

Re: object databases

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2006-11/msg00079.html>

- *From:* frebe73@xxxxxxxxxx
 - *Date:* 4 Nov 2006 03:44:28 -0800
-

Well, I never understood the so called "impedance mismatch" problem. Calling SQL from say java code is one of the easiest tasks in modern programming. Getting results back is straightforward as well. What I also never has been able to grasp is why would you discard this simple idea lightly in favor of learning obscure 20 something mapping rules (TopLink).

that's simple.

take the following small code snippet.

```
using(DataAccessAdapter adapter = new DataAccessAdapter())
{
    adapter.SaveEntity(myCustomer);
}
```

this line of code saves:

- the Customer entity instance myCustomer
- its related Order entities
- its related order detail entities
- and other related entities to these.

all in the right order, it syncs pks with fk's, it pulls back the new PK's from the db if the pk's are identity/sequenced, performs pre-/post save validation etc. etc. etc.

effectively 2 lines of code.

Now, tell me, you can do all that with 2 lines of code with SQL strings etc. ? No, you can't.

But what about DataAccessAdapter? If it is a generic framework class I guess that you have to supply some O/R mapping descriptions that is longer than 2 lines. And there are numerous known problems with O/R mapping that limit the way you can use a RDBMS. If DataAccessAdapter is a custom class I guess you have a lot of SQL code inside it. And you didn't provide the Customer, Order and OrderDetail classes. Using a SQL-based approach, these classes are not necessary.

Re: object databases

Another thing: it's type save.

If you embedded SQL like SQLJ you will also get type safety.

```
CustomerEntity c = new CustomerEntity();
c.CompanyName = "MyCompany Inc.";
c.ContactPerson = "Joe Smith";
//.. save the entity, using the mechanism of the o/r mapper, e.g.:
c.Save();
```

insert into company (name, contactperson) values ('MyCompanyInc','Joe Smith')

My SQL-based approach did it in only one line.

If something changes in the db, e.g. ContactPerson becomes Contact, it still works, OR your code breaks at COMPILE TIME, while your sql string misery breaks at runtime and turns into an unmaintainable nightmare.

Use embedded SQL.

```
Take the example above even further:
OrderEntity o = new OrderEntity();
o.OrderDate = DateTime.Now;
OrderDetailEntity od = new OrderDetailEntity();
od.Price = 10.0f;
od.Quantity = 1;
od.ProductId = 10;
o.OrderDetails.Add(od);
CustomerEntity c = new CustomerEntity();
c.CompanyName = "MyCompany Inc.";
c.ContactPerson = "Joe Smith";
o.Customer = c;
```

Let's save the order, as that's what we're interested in:
o.Save();

```
// or:
using(DataAccessAdapter adapter = new DataAccessAdapter())
{
    adapter.SaveEntity(o);
}
```

insert into customer (companyname, contactperson) values ('MyCompany

Re: object databases

```
Inc', 'Joe Smith')
customerid = insert_id();
insert into order (orderdate, customerid) values (now(), customerid);
orderid = insert_id();
insert into orderdetail (orderid, price, quantity, productid) values
(orderid, 10, 1, 10);
```

Your solution is still more verbose than a SQL-based approach.

The pk's have to be synced with the fk's relying on them, so when customer is saved, the pk it got of the sequence has to be synced with the fk in order, which then has to be saved and after that the order details, which receives the pk of order.

This problem only exists when you use domain objects. A SQL-based solution has no problems with this.

All I did was write some typed, compile time checked code which is simple to follow, easy to update and maintain (!) and everything is taken care of.

My code compile time checked, simple to follow easy to update and maintain.

Can you do that easily with the same # of code I wrote? I doubt it,

Yes, but with less # of code.

I.o.w.: you're doing a lot of plumbing code which comes for free with an o/r mapper. That's the advantage of using a persistence layer which solves the impedance mismatch as it abstracts away the relational aspects of the persistent storage.

If you try to map classes to tables, you have an impedance mismatch. But if you map classes to datatypes (date, SSN, etc) the relational model and OO lives happy together.

Fredrik Bertilsson
<http://frebe.php0h.com>

.

Re: object databases