

Object-Oriented Databases & Relations \square Objects

Robert Keller
CS 133
3/3/04

What is an OODB?

- OODB = Object-Oriented Database
- Some of the earliest data models were special cases:
 - Network or CODASYL model
 - Hierarchical model
- These models, being more focused, *potentially* provide for *more efficient* querying than relational models,
- at the expense of making queries more complex and less dynamic.

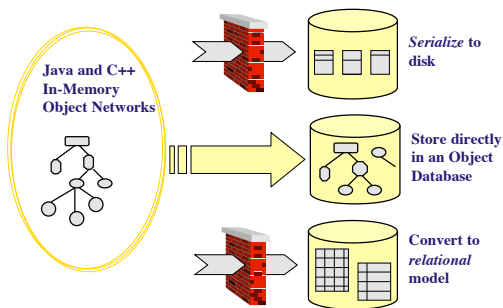
Programming View

- OODB is like Object-Oriented Programming, except that:
 - Objects are *persistent* (survive the current session)
 - Transaction mechanism is available:
 - Changes only **committed** at the end of a transaction.
- Essentially:
 - Your C++ or Java program becomes a database program.

OO vs. Relational

- OO doesn't use SQL or the equivalent
- Queries are typically "wired in"
- There is an emergent OQL (Object Query Language)
- OODB has closer ties to the Entity-Relationship model and UML (Unified Modeling Language).
- Another facet: CORBA standard (Common Object Request Broker)

3 Choices for Object Management ...



OODBMS Access Efficiency

- Typically data will be brought to main memory from secondary memory the first time the data are accessed.
- The code to transfer the data is *transparent* to the program.
- The data will remain in main memory until committing or until the end of the session.

Examples of OODBMS (Object-Oriented Database Management Systems)

- A somewhat aging list:
 - POET™ (<http://www.poet.com/>)
 - ObjectStore™ (<http://www.odi.com/>)
 - Objectivity™ (<http://www.objectivity.com/>)
 - Versant™ (<http://www.versant.com/>) merged with POET
 - Gemstone™ (<http://www.gemstone.com/>)
 - O2™ (<http://www.ardentsoftware.com/>)
- Standards are emerging, thanks to the ODMG (Object Data Management Group)

Use POET™ as "Typical" (but now obsolescent) Use Java as the Language

- Data are stored on disk with symbolic name.
- Schema (through a schema file) determines the DB structure.
- Upon *opening* the database, the *root* of the structure on disk is *bound* to a reference **variable** in memory.
- Further bindings take place as the structure is navigated, using OOP.

POET/Java Examples

- Database db = Database.open(dbName, Database.openReadWrite);
 - class
 - static method
 - static constant
- Extent e = new Extent(trans, classname);
 - transaction
 - string
 - Java Enumeration of class objects

```
while( e.hasMoreElements() )
{
  - e.nextElement() ...
}
```

POET/Java Examples

- db.add(myobject, trans);
 - reference
- ObjectServices.delete(myobject);
 - static method

Transactions

- All access (even read-only) is done within a **transaction**.
- Transaction trans = new Transaction(db);


```
trans.begin();
  .
  .
trans.checkpoint();
  .
  .
trans.commit();
```

Schema File

```
[schemata\poetOgreDict]
oneFile = true

[databases\poetOgreDB]
schema = poetOgreDict
location = SAME
oneFile = true

[classes\poetOgreObject]
persistent = true
hasextent = true
schema = poetOgreDict
```

Connecting Objects to Relations

- Model 1: Store object (reference) as **values** in relations
 - 1 object (reference) = 1 attribute value
 - An object is of a specific data type.
Thus a set of objects of the same type can be stored as a single relation.
 - Object's methods are over-and-above the relational model.

Connecting Objects to Relations

- Model 2: Relations are **associations** in OOP
 - Objects are free-standing
 - Each association between classes is a relation.

Connecting Objects to Relations

- Model 3: Objects identified with tuples
 - Attributes become fields.
 - Values of attributes become values of fields.
 - A whole relation is a set of objects of the same class.

OGRE (Objects Generated From Relations)

A transformation tool developed by
Jeff Polakow '98 and Robert Keller
sponsored by JPL

Genesis of OGRE

- pre-1996 JPL: "Grok"
- 1996 JPL CS Clinic Project: "Condor"
- 1997 JPL CS Clinic Project: "Ogre"
(originally Oracle-Grok Interface)
- 1998-1999: Two new versions, using
some ideas from Condor.

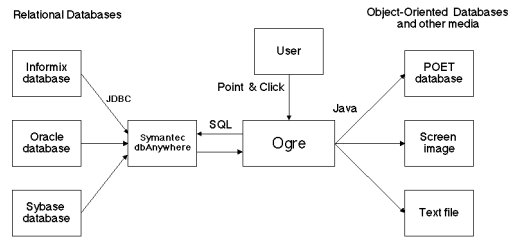
Ogre Features (1)

- "Any" JDBC/ODBC-Compliant
Relational Database □ POET OODB
using Model 2.
- In principle, could be made to work with
any OODB.
- Could also generate text or XML.

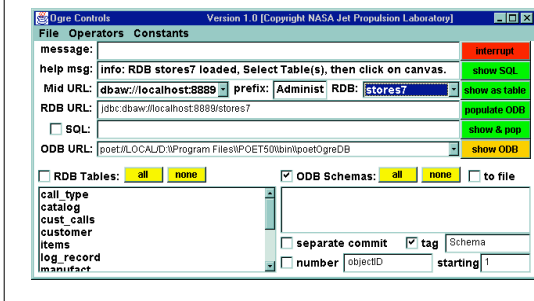
Ogre Features (2)

- GUI allows a naïve user to generate the OODB; essentially converts graphical queries into SQL to access RDB.
- The conversion alone may be of interest, aside from the connection with OODBs.

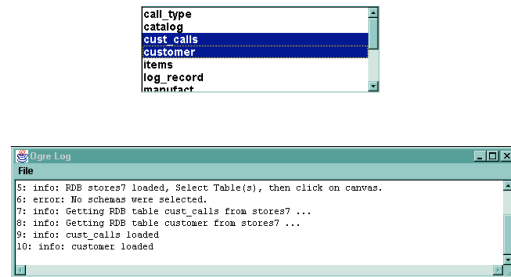
Ogre Architecture



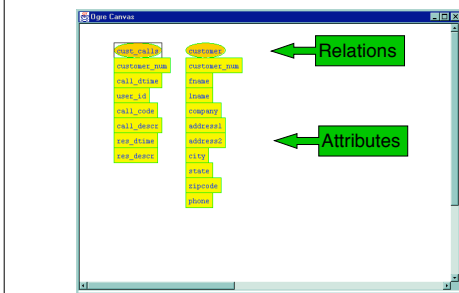
Loading an RDB into Ogre



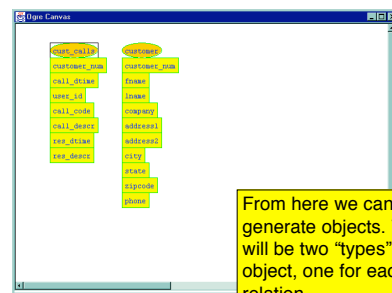
Selecting Tables



Displaying Selected Relations



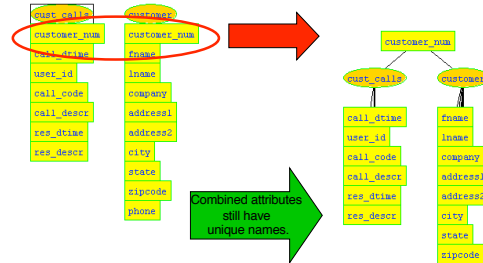
Putative object types: ODB Schemas: all none
cust_calls
customer



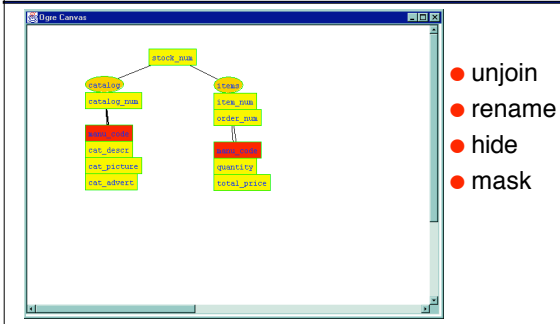
“Universal” Object Type

- In the case of Ogre, objects are actually all the same Java class:
 - The principal field is an *association list* (attribute-value pairs).
 - This is used because the schema would otherwise be unknown until runtime.
- In principle, a static-compilation approach is possible.

Joining Relations to create Single Objects



Joining Nuances



Viewing OODB

```

OODB contents
File
Name = Farnellee
Schema = cust_calls_customer
call_time = 1993-11-28 13:34:00.0

Object number: 7

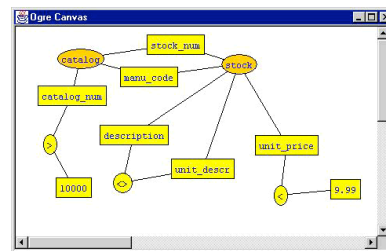
customer_num = 116
Name = Jean
call_code = 1
user_id = mannym
company = Olympic City
Name = Farnellee
Schema = cust_calls_customer
call_time = 1993-12-21 11:24:00.0
    
```

Generated SQL

```

Ogre Log
File
11: info: SQL: Select cust_calls.call_time, cust_calls.user_id, cust_calls.call_code, cu
12: info: Completed query: Select cust_calls.call_time, cust_calls.user_id, cust_calls.c
13: info: SQL: Select cust_calls.call_time, cust_calls.user_id, cust_calls.call_code, cu
14: info: Populating ODB: .....
15: info: Population completed, committed 7 objects to database.
16: info: Completed query: Select cust_calls.call_time, cust_calls.user_id, cust_calls.c
    
```

Graphical Querying with Built-in Operators



Ogre Web Page

- <http://www.cs.hmc.edu/~keller/ogre/>