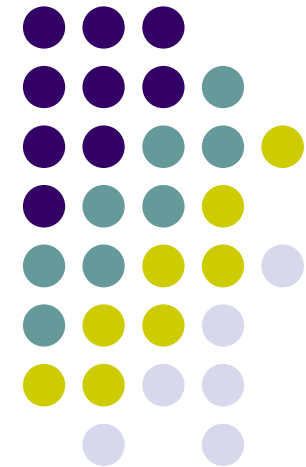


# Object-Persistenz meistern mit db4objects Fast, Easy & Powerful

Prof. Dr. Stefan Edlich



The banner features a dark grey header with the 'db4objects' logo on the left. Below the logo, there are three columns of navigation links: 'About' (Product Information, Customers, Partners, News, Company), 'Community' (Product Test Drive, Free Resources, Professionals, Blogs, Partners), and 'Commercial' (Product Purchase, dDN Member Portal, Partners). The main body of the banner shows a close-up of a runner's legs and feet on a track, with a green bar at the bottom containing a white arrow and the text 'download now'.



# db4objects = db4o



- | Eine reine Java Objektdatenbank (ca. 320Kb jar)
  - | C# / .NET and MONO Versionen sind ebenfalls erhältlich
- | Man braucht sich keine Gedanken um den „impedance mismatch“ zu machen!
- | Adressiert *embedded systems* aber genauso auch J2SE und J2EE Entwickler.
- | Frei für die interne private Verwendung (GPL) aber die Verwendung in einem kommerziell zu verkaufenden Produkt erfordert eine geringe Gebühr (non GPL)
- | Anwendungsfälle:
  - | Komplexe Objektmodelle
  - | In der Regel nur eine Anwendung die auf die Daten zugreift
  - | Kein Datenbank-Administrator nötig

# db4o features



- | Besitzt die einfachste DB API
- | Speichert verschlüsselt in einer beliebigen Datei (z.B. *customerData.yap*)
- | Speichert bis zu 254 GB in dieser Datei
  - | Werden Objekte in Gruppen aufgeteilt, und für jede Gruppe eine andere Datei verwendet, so kann die Dateigröße beliebig groß sein.
- | Man braucht weniger als 5 Minuten vom Download der Datenbank, bis man ein Objekt erfolgreich gespeichert hat!
- | Über Aufbau oder Mapping von Klassen oder Datenbankschemen keine Gedanken machen.
- | Speichert jedes Java-Objekt

# db4o Abfragesprachen



- | Es sind zwei Abfragesprachen enthalten:
  - | QBE = Query By Example
    - | Extrem simpel für einfache Abfragen
  - | S.O.D.A. = Eine graph query Sprache
    - | Sehr nah an Java. Bedingungen sind Objekte.
- | Weitere Abfragesprachen sind in Arbeit
  - | Stichwort: Safe Query Objects





# the db4o API

- | Speichere ein Object: `db.set(object);`
- | Lösche ein Object: `db.delete(object);`
- | Modifiziere ein Object: `object.setAtt(42);`
  - | In den letzten beiden Fällen ist es nötig, dass das Objekt vorher gefunden wurde:
- | Ein Objekt finden: `ObjectSet result = db.get(proto);`
- | Über das ResultSet iterieren:

```
while(result.hasNext()) {  
    System.out.println(result.next());  
}
```



# Query by example

- | `proto` kann ein leeres Objekt sein
  - | Man erhält alle gespeicherten Instanzen die diesem Klassentyp entsprechen
- | `proto` es können aber auch ein oder mehrere Attribute gesetzt sein. (Z.B. `name=„Arol“`, `note=1.0f`)

## Einfachstes Beispiel



```
package de.edlich.db4o;

import com.db4o.*;

public class TestDB4O {

    private void doDBstuff(){
// Db4o.main(null); // Print the Version
// Db4o.configure().messageLevel(0); //0=leise bis 4=laut
    ObjectContainer db = Db4o.openFile("D:/tmp/db001.dat");
// SAVE
    db.set(new Student("Bush", 199004444, 4.3f));
    db.set(new Student("Kohl", 199404444, 3.3f));
    db.set(new Student("Schröder", 199004444, 2.3f));
    db.set(new Student("Ghandi", 199004444, 1.3f));
    db.set(new Student("Einstein", 199004444, 1.0f));

// LIST ALL
// Student x1 = new Student();
// ObjectSet liste44 = db.get(x1);
// while(liste44.hasNext())
//     System.out.println(liste44.next());
```

Weiter hier →

```

//  SERARCH ONE AND UPDATE ONE
//  Student x2 = new Student();
//  x2.name = "Kohl";
//  ObjectSet liste = db.get(x2);
//  Student kohl = (Student) liste.next();
//  kohl.note=3.5f;
//  db.set(kohl);

//  DELETE
//  Student x2 = new Student();
//  x2.name = "Schröder";
//  ObjectSet liste = db.get(x2);
//  Student p2 = (Student) liste.next();
//  db.delete(p2);

//  LIST ALL
Student x5 = new Student();
ObjectSet liste31 = db.get(x5);
while(liste31.hasNext())
    System.out.println(">>>" + liste31.next());
}

public static void main(String args[]){
    TestDB40 run = new TestDB40();
    run.doDBstuff();
}
}

```



Mit einer Studentenklasse, die *String name*, *int nr* und *float note* hat. Dazu einen Konstruktor für alle drei Attribute, einen leeren Konstruktor und *toString()*.

# db4o performance



barcelona benchmarks	read	write	query	delete
db4o	1.0	1.0	1.0	1.0
Hibernate / MySQL	19.0	36.9	6.0	9.0
Hibernate / HSQLDB	14.4	15.5	388.6	6.3
JDBC / MySQL	9.8	18.7	1.0	4.0
JDBC / HSQLDB	0.4	3.5	453.6	0.6
JDBC / Derby	6,367.4	25.6	2,403.2	4.6
JDBC / "redmondsql"	35.9	21.8	8.5	2.8

fastest



slowest

(100 selects on 30,000 objects over 5 levels of inheritance structures)

# db4o performance



- Melbourne - writes 100,000 unstructured flat objects of one kind in bulk mode
- Bahrain - writes 30,000 simple flat objects individually
- Sepang - writes an object tree of depth 14
- Imola - retrieves 300,000 objects by native ID

other benchmarks	melbourne	bahrain	sepang	imola
db4o	1.0	1.0	1.0	1.0
Hibernate / MySQL	7.1	1.1	8.1	19.1
Hibernate / HSQLDB	2.8	0.3	2.4	11.8
JDBC / MySQL	2.9	0.5	2.5	5.7
JDBC / HSQLDB	0.4	0.1	0.9	0.1
JDBC / Derby	1.8	0.5	50.8	2.1
JDBC / "redmondsql"	2.8	0.4	4.9	22.2

fastest

slowest

I Welche Anwendungsfälle leiten wir aus diesen Ergebnissen ab?

# db4o performance



- I Db4o stellt eine Indexierung bereit, die die Performance auf Kosten des Speichers weiter steigert

```
Db4o.configure().objectClass(Foo.class).objectField("bar").indexed(true);
```

- I Das Speichern von 200.000 Objekten in der Sekunde macht db4o sogar für realtime Anwendungen interessant ...



# Transaktionen

- | db4o ist *transaction safe*!
- | `db.rollback()` und `db.commit()` können jederzeit verwendet werden.
- | db4o kann im Client-Server Mode gefahren werden:
  - | `Db4o.openServer()`
- | db4o ist auch für J2EE Apps gut geeignet. Der Container kann an Sessions geknüpft werden:  
`Db4oServlet.sharedTransaction()`



# db4o Objektmanagement

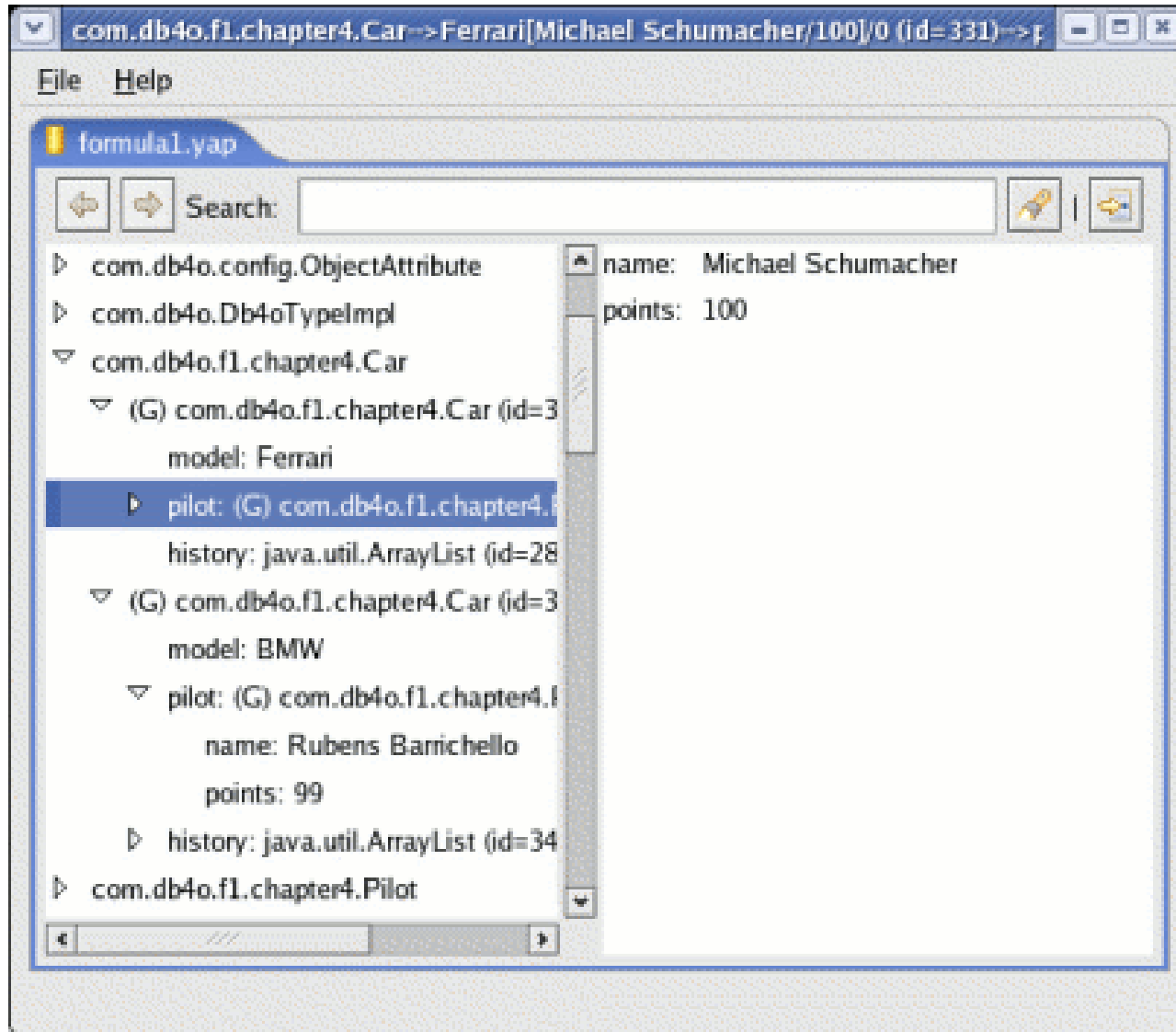
- | db4o hat eine konfigurierbare *activation depth* für das Holen und Speichern von Objekten (default =1)  
`activationDepth(int)` oder  
`activationDepth(Object int)`
- | Private Felder, komplexe Datentypen, innere Klassen oder Vererbung ist kein Problem.

# Replikation



- | Bidirektionale Replikation (default)
- | Der Container kann so konfiguriert werden, dass UUIDs generiert werden, die das Objekt für die Replikation eindeutig identifizieren
- | Konflikte können mit einem *Replication Conflict Handler* in Java gelöst werden
  - | Dieser liefert das Objekt `a`, `b` oder `null` zurück, um zu entscheiden, welche Replikationsinstanz gewinnt.

# Der Objektbrowser

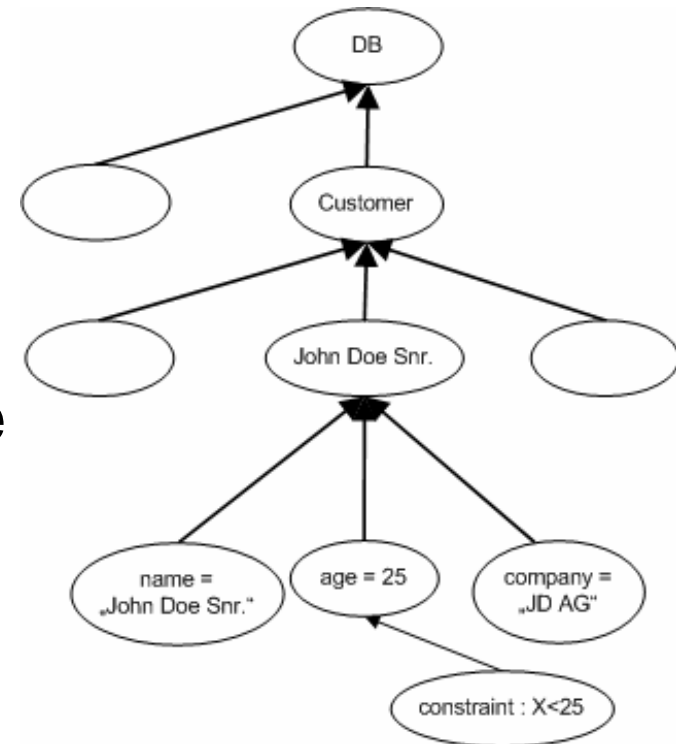


# Abfragen mit S.O.D.A.



- | Es wird ein object query graph erstellt
- | Bedingungen werden mit `constrain()` eingefügt
- | Dann das Ausführen der Abfrage `execute()`

```
Query qG = db.query();  
qG.constrain(Customer.class);  
qG.descend("age").constrain(new Integer(25)).greater();  
qG.descend("age").orderAscending();  
result=qG.execute();
```



# Fazit I



- | In sehr vielen Bereichen die schnellste verfügbare (Objekt) Datenbank
- | Hat wirklich die einfachste API für die Programmierung
  - | Man vergisst, dass man eine Persistenzschicht hat. In vielen Projekten nimmt die Objekt-Persistenz einen nicht unerheblichen Teil der Projektkosten in Anspruch
  - | Praxiserfahrung aus einem Projekt: Sie haben ein *User* Object, führen das Hibernate Mapping durch, generieren das Schema und testen gegen Oracle. Irgendwann teilt ihnen Oracle mit, es mag keine Tabelle *User*. Sie beginnen genervt von vorne!
- | Eine 320k Bibliothek / keine Installation nötig / nur lib in den Classpath
- | Versionen für Java und .NET.
  - | Kann in einem *cross platform* Modus verwendet werden

# Fazit II



- | Hat Transaktionen, kann replizieren und hat graphische Tools
- | Gut geeignet auch für J2EE Anwendungen
- | Wann bietet sich die Verwendung von db4o eher nicht an?
  - | Typischerweise in einer großen Data-Warehouse Umgebung, wo sich Datenbankadministratoren um die Datenbank kümmern, verschiedene Anwendungen mit verschiedenen (Objekt)-Sichten auf die DB zugreifen oder intensiv mit Data-Mining Tools gearbeitet werden muss.
- | Und sonst?
  - | Db4o hat im Download einen exzellenten interaktiven User-Guide als html und als pdf
  - | Auch an db4o Büchern wird gearbeitet...

# Kontakt



*edlich@acm.org*

## ANHANG A

Quelle: <http://www.polepos.org>  
the open source database benchmark



✓ **HSQLDB** looks very **good on many graphs**, because much processing is done in memory. We did not dig deep enough into HSQLDB source code, to understand if the behaviour is fully **compliant with ACID principles**, since ACID transactions would imply slow sync calls that make sure all data is flushed down to the hard disc. Before you consider to use HSQLDB for a complex database application, you should take a close look at Barcelona:query. The query optimizer does not seem to cope with a **query over five tables**.

✓ The use of O-R mapping technology like **Hibernate** or JDO O-R mappers has a strong **negative impact on performance**. If you can't compensate by throwing hardware at your app, you may have to avoid O-R mappers, if performance is important to you. Protagonists of O-R mapping technology may tell you that caching will bring you back performance. Although this may be true, do not forget: Data that comes from a **cache** can always be **stale** in a multi-user application. If you want your data to be up-to-date, you have to reread from the database and an old cache that you can not use will even degrade performance.

✓ As would be expected, object databases like **db4o** provide **best performance** for deep inheritance hierarchies and tree structures.

✓ If you need a relational database for whatever reason, **MySQL should certainly be on your evaluation list**. We did test quite a few other proprietary products with the benchmark suite, products that may not be listed here because of their license agreements. These other products did not look good at all. No wonder MySQL has such a **big and growing installed base**.

✓ If you take a look at the sources of the tests, they are also a measurement, how easy it is to **work against a specific database**. **SQL looks like a foreigner in Java code, very ugly, difficult to write and refactor**. **O-R mappers and object databases make life so much easier and productive**. The task of storing objects is very similar among all the products tested. **If a strong mainstream standard for object oriented querying can be reached, SQL will lose ground. This will happen for sure, the question is, how long it will take.**