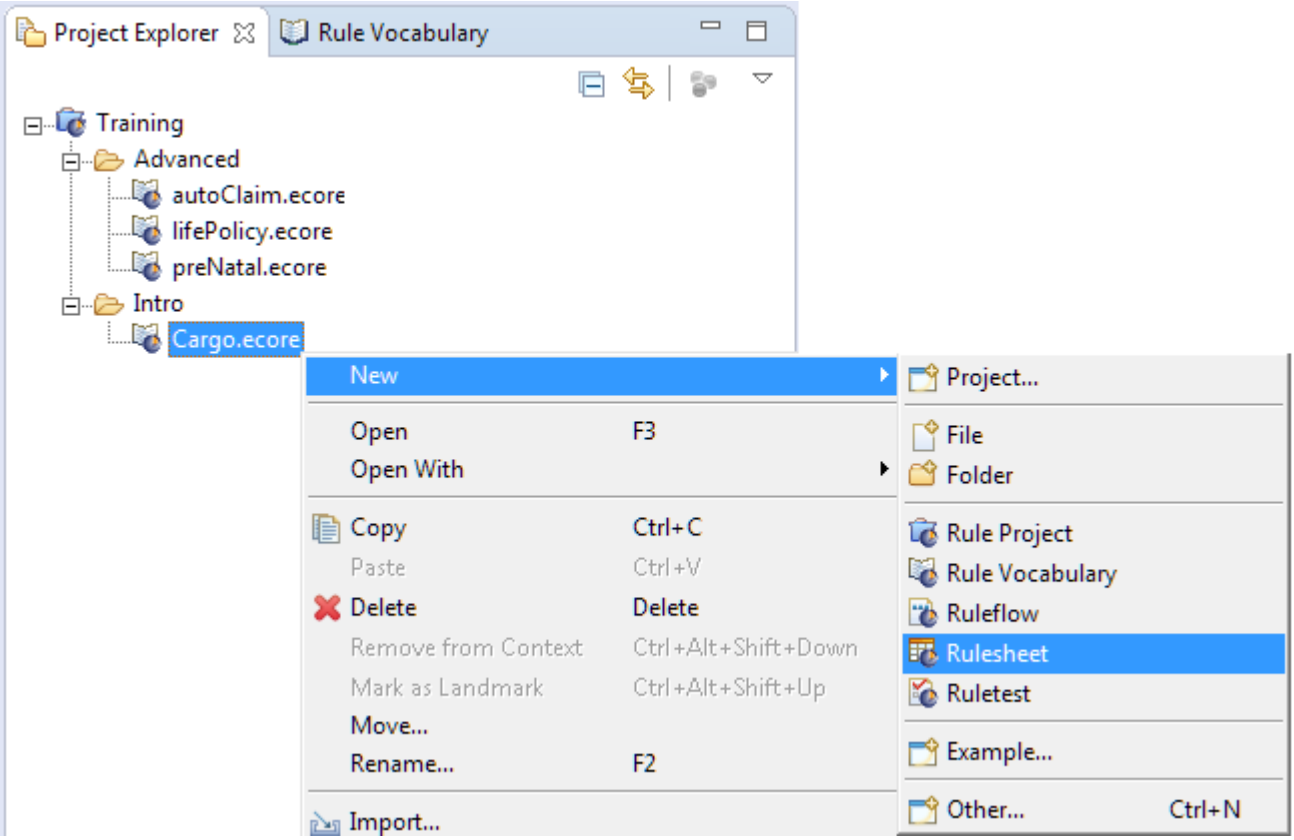
Modeling the rules

In this example, we added data to the database for a flight plan for two cargo containers. An aircraft has been assigned to this flight plan.

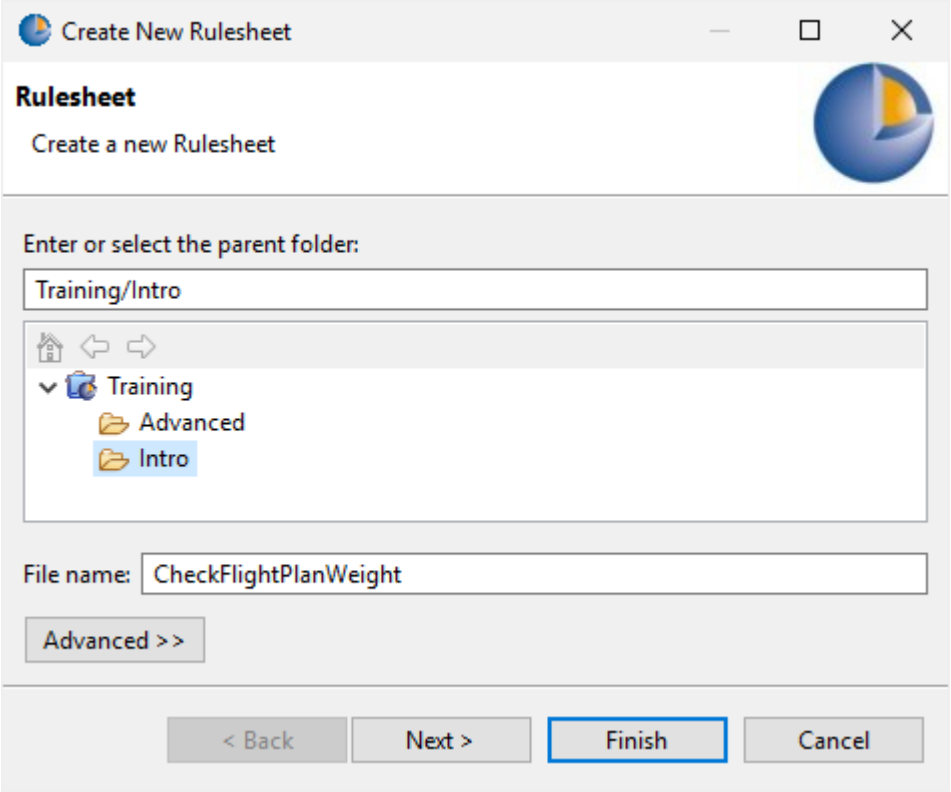
You need to model rules to check if the flight plan is valid by comparing the total of the cargo weights with the maximum cargo weight of the aircraft as follows:

- If the total cargo weight is less than the maximum cargo weight of the assigned aircraft, the rule should throw an Info message.
- If the total cargo weight exceeds the maximum cargo weight of the assigned aircraft, the rule should throw a Violation message.

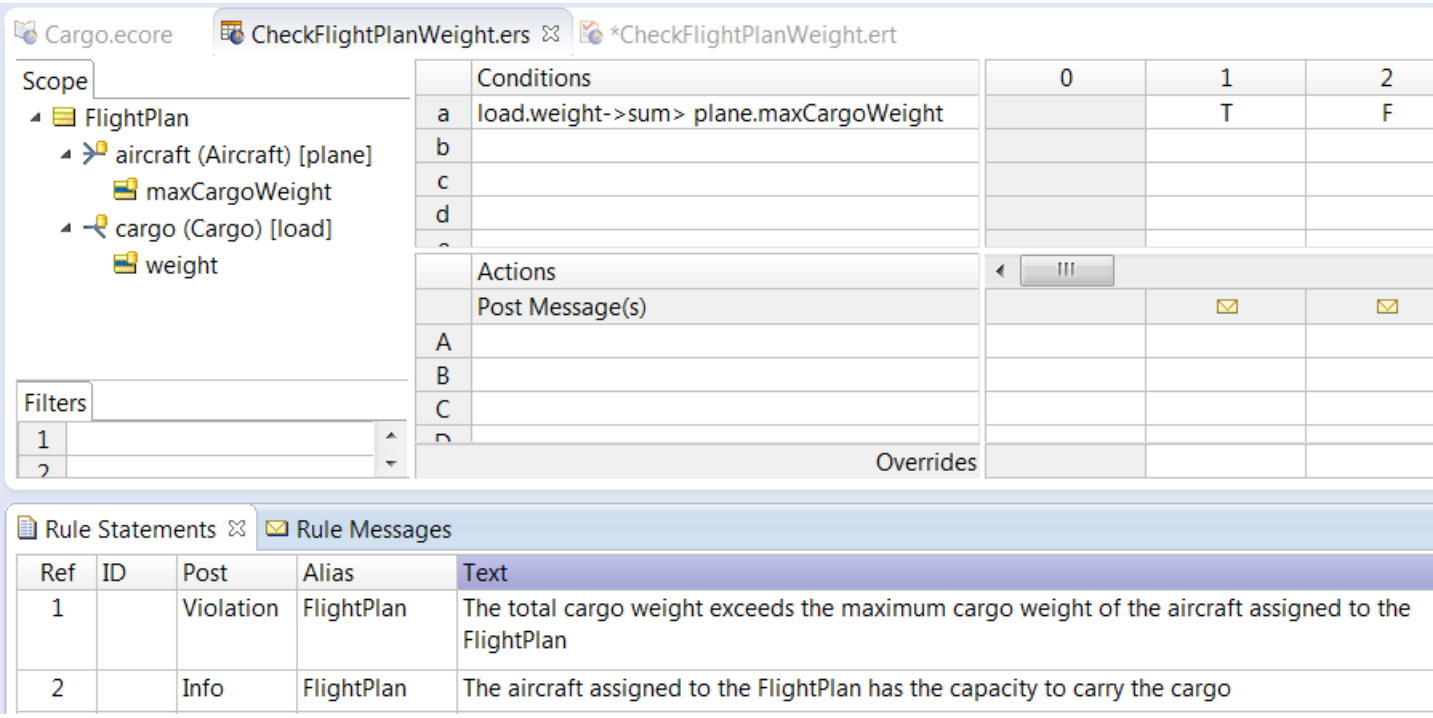
We need to create a Rulesheet by right-clicking on Cargo.ecore, and then choosing **New > Rulesheet**.



Name the Rulesheet **CheckFlightPlanWeight**, and then click **Finish**.



Fill in the Rulesheet as illustrated:



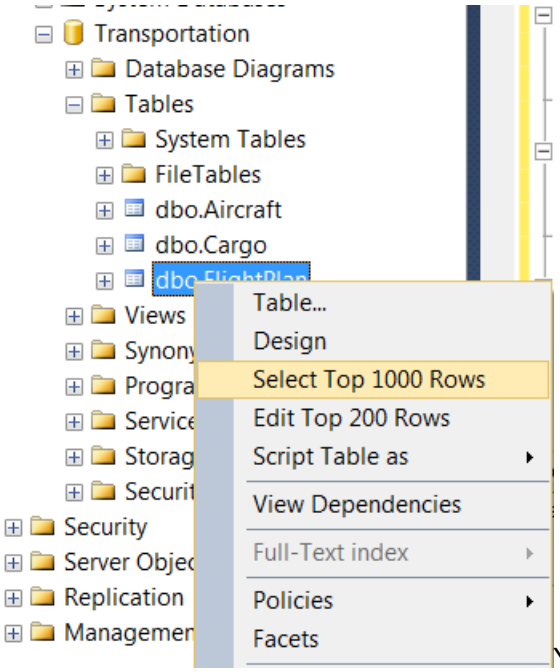
Here are the steps for this task:

1. Choose the menu option **Rulesheet > Advanced View**.
2. From the Vocabulary, drag and drop to the Scope panel:

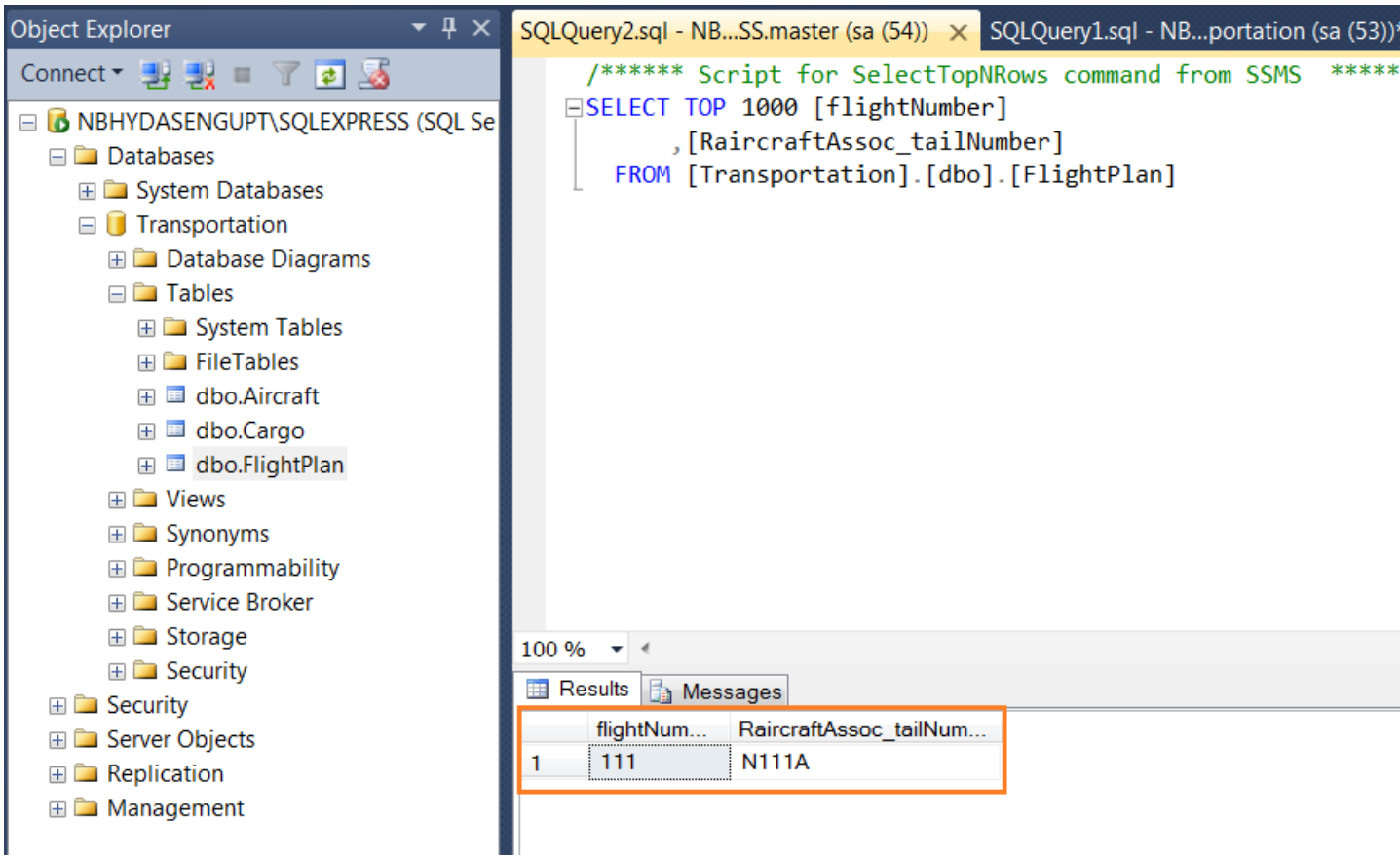
   a. FlightPlan
   b. The FlightPlan association **aircraft** onto FlightPlan in the Scope, and then double click on it to open its alias entry box where you enter **plane.**
   c. The Aircraft attribute **maxCargoWeight** onto aircraft in the Scope.
   d. The FlightPlan association **cargo** onto FlightPlan in the Scope, and then double click on it to open its alias entry box where you enter **load.**
   e. The Cargo attribute **weight** onto cargo in the Scope.

3. Write the condition load.weight->sum>plane.maxCargoWeight
4. Select **T** in column 1 and **F** in column 2.
5. Enter the rule statements as shown.
6. Save the file.

Let's look at the sample data that we want to retrieve. In SQL Server Management Studio, right-click the **dbo.FlightPlan** table and select **Select Top 1000 Rows**.



You should see the following result:



This retrieves one record—a FlightPlan with a flightNumber (the primary key of FlightPlan) set to 111, that is associated with an aircraft whose tailNumber is N111A.

Let's look at the Aircraft records. Right-click the **dbo.Aircraft** table and select **Select Top 1000 Rows**. This should return one record:

Results | Messages

	tailNumb...	aircraftTy...	maxCargoVolu...	maxCargoWei...
1	N111A	747	7500.000000	150000.000000

Note that the tailNumber (which is Aircraft's primary key) **N111A**, corresponds to the tailNumber of the associated aircraft in the FlightPlan's record. Also, notice that the maximum cargo weight is **150000**.

Now, let's look at the cargo records. Right-click the **dbo.Cargo** table and select **Select Top 1000 Rows**. You should see the following records:

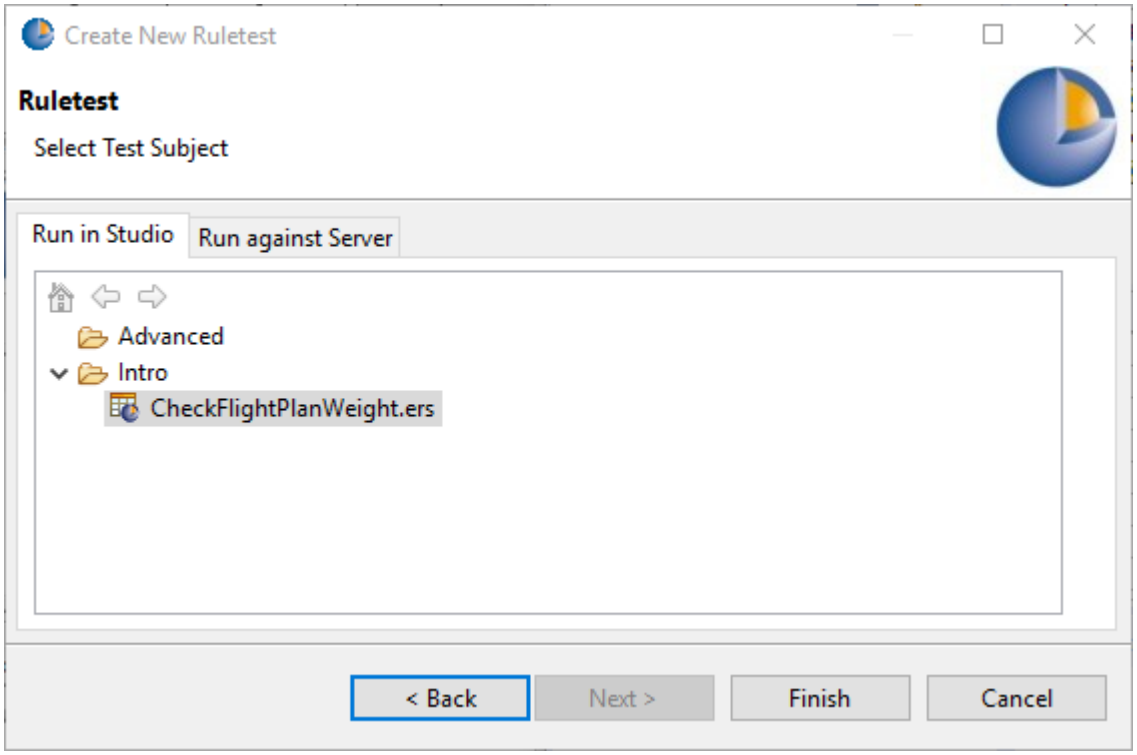
Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111

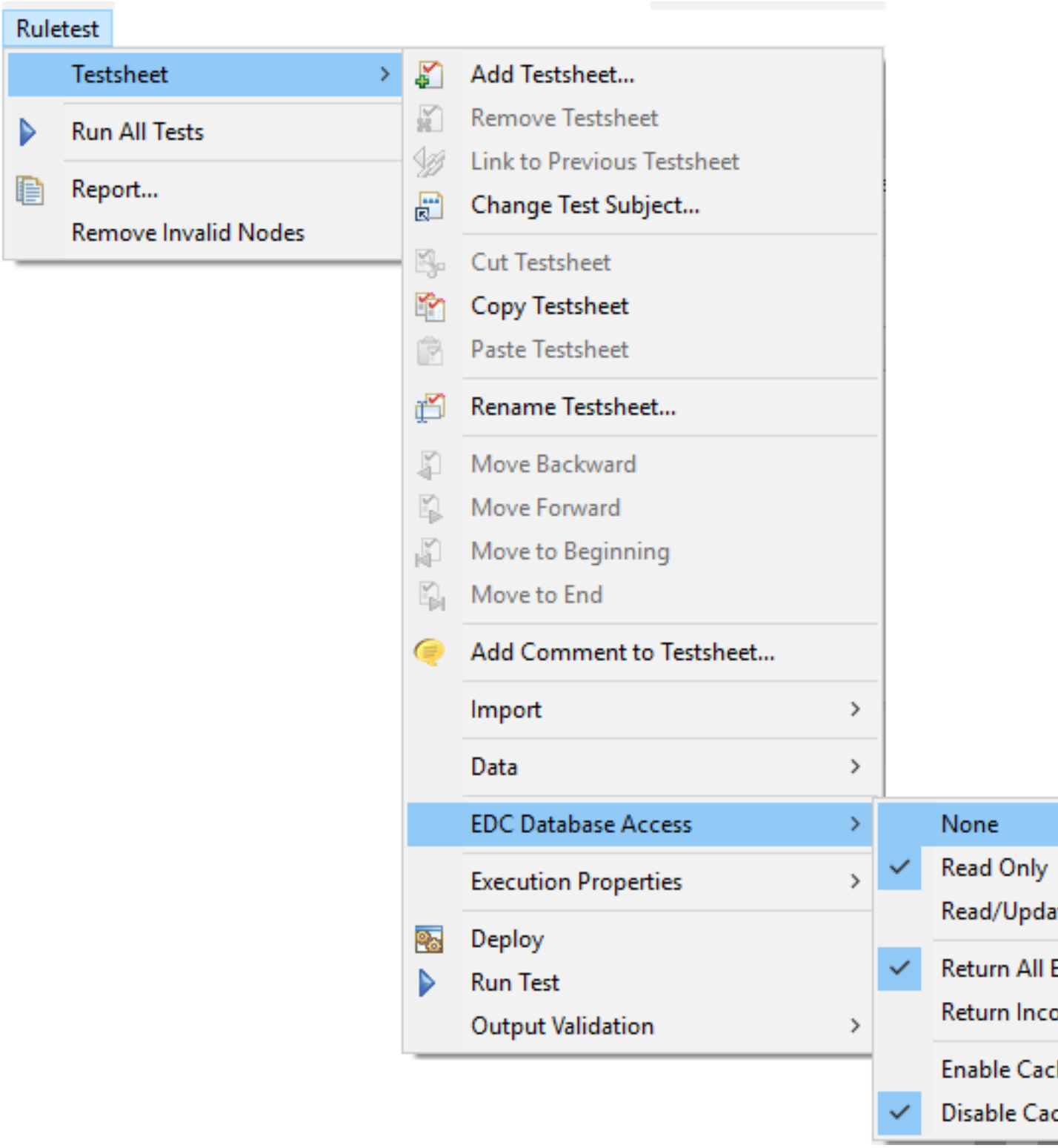
Note that in both records, the associated flightNumber is 111, which is same as the flightNumber of the FlightPlan record retrieved earlier. The sum of the cargo weight is 250000, which exceeds the maximum cargo weight of the Aircraft record.

Testing the rules

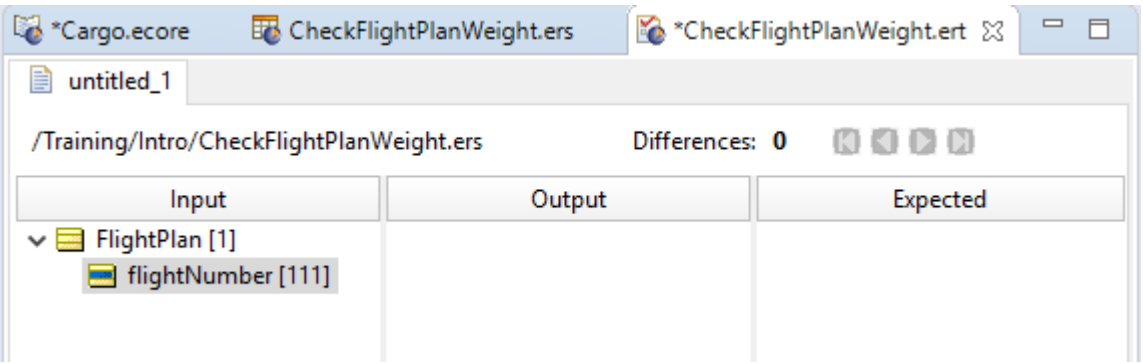
Let's test the Rulesheet. Create a Ruletest named **CheckFlightPlanWeight.ert** that uses CheckFlightPlanWeight.ers as its test subject running in Studio.



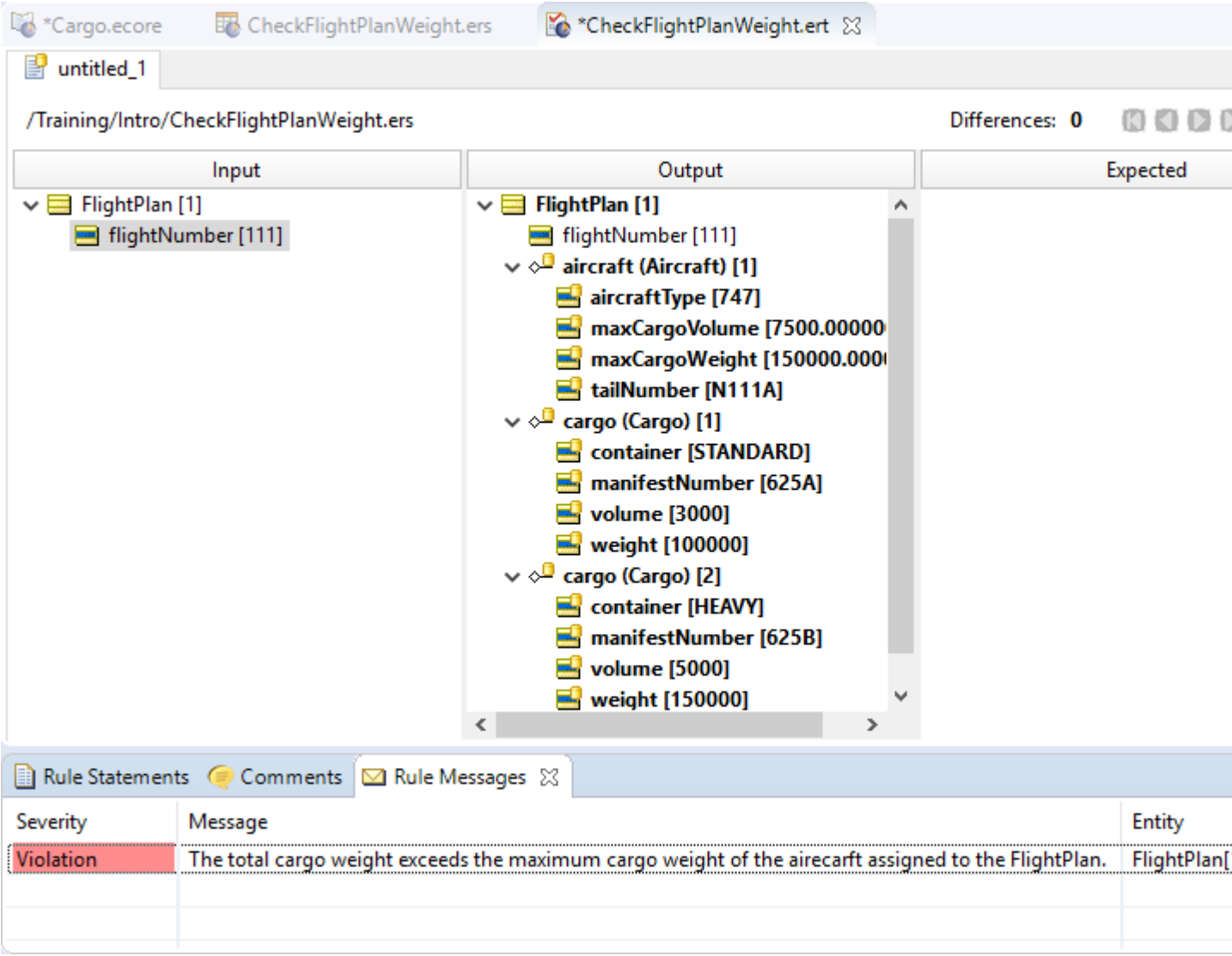
By default, a Ruletest is not configured to read or update a database. You need to change its Database Access setting before you run the Ruletest. Select **Ruletest > Testsheet > Database Access > Read Only**. As you can see below, there is also a **Read/Update** setting that enables you to configure the Ruletest to update the database.



Next, let's define the input. Drag and drop the **FlightPlan** entity from the Rule Vocabulary view to the **Input** pane and specify **111** as the value for flightNumber.



Run the Ruletest. You get the following output:



As you can see, based on the flightNumber (111), Corticon:

1. Retrieved data about the aircraft and cargo associated with the flight plan from the database tables.
2. Calculated the total of the cargo weights.
3. Compared the value of the total of the cargo weights with the maxCargoWeight of the aircraft.
4. Generated a message—in this case, a Violation message, which is displayed in the Ruletest.

Associations play an important role in how a rule reads from a database. As you have seen in this example, if a rule uses a primary key to retrieve records, it retrieves records of not only the entity table to which the primary key belongs, but also records of associated entities.

You have now modeled and tested rules that read a record from a database using a primary key.

4

Reading records from a database without a primary key

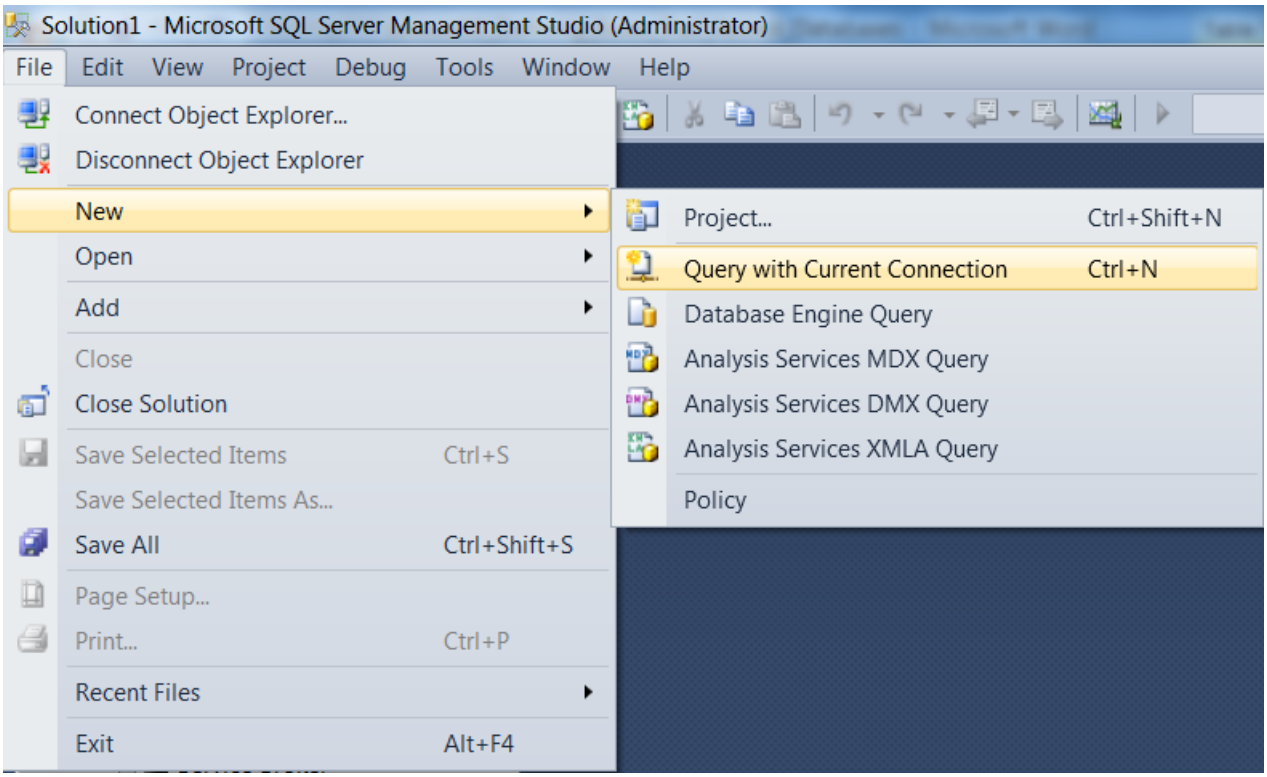
You may also have cases where you need to retrieve all records from multiple tables and compare record values across tables. In this case, you do not need a primary key. Let's look at this scenario.

In this example, assume that only one cargo container can be assigned to a flight plan. You need to model a rule that compares the weight of each cargo container with the maximum cargo weight of each Aircraft to determine which cargo-aircraft combinations are valid. As you know, records about different Cargo containers and different Aircraft are stored in database tables. So the rule must retrieve all Cargo records from the Cargo table, retrieve all Aircraft records from the Aircraft table, and compare the cargo weight of each Cargo record with the maximum cargo weight of each Aircraft record. In this case, using a primary key value is not applicable.

Before modeling the rule, let's add a few more Cargo and Aircraft records to make this example more interesting.

Adding more sample data

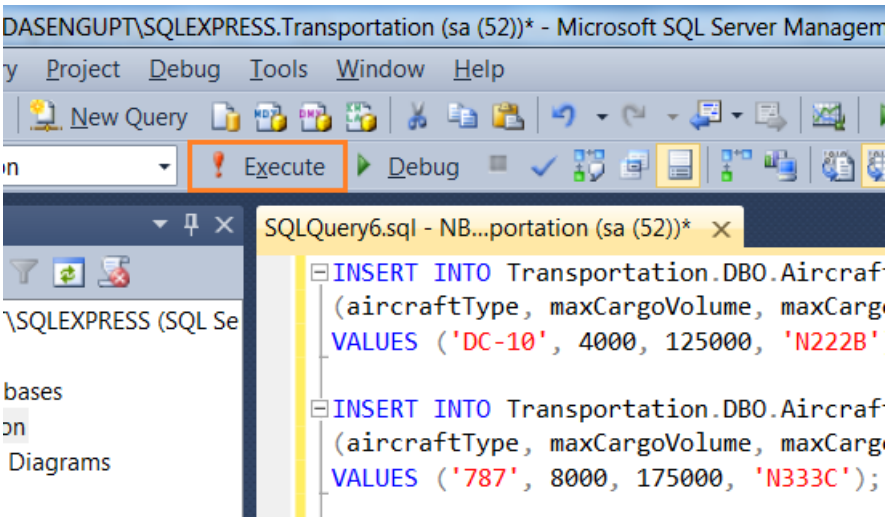
1. In SQL Server Management Studio, create a new SQL script file by selecting **File > New > Query with Current Connection**.



2. Copy and paste these lines of code into the SQL script editor:

```
INSERT INTO Transportation.DBO.Aircraft
(aircraftType, maxCargoVolume, maxCargoWeight, tailNumber)
VALUES ('DC-10', 4000, 125000, 'N222B');

INSERT INTO Transportation.DBO.Aircraft
(aircraftType, maxCargoVolume, maxCargoWeight, tailNumber)
VALUES ('787', 8000, 175000, 'N333C');

INSERT INTO Transportation.DBO.Cargo
(RflightPlanAssoc_flightNumber, container, manifestNumber, volume, weight)
VALUES (NULL, 'STANDARD', '625C', 4000, 125000);
```

3. Click **Execute**.



4. You should see messages indicating that new rows have been added.

Messages

```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```

5. Let's take a quick look at these records. Right-click on **Databases > Transportation > Tables > dbo.Aircraft** and select **Select Top 1000 Rows**. You now see the following: Two new Aircraft records were added.

Results | Messages

	tailNumb...	aircraftTy...	maxCargoVolu...	maxCargoWei...
1	N111A	747	7500.000000	150000.000000
2	N222B	DC-10	4000.000000	125000.000000
3	N333C	787	8000.000000	175000.000000

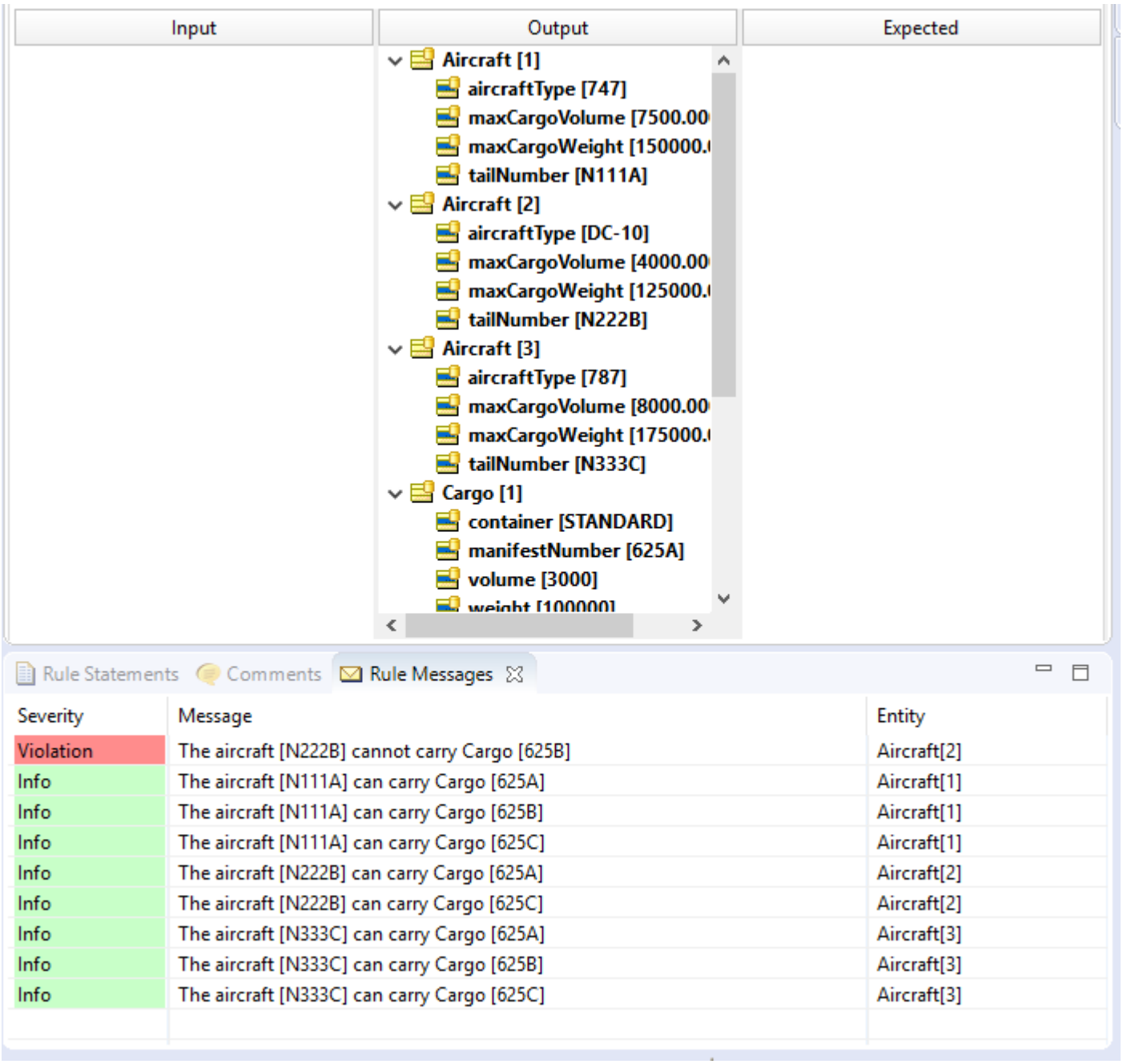
6. Right-click **dbo.Cargo** and select **Select Top 1000 Rows**. You now see the following: Two new Cargo records were added.

Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625C	STANDARD	4000	125000	NULL

Modeling the rules

Create a Rulesheet named **Cargo_Aircraft.ers** and model the rules as shown in this Rulesheet:



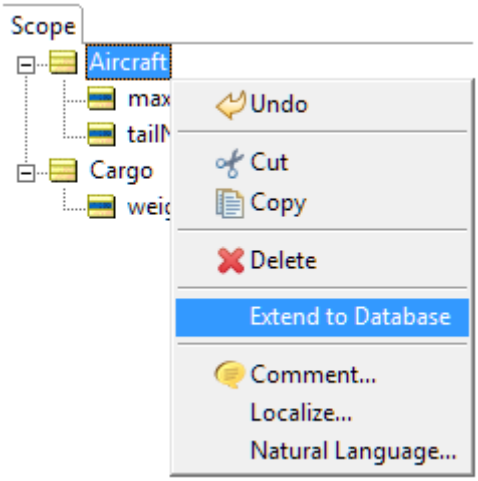
Here are the steps for this task:

1. From the Vocabulary, drag and drop the Cargo attribute **weight** to the condition line 1 panel.
2. Add **>** to the expression.
3. Drag the Aircraft attribute **maxCargoWeight** to complete the expression.
4. Select T in column 1 and F in column 2.
5. Enter the rule statements as shown.
6. Save the file.

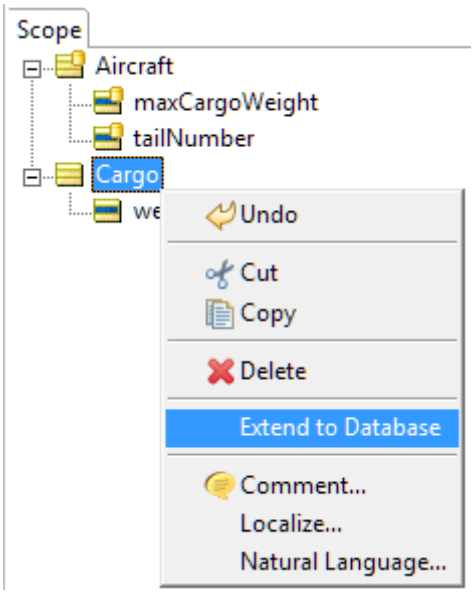
Note that **Aircraft.tailNumber** and **Cargo.manifestNumber** are referenced in the rule statements by enclosing each attribute with [{ }]. That will dynamically retrieve those values in the test so that we can see which Cargo record is being compared with which Aircraft record.

When a rule uses a primary key, it retrieves a limited set of data from the database, which is loaded into memory for processing. However, without a primary key, if a rule has to retrieve multiple records, using root-level entities—for example **Cargo** and **Aircraft**—**ALL** records from the tables must be loaded into memory. Since this takes up a lot of memory, retrieving all root-level entity records is disabled by default. To retrieve all records from tables, you must 'extend' the entities used in the rule to the database. To extend an entity to the database:

1. Switch the Rulesheet to Advanced View by selecting **Rulesheet > Advanced View**.
2. Right-click **Aircraft** in the Scope pane and select **Extend to Database**:



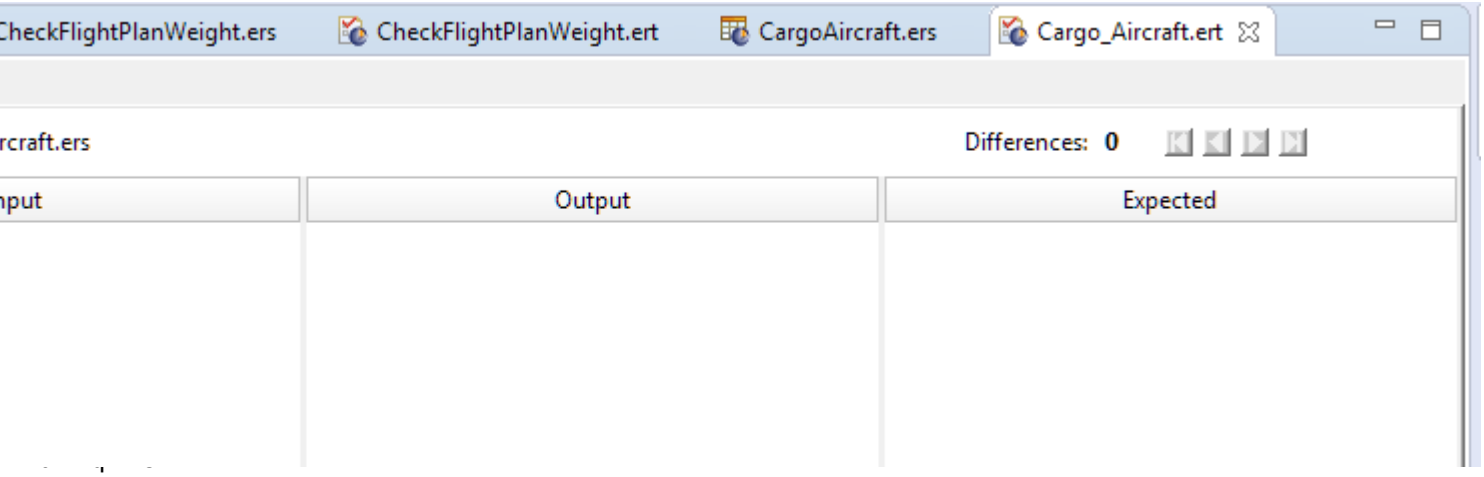
3. The icons of the Aircraft entity and its attributes in the Scope pane change to include the database icon indicating that the entity is extended to the database.
4. Then, right-click **Cargo** in the Scope pane and select **Extend to Database:**



Your Rulesheet is now ready for the test we want to run.

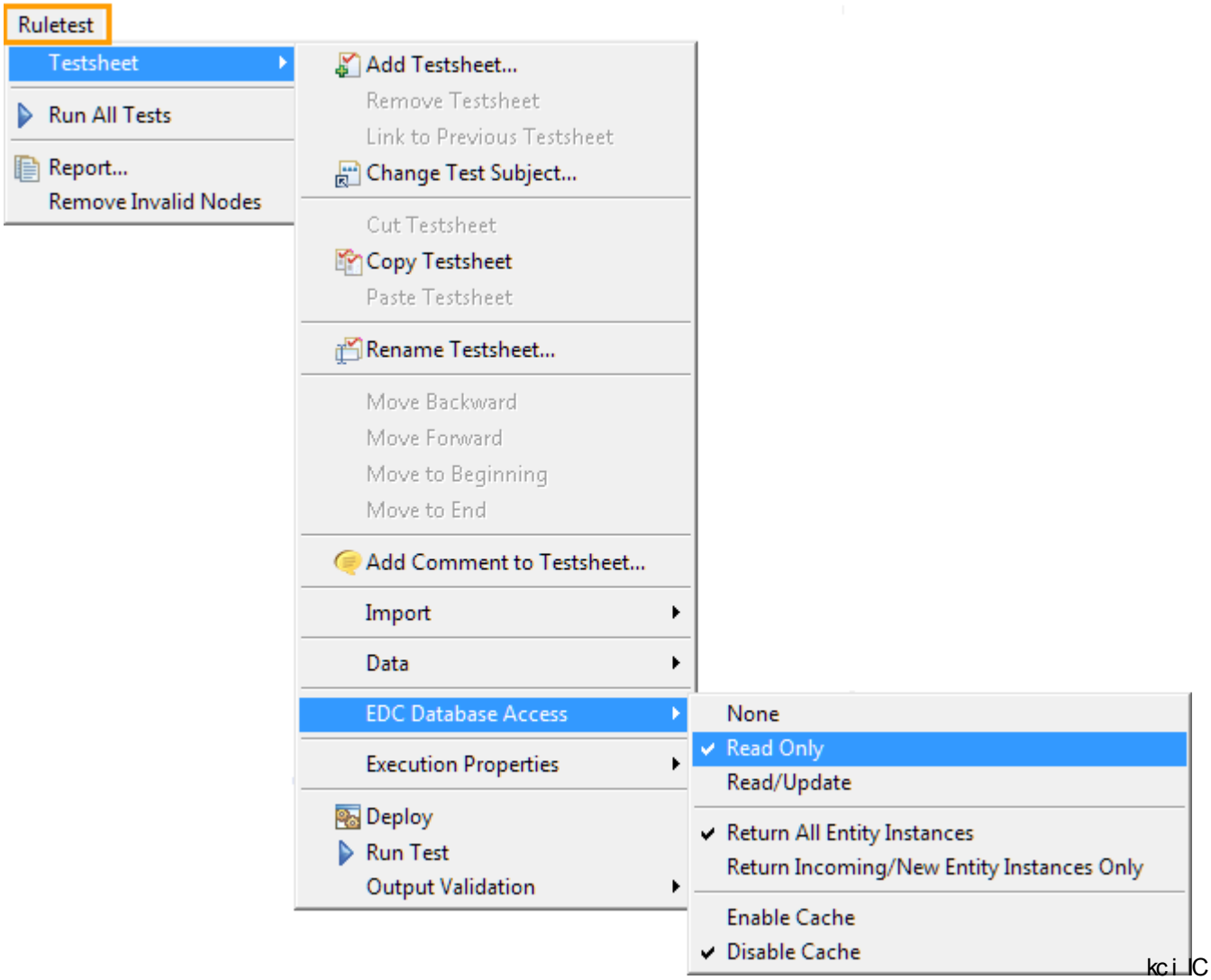
Testing the rules

Create a Ruletest named **Cargo_Aircraft.ert** that uses Cargo_Aircraft.ers as its test subject. The Ruletest must retrieve ALL records from the Cargo and Aircraft tables. So, no input is



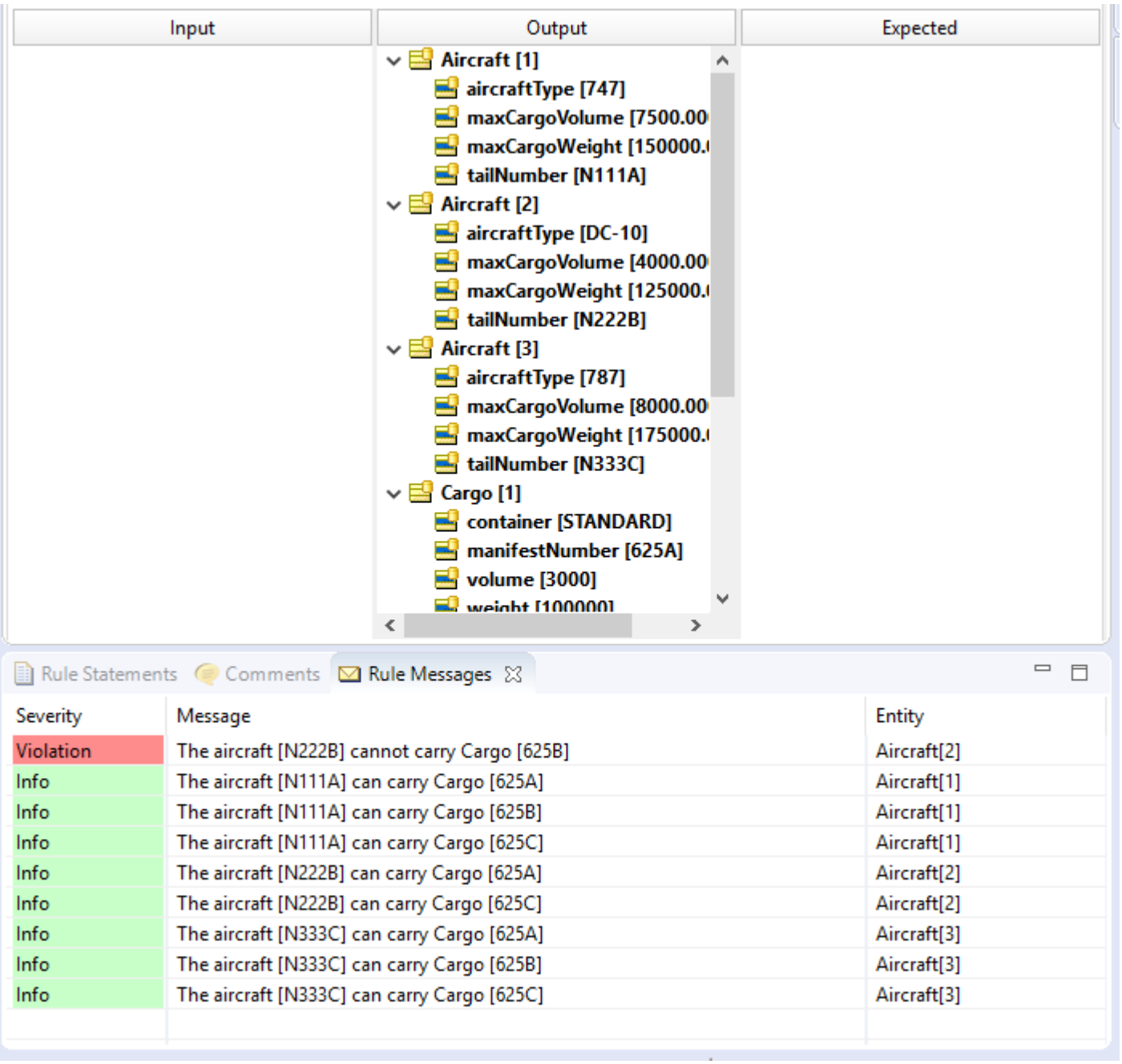
You only need to run the Ruletest.

Since this Ruletest needs to read records from the database, configure the Database Access setting as Read Only:



Run Test to execute the Ruletest.

You should see the following output:



As you can see, the nine permutations of the three aircraft and the three manifests were passed through the rules. The value of Cargo.weight of each Cargo record has been compared with the value of Aircraft.maxCargoWeight of each Aircraft record. The rule messages indicate which Cargo-Aircraft combinations are valid – one failed the test and has been flagged through a Violation message.

5

Writing to the database

Now, that you have an understanding of how to model rules that read from a database, let's look at how to model rules that write to a database. There are two scenarios in which you write to a database:

- You need to model a rule that adds a new record
- You need to model a rule that updates an existing record

Similar to reading records, if the Vocabulary is mapped to a database, any rule that uses the Vocabulary can perform write operations on the database. For this to work, the database needs the primary key. The primary key value can be specified in the following ways:

- If a new record needs to be added:
  - The primary key can come from Corticon—either supplied through an input message or defined in a rule, or,
  - The database can be configured to assign the primary key through identities (in Microsoft SQL Server) or sequences (in Oracle). Note: This way of assigning a primary key is not covered in this tutorial. To learn more, see the "Identity Strategies" section of the Data Integration guide.
- If an existing record needs to be updated, the primary key must come from Corticon—either supplied through an input message or defined in a rule.

For details, see the following topics:

- Adding a new record to the database
- Updating an existing record
- Adding a record using a primary key defined in a rule

Adding a new record to the database

In this example, let's assume that you need to model rules that assign a value to Cargo.container based on the value of Cargo.weight received in input.

As you know, the Cargo table in the Transportation database contains three records:

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625C	STANDARD	4000	125000	NULL

To add a new record, the value of its manifestNumber (the primary key), must be unique.

Create a Rulesheet named **AddRecord.ers** that uses the Cargo.ecore Vocabulary and model rules as shown here:

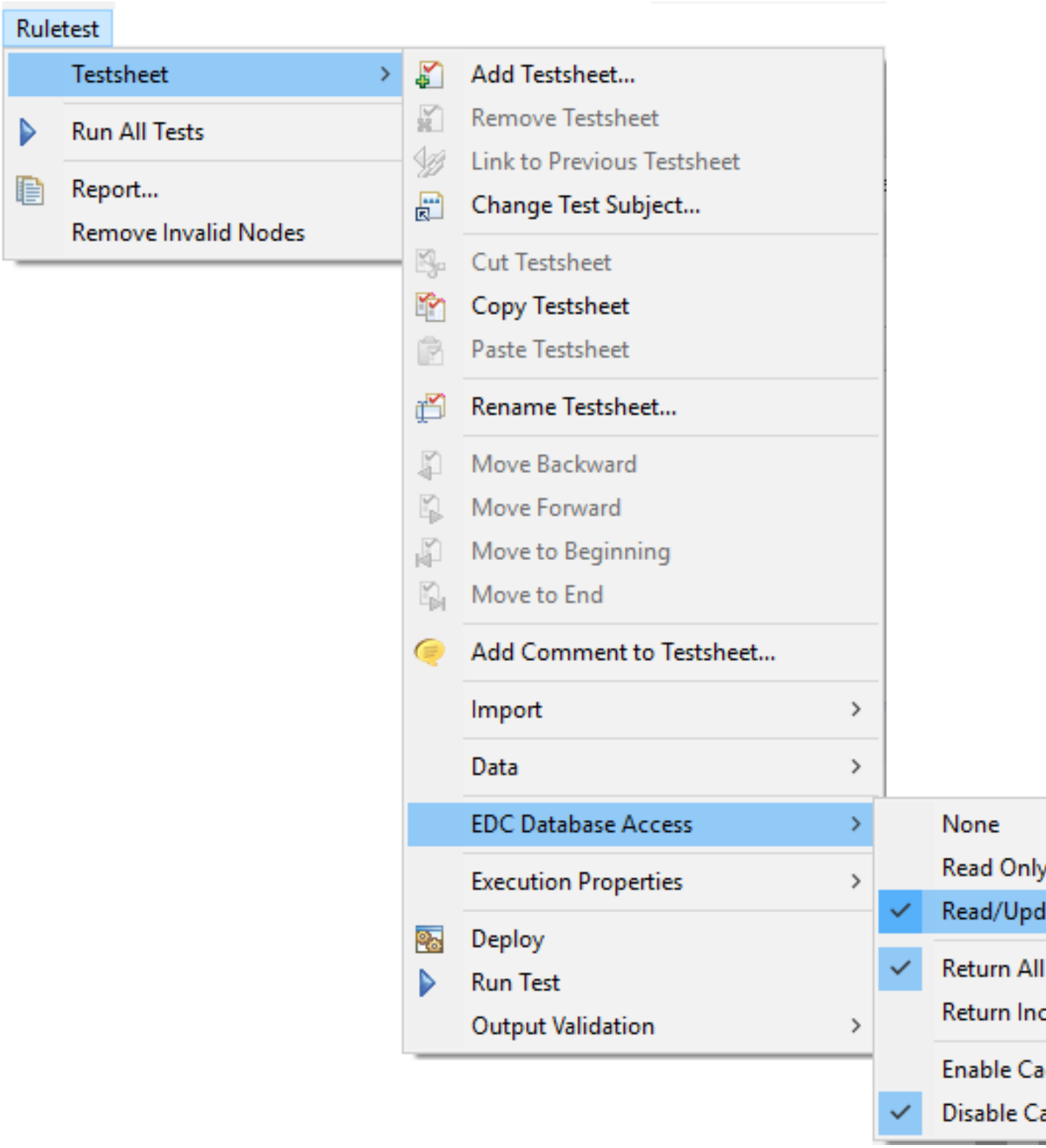
AddRecord.ers

	Conditions	0	1	2	3
a	Cargo.weight		< 100000	100000..200000	> 200000
b					
c					
d					
e					
	Actions				
	Post Message(s)				
A	Cargo.container		'STANDARD'	'HEAVY'	'LARGE'
B					

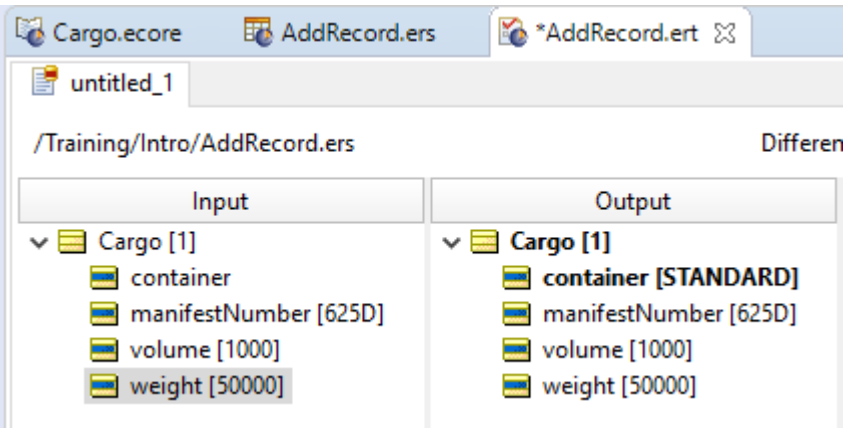
Since this Rulesheet uses the Cargo.ecore Vocabulary that is mapped to the Transportation database we don't need to do anything more here. Let's test this Rulesheet.

Create a Ruletest named **AddRecord.ert** that uses AddRecord.ers as its test subject.

We want this Ruletest to add a new record to the Transportation database. To enable this, we need to configure the Ruletest's database access setting, changing it from the default to Read/Update. Select **Ruletest > Testsheet > Database Access > Read/Update**.



Next, we need to define input. For this example, let's create just one instance of the Cargo entity that we want the Rulesheet to process and add to the Cargo table as a new record. Define input data as shown and run the Ruletest:



As you can see, in the Input, the Cargo entity instance has a unique value (**625D**) for the manifestNumber attribute, which is the primary key of the Cargo table. Because the entity instance has a unique value for the primary key, it will be added as a new record. If we used an existing value, such as 625A, 625B, or 625C, the existing record with that primary key value would get updated instead.

Let's look at the new record in the Cargo table. In SQL Management Server Studio, right-click **dbo.Cargo** and select **Select Top 1000 Rows**. You should be able to see the following result:

Results | Messages

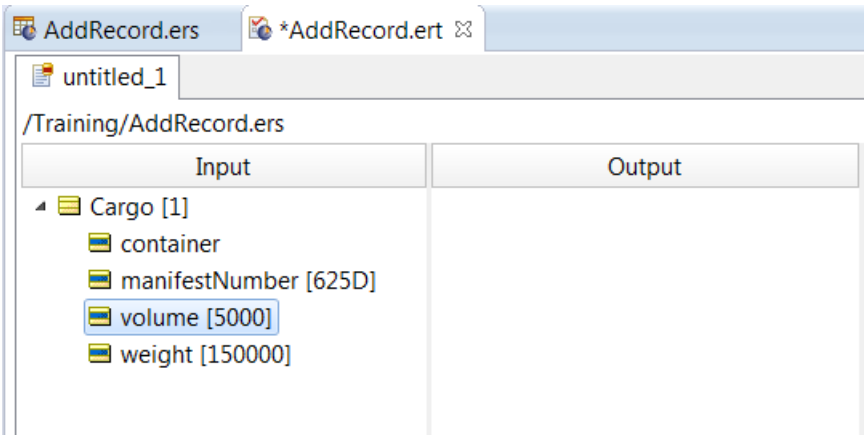
	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625C	STANDARD	4000	125000	NULL
4	625D	STANDARD	1000	50000	NULL

A new record (highlighted here) has been added. The Rulesheet processed the input, assigned the value STANDARD to Cargo.container and added a new record to the Cargo table using the unique primary key value **625D**.

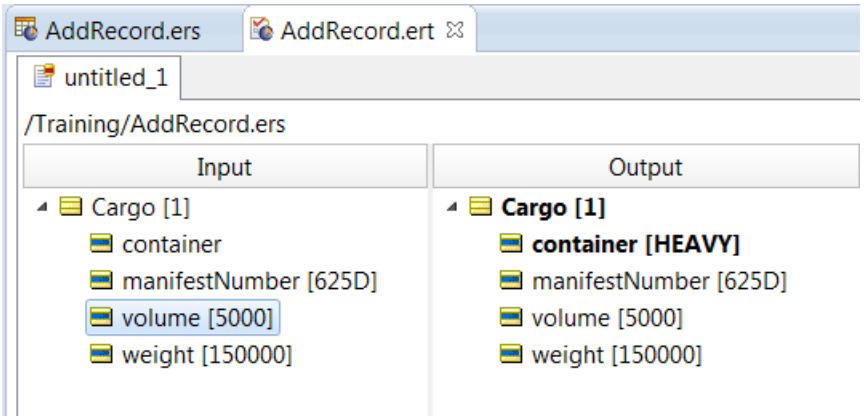
Updating an existing record

As you just learned, if the Rulesheet processes an entity instance that has the same primary key value as an existing record in the database, it updates that record in the database.

Let's update the record that we just added (**625D**). In the **AddRecord.ert** Ruletest, delete the output and modify the **Cargo.volume** and **Cargo.weight** attributes in the Input as shown here (make sure that you DO NOT change the 625D primary key value):



Run the Ruletest. You now see the following Output:



Because we modified the value of Cargo.weight to 150000, the rule in the AddRecord.ers Rulesheet assigns the value **HEAVY** to Cargo.container.

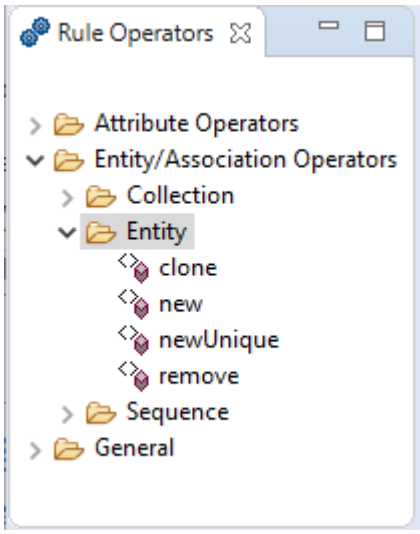
Let's see if this change is reflected in the database. In SQL Server Management Studio, right-click **dbo.Cargo** and select **Select Top 1000 Rows**. You should be able to see the following changes in the record:

Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625C	STANDARD	4000	125000	NULL
4	625D	HEAVY	5000	150000	NULL

Adding a record using a primary key defined in a rule

So far, you have seen examples of adding and updating records, where the primary key value is supplied in input. In some cases, you may want your Rulesheet to define a primary key value for a new record and then add the record to the database. To do this, you use the .new entity operator in a rule action. You can access this operator from **Entity/Association Operators > Entity** in the **Rule Operators** view.



The syntax of the new operator is:

<Entity>.new[attribute1=value, attribute2=value,… association1.attribute1=value.

Let's model a rule that uses the new operator. In this example, assume that you need to model a rule that creates a new FlightPlan instance where an Aircraft instance has a **maxCargoWeight** between 100000 and 200000. If so, an Aircraft instance is assigned to the FlightPlan instance.

Create a Rulesheet named **NewRecord.ers** and model a rule as shown here:

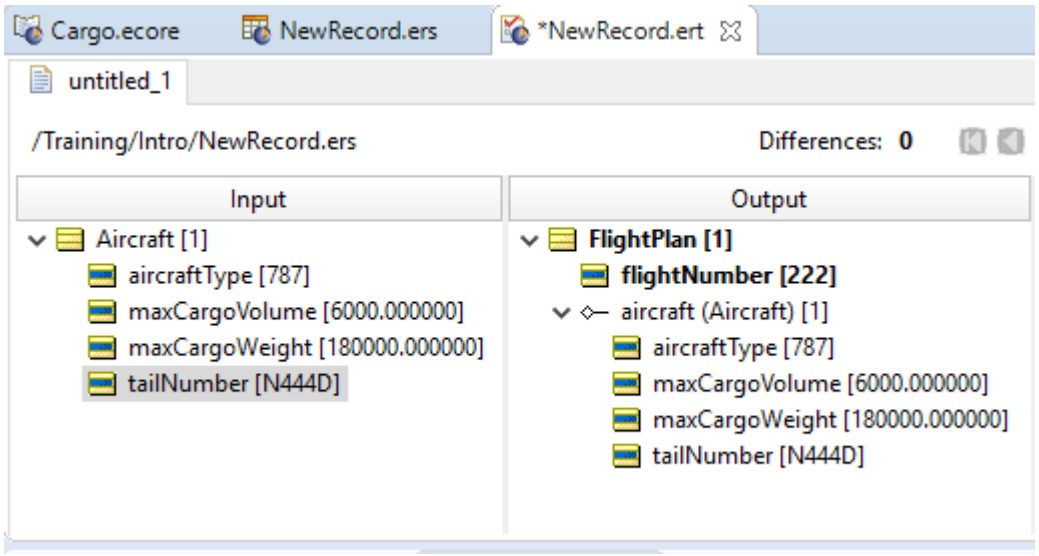
NewRecord.ers

	Conditions	0	1
a	Aircraft.maxCargoWeight		100000..200000
b			
c			
d			
e			
f			
	Actions		
	Post Message(s)		
A	FlightPlan.new[flightNumber=222, aircraft=Aircraft]	☐	☑
B			

The range in column 1 is written as the low limit, two dots, then the high limit. The new operator is used in the action row expression: FlightPlan.new[flightNumber=222,aircraft=Aircraft]

A flightNumber (**222** in this example) must be supplied since flightNumber is the primary key for FlightPlan. The expression **aircraft=Aircraft** in the action row associates the Aircraft instance in input with the FlightPlan entity that will be created.

Let's test this rule. Create a Ruletest named **NewRecord.ert** that uses NewRecord.ers as its test subject. Configure the Ruletest to write to the database by selecting **Ruletest > Testsheet > Database Access > Read/Update**. Define the following input and run the test (make sure that you provide a value such as **N444D** for tailNumber, since tailNumber is the primary key for Aircraft):



As you can see, a new FlightPlan instance is created with the flightNumber 222 and the Aircraft instance is assigned to it.

Let's look at the database tables. Right-click **dbo.Aircraft** in the Transportation database and select **Select Top 1000 Rows**. You should see the following results:

Results | Messages

	tailNumb...	aircraftTy...	maxCargoVolu...	maxCargoWei...
1	N111A	747	7500.000000	150000.000000
2	N222B	DC-10	4000.000000	125000.000000
3	N333C	787	8000.000000	175000.000000
4	N444D	787	6000.000000	180000.000000

A new Aircraft record has been added. Now, right-click **dbo.FlightPlan** and select **Select Top 1000 Rows**. You should see the following results:

	flightNum...	RaircraftAssoc_tailNum...
1	111	N111A
2	222	N444D

A new FlightPlan record has been added with the flightNumber 222. It is also associated with an Aircraft bearing the tailNumber N444D.

6

Deleting records from a database

You can also define a rule to delete a record from a database. As in the case of defining rules to perform read operations, there are two scenarios in which you may want to delete records:

- You want to delete a specific record. In this case, you need to use the record's primary key value.
- You want to delete ALL records that satisfy certain conditions. In this case, you do not need to use primary keys.

You use the .remove operator to delete records. The .remove operator can only be used in a rule action. You can find it in the **Rule Operators** view, under **Entity/Association Operators > Entity**.

For details, see the following topics:

- Deleting a specific record using a primary key
- Deleting multiple records based on rule conditions

Deleting a specific record using a primary key

For this example, assume that an employee has identified a damaged container during manual inspection, and you want to model a rule to delete that container's record from the database so that it will not be assigned to a FlightPlan.

Create a Rulesheet named **DeleteRecord.ers** that uses Cargo.ecore as its Vocabulary. Model a rule as shown here (remember to create a rule statement and link it to the rule):

DeleteRecord.ers

	Conditions	0	1	2
a				
b				
c				

	Actions			
	Post Message(s)			
A	Cargo.remove	☑		
B				
C				
D				
E				
	Overrides			

Rule Statements | Rule Messages

Ref	ID	Post	Alias	Text
A0		Warning	Cargo	This Cargo record [{Cargo.manifestNumber}] has been removed from the database

Note how the .remove operator is used in the action-only rule. No parameters are required in the .remove operator. The rule is non-conditional.

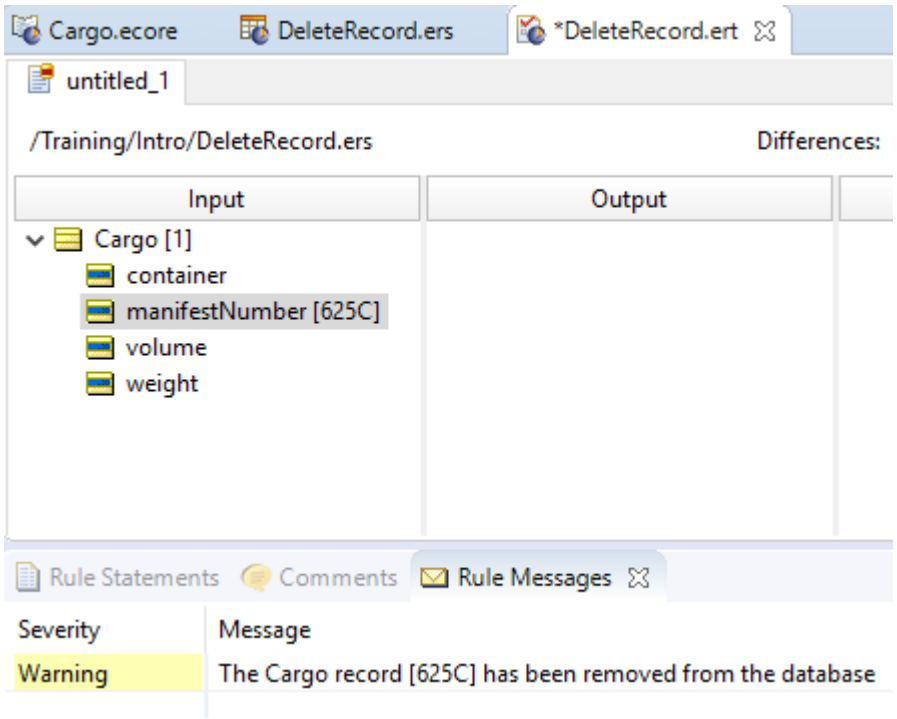
If you have followed all the steps in tutorial, the **dbo.Cargo** table should now have four records:

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625C	STANDARD	4000	125000	NULL
4	625D	HEAVY	5000	150000	NULL

Assuming that 625C is the damaged container, let's test the Rulesheet by supplying this manifestNumber in the Ruletest's input.

Create a Ruletest named **DeleteRecord.ert** that uses DeleteRecord.ers as its test subject. Configure the Ruletest to perform write operations on the database by selecting **Ruletest > Testsheet > Database Access > Read/Update**.

Define the record **625C** as the input and run the test. You should be able to see the Warning message that is linked to the rule:



Let's verify that this record has been deleted from the database. In SQL Server Management Studio, right-click **dbo.Cargo** and select **Select Top 1000 Rows**. You should see the following result:

Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625D	HEAVY	5000	150000	NULL

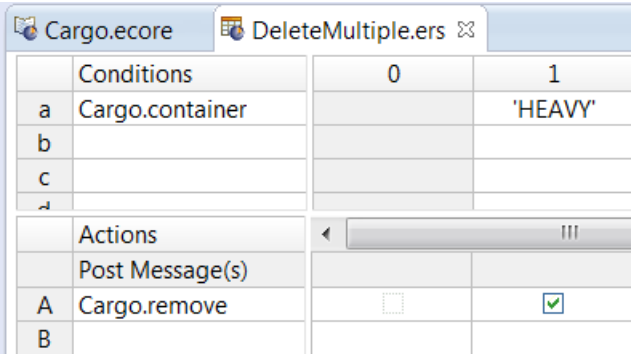
The record 625C is gone. There are only three records in this table now.

Deleting multiple records based on rule conditions

Deleting multiple records based on rule conditions is similar to reading records without using a primary key. The entities used in the rule must be extended to the database. The rule then loads all the records from the database tables into memory and processes them. All records that satisfy the rule conditions are deleted through the rule action, which uses the .remove operator.

Let's assume that all HEAVY containers must be replaced by next generation containers that meet new regulatory requirements. In this case, you want to remove all the current HEAVY containers from the database.

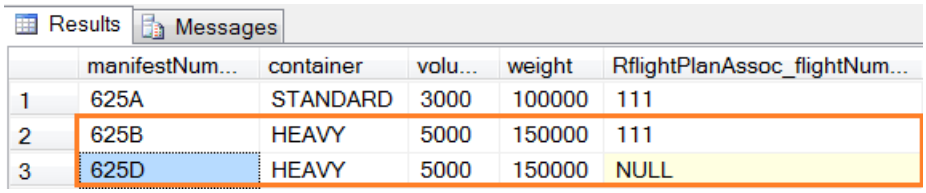
Create a Rulesheet named **DeleteMultiple.ers**. Model a rule as shown here:



Switch to Advanced View by selecting **Rulesheet > Advanced View**. In the **Scope** pane, right-click **Cargo** and select **Extend to Database**.

Save the Rulesheet.

The two HEAVY containers in the Cargo table—**625B** and **625D**—will get deleted when we run the Ruletest without supplying primary key values**:**

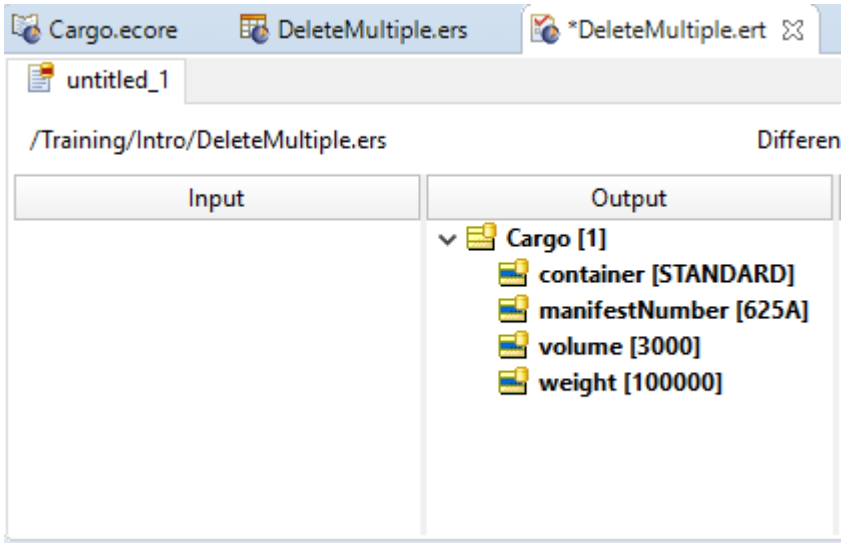


Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111
2	625B	HEAVY	5000	150000	111
3	625D	HEAVY	5000	150000	NULL

Create a new Ruletest named **DeleteMultiple.ert**, configure the Database Access setting to Read/Update, and run the test (you don't need to provide any input).

You get this output:



The Cargo record shown in the Output (625A) is the remaining record in the **dbo.Cargo** table.

Let's verify this in the Transportation database. In SQL Server Management Studio, right-click **dbo.Cargo** and select **Select Top 1000 Rows**. You should be able to see the following result:

Results | Messages

	manifestNum...	container	volu...	weight	RflightPlanAssoc_flightNum...
1	625A	STANDARD	3000	100000	111

Only one record remains, indicating that the rule deleted all records that represented HEAVY containers.

Congratulations! You have now modeled rules that use EDC to read, write, and delete database records.