

Modeling Data Relationships

MoviesDB Diagram

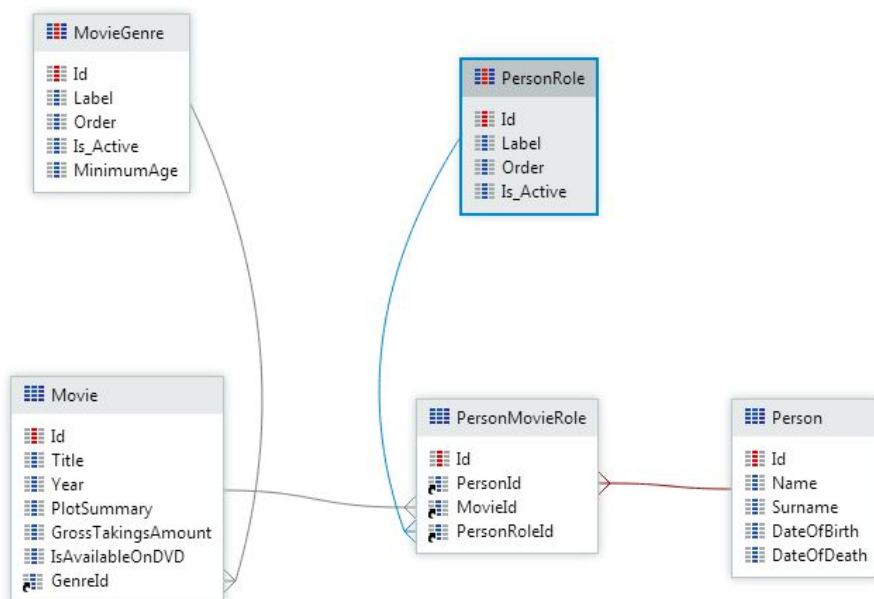


Table of Contents

Table of Contents	2
Introduction	3
Create a One to Many Relationship between Entities	4
Create Many-to-Many Relationships between Entities	8
Publish the OSMDb application's modules	17
End of Lab	21

Introduction

In this exercise lab, we will extend the current data model of the application, to add relationships between the Entities.

Every movie in the database should have a genre. For that reason, we will add a new attribute to the Movie Entity, **Genre Id**, to add a one-to-many relationship between the movie and the MovieGenre Entity.

We will also create three new Entities, **PersonMovieRole**, **UserMovieComment** and **UserMovieRating**, to define many-to-many relationships between the existing Entities. PersonMovieRole will represent the Role of a Person in a Movie. The UserMovieComment Entity will save a comment that a User made to a movie, while the UserMoveRating Entity will be similar, but for ratings. For that, we will need to use the System User Entity, where all registered end-users will be saved.

During this process, we will also create some Unique Indexes in the Entities, to avoid having repeated information in the database.

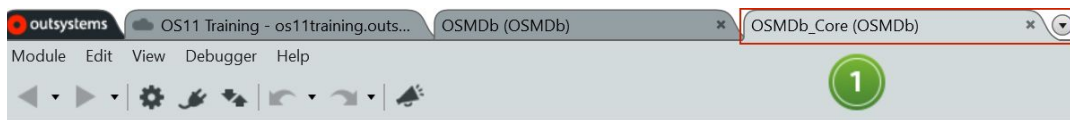
In this specific exercise lab, you will:

- Create dependencies between Entities
- Create a simple relationship between two Entities
- Create many-to-many relationships between two Entities
- Create Unique Indexes
- Create an Entity Diagram
- Publish the module in the OutSystems web server

Create a One to Many Relationship between Entities

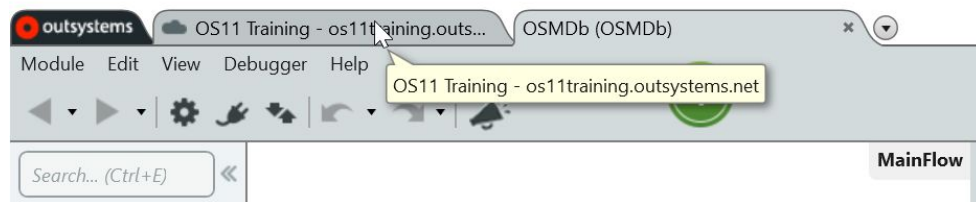
In the previous lab, we created Screens in the OSMDb application to present the data to the users. In this part of the exercise, we will work again in the Core module, where we will create the first relationship between two Entities. Each movie is of a certain genre (comedy, etc.), so the **Movie** Entity will need to reference a **MovieGenre** Static Entity record. To accomplish that we need to create a new attribute.

- 1) Switch to the **OSMDb_Core** module in Service Studio if the module is already opened. Otherwise move on to the second step.



- 2) Open the **OSMDb_Core** module in Service Studio.

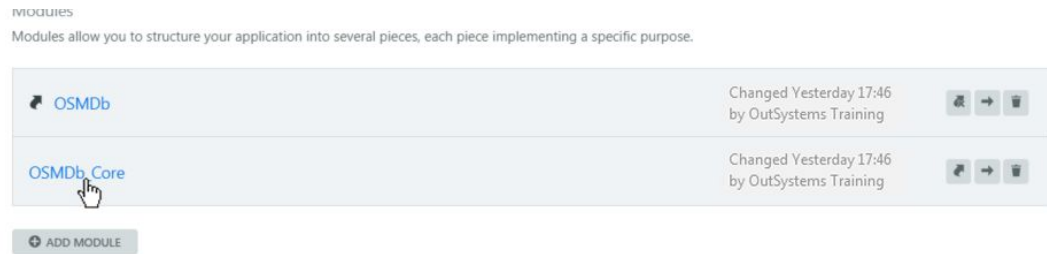
- a) Go back to the Application tab.



- a) Make sure the Application tab has the **OSMDb** application opened and if not open it from the application list.

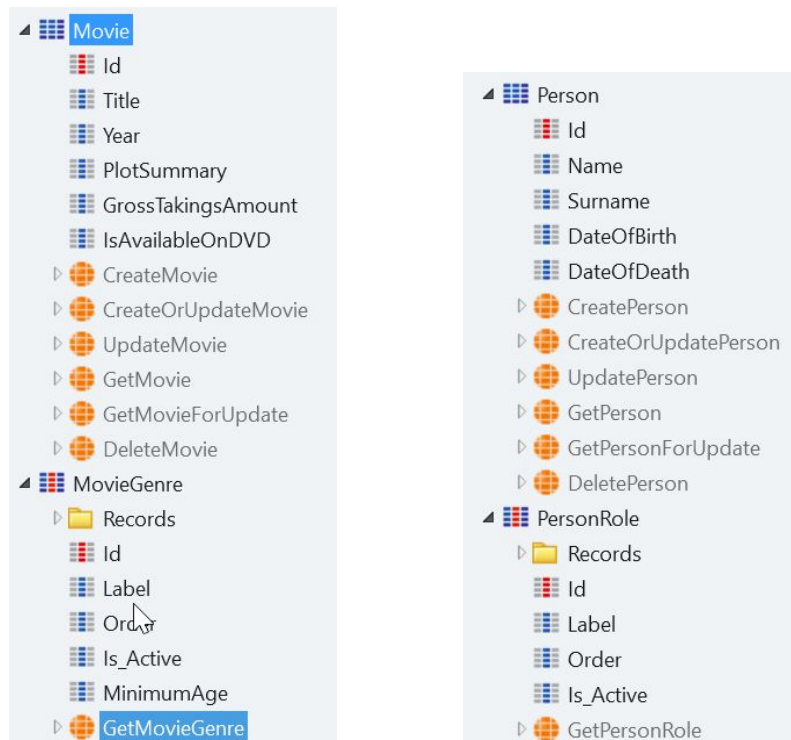


- b) In the MODULES area, click the **OSMDb_Core** module.



3) Let's explore and understand better how OutSystems Entities work.

- On the **OSMDB_Core** module, switch to the **Data** tab.
- Expand the Movie and Person Entities as well as the MovieGenre and PersonRole Static Entities (if they are currently collapsed). Notice that all of the Entities have a default attribute called **Id** that is colored red. This is what is called the **Entity's Identifier**.



NOTE: An identifier of an Entity uniquely identifies each record of an Entity, or Static Entity. This means that the value of the Id attribute for each record in the database will be different. This uniqueness is enforced by the database itself.

- c) Click the **Movie Id** attribute and notice its **Data Type** is *Long Integer*.

Id Entity Attribute	
Name	Id
Description	...
Label	Id
Data Type	Long Integer ▼
Is AutoNumber	Yes ▼
Is Mandatory	Yes
Default Value	

NOTE: The <Entity> Identifier's Data Type encapsulates the underlying type of the Entity Identifier. These can be Integer, Long Integer, Text or other Entities' Identifiers. In this case, they take on the underlying type of the target Entity's identifier. This enforces a dependency between records of different Entities.

- d) Notice that the **Is AutoNumber** property is *Yes*.

NOTE: When **Is AutoNumber** is set to *Yes*, the database automatically increments the identifier number, as new records are inserted into the database, facilitating development. Only Integer and Long Integer-typed identifiers can be Auto Number.

- 4) Add a new attribute with the name *GenreId* to the **Movie** Entity. Set its type to **MovieGenre Identifier**.

- Right-click the **Movie** Entity and select **Add Entity Attribute**.
- Enter *GenreId* for the name of the attribute.
- Change the **Data Type** of **GenreId** to *MovieGenre Identifier*.

GenreId Entity Attribute	
Name	GenreId
Description	...
Label	Genre
Data Type	MovieGenre Identifier ▼
Is Mandatory	
Delete Rule	

- Activity Identifier
- Espace Identifier
- Group Identifier
- MovieGenre Identifier
- Movie Identifier
- Person Identifier

NOTE: This pattern creates one-to-many relationship, where a movie has one genre, but a genre can be assigned to many movies. This GenreId attribute is a foreign key of the Movie Entity, referencing the respective GenreId from the MovieGenre Entity.

Create Many-to-Many Relationships between Entities

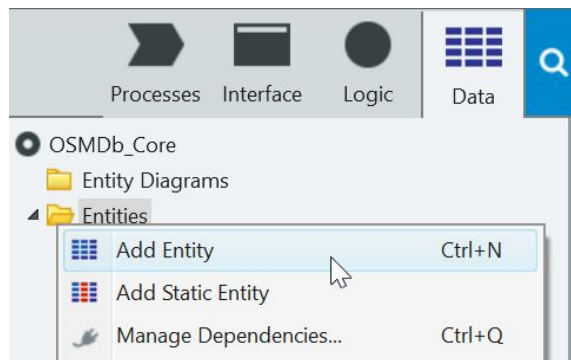
A many-to-many relationship happens when one Entity has an one-to-many relationship with another Entity, and vice-versa. For example, a **Movie** has many people involved, and a **Person** may participate in different Movies.

In OutSystems, these many-to-many relationships can be created by adding a third Entity, which we call the junction Entity. In the OSMDb application, we need some junction Entities to model some of the application concepts.

First, we need to create a many-to-many relationship between the Person and Movie Entities. Then, we need two many-to-many relationships between the System Entity User and the Movie Entity. One for movie comments and other for movie ratings, since a user can submit multiple ratings and comments, and movies can also be commented or rated by multiple users.

- 1) Define a many-to-many relationship between the Person and Movie Entities, by creating the *PersonMovieRole* Entity. Add a *PersonId* (Mandatory, Delete Rule: Delete), *MovieId* (Mandatory, Delete Rule: Protect) and *PersonRoleId* (Mandatory, Delete Rule: Protect) attribute of the appropriate types to the Entity, and finally set it as **Public** and **with write permissions**.

- a) In the Data tab, right click the **Entities** folder and select **Add Entity**.



- b) Enter *PersonMovieRole* for the name of the Entity.
- c) Right-click the **PersonMovieRole** Entity and select *Add Entity Attribute*. **Name** the attribute *PersonId*. Set its **Mandatory** property to Yes and the **Delete Rule** to *Delete*.
- d) Right-click the **PersonMovieRole** Entity and *Add Entity Attribute*. **Name** the attribute *MovieId*. Set its **Mandatory** property to Yes. Make sure the **Delete Rule** is set to *Protect*.

- e) Add a new attribute to the **PersonMovieRole** Entity and name it *PersonRoleId*. Set its **Mandatory** property to *Yes*. Make sure the Delete Rule is set to *Protect*.

NOTE: This PersonRoleId attribute is not part of the **Many-to-Many** relationship implementation. It serves to qualify in which capacity (role) the referenced Person participated in the referenced Movie (director, actor, etc.).

- f) Verify that the types for these attributes were inferred correctly:

PersonId: *Person Identifier*

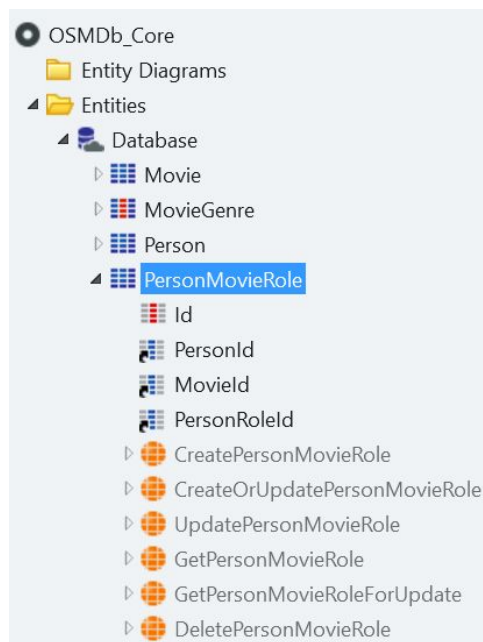
MovieId: *Movie Identifier*


PersonRoleId: *PersonRole Identifier*

- g) Since we will be using the **PersonMovieRole** Entity on our UI module, change the **Public** property to *Yes* and the **Expose Read Only** to *No*.

PersonMovieRole Entity	
Name	PersonMovieRole
Description	...
Public	Yes ▼
Expose Read Only	No ▼
Indexes	...
More...	...

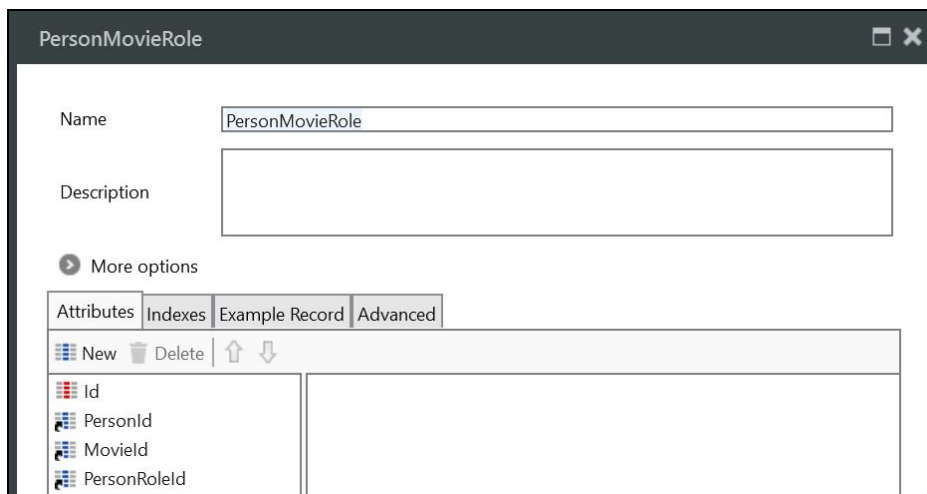
- h) The **PersonMovieRole** Entity should look like this



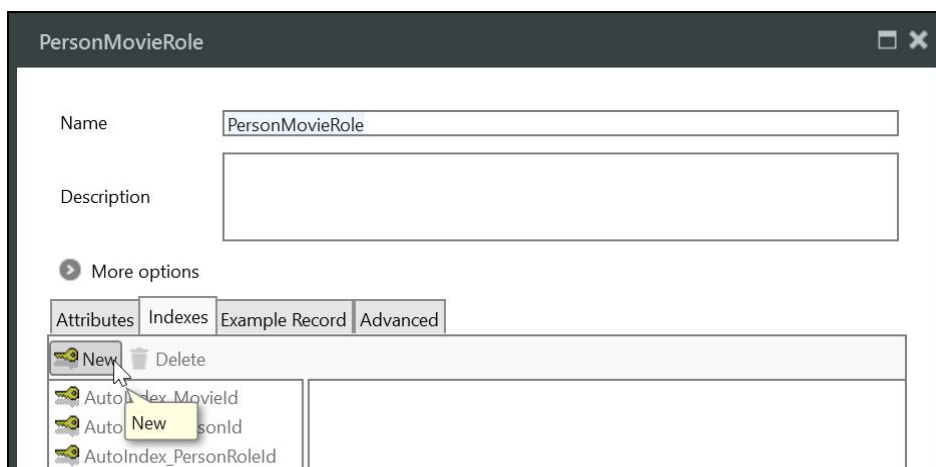
NOTE: Every attribute that references another Entity (i.e. is of the data type *<Entity Name> Identifier*) has an icon with a shortcut overlay 

- 2) The PersonMovieRole Entity should not have two different records with the same values in its attributes (PersonId, MovieId, PersonRoleId), otherwise we would be storing redundant information. Uniqueness of Entity attributes across multiple rows can be achieved by defining indexes over the Entities. So let's create an **Unique Index** over the attributes (PersonId, MovieId, PersonRoleId), in the **PersonMovieRole** Entity, so that a person cannot have the same role in the same movie.

- a) Double click on the **PersonMovieRole** Entity to open the Entity's editor window.



- b) Open the **Indexes** tab and click on the **New** option to create a new Index.



- c) Type *PersonWithUniqueRolePerMovie* for the **Name** of the Index.
d) Set the **Unique** property to Yes.

Attributes	Indexes	Example Record	Advanced
<div> New Delete </div>			
<div> AutoIndex_MovieId AutoIndex_PersonId AutoIndex_PersonRoleId PersonWithUniqueRolePerMovie </div>	<div> <div>Name</div> <div>PersonWithUniqueRolePerMovie</div> </div> <div> <div>Description</div> <div>...</div> </div> <div> <div>Unique</div> <div>Yes</div> </div> <div> <div>Attributes</div> <div>(Add Attribute)</div> </div>		

e) In the **(Add Attribute)** dropdown, select **PersonId**.

Attributes	Indexes	Example Record	Advanced
<div> New Delete </div>			
<div> AutoIndex_MovieId AutoIndex_PersonId AutoIndex_PersonRoleId PersonWithUniqueRolePerMovie </div>	<div> <div>Name</div> <div>PersonWithUniqueRolePerMovie</div> </div> <div> <div>Description</div> <div>...</div> </div> <div> <div>Unique</div> <div>Yes</div> </div> <div> <div>Attributes</div> <div>(Add Attribute)</div> </div>	<div> <div>Id</div> <div>PersonId</div> <div>MovieId</div> <div>PersonRoleId</div> </div>	

f) Repeat the previous step in order to add the **MovieId** and **PersonRoleId** attributes to the **PersonWithUniqueRolePerMovie** index. At the end, the Index should look like this:

Attributes	Indexes	Example Record	Advanced
<div> New Delete </div>			
<div> AutoIndex_MovieId AutoIndex_PersonId AutoIndex_PersonRoleId PersonWithUniqueRolePerMovie </div>	<div> <div>Name</div> <div>PersonWithUniqueRolePerMovie</div> </div> <div> <div>Description</div> <div>...</div> </div> <div> <div>Unique</div> <div>Yes</div> </div> <div> <div>Attributes</div> <div>Attribute 1</div> <div>Attribute 2</div> <div>Attribute 3</div> </div>	<div> <div>PersonId</div> <div>MovieId</div> <div>PersonRoleId</div> </div>	

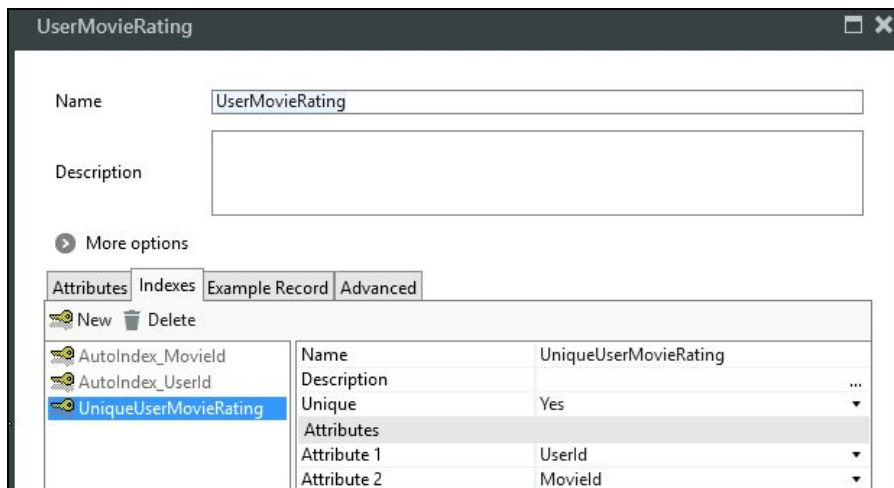
NOTE: With a Unique Index, by inserting a second record with repeated values, the operation will fail and it will cause an error.

- g) Click the **Close** button to dismiss the PersonMovieRole Entity Editor window.
- 3) We also need a many-to-many relationship between the **System User Entity** and the Movie Entity. A **User** of the OSMDb application will be able to rate movies, while a movie could be rated by several users. Therefore, we will create the *UserMovieRating* Entity, with the *UserId* (non mandatory) and *MovieId* (mandatory) attributes, and an Integer *Rating* attribute (mandatory). Set the Entity to **Public** and **Exposed with write permissions**.

- a) Right-click the **Entities** folder and select **Add Entity**. Enter *UserMovieRating* for the name of the Entity.
- b) Add a new *UserId* attribute to the **UserMovieRating** Entity. Leave its **Mandatory** property as *No*. Verify if the Data Type was set to *User Identifier*.
- c) Add a new *Movied* attribute to the **UserMovieRating** Entity. Set its **Mandatory** property as *Yes*. Verify if the Data Type was set to *Movie Identifier*.

NOTE: The **User** Entity is a global system Entity, referenced by default by all application modules. This Entity contains information regarding all the users registered to use the OutSystems applications, namely their names, username and password. It is located under the **(System)** module. At this point, we don't have any users registered, which will be done later in this sequence of Labs.

- d) Add a new *Rating* attribute to the **UserMovieRating** Entity. Set its **Mandatory** property to *Yes* and set its **Data Type** to *Integer*.
- 4) Create the *UniqueUserMovieRating* **Unique Index** over the attributes (*UserId*, *Movied*) in the **UserMovieRating** Entity, so that a user can only rate a given movie once.
- a) Select the **UserMovieRating** Entity to access its Properties editor and double-click the **More...** property to launch the Entity Editor.
 - b) Create a new Index *UniqueUserMovieRating* and set its **Unique** property to *Yes*.
 - c) Add the attributes **UserId** and **Movied** to the Index.
 - d) The *UniqueUserMovieRating* Index should look like this

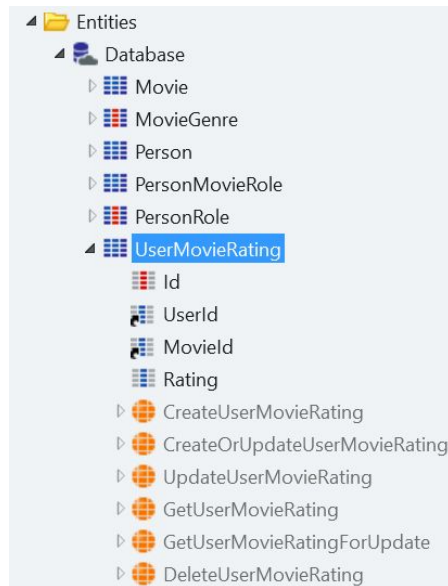


The screenshot shows the 'UserMovieRating' Entity Editor window. The 'Indexes' tab is selected, displaying a table of indexes. The 'UniqueUserMovieRating' index is highlighted. The table shows the index name, description, uniqueness, and the attributes it covers.

Name	Description	Unique	Attributes
UniqueUserMovieRating		Yes	UserId, Movied

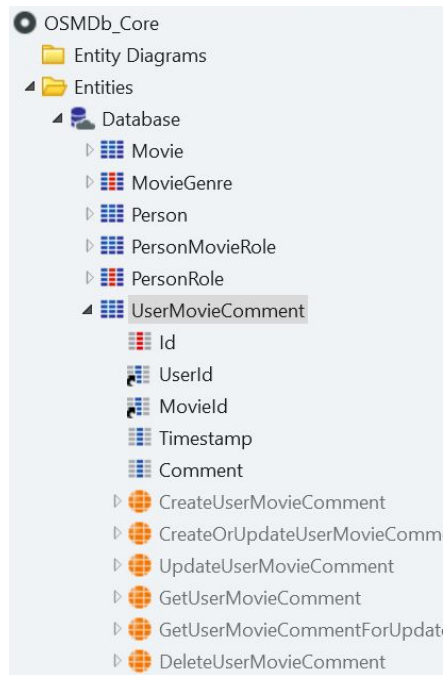
- e) Click the **Close** button to dismiss the UserMovieRating Entity Editor window.

- f) Since we will be using this Entity on our UI module, in the properties editor, change the **Public** property to *Yes* and the **Expose Read Only** to *No*.
- g) Your Entity should look like this:



- 5) The last many-to-many relationship that we need to create is between the User and Movie Entities. A registered User of the OSMDb application will be able to comment movies and a single movie can be commented by several users. Therefore, let's create the **UserMovieComment** Entity, with the **UserId** (non mandatory) and **MovieId** (mandatory) attributes. The Entity will also need a DateTime attribute, Timestamp, and a Text attribute, Comment, which are not mandatory. Set the Entity to **Public** and **Exposed with write permissions**.
 - a) Add a new *UserMovieComment* Entity.
 - b) Add a new *UserId* attribute to the **UserMovieComment** Entity. Set its **Mandatory** property as *No*.
 - c) Add a new *MovieId* attribute to the **UserMovieComment** Entity. Set its **Mandatory** property as *Yes*.
 - d) Add a new *Timestamp* attribute to the **UserMovieComment** Entity. Set its **Data Type** to *Date Time*.
 - e) Finally, add a last attribute to the Entity and name it as *Comment*. Ensure its **Data Type** is *Text*. Set its **Length** to 300.
 - f) Set the **Public** property of **UserMovieComment** to *Yes* and the **Expose Read Only** to *No*.

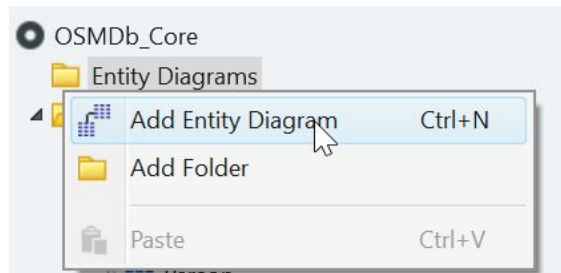
g) The UserMovieComment Entity should look like this



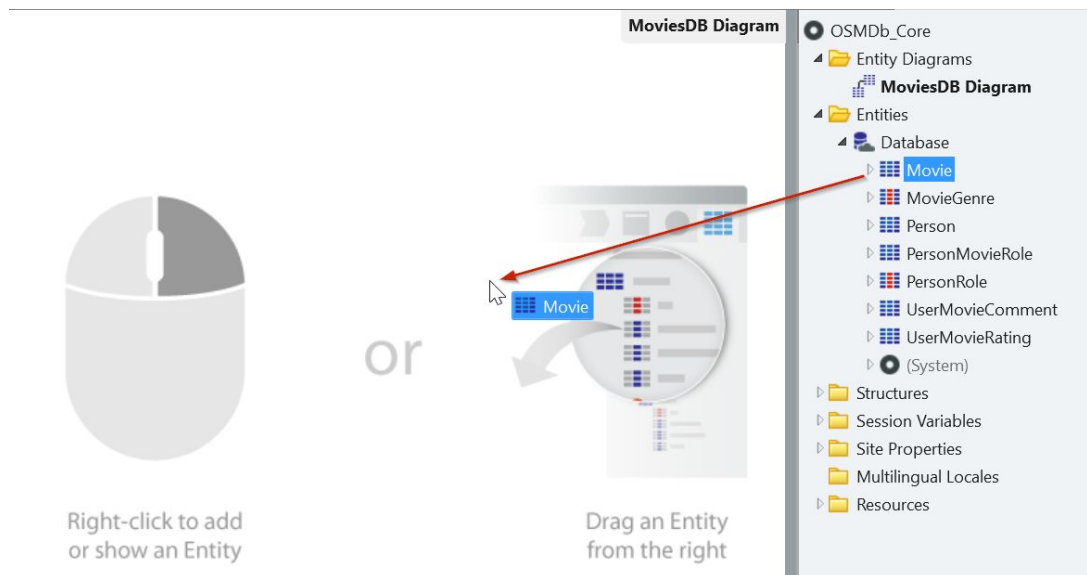
Create an Entity Diagram

In this part of the exercise, we will create an Entity Diagram. This diagram allows us to quickly see the Entities and the relationships between them.

- 1) Create an Entity Diagram. Add the **Movie** and **MovieGenre** Entities to it.
 - a) In the elements area of the **Data** tab, right click the **Entity Diagrams** folder and select *Add Entity Diagram*.

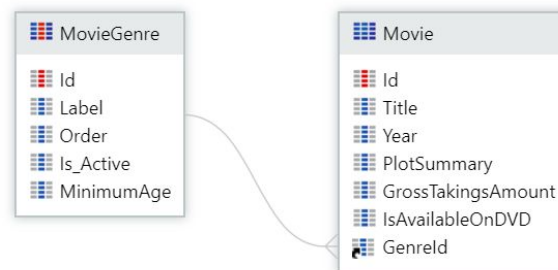


- b) Name this diagram *MoviesDB Diagram*.
 - c) Drag and drop the **Movie** Entity into the *MoviesDB Diagram* canvas.



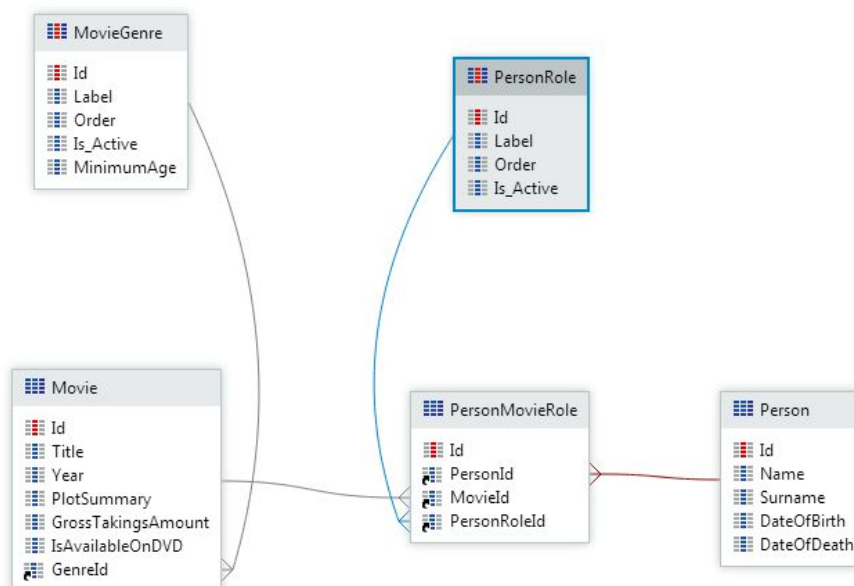
- d) Drag and drop the **MovieGenre** Static Entity above the **Movie** Entity in the diagram. Notice the connector that links the **GenreId** attribute to the **MovieGenre** Static Entity highlighting the one-to-many relationship.

MoviesDB Diagram



- 2) Add the **Person** Entity and **PersonMovieRole** and **PersonRole** Static Entities to the Entity Diagram. The Diagram should look like this:

MoviesDB Diagram



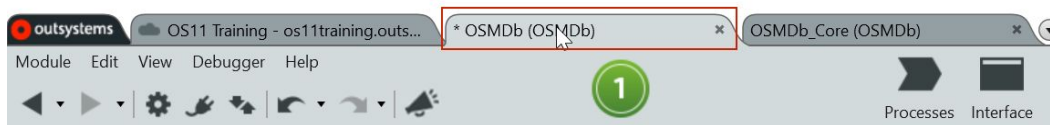
NOTE: The Entity Diagrams are simply a design-time visualization aid. You can include as many or as few Entities (and Static Entities) in an Entity Diagram, to improve the understandability of the data model by the developers. This will not influence the code produced on publication in any form.

Publish the OSMDb application's modules

With all the new Entities created, it's time to publish the OSMDb_Core module to the server. This will create a new version of the module, which is a Producer of the OSMDb module.

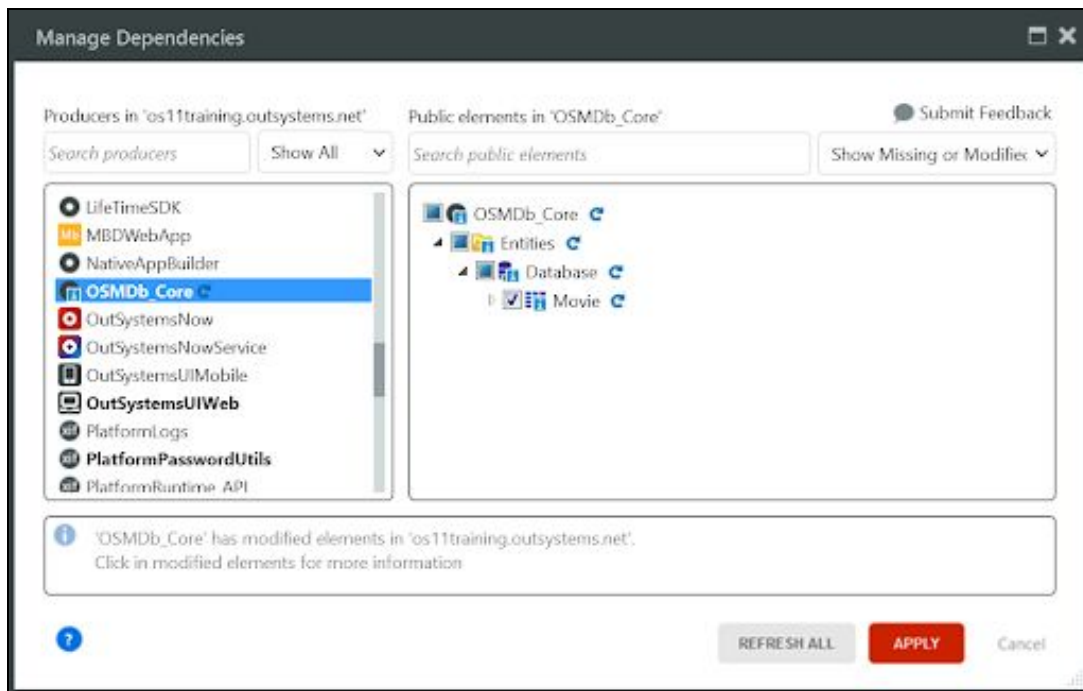
After publishing the module, go back to the **OSMDb** module, open the **Manage Dependencies** dialog, refresh the existing OSMDb_Core references, to include the new GenreId attribute of the Movie Entity, and add the recently created **PersonMovieRole** and **UserMovieRating** Entities.

- 1) Click the **1-Click Publish** button to publish the **OSMDb_Core** module. Verify that it publishes successfully. Ignore the Potential Incompatible Consumer warning message.
- 2) Switch to the **OSMDb** module in Service Studio for editing.



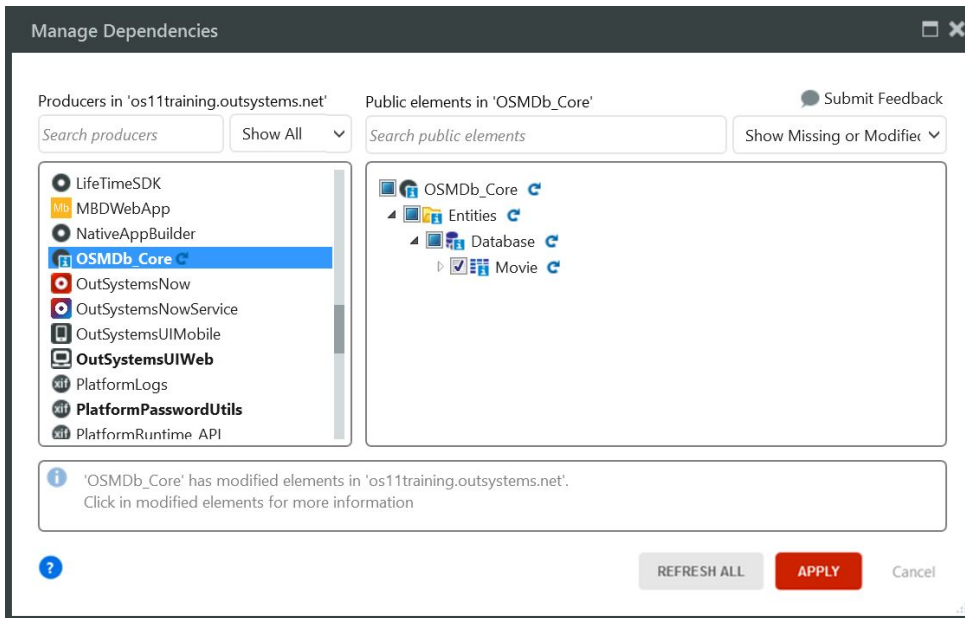
- 3) Open the **Manage Dependencies** dialog by pressing the icon in the toolbar.
- 4) In the **Manage Dependencies** dialog, select the **OSMDb_Core** module from the list of Producers. Notice the blue circular arrow on the module and some of the elements. This means that this module has elements that were modified. However, the consumer module (OSMDb) is still using the old version of the element. Clicking on that icon in that element will refresh the reference to the element, making the OSMDb (producer) use the

new version of the element that was published in the OSMDb_Core (consumer).

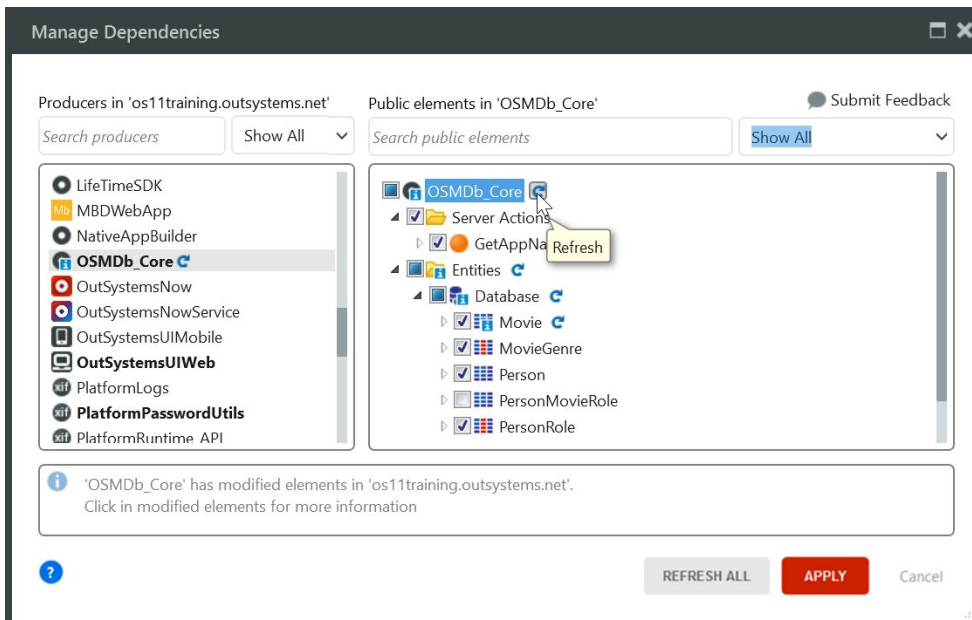


NOTE: The **Refresh All** option automatically refreshes all the references that are not up to date. Clicking on the blue icon at the level of the module will refresh all the outdated references in that specific producer module. Clicking OK without refreshing **does not** update the references.

- 5) In the **Public elements** list for the Core module (on the right side of the **Manage Dependencies** dialog), select the **Show All** filter.

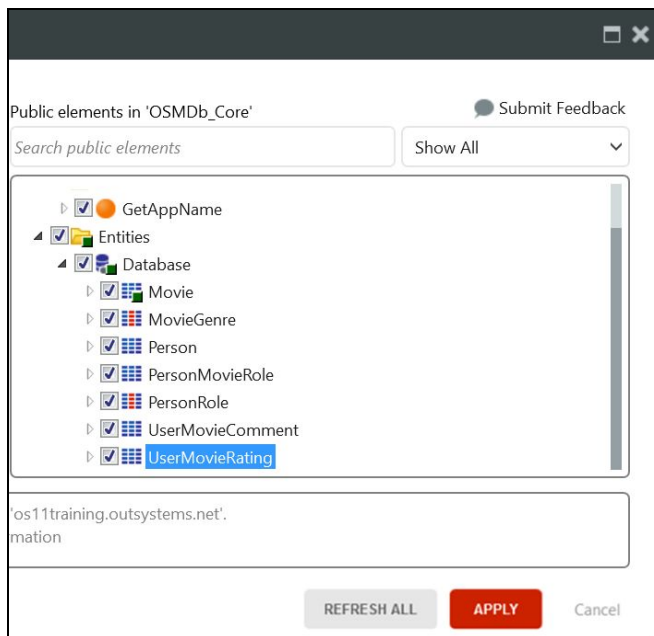


- 6) Click the refresh icon next to **OSMDB_Core** to refresh the references.



- 7) So far, only the used elements were refreshed. The new Entities added to the OSMDB_Core module are still not referenced. Select all the new Entities by checking

them in the **Entities** folder, and press **Ok** to close the dialog.



All elements that are not referenced yet are not affected by the Refresh operation. We need to explicitly add the reference.

- 8) Publish the OSMDb module using the **1-Click Publish** button, so the server commits these dependencies between modules.

End of Lab

In this exercise, you extended the initial data model for the OSMDb web application by creating relationships between them.

A movie can now have a movie genre associated, with a simple one-to-many relationship between the Movie and MovieGenre Entities.

The data model also allows the association of a Person to a Movie, through its Role in the Movie, with the new junction Entity PersonMovieRole. Additionally, we added two more Entities to represent the ratings and comments a user can make to a given movie. All these three Entities define many-to-many relationships between the existing Entities.

Also, to avoid having a Person with the same Role in the same Movie twice, we added a Unique Index on the PersonMovieRole Entity, which guarantees that two records cannot exist in the Entity, with the same Person Id, PersonRole Id and Movie Id. The same logic was applied to the UserMovieRating, since a User cannot rate the same movie twice.

Finally, we created the Entity Diagram where all the Entities and their relationships are visually represented in Service Studio.