COS 326
Database Systems

Lecture 4
Object-oriented Databases (3)
Chapters 27 and 28
29 July 2015
Announcements

- Submission link prac 1 active on website
- Prac1 marking check website on how to book time slot
- Consultation hours for Teaching Assistants

<table>
<thead>
<tr>
<th>Name</th>
<th>Contact details</th>
<th>Consultation hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruth Ojo</td>
<td><a href="mailto:ruthojo93@gmail.com">ruthojo93@gmail.com</a></td>
<td>Tuesday 14:30 – 15:20</td>
</tr>
<tr>
<td>Neo Thobejane</td>
<td><a href="mailto:timjozi911@gmail.com">timjozi911@gmail.com</a></td>
<td>Thursday 12:30 – 13:20</td>
</tr>
<tr>
<td>Lindelo Mapumulo</td>
<td><a href="mailto:u12002862@tuks.co.za">u12002862@tuks.co.za</a></td>
<td>Monday 10:30 – 11:20</td>
</tr>
</tbody>
</table>
In this lecture

- Case studies of OODBMSs
  - db4objects (recap)
  - Objectstore (ch.28)

- OO Database design (ch. 27, section 27.9)
  - Reading for the student

- OODBMS standards (ch. 28)
  - Reading for the student

How do OODBMSs overcome the weaknesses of RDBMSs as discussed previously?
Recap:
Strategies for developing an OODBMS

One of the strategies is: provide extensible OODBMS library

- 'Off-the-shelf' library
- Library provides persistence and database capabilities
- No need for separate compiler
- Work in known programming language
- examples are:
  - db4objects (Java, VB.NET, C#.NET)
  - ObjectStore (C++, Java)
 ООDBMS Case study: db4objects for Java

- **Target environments:**
  - persistence architectures where there is **no DBA present** &
  - for equipment, mobile and desktop clients

*(refer to: db4o-reference-java.pdf)*
  - Seems to be getting replaced with Versant Object Database which supports:
    - Single machine (client-server)
    - Network (client-server)

Refer to:
http://supportservices.actian.com/versant/default.html

Java DB engine for:
Java 1.5 and later
*db4o-8.0-core-java5.jar*
built for Java JDK 5 and JDK 6
What is ObjectStore?

- ObjectStore is a ‘pure’ object database
- Stores C++ objects in the same format as they are used in memory
- Accessed and updated using the same C++ syntax as heap-allocated objects
What is ObjectStore?

- Database contains memory pages
- Client starts a transaction and uses persistent objects exactly ‘as if’ they were heap allocated
- Pages fetched automatically on an ‘as needed’ basis as the client navigates persistent pointers
What is ObjectStore?

- Server coordinates sharing of pages between multiple clients
- Page read/write permits and locks are managed automatically to ensure transaction consistency
ObjectStore is Distributed

Client can access objects in different remote databases in same txn
The C++ client is the program you write
- It is linked with the ObjectStore libraries
- It opens databases and runs transactions against them
- It allocates objects, calls methods on them, and deletes them
- Pages are fetched automatically from the DB as needed
- ObjectStore automatically maintains a cache of recently accessed pages

Client is a C++ application program
ObjectStore Programming

- Programmer uses the ObjectStore libraries
- some classes
  - objectstore
  - os_database
  - os_transaction
  - os_typespec
  - os_database_root
  - os_segment
  - os_cluster
Opening and Closing a DB

```c++
#include <ostore/ostore.hh>
int main (int argc, char** argv)
{
    objectstore::initialize()
    OS_ESTABLISH_FAULT_HANDLER
    os_database* db = os_database::open(
        "d:/x.odb", 0, 0664);

    // ...your code
    db->close();
    OS_END_FAULT_HANDLER
    objectstore::shutdown();
    return 0;
}
```
Creating a Persistent Object

```cpp
os_database* db = os_database::open("d:/x.odb");

OS_BEGIN_TXN(use_case1, 0, os_transaction::update)
{
    Foo* f = new (db, ts<Foo>()) Foo();

    //...other code...
}
OS_END_TXN(use_case1)

//... other code ...
```
Section 27.9 Object Oriented Database design

- 27.9.1 Comparison of
  » OO data modeling and
  » conceptual data modeling (INF214)

- 27.9.2 Relationships and referential integrity

27.10 OO analysis and design with UML
Reading for the student

- Object Management Group (OMG)
  - CORBA, and other OMG standards

- Main features of ODMG Object Standard:
  - Object model
  - Object Definition Language (ODL)
  - Object Query Language (OQL)
Recap: weaknesses of RDBMs (1)

1. Poor representation of “real world” entities
   - *Normalization* leads to relations that do not correspond to entities in the “real world”
   - *joins needed* to obtain information from relations (tables)

2. Semantic Overloading
   - Single construct for representing data and data relationships: the relation
   - No mechanism to distinguish between relations & relationships or between different types of relationships

How do OODBMSs e.g. db4objects overcome the above problems?
Solutions provided by OODBMs e.g db4object

1. Representation of “real world” entities
   - **OOP constructs are used**: classes, inheritance, composition, etc.
   - **No joins needed** to access object (entity) information

2. No semantic overloading.

- **Object identifier (OID) generated by DBMS**
  - **Logical OID**: Independent of physical location on disk
  - **Physical OID**: Points to an actual location on disk
  - (for db4objects refer to reference manual: object identity)

- **Representation of relationships (business rules)**:
  - one-to-one, one-to-many, many-to-many relationships
  - (OBJECT COMPOSITION) done using OIDs

- **Representation of inheritance relationships**:
  - done through class definitions
Object access: e.g. Resident object table (ROT)

- **Branch:** OID1
  - branchNo: B003
  - street: 163 Main St
  - city: Glasgow
  - postcode: G11 9QX
  - staff: {OID4, OID5, ...}
  - property: {OID2, OID3, ...}
  - manager: OID6

- **Resident Object Table**
  - **OID** | memory address
  - 1 | ...
  - 2 | ...
  - 3 | ...
  - 4 | ...
  - ... | ...

- **PropertyForRent:** OID2
  - propertyNo: PG4
  - street: 6 Lawrence St
  - type: Flat
  - rooms: 3
  - rent: 350
  - staff: OID4
  - branch: OID1

- **PropertyForRent:** OID3
  - propertyNo: PG16
  - street: 5 Novar Dr
  - type: Flat
  - rooms: 4
  - rent: 450
  - staff: OID4
  - branch: OID1

---

**e.g. of locating objects:**

- Lookup virtual memory pointer with each object access

**OIDs obtained from DBMS**

- Using *constructors* & overloaded *new operator* (ch. 28, sec 28.3.3)
1:1 Relationship between objects A and B

One-to-one relationship

- Add reference attributes to both objects
- Reference is an OID (Recall: OIDs obtained from DBMS)

**Branch: OID1**

- branchNo: B003
- street: 163 Main St
- city: Glasgow
- postcode: G11 9QX
- staff: {OID4, OID5, ...}
- property: {OID2, OID3, ...}
- manager: OID6

**Manager: OID6**

- staffNo: SG5
- fName: Susan
- lName: Brand
- position: Manager
- sex: F
- DOB: 3-Jun-40
- salary: 24000
- branch: OID1
1:* Relationship between objects A and B

One-to-many relationship

Add reference attribute to one object (branch)

AND

attribute containing set of references to the other object (property)

So, why are joins not needed?
Many-to-many ( *:* ) relationships

Client can view many properties.
A property can be viewed by many clients

**Add** attribute containing set of references to each object
Recap: weaknesses of RDBMs (2)

3. **Limited Operations**

- RDBMSs only have a fixed set of operations which cannot be extended i.e. *operations on relations (sets of tuples) & tuples*
- Not possible to specify new operations

- How do OODBMSs e.g. db4objects overcome the above problems?
Solutions: Limited operations

3. Limited Operations

- Solution: OOP methods are defined for classes

- Where are the methods stored?
  - Store methods in external files
    » Link libraries into the application at compile/link time
  OR
  - Store methods in database
    » Dynamically bound to application at run-time
    » Eliminates redundant code,
    » Simplifies modifications,
    » Methods are more secure,
    » Methods can be shared concurrently, Improved integrity
Storing and executing methods

- **Store methods in external files**
  - Link libraries into the application at compile/link time

- **Store methods in database**
  - Dynamically bound to application at run-time
5. Homogeneous data structure
   - Horizontal and vertical homogeneity is restrictive for ‘real world’ objects
   - intersection of row and column must be atomic

6. Impedance mismatch
   - Mixing different programming paradigms & different models to represent data
   - => increased programming effort

◆ How do OODBMSs e.g. db4objects overcome the above problems?
Solutions (3)

5. Homogeneous data structure
6. Impedance mismatch

◆ How do OODBMSs e.g. db4objects overcome the above problems?

– Heterogenous data structures used, i.e. different objects based on different classes

– Code for DB operations uses same syntax as that of the programming language (e.g. Java and db4o, C++ for ObjectStore)
7. Transactions
   - Short-lived (business) vs long-duration (design)

- How do OODBMSs e.g. db4objects overcome the above problem?

- Transaction is a logical unit of work
  - Takes database from one consistent state to another
  - Object is the unit of concurrency control and recovery
  - db4objects and ObjectStore are transaction based

- Long-duration transactions
  - Require different protocols e.g. version management
Object versions

- **Object version**: Identifiable state of an object e.g.
  - Transient - can be updated or deleted
  - Working - cannot be updated, can be deleted
  - Released - cannot be updated or deleted

- **Version management**
  - Process of maintaining evolution of objects
  - classes are declared as *versionable* (or not)
  - different versions of a (e.g. design) objects are stored in OO database
Object versions

Public database

Private workspace

- Transient - can be updated or deleted
- Working - cannot be updated, can be deleted
- Released - cannot be updated or deleted
Recap: weaknesses of RDBMs (5)

8. Schema changes are difficult
   – Database change requires program changes

How do OODBMSs e.g. db4objects overcome the above problem?

READING FOR THE STUDENT
Common changes are:

1. Changes to class definition:
   a. Modifying Attributes
   b. Modifying Methods

2. Changes to inheritance hierarchy:
   a. Making a class $S$ superclass of a class $C$
   b. Removing $S$ from list of superclasses of $C$
   c. Modifying order of superclasses of $C$

3. Changes to set of classes
   a. creating and deleting classes
   b. modifying class names
Referential integrity: any referenced object must exist

Techniques to handle referential integrity:

1. System is responsible for “garbage collection”

2. User deletes objects when they are not required anymore
   - System detects invalid references
   - sets reference to NULL; or
   - does not allow deletion of referenced object

3. User deletes/ modifies objects as required
   - System automatically maintains the integrity of objects
Homework (1)

CAD database stores data relating to engineering designs e.g.
(1) Cars
(2) Buildings
(3) Aircraft
(4) Integrated Circuit chips

Source: www.wikipedia.org

Question 1:

a. Write down the characteristics of CAD databases that make it difficult to store CAD data in a relational database.

b. Explain how each of the characteristics you have identified can be supported by an object oriented database
Question 2:
Choose any two of the following applications: CAM, OIS, GIS, and do the following:

a. Write down the characteristics of the application’s database that make it difficult to store data in a relational database.

b. Explain how each of the characteristics you have identified can be supported by an object oriented database

Question 3:
Refer to the Object-Oriented-Database System Manifesto (pg. 825-828 of textbook). To what extent does the db4objects OODBMS meet the requirements of this manifesto?