

6

Mapping Object Model to Data Model and vice versa

Chapter 6 - Mapping Object Model to Data Model and vice versa

Being an Object-Relational Mapping tool, DB Visual ARCHITECT (DB-VA) automates the mapping between object and data models. This chapter describes how DB-VA maps the object model to data model and vice versa, and introduces the usage of ORM diagram.

In this chapter:

- Introduction
- Mapping Object Model to Data Model
- Mapping Data Model to Object Model
- Showing Mapping by ORM Diagram

Introduction

DB Visual ARCHITECT (DB-VA) supports Object Relational Mapping (ORM) which maps object models to entity relational models and vice versa.

DB-VA helps mapping between Java objects to relational database. It not only preserves the data, but also the state, foreign/primary key mapping, difference in data type and business logic. Thus, you are not required to handle those tedious tasks during software development.

Mapping Object Model to Data Model

This section shows you how DB-VA maps object models to data model.

Mapping Classes to Entities

Object Model can map to Data Model due to the persistent nature of classes. Persistent classes can act as persistent data storage during the application is running. Hence, all persistent classes can map to entities using a one-to-one mapping.

Example:

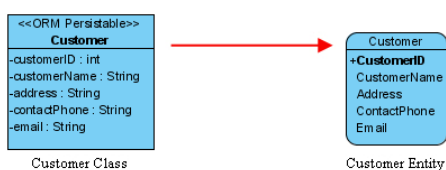


Figure 6.1 - Mapping Classes to Entities

In the above example, the Customer Class is mapped with the Customer Entity as the Customer instance can store the customer information from the Customer Entity.

Mapping Attributes to Columns

Since the persistent classes map to the entities, persistent attributes map to columns accordingly. DB-VA ignores all non-persistent attributes such as derived values.

Example:

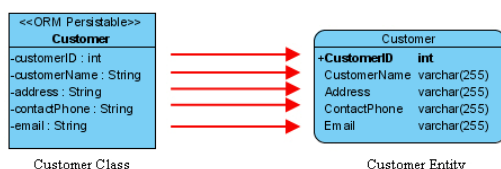


Figure 6.2 - Mapping Attributes to Columns

In the above example, the following table shows the mapping between the attributes of the Customer Class and the columns of the Customer Entity.

Customer Class	Customer Entity
CustomerID	CustomerID
CustomerName	CustomerName
Address	Address
ContactPhone	ContactPhone
Email	Email

Table 6.1

Mapping Data Type

DB-VA automatically maps the persistent attribute type to an appropriate column data type of the database you desired.

Example:

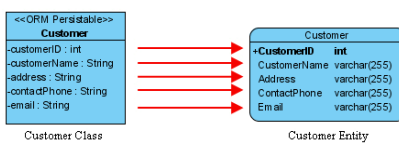


Figure 6.3 - Mapping Data Type

In the above example, the following table shows the mapping between data types

Customer Class	Customer Entity
int	int (10)
String	varchar(255)

Table 6.2

A table shows the data type mapping between object model and data model.

Object Model	Data Model
Boolean	Bit (1)
Byte	Tinyint (3)
Byte[]	Binary(1000)
Blob	Blob
Char	Char(1)
Character	Char(1)
String	varchar(255)
Int	Integer(10)
Integer	Integer(10)
Double	Double(10)
Decimal	Integer
Bigdecimal	Decimal(19)
Float	Float(10)
Long	Bigint(19)
Short	Smallint(5)
Date	Date
Time	Time(7)
Timestamp	Timestamp(7)

Table 6.3

Mapping Primary Key

You can map an attribute to a primary key column. When you synchronize the ORM-Persistable Class to the ERD, you will be prompted by a dialog box to select primary key.

- You can select an attribute as the primary key.
- You can let DB-VA generate the primary key automatically.

Example:

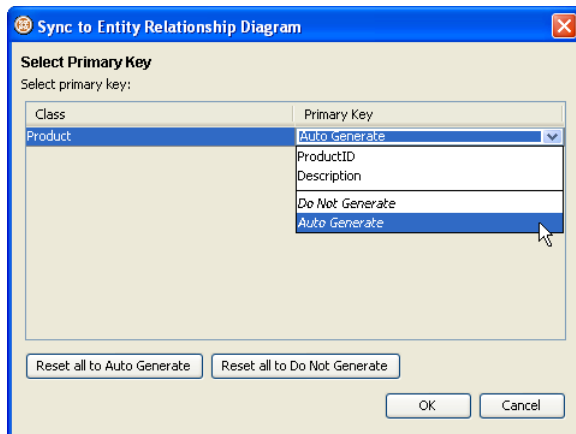


Figure 6.4 - Sync to Entity Relationship Diagram dialog

In the above example, when synchronizing the class of Product to entity relationship diagram, the above dialog box is shown to prompt you to select the primary key of the Product class.

Under the drop-down menu, you can select either one of the attributes of the Product class to be the primary key, or assign DB-VA to generate the primary key automatically, or select "**Do Not Generate**" to leave the generated entity without primary key.

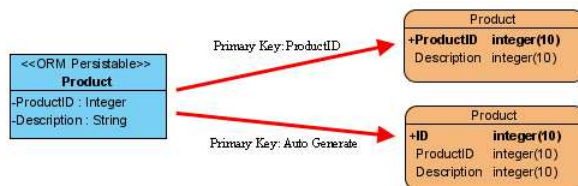


Figure 6.5 - Different between specify a primary key and auto generated

The above diagram shows if you assign ProductID as primary key, the ProductID of the generated entity, Product will become bold; whereas if you select "**Auto Generate**" for the primary key, DB-VA generates an additional attribute ID as the primary key of the Product entity.

Mapping Association

Association represents a binary relationship among classes. Each class of an association has a role. A role name is attached at the end of an association line. DB-VA maps the role name to a phrase of relationship in the data model.

Mapping Aggregation

Aggregation is a stronger form of association. It represents the "has-a" or "part-of" relationship.

Example:

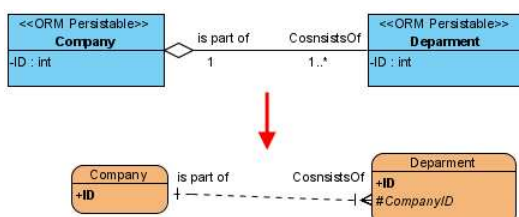


Figure 6.6 - Mapping Aggregation

In the above example, it shows that a company consists of one or more department while a department is a part of the company.

Note

You have to give the role names, "ConsistsOf" and "is Part Of" to the classes, Company and Department in the association respectively in order to proceed to the generation of code.

Mapping Composite Aggregation

Composite aggregation implies exclusive ownership of the "part-of" classes by the "whole" class. It means that parts may be created after a composite is created, meanwhile such parts will be explicitly removed before the destruction of the composite.

Example:

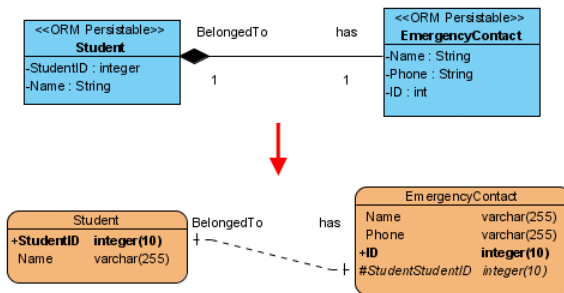


Figure 6.7 - Mapping Composite Aggregation

In the above example, DB-VA performs the Primary/Foreign Key Column Mapping automatically. StudentID of the student entity is added to the entity, EmergencyContact as primary and foreign key column.

Mapping Multiplicity

Multiplicity refers to the number of objects associated with a given object. There are six types of multiplicity commonly found in the association. The following table shows the syntax to express the Multiplicity.

Table shows the syntax expressing the Multiplicity

Type of Multiplicity	Description
0	Zero instance
0..1	Zero or one instances
0..*	Zero or more instances
1	Exactly one instance
1..*	One or more instances
*	Unlimited number of instances

Table 6.4

Example:

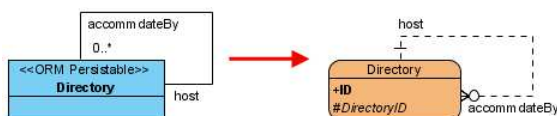


Figure 6.8 - Mapping Multiplicity

In the above example, it shows that a parent directory (role: host) contains zero or more subdirectories (role: accommodated by).

When DB-VA transforms a class with multiplicity of zero, the foreign key of parent entity can be nullable in the child entity. It is illustrated by the DirectoryID.

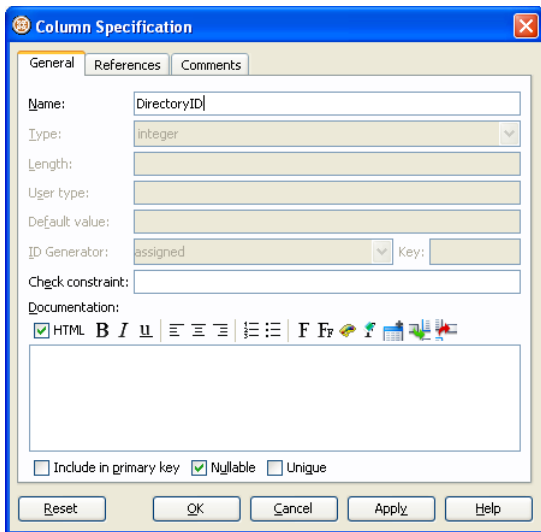


Figure 6.9 - Column Specification of the foreign key

Table shows the typical mapping between Class Diagram and Entity Relationship Diagram.

Class Diagram	Entity Relationship Diagram

Table 6.5

Mapping Many-to-Many Association

For a many-to-many association between two classes, DB-VA will generate a Link Entity to form two one-to-many relationships in-between two generated entities. The primary keys of the two entities will migrate to the link entity as the primary/foreign keys.

Example:

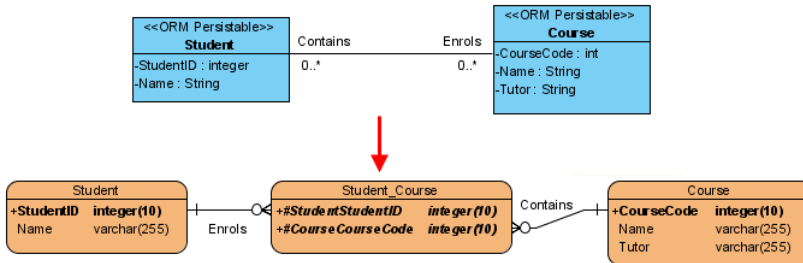


Figure 6.10 - Mapping Many-to-Many Association

In the above example, DB-VA generates the link entity, Student_Course between entities of Student and Course when transforming the many-to-many association.

Mapping Inheritance/Generalization

Generalization distributes the commonalities from the superclass among a group of similar subclasses. The subclass inherits all the superclass's attributes and it may contain specific attributes.

DB-VA provides two strategies for transforming the generalization hierarchy to relational model. The two strategies for transformation are table per class hierarchy and table per subclass.

Using Table per Class Hierarchy Strategy

Transforming generalization hierarchy to relational model with the table per class hierarchy strategy, DB-VA not only combines all the classes within the hierarchy into one single entity containing all the attributes, but also generates a discriminator column to the entity. The discriminator is a unique value identifying the entity which hierarchy it belongs to.

By using the table per class hierarchy strategy, the time used for reading and writing objects can be saved. However, more memory is used for storing data. It is useful if the class will be loaded frequently.

Example:

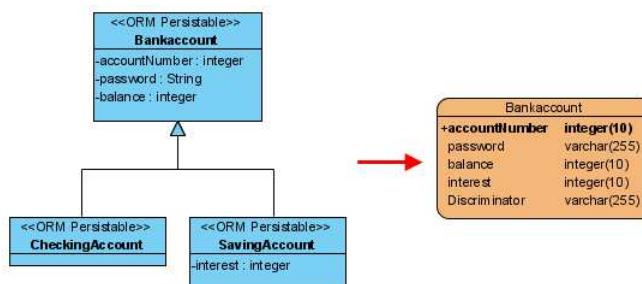


Figure 6.11 - Mapping Inheritance/Generalization using Table per Class Hierarchy Strategy

In the above example, it shows how DB-VA transforms the generalization with CheckingAccount, SavingsAccount and their superclass, BankAccount applying the table per class hierarchy strategy.

Using Table per Subclass Strategy

When DB-VA transforms a generalization hierarchy using the table per subclass strategy to relational model, each subclass will be transformed to an entity with a one-to-one identifying relationship with the entity of the superclass.

By using the table per subclass strategy, it can save memory for storing data. However, it takes time for reading and writing an object among several tables which slows down the speed for accessing the database. It is useful if the class, which contains a large amount of data, is not used frequently.

Example:

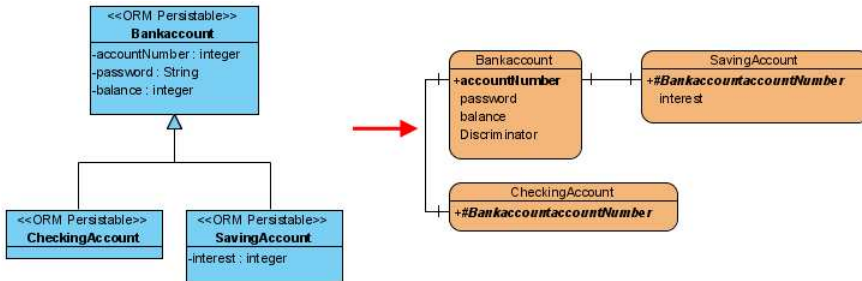


Figure 6.12 - Mapping Inheritance/Generalization using Table per Subclass Strategy

In the above example, it shows how DB-VA transforms the generalization which applies the table per subclass hierarchy strategy to the **CheckingAccount** and **SavingsAccount**.

Using Mixed Strategies

DB-VA allows applying the two inheritance strategies to different subclasses within a generalization hierarchy in Java project. By applying different strategies to different subclasses within a generalization hierarchy, DB-VA transforms the generalization hierarchy with respect to the specified inheritance strategies.

Example:

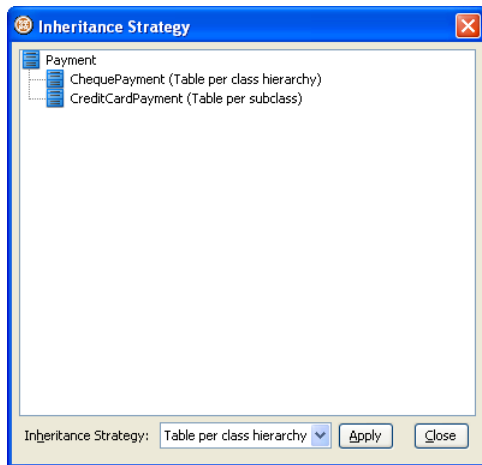


Figure 6.13 - Using Mixed Strategies

Applying the above inheritance strategy to the generalization with the **ChequePayment** and **CreditCardPayment** and their superclass, **Payment**, DB-VA transforms it into two entities as shown below.

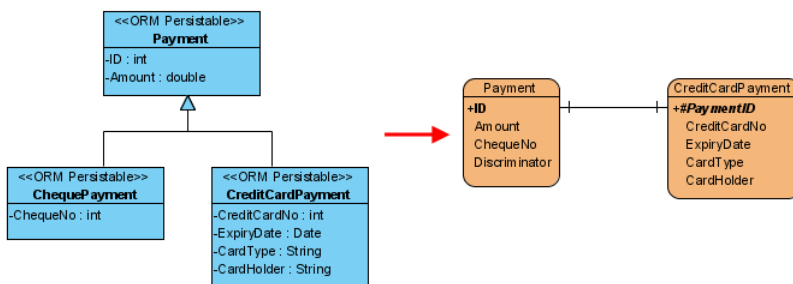


Figure 6.14 - Mapping with mixed Strategies

In the above example, it shows that applying table per hierarchy inheritance strategy will result in combining the attributes of the superclass and subclass into one single entity while table per subclass will result in forming an entity of subclass and a one-to-one identifying relationship between entities of superclass and subclass.

Note



Applying two inheritance strategies to different subclasses within a generalization hierarchy is only available in Java project. If mixed strategies are applied to the generalization hierarchy in .NET project, it will result in error when the generation of code and database.

Mapping Collection of Objects to Array Table

For a persistent class acting as persistent data storage, it may consist of a persistent data containing a collection of objects. DB-VA promotes the use of Array Table to allow users retrieve objects in the form of primitive array.

When DB-VA transforms a class with an attribute of array type modifier, this attribute will be converted to an Array Table automatically. The generated entity and the array table form a one-to-many relationship.

Example:

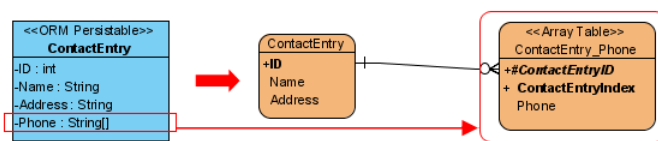


Figure 6.15 - Mapping Collection of Objects to Array Table

In the above example, the phonebook has a contact entry for each contact person. Each contact person may have more than one phone numbers. In order to ease the retrieval of a collection of phone objects, DB-VA converts the phone attribute into a ContactEntry_Phone array table.

Mapping Object Model Terminology

Table shows the shift from object model to data model terminology.

Object Model Term	Data Model Term
Class	Entity
Object	Instance of an entity
Association	Relationship
Generalization	Supertype/subtype
Attribute	Column
Role	Phrase
Multiplicity	Cardinality

Table 6.6

Mapping Data Model to Object Model

This section shows you how DB-VA maps the data model to object model.

Mapping Entities to Classes

All entities map one-to-one to persistent classes in an object model.

Example:

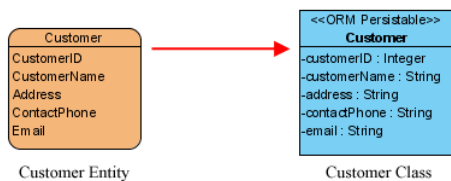


Figure 6.16 - Mapping Entities to Classes

In the above example, the Customer Entity map one-to-one the Customer Class as the Customer instance can store the customer information from the Customer Entity.

Mapping Columns to Attributes

Since all entities map one-to-one to persistent classes in an object model, columns in turn map to attributes in a one-to-one mapping. DB-VA ignores all specialty columns such as computed columns and foreign key columns.

Example:

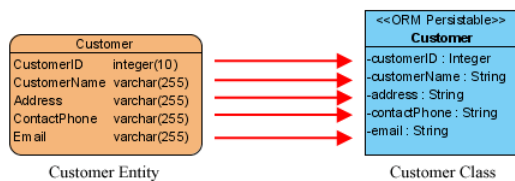


Figure 6.17 - Mapping Columns to Attributes

In the above example, the following table shows the mapping between the columns of the Customer Entity and the attributes of the Customer Class.

Customer Entity	Customer Class
CustomerID	CustomerID
CustomerName	CustomerName
Address	Address
ContactPhone	ContactPhone
Email	Email

Table 6.7

Mapping Data Type

DB-VA automatically maps the column data type to an appropriate attribute type of object model.

Example:

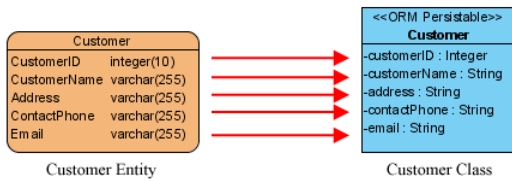


Figure 6.18 - Mapping Data Type

In the above example, the following table shows the mapping between data types

Customer Entity	Customer Class
int (10)	int
varchar(255)	String

Table 6.8

A table shows the data type mapping between Object model and Data model.

Data Model	Object Model
Bigint	Long
Binary	Byte[]
Bit	Boolean
Blob	Blob
Varchar	String
Char	Character
Char(1)	Character
Clob	String
Date	Date
Decimal	BigDecimal
Double	Double
Float	Float
Integer	Integer
Numeric	BigDecimal
Real	Float
Time	Time
Timestamp	Timestamp
Tinyint	Byte
Smallint	Short
Varbinary	Byte[]

Table 6.9

Mapping Primary Key

As the columns map to attributes in a one-to-one mapping, primary key columns in the entity map to attributes as a part of a class.

Example:



Figure 6.19 - Mapping Primary Key

In the example, the primary key of entity Product, **ProductID** maps to an attribute ProductID of the class Product.

Mapping Relationship

Relationship represents the correlation among entities. Each entity of a relationship has a role, called Phrase describing how the entity acts in it. The phrase is attached at the end of a relationship connection line. DB-VA maps the phrase to role name of association in the object model.

There are two types of relationships in data model mapping to object model - identifying and non-identifying.

Identifying relationship specifies the part-of-whole relationship. It means that the child instance cannot exist without the parent instance. Once the parent instance is destroyed, the child instance becomes meaningless.

Non-identifying relationship implies weak dependency relationship between parent and child entities. There are two kinds of non-identifying relationships, including optional and mandatory. The necessity of the parent entity is "exactly one" and "zero or one" in the mandatory and optional non-identifying relationship respectively.

Mapping Identifying Relationship

Since the identifying relationship specifies the part-of-whole relationship, it maps to composite aggregations which implies that the part cannot exist without its corresponding whole.

Example:

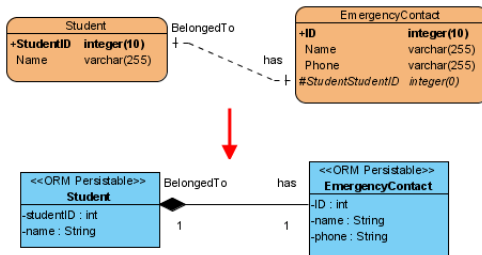


Figure 6.20 - Mapping Identifying Relationship

In the above example, DB-VA maps the identifying relationship between the entities of EmergencyContact and Student to composition aggregation.

Mapping Non-identifying Relationship

Since non-identifying relationship implies weak relationship between entities, DB-VA maps it to association.

Example:

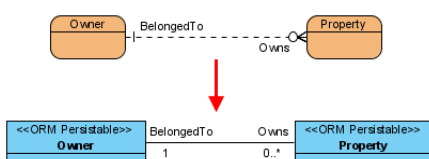


Figure 6.21 - Mapping Non-identifying Relationship

In the above example, non-identifying relationship between entities Owner and Property maps to association between Classes of Owner and Property.

Mapping Cardinality

Cardinality refers to the number of possible instances of an entity relate to one instance of another entity. The following table shows the syntax to express the Cardinality.

Table shows the syntax expressing the Cardinality

Type of Cardinality	Description
+○—	Zero or one instance
≥○—	Zero or more instances
+—	Exactly one instance
≥—	One or more instances

Table 6.10

Table shows the typical mapping between Entity Relationship Diagram and Class Diagram.

Entity Relationship Diagram	Class Diagram

Mapping Array Table to Collection of Objects

DB-VA promotes the use of Array Table to allow users retrieve objects in the form of primitive array. When DB-VA transforms an array table, the array table will map to an attribute with array type modifier. Example:

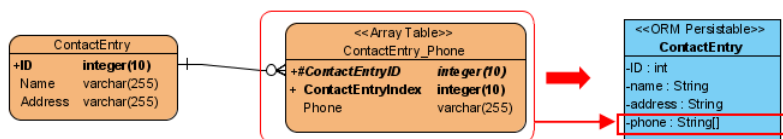


Figure 6.23 - Mapping Array Table to Collection of Objects

In the above example, the phonebook has a contact entry for each contact person. Each contact person may have more than one phone numbers. The array table of ContactEntry_Phone maps into the phone attribute with array type modifier in the ContactEntry class.

Mapping Data Model Terminology

The following table shows the shift from data model to object model terminology.

Data Model Term	Object Model Term
Entity	Class
Instance of an entity	Object
Relationship	Association
Supertype/subtype	Generalization
Column	Attribute
Phrase	Role
Cardinality	Multiplicity

Table 6.12

Showing Mapping by ORM Diagram

In order to identify the mapping between the object and data models clearly, DB-VA provides you an ORM diagram to show the mappings between the ORM-Persistable class and its corresponding entity.

As DB-VA allows you to name the ORM-Persistable class and its corresponding entity differently, and also the attributes and columns as well, you may find it difficult to identify the mappings not only between the ORM-Persistable classes and the corresponding entities, but also the attributes and columns. Taking the advantage of ORM diagram, the mappings between ORM-Persistable classes and entities and between attributes and columns can be clearly identified.

DB-VA provides you two ways to create the ORM diagram for showing the mapping between the object and data models:

1. Creating an ORM diagram from the existing class diagram and/or ERD.
2. Drawing an ORM Diagram

Creating an ORM Diagram from Existing Diagrams

The following example shows you how to show the mapping between the existing object and data models by the ORM diagram.

Let us assume the following class diagram has been created and synchronized to the entity relationship diagram (ERD).

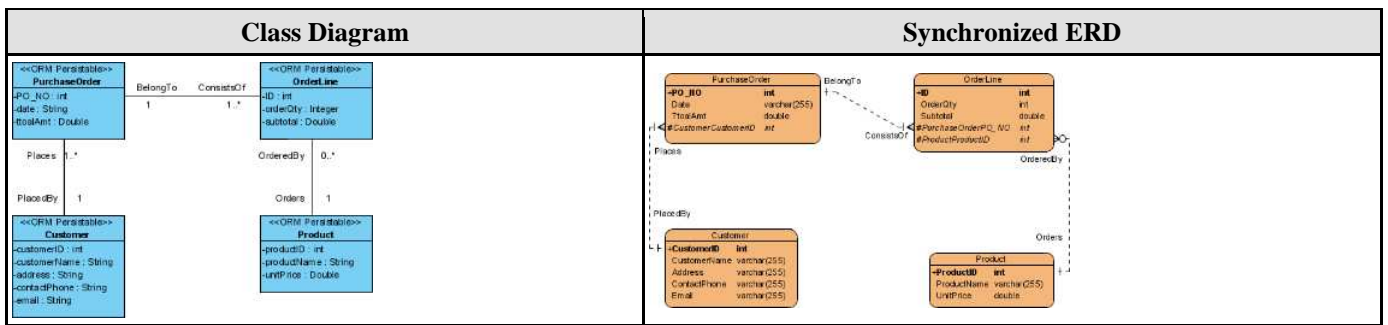


Table 6.13

You can create an ORM diagram by either the object model, data model, or both. The following steps show you how to create the ORM diagram by using the object model.

1. Right-click on the class diagram, select **Send to > ORM Diagram > New ORM Diagram** from the pop-up menu.

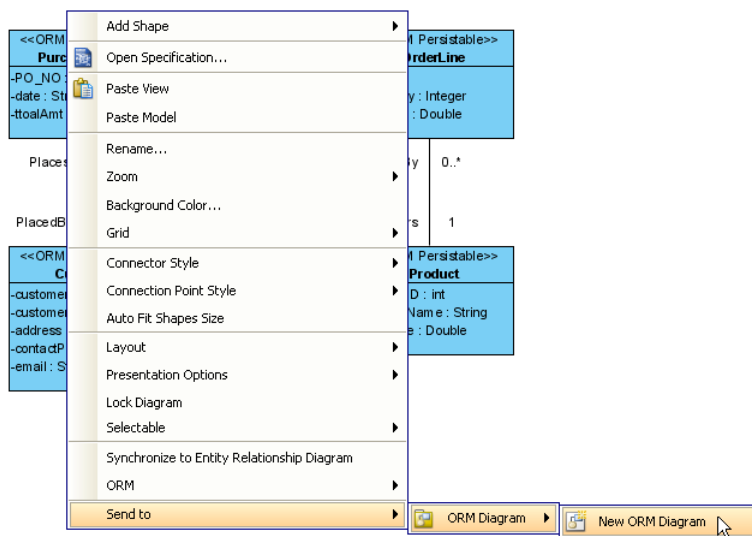


Figure 6.24 - Send to a blank new ORM Diagram

A new ORM diagram is created which displays the ORM-Persistable classes.

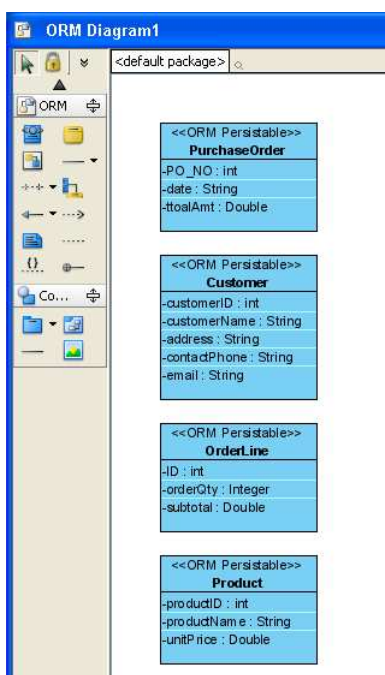


Figure 6.25 - The classes in ORM Diagram

Note

Alternatively, you can create the ORM diagram from the data model by right-clicking on the ERD, and selecting **Send to > ORM Diagram > New ORM Diagram** from the pop-up menu.

2. Mouse over the ORM Persistable class, click the **Class-Entity Mapping - > Entity** resource.

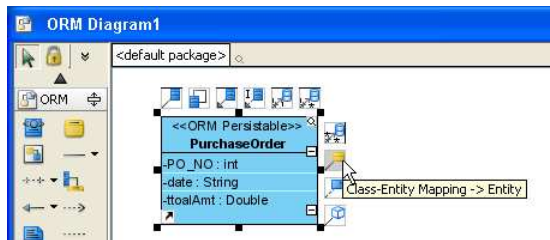


Figure 6.26 - "Class-Entity Mapping - > Entity" resource

3. Drag the resource on the ORM diagram to show the corresponding entity associated in the mapping.



Figure 6.27 - Drag the resource to the diagram

A class-to-entity mapping is shown on the diagram.

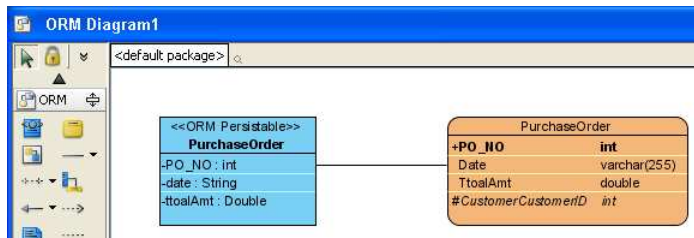


Figure 6.28 - The Mapping Entity will be created

- By repeating steps 2 and 3, all class-to-entity mappings for all classes can be shown.

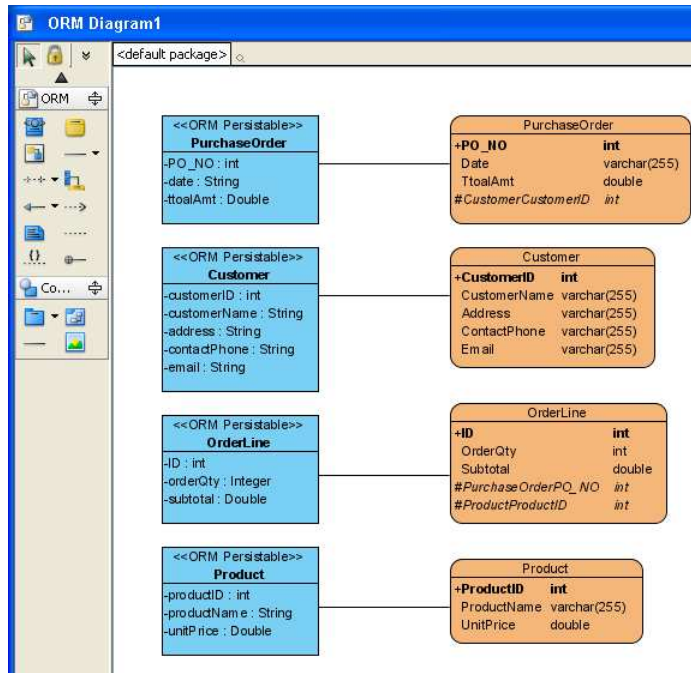


Figure 6.29 - Mapping entities for all classes

Note
 If you have created the ORM diagram from the data model, you can show the class-to-entity mapping by dragging the **Class-Entity Mapping -> Class** resource.

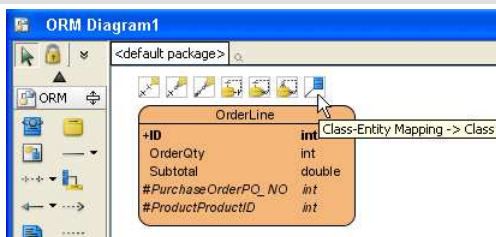


Figure 6.30 - "Class-Entity Mapping -> Class" resource on Entity

Drawing an ORM Diagram

You can create a new ORM diagram in one of the three ways:

- On the menu, click **File > New Diagram > Others > ORM Diagram**.

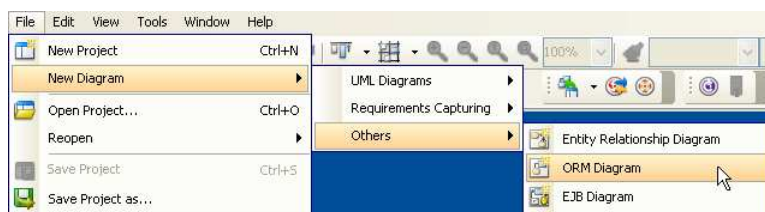


Figure 6.31 - Create an ORM Diagram

- On the **Diagram Navigator**, right-click **ORM Diagram** > **Create ORM Diagram**.

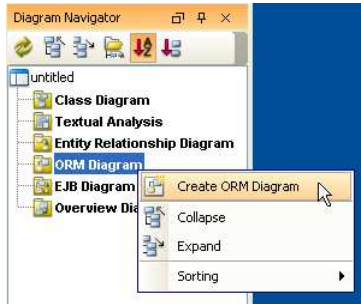


Figure 6.32 - Create ORM Diagram on Diagram Navigator

- On the toolbar, click the **New ORM Diagram** icon.

A new ORM diagram is displayed.

Creating ORM-Persistable Class and Mapping Entity to the ORM Diagram

After created a new ORM diagram, you can create ORM-Persistable class and its mapping entity on the ORM diagram.

To create an ORM-Persistable class on ORM diagram:

1. On the diagram toolbar, click **ORM-Persistable Class** shape icon.

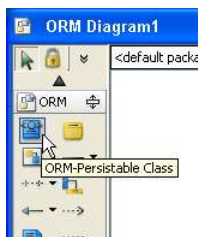


Figure 6.33 - ORM Persistable Class icon

2. Click a location in the diagram pane.

DB-VA places a class shape icon which is marked with **<<ORM-Persistable>>** on the diagram.

3. Type the name for the **ORM-Persistable Class**.
 - You can edit the name by double-clicking the name or by pressing the **F2** button.

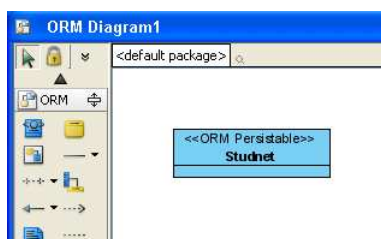


Figure 6.34 - ORM Persistable Class

Creating Associated ORM-Persistable Class to the ORM Diagram

You can create an associated ORM-Persistable class by using the association resources of an ORM-Persistable class.

1. Mouse over the ORM-Persistable class, drag the **One-to-One Association -> Class** resource to the diagram to create another ORM-Persistable class with a one-to-one directional association.

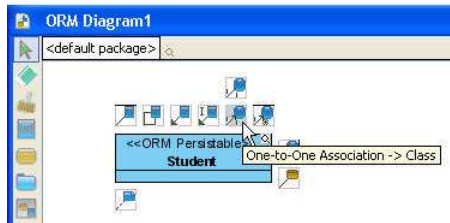


Figure 6.35 - "One-to-One Association -> Class" resource

2. Enter the name to the newly created class.

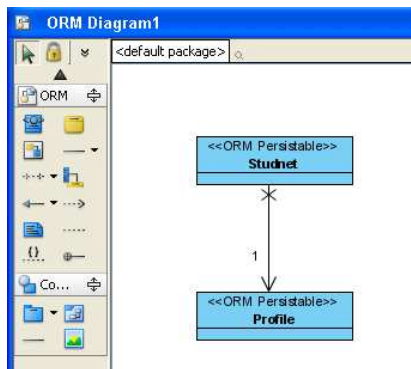


Figure 6.36 - An ORM Persistable Class are created

Creating Mapping Entity to the ORM Diagram

To create the mapping entity of the ORM-Persistable class:

1. Mouse over the ORM-Persistable class, click and drag the **Class-Entity Mapping -> Entity** resource to the diagram.

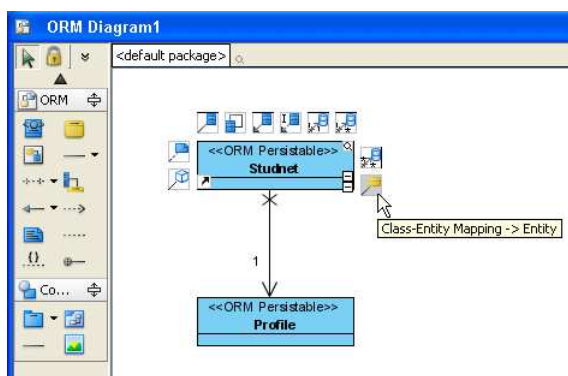


Figure 6.37 - "Class-Entity Mapping -> Entity" resource

The corresponding mapping entity, **Student** is created automatically.

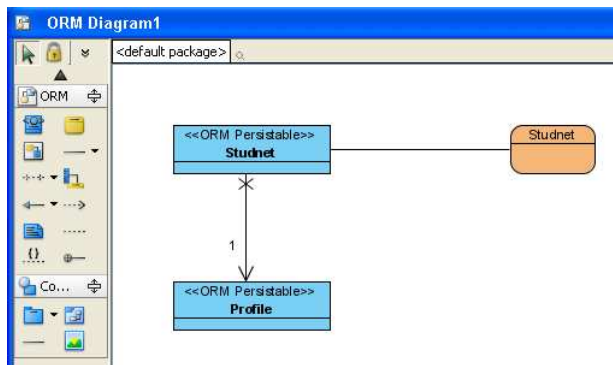


Figure 6.38 - Mapping Entity are created

2. Create the mapping entity of Profile class by using the **Class-Entity Mapping - > Entity** resource.

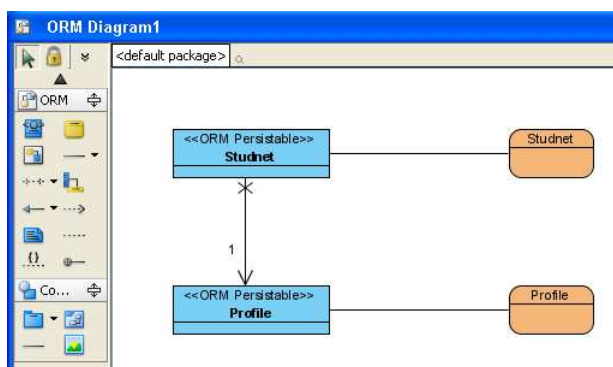


Figure 6.39 - Mapping Entities for the Classes

Note



You can create the Entity and the mapping ORM-Persistable class using the same approach of Creating ORM-Persistable Class and Mapping Entity by using the **Entity** icon on the diagram toolbar and the **Class-Entity Mapping - > Class** resource.

Showing Attribute Mapping

The object-relational mapping exists not only between the ORM-Persistable class and entity, but also the attributes and columns. You can investigate the mapping between the attributes and columns by using the Attribute Mapping feature.

To view the attribute mapping:

1. Right-click on the ORM diagram, select **View > Attribute Mapping** from the pop-up menu.

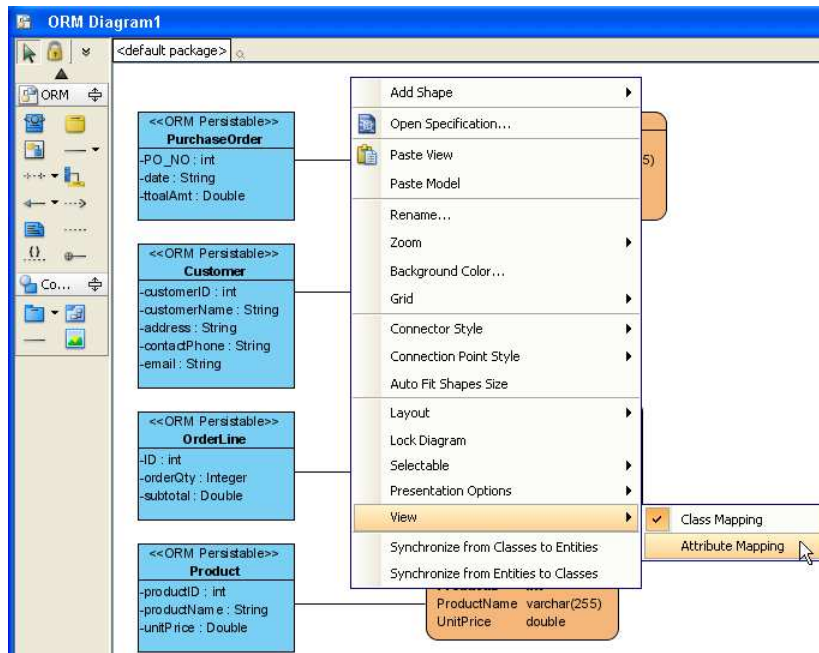


Figure 6.40 - Switch to Attribute Mapping view

The class-to-entity mapping shown on the ORM diagram is changed to attribute-to-column mapping automatically.

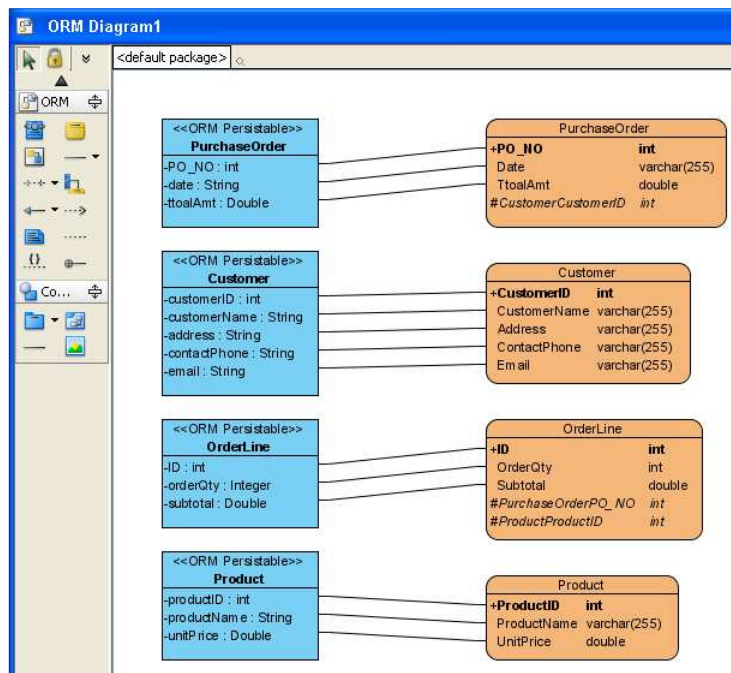


Figure 6.41 - ORM Diagram in Attribute Mapping view

Supporting Real-time Synchronization

ORM diagram supports real-time synchronization; that is, any change in the class diagram, entity relationship diagram and/or ORM diagram will automatically synchronize to each other.

Let us take the ORM diagram created in the [Drawing ORM Diagram](#) section as an example to modify the ORM-Persistable Class and Entity.

Forming a Class Diagram

You can create a class diagram from the existing models.

1. Create a new class diagram by using **New Class Diagram** icon.
2. Select the classes from the **Class Repository**, drag to the newly created class diagram.

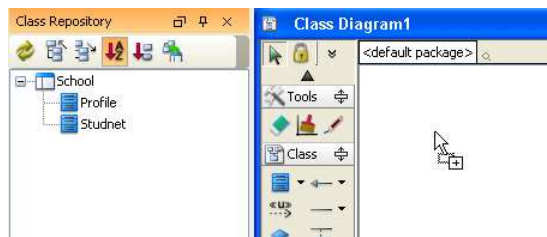


Figure 6.42 - Drag the classes from Class Repository

The following class diagram is created.

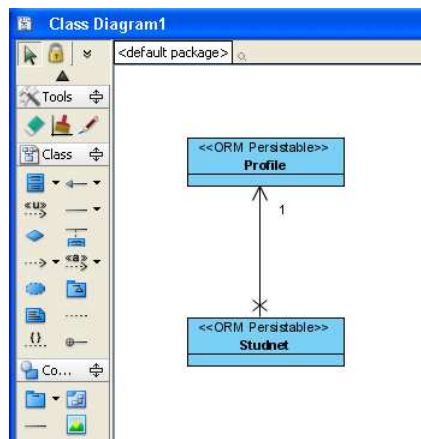


Figure 6.43 - Class Diagram form by the model in Class Repository

Modifying ORM-Persistable Class

You can modify the ORM-Persistable class such as renaming the class name and adding attributes to the class.

1. Right-click the **Student** class, select **Add > Attribute** from the pop-up menu.

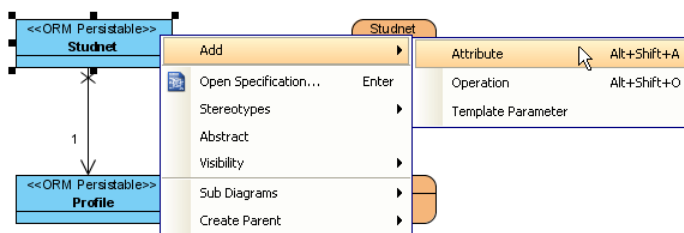


Figure 6.44 - Add an attribute in Class

An attribute is added to the Student class, and the mapping attribute is added to the mapping Student entity automatically.

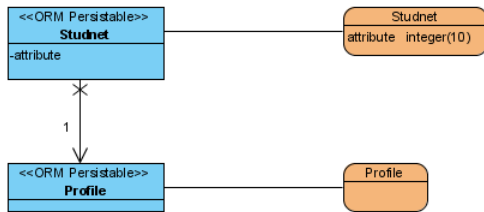


Figure 6.45 - The column will added automatically

2. Enter "**StudentID : String**" to the attribute of Student by double-clicking on the attribute. The type of the mapping column is mapped to varchar(255) automatically.

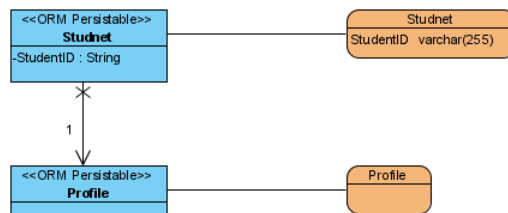


Figure 6.46 - The column will update when the mapping attribute modified

Modifying Entity

You can modify the entity such as renaming the entity name and adding columns to the entity.

1. Rename the column of **Student** entity from **attribute** to **ID** by double-clicking on it.

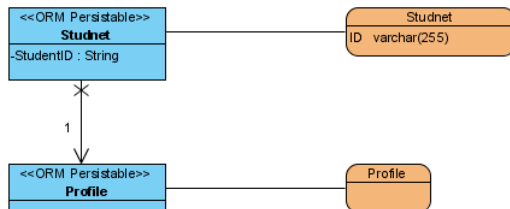


Figure 6.47 - Rename the column name and select not update to attrbiute

2. Right-click the **Student** entity, select **New Column** from the pop-up menu.

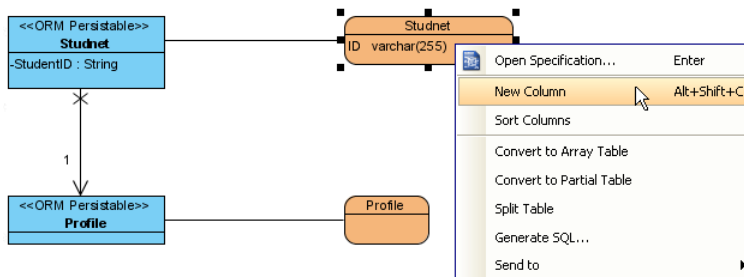


Figure 6.48 - Add a column to an Entity

A new column is added to the Student entity and the corresponding attribute is added to the Student ORM-Persistable class automatically.

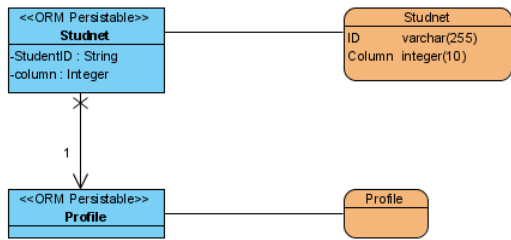


Figure 6.49 - Attribute will be added automatically

3. Enter "Name : varchar(255)" to the column by double-clicking on it. The type of the mapping attribute is mapped to String automatically.

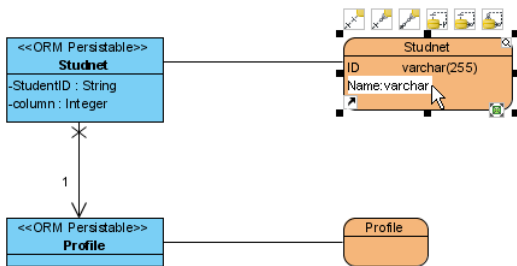


Figure 6.50 - Rename the column and data type

4. Double-click the column attribute of the Student class, rename from **column** to **Name**.

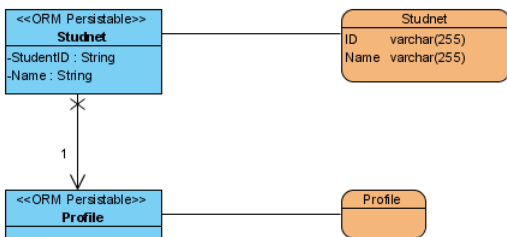


Figure 6.51 - The attribute changed

5. Modify the classes and entities on the ORM diagram as shown below:

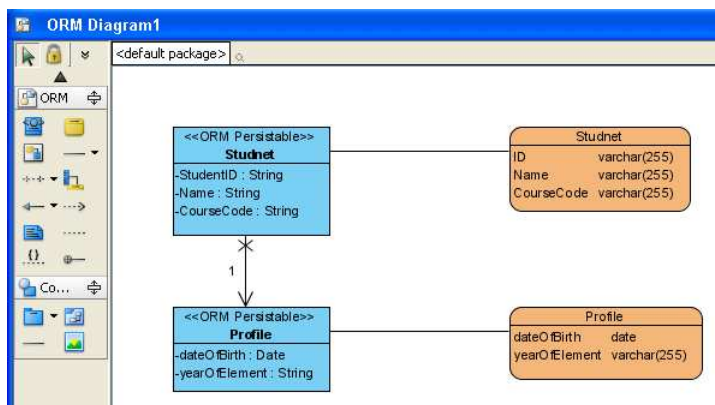


Figure 6.52 - Modify the Class in ORM Diagram

- Navigate to the class diagram, the class diagram is updated automatically.

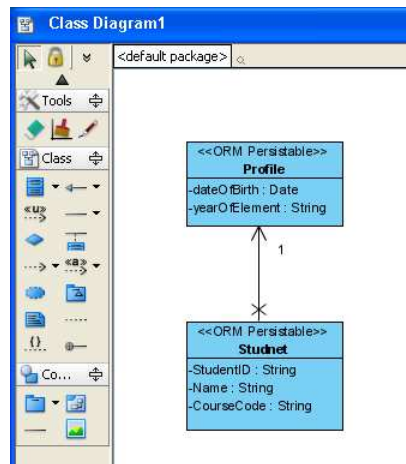


Figure 6.53 - The classes in Class Diagram is updated automatically

Switching the View of Mapping

As ORM diagram provides two views of mapping, including the mapping between ORM-Persistable class and entity (called Class Mapping), and the mapping between attributes and columns (called Attribute Mapping).

To change the view of mapping, right-click on the ORM-diagram, select the desired view from the sub-menu of **View**.

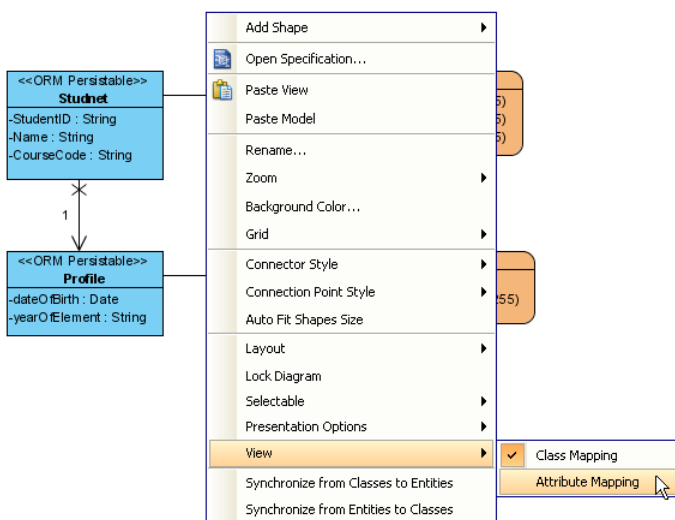


Figure 6.54 - Switch to Attribute mapping view

By selecting the **View > Attribute Mapping** from the pop-up menu, The class-to-entity mapping shown on the ORM diagram is changed to attribute-to-column mapping automatically.

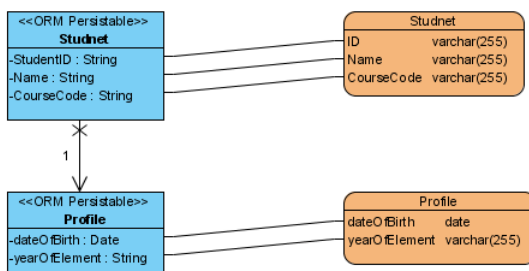


Figure 6.55 - The ORM Diagram in Attribute Mapping

