Object Database Management Systems (ODBMSs)

CSC 436 – Fall 2003

Evolution and Definition of the ODBMS

- **Object data model (ODM):** A data model that captures the semantics of objects supported in object-oriented programming.
- **Object database (ODB):** A persistent and sharable collection of objects defined by an ODM.
- **ODBMS:** The manager of an ODB.
Object Data Model

**Definition One:** An ODBMS must, at a minimum:
- Provide database functionality
- Support object identity
- Provide encapsulation
- Support objects with complex state

**Definition Two:**
- Object-orientation = ADTs + Inheritance + Object identity
- Object-oriented database = Object-orientation + Database capabilities

**Definition Three:** An ODBMS must provide:
1. High-level query language with query optimization capabilities in the underlying system
2. Support for persistence and atomic transactions: concurrency and recovery control
3. Support for complex object storage, indexes and access methods for fast and efficient retrieval
   
   Object-oriented database = object-oriented system + (1)-(3)

---

Issues in ODBMSs

- **Traditional database Systems**
  - Persistence
  - Sharing
  - Transactions
  - Concurrency control
  - Security
  - Integrity
  - Querying

- **Semantic data models**
  - Generalisation
  - Aggregation

- **Object-Oriented programming**
  - Object identity
  - Encapsulation
  - Inheritance
  - Types and classes
  - Methods
  - Complex objects
  - Polymorphism
  - Extensibility

- **Special requirements**
  - Versioning
  - Schema evolution

Object Data Model
Transactions

- Transactions involving complex objects can continue for several hours (or even days)
  => Support long duration transactions => re-examine concurrency and recovery protocols

- In an ODBMS, the unit of concurrency and recovery control is logically an object.

- Lock conflict in long duration transactions is unacceptable
  => Use versioning

Versioning

- The process of maintaining the evolution of objects is known as version management
- An object version represents an identifiable state of an object
Engineering design is an incremental process and evolves with time. To support this process, applications require considerable flexibility in dynamically defining and modifying the database schema.

Typical changes to the schema include:

- Changes to the class definition:
  - Modifying attributes
  - Modifying methods

- Changes to the inheritance hierarchy:
  - Making a class S the superclass of class C
  - Removing a class S from the list of superclasses of C
  - Modifying the order of the superclasses of C

- Changes to the set of classes, such as creating and deleting classes and modifying class names

Alternative Strategies for an ODBMS

- Extend an existing object-oriented programming language with database capabilities (e.g. GemStone)

- Provide extensible object-oriented DBMS libraries (e.g. Ontos and ObjectStore)

- Embed ODB language constructs in a conventional host language (e.g. O2)

- Extend and existing database language with object-oriented capabilities (Ontos, Versant O2 and provide a version of Object SQL)

- Develop a novel database data model/data language (e.g. Semantic Information Manager - SIM)
The ODBMS Manifesto - 1989

The first 8 rules apply to the object-oriented characteristics:

1 Thou shalt support complex objects
2 Thou shalt support object identity
3 Thou shalt encapsule thine objects
4 Thou shalt support types or classes
5 Thine classes or types shalt inherit from their ancestors
6 Thou shalt not bind prematurely
7 Thou shalt be computationally complete
8 Thou shalt be extensible

The ODBMS Manifesto (cont.)

The final 5 rules apply to the DBMS characteristics:

9 Thou shalt remember thy data
10 Thou shalt manage very large databases
11 Thou shalt accept concurrent users
12 Thou shalt recover from hardware and software failures
13 Thou shalt have a simple way of querying data
The ODBMS Manifesto (cont.)

**Optional features: the goodies**
- Multiple inheritance
- Type checking and type inferencing
- Distribution
- Design transactions
- Versions

**Open choices**
- Programming paradigm
- Representation system
- Type system
- Uniformity

ODBMS - Advantages

**Enriched modeling capabilities**
**Extensibility**
**Removal of impedance mismatch**
**More expressive query language**
**Schema evolution**
**Support for long duration transactions**
**Applicability to advanced database applications**
**Improved performance**
ODBMS - Disadvantages

- Lack of universal data model
- Lack of experience
- Lack of standards
- Query optimisation
- Locking
- Complexity
- Views
- Security

Extended Relational Systems

- Many vendors of RDBMS products agree that their systems are not currently suited to the advanced applications and that added functionality is required.

- Extended relational data models are based on enhancements of the relational data model to incorporate procedures, objects, versions and other new capabilities.

- There is no single extended relational model; rather, there are a variety of these models, whose characteristics depend upon the way and the degree to which extensions were made.
The Third-Generation Database Manifesto
1990

Three Tenets:

// TENET 1: Besides traditional data management services, third generation DBMSs will provide support for richer object structures and rules.

// TENET 2: Third generation DBMSs must subsume second generation DBMSs.

// TENET 3: Third generation DBMSs must be open to other subsystems.

Propositions Concerning Object and Rule Management:

1 A third-generation DBMS must have a rich type system
2 Inheritance is a good idea
3 Functions, including database procedures and methods and encapsulation, are a good idea
4 Unique identifiers for records should be assigned by the DBMS only if a user-defined primary key is not available
5 Rules (triggers, constraints) will become a major feature in future systems, They should not be associated with a specific function or collection
Propositions Concerning Increasing DBMS Function:

6 Essentially all programmatic access to a database should be through a non-procedural, high-level access language

7 There should be at least two ways to specify collections, one using enumeration of members and one using the query language to specify membership

8 Updateable views are essential

9 Performance indicators have almost nothing to do with data models and must not appear in them

Propositions that Result from the Necessity of an Open System:

10 Third generation DBMSs must be accessible from multiple high-level languages

11 Persistent forms of a high-level language, for a variety of high-level languages are a good idea. They will all be supported on top of a single DBMS by compiler extensions and a complex runtime system

12 For better or worse, SQL is ‘intergalactic dataspeak’

13 Queries and their resulting answers should be the lowest level of communication between a client and a server
Object-Oriented Database Design

Comparison between object data modeling (ODM) and logical data modeling (LDM)

<table>
<thead>
<tr>
<th>ODM</th>
<th>LDM</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Entity</td>
<td>Object includes behavior</td>
</tr>
<tr>
<td>Attribute</td>
<td>Attribute</td>
<td>None</td>
</tr>
<tr>
<td>Relationship</td>
<td>Relationship</td>
<td>Associations are the same but inheritance in ODM includes both state and behavior</td>
</tr>
<tr>
<td>Messages</td>
<td>Entity type</td>
<td>No corresponding concept in LDM</td>
</tr>
<tr>
<td>Class</td>
<td>Entity</td>
<td>None</td>
</tr>
<tr>
<td>Instance</td>
<td>Entity</td>
<td>None</td>
</tr>
<tr>
<td>Encapsulation</td>
<td></td>
<td>No corresponding concept in LDM</td>
</tr>
</tbody>
</table>

Relationships

//Relationships are represented in an ODM using reference attributes (a reference attribute contains a value, or collection of values, that are themselves objects)

//A 1:1 relationship between objects A and object B is represented by adding a reference attribute to object A and, to maintain referential integrity, a reference attribute to object B.

//A 1:M relationship between objects A and object B is represented by adding an attribute containing a set of references to each object.

//An M:N relationship between objects A and B is represented by adding an attribute containing a set of references to each object.
Referential Integrity

There are different techniques to handle referential integrity:

- **Do not allow the user to explicitly delete objects**
  - The system is responsible for ‘garbage collection’ (GemStone)

- **Allow the user to delete objects when they are no longer required**
  - The system may detect invalid references automatically and set the reference to NULL or disallow the deletion (Versant)

- **Allow the user to modify and delete objects and relationships when they are no longer required**
  - The system automatically maintains the integrity of objects (Ontos, Objectivity/DB and ObjectStore)

Identifying Methods

- **Constructors and destructors**: Constructor methods generate new instances of a class. Each new instance is given a unique OID. Destructor methods delete class instances that are no longer required.

- **Access**: Access methods return the value of an attribute or set of attributes of a class instance. It may return a single attribute value, multiple attribute values or a collection of values.

- **Transform**: Transform methods change (transform) the state of a class instance.
Further Reading

To know more about object-oriented analysis and design:

– Coad, P. and Yourdon E.: *Object-Oriented Analysis 2nd edn*, Yourdon Press/Prentice-Hall, 1991

The Object-Oriented Database System Manifesto:

http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/Manifesto.PS.gz

The Third Generation Database System Manifesto: