

CS319 : Theory of Databases

OODBMS, ORDBMS

presented by
Timothy Heron*

February 19, 2004

*E-mail: theron@dcs.warwick.ac.uk

Data Models

When developing database applications we usually have to consider two different data models :

- Relational Data Model : Used in RDBMS.
- Object-Oriented Data Model : Used in programming.

Our data within our program is stored in an object model while the data in our database is a relational model.

OO models can represent structures found in the real world more naturally than the relational model.

But we still want to *persist* the data in our OO model in a database.

Comparing OO and Relational Data Models

- A relation or table can be considered to be analogous to a class.
- A tuple is similar to a instance of a class (object) but it only has attributes and no methods.
- A column in a tuple is similar to an object attribute except that a column can only hold primitive types while an object attribute can hold complex types.

The OO model has features that are not easily represented in the relational model. Can this functionality be added to a DBMS ?

Object Database Systems

Object Database Systems have developed along two distinct paths :

- Object-Oriented Database Systems (OODBMS)
Heavily influenced by OO programming languages. Attempts to add database functionality to an OO programming language.
- Object-Relational Database Systems (ORDBMS)
An attempt to extend relational database systems so that a bridge can be made between the OO paradigm and the relational paradigm.

Object-Oriented Database Systems

OODBMS are not simply a database, they have to include a full OO development environment (since the language is very closely tied to the database).

Interactions with the objects within the database are indistinguishable from interaction with normal objects in the language.

A query language is not required to create/select/update/delete objects (although one does exist).

A Java example

This segment of Java code example uses the *ObjectStore* OODBMS :

```
//Open database
Database db = Database.open("IMdatabase", ObjectStore.UPDAT

//Get hashtable of user objects from DB
OSHashMap users = (OSHashMap) db.getRoot("IMusers");

//Get password and username
String username = "theron";
String passwd = "not_a_very_secure_password";

//Get user object from database
UserObject user = (UserObject) users.get(username);
```

```
//See user exists and whether password is correct
if (user == null)
    System.out.println("Non-existent user");
else
    if (user.getPassword().equals(passwd))
        System.out.println("Successful login");
    else
        System.out.println("Invalid Password");
```

Notice that the object retrieved from the database not only contains attributes but methods as well.

Object ID's

Every object has a unique OID (Object ID) that is :

- Unique
- Immutable
- Generated by the database system
- Not based on any data within the object

When the client application requests an object it is transferred from the database into the application's cache where it can be used in two ways :

- As a transient value that is disconnected from its representation in the database.
- As a mirror of the object in the database, in this case updates to the object also update the object in the database and changes to the object in the database require that the object is refetched from the OODBMS.

OODBMS Standards

The ODMG (Object Data Management Group)

<http://www.odmg.org> has created a data model that forms the basis for an OODBMS in a similar way that the relational data model forms the basis for a RDBMS.

They use OQL (Object Query Language) instead of SQL (although it looks similar).

They use ODL (Object Definition Language) instead of SQL DDL (like CREATE TABLE).

The ODMG and Sun have defined JDO (Java Data Objects) which extends OODBMS capabilities to Java in a database independent way.

Object Definition Language (ODL)

```
class <name> {
  <list of element declarations, separated by semi-colons>
}
```

Elements can either be Attributes or Relationships :

Attributes :

```
attribute <type> <name>;
```

Relationships :

```
relationship <type> <name> inverse <relationship>;
```

Class Declaration Example

```
class Album {
  attribute string album;
  attribute integer num_tracks;
  relationship Artist recorded_by inverse Artist::recorded;
}
```

```
class Artist {
  attribute string name;
  attribute int no_members;
  relationship Set<Album> recorded inverse Album::recorded;
}
```

Instead of Set<Album>, you can also have Bag<Album>, List<Album>, Array<Album>.

Object Query Language (OQL)

OQL queries are used to lookup and query objects from the database. OQL queries are similar to SQL queries, but use object names instead of SQL names and do not require join clauses.

For example, if the object being loaded is of type `Person`, the OQL query will load from collection `FROM People`, if a join is required to load related objects, it will be performed automatically.

```
OQLQuery oql;
QueryResults results;

//Construct a new query and bind its parameters
oql = db.getOQLQuery("SELECT p FROM People WHERE p.age < 18");

//Retrieve results and print each one
results = oql.execute();
while (results.hasMore())
{
    Person per = (Person) results.next();
    System.out.println(per.name());
}
```

Commercial OODBMS

- ObjectStore, <http://www.objectstore.com>
- Objectivity, <http://www.objectivity.com>
- Jasmine, <http://www.ca.com>
- Versant, <http://www.versant.com>

Open Source

- Castor, - <http://castor.exolab.org>
- Xorm, - <http://www.xorm.org>

More can be found at <http://www.jdocentral.com>.

Notice that all these are niche products from small vendors, the major database vendors have adopted a different approach.

Object-Relational Database Systems

SQL:1999 defines standard Object-Relational extensions to SQL92, it maintains backwards compatibility and this has placed a number of restrictions on the OO enhancements.

It is a (relatively) new technology and database vendors support it to varying extents.

Oracle supports SQL:1999 (with some exceptions).

Data Types

An Object-Relational DBMS provides a wider range of data types than the primitive CHAR, VARCHAR, NUMBER, etc available in a RDBMS.

- User-defined data types. The user can define their own more complicated data types.
- Inheritance of data types. As the number of data types increases it is important to take advantage of the commonality between different types.
- Object Identity. We want to be able to link objects together and reference one object's data from another.

User-defined Data Types

A user-defined type is a class definition with attributes and methods.

It can be used as :

- A row type
- An attribute type

```
CREATE TYPE Band AS OBJECT (
  name VARCHAR(40),
  no_of_members NUMBER,
);
```

Row Type

Define a table to be of a particular type :

```
CREATE TABLE <table name> OF <type name>;
so
CREATE TABLE artists OF Band;
```

creates a structure equivalent to the following table :

```
CREATE TABLE artists AS (
  name CHAR(40),
  no_of_members NUMBER,
);
```

Attribute Type

Specify attributes in a table to be of a particular type.

```
CREATE TABLE Pop_Album (
  barcode CHAR(12),
  name CHAR(40),
  artist Band
);
```

In this table the attribute *artist* is of type Band that we defined earlier.

Adding data

The database provides default *constructors* that we can use to populate our objects with data :

```
INSERT INTO Pop_Album VALUES
  (003948827895,"Unforgettable Fire",Band("U2",4));
```

Manipulating user defined data types

Oracle requires us to name any object types we use in a query (we cannot just use the class name).

```
SELECT album.artist.name FROM CD_Album album;
```

will select the band names of the CD's we have in our collection.

we cannot use :

```
SELECT CD_Album.artist.name FROM CD_Album;
```

Object Methods

Objects encapsulate data and methods together so that the collection of methods presented by an object are the only way to manipulate the data within that object. This way the internal representation of the data within the object is independent of the way the object is used.

This allows the implementation to change without impacting other areas of our system.

We have seen one method already, the constructor that creates a new object for us.

Circles Example

The following type stores details about circles :

```
CREATE TYPE CircleType AS OBJECT (
  x NUMBER,
  y NUMBER,
  radius NUMBER,
  MEMBER FUNCTION area() RETURN NUMBER;
  PRAGMA RESTRICT_REFERENCES(area,WNDS)
);
```

NB. The 'PRAGMA' specifies that the function `area` will not modify the database state, WNDS means Write No Database State.

The Method Body

```
CREATE TYPE BODY CircleType AS
  MEMBER FUNCTION area() RETURN NUMBER IS
  BEGIN
    RETURN 3.14 * SELF.radius * SELF.radius;
  END;
END;
```

We can then create a table :

```
CREATE TABLE Circles (
  CircleID NUMBER,
  Circle CircleType
);
```

We can then insert some data about circles :

```
INSERT INTO Circles VALUES (1,CircleType(3,4,2));
INSERT INTO Circles VALUES (2,CircleType(1,7,4));
```

And finally perform a query including a calculation that calls our method :

```
SELECT CircleID, circ.area() FROM Circles circ;
```

Methods can be written in different languages depending on the database. Oracle allows methods to be written in either PL/SQL or Java.

REF's

The special type REF allows us to create references (or pointers) to other objects.

```
CREATE TABLE BigCircles (
  circle REF CircleType,
);
```

```
INSERT INTO BigCircles
  SELECT REF(c)
  FROM Circles c
  WHERE c.radius > 10;
```

This does not create a copy of the circle information but creates a table that contains references to all the large circles.

LOB's

SQL:1999 defines two new data types :

- BLOB - Binary Large Object
- CLOB - Character Large Object

LOB's are references (or pointers) to data stored outside of the database. They are used for images, sound files, application executables, etc.

The database does not know anything about the internal structure of the LOB and so it cannot index, sort or ensure the type safety of LOB's unlike built-in or user-defined data types.