Mapping object model to data model

Visual Paradigm supports Object Relational Mapping (ORM) which maps object models to entity relational models and vice versa. Visual Paradigm 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 classes to entities

Generally speaking, the mapping between class and entity is a one-to-one mapping, meaning that one class in object model maps with one entity in data model. Classes that map with entities are represented by the stereotype .

In the above example, the *Customer* class is the object equivalent of the *Customer* entity. This means that in application development or in runtime, an instance of *Customer* (class) stores the information of a customer retrieved from the *Customer* table of database.

Mapping attributes to columns

Since the persistent classes map to the entities, persistent attributes map to columns accordingly. Visual Paradigm ignores all non persistent attributes such as derived values.

Mapping data type

Persistent attribute types are automatically mapped to appropriate column data types of the database you desired. The following table lists out the typical mapping between object model and data model. Note that the actual data type to map to depends on the default database you selected in database configuration.

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)

Object Model	Data Model
Date	Date
Time	Time (7)
Timestamp	Timestamp (7)

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 window to select primary key.

You can select an existing attribute as primary key, let us generate one for you. or select **Do Not Generate** to leave the generated entity without primary key.

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, Visual Paradigm 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. Visual Paradigm 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.

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 composition

Composition 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.

In the above example, Visual Paradigm performs the Primary/Foreign Key Column Mapping automatically. *ID* 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 Multiplicity.

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

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

Mapping many-to-many association

For a many-to-many association between two classes, Visual Paradigm 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.

In the above example, Visual Paradigm 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.

We provide multiple strategies for transforming the generalization hierarchy to relational model. Click here to learn more about the various inheritance strategies.

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. Visual Paradigm promotes the use of Array Table to allow users retrieve objects in the form of primitive array.

When Visual Paradigm 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.

In the above example, each contact person may have more than one phone numbers. In order

to ease the retrieval of a collection of phone objects, Visual Paradigm converts the phone attribute into a *ContactEntry_Phone* array table.

Typical mapping between object model and data model

Object Model	Data Model