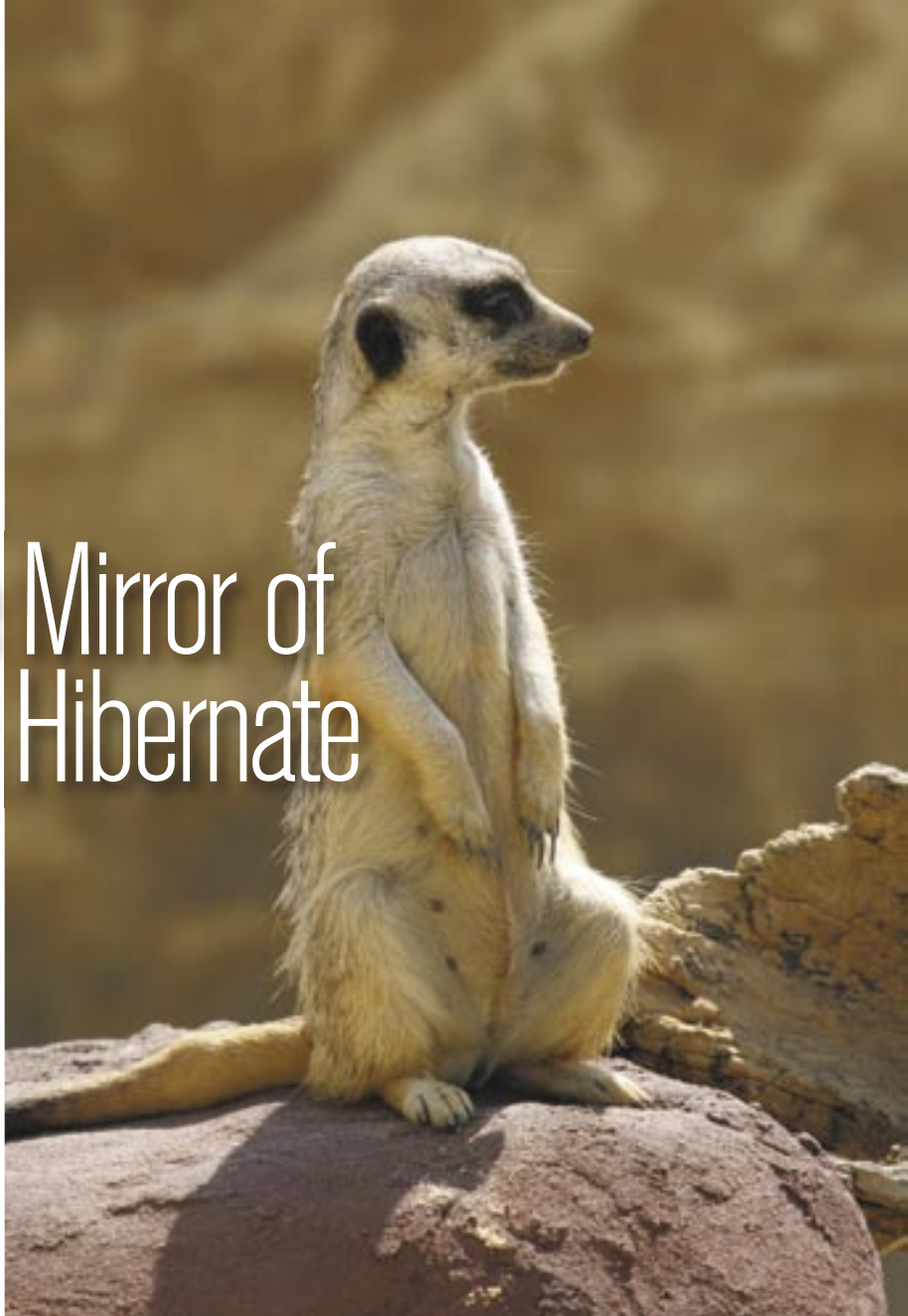


# db4o in the Mirror of JPA/EJB and Hibernate

## When or why should I use an object database?

By Stefan Edlich and Daniel Oltmanns



### About the Authors

Stefan Edlich is professor of software engineering at the Technical University of Applied Sciences in Berlin. The focus of his research includes enterprise applications, object databases, and mobile applications. The results of his research have produced many books on open source, Web frameworks, J2EE, and db4o. Stefan is organizing the object database conference ICOODB.org 2008.

Daniel Oltmanns is a research assistant at the Technical University of Applied Sciences in Berlin and an expert in the field of AJAX-technologies, databases, and Spring-architectures

sedlich@tfh-berlin.de

oltmanns@tfh-berlin.de

db4o, an open source object database system with broad industry applicability, belongs to a popular database management systems that has close to 2 million downloads to date. Here we'll illustrate the features and application areas of such a database and compare db4o against relational DBMS/object relational (OR) mappers.

It seems as if object databases are enjoying a rebirth. After the disastrous but probably quite purifying hype of the '90s, many providers now offer sophisticated products. Besides db4o, the products by Intersystems, Objectivity, Gemstone, and Versant deserve to be mentioned. At this point the most important question is: When or why should I use an object database?

There are many studies available that compare the benefits of object database management systems (ODBMS) and relational database management systems (RDBMS):

- Because of their independent representation, relational database systems are ideally suited to various access modes. Besides being simple to create reports on relational data, the management of large amounts of data is very fast.

Nevertheless a number of disadvantages caused by the impedance mismatch occur when mapping objects to a relational database. These disadvantages create opportunities for object database systems:

- Object databases enable easier persistence code in the context of complex object structures. Object navigation and the elimination of "joins" are an advantage in some cases.

Every advantage generally involves a disadvantage. This applies to relational database systems with their OR mappers as well as to object database systems. Specifically, the mapping of deep and complex object

structures with OR mappers isn't easy and can be a challenge to maintain. On the other hand the administration and reporting of data structures in ODBMS is not as sophisticated. We'll go into that later.

For these reasons there have been many discussions lately about which is the ideal solution. Gavin King is defending the relational world against all challengers with his impressive flagship product Hibernate. In May 2007, King published a long blog post called "In Defence of the RDBMS," portraying solutions such as Active Records from Ruby on Rails or object database systems as unreliable.

Famous .NET and Java developers like Ted Neward argue against this point because it's obvious that there's absolutely no need for the relational world and its ORM solutions to defend themselves since they completely dominate the enterprise world. Hence Ted takes a "relatively" neutral position arguing that there's always a "right solution for the right problem" and that the world of persistence could use a little more versatility than just RDBMS (e.g., Oracle or MySQL) + Hibernate.

But there must be a reason for the attacks on the ORDBMS world. So let's take a look at the application areas.

One of the first things to do before choosing a database system should always be an analysis of the data format required. In doing so, the following questions will arise:

1. Do I need the data in class-independent or relational format? Do I have to access the data via different views, i.e., different classes (e.g., also in different languages)? Do I need lots of OLAP-style reports? Are there many departments with different domain models accessing the data?
2. Do I need cluster architectures out-of-the-box and a stringent and secure administration of the database system?

These are some of the core questions that every project leader should ask himself. If most of the answers are "yes," it's obvious that a high-performing relational database system is the only feasible choice.

It's a very different story if these questions can't be answered "yes." In many cases persistence is only one aspect and not a primary requirement of the application. This means that the requirements regarding persistence aren't too high. Many of the LAMP or Ruby-on-Rails applications are developed as "single-user" applications. Many applications deploy one single (e.g., Java or C#) domain model, where the database doesn't need an administrator and data access is nowhere near the range of a big portal. In this case one can consider an object database system.

Another example is embedded applications. All elements of both questions 1 and 2 can clearly be answered "no." It's here where db4o becomes relevant.

## db4o and the Competition

db4objects was founded in 2004 and achieved close to 2 million downloads and 30,000 registered users in just two years. The database system isn't geared for big enterprises or big portals with highly scalable multithreaded access. db4o is a lot more successful in BMW cars, Seagate hard disks, Ricoh office equipment, and Bosch robots. But the application area is much broader than just devices.

We'll compare db4o's features with those of OR-Mappers + RDBMS (e.g., Hibernate or JPA/EJB 3 with MySQL). This is definitely a comparison of apples and trumpets, though readers who are familiar with EJB/Hibernate might be able to assess the ODBMS features better.

## System Environment

db4o itself is just a small Java JAR or MS.NET DLL library of about 600K. And it only requires a few system resources. Usually a few megabytes of RAM are sufficient to work with db4o.

db4o stores Java as well as .NET objects, so any .NET-compatible language such as C# or Visual Basic can be used.

A db4o system can't be administered externally. Administration is only possible via program code, which has many disadvantages but also a few benefits: The system can be installed quickly. From download to integration into the classpath to storing objects, db4o needs less than two minutes.

Another interesting feature is that db4o is very strong in mixed environments. For example, applications written in Java and C# can access the same database. The Java version also enables Unix versions of db4o.

Object database systems are often criticized for not having a data-independent schema because the object class, in essence, is the database schema. This would make class-independent data processing impossible. In db4o this is bypassed by the concept of aliases. Using package and class aliases even names can diverge. Because class methods can differ and are ignored by database systems, it's no problem using similar, if unequal classes, e.g., in Java or C# to access the database.

## The API

Listing 1 is a simple example illustrating how to create, store, modify, and finally delete a Customer object. Developers familiar with EJB 3/JPA (the Java Persistence API) or Hibernate shouldn't have a problem adapting.

In db4o, an object container is initially created. This is an object analog to the JPA/Entity Manager responsible for object administration.

The JPA/Entity Manager, however, gets its environment from a configuration of diverse factories and, if applicable, even from the Spring Bean repository. In db4o, only one file or server (via TCP/IP) has to be named.

In the first case, db4o uses just one proprietary file. This makes db4o the ideal choice for mobile devices. But the exchange of db4o database files, for example, by e-mail, is also a benefit that many companies appreciate even though the data isn't available in an independent format.

With the ObjectContainer, the user has implicitly opened a transaction. That's why there's no need for a call like `entityManager.getTransaction()` in Hibernate or JPA.

Taking the notion of CRUD functionality, we notice many differences but also similarities.

- `db.set()` stores an object. In JPA this equals `em.persist(...)`. In an ActiveRecord environment like Ruby on Rails, creating an object would be sufficient to persist it.
- `db.delete()` deletes an object, analogous to `em.remove()` in JPA.
- In JPA, one would update modified objects with `em.merge(...)`. In db4o this is done simply by modifying and storing the object with `set()`. In this case it's necessary for the object to be found via a previous query; a "weak reference" between the object and the ObjectContainer will have been established. So an update command isn't necessary. In JPA, the object just needs to be attached, which can be done with `em.refresh()`.

The commands (not, however, the ideas and approaches) for retrieving objects also differ slightly and will be examined later on.

### Object Management

Without a doubt the biggest benefit of object databases such as db4o is that all objects can be stored just the way they are.

The Customer object stored in Listing 1 could just as easily be extremely complex:

- It could be at the end of a long inheritance hierarchy
- It could contain many sub-objects that in turn can be as deep as required

In industry applications, object depths of 10 to 20 aren't unusual. In object databases, all objects can be stored instantly just the way they are. All objects can be stored type-independent (type-orthogonal). Complex collections and the deep object structures mentioned above aren't a problem at all. The code or the domain objects don't need modification. There's no need for "implements," bytecode enhancements, or annotations.

Using OR-mappers is a lot more complex. The mapping of deep objects and especially the management of schema changes is much more time-consuming. Nevertheless JPA or Hibernate 3.2 already have a strong implementation and are well on their

way to improving their ease of use. The annotation of classes with Java annotations (e.g., `@Entity`) strongly reduces time and effort. This used to be very difficult in big projects with deep XML classes.

Only the notation of JPA relations (`@one-to-one`, `@one-to-many`, `@many-to-many`), the bi-directionality, and the special cases (e.g., the database differs from the object schema) are still a little annoying.

When storing, updating, and loading objects in db4o or JPA/EJB/Hibernate, one previously had to consider similar aspects: Up to what object depth are objects to be stored, updated, or activated? What is the default depth? In db4o all this can be adjusted with just a single line of code. Sub-objects that aren't active can also be activated with `activate()`.

In versions 6.2 and 6.3, db4o introduced transparent activation (TA). With TA, objects are only retrieved when requested. Initially sub-objects aren't retrieved. In case these sub-objects are required, db4o will automatically activate and retrieve them.

This isn't as fast as manual activation but in exchange the application developer doesn't need to worry about update levels.

### Transactions

As shown in Listing 1, the ObjectContainer implicitly starts a transaction. This transaction can be completed or undone with `commit()` or `rollback()`. Actions in the transaction or single commands are always ACID in db4o. In db4o this is achieved by quasi-serialising all operations. As a result db4o is extremely transaction-safe but on the other hand inappropriate for multithreaded access. A Web application with dozens of database queries a second isn't as easily realizable as other replicated and highly scalable database architectures (e.g., as in Oracle/DB2) even if single-threaded access is extremely fast.

db4o has only implemented read-committed access mode. Switching to a different isolation level isn't possible.

### Query Languages

The most interesting analysis might be the comparison of query languages. Even here db4o, EJB, JPA, and Hibernate resemble each other more than one might think.

Depending on db4o's implementation, queries are either executed in the application itself or in the db4o server instance.

Let's go through the requirements step-by-step:

### Simple Queries

Here, many of the popular persistence solutions provide finders and Query-by-Example (QBE). JPA or ActiveRecord in Ruby on Rails provide to find objects, for example, via their primary key. Hibernate, like db4o, provides QBE.

In this case an empty object is instantiated (see Listing 1) and if necessary, some fields are set. The

query engine will then return all objects with matching fields. Using an empty object or null in db4o, all objects of the class will be retrieved.

This concept was already developed in the seventies by Moshé M. Zloof (IBM Research) and is presented here in a simpler form.

### Standard Queries

Query languages like HQL (Hibernate Query Language) or JPQL (Java Persistence Query Language) provide rich keywords as opposed to db4o's Native Queries (NQ).

db4o's Native Queries have other benefits. NQs are notated directly in the programming language. So – unlike HQL or JPQL strings – they're refactorable and type-safe during runtime. This eliminates many query errors (e.g., typing errors), which normally would only be discovered much later. One corresponding example in C# 2.0 and C# 3.0 is given in Listing 2 and Listing 3.

In more advanced – closure/block-capable – programming languages, anonymous delegates can be used to integrate queries seamlessly into the code. There's no need for the developer to learn a new language. The price here is that many advanced keywords such as aggregations or areas aren't available.

Native queries are never actually executed directly. db4o treats the query as domain-specific language capable of reading it during runtime to translate it into another language (SODA) and execute it afterwards. This is a powerful concept that we'll continue to encounter in dynamic languages like Ruby.

Only Microsoft's LINQ was able to realize both benefits: A complex query language in the programming language with a high variety of keywords. This undertaking is easier if you have the option of changing the language correspondingly as happened in C# 3.0. Here, in some areas LINQ is still significantly ahead of db4o, JPA/EJB, and Hibernate.

### Complex Queries

In Hibernate there's the so-called Criteria API for complex queries. Here object trees are built, constrained, and then executed. Listing 4 shows a manual example where all stored cat-objects are checked. Eventually only the ones starting with Fritz are to be chosen.

SODA, surprisingly enough, is a very similar API that's older than the Hibernate Criteria API. Since it started, SODA was open source and uploaded to sourceforge.net by db4o chief software architect Carl Rosenberger (a member of the EJB 3 JSR expert group) and has evolved through community contributions. Listing 5 shows a SODA Query.

### More Comparative Parameters

We'll illustrate below more of db4o's features and compare them to other popular solutions.

### Performance

As shown in many reports, db4o can be a very fast but at times fairly slow database system.

A scenario where db4o is strongest is in Seagate's hard disks using single-threaded access in the embedded mode. In most cases db4o is significantly faster than relational database management systems and OR-Mappers.

However, if the application is a Web application with many users triggering multithreaded queries via the application server, db4o can't keep up with relational solutions. In this case relational DBMS can be much faster. The reason for this is in db4o's design, as mentioned in the section about Transactions.

### Client/Server & Replication

It's very easy to start db4o in client/server mode. The only difference lies in creating an ObjectContainer that can be set up as a client or a server. This makes db4o an ideal choice for scenarios where the client collects data and sends it to the db4o server.

For these and similar scenarios, db4o provides two replication modes:

- 1) Using just a few lines of code, objects can be replicated either uni- or bi-directionally between db4o server instances.
- 2) This is even possible with relational databases.

With specific settings and Hibernate mappings, db4o data can be transferred to all databases that Hibernate supports: The perfect tool to combine both worlds.

### Callbacks

As an equivalent to triggers on the database system level, there are callbacks on the object level that make it easy to hook into the object lifecycle.

This tool is available in both Hibernate and Ruby on Rails with `before_validation()` or `after_save()`.

db4o's approach is easier. Here we have internal (in the data objects) and external callbacks. Internal callbacks are methods in the class like `onDelete()`, `onActivate()`, `canUpdate()`, and `canDeactivate()`, which are automatically detected and executed via reflection. This makes it easy to influence the persistence process in many ways such as executing object validations or setting fields.

In some cases, it's desirable to act on object state changes without a class so db4o introduced many external callbacks in version 6: `queryStarted`, `Creating`, `Activated`, `Updated`, `Deleted`, `Committed`, etc. They are simply registered with the `ObjectContainer` and make db4o an easily upgradeable and powerful persistence tool.

### Conclusion

Working with db4o is incredibly easy and a natural

extension of OO languages. Persistence developers will be thrilled by db4o's transparent and simple administration.

db4o also handles some of the most important schema changes transparently. Field types can be easily modified and exceptions can be easily added or deleted, without any drama. The database does what's expected, and class fields and class names can be adapted or migrated with just one line of code.

In a Hibernate/MySQL scenario, we would encounter many exceptions.

Nevertheless there's a downside to db4o, like the price paid for easily storing object data directly in a native format. There's another example besides the scalability drawbacks: Database management, data analysis, or OLAP reporting is done more expertly in relational solutions. This is where relational products show more maturity in having an architecture designed to please the enterprise application.

In conclusion we recommend that anyone interested in databases should evaluate db4o and compare it to established database solutions since a DAO only requires a little effort but quickly shows db4o's benefits and limits.

db4o is provided under a dual-license model like MySQL. Private non-commercial or open source GPL-use is free. There's a third license option similar to classpath licensing called dOCL so db4o can be embedded into other kinds of open

source products, an appealing option for open source developers.

Finally it's worth mentioning that the Object Database Management Group (ODMG) – which also standardized OQL many years ago – is working on a “fourth-generation” standard. It remains to be seen how this contribution will influence the database system world.

## References

- William Cook and Siddhartha Rai. “Safe Query Objects.” 2005. <http://www.cs.utexas.edu/users/wcook/>
- William Cook and Carl Rosenberger. “Native Queries for Persistent Objects.” <http://www.cs.utexas.edu/~wcook>
- Pieter van Zyl. “Object-Oriented Databases and Object Relational Mapping Tools in the Persistence Layer.” 2005. <http://espresso.cs.up.ac.za/>
- <http://blog.hibernate.org/cgi-in/blosxom.cgi/2007/05/23#in-defence>.
- <http://blogs.tedneward.com>. June 11, 2007.
- Dave Thomas and David Heinemeier Hansson. “Agile Web Development with Rails.” Pragmatic Programmers. Second Edition. 2007. ●

### LISTING 1 STORING AN OBJECT INSTANCE

```
ObjectContainer db=Db4o.openFile("C:/example.
db");
try { // Saves three objects
    db.set(new Customer("Max Fisch",
        "TOP INC",25));
    db.set(new Customer("John Doe",
"NEW Ltd.",50));
    db.set(new Customer("Jane Doe",
"NEW Ltd.",35));

    ObjectSet result=db.get(new Customer());
    while(result.hasNext()) { // Show all
        System.out.println(result.next());
    }
}
```

```
result=db.get(new Customer("Jane Doe"));
Customer found = (Customer)
result.next();
found.setAge(36); // Has birthday...
db.set(found); // Store object again
db.delete(found); // Had an accident...
db.commit();
}
finally {
    db.close();
}
```

### LISTING 2 NATIVE QUERY IN C# 2.0

```
IList<Person> persons = db.Query<Person>(delegat
e(Person person) {
    return person.Salary < 1000;
})
```

**LISTING 3 NATIVE QUERY IN C# 3.0**

```
var persons = db.Query<Person>(s => s.Salary < 1000);
```

**LISTING 4 HIBERNATE CRITERIA QUERY**

```
List cats = sess.createCriteria(Cat.class)
    .add(Expression.like("name", "Fritz%"))
    .add(Expression.between("weight",
        minWeight, maxWeight))
    .list();
```

**LISTING 5 DB4O SODA QUERY**

```
Query query=db.query();
query.constrain(Person.class);
```

```
Constraint firstConstr = query.descend("_age").
constrain(50).greater();
query.descend("_age").constrain(80).smaller().
and(firstConstr);
ObjectSet result = query
```

db4o as well as other solutions always provides multiple query options that can adapt to specific query requirements.

**franklins.net**  
Training Developers to Work Smarter.

www.franklins.net  
info@franklins.net  
Toll Free (866) 373-0730

# SharePoint. 2007 Video Training

Go deep into SharePoint 2007 in this 9-hour intensive class with Microsoft SharePoint MVP Sahil Malik hosted by MVP and Regional Director Carl Franklin from .NET Rocks! and dnrTV.

**Topics covered include:**

- Preparing a Good SharePoint Environment
- Building SharePoint Sites
- SharePoint Security
- Content Types
- Leveraging VSeWSS for Team Development
- SharePoint as a WCM
- Writing Custom WebParts
- Excel Services
- Leveraging InfoPath
- Workflows in MOSS
- BDC, the Business Data Catalog Administration and Monitoring



The SharePoint 2007 with Sahil Malik Video comes on a DVD with AVI Videos created with Camtasia™ Screen Recording Software at 1280x1024, as well as the full source code. Sahil is a master teacher, and Carl Franklin's commentary and questions keep you engaged in the learning process.

The DVD comes with a print copy of the course materials. Additional copies of the course materials can be purchased for an extra charge.



Microsoft  
Regional Director  
PROGRAM



Connect right now to [www.franklins.net](http://www.franklins.net) to order your copy!