Object-relational mapping

From Wikipedia, the free encyclopedia

Not to be confused with Object-Role Modeling.

Object-relational mapping (**ORM**, **O/RM**, and **O/R mapping**) in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

Contents

Overview

In object-oriented programming, data management tasks act on object-oriented (OO) objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as structured query language database management systems (SQL DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping is used to implement the first approach.[1]

The heart of the problem is translating the logical representation of the objects into an atomized form that is capable of being stored in the database, while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent.[1]

Simple Explanation

[2]A simple answer is that you wrap your tables or stored procedures in classes in your programming language, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

In other words, instead of something like this:

```
String sql = "SELECT ... FROM persons WHERE id = 10"
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

you do something like this:

```
Person p = repository.GetPerson(10);
String name = p.FirstName;
```

or similar code (lots of variations here.) Some frameworks also put a lot of the code in as static methods on the classes themselves, which means you could do something like this instead:

```
Person p = Person.Get(10);
```

Some also implement complex query systems, so you could do this:

```
Person p = Person.Get(Person.Properties.Id == 10);
```

The framework is what makes this code possible.

Now, benefits. First of all, you hide the SQL away from your logic code. This has the benefit of allowing you to more easily support more database engines. For instance, MS SQL Server and Oracle has different names on typical functions, and different ways to do calculations with dates, so a query to "get me all persons edited the last 24 hours" might entail different SQL syntax just for those two database engines. This difference can be put away from your logic code.

Additionally, you can focus on writing the logic, instead of getting all the SQL right. The code will typically be more readable as well, since it doesn't contain all the "plumbing" necessary to talk to the database.

Comparison with traditional data access techniques

Compared to traditional techniques of exchange between an object-oriented language and a relational database, ORM often reduces the amount of code that needs to be written.[3]

Disadvantages of O/R mapping tools generally stem from the high level of abstraction obscuring what is actually happening in the implementation code. Also, heavy reliance on ORM software has been cited as a major factor in producing poorly designed databases.[4]

NoSQL (Not Only SQL) databases

Another approach is to use an object-oriented database management system (OODBMS) or document-oriented databases such as native XML databases that provide more flexibility in data modeling. OODBMSs are databases designed specifically for working with object-oriented values. Using an OODBMS eliminates the need for converting data to and from its SQL form, as the data is stored in its original object representation and relationships are directly represented, rather than requiring join tables/operations.

Document oriented databases also prevent the user from having to "shred" objects into table rows. Many of these systems also support the XQuery query language to retrieve datasets.

Object-oriented databases tend to be used in complex, niche applications. One of the arguments against using an OODBMS is that switching from an SQL DBMS to a purely object-oriented DBMS means that you may lose the capability to create application independent queries for retrieving ad hoc combinations of data without restriction to access path. For this reason, many programmers find themselves more at home with an object-SQL mapping system, even though most object-oriented databases are able to process SQL queries to a limited extent. Other OODBMS (such as RavenDB[5]) provide replication to SQL databases, as a means of addressing the need for ad hoc queries, while preserving well-known query patterns.

Challenges

There are a variety of difficulties that arise when considering how to match an object system to a relational database. These difficulties are referred to as the object-relational impedance mismatch.

An alternative to implementing ORM is use of the native procedural languages provided with every major database on the market. These can be called from the client using SQL statements. The Data Access Object (DAO) design pattern is used to abstract these statements and offer a lightweight object-oriented interface to the rest of the application.[6]

See also

- List of object-relational mapping software
- Comparison of object-relational mapping software
- AutoFetch - automatic query tuning
- CORBA
- Object database
- Object persistence
- Object-relational database
- Object-relational impedance mismatch

- Relational model
- Java Data Objects
- Service Data Objects
- Entity Framework
- Active record pattern

References

1. ^ *a b* "What is Object/Relational Mapping?" (http://www.hibernate.org/about/orm). *Hibernate Overview*. JBOSS Hibernate. Retrieved 19 April 2011.
2. ^ "Stackoverflow answers" (http://stackoverflow.com/questions/1152299/what-is-an-object-relational-mapping-framework). *Stackoverflow*. Lasse V. Karlsen.
3. ^ Douglas Barry, Torsten Stanienda, "Solving the Java Object Storage Problem," Computer, vol. 31, no. 11, pp. 33-40, Nov. 1998, http://www2.computer.org/portal/web/csdl/doi/10.1109/2.730734, Excerpt at http://www.service-architecture.com/object-relational-mapping/articles/transparent_persistence_vs_jdbc_call-level_interface.html. Lines of code using O/R are only a fraction of those needed for a call-level interface (1:4). *For this exercise, 496 lines of code were needed using the ODMG Java Binding compared to 1,923 lines of code using JDBC.*
4. ^ Josh Berkus, "Wrecking Your Database", Computer, Aug. 2009, http://it.toolbox.com/blogs/database-soup/wrecking-your-database-33298, Webcast at http://www.youtube.com/watch?v=uFLRc6y_O3s
5. ^ "Index Replication" (http://ravendb.net/docs/server/bundles/index-replication). *Index Replication Bundle*. Hibernating Rhinos.
6. ^ Feuerstein, Steven; Bill Pribyl (September 1997). "Oracle PL/SQL Programming" (http://docstore.mik.ua/orelly/oracle/prog2/ch18_05.htm). 18.5 Modifying Persistent Objects. Retrieved 23 August 2011.

External links

- About ORM (http://www.artima.com/intv/abstract3.html) by Anders Hejlsberg
- Mapping Objects to Relational Databases: O/R Mapping In Detail (http://www.agiledata.org/essays/mappingObjects.html) by Scott W. Ambler
- Core J2EE Design Pattern: Data Access Objects (http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html)
- Choosing an Object-Relational mapping tool (http://madgeek.com/Articles/ORMapping/EN/mapping.htm)
- Object-Relational Mapping Concepts (http://lalit-bhatt.blogspot.in/2014/07/object-relationship-mapping-orm.html)