

Corticon Tutorial:  
Using Enterprise Data Connector  
(EDC)



---

# Notices

---

## Copyright agreement

© 2014 Progress Software Corporation and/or 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.

Business Making Progress, Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Fathom, Making Software Work Together, OpenEdge, Powered by Progress, Progress, Progress Control Tower, Progress OpenEdge, Progress RPM, Progress Software Business Making Progress, Progress Software Developers Network, Rollbase, RulesCloud, RulesWorld, SequeLink, SpeedScript, Stylus 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. AccelEvent, AppsAlive, AppServer, BusinessEdge, Progress EasyI, DataDirect Spy, DataDirect SupportLink, EasyI, Future Proof, High Performance Integration, Modulus, OpenAccess, Pacific, ProDataSet, Progress Arcade, Progress Pacific, Progress Profiles, Progress Results, Progress RFID, Progress Responsive Process Management, Progress Software, ProVision, PSE Pro, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress 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.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.



---

# Table of Contents

|   |               |
|---|---------------|
| <b>Preface.....</b>   | <b>7</b>      |
| Progress Corticon documentation.....  | 7             |
| Overview of Progress Corticon.....  | 9             |
| <br><b>Chapter 1: Introduction.....</b>                                     | <br><b>11</b> |
| <br><b>Chapter 2: Overview.....</b>   | <br><b>13</b> |
| <br><b>Chapter 3: Setting up the EDC Tutorial.....</b>                      | <br><b>15</b> |
| <br><b>Chapter 4: Enabling EDC in Corticon Studio.....</b>                  | <br><b>17</b> |
| <br><b>Chapter 5: Sample Project.....</b>                                   | <br><b>19</b> |
| <br><b>Chapter 6: Downloading and installing the database.....</b>          | <br><b>23</b> |
| <br><b>Chapter 7: Vocabularies and databases.....</b>                       | <br><b>25</b> |
| <br><b>Chapter 8: Creating a new database in SQL Server.....</b>            | <br><b>27</b> |
| <br><b>Chapter 9: Connecting a Vocabulary to a database.....</b>            | <br><b>29</b> |
| <br><b>Chapter 10: Using Studio to create a schema in the database.....</b> | <br><b>35</b> |
| <br><b>Chapter 11: Verifying Vocabulary mappings.....</b>                   | <br><b>41</b> |
| <br><b>Chapter 12: Populating the database.....</b>                         | <br><b>45</b> |
| <br><b>Chapter 13: Using database access in Studio tests.....</b>           | <br><b>51</b> |
| Read-only database access in a Studio test.....                             | 51            |

---

|  |           |
|--|-----------|
| Read/Update database access in a Studio test.....                                      | 56        |
| <b>Chapter 14: Importing an attribute's possible values from database tables.....</b>  | <b>61</b> |
| <b>Chapter 15: Creating a database access properties file.....</b>                     | <b>69</b> |
| <b>Chapter 16: Using database access in Corticon Server.....</b>                       | <b>71</b> |
| <b>Appendix A: Corticon features and requirements in supported data sources.....</b>   | <b>77</b> |
| DataDirect Cloud: Rollbase.....  | 78        |
| DataDirect Cloud: Salesforce.....  | 79        |
| IBM DB2.....   | 79        |
| Microsoft SQL Server.....  | 80        |
| Oracle Database.....   | 81        |
| Progress OpenEdge.....   | 81        |
| <b>Appendix B: Continuing to learn about Corticon's Enterprise Data Connector.....</b> | <b>83</b> |

---

# Preface

---

For details, see the following topics:

- [Progress Corticon documentation](#)
- [Overview of Progress Corticon](#)

## Progress Corticon documentation

The following documentation, as well as a *What's New in Corticon* document, is included with this Progress Corticon release:

| Corticon Tutorials                                      |   |
|---|---|
| <i>Corticon Studio Tutorial: Basic Rule Modeling</i>    | Introduces modeling, analyzing, and testing rules and decisions in Corticon Studio. Recommended for evaluators and users getting started. <i>See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's <b>Help</b> menu.</i>   |
| <i>Corticon Studio Tutorial: Advanced Rule Modeling</i> | Provides a deeper look into Corticon Studio's capabilities by defining and testing vocabularies, scope, collections, messages, filters, conditions, transient data, and calculations in multiple rulesheets that are assembled into a Ruleflow. <i>See also the PowerPoint-as-PDF version of this document that is accessed from the Studio's <b>Help</b> menu.</i> |

|  |   |
|--|---|
| <i>Corticon Tutorial: Using Enterprise Data Connector (EDC)</i>            | Introduces Corticon's direct database access with a detailed walkthrough from development in Studio to deployment on Server. Uses Microsoft SQL Server to demonstrate database read-only and read-update functions.   |
| <b>Corticon Studio Documentation: Defining and Modeling Business Rules</b> |   |
| <i>Corticon Studio: Installation Guide</i>                                 | Step-by-step procedures for installing Corticon Studio on computers running Microsoft Windows as a standalone installation and as a part of an existing Eclipse installation such as Progress Developer Studio for OpenEdge. Shows how to enable internationalization on Windows.   |
| <i>Corticon Studio: Rule Modeling Guide</i>                                | Presents the concepts and purposes the Corticon Vocabulary, then shows how to work with it in Rulesheets by using scope, filters, conditions, collections, and calculations. Discusses chaining, looping, dependencies, filters and preconditions in rules. Presents the Enterprise Data Connector from a rules viewpoint, and then shows how database queries work. Provides information on versioning, natural language, reporting, and localizing. Provides troubleshooting and many <i>Test Yourself</i> exercises.                                     |
| <i>Corticon Studio: Quick Reference Guide</i>                              | Reference guide to the Corticon Studio user interface and its mechanics, including descriptions of all menu options, buttons, and actions.  |
| <i>Corticon Studio: Rule Language Guide</i>                                | Reference information for all operators available in the Corticon Studio Vocabulary. A Rulesheet example is provided for many of the operators. Includes special syntax issues, handling arithmetic and character precedence issues.  |
| <i>Corticon Studio: Extensions Guide</i>                                   | Detailed technical information about the Corticon extension framework for extended operators and service call-outs. Describes several types of operator extensions, and how to create a custom extension plug-in.   |
| <b>Corticon Server Documentation: Deploying Rules as Decision Services</b> |   |
| <i>Corticon Server: Integration &amp; Deployment Guide</i>                 | An in-depth, technical description of Corticon Server deployment methods, including preparation and deployment of Decision Services and Service Contracts through the Deployment Console tool. Describes JSON request syntax and REST calls. Discusses relational database concepts and implementation of the Enterprise Data Connector. Goes deep into the server to discuss state, persistence, and invocations by version or effective date. Includes server monitoring techniques, performance diagnostics, and recommendations for performance tuning. |



|  |   |
|--|---|
| <i>Corticon Server: Deploying Web Services with Java</i> | Details installing the Corticon Server as a Web Services Server, and then deploying and exposing Decision Services as Web Services on the Pacific Application Server (PAS) and other Java-based servers. Includes samples of XML and JSON requests. Presents the features and functions of the browser-based Server Console. Provides administrative instructions for the Pacific Application Server. |
| <i>Corticon Server: Deploying Web Services with .NET</i> | Details installing the Corticon Server as a Web Services Server, and then deploying and exposing decisions as Web Services with .NET. Includes samples of XML and JSON requests. Provides installation and configuration information for the .NET Framework and Internet Information Services (IIS) on various supported Windows platforms.   |

## Overview of Progress Corticon

Progress® Corticon® is the Business Rules Management System with the patented "no-coding" rules engine that automates sophisticated decision processes.

### Progress Corticon products

Progress Corticon distinguishes its development toolsets from its server deployment environments.

- **Corticon Studio** is the Windows-based development environment for creating and testing business rules:
  - When installed as a standalone application, Corticon Studio provides the complete Eclipse development environment for Corticon as the **Corticon Designer** perspective. You can use this fresh Eclipse installation as the basis for adding other Eclipse toolsets.
  - When installed into an existing Eclipse such as the **Progress Developer Studio (PDS)**, our industry-standard Eclipse and Java development environment, the PDS enables development of Corticon applications in the **Corticon Designer** perspective that integrate with other products, such as Progress OpenEdge.

---

**Note:** Corticon installers are available for 64-bit and 32-bit platforms. Typically, you use the 64-bit installer on a 64-bit machine, where that installer is not valid on a 32-bit machine. When adding Corticon to an existing Eclipse, the target Eclipse must be an installation of the same bit width. Refer to the *Corticon Studio: Installation Guide* to access, prepare, and install Corticon Studio.

---

**Studio Licensing** - Corticon embeds a time-delimited evaluation license that enables development of both rule modeling and Enterprise Data Connector (EDC) projects, as well as testing of the projects in an embedded Axis test server. You must obtain Studio development licenses from your Progress representative.

- **Corticon Servers** implement web services for business rules defined in Corticon Studios:
  - **Corticon Server for deploying web services with Java** is supported on various application servers, and client web browsers. After installation on a supported Windows platform, that

server installation's deployment artifacts can be redeployed on various UNIX and Linux web service platforms as Corticon Decision Services. The guide *Corticon Server: Deploying web services with Java* provides details on the full set of platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.

- **Corticon Server for deploying web services with .NET** facilitates deployment of Corticon Decision Services on Windows .NET Framework and Microsoft Internet Information Services (IIS). The guide *Corticon Server: Deploying web services with .NET* provides details on the platforms and web service software that it supports, as well as installation instructions in a tutorial format for typical usage.

**Server Licensing** - Corticon embeds a time-delimited evaluation license that enables evaluation and testing of rule modeling projects on supported platform configurations. You must obtain server deployment licenses and server licenses that enable the Enterprise Data Connector (EDC) from your Progress representative.

---

# Introduction

---

This *Enterprise Data Connector Tutorial* explores the key features and functionality of Progress Corticon with Enterprise Data Connector (EDC).

You will learn how to:

- Connect Corticon Studio to an external database, and then create a table based on a Vocabulary.
- Produce Rulesheets and Ruletests in Studio that interact with the database in read-only mode and then in read/update mode.
- Connect a Decision Service deployed on Corticon Server to an external database and access it during execution.

The format of this tutorial defines its requirements narrowly, and the procedures must be followed accurately and sequentially so that the exercises respond as intended. Once you have completed the tutorial, the concepts apply readily to other supported databases and platforms.

## Purpose and intended audience

This tutorial is intended for use by professionals who want to integrate a commercial RDBMS with Corticon Decision Services.

At a minimum, this Tutorial assumes:

- You have installed Corticon Studio, as described in the *Corticon Studio: Installation Guide*.
- You have installed Corticon Server, as described in the *Corticon Server: Deploying Web Services with Java*.
- You have completed both the *Basic Tutorial for Corticon Studio – Basic Rule Modeling* and *Tutorial for Corticon Server – Deploying Web Services*. This Tutorial builds on those two documents.

- You have some experience building and managing databases with relational database management systems (RDBMS) products or tools
- You are familiar with basic relational theory and terminology, such as primary and foreign keys, tables, columns, fields, and joins.

---

## Overview

---

Corticon provides database access for Corticon rules. Corticon can read from a database to enrich its data, and can add or modify database records. This direct database access (DDA) functionality is referred to as the Corticon Enterprise Data Connector (EDC). Before we start the tutorial tasks, let's review a few of the concepts and features of EDC.

---

**Note:** The concepts introduced here are discussed in greater depth in the other areas of the Progress Corticon Documentation set. See [Continuing to learn about Corticon's Enterprise Data Connector](#) on page 83 for more information.

---

Consider the following concepts and features of Corticon EDC:

- **Technologies** - Corticon enables mapping to supported RDBMS brands using Progress Software's enterprise-grade Data Direct drivers, object/relational mapping in Hibernate, and the algorithms in C3PO's open-source JDBC connection pooling. Corticon creates Corticon Data Objects (CDOs) that are Hibernate-ready, and requests Hibernate to validate them against the database. These tools are embedded in the Corticon products, and maintained as product upgrades are applied. There is no initial need for configuring any of these supporting technologies, yet common tuning parameters are documented.
- **Corticon's Vocabulary binds to specified RDBMS metadata** - The Corticon Vocabulary stores the database connection and database metadata (tables, columns, primary keys, and foreign keys) that Corticon loads into its working memory as Corticon Data Objects (CDOs) for use in rule execution. Database metadata is used to infer the values of database-related fields based on rules. The import of database metadata imported into the Corticon Vocabulary can be constrained to specified entities to minimize the amount of metadata stored inside the Vocabulary asset. Dynamic validation tries to ensure that the Vocabulary always makes sense with respect to imported metadata. All Corticon assets that share a Vocabulary are relating to the same database.

- **Persistence** - Vocabulary entities can be *transient* or *datastore persistent* (that is, database-bound). Transient entities can have associations to persistent entities and vice versa. The user must supply instances of transient entities in the input message; conversely, datastore persistent entities are retrieved/updated in the database.
- **Database Keys** - Entities can use either *application identity* (that is, primary keys) or *datastore identity*. With application identity, the application supplies the key values to create new instances (which we will do in this tutorial), whereas with datastore identity, the key values can be established through predefined identity strategies (or the default `AutoGenID` mechanism.)

Those are the basic EDC concepts. Now let's set up and experience Corticon's Enterprise Data Connector in action!

## Setting up the EDC Tutorial

---

### Downloading and installing Corticon 5.4

1. Download and install Progress Corticon Studio 5.4 .
2. Download and install Progress Corticon Server 5.4.

---

**Note:** While you could use the Corticon Server for .NET, this tutorial focuses on the Java Server. The variations that enable use of .NET as the server with EDC are discussed in that product's tutorial, *Corticon Server: Deploying Web Services with .NET*.

---

### Pacific Application Server

Your installation of Progress Corticon Server 5.4 for Java includes the Pacific Application Server (PAS), a platform that provides Web server support for Progress applications. The foundation of PAS is Apache Tomcat. Once you have defined decision services that have database access, you will run the PAS web server to test the Decision Services ability to connect, and then read and write data from the database. See the *Corticon Server: Deploying Web Services with Java* for more information on deployment.





---

## Enabling EDC in Corticon Studio

---

Progress Corticon embeds an evaluation license in its products to help you get started.

- Corticon Studio evaluation licenses let you use database access (Enterprise Data Connector or "EDC"), and are timed to expire on a preset date. While you typically do not need to acquire a license for Studio during evaluation and early stage development, in the event that you are alerted that your license is invalid or expired, contact your Progress Corticon representative to obtain a workable license.
- Corticon Server evaluation licenses do not enable use of Enterprise Data Connector, and limit the number of decision services, rules, and pools in use. They too are timed to expire on a preset date. You will need to obtain a license that enables EDC on Corticon Server for that section of this tutorial.

For this tutorial, the following procedure shows how to set Corticon Studio to use your Corticon license, and then proceed through all but the chapter [Using database access in Corticon Server](#) on page 71 where the procedures that enable EDC for Corticon Server are discussed.

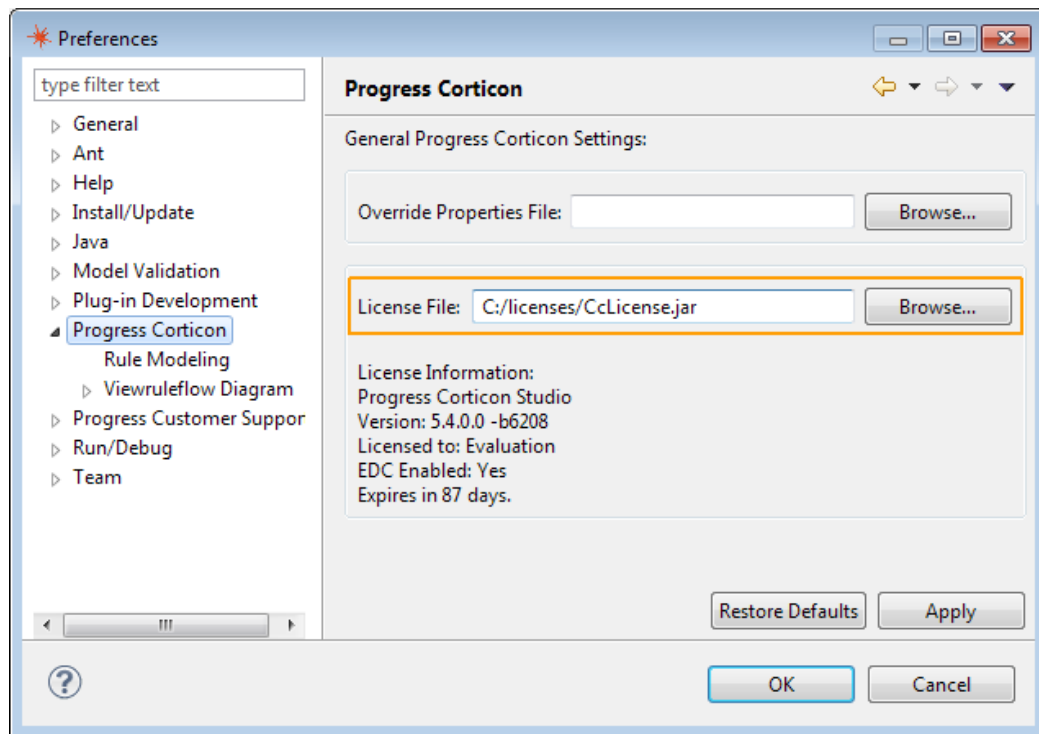
Launch Corticon Studio using one of the following methods:

- Double-click the Corticon Studio shortcut on your desktop.
- Choose Windows **Start > All Programs > Progress > Corticon 5.4 > Corticon Studio**.
- Launch the Progress Developer Studio application by double-clicking on `C:\Program Files\Progress\Corticon 5.4\Studio\eclipse\eclipse.exe`.

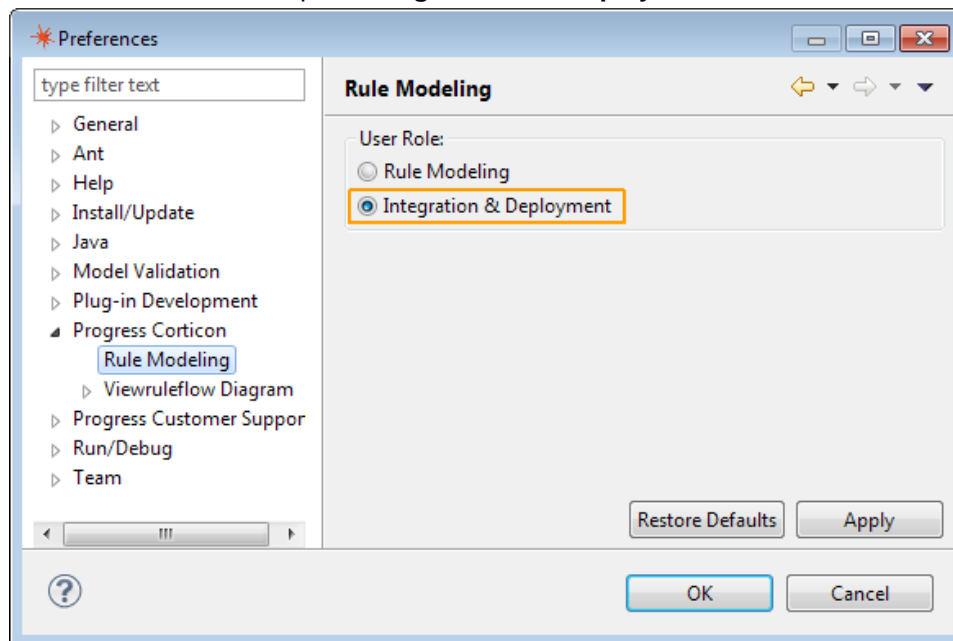
**To configure Corticon Studio for database access:**

1. Select the menu command **Window > Preferences**.
2. Expand the **Progress Corticon** group.

3. If you need to update the license or add your non-evaluation Studio license, enter its location into the **License File** entry area, or browse to its location, as in this example where a `licenses` folder was created at root:



4. Click **Apply**.
5. Expand the **Progress Corticon** group, and then click on **Rule Modeling**:
6. Choose the **User Role** option **Integration and Deployment**, as shown:



7. Click **Apply**, and then click **OK**.

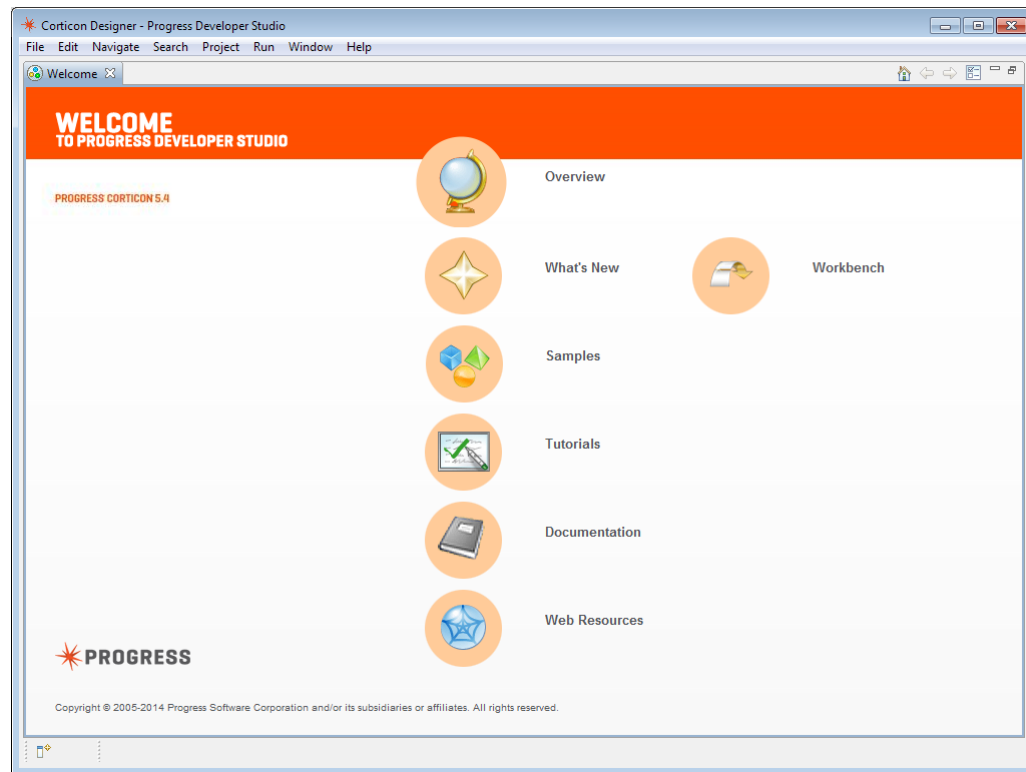
## Sample Project

---

We will use the `Cargo` Project that we used in the *Corticon Studio: Basic Rule Modeling* and the *Corticon Server Tutorial for Deploying Web Services*.

If you haven't set up the `Cargo` Project, do the following:

1. When you started Corticon Studio for the first time, it displayed its **Welcome** page, as shown:

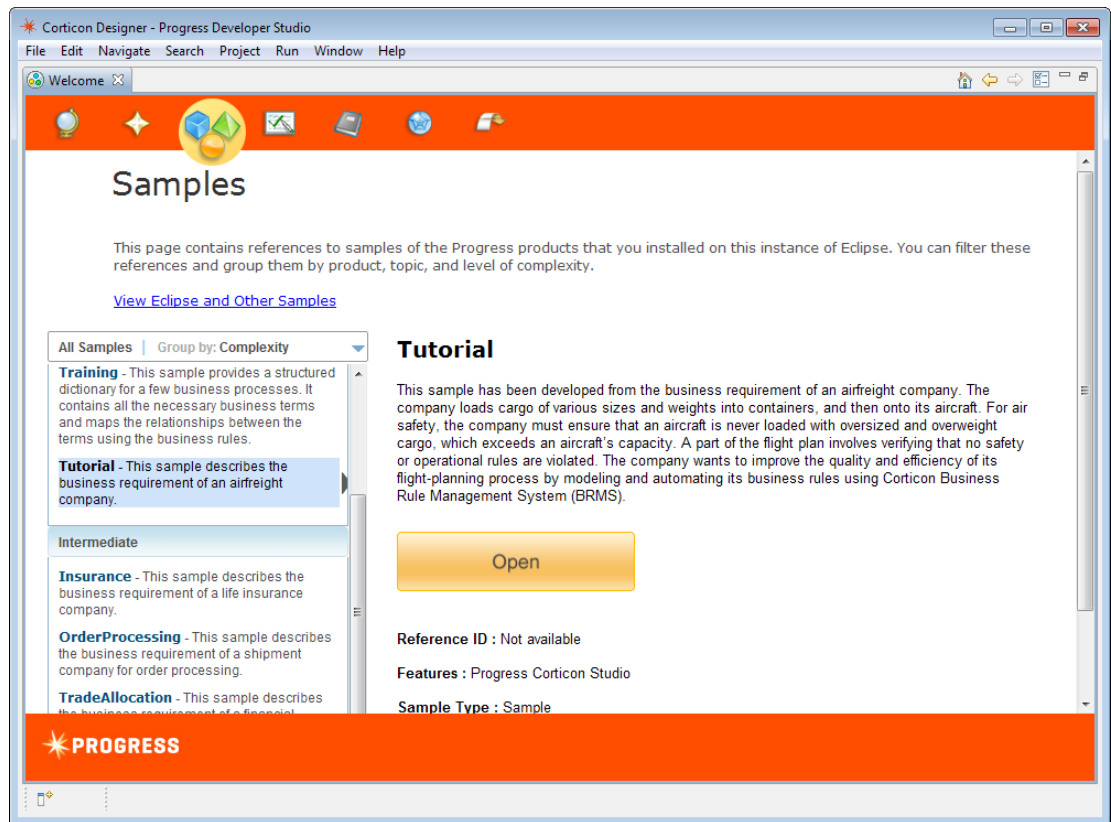


---

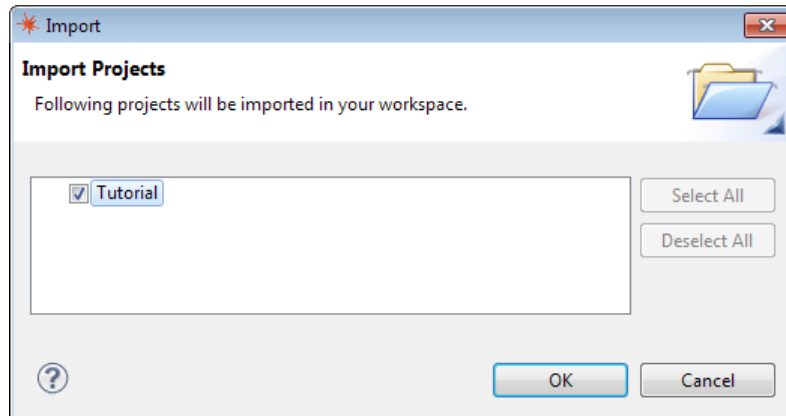
**Note:** If the page does not display as shown, you might need to re-open it by choosing **Help > Welcome**, and, if necessary, clicking the **Restore Welcome** link.

---

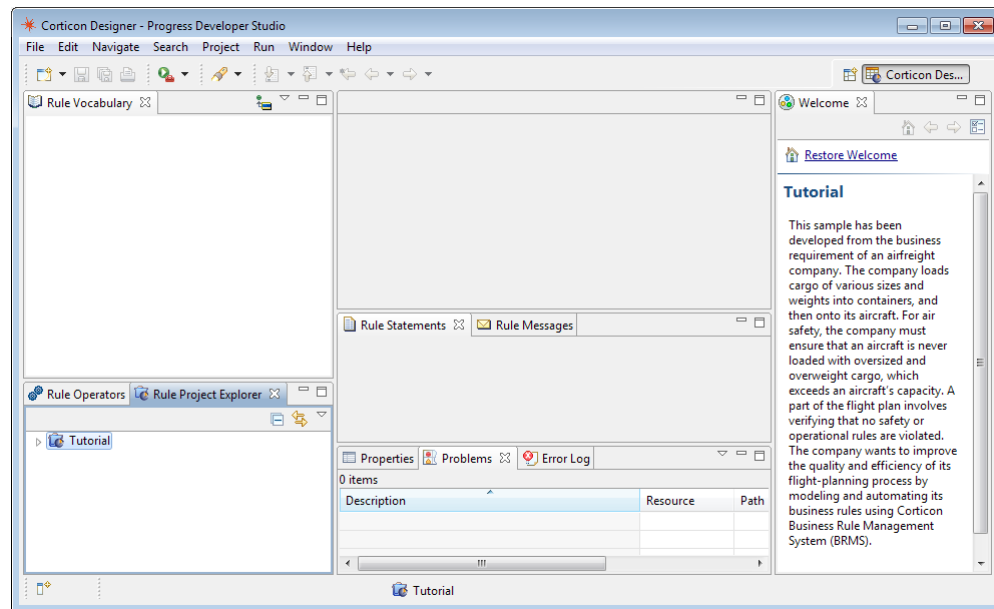
2. Click **Samples** to open the Samples page, and then scroll through the list of samples to select **Tutorial**, as shown:



3. Click **Open**. The **Import** dialog box opens. Select the **Tutorial**, as shown:



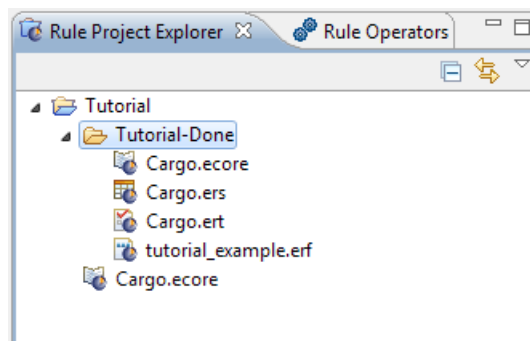
4. Click **OK** to load the project into the workspace while displaying the Welcome info in the right panel, as shown:



The Cargo Project is now set up in your Corticon Studio.

You can review the project on the **Rule Project Explorer** tab, as shown:

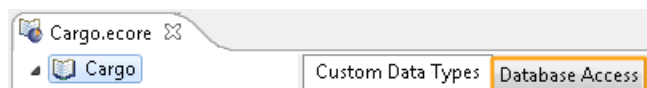
**Figure 1: The Tutorial Project in the Rule Project Explorer**



Unless you completed the *Basic Rule Modeling* tutorial, delete `Cargo.ecore` that is outside the `tutorial-done` folder.

**Note:** IMPORTANT - A step in the basic rule modeling tutorial showed the addition of the `Cargo` attribute `needsRefrigeration`. The EDC tutorial and the sample data expect that attribute to exist in the Vocabulary and the Rulesheet. Your completed basic tutorial results and the `Tutorial-Done` folder provide that added attribute.

On the **Rule Project** tab, expand `Tutorial > tutorial-done`, and then double-click the Vocabulary file, `Cargo.ecore`. Because you enabled the Integration and Deployment user role, the Vocabulary displays its **Database Access** tab, as shown:



## Downloading and installing the database

---

Progress Corticon does not include tools for creating or managing a database. This tutorial uses the Microsoft SQL Server 2008 R2 database in its free Express edition. Microsoft SQL Server, an RDBMS brand Corticon supports in production, illustrates database setup and connection to both Corticon Studios and both Corticon Servers. Once setup, the database procedures intend to be generic yet they might vary when using another RDBMS brand (such as Oracle) or another platform (such as Linux.)

### Microsoft SQL Server 2008 R2 RTM - Express with Management Tools

Access the Microsoft Download Center page

<http://www.microsoft.com/en-us/download/details.aspx?id=23650> (at the time this tutorial was created.) Register and download this free product, then fulfill the system requirements, and installation instructions. (Be sure to install the .NET framework first.)

---

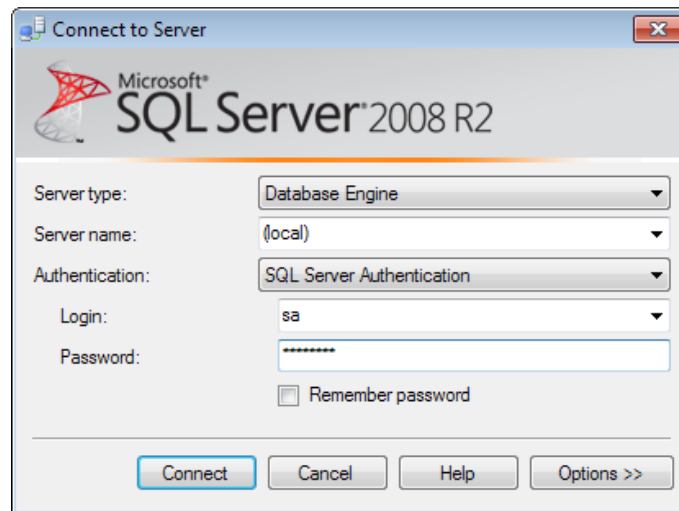
**Note:** When you create user credentials in an RDBMS product that you intend to use with Corticon's Enterprise Data Connector, the username and password must each contain at least one alphabetic character, even if the database supports only numbers.

---

Confirm that the installation is running by choosing the **Start** menu, path **All Programs > Microsoft SQL Server 2008 R2 > SQL Server Management Studio**.

In the **Connect to Server** dialog, you should be able to connect with the following default credentials and the password you specified, typically `Password1`:

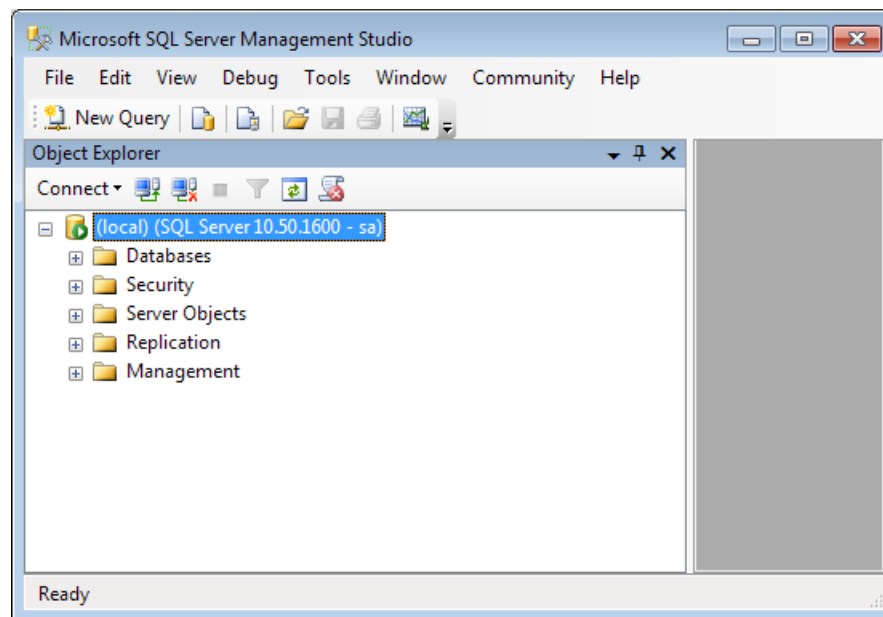
**Figure 2: Connecting the SQL Server Management Studio to the installed SQL Server instance**



Click **Connect**.

When you connect the initial window should look like this:

**Figure 3: SQL Server Management Studio**



**Note:** When you advance the rules projects toward production, it is good planning to create some user credentials that are constrained from dropping tables. A lead administrative user should retain that function and all other users should get an error if they try to create and update the schema.

## READY!

That completes the software setup for this tutorial. Now let's get familiar with the concepts that we will explore.



---

## Vocabularies and databases

---

It is not necessary to be an expert in any database product to work through this Tutorial successfully. However, it is certainly helpful to have knowledge of database relational theory or experience with building and managing databases using commercial relational database management systems (RDBMS).

We will step through the essential steps of configuring a set of tools (described in the next chapter); the concepts addressed are consistent with any supported RDBMS you choose.

### Which comes first: the Vocabulary or the Database?

You have probably already recognized the fundamental relational nature of the Corticon Vocabulary. Those with database backgrounds are quick to see that the elements of a Vocabulary are conceptually equivalent to the elements of a typical relational database.

**Table 1: Equivalent Studio—DB Concepts**

| Corticon Studio Element | Equivalent Database Concept |
|-------------------------|-----------------------------|
| Vocabulary              | Schema                      |
| Vocabulary Entity       | Table                       |
| Vocabulary Attribute    | Table Column or Field       |
| Vocabulary Association  | Relationship between Tables |
| Test Case or Use Case   | Table Row(s) or Record(s)   |

You can choose which comes before the other. You can start with a pre-existing database schema and create a Vocabulary from it, or vice versa. In practice, you might:

- Start from the business perspective by converting a Vocabulary into a database. Because Studio is a powerful modeling environment, users often build Vocabularies “on-the-fly” in order to support their rule modeling. Assuming the Vocabulary design is acceptable to IT as the basis for a database, then Studio can be used to export schema information directly to a database engine and generate the necessary table structure within a defined tablespace.
- Start from an IT perspective by abstracting a data model from an existing database and using its terms and structure to create a Vocabulary for rule modelers to use in their rule building and testing.

We will use as our working example the scenario constructed in the *Basic Tutorial for Corticon Studio – Basic Rule Modeling*. Since we already have a Vocabulary named `Cargo.ecore`, that's what we'll use as the basis for our database.

### **Validation of names of entities, attributes and associations against SQL keywords and database restrictions**

Commercial databases, such as Microsoft SQL Server and Oracle, use specific words for defining, manipulating, and accessing databases. These reserved keywords are part of the grammar used to parse and understand statements. Do not use database reserved words for Corticon Entity, Attribute, and Association names when creating the schema in Corticon. Your database support pages list reserved words -- for example, [SQL Server 2012](#) -- that you should review as you prepare your Vocabulary for enterprise data connection.

Corticon makes a best-effort to validate the names against the SQL keywords (such as `Order`), validate names against the database restrictions for column and table naming (such as length of a table name length), and validate generated column names (such as Foreign Key (FK) columns) against SQL keywords and table/column name restrictions.

---

## Creating a new database in SQL Server

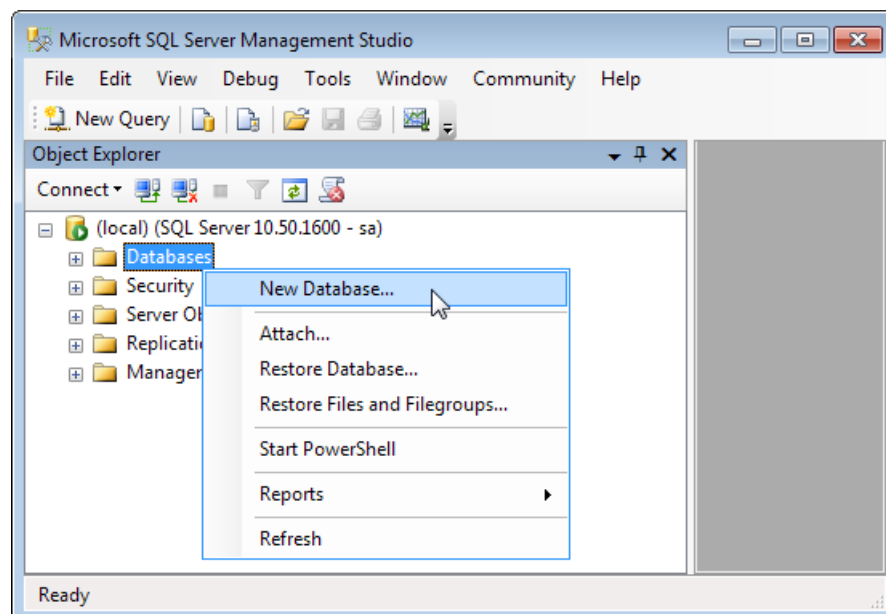
---

In this section, we will create a new database in SQL Server. Because SQL Server set itself up as a Windows Service with automatic startup, it should always be running. You confirmed that by connecting to it through its Management Studio. We'll create a new table, *Cargo*, in this database, and then use it throughout the tutorial.

To create a new table in the SQL Server database:

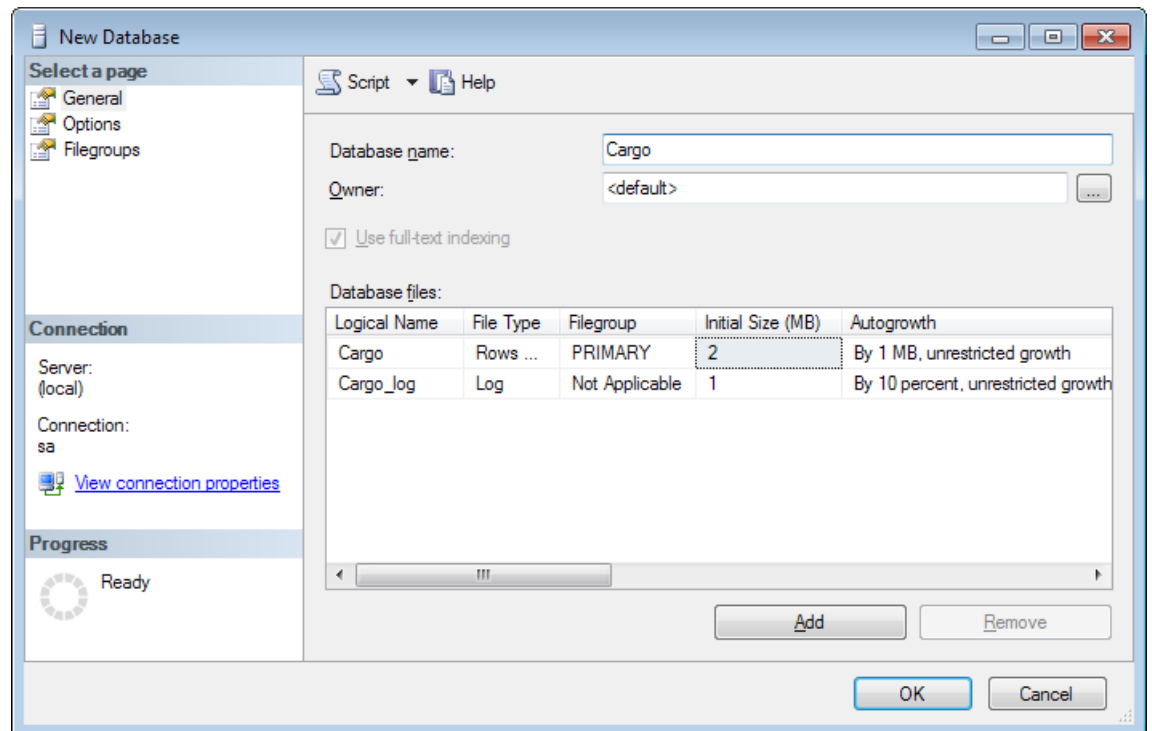
1. In your SQL Server Management Studio connection, right-click on **Databases**, and then choose **New Database**, as shown:

**Figure 4: Creating a Database**



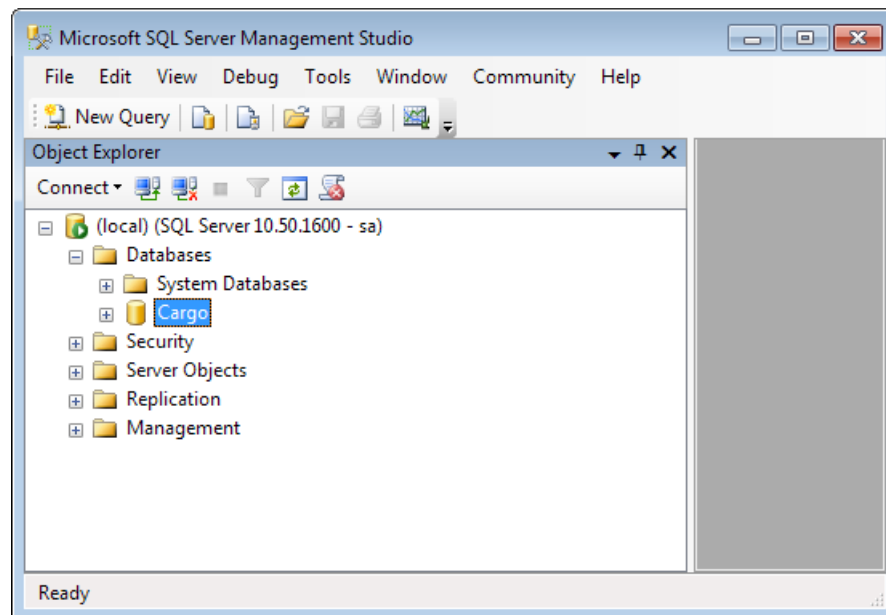
The **New Database** dialog opens, as shown:

**Figure 5: New Database dialog**



2. In the **Database name** entry area, type `Cargo`, and then click **OK**.
3. In the Object Explorer view, expand **Databases** to show the **Cargo** tablespace.

**Figure 6: The New Database, Cargo**



The database is now ready to interact with the Corticon Studio.

---

## Connecting a Vocabulary to a database

---

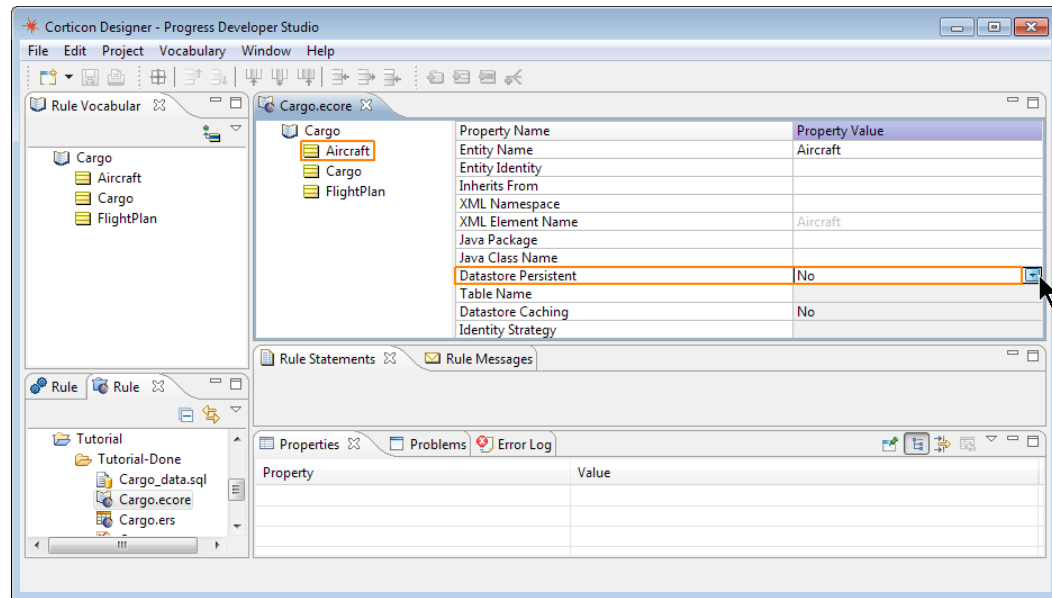
When we set our user role to **Integration & Deployment** in [Enabling EDC in Corticon Studio](#) on page 17 to expose the **Database Access**, that action also exposed properties on entities and attributes that extend their functionality. One in particular is important right now: We need to specify the entities that will be mapped to the database. If we do not, nothing happens when we create the database schema.

With our `Cargo` Vocabulary file open in its editor, we will specify the entities that want to persist in the database, and then create a new schema in our database that matches the structure of our Vocabulary.

### Setting entities to be database persistent

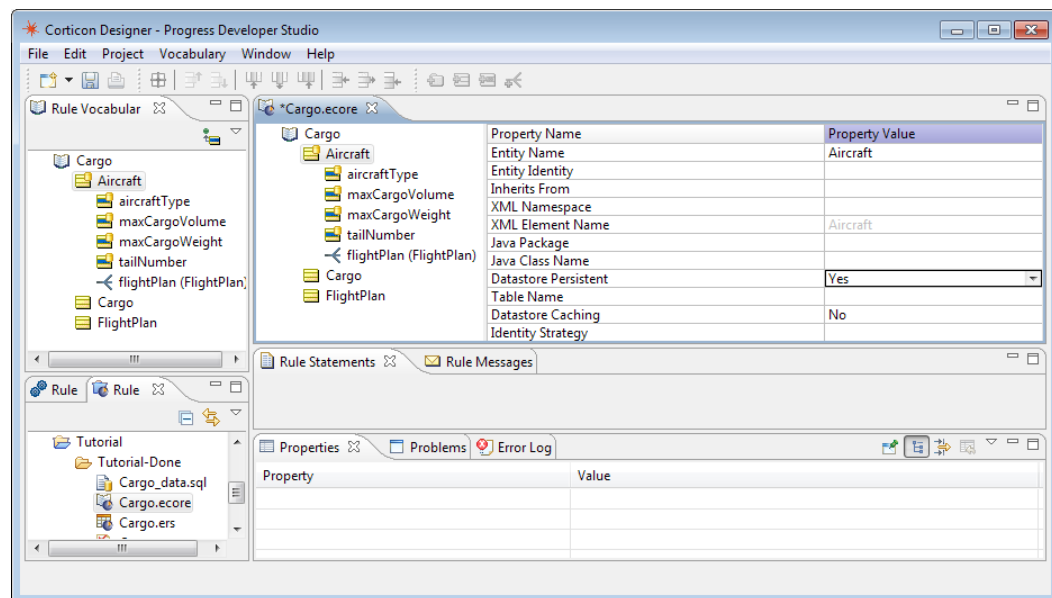
Click on the `Aircraft` entity to display its properties, then click on its **Database Persistent** Property Value dropdown menu, as shown:

Figure 7: Setting an Entity to be Database Persistent



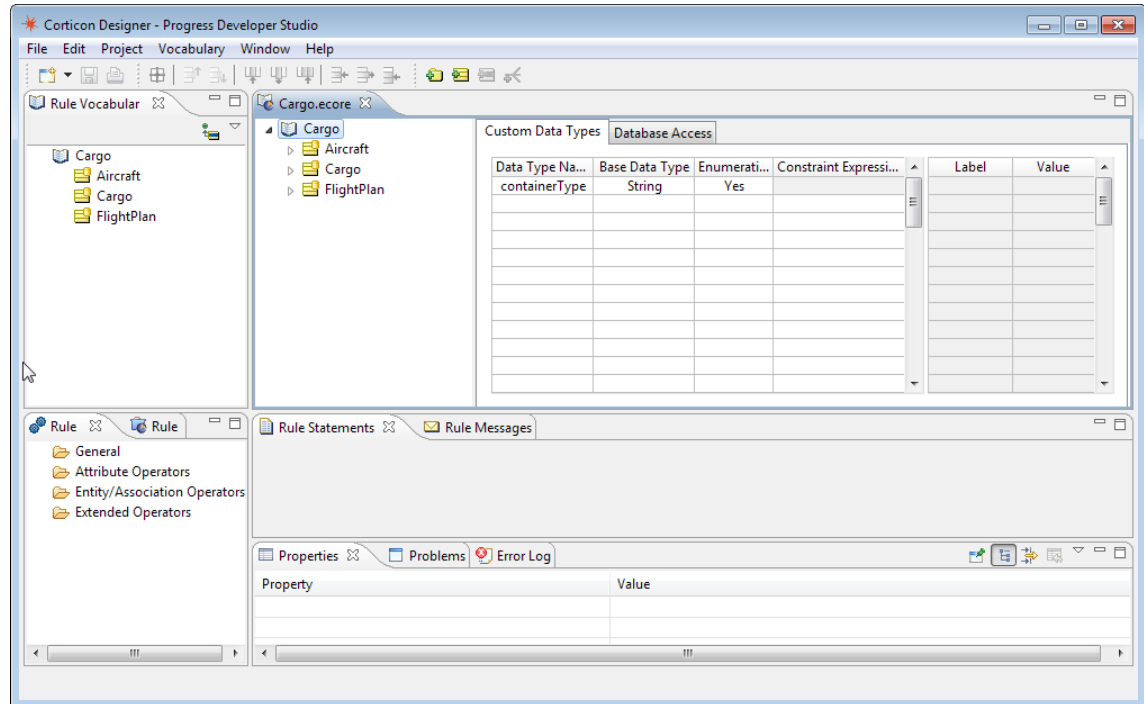
Choose **Yes**. The entity and all its attributes display their icon with a database 'decoration', as shown:

Figure 8: One entity and its attributes set to Database Persistent



Perform this action on the `Cargo` and `FlightPlan` entities. When all three entities display their database decoration, you are ready for database mapping.

**Figure 9: Vocabulary ready for database mapping**

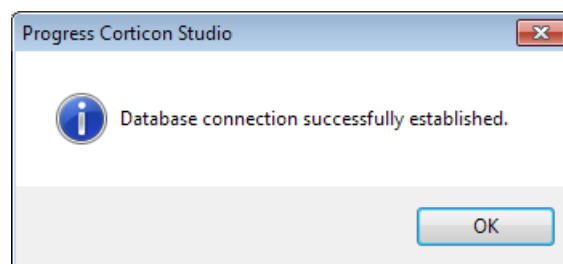


### Specifying database access properties

To connect the Vocabulary to our new `Cargo` database tablespace running on SQL Server, choose the **Database Access** tab, and then enter the following connection parameters for this tutorial (including the credentials we defined for access to our RDBMS), as follows:

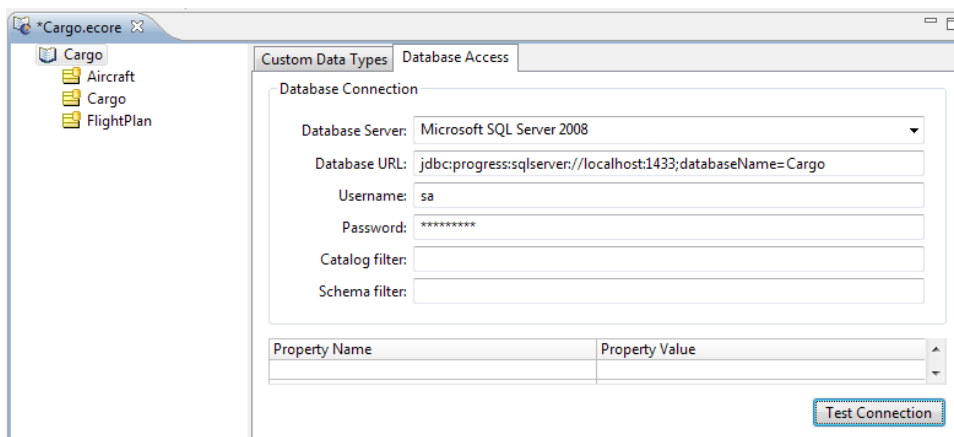
1. **Database Server:** Pull down to select **Microsoft SQL Server 2008**.
2. **Database URL:** Edit the default entry to replace (1) `<server>` with `localhost`, `127.0.0.1`, or the machine's DNS-resolvable hostname, and (2) `<database name>` with `Cargo`
3. **Username:** `sa`
4. **Password:** `Password1`
5. Click **Test Connection**. The following alert indicates success:

**Figure 10: Successful Connection**



**Note:** The **Username** and **Password** are the credentials specified in [Downloading and installing the database](#) on page 23.

Figure 11: Database Access Configuration Settings



**Note: Features when using other database brands** - The tutorial format of this guide focuses on the features and functionality of Microsoft SQL Server as the database. Other database brands are supported. For more details on Corticon supported data sources, see the section at the end of this guide, [Corticon features and requirements in supported data sources](#) on page 77, and the web location [Progress Corticon 5.4 - Supported Platforms Matrix](#).

## Troubleshooting connection to SQL Server

There are common reasons for not succeeding in an SQL Server connection. These are in order of likelihood plus complexity. Retest the connection as you work through these suggestions:

- *The database name is case-sensitive.* It must be `Cargo` in the Database Access parameters and in the database.
- *The hostname is not correct.* If you are on the same machine, `127.0.0.1` has a higher likelihood of avoiding problems.
- *The database needs to extend its authentication.* In SQL Server Management Studio, right-click on the database connection (`local`) (`SQL Server...`), and then choose **Properties**. In the **Properties** dialog, click **Security**, then set the **Server authentication** option to **SQL Server and Windows Authentication**.
- *The password is not correct.* In SQL Server Management Studio, expand **Security > Logins** and then choose the login name you used. Reset the password. Choose the **Status** page, and then confirm that the login name is **Enabled**.
- *The port, 1433, is not active or not listening.* In SQL Server Management Studio, you can see which port is active in **Management > SQL Server logs: Current** log items by locating the line `Server is listening on....`
- *Set the port and named pipes.* Launch SQL Server's **Configuration Tools > SQL Server Configuration Manager**, and expand **SQL Server Network Configuration**, and then click on **Protocols for SQLEXPRESS**.
  - *Port:* Double-click on **TCP/IP**. Click the **IP Addresses** tab, and then set **IPAll's TCP Port** to 1433. (You could instead set another IP configuration to **Active=Yes**, **Enabled=Yes** and the **TCP Port** to 1433.) Click on the **Protocol** tab to confirm that **Enabled** is now **Yes**, and listed after you click **OK**.
  - *Named Pipes:* Click on **Named Pipes** then set **Enabled** to **Yes** and the **Pipe Name** to `\\.\pipe\sql\query`



---

For more information about enumerations and retrieving values from databases, see:

- *"Enumeration" in the Rule Modeling Guide*
- *"Enumerated values" in the Quick Reference Guide.*
- *"Label and Value" in the Rule Modeling Guide*
- *"Importing an attribute's possible values from database tables" in the Using EDC Guide*
- *"Mapping database tables to Vocabulary Entities" in the Integration and Deployment Guide*



---

## Using Studio to create a schema in the database

---

Once a connection between the Studio Vocabulary and SQL Server has been established, we can automatically create a new schema in our database that corresponds to the Vocabulary's structure.

When we use a Vocabulary to "forward-generate" a database schema, we need to make sure the database has enough information to create a complete schema. This means that some properties of the Vocabulary we might otherwise be able to ignore or leave empty when just modeling rules now need to be completed.

---

**Note:** This feature is supported only on certain database brands. When you choose a Database Server, the feature will be enabled only if it is supported for that brand.

---

For example, when modeling rules, it may not be important which attributes act as an entity's identity. But when we use the Vocabulary to forward-generate a schema into a database, identity (also known as "key") information is required. So before we can generate the schema, we must first ensure the Vocabulary has all necessary properties.

### Completing the Vocabulary

Our `Cargo.ecore` Vocabulary already contains attributes that serve as unique identifiers for each entity.

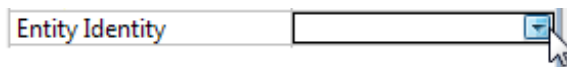
---

**Note:** By assigning a Vocabulary attribute to serve as an entity identity, we are using "application" identity. Datastore identity, where a database table's identity is not present in the Vocabulary, is also possible but not described in this Tutorial. See the *Integration & Deployment Guide* for more details on all available identity types.

---

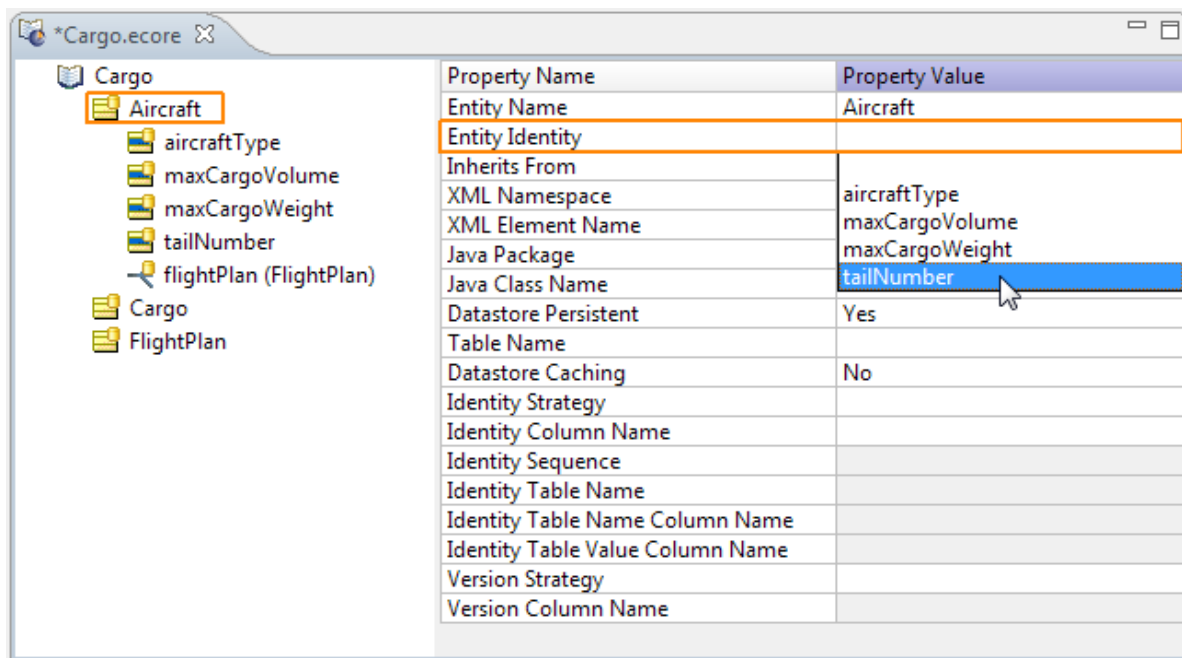
In the case of the `Aircraft` entity, the `tailNumber` attribute is the unique ID. To designate this attribute as the identity for `Aircraft`, select `Aircraft`, and then click on the right edge of the **Entity Identity** Property Value to expose its drop-down menu, as shown:

**Figure 12: Accessing the Drop-down Menu**



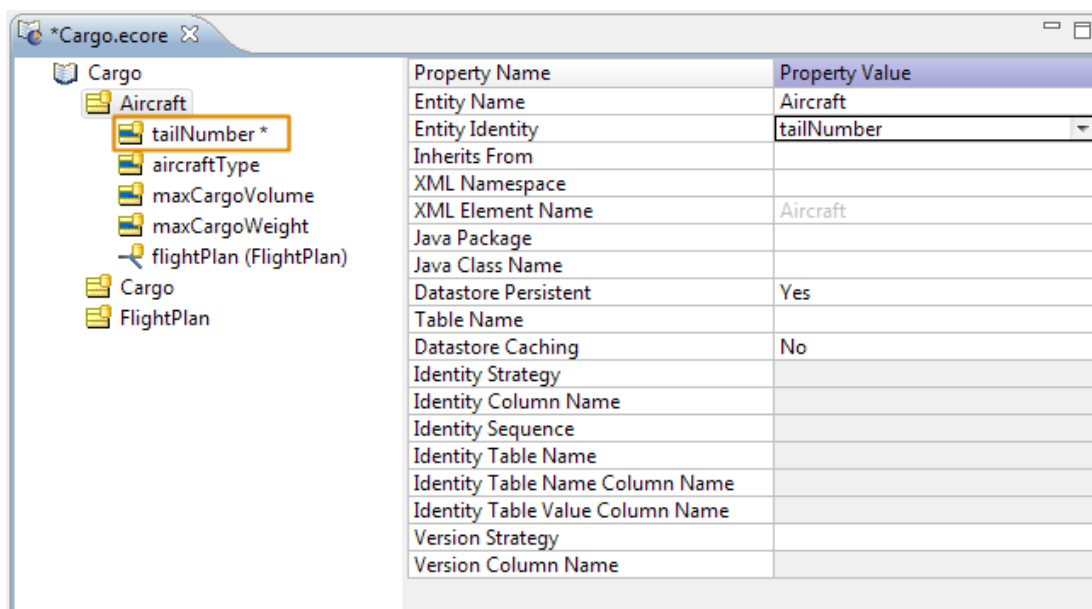
Select `tailNumber` from the drop-down menu, as shown:

**Figure 13: Selecting the Entity Identity**



Once selected as the identity for the `Aircraft` entity, `tailNumber` displays an asterisk to the right of its attribute name in the Vocabulary tree-view, as shown:

**Figure 14: Aircraft Entity with `tailNumber` Identity Selected**



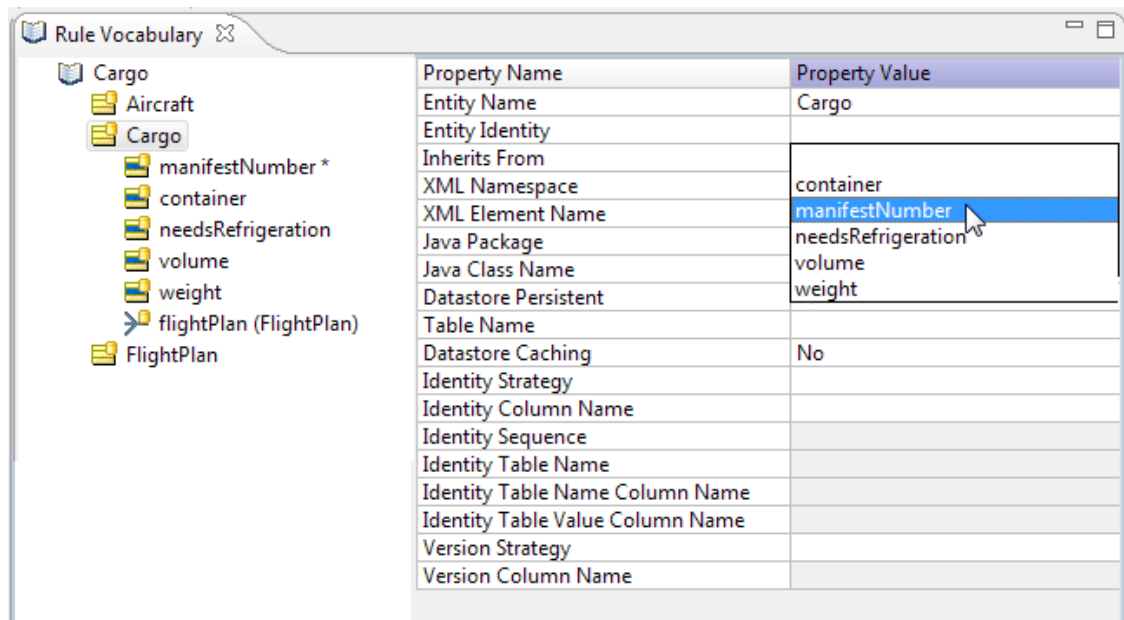
---

**Note:** To expose the asterisk in the tree-view, you might need to “click away” from the field, or press **Enter** to remove focus from the field.

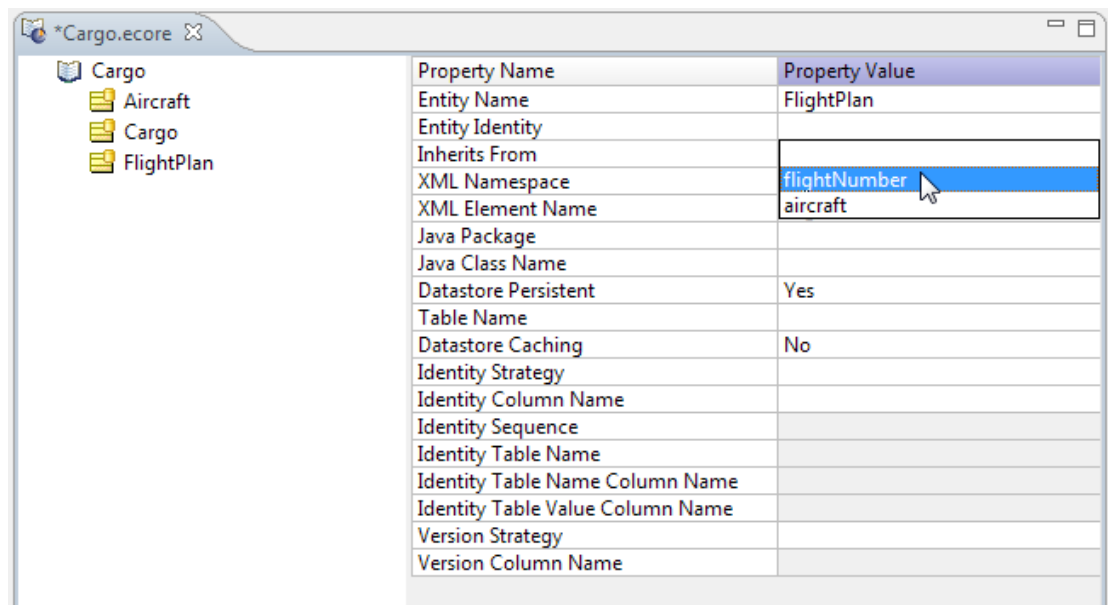
---

We need to repeat this for each entity in our Vocabulary because each entity will become a table in our database schema, and each table needs a defined identity.

**Figure 15: Cargo Entity with manifestNumber Identity Selected**

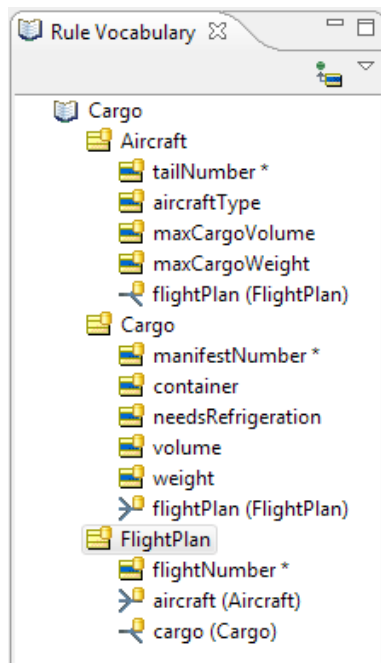


**Figure 16: FlightPlan Entity with flightNumber Identity Selected**



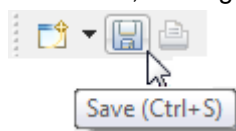
Once an identity has been selected for each entity, our `Cargo.ecore` will have one attribute marked with an asterisk assigned in each entity, as shown:

Figure 17: Cargo .ecore with Identities Designated for All Entities



Our Vocabulary is now ready to generate a corresponding schema into our new database on SQL Server.

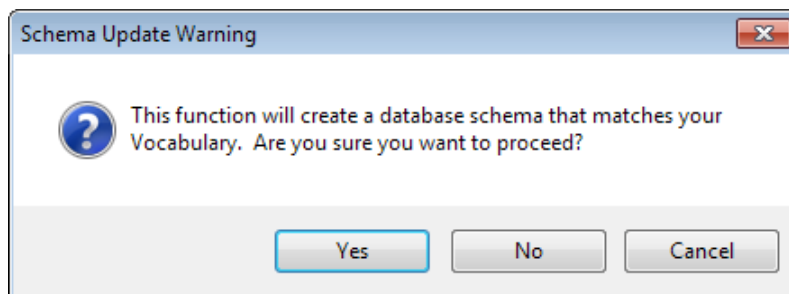
Before proceeding, save your changes to `Cargo.ecore` by choosing the menu command **File>Save**, clicking the **Save** button on the toolbar, or pressing **Ctrl+S**:



## Forward-Generating a Schema into SQL Server

Now that our Vocabulary has its identities defined, we can use it to create a corresponding schema in the `Cargo` database running on SQL Server. On the Studio menubar, select **Vocabulary > Database Access > Create/Update Database Schema**. The following alert window opens:

Figure 18: Schema Update Warning Message

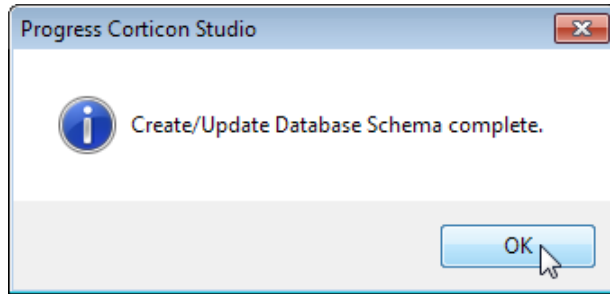


The alert warns us that if a schema already exists in the database connected to this Vocabulary, the **Create/Update Database Schema** function may change it. Since our `tutorial` database is brand new and has no schema whatsoever (yet), we can safely continue with the creation process.

Click **Yes** to proceed. If the create/update process succeeds, we'll see the following alert:

---

**Figure 19: Database Schema Update Success Message**



Click **OK** to dismiss the notification.

Let's see how that action changed the Vocabulary and the database !



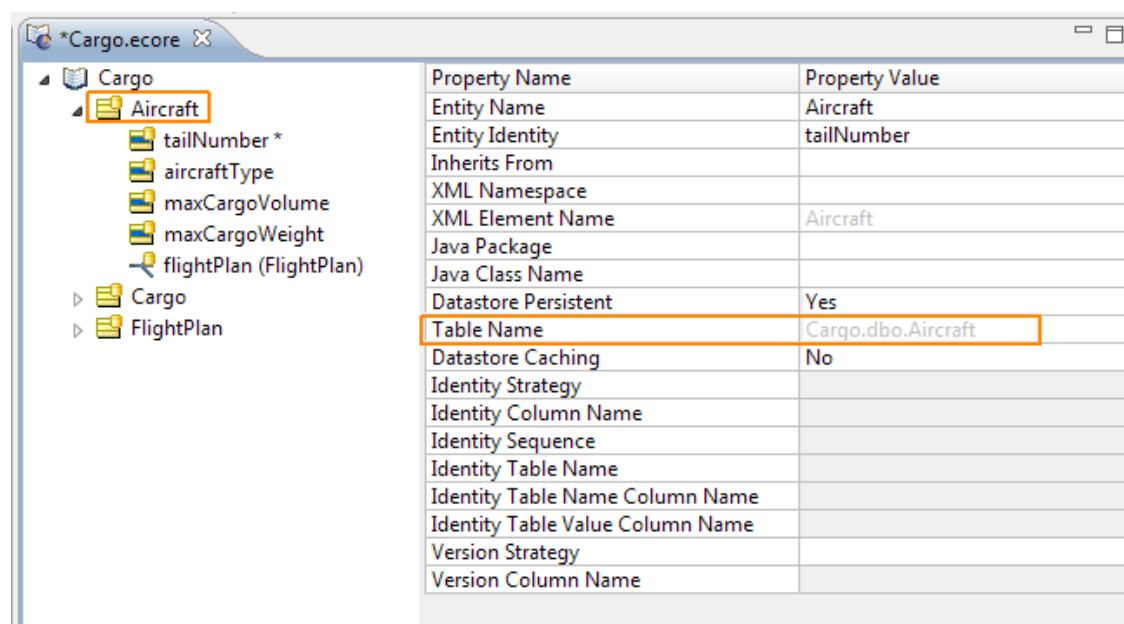


## Verifying Vocabulary mappings

The **Create/Update Database Schema** function does more than just create a schema in the `Cargo` database. It also automatically maps the resulting tables and columns in the database *back into* our Vocabulary. We can see this by selecting any node in our Vocabulary.

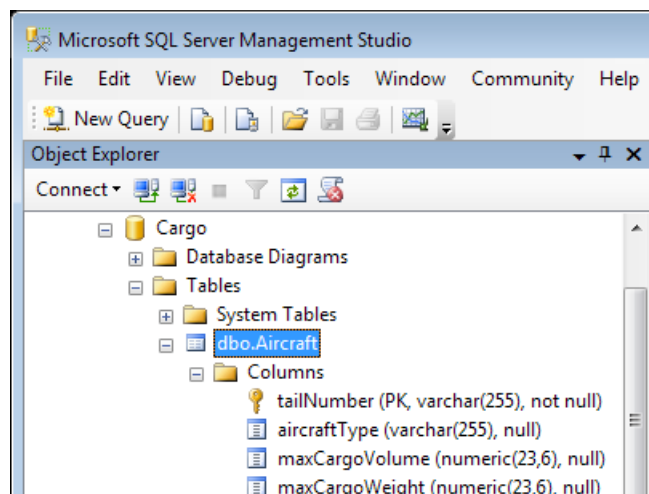
Vocabulary entities correspond to database tables, therefore the new database table `Aircraft` has been automatically mapped to the `Aircraft` entity, as shown:

**Figure 20: Entity Mapping**



The Entity is mapped in a corresponding SQL Server table, as shown:

Figure 21: Entity Mapping to SQL Server Table

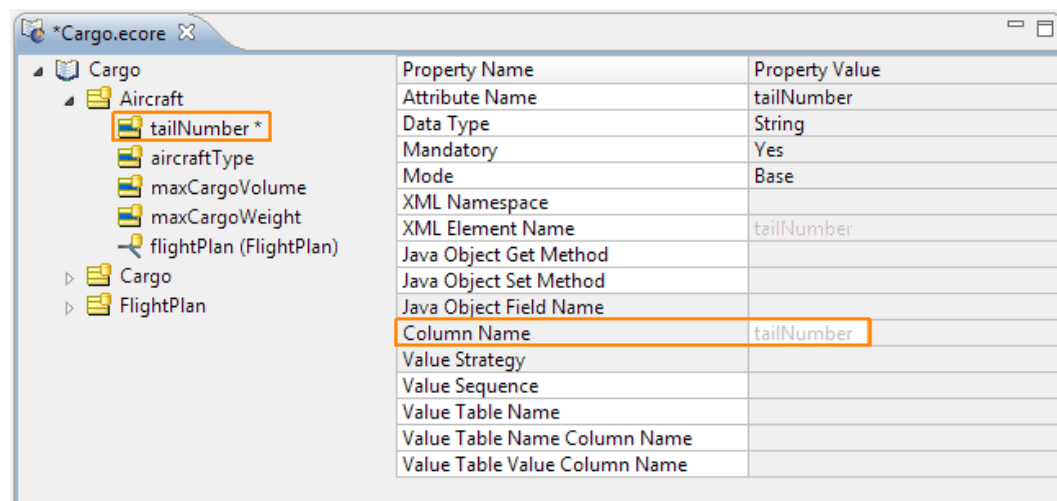


Vocabulary attributes correspond to database columns (or fields), therefore new columns/fields in database table `Aircraft` have been mapped to the attributes of entity `Aircraft`. Attribute `tailNumber`, therefore, is mapped to column/field `tailNumber`. Again, the light gray text color indicates the mapping was assigned using Studio's automatic mapping logic.

The characteristics of `tailNumber` show a key symbol and the text **PK** to indicate that it is the **Primary Key** to each row, just as we set it the Vocabulary's Entity Identity. Also it is declared as **not null** which means that the Primary Key must have a value.

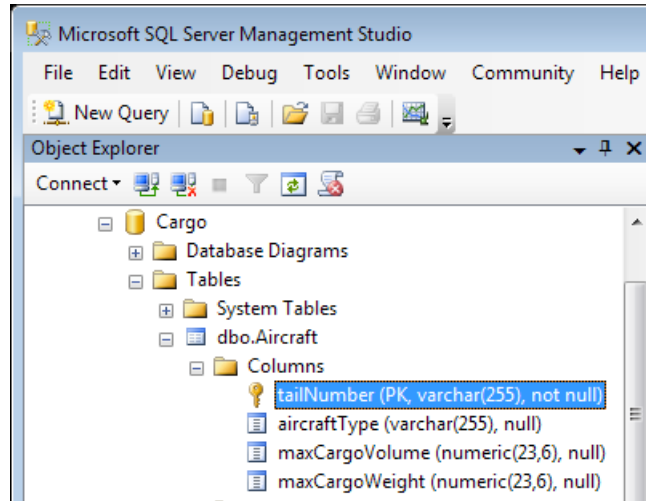
Click on an `Aircraft` attribute such as `tailNumber` to see how it was mapped to the database:

Figure 22: Attribute Mapping



The Attribute is mapped in a corresponding column in the appropriate SQL Server table, as shown:

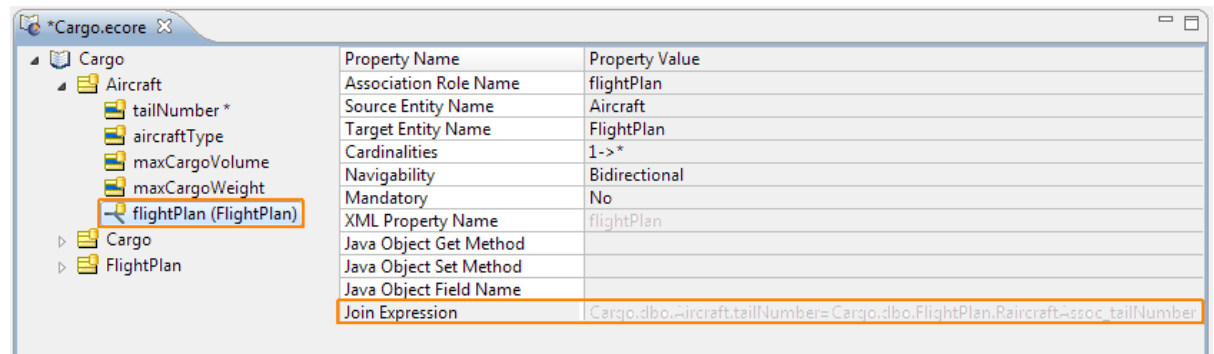
**Figure 23: Attribute Mapping to Table Column in SQL Server**



Vocabulary associations correspond to database foreign key relationships, therefore the new relationship in the database between table `Aircraft` and table `FlightPlan` has been mapped to the association between Vocabulary entity `Aircraft` and `FlightPlan`. Again, the light gray text color indicates the mapping was assigned using Studio's automatic mapping logic.

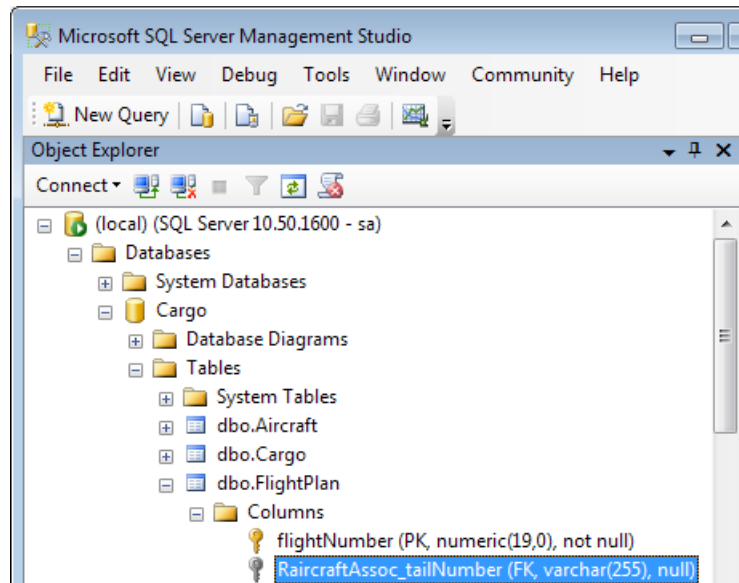
Click on the `Aircraft` association `flightPlan` to see how it was mapped to the database:

**Figure 24: Association Mapping**



The Association is mapped in a corresponding relationship between SQL Server tables, as shown:

Figure 25: Association Mapping to relationship between tables in SQL Server



As you can see from the mapping, a Vocabulary association is implemented in a relational database as a SQL Join expression. This join expression requires that the `Cargo_dbo_Aircraft.TailNumber` (primary key or identity) match the `Cargo_dbo_FlightPlan.RaircraftAssoc_tailNumber`, which is the name of the foreign key in the `dbo_FlightPlan` table. In this case, the `FlightPlan` foreign key is the `Aircraft` primary key (identity). The cardinality of the association from `Aircraft` to `FlightPlan` in the Vocabulary is one-to-many, therefore the primary key of the “one side” of the relationship becomes the foreign key of the “many side”.

There should be no need to change any of these automatic mappings. Since the database was generated directly from the Vocabulary, all the mappings will be correct.

Before continuing, take a moment to save the Vocabulary so we retain these mappings.

## Populating the database

---

We'll now use SQL Server's Management Studio to populate the database.

---

**Note:** We could use a Ruletest to load the data but we are going look at read-only functions first so we'll preload the database with some data.

---

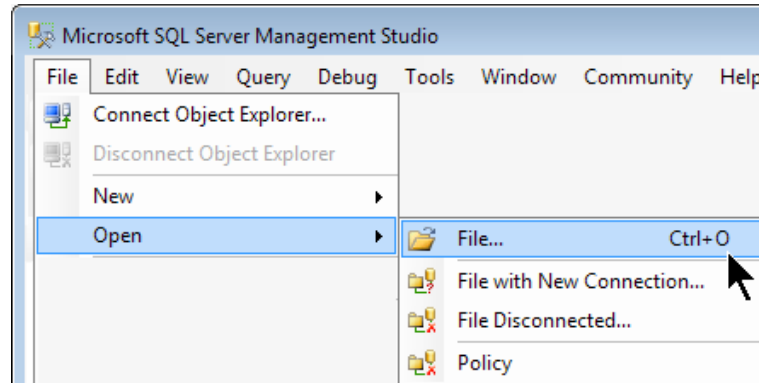
### Loading Data into a Database

The Corticon sample provides you with a `.sql` text file, `Cargo_data.sql`, that has well-formed SQL `INSERT` statements that will add some data to the tables you created in the database.

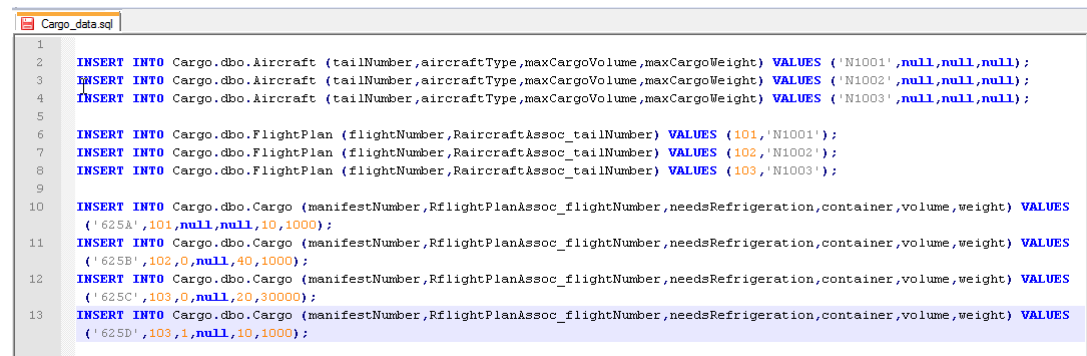
**To load the sample data into the RDBMS:**

1. In the SQL Server Management Console (still connected to the `Cargo` database), click **File > Open**, as shown:

Figure 26: Creating a New Query in SQL Server



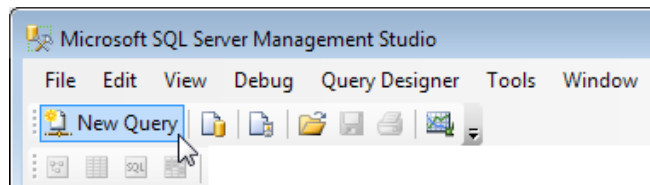
2. Navigate to the file `Cargo_data.sql`, located at `[CORTICON_WORK_DIR]\Samples\Rule Projects\Tutorial\Tutorial-Done`. Its contents look like this:

Figure 27: Contents of `Cargo_data.sql` in a text editor

3. Copy the contents of `Cargo_data.sql` in the text editor, and then paste it in the SQLQuery text area in the SQL Server Management Console.

---

**Note:** An alternative is to click **New Query**, as shown:



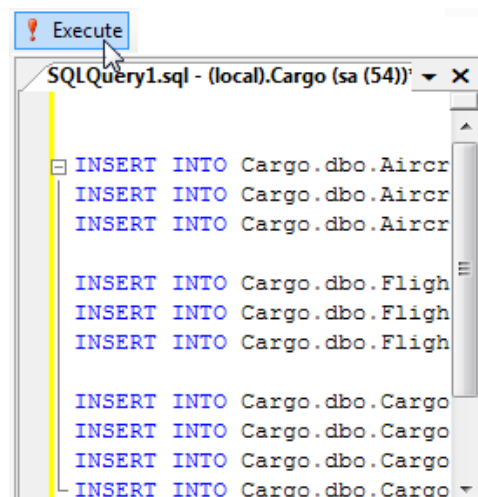
. Copy the following text, and then paste it in the SQLQuery text area:

```
INSERT INTO Cargo.dbo.Aircraft
(tailNumber,aircraftType,maxCargoVolume,maxCargoWeight) VALUES
('N1001',null,null,null);
INSERT INTO Cargo.dbo.Aircraft
(tailNumber,aircraftType,maxCargoVolume,maxCargoWeight) VALUES
('N1002',null,null,null);
INSERT INTO Cargo.dbo.Aircraft
(tailNumber,aircraftType,maxCargoVolume,maxCargoWeight) VALUES
('N1003',null,null,null);
INSERT INTO Cargo.dbo.FlightPlan (flightNumber,RaircraftAssoc_tailNumber)
VALUES (101,'N1001');
INSERT INTO Cargo.dbo.FlightPlan (flightNumber,RaircraftAssoc_tailNumber)
VALUES (102,'N1002');
INSERT INTO Cargo.dbo.FlightPlan (flightNumber,RaircraftAssoc_tailNumber)
VALUES (103,'N1003');
INSERT INTO Cargo.dbo.Cargo
(manifestNumber,RflightPlanAssoc_flightNumber,needsRefrigeration,container,volume,weight)
VALUES ('625A',101,null,null,10,1000);
INSERT INTO Cargo.dbo.Cargo
(manifestNumber,RflightPlanAssoc_flightNumber,needsRefrigeration,container,volume,weight)
VALUES ('625B',102,0,null,40,1000);
INSERT INTO Cargo.dbo.Cargo
(manifestNumber,RflightPlanAssoc_flightNumber,needsRefrigeration,container,volume,weight)
VALUES ('625C',103,0,null,20,30000);
INSERT INTO Cargo.dbo.Cargo
(manifestNumber,RflightPlanAssoc_flightNumber,needsRefrigeration,container,volume,weight)
VALUES ('625D',103,1,null,10,1000);
```

---

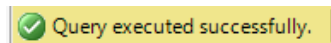
4. Click **Execute**, as shown:

**Figure 28: Executing the Query**



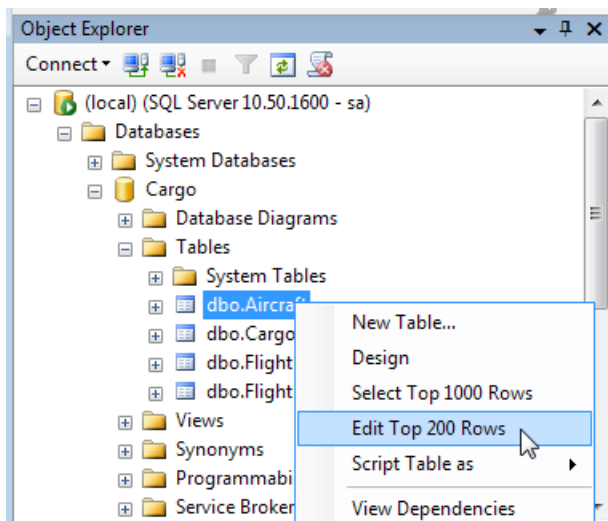
When the query executes, it displays success in its status bar, as shown:

Figure 29: Successful Query Execution



## Viewing Data in the Database

We can review the database rows that the `INSERT` commands created to see that they are as we expected. In the SQL Server Management Studio, expand the tree to **Databases: Cargo : Tables**, and then right-click on a table to choose **Edit Top 200 Rows**, as shown for the `dbo.Aircraft` table:

Figure 30: Choosing to view rows in the `dbo.Aircraft` table

The right panel opens a tab to display the following data:

Figure 31: Rows in the `Aircraft` table

| dbo.Aircraft |            |              |                |                |
|--------------|------------|--------------|----------------|----------------|
|              | tailNumber | aircraftType | maxCargoVolume | maxCargoWeight |
| ▶            | N1001      | NULL         | NULL           | NULL           |
|              | N1002      | NULL         | NULL           | NULL           |
|              | N1003      | NULL         | NULL           | NULL           |
| *            | NULL       | NULL         | NULL           | NULL           |

Similarly, choosing the same command on the `dbo.Cargo` table, opens a tab to display the following data:

Figure 32: Data loaded into the `Cargo` table

| dbo.Cargo |                |           |                    |        |        |                         |
|-----------|----------------|-----------|--------------------|--------|--------|-------------------------|
|           | manifestNumber | container | needsRefrigeration | volume | weight | RflightPlanAssoc_flight |
| ▶         | 525A           | NULL      | NULL               | 10     | 1000   | 101                     |
|           | 625B           | NULL      | False              | 40     | 1000   | 102                     |
|           | 625C           | NULL      | False              | 20     | 30000  | 103                     |
|           | 625D           | NULL      | True               | 10     | 1000   | 103                     |
| *         | NULL           | NULL      | NULL               | NULL   | NULL   | NULL                    |

Then, choosing the same command on the `dbo.FlightPlan` table, opens a tab to display the following data:



Figure 33: Data loaded into the FlightPlan table

| dbo.FlightPlan |              |                   |
|----------------|--------------|-------------------|
|                | flightNumber | RaircraftAssoc... |
| ▶              | 101          | N1001             |
|                | 102          | N1002             |
|                | 103          | N1003             |
| *              | NULL         | NULL              |



---

## Using database access in Studio tests

---

In this section of the EDC tutorial, we will learn how to configure and execute a Studio Test with read-only database access, and then with read/update database access.

For details, see the following topics:

- [Read-only database access in a Studio test](#)
- [Read/Update database access in a Studio test](#)

### Read-only database access in a Studio test

To execute a Test in Studio, we will use the Rulesheet (.ers file) built from the Vocabulary we have been preparing. If you have previously completed the *Tutorial for Corticon Studio – Basic Rule Modeling*, then you are familiar with the rules in `Cargo.ers`. This basic Rulesheet applies well to this tutorial.

#### Opening a Rulesheet in Studio

In the **Rule Project Explorer**, double-click on `Cargo.ers`. The Rulesheet opens, as shown:

Figure 34: Rulesheet Cargo.ers as Completed in the Basic Rule Modeling Tutorial

| Conditions |                          | 0 | 1        | 2    | 3       | 4 |
|------------|--------------------------|---|----------|------|---------|---|
| a          | Cargo.weight             | - | <= 20000 | -    | > 20000 | - |
| b          | Cargo.volume             | - | -        | > 30 | <= 30   | - |
| c          | Cargo.needsRefrigeration | - | -        | -    | -       | T |
| d          |                          |   |          |      |         |   |

| Actions         |                 |  |          |          |             |
|-----------------|-----------------|--|----------|----------|-------------|
| Post Message(s) |                 |  |          |          |             |
| A               | Cargo.container |  | standard | oversize | heavyweight |
| B               |                 |  |          |          |             |
| C               |                 |  |          |          |             |

|  |  | Overrides |        |
|--|--|-----------|--------|
|  |  |           | {1, 4} |
|  |  |           | {1, 3} |

| Ref | ID | Post | Alias | Text   |
|-----|----|------|-------|--|
| 1   |    | Info | Cargo | Cargo weighing <= 20,000 kilos must be packaged in a standard container.                                 |
| 2   |    | Info | Cargo | Cargo with volume > 30 cubic meters must be packaged in an oversize container.                           |
| 3   |    | Info | Cargo | Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be packaged in a standard container. |
| 4   |    | Info | Cargo | Cargo requiring refrigeration must be packaged in a reefer container.                                    |

Reflecting on the *Basic Rule Modeling Tutorial*, the business purpose of this Rulesheet is to examine the weight, volume, and nature of cargo to determine how it should be packaged for shipment. More accurately, for any given cargo, the rules test its weight, volume, and refrigeration requirements, and then assigns a value to its `container` attribute.

From our inspection of the database tables in **Viewing Data in the Database**, we know that the `Cargo` table in our database already contains weight and volume data for each record (we can see this data in [Figure 32](#) on page 48).

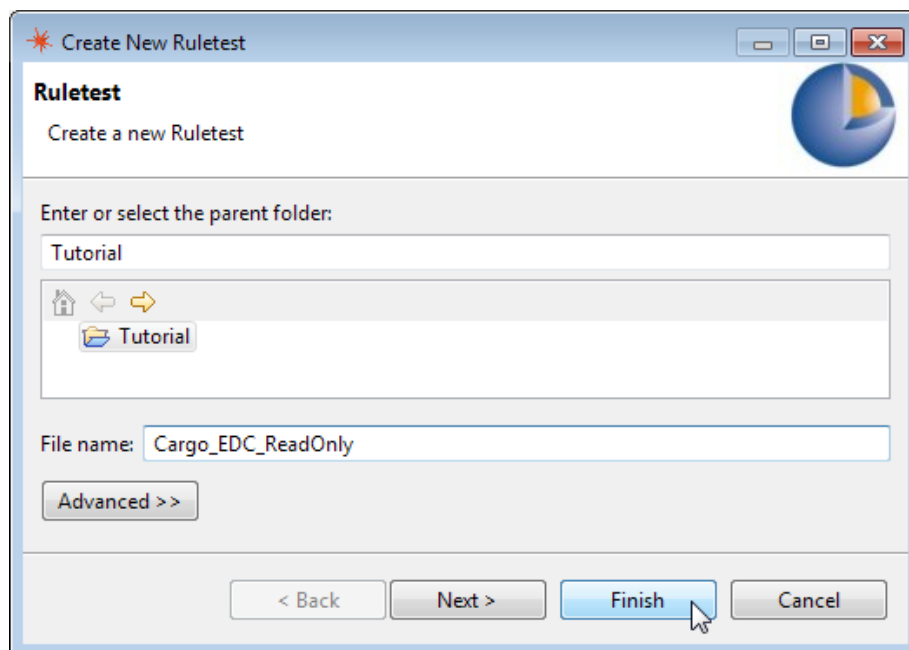
Given the structure of our Vocabulary and database, Studio should be able to “look up” a particular `Cargo` record if it knows its `manifestNumber` (primary key). To test this functionality, we create a simple Test that uses read-only database access to retrieve `Cargo` data from the database, and then use it to evaluate the rules in the previous figure.

### Configuring a Studio Test for Read-Only Database Access

Open a new Ruletest, as follows:

1. Select the menu command **File > New > Ruletest**.
2. In the **Create New Ruletest** dialog, select the `Tutorial Project`, and enter a file name, such as `Cargo_EDC_ReadOnly`, as shown:

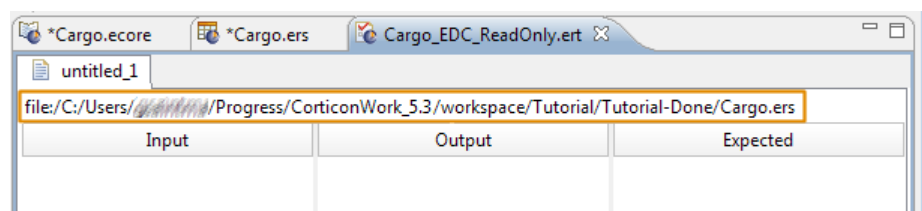
Figure 35: New Ruletest in the Tutorial Project



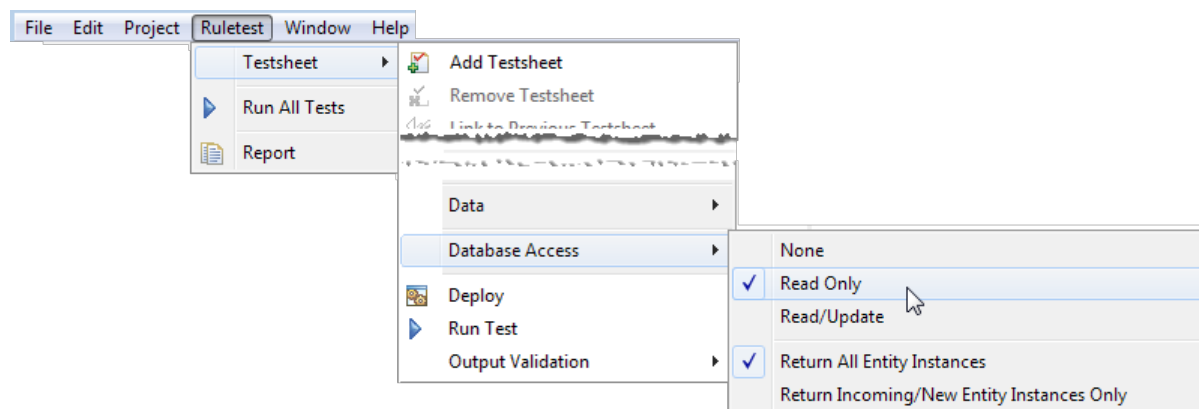
### 3. Click **Finish**.


By default, the new Ruletest will bind to the single Rulesheet in the project, `Cargo.ers` as its 'test subject', and displays its name and location the Testsheet's header, as shown:

Figure 36: A New Testsheet with the Rulesheet to be Tested in its Header



To set this Ruletest's Testsheet `Untitled-1` for read-only access to our database, select the menu option **Ruletest > Testsheet > Database Access > Read Only**, as shown:



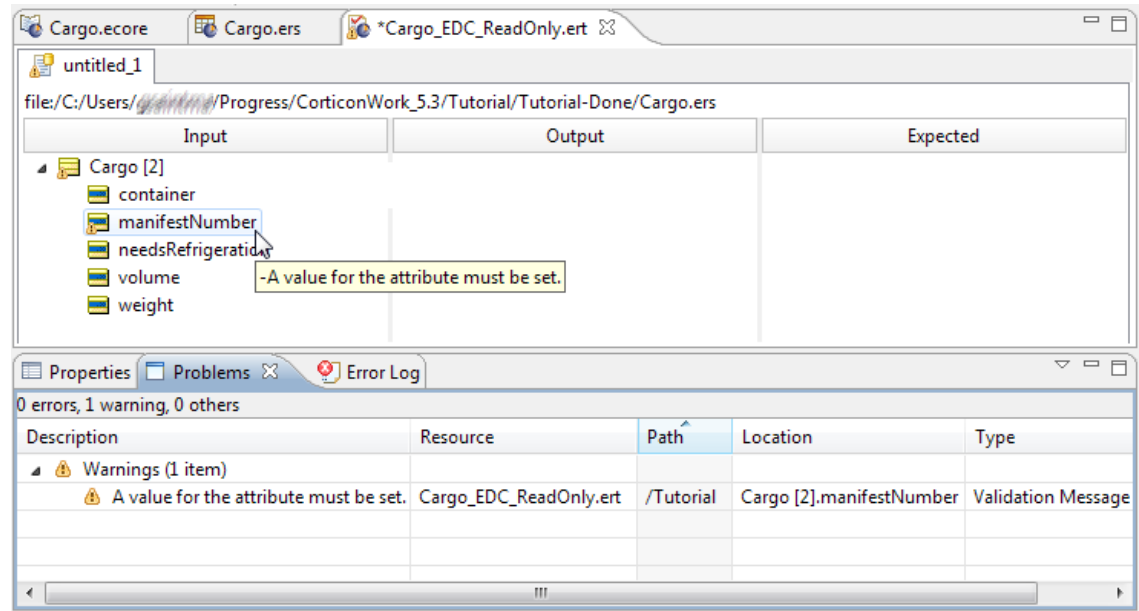
The Testsheet's tab displays a modified icon:  -- a yellow cylinder representing a database. That shows that it is set to **Read Only** access, so we will be able to extract data as needed from the database to use in rule execution.

### Defining and Executing the Read Only Test

Drag a new `Cargo` entity from the Vocabulary and drop it on the Ruletest's Input area.

Notice the alert symbol  that decorates the `Cargo` and its `manifestNumber` attribute, as shown:

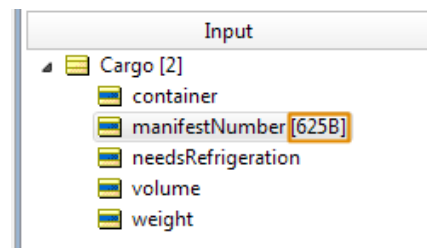
**Figure 37: Tooltip Indicating That Key Data is Missing**



Hovering your cursor over the attribute restates the warning message that we need to provide data for the `manifestNumber` attribute. That's because `manifestNumber` is the identity value for the `Cargo` entity and also the primary key for the `Cargo` table in our database.

Double-click on `manifestNumber` and then enter `625B`, a value in the `Cargo` table, in the entry area to the right of the attribute name, as shown:


**Figure 38: Key Data Entered for `manifestNumber`**



With key data entered, the text turns black, indicating that there are no further errors.

Finally, let's intentionally remove an attribute. That will compel Studio to retrieve the attribute and its value for the specified primary key. Click on `volume` in the Input column, and then press the **Delete** key.

Execute the Test by selecting the **Ruletest > Run All Tests** menu command.

The Output shows that the `volume` attribute was retrieved. Its icon is  to show that it came from the database.

**Figure 39: Read-Only Testsheet that Retrieved a Missing Attribute and All of the Record's Values**

| Input   | Output  | Expected |
|---|---|----------|
| <div>Cargo [2]</div> <div> <div>container</div> <div>manifestNumber [625B]</div> <div>needsRefrigeration</div> <div>weight</div> </div> | <div>Cargo [2]</div> <div> <div><b>container [oversize]</b></div> <div>manifestNumber [625B]</div> <div>needsRefrigeration [false]</div> <div>volume [40]</div> <div>weight [1000]</div> </div> |          |

| Severity | Message  | Entity   |
|----------|--|----------|
| Info     | Cargo with volume > 30 cubic meters must be packaged in an oversize container. | Cargo[2] |

**Understanding the Missing-attribute Test Example**

The following sequence of events occurred when we executed this test:

1. Corticon attempted to execute the rules in `Cargo.ers` using the data provided in the Input Ruletest.
2. The rules expects `Cargo.volume` but that attribute is not in the Input Ruletest.
3. Because the Test is set to **Read Only** database access, Studio connects to the database configured in the Vocabulary `Cargo.ecore`.
4. Studio uses the “seed” data available to it to query the connected database. As we have specified a unique `Cargo` entity (identified by its primary key, the `manifestNumber`) that is manifest 625B.
5. Studio executes an SQL query to retrieve all the attributes it needs for the rules, specifically `Cargo.volume`.
6. With necessary data in memory, Studio evaluates and fires the rules. We can see from the message posted in the **Rule Messages** window that one of the rules has fired.

This Test used **Read Only** database access, so even though the evaluated rules assigned a value to `container` (highlighted in bold text), that data is not written back to the database. The original `container` data (`null`) stays unchanged in the database, as shown:

**Figure 40: Data in the Cargo Table after the Read-Only Test**

| dbo.Cargo |                |           |                    |        |        |                        |
|-----------|----------------|-----------|--------------------|--------|--------|------------------------|
|           | manifestNumber | container | needsRefrigeration | volume | weight | RflightPlanAssoc_fligh |
| ▶         | 525A           | NULL      | NULL               | 10     | 1000   | 101                    |
|           | 625B           | NULL      | False              | 40     | 1000   | 102                    |
|           | 625C           | NULL      | False              | 20     | 30000  | 103                    |
|           | 625D           | NULL      | True               | 10     | 1000   | 103                    |
| *         | NULL           | NULL      | NULL               | NULL   | NULL   | NULL                   |

## Read/Update database access in a Studio test

A Testsheet that uses **Read/Update** access changes information in the database records.

We are moving beyond just inquiring the database while staying in the scope of Corticon Studio's working memory. We are taking on the responsibilities of maintaining database records -- adding, deleting, and modifying as determined by the execution of Corticon rules. To help ease into this transition, let's see how our rules will behave *before* we execute Tests in **Read/Update** database access mode.

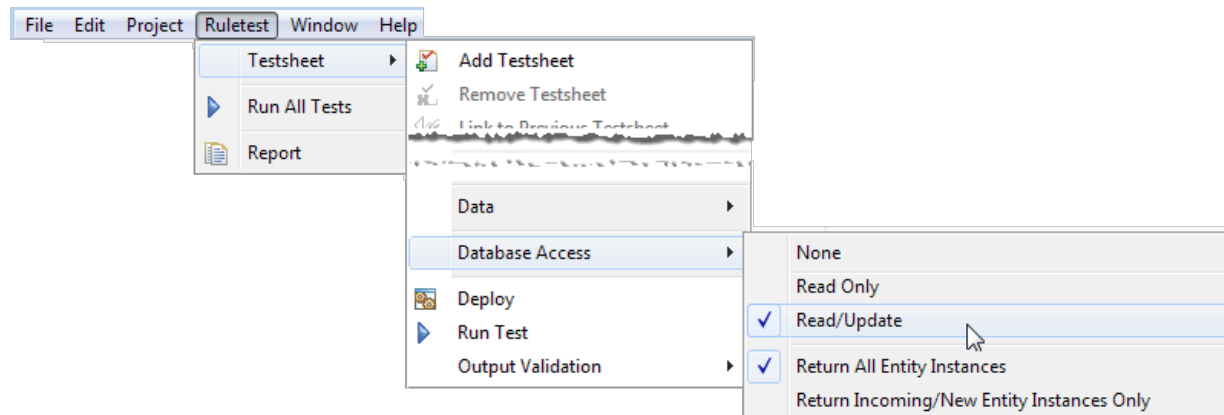
### Configuring a Studio Test for Read/Update Database Access

Open a new Ruletest, as follows:

1. Select **File>New>Ruletest** on the Studio menubar.
2. In the **Create New Ruletest** dialog, select the `Tutorial Project`, and enter a file name, such as `Cargo_EDC_ReadUpdate`, and then click **Finish**.

The new Ruletest will bind to the single Rulesheet in the project, `Cargo.ers`.

To set this Ruletest for read/update access to our database, select the menu option **Ruletest > Database Access > Read/Update**, as shown:



The Testsheet's tab displays a modified icon: -- a red arrow pointing toward the yellow cylinder. That shows that it is set to **Read/Update** access, so that we will be able to extract data as needed from the database to use in rule execution, *and then write changes and additional data back to the database*.

### Defining and Executing the Read/Update Test

Drag a new `Cargo` entity from the Vocabulary and drop it on the Ruletest. Just as we saw in Read-Only mode, the `Cargo` and its `manifestNumber` attribute are decorated with an alert because it requires a value (remember we set this attribute as the primary application identity key).

Double-click on `manifestNumber` and then enter `625B` in the entry area to the right of the attribute name.

With key data entered, the text turns black, indicating that there are no further errors.

Let's do the added step we did in the previous test. In the Input column, select `volume`, and then press **Delete**.

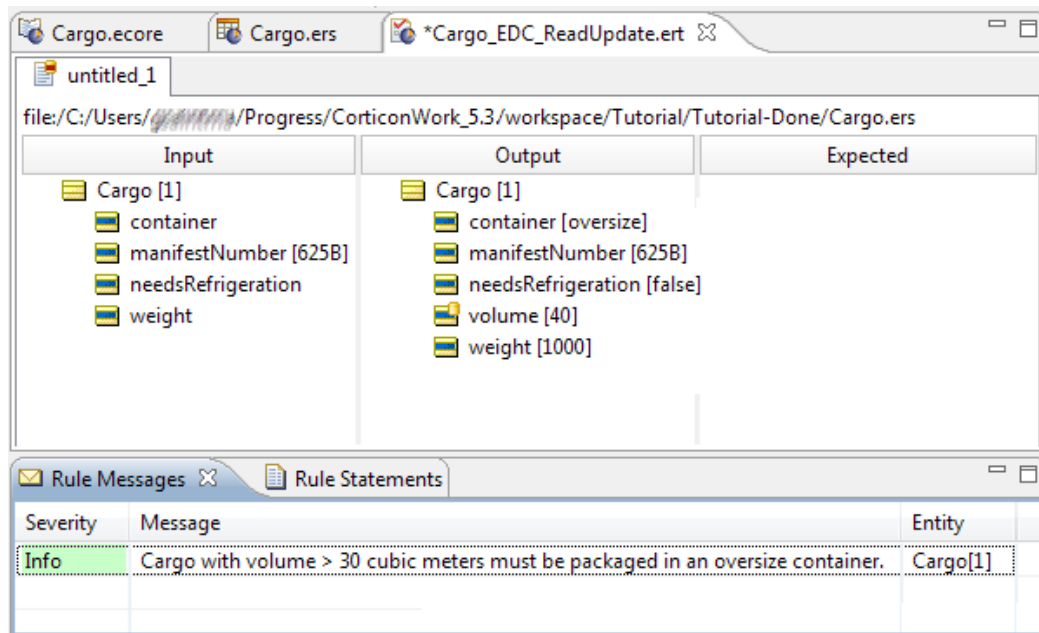
Now let's run the test.

### Executing the Read/Update Test



In Corticon Studio, execute the Testsheet by selecting the **Ruletest > Testsheet > Test > Run Test** menu command.

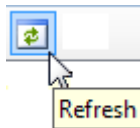
**Figure 41: Read/Update Ruletest that Retrieved a Missing Attribute and All of the Record's Values**



The results, shown above, look a lot like they did before ([Read-Only Testsheet that Retrieved a Missing Attribute and All of the Record's Values](#)). The only difference is that the `container` attribute, and its value, `oversize`, is not in bold text.

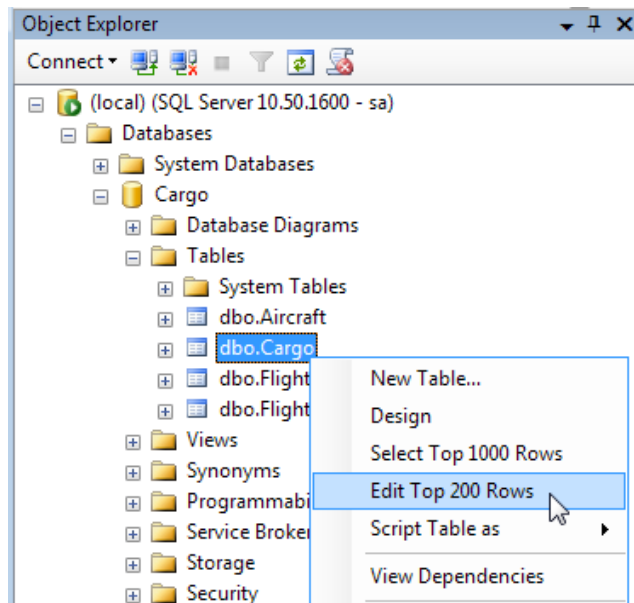
But what has happened in our database? What was the impact of the Read/Update Test on the database?

In the Microsoft SQL Server Management Studio, click **Refresh**, as shown:



Expand the tree to **Databases: Cargo : Tables**, right-click on **Cargo**, and then choose **Edit Top 200 Rows**, as shown:

**Figure 42: Accessing the Data in the Cargo table after Read/Update**



The right panel lists the records in the `Cargo` table, as shown:

#### Data in the `Cargo` table after Read/Update

|   | manifestNumber | container | needsRefrigeration | volume | weight | RflightPlanAssociation |
|---|----------------|-----------|--------------------|--------|--------|------------------------|
|   | 625A           | NULL      | NULL               | 10     | 1000   | 101                    |
| ▶ | 625B           | oversize  | False              | 40     | 1000   | 102                    |
|   | 625C           | NULL      | False              | 20     | 30000  | 103                    |
|   | 625D           | NULL      | True               | 10     | 1000   | 103                    |

Why was only the second row updated? Because our test `Cargo` was only for `manifestNumber` 625B.

#### Performing a Test with Multiple Scenarios

One Corticon Studio Testsheet can contain many data scenarios. The following example shows how to perform multiple tests that also update the database.

1. Copy the `Cargo` entity onto the testsheet four times.
2. Delete all the attributes except each `manifestNumber`.
3. Run all tests.

The results show that the additional attributes were retrieved, and that the container type was decided:

Figure 43: Read/Update Testsheet that Retrieved Missing Attributes and Every Record's Values

| Input   | Output   | Expected |
|---|--|----------|
| <div>Cargo [1]<div>manifestNumber [625A]</div></div> <div>Cargo [2]<div>manifestNumber [625B]</div></div> <div>Cargo [3]<div>manifestNumber [625C]</div></div> <div>Cargo [4]<div>manifestNumber [625D]</div></div> | <div>Cargo [1]<div>container [standard]<div>manifestNumber [625A]<div>needsRefrigeration<div>volume [10]<div>weight [1000]</div></div></div></div></div></div> <div>Cargo [2]<div>container [oversize]<div>manifestNumber [625B]<div>needsRefrigeration [false]<div>volume [40]<div>weight [1000]</div></div></div></div></div></div> <div>Cargo [3]<div>container [heavyweight]<div>manifestNumber [625C]<div>needsRefrigeration [false]<div>volume [20]<div>weight [30000]</div></div></div></div></div></div> <div>Cargo [4]<div>container [reefer]<div>manifestNumber [625D]<div>needsRefrigeration [true]<div>volume [10]<div>weight [1000]</div></div></div></div></div></div> |          |

| Severity | Message   | Entity   |
|----------|---|----------|
| Info     | Cargo weighing <= 20,000 kilos must be packaged in a standard container.                            | Cargo[1] |
| Info     | Cargo with volume > 30 cubic meters must be packaged in an oversize container.                      | Cargo[2] |
| Info     | Cargo weighing > 20,000 kilos, with volume <= 30 cubic meters, must be packaged in a heavyweight... | Cargo[3] |
| Info     | Cargo requiring refrigeration must be packaged in a reefer container.                               | Cargo[4] |

In the Microsoft SQL Server Management Studio, click **Refresh**.

Expand the tree to **Databases: Cargo : Tables**, right-click on **Cargo**, and then choose **Edit Top 200 Rows**.

The records in the Cargo table show that all the container values were updated, as shown:

Figure 44: Data in the Cargo table after Read/Update of All Records

|   | manifestNum... | container   | needsRefrigeration | volume | weight | RflightPlanAssoc_flightNumber |
|---|----------------|-------------|--------------------|--------|--------|-------------------------------|
| ▶ | 625A           | standard    | NULL               | 10     | 1000   | 101                           |
|   | 625B           | oversize    | False              | 40     | 1000   | 102                           |
|   | 625C           | heavyweight | False              | 20     | 30000  | 103                           |
|   | 625D           | reefer      | True               | 10     | 1000   | 103                           |
| * | NULL           | NULL        | NULL               | NULL   | NULL   | NULL                          |



---

## Importing an attribute's possible values from database tables

---

Your database connection can also provide designers and testers of Rulesheets and Ruletests with lists of possible values, also known as *enumerations*. While these lists can be created and maintained by hand on the Custom Data Types tab of a Vocabulary, you can retrieve lists from the connected database.

Consider the general behavior of enumerations, especially when retrieving labels and values from a database:

- There can be only one instance of any label and any value in the list, whether created manually or imported. An exception will make the Vocabulary invalid. The database retrieval will work as expected but you will have to groom the results to make the lists valid. You can get optimal results when your database source prevents duplicates in the table columns you are using for your values or label-value pairs.
- If you chose a label in a Rulesheet and that label is no longer available after an update, an error will occur. Any Rulesheet expressions that refer to the defunct label will be flagged as invalid. You must update the Rulesheet expressions to correct the problem.
- If you chose a label in a Rulesheet and that label takes on a different value after an update, the current value is what is evaluated.
- The value assigned - whether directly or as the label's value - at the time of deployment does not change thereafter on the server.

As you will see as we create some examples, it is good practice to ensure that the data types of the retrieved values in the database are consistent with the Custom Data Type, and then extend the corresponding base data value in the attribute.

## Procedures

The steps to implement custom data types retrieved from a database are, in summary, as follows:

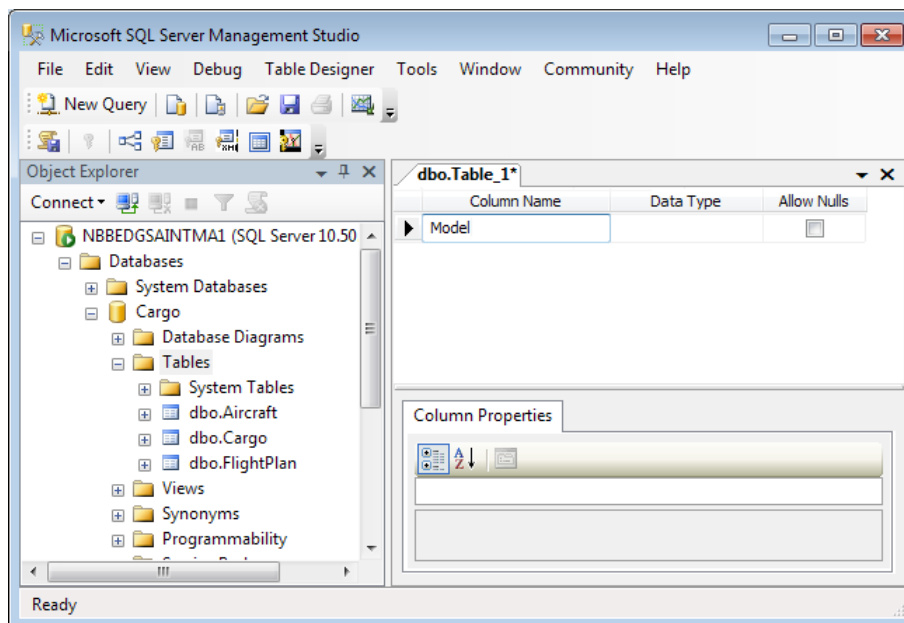
- A - Create or locate the database table and columns you want to retrieve.
- B - Verify the database connectivity, and then import its metadata.
- C - Define the Custom Data Type lookup information.
- D - Import the enumeration elements.
- E - Check the lists for duplicates.
- F - Set the Data Type of appropriate attributes to the Custom Data Type.
- G - Verify that the list functions correctly.

### A - Create or locate the database table and columns you want to retrieve.

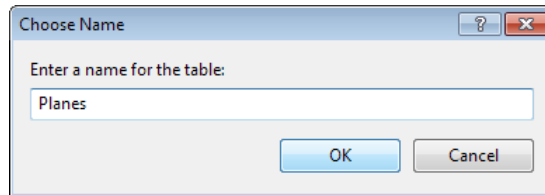
**Note:** This step uses the procedures detailed in [Populating the database](#) on page 45 when we set up the database.

Continuing with our `Cargo` database in SQL Server, we'll add two tables to demonstrate both value-only and label+value enumerations:

1. Start the SQL Server Management Studio, and then expand the tree for **Databases : Cargo : Tables**. Right-click on **Tables** and choose **New Table**. Enter `Model` as the only column name, as shown:



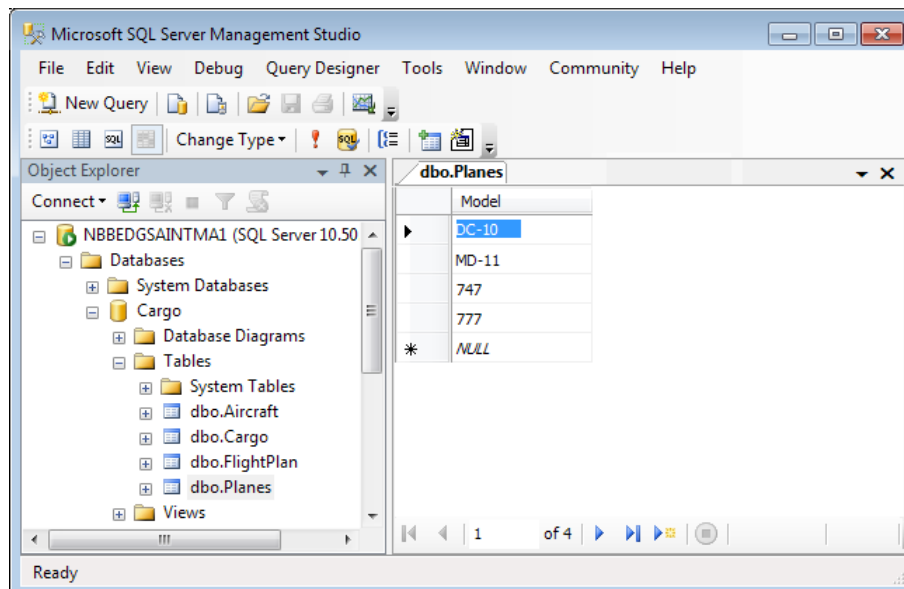
2. Choose the menu command **File > Save Table\_1**, enter the name `Planes`, and then click **OK**.



3. Create another table, now with two columns named `planeCarrier` and `planeID`, saving it as `Carrier`.
4. Click **New Query**, copy/paste the following text, and then click **Execute**.

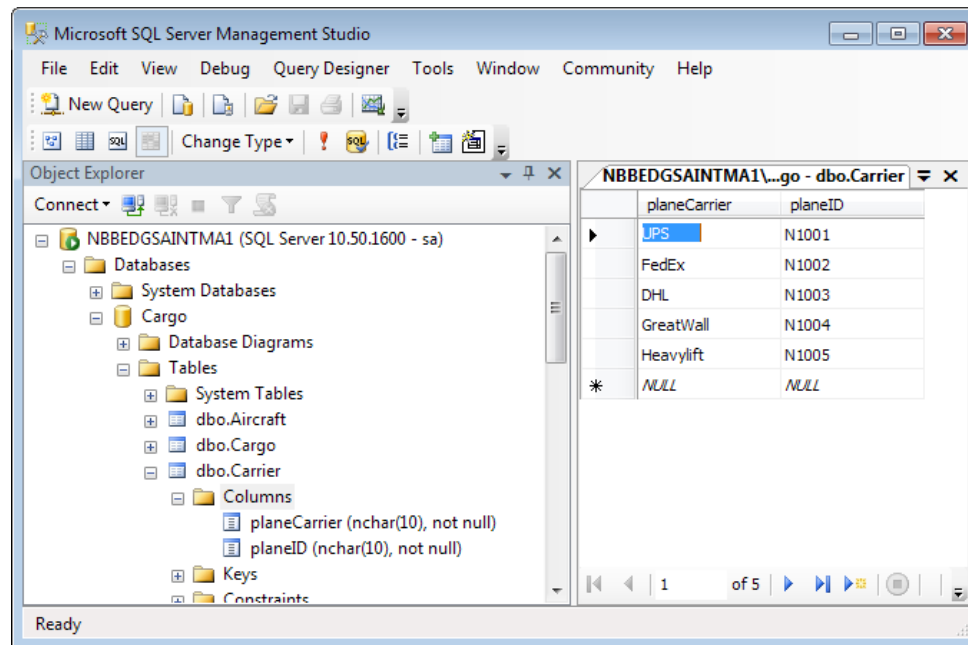
```
INSERT INTO Cargo.dbo.Planes (Model) VALUES ('DC-10');
INSERT INTO Cargo.dbo.Planes (Model) VALUES ('MD-11');
INSERT INTO Cargo.dbo.Planes (Model) VALUES ('747');
INSERT INTO Cargo.dbo.Planes (Model) VALUES ('777');
INSERT INTO Cargo.dbo.Carrier (planeCarrier,planeID) VALUES ('UPS','N1001');
INSERT INTO Cargo.dbo.Carrier (planeCarrier,planeID) VALUES ('FedEx','N1002');
INSERT INTO Cargo.dbo.Carrier (planeCarrier,planeID) VALUES ('DHL','N1003');
INSERT INTO Cargo.dbo.Carrier (planeCarrier,planeID) VALUES ('GreatWall','N1004');
INSERT INTO Cargo.dbo.Carrier (planeCarrier,planeID) VALUES ('Heavylift','N1005');
```

5. In the tree, right-click on **dbo.Planes**, and then choose **Edit Top 200 Rows**.



The **Planes** data is as we intended. It is ready for our use in the Corticon Studio.

6. Similarly, right-click on **dbo.Carrier**, and then choose **Edit Top 200 Rows**.

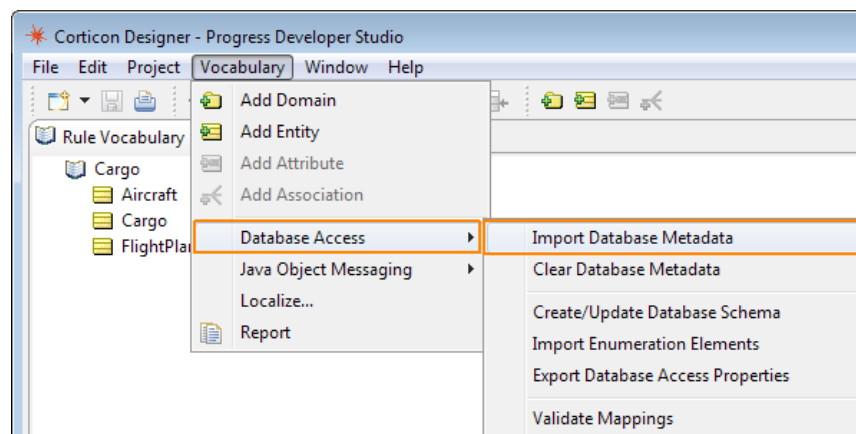


The **Carrier** data is as we intended. It is ready for our use in the Corticon Studio.

## B - Verify the database connectivity, and then import its metadata.

We want to bring the information about the table definitions into the Studio:

1. In Corticon Studio, confirm that you have the same good connection you achieved in [Connecting a Vocabulary to a database](#) on page 29
2. With `Cargo.ecore` open its editor, choose the menu command **Database Access > Import Database Metadata**, as shown:

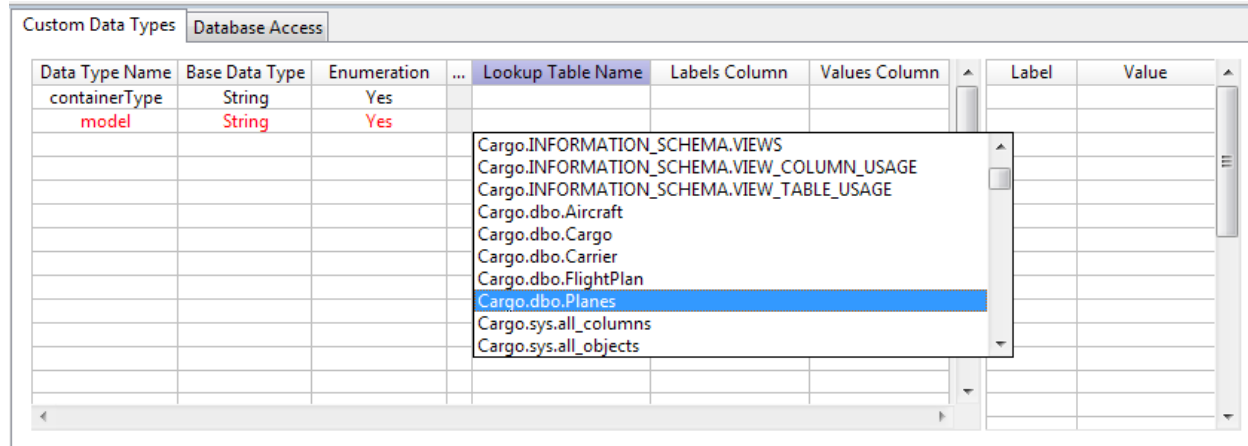


## C - Define the Custom Data Type lookup information.

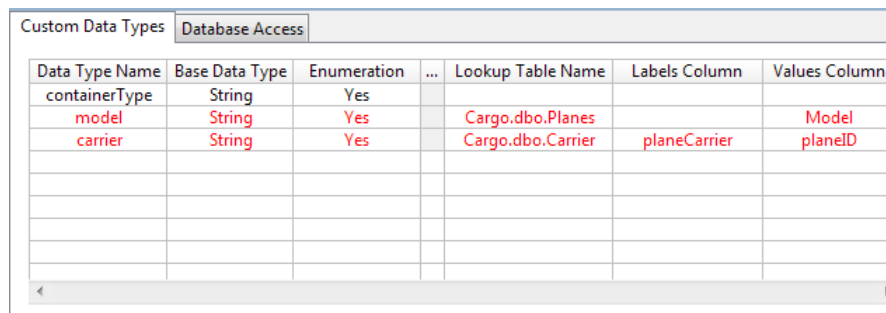
We now can specify how we want to use the data and then bind it to the appropriate database table and columns:



1. Click on `Cargo` to get to its top level, and then select the **Custom Data Types** tab.
2. Click on the next empty row, enter `model` as the Data Type Name, select `String` as the Data Type, and `Yes` as the Enumeration.
3. Click on the Lookup column in the row to expose its dropdown, and then choose `Cargo.dbo.Planes` that we imported in the database metadata.



4. We are using a values-only lookup, click on the row's Values Column to select its one database column, `Model`:
5. For the other table, click on the next empty row, enter `carrier` as the Data Type Name, select `String` as the Data Type, and `Yes` as the Enumeration.
6. Click on the Lookup Table Name in the row to expose its dropdown, and then choose `Cargo.dbo.Carrier` that we imported in the database metadata.
7. We are using a label-values lookup, so click on the row's Labels Column to select `planeCarrier`, and then in the Values Column to select `planeID`:

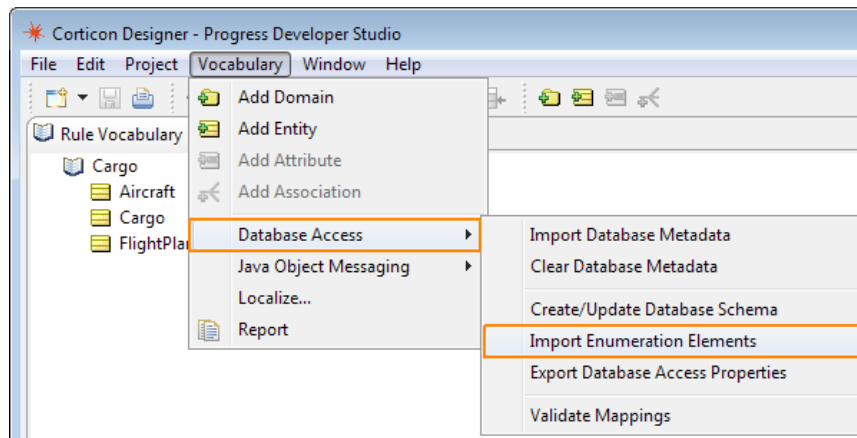


*Everything we have entered is red!* That's because Studio has no data for either of these enumeration sets.

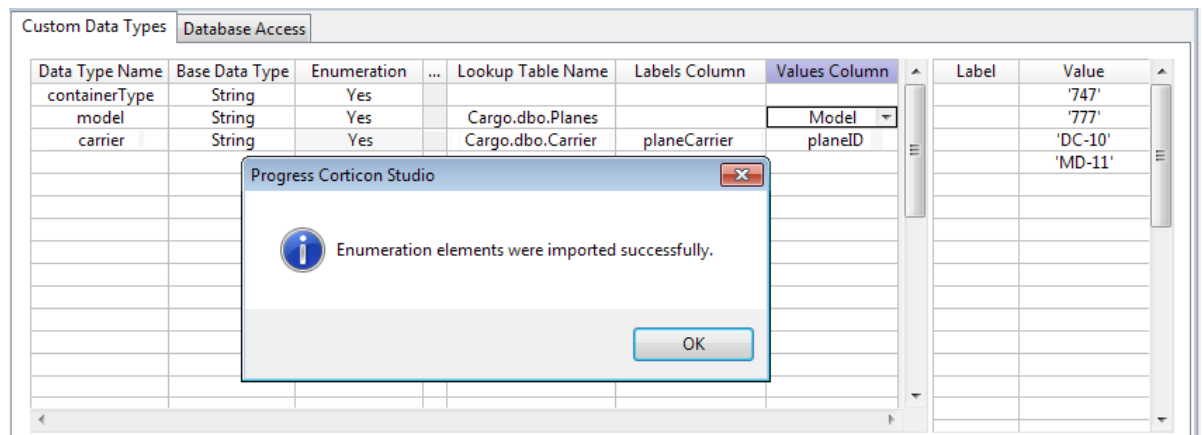
## D - Import the enumeration elements.

Once you have defined the database table and columns you want, you can retrieve the data:

1. Choose the menu command **Database Access > Import Enumeration Elements**, as shown:



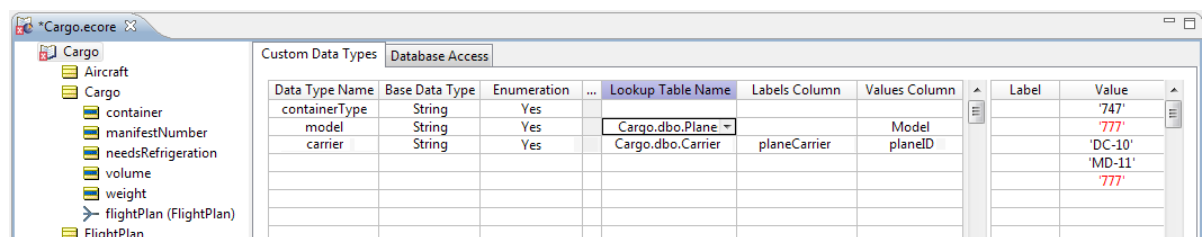
- The retrieved values are displayed in the associated Labels and Values window to the right, as shown for the `model`:



## E - Check the lists for duplicates.

Unless you enforced uniqueness in the source database. To demonstrate what happens, we'll add an existing value to the `model` enumerations.

- In the Values retrieved column, enter a new value that is already there, such as `777`, as shown:



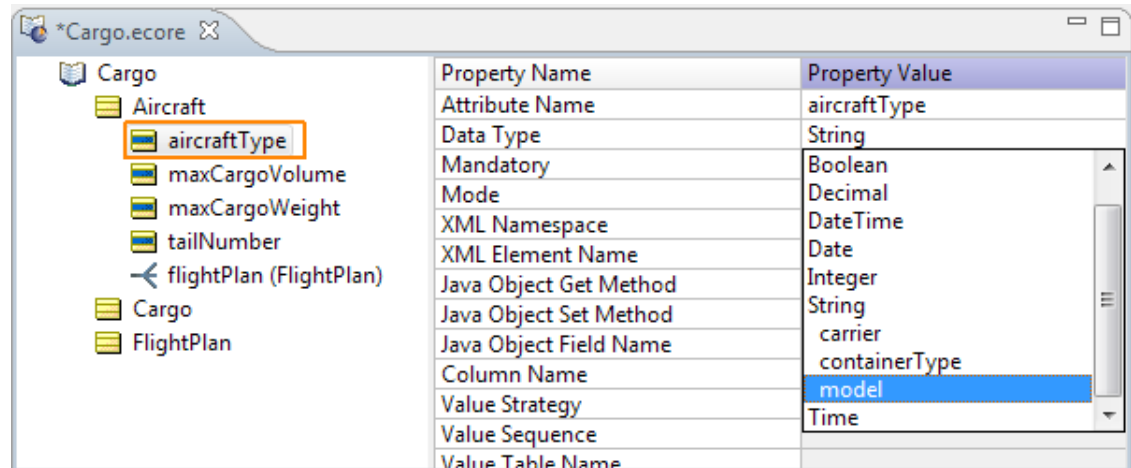
The duplicates are both highlighted in red, and the `Cargo.ecore` file is marked as being in an error state.

- Remove the line (or change it to something unique) and the Vocabulary is again valid.

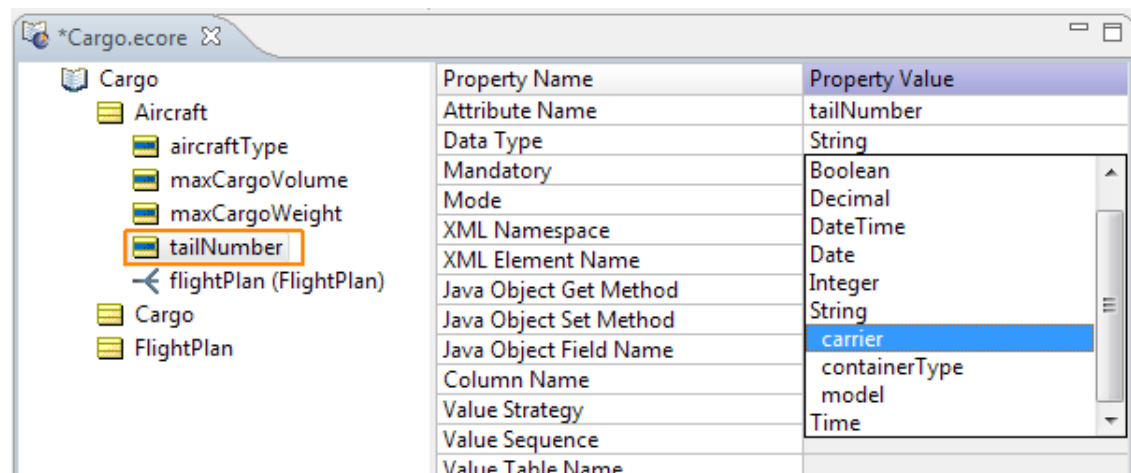
## F - Set the Data Type of appropriate attributes to the Custom Data Type.

With our enumeration lists imported from the database and verified as free of duplicate labels or values, we can link them to the attributes that will use them:

1. Aircraft.aircraftType:



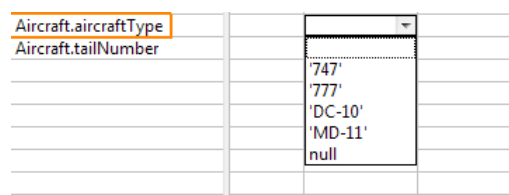
2. Aircraft.tailNumber:



## G - Verify that the list functions correctly.

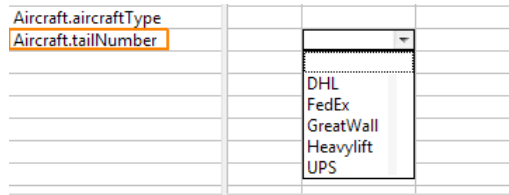
To verify that the lists perform as expected, use them in a Rulesheet or Ruletest :

1. In a Rulesheet Actions area, enter two new lines, one with the attribute syntax Aircraft.aircraftType and the other with Aircraft.tailNumber, as shown:
2. Click on the aircraftType where it intersects with column 1, as shown:



The pulldown displays our imported values, as well as blank and `null`.

3. Click on the `tailNumber` where it intersects with column 1, as shown:



The pulldown displays our imported label, as well as blank. The label is a place holder for its value.

---

**Note:** For more information about enumerations and retrieving values from databases, see:

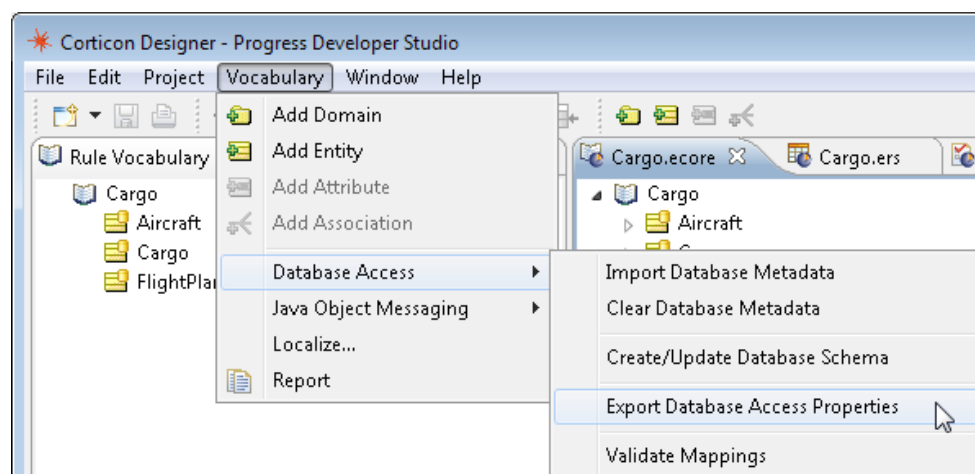
- *"Enumerated values" in the Quick Reference Guide.*
  - *"Enumerations retrieved from a database" in the Rule Modeling Guide*
-

---

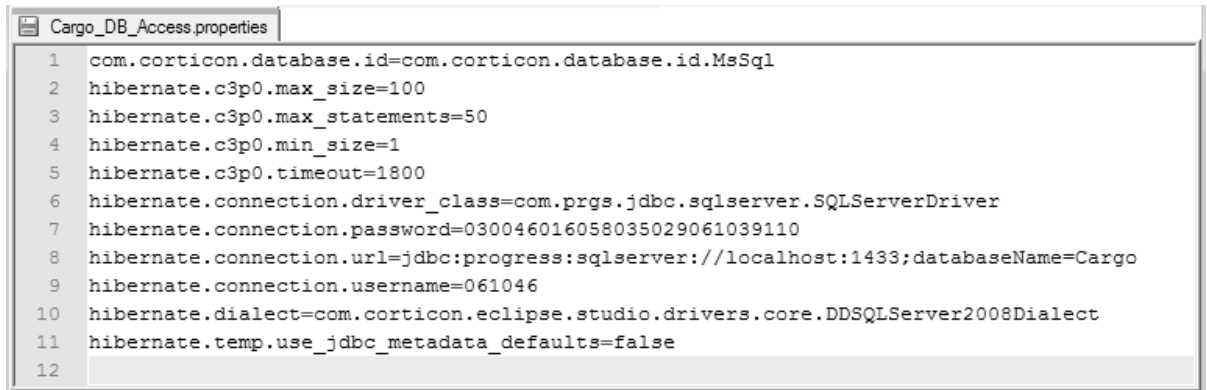
## Creating a database access properties file

---

As a last step in the Studio before we move to the Deployment Console and the Corticon Server, we'll generate the database access properties from our Vocabulary. Take a moment to test the connection to ensure that the properties are correct. Then, all we need to do is choose the Vocabulary and then select **Vocabulary > Database Access > Export Database Access Properties**, as shown:



You can specify a preferred name and location for the file, although colocating it within its related project folder is a good idea. The generated properties file looks like this:

A screenshot of a text editor window titled "Cargo\_DB\_Access.properties". The window contains 12 lines of text, each starting with a line number from 1 to 12. The text represents database connection properties for Hibernate, including driver class, connection URL, username, password, and dialect. The password is a long alphanumeric string, and the URL specifies a local database instance.

```
1 com.corticon.database.id=com.corticon.database.id.MsSql
2 hibernate.c3p0.max_size=100
3 hibernate.c3p0.max_statements=50
4 hibernate.c3p0.min_size=1
5 hibernate.c3p0.timeout=1800
6 hibernate.connection.driver_class=com.prgs.jdbc.sqlserver.SQLServerDriver
7 hibernate.connection.password=030046016058035029061039110
8 hibernate.connection.url=jdbc:progress:sqlserver://localhost:1433;databaseName=Cargo
9 hibernate.connection.username=061046
10 hibernate.dialect=com.corticon.eclipse.studio.drivers.core.DDSQLServer2008Dialect
11 hibernate.temp.use_jdbc_metadata_defaults=false
12
```

Notice that the username and password values are very different from our credentials. These values were encrypted when the database access file was created, and will be decrypted when they are implemented in a decision service.

You shouldn't have to edit this file but if the Server is not colocated with the Studio, you see why it was a good idea to use the explicit host identifier rather than `localhost`.

## Using database access in Corticon Server

---

---

### Note:

This portion of the EDC Tutorial expects that you have completed the *Corticon Server Tutorial: Deploying Web Services*. We will use the same methods for deploying a Decision Service, invoking it with test data, and inspecting the results as we did in that guide. If you have not completed that tutorial, we strongly recommend that you do so before continuing in this EDC deployment tutorial.

---

The tasks in this chapter include:

- A - Registering a license that enables EDC.
- B - Accessing our *Cargo* Ruleflow file to define its database access mode and properties in a Deployment Descriptor file.
- C - Starting Corticon Server and monitoring the deployed Decision Service.
- D - Adjusting the data in the database to recreate our Studio test on a Server.
- E - Creating a Ruletest that uses the remote decision service.
- F - Running the test.
- G - Observing the results.

### A - Installing your EDC-enabled Corticon Server license

Corticon 5.4 servers require a license that enables EDC to perform the tasks in this chapter. If you need to acquire a license, contact Progress Corticon Technical Support or your Progress Software representative.

Once you obtain a license file, you must replace the existing licenses in your Corticon Server installation.

**To install an EDC-enabled license for Corticon Java Server:**

Once you have run Corticon Server, you can update its license locations with your license.

**To configure Corticon Java Server to access your license file:**

1. Stop Corticon Server.
2. Copy your license JAR by its default name, `CcLicense.jar`.
3. Navigate to the installation's `[CORTICON_HOME]\Server\lib` directory to paste the file and overwrite the existing file in that location.
4. Navigate to the installation's `[CORTICON_HOME]\Server\pas\corticon\webapps\axis\WEB-INF\lib` directory to paste the file and overwrite the existing file in that location.

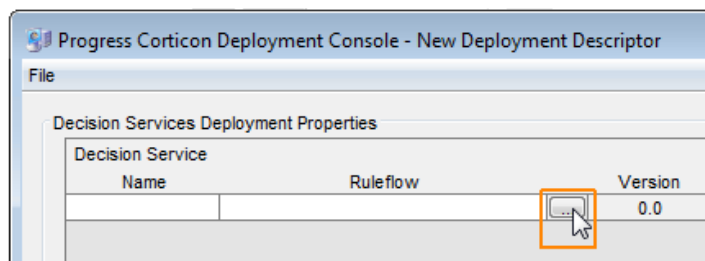
When you restart Corticon Server, your license file will be referenced.

**Note:** For .NET Server, the locations are `[CORTICON_HOME]\Server .NET\samples\lib` and `[CORTICON_HOME]\webservice\lib`

**B - Creating a Deployment Descriptor file from the ruleflow**

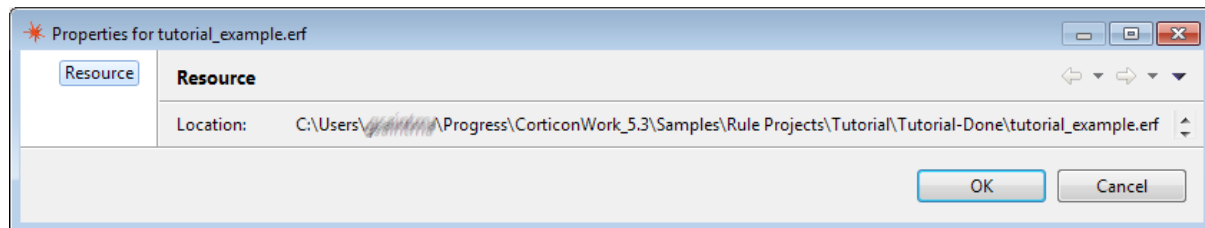
We'll use the Corticon Deployment Console to assemble our ruleflow with database access preferences and properties -- as well as the *location* of its supporting files -- into a Deployment Descriptor ( `.cdd`). These dependent files must be the right ones and retained without change in their specified path. (Compiling these files into a Decision Service ( `.eds`) would create a package of the dependent files that can be relocated for deployment.) , as follows:

1. Launch the Deployment Console by choosing the **Start** menu command **All Programs > Progress > Corticon 5.4 > Corticon Deployment Console**.
2. Click the **Select Ruleflow** button, as shown:



3. Navigate to the Ruleflow file `tutorial_example.erf` in the Cargo project.

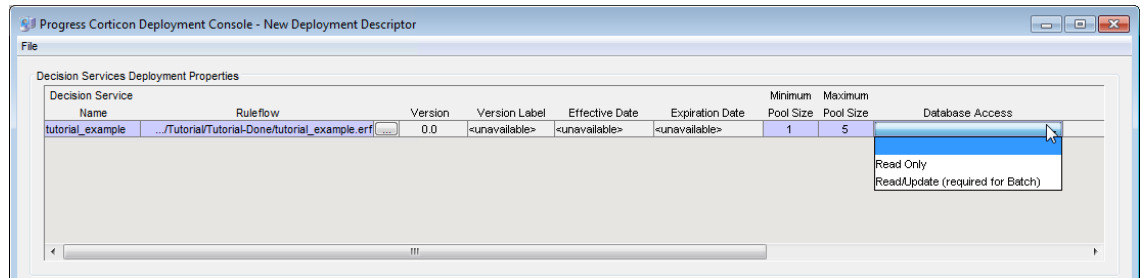
**Note:** If you need to verify that you are getting the intended `.erf` file, click on the Ruleflow file you are working on in the Studio's Rule Project Explorer, and then click **Alt+Enter**. The **Properties** file shows its complete path, as in this example:





- Enter or revise the **Name** on the line, especially if you are redeploying and want to differentiate it from a prior Deployment Descriptor.
- Scroll across the panel, and then click the **Database Access** pulldown for this entry, as shown:

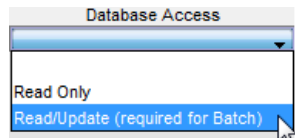
**Figure 45: Deployment Console with Database Access Options**



**Note:** The pulldown is functional only when your Corticon 5.4 license enables EDC.

- Select **Read/Update**.

**Figure 46: Database Access Read/Update Mode Selected**



- Select an option for the **Entities Returned** to specify whether the Corticon Server response message includes *all* data used by the rules including data retrieved from a database (**All Instances**), or *only* data provided in the request and created by the rules themselves (**Incoming/New Instances**). For this example, select **All Instances**.
- Click the **Select Database Access Properties** button, highlighted in the following screen, to locate and select the properties file you created.  
  
The required information for database access is now complete for this Deployment Descriptor file.
- Choose **File > Save**. Save the `tutorial_example.cdd` file in the `[SERVER_WORK_DIR]\cdd` directory as that is where Corticon Server looks for `.cdd` files when it starts up.
- Choose **File > Exit** to close the Corticon Deployment Console.

## C - Starting Corticon Server and the Corticon web service

Choose **Start > All Programs > Progress > Corticon 5.4 > Start Corticon Server..** (During Corticon Server startup, our `.cdd` file will load the `tutorial_example` Decision Service in **Read/Update** database access mode.)

## D - Resetting our sample data in the database

In the Read/Update test, our Ruletest updated the `Container` values in the `Cargo` table:

|   | manifestNumber | container   | needsRefrigeration | volume | weight | FlightPlanAssociation |
|---|----------------|-------------|--------------------|--------|--------|-----------------------|
| ▶ | 625A           | standard    | NULL               | 10     | 1000   | 101                   |
|   | 625B           | oversize    | False              | 40     | 1000   | 102                   |
|   | 625C           | heavyweight | False              | 20     | 30000  | 103                   |
|   | 625D           | reefer      | True               | 10     | 1000   | 103                   |

As we want to perform the identical test in Corticon Server, let's remove the updated data.

In the SQL Server Management Studio, expand the tree to **Databases : Cargo : Tables**, and then right-click on the `Cargo` table to choose **Edit Top 200 Rows** in the tab area.

In the `Container` column of the `Cargo` table, click on a line's container value, and then type `NULL`. (It is important that you type all capital letters.) If you type it correctly, the value displays in italics. Do the same for all the `Container` values, as shown:

|  | manifestNumber | container   | needsRefrigeration | volume | weight | RflightPlanAssociation |
|--|----------------|-------------|--------------------|--------|--------|------------------------|
|  | 625A           | <i>NULL</i> | <i>NULL</i>        | 10     | 1000   | 101                    |
|  | 625B           | <i>NULL</i> | False              | 40     | 1000   | 102                    |
|  | 625C           | <i>NULL</i> | False              | 20     | 30000  | 103                    |
|  | 625D           | <i>NULL</i> | True               | 10     | 1000   | 103                    |

## E - Creating a New Test in Studio

Even though we are using Studio to run the test, we are using Studio's *remote* testing feature, which executes a Decision Service on Corticon Server ("remote"), not a Rulesheet open in Studio ("local"). To keep this distinction clear, we will **not** open `tutorial_example.erf` in Studio – it is not necessary since we are actually testing the Decision Service running on Corticon Server. This means we do not need to worry about the database access settings on the Vocabulary -- they only apply to tests of Rulesheets open in Studio.

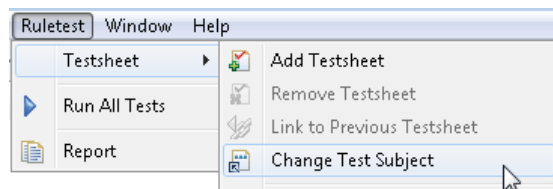
### Note:

When Ruletest Editor operates against a remote decision service, database mode menu items and database mode icon decorations are not displayed. This is because these options have no effect on the remote decision service.

To create the test:

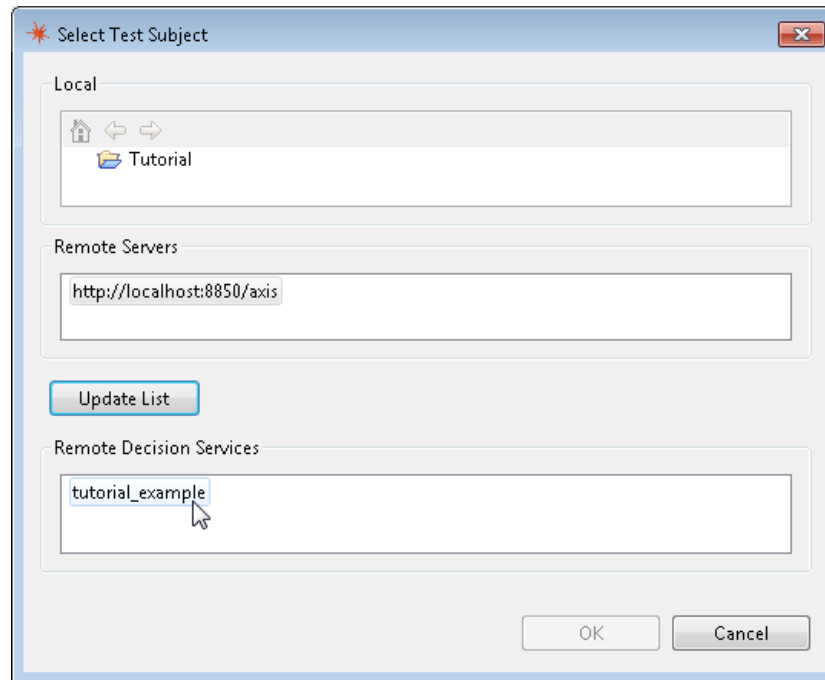
1. The database access settings for the Decision Service are specified in the Deployment Descriptor file (`.cdd`) loaded by the Corticon Server when it started up on Apache so you do not need to touch that in the Vocabulary.
2. In Corticon Studio, open `Cargo.ecore` and then, *without opening any Rulesheets*, choose **New > Ruletest**.
3. Select the menu command **Ruletest > Testsheet > Change Test Subject**, as shown:

**Figure 47: Opening the Select Test Subject dialog**



4. In the dialog box, click on the line item in the **Remote Servers** section, `http://localhost:8850/axis`, and then click **Update List**.
5. Select the name you entered for the Ruleflow line (not the saved file name). Here, our example accepted the default, `tutorial_example`, as shown:

**Figure 48: Selecting a remote decision service**

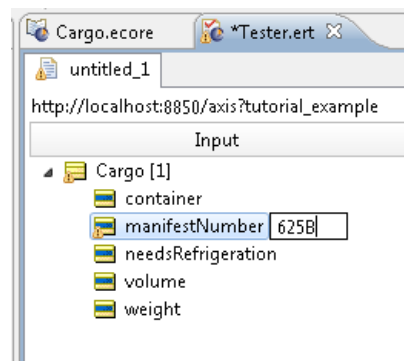


6. Click **OK**.

Now let's set up the test itself.

7. Drag a **Cargo** entity from the Vocabulary, and then enter sample data for `manifestNumber`. Since we are trying to test a remote Decision Service connected to an external database using the Application identity strategy, we need to supply an existing primary key, `manifestNumber`, that is in our tutorial database. Click on `manifestNumber` in the **Input** column to open its input area, and then enter the `625B`.

**Figure 49: Sample Data in a Studio Remote Test**

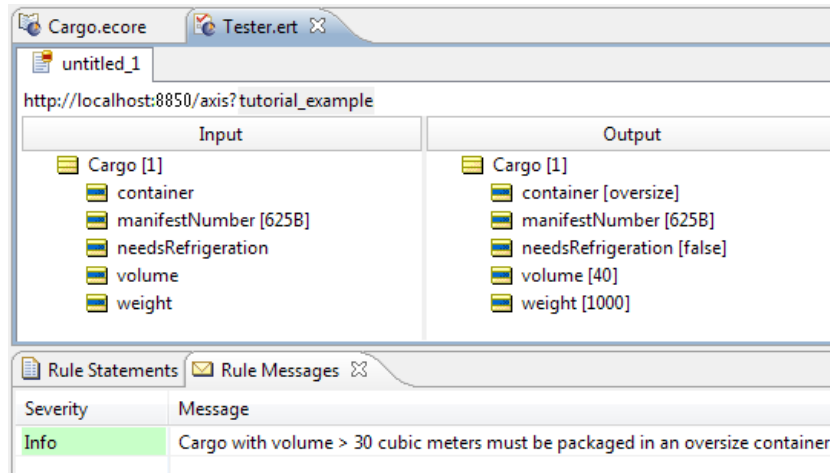


## F - Running the Remote Server Test

Select the menu command **Ruletest > Testsheet > Run Test**.

We see a Results Ruletest similar to this:

Figure 50: Response from Remote Decision Service



**Note:** The `Cargo` database has been accessed, but it was Corticon Server that accessed it and retrieved the data it needed, not Corticon Studio.

## G - Observing the Results in the Database

Return to SQL Server and view the **Content** of the `Cargo` table. Refresh the **Content** if necessary to see the changes. We should see the second row has now been updated to `container`, as shown:

Figure 51: Updated Container data in External Database

|   | manifestNumber | container | needsRefrigeration | volume | weight | RflightPlanAsso.. |
|---|----------------|-----------|--------------------|--------|--------|-------------------|
|   | 625A           | NULL      | NULL               | 10     | 1000   | 101               |
| ▶ | 625B           | oversize  | False              | 40     | 1000   | 102               |
|   | 625C           | NULL      | False              | 20     | 30000  | 103               |
|   | 625D           | NULL      | True               | 10     | 1000   | 103               |

This indicates that Corticon Server successfully accessed the `Cargo` database in **Read/Update** mode during Decision Service execution.

---

## Corticon features and requirements in supported data sources

---

Corticon EDC can connect a Vocabulary to an instance of a supported data source. All data sources support **Import/Clear Database Metadata** and **Validate Mappings**. Some Corticon EDC features are not supported in certain supported data sources. Data manipulations and data source startup functions that might be required to ensure error-free interaction between Corticon EDC and a data source are noted.

The following table summarizes the features:

| Data Source                  | Import Vocabulary | Create Schema | DB Rows: Read/Update | DB Rows: Read-only | Import Metadata | Import Enumerations | Extend to database |
|------------------------------|-------------------|---------------|----------------------|--------------------|-----------------|---------------------|--------------------|
| DataDirect Cloud: Rollbase   | -                 | -             | Yes                  | -                  | Yes             | Yes                 | Yes                |
| DataDirect Cloud: Salesforce | -                 | -             | Yes                  | -                  | Yes             | Yes                 | Yes                |
| IBM DB2                      | -                 | Yes           | Yes                  | Yes                | Yes             | Yes                 | Yes                |
| Microsoft SQL Server         | -                 | Yes           | Yes                  | Yes                | Yes             | Yes                 | Yes                |
| Oracle Database              | -                 | Yes           | Yes                  | Yes                | Yes             | Yes                 | Yes                |
| Progress OpenEdge            | Yes               | Yes           | Yes                  | Yes                | Yes             | Yes                 | Yes                |

---

**Note:** The feature of importing database metadata will infer associations when the information (foreign keys) is available in the data source's metadata.

---

For the current list of supported data sources and versions, access the web location [Progress Corticon 5.4 - Supported Platforms Matrix](#).

For details, see the following topics:

- [DataDirect Cloud: Rollbase](#)
- [DataDirect Cloud: Salesforce](#)
- [IBM DB2](#)
- [Microsoft SQL Server](#)
- [Oracle Database](#)
- [Progress OpenEdge](#)

## DataDirect Cloud: Rollbase

### Vocabulary: Database Connection definition

- **Database URL** - DataDirect Cloud Server. Must include `TransactionMode=ignore`; Default port: 443
- **Username and Password** - DataDirect Cloud credentials. Rollbase credentials are specified in DataDirect Cloud.
- **Catalog filter** - Yes.
- **Schema filter** - Yes.
- **Additional properties** - None required.

### Vocabulary: Database Access actions

- **Create/Update Database Schema** - No.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

### Ruletest: Database Access options

- **Read Only** - No.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

### Other requirements and considerations:

- Many-to-many associations are not supported.

## DataDirect Cloud: Salesforce

### Vocabulary: Database Connection definition

- **Database URL** - DataDirect Cloud Server. Must include `TransactionMode=ignore`; Default port: 443
- **Username and Password** - DataDirect Cloud credentials. Salesforce credentials are specified in DataDirect Cloud.
- **Catalog filter** - Yes.
- **Schema filter** - Yes.
- **Additional properties** - None required.

### Vocabulary: Database Access actions

- **Create/Update Database Schema** - No.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

### Ruletest: Database Access options

- **Read Only** - No.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

### Other requirements and considerations:

- While Corticon rules allow a Boolean to be `true`, `false`, or `null` (the absence of any value), Salesforce only accepts `true` or `false`. You must trap and handle `null` Booleans in Corticon before passing them to Salesforce.

## IBM DB2

### Vocabulary: Database Connection definition

- **Database URL** - Default port: 50000
- **Username and Password** - DB2 credentials.
- **Catalog filter** - Yes.
- **Schema filter** - Yes.
- **Additional properties** - None required.

**Vocabulary: Database Access actions**

- **Create/Update Database Schema** - Yes.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

**Ruletest: Database Access options**

- **Read Only** - Yes.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

**Other requirements and considerations:**

- None.

## Microsoft SQL Server

**Vocabulary: Database Connection definition**

- **Database URL** - Default port: 1433
- **Username and Password** - SQL Server credentials.
- **Catalog filter** - Yes.
- **Schema filter** - Yes.
- **Additional properties** - None required.

**Vocabulary: Database Access actions**

- **Create/Update Database Schema** - Yes.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

**Ruletest: Database Access options**

- **Read Only** - Yes.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

**Other requirements and considerations:**

- None.



# Oracle Database

## Vocabulary: Database Connection definition

- **Database URL** - Default port: 1521
- **Username and Password** - Oracle DB credentials.
- **Catalog filter** - No.
- **Schema filter** - Yes.
- **Additional properties** - None required.

## Vocabulary: Database Access actions

- **Create/Update Database Schema** - Yes.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

## Ruletest: Database Access options

- **Read Only** - Yes.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

## Other requirements and considerations:

- None.

# Progress OpenEdge

## Vocabulary: Database Connection definition

- **Database URL** - Default port: 5566
- **Username and Password** - OpenEdge credentials.
- **Catalog filter** - Yes.
- **Schema filter** - Yes.
- **Additional properties** - None required.

## Vocabulary: Database Access actions

- **Create/Update Database Schema** - Yes. Only for Vocabulary elements that were not created through a BRVD import.
- **Import Enumeration Elements** - Yes.
- **Export Database Access Properties** - Yes.

**Rulesheet: Set an alias to Extend to Database** - Yes.

**Ruletest: Database Access options**

- **Read Only** - Yes.
- **Read/Update** - Yes.
- **Return All Entity Instances** - Yes.
- **Return Incoming/New Entity Instances Only** - Yes.

**Other requirements and considerations:**

- **Importing a file to create a Vocabulary** - Import requires a `.brvd` file created in OpenEdge (see Progress OpenEdge documentation for details.) The import function into Corticon is described in *"Importing an OpenEdge Business Rules Vocabulary Definition (BRVD) file" in the Rule Modeling Guide*.
- **Startup of OE server** - It is recommended that you start the OpenEdge database server with the following parameters within the `Proenv` window, shown here with values used in a test environment:

```
proserve db_name -n 65 -Mn 20 -Mpb 4 -Ma 20 -Mi 3 -S port_number
```

where:

- `db_name` is the database name
- `port_number` is the port number
- Other OpenEdge parameters as described in [OpenEdge Database Server parameters](#).

---

## Continuing to learn about Corticon's Enterprise Data Connector

---

Congratulations! You have completed the tutorial on the Enterprise Data Connector.

To continue learning about EDC, see these documentation locations:

- *Writing Rules to access external data* chapter in the *Rule Modeling Guide* extends the tutorial into scope, validation, collections, and filters.
- *Relational database concepts in the Enterprise Data Connector (EDC)* in the *Integration and Deployment Guide* discusses identity strategies, key assignments, catalogs and schemas, database views, table names and dependencies, inferred values, and join expressions.
- *Implementing EDC* in the *Integration and Deployment Guide* discusses the mappings and validations in a Corticon connection to an RDBMS.
- *Deploying Corticon Ruleflows* in the *Integration and Deployment Guide* describes the Deployment Console parameters for Deployment Descriptors and compiled Decision services that use EDC.
- *Vocabularies: Populating a New Vocabulary: Adding nodes to the Vocabulary tree view* in the *Quick Reference Guide* extends its subtopics to detail all the available fields for Entities, Attributes, and Associations.

