Database Management Systems

Winter 2003

CMPUT 391: Object Oriented Databases

Dr. Osmar R. Zaïane

University of Alberta

Chapter 23 of Textbook

Course Content

- Introduction
- Database Design Theory
- Query Processing and Optimisation
- Concurrency Control
- Data Base Recovery and Security
- **Object-Oriented Databases**
- Inverted Index for IR
- XML
- Data Warehousing
- Data Mining
- Parallel and Distributed Databases
- Other Advanced Database Topics

Objectives of Lecture 6

Object-Oriented Databases

- Discuss limitations of the relational data model.
- Introduce object databases, databases that handle complex data types.
- Understand the difference between object-oriented databases and object-relational databases.
- (By no means comprehensive)

Object-Oriented Databases

- Shortcomings of Relational Databases
- The Concept of Object data Model
- Object-Oriented Database Systems
- Object-Relational Database Systems
- CORBA

The Need for a DBMS

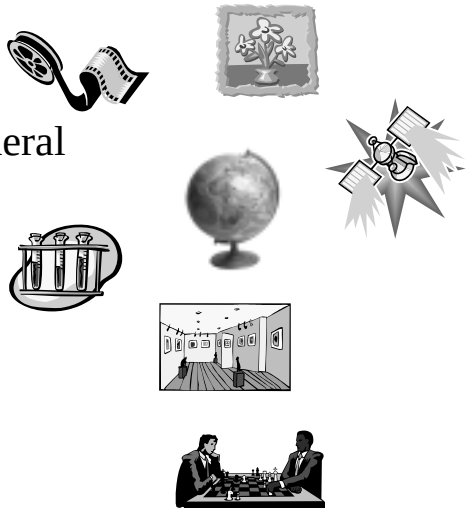
- On one hand we have a tremendous increase in the amount of data applications have to handle, on the other hand we want a reduced application development time.
  - Object-Oriented programming
  - DBMS features: query capability with optimization, concurrency control, recovery, indexing, etc.
- Can we merge these two to get an object database management system since data is getting more complex?

Manipulating New Kinds of Data

- A television channel needs to store video sequences, radio interviews, multimedia documents, geographical information, etc., and retrieve them efficiently.
- A movie producing company needs to store movies, frame sequences, data about actors and theaters, etc. (textbook example)
- A biological lab needs to store complex data about molecules, chromosomes, etc, and retrieve parts of data as well as complete data.
- Think about NHL data and commercial needs.

What are the Needs?

- Images
- Video
- Multimedia in general
- Spatial data (GIS)
- Biological data
- CAD data
- Virtual Worlds
- Games
- List of lists
- User defined data types



Shortcomings with RDBMS

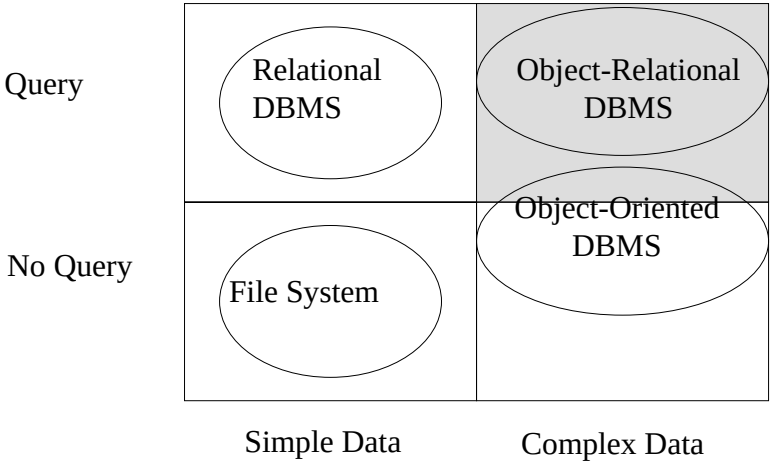
- Supports only a small fixed collection of relatively simple data types (integers, floating point numbers, date, strings)
- No set-valued attributes (sets, lists,…)
- No inheritance in the Is-a relationship
- No complex objects, apart from BLOB (binary large object) and CLOB (character large object)
- Impedance mismatch between data access language (declarative SQL) and host language (procedural C or Java): programmer must explicitly tell how things to be done.

➔ Is there a different solution?

Existing Object Databases

- Object database is a persistent storage manager for objects:
  - Persistent storage for object-oriented programming languages (C^{++}, SmallTalk,etc.)
  - Object-Database Systems:
    - Object-Oriented Database Systems: alternative to relational systems
    - Object-Relational Database Systems: Extension to relational systems
- Market: RDBMS ($8 billion), OODMS ($30 million) world-wide
- OODB Commercial Products: ObjectStore, GemStone, Orion, etc.

DBMS Classification Matrix



Object-Oriented Databases

- Shortcomings of Relational Databases
- The Concept of Object data Model
- Object-Oriented Database Systems
- Object-Relational Database Systems
- CORBA

Object Data Model

- The object data model is the basis of object-oriented databases, like the relational data model is the basis for the relational databases.
- The database contains a collection of Objects (similar to the concept of entities)
- An object has a unique ID (OID) and a collection of objects with similar properties is called a class.
- Properties of an object are specified using ODL and objects are manipulated using OML.

Properties of an Object

- **Attributes**: atomic or structured type (set, bag, list, array)
- **Relationships**: reference to an object or set of such objects.
- **Methods**: functions that can be applied to objects of a class.

Abstract Data Type

- One key feature of object database systems is the possibility for the user to define arbitrary new data types.
- A new data type should come with its associated methods to manipulate it. The new data type and its associated methods is called abstract data type (ADT).
- DBMS has built-in types.
- How does the DBMS deal with new data types that were never seen before.

Encapsulation

- Encapsulation = data structure + operations
- It is the main characteristic of object-oriented languages.
- The encapsulation hides the abstract data type internals. ADT= opaque type.
- The DBMS does not need to know how the ADT's data is stored nor how the ADT's methods work. DBMS only needs to know the available methods and how to call them (input/output types of the methods)

Inheritance

•A value has a type
•An object belongs to a class

- Type (class) hierarchy
  - System permits the definition of new types based on other existing types
  - A subtype inherits all properties of its supertype
- Class hierarchy
  - A sub-class C' of a class C is a collection of objects such that each object in C' is also an object in C.
  - An object in C' inherits all properties of C
    - may change the behaviour of some methods (overloading/overriding of methods)
    - typically adds additional attributes and methods
- Multiple inheritance (inherits from more than just one superclass)
- Selective inheritance (inherits only some of the properties of a superclass)

Common Structured Types

- Type constructors are used to combine atomic types and user defined types to create more complex structures:
- ROW(n_1, t_1, …n_n,t_n) : tuple of n fields
- listof(base): list of base-type items
- ARRAY(base): array of base-type items
- setof(base): set of base-type items without duplicates
- bagof(base): multiset of base-type items

Not all collection types supported by all systems

Objects, OIDs, and Reference Types

- An object has an identity and the system can generate an object identifier (OID) for objects which is unique in the database across time
- Reference types - REF(basetype) - have object ids as values, i.e., an object of type REF(*basetype*) is basically a "pointer" to an object of type *basetype*.

Object-Oriented Databases

- Shortcomings of Relational Databases
- The Concept of Object data Model
- Object-Oriented Database Systems
- Object-Relational Database Systems
- CORBA

Object-Oriented Databases

- OODBMS aims to achieve seamless integration with an object-oriented programming language such as C++, Java or Smalltalk.
- OODBMS is aimed at applications when an object-centric view point is appropriate. (occasional fetch from object repository)
- No efficient implementations for DML. There are no good optimizations for a query language such as OQL in OODBMSs today.

ODL in OODBMS

- ODL supports atomic types as well as set, bag, list array and struct type
- Interface defines a class

```
interface Movie (extent Movies key movieName)
        { attribute date start;
          attribute date end;
          attribute string   movieName;
          relashionship Set<Theater> ShownAt inverse Theater::nowShowing;
        }
interface Theater (extent Theaters key theaterName)
        { attribute string theaterName;
          attribute string address;
          attribute integer ticketPrice;
          relationship Set <Movie> nowShowing inverse Movie::shownAt;
          float numshowing() raises(errorCountingMovies);
        }
```


OML in OODBMS

- The most poplar query language is OQL which is designed to have a syntax similar to SQL.
- OQL is an extension to SQL. It has select, from and where clauses.
- The extensions are to accommodate the properties of objects and the operators on complex data types.

OQL Examples

Find the movies and theaters such that the theaters show more than one movie.

```
SELECT mname: M.movieName, tname: T.theaterName
FROM Movies M, M.shownAt T
WHERE T.numshowing() >1
```

Use of path expression
T is bound to each theater
Related to movie M by
relationship shownAt

Method of Objects can be called
everywhere in the query

Find the different ticket prices and the average number of movies shown at theaters with that ticket price.

Partitioning in OQL

```
SELECT T.ticketPrice,
        avgNum:AVG(SELECT P.T.numshowing() FROM partition P)
FROM Theaters T
GROUP BY T.ticketPrice
```


Language Bindings

- Mapping of ODL object definitions to the native syntax of the host language; allows to define and implement database objects in the host language
- Allows accessing and querying database objects from within the host language
- Allows to make objects of particular classes persistent, e.g. in Java by explicitly calling the method “persist(*object*)”, defined in the interface *Database*, or implicitly if referenced by a another persistent object.

Java Binding Example

```
import org.odmg.*;
import java.util.Collection;

Implementation impl = new com.vendor.odmg.Implementation();
Database db = impl.newDatabase();
Transaction txn = impl.newTransaction();
try {
    db.open("movieDB", Database.OPEN_READ_WRITE);
    txn.begin();
    OQLQuery query = new OQLQuery(
       "select t from Theaters t where t.ticketprice < $1");
    query.bind(uInput1()); //bind $1 to a user specified value
    Collection result = (Collection) query.execute();
    Iterator iter = result.iterator();
    while ( iter.hasNext() ) {
        Theater theater = (Theater) iter.next();
        theater.ticketprice = theater.ticketprice * 1.5;
    }
    txn.commit();
    db.close();
}
//exception handling would go here ...
```


Object-Oriented Databases

- Shortcomings of Relational Databases
- The Concept of Object data Model
- Object-Oriented Database Systems
- Object-Relational Database Systems
- CORBA

ORDBMS: What's new? (SQL 1999)

- Support for storage and manipulation of large data types (BLOB and CLOB)
- Mechanisms to extend the database with application specific types and methods
  - User defined types
  - User defined procedures
  - Operators for structured types
  - Operators for reference types
- Support for inheritance

User Defined ADT

- A user must define methods that enable the DBMS to read in and to output objects for each new atomic type defined.
- The following methods must be registered with the DBMS:
  - Size: returns the number of bytes of storage
  - Import: creates a new object from textual input
  - Export: maps item to a printable form

CREATE ABSTRACT DATA TYPE jpeg_image
(internallength =VARIABLE, input=jpeg_in, output=jpeg_out);

Built-in Operators for Structured Types

- Path expression
- Comparisons of sets ($\subset \subseteq = \supseteq \supset \in \cup \cap -$)
- Append and prepend for lists
- Postfix square bracket for arrays
- -> for reference type

Object-Relational Features of Oracle

Object table: table in which each row represents an object.

```
CREATE TYPE person AS OBJECT (
  name        VARCHAR2(30),
  phone       VARCHAR2(20) );

CREATE TABLE person_table OF person;

INSERT INTO person_table VALUES
       person("John Smith", "1-800-555-1212");

SELECT VALUE(p) FROM person_table p
        WHERE p.name = "John Smith";
```


Object-Relational Features of Oracle

Methods

```
CREATE TYPE Rectangle_typ AS OBJECT (
  len NUMBER,
  wid NUMBER,
  MEMBER FUNCTION area RETURN NUMBER,
);

CREATE TYPE BODY Rectangle_typ AS
  MEMBER FUNCTION area RETURN NUMBER IS
  BEGIN
     RETURN len * wid;
  END area;
END;
```


Object-Relational Features of Oracle

REF datatype: reference to other objects

```
CREATE TABLE people (
  id            NUMBER(4)
  name_obj      name_objtyp,
  address_ref   REF address_objtyp
                  SCOPE IS address_objtab);
```

can be "scoped" for more efficient access

De-referencing (assume X is an object of type people)

```
X.deref(address_ref).street
```

In Oracle also implicitly: `X.address_ref.street`

Obtaining references

```
SELECT REF(po) FROM purchase_order_table po
WHERE po.id = 1000376;
```

Object-Relational Features of Oracle

Collection types / nested tables

```
CREATE TYPE PointType AS OBJECT (
     x NUMBER,
     y NUMBER);
CREATE TYPE PolygonType AS TABLE OF PointType;
CREATE TABLE Polygons (
     name   VARCHAR2(20),
     points PolygonType)
     NESTED TABLE points STORE AS PointsTable;
```

The relations representing individual polygons are not stored directly as values of the points attribute; they are stored in a single table, PointsTable

Object-Relational Features of Oracle

Collection types / VARRAYS

```
CREATE TYPE prices AS VARRAY(10) OF NUMBER(12,2);
```

- The VARRAYs of type PRICES have no more than ten elements, each of datatype NUMBER(12,2).
- Creating an array type does not allocate space. It defines a datatype, which you can use as:
  - the datatype of a column of a relational table.
  - an object type attribute.

Object-Relational Features of Oracle

Type Inheritance / Subtyping

```
CREATE TYPE Person_t AS OBJECT
( ssn NUMBER,
  name VARCHAR2(30),
  address VARCHAR2(100)) NOT FINAL;

CREATE TYPE Student_t UNDER Person_t
( deptid NUMBER, major VARCHAR2(30)) NOT FINAL;

CREATE TYPE Employee_typ UNDER Person_t
( empid NUMBER, mgr VARCHAR2(30));

CREATE TYPE PartTimeStud_t UNDER Student_t
( numhours NUMBER);
```

To permit subtypes, the object type must be defined as not final.

Object-Relational Features of Oracle

Method Overloading and Overriding

```
CREATE TYPE MyType_typ AS OBJECT (...,
  MEMBER PROCEDURE Print(),
  MEMBER PROCEDURE foo(x NUMBER), ...)
  NOT FINAL;

CREATE TYPE MySubType_typ UNDER MyType_typ
(...,
  OVERRIDING MEMBER PROCEDURE Print(),
  MEMBER PROCEDURE foo(x DATE), ...);
```

MySubType_typ contains two versions of foo( ): one inherited version, with a NUMBER parameter, and a new version with a DATE parameter

Object-Oriented Databases

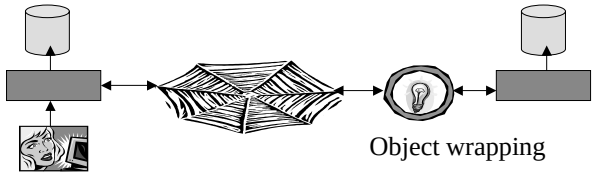
- Shortcomings of Relational Databases
- The Concept of Object data Model
- Object-Oriented Database Systems
- Object-Relational Database Systems
- CORBA

Distributed Objects

- To integrate different applications running on the same or different computers, we use a middleware for distributing objects.
- There are many technologies: COM/DCOM (Distributed / Component Object Model) from Microsoft, CORBA from OMG, RMI with Java, SOAP with XML, etc.
- Heterogeneity is due to:
  - **Engineering tradeoffs**: different solutions across the enterprise
  - **Cost effectiveness**: best system at the lowest price in ≠ times
  - **Legacy systems**: systems too critical or too costly to replace
- Dealing with heterogeneity in distributed computing enterprise & develop open applications is very challenging

Concepts of Middleware

- Objects are sent from one application to the other via a middleware.
- The middleware wraps objects with a network layer
- Some technologies rely on TCP/IP, other on HTTP



CORBA

- CORBA stands for Common Object Request Broker Architecture.
- It is defined and managed by the Object management Group (OMG)
- CORBA is known for Object Orientation, Interoperability, Heterogeneity and Transparent-Distribution.
- Not a product. It is a standard used to exchange data in a heterogeneous environment, large scale enterprise applications distributed on a network.

CORBA con't

- CORBA makes it easier to implement new applications that must place components on different hosts on the network or use different programming languages.
- CORBA encourages the writing of open applications, applications that can be used as components of large systems, each application is made up of components and integration is supported by allowing other applications to communicate directly with these components.

CORBA con't

- OO, Interoperability, and Distribution Out-of-the-box
- Interoperability across languages (Java, C/C++, Ada, Smalltalk, Common LISP, COBOL, etc.)
- Interoperability and Portability across Operating-Systems and Networks (CORBA is available on virtually every OS that you might want to use)
- Distribution / Location Transparency are Fundamental

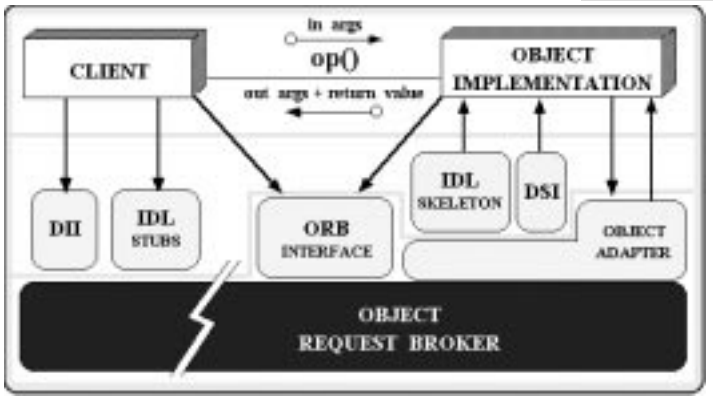
The Notion of Client-Server

- With CORBA there is not rigid notion of a client and a server; components communicate with others on a peer-to-peer basis
- Client and server are roles filled on a per-request basis
- A component can be a client and a server at the same time: client for other services and server for the services it provides

CORBA ORB Architecture

the program entity that invokes an operation on an object implementation

a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*



S. Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.

References for CORBA & OMG

Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.
http://www.cs.wustl.edu/~schmidt/PDF/vinoski.pdf

http://www.cs.wustl.edu/~schmidt/tutorials-corba.html

http://www.omg.org/

http://www.iona.com/