
Introduction to ObjectSpark[®] Mapping

DataTern, Inc.
330 Madison Ave 31st Floor
New York, NY 10017
(212) 210-6221
www.datatern.com



© 2009 DataTern, Inc.

The information in this Document is the intellectual property of DataTern, Inc..

This Document may not be reproduced in whole or in part, by any means, without the written consent of DataTern, Inc.. All Rights Reserved.

The Software described in this Document and all copies of the Software are the property of DataTern, Inc.. The Software is provided under a license agreement containing restrictions on use and disclosure. The software contains trade secrets of DataTern, Inc.. Reverse engineering of the Software is prohibited.

If provided to the U.S. Government, this Software and this Document are provided with Restricted Rights. Use, duplications, or disclosure by the Government is subject to restrictions set forth in FAR 52-227-19 (c) (2) (June 1987) or DOD FAR Supplement 252.227-7013 (c) (1) (ii) (June 1988) or the NASA FAR Supplement as applicable. Contractor/Manufacturer is DataTern, Inc..

Trademarks

ObjectSpark Software is a registered trademark and ObjectSpark[®], the ObjectSpark[®] logo, and the DataTern, Inc. logo are trademarks of DataTern, Inc..

Microsoft and Visual Studio are registered trademarks and Windows, Windows NT, Visual Basic, Visual C++, SQL Server, and Microsoft Transaction Server are trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

IBM and DB2 are trademarks of International Business Machines Corporation.

Rational Rose is a trademark or registered trademark of Rational Software, Inc., in the United States and/or other countries.

Sun, Sun Microsystems, the Sun logo and JavaBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All other product and company names mentioned in this document are trademarks or registered trademarks of their respective companies or organizations.

DataTern, Inc.
330 Madison Ave 31st Floor
New York, NY - 10017
(212) 210-6221
www.datatern.com

Introduction to ObjectSpark[®] Mapping

For relational data sources, ObjectSpark[®] defines the mapping between the object model and the data store in two phases:

Defining *where* the object state is stored – a set of base mappings that establish where the values that represent the state of each object are stored.

Defining *how* the object state is stored – a set of operations that execute the appropriate commands on the back end in order to store and retrieve the state of each object and manipulate object relationships.

The ObjectSpark[®] Designer uses the information defined in the base mappings to automate the definition of operations.

Base Mapping: Defining Where State is Stored

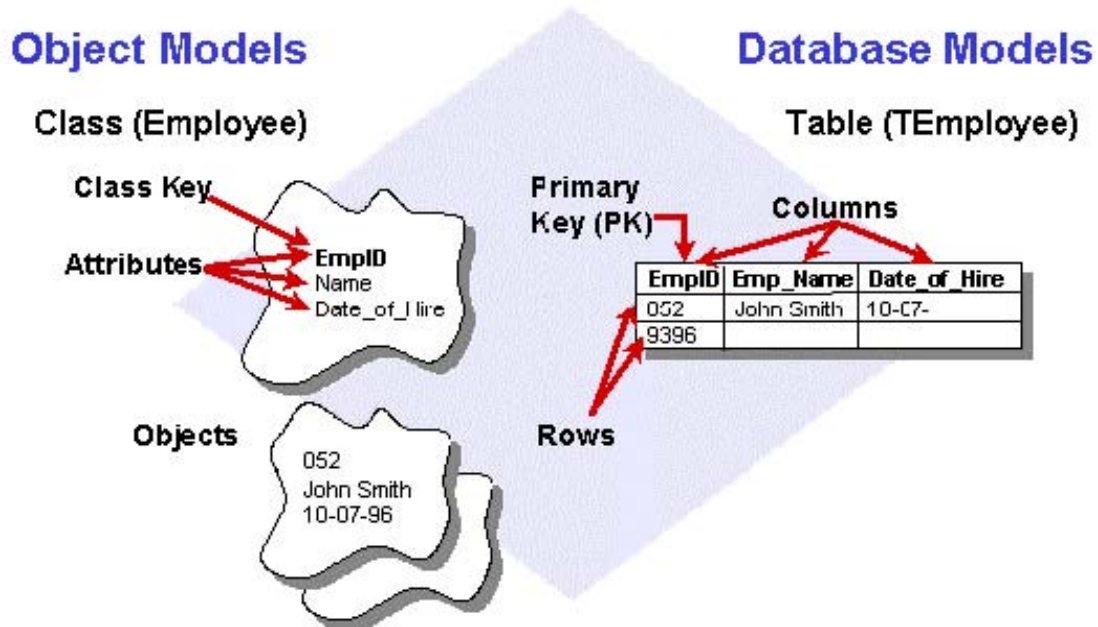
Base mapping information is initially defined at the class level, to create mappings between the scalar attributes of each class and their data sources. Then, on each of the related class attributes, you create an association between the object relationship and the corresponding relationship in the data source.

ObjectSpark[®] automates as much of the mapping process as possible.

How Class Mapping Works

ObjectSpark[®] allows you to map classes to data sources by creating mappings between class.attributes and a variety of attribute data sources. Typically, the data source for an attribute is a table.column. However, you can also map attributes to constants, named variables, or stored procedure parameters.

In the simplest case, a class is mapped to a single table that has a column corresponding to each of the attributes of the class. Each row of the table stores the data for an object that is a member of the class.



However, a simple one-to-one mapping between the classes in the object model and the tables in the database model rarely exists. Databases have denormalized tables or redundant data values stored in multiple tables. In some cases, the state of an object may be stored in multiple tables, or even multiple databases. Object models can define complex object relationships and class hierarchies using inheritance. ObjectSpark® provides an approach to mapping that provides the flexibility needed to bridge the structural differences that often exist between object models and database models. (See the "Class Mapping Patterns" white paper for an overview of the different ways in which you can define class mappings.)

Where Relationships are Stored

To map relationships, the first thing you need to understand is how relationships are persisted in your data store.

In most cases, your object relationships are stored in a relational database. Relational databases store relationships in the form of foreign key definitions. A foreign key consists of one or more columns that hold a value related to the primary key value of a row in another table. Your object relationships are persisted by these foreign key values. Many-to-many relationships are stored as rows in a join table. A join table has a multi-column primary key where each column of the primary key is also a foreign key to one of the tables to be joined. If a foreign key definition does not exist to support a given object relationship, then you will need to identify one or more columns that logically serve the same purpose as a foreign key for storing your object relationships. In the case of a many-to-many relationship, you would need to create a join table.

If you look at the cardinality of the relationships in your object model, you can put each relationship in one of the following categories:

One-to-One – where max. cardinality = 1 on both sides of the relationship

One-to-Many – where max. cardinality = 1 on one side and * (many) on the other side

Many-to-Many – where max. cardinality = * (many) on both sides of the relationship

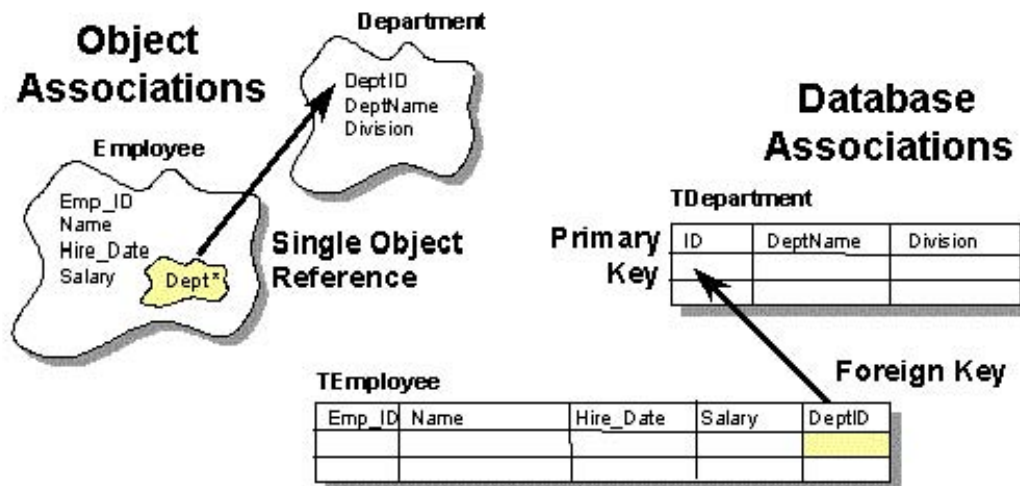


Object relationships are often further differentiated by their relationship types (association, aggregation or composition). In mapping relationships, the relationship type definition mostly affects how relationship operations are defined, which is discussed later in this document.

Relationship mappings are based on joins that you define between the tables to which the related classes are mapped. In some cases, you may need to define a multi-way join. The joins are defined using the foreign key that stores the relationship. If you can determine how a relationship is stored, then you can define the join that defines the relationship mapping.

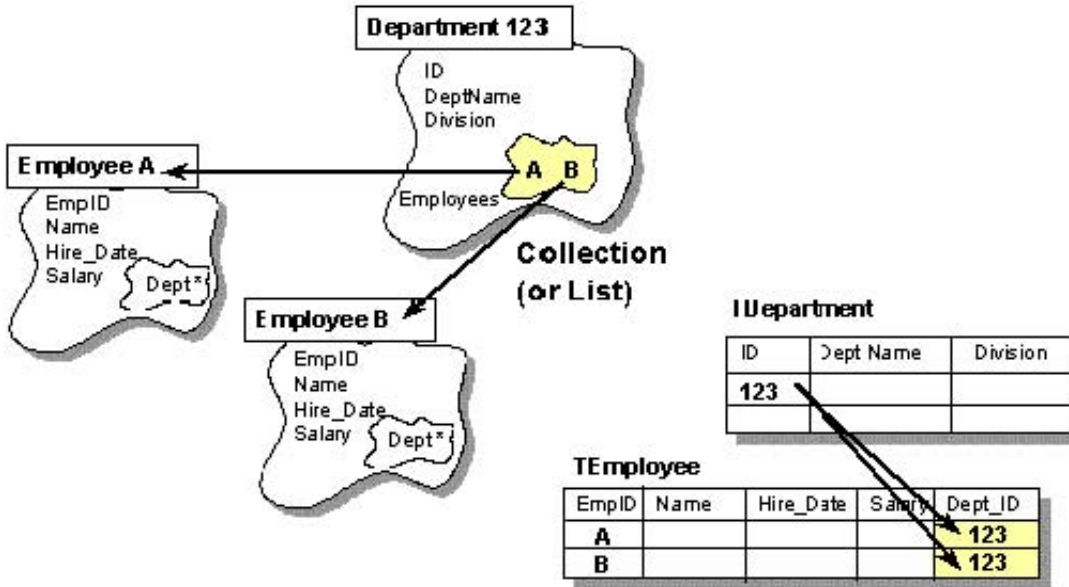
One-to-One Relationships

A one-to-one relationship is stored as a value in the foreign key column that is either in the table to which the relationship owner is mapped or in the table to which the related object is mapped. The relationship mapping is based on a join that you define between the foreign key that stores the relationship and the primary key of the related table.



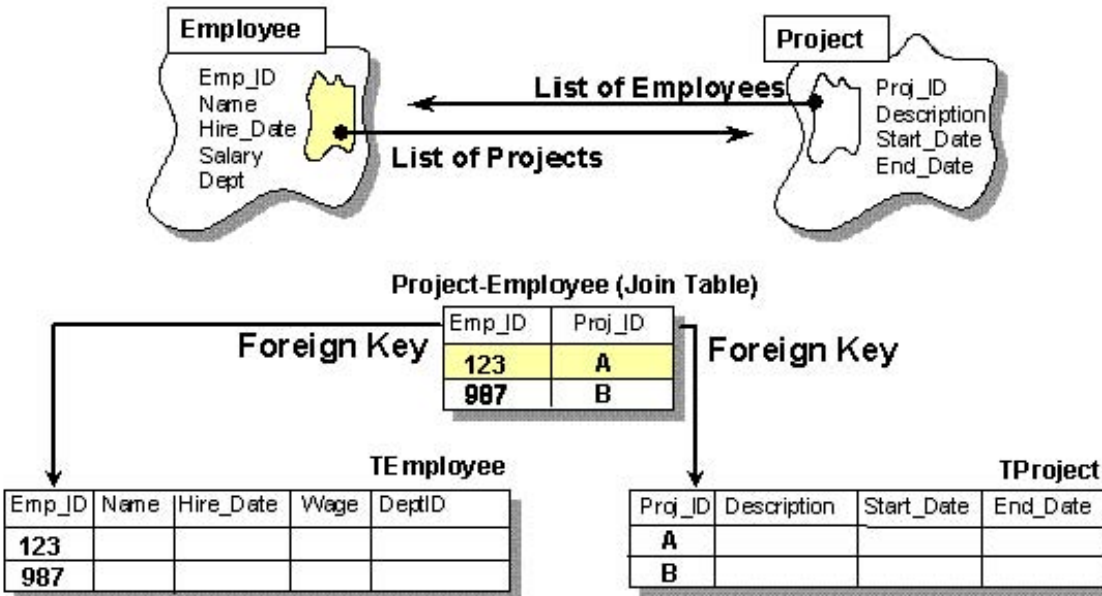
One-to-Many Relationships

A one-to-many relationship is typically stored as the value in the foreign key column in the table to which the related class is mapped. This allows many related object rows to reference the row that is mapped to the object that holds the "many". The foreign key points to the primary key of the table to which the object that owns the relationship is mapped. Zero or more rows in the related object table may hold values that point to the row to which the relationship owner is mapped. The relationship mapping for a one-to-many relationship is based on a join that you define between the foreign key of the table that stores the "many" and the primary key of the table that stores the "one".



Many-to-Many Relationships

A many-to-many relationship can only be stored by a join table in which each row stores an instance of a relationship between two objects. The relationship mapping for a many-to-many relationship is based on a two-way join that you define, first from the relationship owner's table to the join table, then from the join table to the related object's table.



Operations: Defining How State is Stored

In an ObjectSpark® map, operations define the persistence behavior of your components. Operations are composed of one or more commands. The ObjectSpark® data services layer executes these commands on the back end to manipulate the data that represents the persistent state of the objects that are used by your application.

There are two basic types of operations:

Class Operations

Read, Read List, Add, Update, Remove

Typically operate on rows to persist the state of scalar attributes

Relationship Operations

Read Relationship, Add Relationship, Remove Relationship, Remove All Relationships

Typically operate on foreign key values to persist the state of relationships

Operations are defined as a series of SQL commands (SELECT, INSERT, UPDATE, DELETE, and stored procedure calls). The SQL commands define any additional steps that need to be taken (besides writing or deleting mapped attribute data) whenever the method that calls the operation is invoked. In addition to editing the commands themselves, you can modify the order of the SQL commands.

ObjectSpark® provides a graphical interface for defining commands and gives you the option to display a textual representation of each command.

Class Operations

Class operations define commands that are executed at the class level when an object is stored, retrieved, updated, or removed. These operations primarily serve to persist the scalar attributes of a class. (The relationship operations handle the values held by related class attributes.)

You can define up to five different operations on a class:

Read – defines the commands that are executed when a single instance of the class is retrieved. The operation assumes that the object exists in the data source.

Read List – defines the commands that are executed when members of the class are retrieved as a standalone list. This operation effectively defines the membership criteria for the class.

Add – defines the commands that are executed when a new object is saved for the first time as a member of the class. The operation assumes that the new object has been initialized in memory.

Update – defines the commands that are executed when changes to an existing member of the class is saved. The operation assumes that the object was previously retrieved from the data source.

Remove – defines the commands that are executed when a single object is removed. The operation assumes that the object exists in the data source.

Adding a *new* object typically means inserting a new row in the table to persist its state. When an existing object is retrieved (or read), its attribute values are read from the row that persists it, which is identified by a key value. When changes that have been made to the object are saved, the row is updated. Removing the object means deleting the row.

Relationship Operations

Relationship operations define commands that create or remove relationships between objects. They primarily operate on the foreign key values that persist the relationships held by the related class attributes.

You can define up to four different relationship operations on a related class attribute:

Read Relationship – defines the commands that are executed either to populate a single related object or to populate a related list, depending on the cardinality of the relationship. In the case of a related list, this operation effectively defines the membership of the related list.

Add Relationship – defines the commands that are executed to create a relationship between the object that owns the relationship and another object that is a member of the related class.

Remove Relationship – defines the commands that are executed when the relationship between the object that owns the relationship and a related object is removed. (The definition of this particular relationship operation is highly dependent on the relationship type.)

Remove All Relationships – defines the commands that are executed when the object that owns the relationship is removed. The operation can only be defined on relationships where the cardinality is greater than one.

Relationship operations are defined differently, depending on the cardinality of the relationship.

One-to-One Relationships

To create or change the relationship, you would update the value of the foreign key. To remove the relationship, you would update the value of the foreign key to null (assuming the key is nullable), or perform some other update that has the same meaning. For example, to set the value of an employee's relationship to a particular department, you would update the employee's deptID foreign key field to hold the value of the related department's ID.

One-to-Many Relationships

To create or change the relationship, you would update the value of the foreign key in the related object table. To remove the relationship, you would update the value of the foreign key to null (assuming the key is nullable), or perform some other update that has the same meaning. For example, a department sees employees in a one-to-many relationship. To add an employee to the list of employees that are already related to a given department, you would put the value of the department's ID into the deptID foreign key field of the employee you are adding.

Many-to-Many Relationships

To create a many-to-many relationship, you insert a row in the join table that points to each of the related object records. To remove a many-to-many relationship, you delete the row that stores the relationship. For example, to add a project to an Employee's list of projects, you would insert a row in the join table that pairs up the employee ID with the related project ID. To remove the relationship, you delete that row.

How Mapping Information is Used at Run Time

Operations are called in response to requests that are called on data components at run time. Operations can be comprised of one or more commands (INSERT, UPDATE, etc.).

Run-time Request	API Call	Operation Called
Retrieve the state of a single member of the class	Find , or Search (on single object)	Read
Retrieve the state of all members of the class	RetrieveData (on standalone list)	Read List
Save the state of a new member of the class	Save (on single object when isNew = True)	Add
Save changes to the state of an existing member of the class	Save (on single object when isNew = False)	Update
Remove the state of a member of the class	Remove (on single object)	Remove
Retrieve the related object or list of related objects held by this related class attribute	Set obj = attribute value (on single object, returns related list)	Read Relationship
Create a relationship between this object and the owner of this related class attribute	Set attribute value = obj (on single object) Add (on related list)	Add Relationship
Remove the relationship between this object and the owner of this related class attribute	Set attribute value = obj (on single object) Remove (on related list)	Remove Relationship
Remove all relationships to other objects that are held by this related class attribute that need to be removed before removing this object	Remove (on single object)	Remove All Relationships (on each related class attribute that holds a list)

In addition to commands that execute the primary intent of the operation, operations may include other commands that you want to have executed in the context of a particular operation. For example, whenever a new object is stored, you may want to insert a record in a log table. You could also use extra commands in an Update operation to update redundant data values.

You can choose which operations you define, depending on the requirements of your application. The existence of an operation determines whether the corresponding API method is supported by your components. For example, if a class Remove operation is not defined, the component will return an error when the Remove method is called at run time.

To define an operation, you define the set of commands that you want to have executed on the back end when the given operation is called. The Operations Editor in the ObjectSpark® Designer provides a graphical interface for defining the commands and editing the command sequence. The ObjectSpark® Designer provides support for displaying the SQL for any command so you can check whether the operation is defined correctly.



Whenever you are defining operations, keep in mind that, at run time, the operation is always executed in the context of a particular object. The object provides the actual values used in the commands defined by the operation. The attribute names defined in the command just indicate where the values come from.

For classes that are mapped to relational data sources, the ObjectSpark® Designer generates default operations, which you can then edit to change or extend the behavior. For example, you may want to change a generated Remove operation to execute a stored procedure instead of a DELETE command. Operations may include other commands that you want to have executed in the context of a particular operation. For example, whenever a new object is stored, you may want to add another command to insert a record in a log table.