db4o
the
alternative?

Agenda

- First steps
- Native Queries
- Dynamic Queries
- Different examples of using db4o.
- What is db4o?

What is db4o?

- db4o is an open source object oritened database.
- db4o is an embedded database.
- db4o is open source under GPL.

Preperations

```
public class Person {
     private String name;
     private int age;

     public Person(String name, int age) {
          this.name = name;
          this.age = age;
     }
     public int getAge() {...}
     public void setAge(int age) {...}
     public String getName() {...}
     public void setName(String name) {...}
}
```

First step

- Create the database and store an object(Person) into the database.

```
ObjectContainer db = Db4o.openFile(DB_FILE_NAME);
Person p = new Person("Karl", 40);
db.set(p);
db.close();
```

*error handling has been removed for clarity.

Second step

- Get the stored objects back from the database.

```
ObjectContainer db = Db4o.openFile(DB_FILE_NAME);
ObjectSet result = db.get(Person.class);
for (Iterator iter = result.iterator(); iter.hasNext();) {
      Person p = (Person) iter.next();
      System.out.println(p);
}
db.close();
```

Query by Example (QBE)

```
ObjectContainer db = Db4o.openFile(DB_FILE_NAME);
Person example = new Person(„Karl“, 0);
ObjectSet result = db.get(example);
for (Iterator iter = result.iterator(); iter.hasNext();) {
      Person p = (Person) iter.next();
      System.out.println(p);
}
db.close();
```

Limits of QBE

- Assume the following objects have been stored to a database.

```
Person p1 = new Person(„Heinz“, 20);
Person p2 = new Person(„Linda“, 45);
Person p3 = new Person(„Eric“, 22);
...
db.set(p1);
db.set(p2);
db.set(p3);
```

Limits of QBE

- How can we get the Persons who's age is between 40 and 50?
  - You can't use QBE.
  - The approach which solves this is called „Native Queries“.

Native Queries

```
List<Person> persons = db.query(
        new Predicate<Person>() {
            public boolean match(Person person) {
                return     person.getAge() >= 40
                    &&     person.getAge() <= 50;
            }
        }
);
```

Native Queries

- „Native Queries“ have the advantage that everything has been checked by the compiler.
  - Reduces the possiblity of errors.
  - Type safe.

Relations between objects

- Assume the following situation:
  - A Person has an account at the bank.

Relations between objects

- How do we write this in Java?

```
public class Account {
        private double amount;
        private double credit;
        ...
}
public class Person {
        private Account account;
        private String name;
        private int age;...
}
```

Relations between objects

```
Account a1 = new Account(1500, 500);
Person p1 = new Person("Heinz", 20);
p1.setAccount(a1);

Account a2 = new Account(5000, 1500);
Person p2 = new Person("Linda", 45);
p2.setAccount(a2);

Person p3 = new Person("Eric", 22);
Account a3 = new Account(150, 0);
p3.setAccount(a3);
```

Relations between objects

- How do we save those objects into the database?
  - Very simple just call „db.set(object).

```
db.set(p1);
db.set(p2);
db.set(p3);
```

Relations between objects

- How do we extract the Persons who's age is between 40 and 50 and the credit is greater than 1000?
- This can be solved using the „Native Queries“.

NQ's for relations

```
List<Person> persons = db.query(
        new Predicate<Person>() {
            public boolean match(Person person) {
            return      person.getAge() >= 40
                &&      person.getAge() <= 50
 &&     person.getAccount().getCredit() > 1000.0;
            }
        }
);
```

What about dynamic queries?

- You can't create dynamic queries using the „Native Query“ approach.
- Within this kind of situations, you have to use SODA (Simple Object Data Access).

Simple SODA

- Extract all Person objects from the database.

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

Simple SODA

- Extract all Person objects from the database who's name is „Eric“.

```
Query query=db.query();
query.constrain(Person.class);
query.descend(“name“).constrain(“Eric“);
ObjectSet result=query.execute();
```

Non simple SODA's

- Example of NQ's with SODA I:

```
Query query=db.query();
query.constrain(Person.class);
Constraint equconst50 = query.descend("age")
      .constrain(new Integer(50)).equal();
Constraint agele50 = query
      .descend("age")
      .constrain(new Integer(50))
      .smaller()
      .or(equconst50);
```

Non simple SODA's

- Example of NQ's with SODA II:

```
Constraint equconst40 = query
        .descend("age")
        .constrain(new Integer(40)).equal();

 Constraint agege40 = query
        .descend("age")
        .constrain(new Integer(40))
        .greater()
        .or(equconst40);
```

Non simple SODA's

- Example of NQ's with SODA III:

```
query.constrain(agege40).and(agele50);

ObjectSet result=query.execute();
```

Advanced SODA

- „Like“ Example:

```
Query query=db.query();
query.constrain(Person.class);
query.descend("name").constrain("H").like();

ObjectSet result=query.execute();
...
```

Advanced SODA

- „Order“ Example:

```
Query query=db.query();
query.constrain(Person.class);
query.descend("name").orderAscending();
ObjectSet result=query.execute();

...
```

Arrays

- You can search using the NQ approach:

```
Query query=db.query<Object>(
        public boolean match(Object item) {
            //Search for the particular values.
        }
);

...
```

Arrays

- You can search using the SODA approach:

```
Query query=db.query();
query.constrain(Object.class);
Query queryarr = query.descend(„fieldname“));
queryarr.constrain(1.0);
queryarr.constrain(2.3);

ObjectSet result = query.execute();
```

Collections

- You can search using the NQ approach:

```
Query query=db.query<Object>(
        public boolean match(Object item) {
            for(....) {
                //check for the particular values
            }
        }
);
```

Updating Objects

- Update of an object.

```
ObjectSet result = (ObjectSet)
      db.get(new Person(„Hugo“, 0));
Person person = (Person)result.next();

person.setAge(49);
db.set(person);
...
```

Deleting Objects

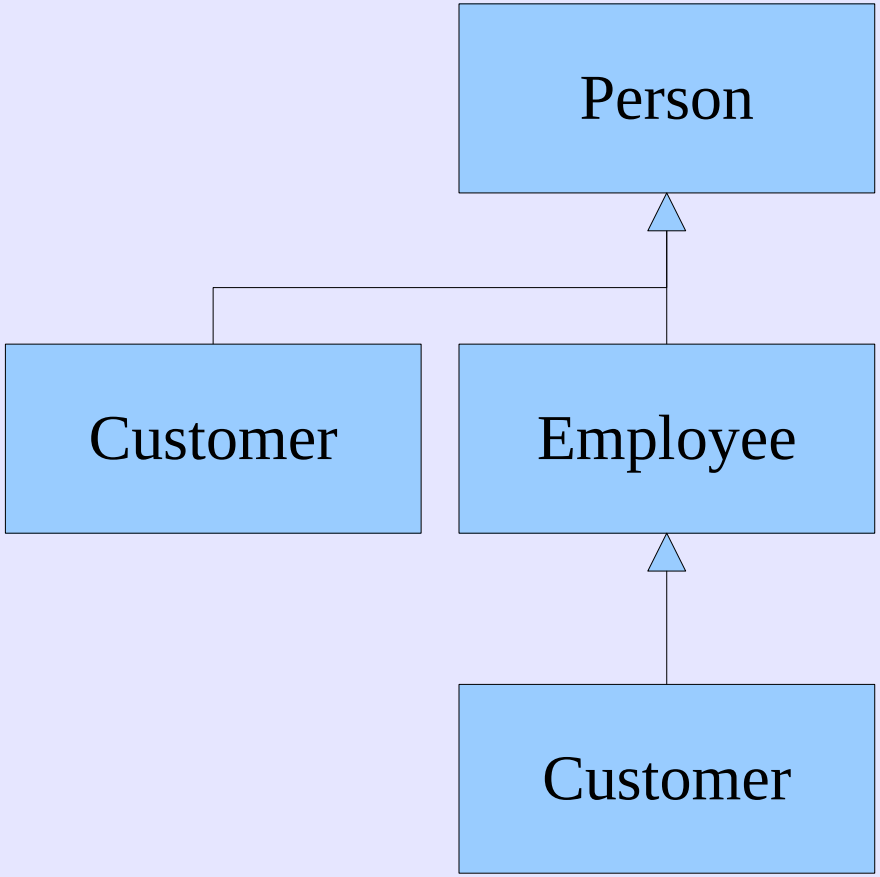
- Delete of an object.

```
ObjectSet result = (ObjectSet)
        db.get(new Person(„Hugo“, 0));
Person person = (Person)result.next();

db.delete(person);
...
```

*Cacade on delete can be configured

Inheritance



Inheritance

- Save a few objects

```
Person p1 = new Person(...);
Customer c1 = new Customer(...);
Employee e1 = new Employee(...);
Manager m1 = new Manage(...);
db.set(p1);
db.set(c1);
db.set(e1);
db.set(m1);
```

Inheritance

- Get objects

```
db.get(Person.class);

db.get(Customer.class);

db.get(Employee.class);

db.get(Manager.class);
```

Transactions

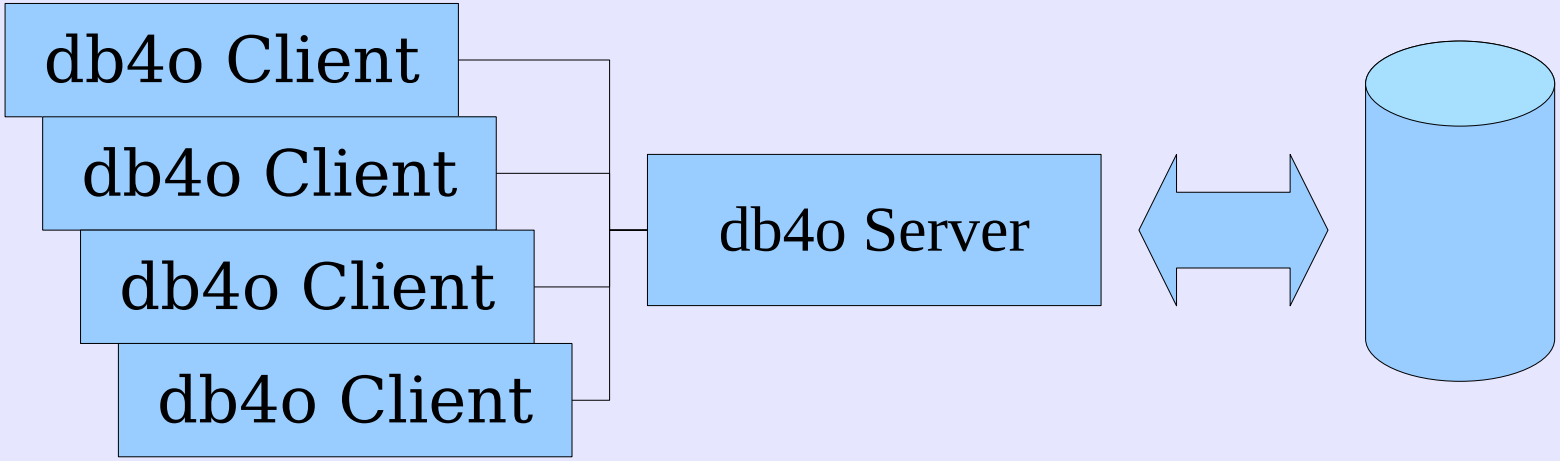
```
ObjectContainer db = Db4o.openFile(DB_FILENAME);
Person p1 = new Person("Heinz", 20);
try {
      db.set(p1);
} catch (Exception e) {
      db.rollback();
} finally {
      db.commit();
      db.close();
}
```

Indexes

- If you need you can turn indexes on or off for different attributes.

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

Client/Server Mode



Client/Server

- Good examples can be found in the documentation online or within the distribution.

Replication to „SQL“ DB

- You can use „dRS“ to replicate information from the OODB (db4o) to the RDMS using Hibernate.
- This can handy if you have application which are working with data in RDMS.

Sumary: Why using db4o?

- Very simple to use
- No need to make an distinction between OO model and the database model, cause the OO model ist the „database“ model.
- Type safe „Native Queries“.

db4o vs. RDBMS

- Pro's:
  - No special layer needed (ORM like Hibernate etc.). Mapping of objects to tables v.v. etc.

db4o vs. RDBMS

- Con's:
  - A thing like SQL does not exist in db4o ;-(
    The programming language is the „Query language“.
  - Constraints do not exist, your application has to implement them.

db4o vs. RDBMS

- Con's:
  - Limited file size of db4o
    - Based on the configuration(blocksize) from 2 GB to 264 GB.
    - A work around might to use different database files, but this won't work all the time.

db4o vs. RDBMS

- Con's:
  - No database clustering available.
  - No permission concept; If you need it you have to implement it yourself.
  - Not established, lack of third party tools for data mining and reporting.

Embedded vs. Server

- Pro's:
  - More or less no administration, cause we have no real (db)-server like other embedded db's as well (defragmentation has to be done by hand.)
  - No installation procedure.

Online Resources

- db4o Homepage
  http://www.db4o.com
- http://ootips.org/
- http://odbms.org/about_editor.html
- http://www.odbmsjournal.org/

Questions?

Thank for your attention.

Blog:

blog.soebes.de

Contact:

osdbc@soebes.de