Admin matters (1)

1. Study guide available on COS326 website
   - Lecture schedule
   - Assignment essay presentation dates
   - List of essay topics available Friday 31 July
   - 1st presentation: 11 August
   - Assignment topic bookings on line

2. Tutorial 1 & prac 1 for this week: db4objects
   - Documents available on website
   - Marking will be done during prac session
   - Bookings for marking
   - Teaching Assistant consultation hours: to be announced
## Admin matters (2)

### Activities for next 2 weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>28 Jul</td>
<td>Tues</td>
<td>L3: Object DBMS</td>
</tr>
<tr>
<td></td>
<td>29 Jul</td>
<td>Wed</td>
<td>L4: Object DBMS</td>
</tr>
<tr>
<td></td>
<td>31 July</td>
<td>Fri</td>
<td><strong>Practical 1:</strong> Object DBMS (<em>db4objects</em>)</td>
</tr>
<tr>
<td>3</td>
<td>4 Aug</td>
<td>Tues</td>
<td>L5: Object-Relational DBMS</td>
</tr>
<tr>
<td></td>
<td>5 Aug</td>
<td>Wed</td>
<td>L6: Object-Relational DBMS</td>
</tr>
<tr>
<td></td>
<td>7 Aug</td>
<td>Fri</td>
<td><strong>Practical 2:</strong> Object-Relational DBMS (<em>PostgreSQL</em>)</td>
</tr>
</tbody>
</table>
In this lecture:

1. **Introduction to OODBMSs (27.5)**
   a. Definitions & origins of OODBs(27.5.1)
   b. Persistent programming languages (27.5.2)
   c. OO Database system Manifesto (27.5.4)
   d. Alternative strategies for OODBMSs
   e. advantages and disadvantages of OODBMSs
      (reading for student)

2. **Case study: db4objects OODBMS**
   a. Tutorial documents
   b. Reference document
   c. Tutorial 1 & Prac 1
27.5.1 Definitions of OODBMSs

- **OO data model**
  - (logical) data model that captures the semantics of objects supported in object-oriented programming

- **OO database**
  - persistent sharable collection of objects defined by an OO data model

- **OODBMS**
  - manager (software) of an OODB
**OODB environment**

**OO data model defines contents of OODB**

**PropertyForRent** (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)

**PrivateOwner** (ownerNo, fName, lName, address, telNo)

**Client** (clientNo, fName, lName, address, telNo, prefType, maxRent)

**Lease** (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentFinish)
## 27.5.3 Persistent programming languages

- **Persistent programming language**
  - provides users with ability to preserve data across successive executions of program (e.g. PS-Algol, Napier88: historical)
  - strong influence on modern OODBMSs

- **Database programming language**
  - Integrates ideas from database programming model with traditional programming languages
    - Embedding SQL in C, Fortran, Pascal, COBOL, etc.
      - OR
    - API with database functionality e.g ODBC, JDBC
  - Impedence mismatch
import java.sql.*
public class JDBCSample {
    public static void main( String args[ ]) {
        String connectionURL =
            "jdbc:postgresql://localhost:5432/movies;user=java;password=samples";
        try {
            Class.forName("org.postgresql.Driver");
            Connection con = DriverManager.getConnection (connectionURL);
            Statement stmt = con.createStatement();
            ResultSet rs =
                stmt.executeQuery("SELECT moviename, releasedate FROM movies");
            while (rs.next())
                System.out.println("Name= " + rs.getString("moviename")
                    + " Date= " + rs.getString("releasedate");
        }
        finally     {  con.close();     }
    }
}
String connectionURL = "jdbc:postgresql://localhost:5432/movies;user=java;password=samples";
try {
    Class.forName("org.postgresql.Driver");
    Connection con = DriverManager.getConnection (connectionURL);
    Statement stmt = con.createStatement();
    ResultSet rs =
        stmt.executeQuery("SELECT moviename, releasedate FROM movies");
    while (rs.next())
        System.out.println("Name= " + rs.getString("moviename") + .....}

More serious than impedance mismatch:
- errors in the strings cannot be detected by Java compiler
- results in run-time errors
=> slows down program development
27.5.5 Alternative Strategies for developing an OODBMS

1. **Extend** existing object-oriented programming language
   - add traditional database capabilities to existing OOP language, e.g. Smalltalk, C++, Java
   - *e.g.* GemStone extended Smalltalk, C++, Java

2. **Provide extensible OODBMS library**
   - **No extensions** to the OOP language
   - provide off-the-shelf library
   - Library provides persistence and database capabilities
     - Work in known programming language with same compiler
     - **examples are**: ObjectStore and db4objects
Reading for student

3. Imbed OODB language constructs in conventional host language: $O_2$: extensions for C

4. Extend existing database language (SQL) with object-oriented capabilities
   - RDBMS and OODBMS vendors
     » SQL (1999) by ISO supports OO features, Object SQL standard by ODMG
   - Ontos and Versant provide a version of Object SQL

5. Develop novel database data model/language
   - No history/baggage, start from scratch
   - No installed base, no skills
   - Semantic Information Manager (SIM)
Case study: db4objects and Java (1)

- Provides extensible OODBMS library: Tutorial and prac for this week

The database location e.g. using Eclipse

Database file in folder Users\user
db4objects and Java (2): Eclipse (Java) Project files
db4objects and Java (3)

✈ Code for db4o first example

```java
//package com.db4o.f1.chapter1; in tutorial
package pilotPackage;
import java.io.*;
import java.util.*;
//from referenced library (db4objects)
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.query.Query;
import com.db4o.Db4oEmbedded;
```
public class FirstStepsExample extends Util {
    final static String DB4OFILENAME = 
            System.getProperty("user.home") + "/formula1.db4o";

    public static void main(String[] args) {
        new File(DB4OFILENAME).delete();     accessDb4o();
        new File(DB4OFILENAME).delete();
        ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded
            .newConfiguration(), DB4OFILENAME);

        try {
            storeFirstPilot(db);   // plus code to store other pilots
            retrieveAllPilots(db);
            retrievePilotByName(db);
            retrievePilotByExactPoints(db);
            updatePilot(db);
            deleteFirstPilotByName(db);
        } finally {
            db.close();
        }
    }
}
public static void storeFirstPilot(ObjectContainer db) {
    Pilot pilot1 = new Pilot("Sebastian Vettel", 100);
    db.store(pilot1);
    System.out.println("Stored " + pilot1);
}

public static void retrieveAllPilotQBE(ObjectContainer db) {
    Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.queryByExample(proto);
    listResult(result);
}

public static void deleteFirstPilotByName(ObjectContainer db) {
    ObjectSet result = db.queryByExample(new Pilot("Sebastian Vettel", 0));
    Pilot found = (Pilot) result.next();
    db.delete(found);
    System.out.println("Deleted " + found);
}
Browsing database objects: OME perspective

- db4o Browser

![db4o Browser](image-url)
db4objects query types

- **Query By Example (QBE)**
  - simplest type of query

- **Native Queries (NQ)**
  - completely expressed in the implementation language (eg. Java), & fully obey all language semantics

- **SODA**
  - the low level API
  - QBE & NQ queries are translated into SODA

- **All support: store, retrieve, update, delete operations**
### Comparison of RDBMS & OODBMS QUERIES

<table>
<thead>
<tr>
<th>Query</th>
<th>RDBMS SQL</th>
<th>OODBMS: db4objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a database</td>
<td>?</td>
<td>e.g.</td>
</tr>
</tbody>
</table>

```
final static String DB4OFILENAME = System.getProperty("user.home") + "/formula1.db4o";

new File(DB4OFILENAME).delete();
accessDb4o();

new File(DB4OFILENAME).delete();

ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), DB4OFILENAME);
```
## Table with query types (2)

<table>
<thead>
<tr>
<th>Query</th>
<th>RDBMS SQL</th>
<th>OODBMS: db4objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a database item</td>
<td>CREATE TABLE ... CREATE INDEX ... CREATE VIEW ... etc</td>
<td>N/A in OME</td>
</tr>
<tr>
<td>Store data in the database</td>
<td>INSERT INTO ...</td>
<td>e.g. Pilot pilot1 = new Pilot(&quot;Michael Schumacher&quot;, 100); db.store(pilot1);</td>
</tr>
<tr>
<td>Get data from database</td>
<td>SELECT ...... FROM ......</td>
<td>QBE: Pilot proto = new Pilot(null, 0); ObjectSet result = db.queryByExample(proto); alternatively: ObjectSet result = db.queryByExample(Pilot.class);</td>
</tr>
</tbody>
</table>

NQ: Exercise for the student
SODA: Exercise for the student
### Table with query types (2)

<table>
<thead>
<tr>
<th>Query</th>
<th>RDBMS SQL</th>
<th>OODBMS: db4objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get data from database</td>
<td>SELECT ......&lt;br&gt;FROM......&lt;br&gt;WHERE .....</td>
<td><strong>QBE:</strong>&lt;br&gt;Pilot proto = new Pilot(&quot;Michael Schumacher&quot;, 0);&lt;br&gt;ObjectSet result =&lt;br&gt;db.queryByExample(proto);&lt;br&gt;ObjectSet result =&lt;br&gt;db.queryByExample(proto);&lt;br&gt;NQ: Exercise for the student&lt;br&gt;SODA: Exercise for the student</td>
</tr>
<tr>
<td>Update database data</td>
<td>UPDATE ...&lt;br&gt;WHERE ...</td>
<td><strong>QBE:</strong>&lt;br&gt;ObjectSet result =&lt;br&gt;db.queryByExample(new Pilot(&quot;Michael Schumacher&quot;, 0));&lt;br&gt;Pilot found = (Pilot) result.next();&lt;br&gt;found.addPoints(11);&lt;br&gt;db.store(found);&lt;br&gt;NQ: Exercise for the student&lt;br&gt;SODA: Exercise for the student</td>
</tr>
<tr>
<td>Query</td>
<td>RDBMS SQL</td>
<td>OODBMS: db4objects</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| Delete a database item | DELETE ... WHERE ... | QBE: ObjectSet result = 
  db.queryByExample(new Pilot 
    ("Michael Schumacher", 0));
  Pilot found = (Pilot) result.next();
  db.delete(found); |
|       |           | NQ: Exercise for the student |
|       |           | SODA: Exercise for the student |
Advantages of OODBMSs

Reading for the student

- Enriched modeling capabilities
- Extensibility
- Removal of impedance mismatch
- More expressive query language
  - Tied to programming language?
- Support for schema evolution
- Support for long duration transactions
- Applicability to advanced database applications
- Improved performance
Disadvantages of OODBMSs

Reading for the student

- Lack of universal data model
  - ODMG object data model
- Lack of experience
- Lack of standards
  - ODMG object query language (ObjectSQL)
- Competition from RDBMS and ORDBMS
- Query optimization compromises encapsulation
- Object level locking may impact performance
- Complexity
- Lack of support for views
- Lack of support for security