BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2102

DATABASE MANAGEMENT SYSTEM

Unit 14

Object Relational Mapping

Table of Contents

1. INTRODUCTION

We have already discussed so far that the database management task is done so that all the data working with the application system can be retained, resulting in information and knowledge. This task may be done by structured query language database management systems (SQL DBMS) or by object oriented programming techniques or other traditional methods. SQL DBMS manipulates scalar values such as integers and strings which are organized within normalized tables whereas the Object-oriented programming technique implements data management by manipulating the object that represents non-scalar values. In order to maintain and manage a database, a programmer can either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval) or can use simple scalar values within the program.

In this unit, we will discuss Object-relational mapping (ORM or O/RM or O/R mapping) which is a programming technique for converting data between two different paradigms (a relational database and an object-oriented programming language). We will also discuss the significance of mapping, modeling, and implementation aspects related to Database management.

1.1 Objectives

After studying this unit, you should be able to:

- *Describe Object Relational Mapping and its significance.*
- *Discuss the Mapping basics*
- *Explain strategies for representing a class inheritance tree*
- *Describe how object relationships can be mapped to the relational model*
- *Explain the open-source object relational mapping software.*

2. SIGNIFICANCE OF MAPPING

We have already mentioned that Object-relational mapping (ORM or O/RM or O/R mapping) is a programming technique for converting data between two different paradigms (a relational database and an object-oriented programming language) where data representations, data manipulation, modeling techniques and other entities related to it are different. It helps to create a "virtual object database" that can be used from within the programming language so that the objects can be translated into forms that can be stored in the database for easy retrieval while preserving the properties of the objects and their relationships.

Persistent Object

While dealing with ORM you have to know what a persistent object is. It is the object that can automatically store and retrieve itself in the database while preserving the properties of objects and their relationships.

Although the advantage of ORM is that it often reduces the amount of code needed to be written, the disadvantage would be due to some O/R mapping tools that do not perform well during bulk deletions of data. Hence ORM software has been pointed to as a major factor in producing poorly designed databases.

Now let us discuss what the significance of mapping is. When we write an application, for example in Java, which stores data in an RDBMS by relying on data-aware controls (e.g. data-aware GUI components) to interface directly with the database then such applications are not object oriented, and by using such two-layer (GUI/Database) applications, we lose the benefits of object oriented design principle i.e. encapsulation. Applications that use data-aware widgets or controls, the client and the database are very tightly coupled where GUI code, business logic, and SQL statements are all interwoven throughout the application source code. So, any changes in the database schema will surely cascade into unexpected failures resulting in a maintenance nightmare. This type of problem where technical difficulties are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style is known as object-relational impedance mismatch. This type of problem occurs when objects

or class definitions are mapped in a straightforward way to database tables or relational schema.

Object-relational impedance mismatch is due to the following reasons:

- objects can't be directly saved to and retrieved from relational databases
- objects have an identity, state, and behavior in addition to data whereas RDBMS stores data only.
- even data alone can present a problem since there is often no direct mapping between Java and RDBMS data types.
- objects are traversed using direct references while RDBMS tables are related via like values in foreign and primary keys.
- current RDBMS has no parallel to Java's object inheritance for data and behavior.
- the goal of relational modeling is to normalize data (i.e., eliminate redundant data from tables), whereas the goal of object-oriented design is to model a business process by creating real-world objects with data and behavior.
- One of the main objectives of Object-Relational mapping is to solve the problem stated above (i.e. the object-relational impedance mismatch).

Self-Assessment Questions - 1

1. Object-relational impedance mismatch occurs when objects or class definitions are mapped in a straightforward way to_____.
2. The main objective of Object-Relational mapping is to solve object-relational _____.

3. MAPPING BASICS

We can see that classes of OOP language can be mapped to RDBMS tables. For e.g. JAVA classes can be mapped to RDBMS tables.

We can map a persistent class and a table in so many ways. The simplest mapping between a persistent class and a table is one-to-one. In this case, all attributes in the persistent class are represented by all columns of a table. Each instance of a business class is then stored in a row of the table.

Although this type of mapping is straightforward, it might conflict with the existing object and entity-relation (ER) models. This is **because** the goal of object modeling is to model a business process using real-world objects, whereas the goal of ER modeling is normalization and fast retrieval of data.

Let us take an example of the Visual Business Sight Framework ™ (VBSF) which is an object-relational Java framework that allows Java objects to be easily saved and retrieved from relational databases. Here a Java developer has to work with two different models: (i) an object model and (ii) a relational model. As Java has to access relational databases and must deal with tables, rows, and columns, Developers have to go work with tedious routines that convert rows and columns into Java objects. This is known as the impedance mismatch between Java objects and relational databases. VBSF contains an object-relational mapping engine that automatically implements object persistence to relational databases, allowing a programmer to forget about JDBC and stay focused on the object model.

As a result, VBSF supports two types of class-to-table mappings that help overcome the differences between relational and object models: **SUBSET and SUPERSET.**

SUBSET Mapping: This type of mapping is used when all the attributes of a persistent class are mapped to the same table. It is also used when a persistent class is not concerned with some of the columns (not part of the business model) of its corresponding table in the database.

The attributes of a persistent class with a subset mapping represent either a portion of the columns in a table or all of the columns in the table. Subset mappings can also create

"projection classes" for tables with a large number of columns. A projection class allows a user to select a row for full retrieval from the database. The full row can be mapped to another persistent class. This type of design reduces the amount of information passed across the network. Subset mappings are also used to map a class inheritance tree to a table using filters.

SUPERSET Mapping: This is done when a persistent class with a superset mapping contains attributes derived from columns of multiple tables. This type of mapping is also known as table spanning. Superset mappings can be used to create "view classes" that hide the underlying data model. It also can map a class inheritance tree to the database using a Vertical mapping approach. VBSF fully supports performing insertions, updates, and deletions of objects with this type of mapping, while transparently updating and maintaining all foreign key columns that join the tables.

Self-Assessment Questions - 2

3. SUBSET Mapping is used when all the attributes of a persistent class are mapped to a different table. (True/False)
4. SUPERSET Mapping is done when a persistent class with a superset mapping contains attributes derived from columns of_____.

.

4. MAPPING A CLASS INHERITANCE TREE

A class inheritance tree can be mapped in the RDBMS as **vertical** mapping, **horizontal** mapping, and **filtered** mapping.

VBSF supports all three methods. A combination of these types of mappings can also be used within an inheritance tree.

- **Vertical Mapping:** In vertical mapping, each class in the tree, whether abstract or concrete, is mapped to a different table. All branch and leaf tables in the tree must be linked to their parent tables. This is done by the use of a foreign key column that references the primary key of the parent table.
- **Horizontal Mapping:** Under horizontal mapping, each concrete class in the tree is mapped to a different table. Each of these mapped tables contains columns for all attributes in its concrete class, plus all attributes inherited from all its abstract parent classes. In other words, abstract classes are not mapped to their own table. This approach results in fast performance and is simple to design. However, if an attribute of an abstract parent class is changed, then potentially many tables must be modified. Consequently, this method is most useful if the inheritance tree is more method-driven than attribute-driven. In Short, if a substantial number of classes inherit a large number of attributes from an abstract parent class, then the vertical or filtered methods might be better choices.
- **Filtered Mapping:** In filtered mapping, all concrete classes in the tree are mapped to the same table. The table must contain columns for all attributes of all the abstract and concrete classes in the inheritance tree (or the part of the tree using this mapping). In addition, a filter column is created in the table. The value of the filter column is used to distinguish between subclasses. Abstract classes are not mapped to this table. This approach provides adequate performance but violates table-normalization rules. More specifically, it could lead to a substantial number of NULL columns in the table resulting in wastage of space. Consequently, this method is most useful if most of the attributes are inherited from the abstract parent classes.

Self-Assessment Questions - 3

5. In vertical mapping, each class in the tree, whether abstract or concrete, is mapped to a_____table.
6. In filtered mapping, a filter column is created in the table and the value of the filter column is used to distinguish between subclasses. (True/False)

5. MAPPING OBJECT RELATIONSHIPS

We have discussed the class inheritance tree mapping in the previous section. Now we will discuss the art of relationship mapping. There are three types of object relationships that may be needed to map: association, aggregation, and composition. But as of now, let us treat these three types of relationships the same (i.e. They are mapped in the same way although mapping, in the same way, creates a problem when it comes to referential integrity.)

5.1 Types Of Relationships

Basically object relationships (for mapping to the relational model) can be categorized based on multiplicity, directionality, ownership, and reference.

Based on the multiplicity, object relationships can be **One-to-one relationships, One-to-many relationships, and Many-to-many relationships.**

- **One-to-one relationships:** This is a relationship where the maximum of each of its multiplicities is one.

  In the relational model, a one-to-one relationship is usually maintained by means of an embedded foreign key column. This foreign key column holds the value of the primary key (the object ID) of the row (object) being referenced.
  For e.g. a one-to-one relationship is implemented in VBSF by defining a Reference attribute. VBSF transparently converts the foreign key reference to an object and updates the value of the foreign key column from referenced objects.

  It is also possible to define a one-to-one relationship in the relational model using a join table. VBSF also supports this scenario.

- **One-to-many relationships:** This is also known as a many-to-one relationship, this occurs when the maximum of one multiplicity is one and the other is greater than one.

  In the object model, there are two types of one-to-many relationships: aggregation (part-of), and association (acquaintance).
  Under VBSF an aggregation relationship is defined by means of an Owned Collection attribute, and an association relationship by means of a Referenced Collection attribute.

The difference between the two is that in an owned relationship when the owner is updated in the database, all objects in all its owned collections are automatically updated (this default behavior can be overridden at runtime if necessary).

In the relational model, a one-to-many relationship can be defined either using an embedded foreign key column or using a join table. An embedded foreign key is a column defined in the table on the many sides of the relationship that holds a key to the table on the one side of the relationship. A join table is a table whose main purpose is to store mapping values between the key and foreign key of the two tables involved in the relationship.

An owned relationship can only be implemented using an embedded foreign key column. A referenced relationship can be implemented using either method-embedded foreign key or join table.

- **Many-to-many relationships:** This is a relationship where the maximum of both multiplicities is greater than one.

  A many-to-many relationship can be thought of as a bi-directional one-to-many association. To create this type of relationship you simply define a Referenced Collection attribute in each of the classes involved in the relationship. In the relational model, a many-to-many relationship can be defined either using foreign key columns or using a join table.

  To use foreign keys, an embedded foreign key column is defined in each of the tables involved in the relationship. Each foreign key column holds the key to the other table.

Based on directionality, objects' relationship can be **uni-directional** relationships and **bi-directional** relationships.

- **Uni-directional relationships**: A uni-directional relationship exists between two objects when an object knows about the object(s) it is related to but the other

object(s) does not know about the original object. Uni-directional relationships are easier to implement than bi-directional relationships.

Example - The holds relationship between Employee and Position. Employee objects know about the position that they hold, but Position objects do not know which employee holds it (also there is no need to do so).

- **Bi-directional relationships:** A bi-directional relationship exists when the objects on both end of the relationship know each other.

  Example – The works-in relationship between the Employee and Department. Employee objects know in which department they work and Department objects also know what employees work in them.

The other types of relationships can be:

Owned relationships (Aggregation): In an owned relationship, the owner class holds a reference to its owned collection using an Owned Collection attribute and the owned class holds a direct reference to its owner using a Reference attribute. An owner object automatically handles all inserts, updates, and deletions of all its owned objects in the database.

VBSF automatically and transparently takes care of updating owned objects in the database whenever an owner object is updated in the database. This behavior is recursive, so if any owned objects are in turn owners (i.e. they hold other owned collections), all their owned objects will also be updated in the database too, and so on. In this manner, whole object graphs can be updated in the database in one operation. This type of behavior is known as persistence by reachability.

An owned relationship is implemented in the relational model using an embedded foreign key column in the table on the many sides of the relationship.

Referenced relationships (Association): This type of relationship exists when (for e.g. in a one-to-many association) the class on the one side of the relationship merely references other classes and does not own it. In this case (one-to-many association) an object on the one side (also referred to as the holder of the collection) of the relationship cannot create, update, or delete objects on the other side of the relationship and can only retrieve them. So the holder has to use a Referenced Collection attribute to reference its associated collection

instead of an Owned Collection attribute. A Referenced Collection attribute can also be used to hold collections of objects of the same class as the holder of the collection.

A Referenced Collection attribute can handle the association in the database by using either an embedded foreign key column or a join table. To handle the association using a join table, the referenced collection attribute must be mapped to a join table.

5.2 Implementation Of Object Relationships

It is already mentioned in the previous section that relationships in object schemas can be implemented by a combination of references to objects and operations. This section says how the object relationships are implemented.

When the multiplicity is one (e.g. 0...1 or 1) the relationship is implemented with a reference to an object by the use of a getter operation, and a setter operation. The attribute(s) and operations required to implement a relationship are often referred to as scaffolding.

When the multiplicity is many (e.g. N, 0...*, 1...*) the relationship is implemented via a collection attribute, such as an Array or a HashSet in Java, and operations to manipulate that array.

When a relationship is uni-directional, the code is implemented only by the object that knows about the other object(s). When the relationship is Bi-directional then codes are implemented by both classes.

5.3 Implementation Of Relational Database Relationships

In the earlier sections, it is mentioned that relationships in relational databases are maintained through the use of foreign keys. A foreign key is a data attribute(s) that appears in one table that may be part of or is coincidental with the key of another table.

In the case of a one-to-one relationship, the foreign key needs to be implemented by one of the tables. To implement a one-to-many relationship a foreign key is implemented from the “one table” to the “many tables”. It can be also chosen to overbuild the database schema and implement a one-to-many relationship via an associative table, effectively making it a many-to-many relationship.

Many-to-many relationships can be implemented in two ways in a relational database. The first one is to implement in each table the foreign key column(s) to the other table several times. But unfortunately, with this approach, a problem occurs when more tasks are assigned to a single task. So another better approach is to implement what is called an associative table, which includes the combination of the primary keys of the tables that it associates. In this approach, the many-to-many relationship is converted into two one-to-many relationships, both of which involve the associative table.

A consistent key strategy within the database can simplify the relationship mapping efforts to a greater extent. To make it simpler the first step is to prefer single-column keys and the next step is to use a globally unique surrogate key.

Because foreign keys are used to join tables, all relationships in a relational database are effectively bi-directional. This is why it doesn't matter in which table you implement a one-to-one relationship; the code to join the two tables is virtually the same.

5.4 Relationship Mappings

We have already seen how object relationships and Relational Database relationships are implemented. In this section, we will mention how mapping is done.

We will mention the mappings from the point of view of mapping the object relationships into the relational database. Note that in some cases, we may also choose to design mapping.

A general rule of thumb with relationship mapping is that the multiplicities should be kept the same. Therefore a one-to-one object relationship maps to a one-to-one data relationship, a one-to-many maps to a one-to-many, and a many-to-many maps to a many-to-many relationship. But this doesn't have to be the case always; a one-to-one object relationship can be implemented into a one-to-many or even a many-to-many data relationship. This is because a one-to-one data relationship is a subset of a one-to-many data relationship and a one-to-many relationship is a subset of a many-to-many relationship.

One-to-one mappings: In one-to-one mapping, when an object of a one-to-one object relationship is read into memory then the application will automatically traverse the holds relationship and automatically read in the corresponding object. Or the application will

manually traverse the relationship in the code, taking a lazy read approach (where the other object is read at the time) when required by the application.

One-to-many mappings: In a one-to-many mapping, when an object of a one-to-many relationship is read into memory the relationship is automatically traversed to read in the division that they work in.

Many-to-many mappings: To implement many-to-many relationships one has to know the concept of an associative table, a data entity whose main objective is to maintain the relationship between two or more tables in a relational database.

In relational databases the attributes contained in an associative table are traditionally the combination of the keys in the tables involved in the relationship. The name of an associative table is typically either the combination of the names of the tables that it associates with or the name of the association that it implements.

The rule for mapping is that the multiplicities should "cross over" once the associative table is introduced. A multiplicity of 1 is always introduced on the outside edges of the relationship within the data schema to preserve the overall multiplicity of the original relationship.

Many-to-many relationships are interesting because of the addition of the associative table. But two business classes are being mapped to three data tables to support this relationship. And so this creates another extra work to do.

5.5 Mapping Ordered Collections

Sometimes we keep models in some order with an aggregation association between the two classes. In such a case, we get an ordered constraint placed on the relationship i.e. users care about the order in which items appear on an order. While mapping this to a relational database you need to add an additional column to track this information.

Although this mapping seems straightforward on the surface, there are several issues that are needed to be taken into consideration. These issues become apparent when you consider basic persistence functionality for the aggregate. These issues are (i) Reading the data in the proper sequence

(ii) not to include the sequence number in the key (iii) updating the sequence numbers after rearranging the order items (iv) updating sequence numbers after deleting an order item (v) Considering sequence number gaps greater than one.

5.6 Mapping Recursive Relationships

A recursive relationship, also called reflexive relationship exits when the same entity (class, data entity, table …) is involved with both ends of the relationship.

For example, an employee may manage several other employees. The aggregate relationship that the Team class has with itself is recursive. For e.g. a team may be a part of one or more other teams.

The many-to-many recursive aggregation is mapped to the Subteams associative table in the same way a normal many-to-many relationship is mapped. The only difference is that both columns are foreign keys into the same table. Similarly, the one-to-many association is mapped in the same way a normal one-to-many relationship is mapped.

Self-Assessment Questions - 4

7. Based on directionality, objects relationship can be _____ relationships and_____relationships.
8. In Referenced Relationships (Association) a holder has to use a _____ to reference its associated collection instead of an Owned Collection attribute.
9. While implementing relational database relationships, in a one-to-manyrelationship a _____key is implemented from the "one table" tothe "many table".
10. A general rule of thumb with relationship mapping is that the multiplicities in relationship should be kept_____. (Choose correct option)
    a) same
    b) different
    c) uni-directional
    d) bi-directional

6. MODELING WITH JOIN TABLES

In order to implement many-to-many relationships in the relational model, a Join table is commonly used. We will discuss how join tables can be used to implement many-to-many associations in the object model. A join table is used to maintain a relationship between the rows of two tables (or two different rows in the same table) in a relational database. A join table must contain a foreign key column that points to each of the primary keys of the tables in the association. Additional information may also be maintained in the join table for more clarity.

While modeling many-to-many relationships in the object model, we should always think of a class that represents the join table. If the join table is transactional, then a corresponding persistent class should be created in the object model.

The join is said to be transactional if one or more additional fields such as a quantity or date are defined in a join table. The join is said to be **transparent** If the join only contains foreign key columns.

Transactional Joins

In a transactional join, the join table is modeled by two or more transaction classes. Each transaction class is owned by one of the classes involved in the many-to-many relationship.

Transparent Joins

In Transparent joins, the join table is not modeled by a class. Here each class involved in the relationship has to **get** methods that return its related objects to the other class. This is implemented using a Referenced Collection attribute on each class involved in the relationship.

When two classes are related via a join table, one of the classes must be designated as the **join manager.** The join manager has no functionality in the object model. Only one persistent class can be assigned as the join manager of a join table.

Self-Join Tables

A join does not always have to involve two different tables. You can join a table to itself, creating a self-join. Joining a table to itself can be useful when you want to compare values in a column to other values in the same column.

Self-Assessment Questions - 5

11. The join is said to be of a_____, if one or more additionalfields such as a quantity or date is defined in a join table. (Choose correct option).
    a) transactional
    b) transparent
    c) self-join
    d) join manager
12. When two classes are related via join table, one of the classes must bedesignated as the _____.

7. OPEN SOURCE OBJECT RELATIONAL MAPPING SOFTWARE

There are many open-source object relational mapping software supported by many high-level languages. A few well-known object-relational mapping software are listed below.

LiteSQL: It is an open-source mapper and is a C++ library that integrates C++ objects tightly to a relational database and thus provides an object persistence layer. LiteSQL supports SQLite3, PostgreSQL and MySQL as backends. This library is distributed under the terms of the BSD License.

Java: The open-source mappers using JAVA are given below:

- **Cayenne:** Apache Cayenne is an open-source persistence framework licensed under the Apache License, providing object-relational mapping (ORM) and remoting services. Cayenne seamlessly binds one or more database schemas directly to Java objects, managing atomic commits and rollbacks, SQL generation, joins, sequences, and more.
- **Carbonado:** It is an open-source framework, backed by Berkeley DB or JDBC. It is an extensible, high-performance persistence abstraction layer for Java applications, providing a relational view to the underlying persistence technology. It also supports queries, joins, indexes, and it performs query optimization.
- **Hibernate:** It is open source and provides relational Persistence for Java and .NET. Hibernate facilitates the storage and retrieval of Java domain objects via Object/Relational Mapping.
- **KeyAccess:** KeyAccess is a lightweight Object-Relational Mapping (ORM) tool, that uses existing databases as the starting point to generate a domain model. KeyAccess doesn't mandate a special architecture or framework – it will work in any application using JDBC.

Similarly, the name of other Open-source object-relational mapping software using JAVA are Java Data Objects (JDO), Java Persistence API (JPA), DataNucleus, JPOX, Object Relational Bridge, OpenJPA, ORMLite, QuickDB ORM, Sobat, etc.

.NET: The names of open source mappers using .NET are given below:

- **NHibernate:** It is an open-source port of Hibernate Core for Java to the.NET Framework. It handles persisting plain .NET objects to and from an underlying relational database. Given an XML description of the entities and relationships, NHibernate automatically generates SQL for loading and storing the objects. NHibernate supports transparent persistence and the object classes don't have to follow a restrictive programming model.
- **Castle ActiveRecord:** It is an implementation of the ActiveRecord pattern for .NET. The ActiveRecord pattern consists on instance properties representing a record in the database, instance methods acting on that specific record, and static methods acting on all records.

Similarly, the name of other open source using .NET (some may be with commercial support) mappers are Base One Foundation Component Library, BLToolkit, ECO, Habanero, iBATIS, Neo, nHydrate, Persistor.NET, Picasso, Quick Objects, SubSonic and AgileFx.

PHP: Open source mapper using PHP are:

- **PHP-ActiveRecord:** It is an open-source ORM library based on the ActiveRecord pattern. It aims to massively simplify the interactions with your database and eliminate the chore of handwritten SQL for common operations. Unlike other ORMs, no code generators nor maintenance of mapping files for the tables are required.
- Propel: It is an open-source Object-Relational Mapping (ORM) and Query-Toolkit for PHP 5, inspired by Apache Torque. It allows you to access your database using a set of objects, providing a simple API for storing and retrieving data.
  Name of the other open-source software using PHP is CakePHP, Doctrine, PdoMap, Rocks, Qcodo, and Torpor.

There are other open-source mappers also using languages like Python, Ruby, Perl, Scala, Delphi, VB6, etc. For example, the open-source mappers using Python are Django, SQLObject, Storm, XRecord, Autumn and Tryton ActiveRecord, Datamapper, and iBATIS. The sequel is used in Ruby.

.

Self-Assessment Questions - 6

13. Hibernate is an open-source mapper and provides relational Persistence for Java and .NET. (True/False)
14. Castle ActiveRecord is an implementation of the ActiveRecord pattern for_____.

8. SUMMARY

There is a hot debate among many whether the object-relational impedance mismatch is a problem or not. Few programmers advocate the use of Object-Oriented databases as a solution for impedance mismatch and not ORM. The objective of Object-Relational mapping is to solve the problem of object-relational impedance mismatch.

Classes of Objects can be mapped to the RDBMS table. For e.g. VBSF supports two types of class-to-table mappings that help overcome the differences between relational and object models: **SUBSET and SUPERSET**. A class inheritance tree can be mapped in the RDBMS as: **vertical** mapping, **horizontal** mapping, and **filtered** mapping.

Object relationships for mapping to the relational model can be categorized based on multiplicity, directionality, ownership, and reference.

Based on the multiplicity, object relationships can be **One-to-one relationships, One-to-many relationships, and Many-to-many relationships.** Based on directionality, objects' relationships can be unidirectional relationships and bi-directional relationships.

A general rule of thumb with relationship mapping is that the multiplicities should be kept the same.

The join is said to be **transactional** if one or more additional fields such as a quantity or date is defined in a join table. The join is said to be **transparent**. If the join only contains foreign key columns.

There are many open-source object relations mapping software supported by C++, JAVA, Python, Ruby, Perl, Scala, Delphi, VB6, etc.

9. TERMINAL QUESTIONS

1. What is object-relational impedance mismatch?
2. Explain vertical mapping, horizontal mapping, and filtered mapping.
3. What is the difference between owned relationship and referenced relationship?
4. What are the different open source mappers?

10. ANSWERS

Self-Assessment Questions

1. database tables or relational schema
2. impedance mismatch
3. False
4. multiple tables
5. different
6. True
7. uni-directional, bi-directional
8. Referenced Collection attribute
9. Foreign
10. a) same
11. a) transactional
12. join manager
13. True
14. .NET

Terminal Questions

1. During Object-relational mapping when an RDBMS is being used by a program written in an object-oriented programming language or style then any changes in database schema will result in unexpected failure of system. This type of phenomenon is known as an impedance mismatch. (Refer to 2 for detail)
2. In vertical mapping, each class in the tree, whether abstract or concrete, is mapped to a different table. In horizontal mapping, each concrete class in the tree is mapped to a different table. In filtered mapping, all concrete classes in the tree are mapped to the same table. (Refer to 4 for detail)
3. In an owned relationship, the owner class holds a reference to its owned collection using an Owned Collection attribute and the owned class holds a direct reference to its owner using a Reference attribute. Inreferenced relationship an object on the one side (also referred to as the holder of the collection) has to use a Referenced Collection attribute to reference its associated collection instead of an Owned Collection attribute.
4. Few of the open-source mappers are Cayenne, Hibernate, Castle ActiveRecord, PHP-ActiveRecord, etc. (Refer section 7 for detail)