

COMP67321

# Introduction to Object-oriented Databases

**Goran Nenadic**

School of Computer Science  
University of Manchester

1

---

---

---

---

---

---

---

---

## Aims

- Understand the need for object oriented DBs
- Learn basic concepts and principles
- Learn the basics of ODL and OQL
- Understand advantages and disadvantages of object databases

2

---

---

---

---

---

---

---

---

## Plan

- Overview of object-oriented methodology
- Object data model (ODMG)
- Object definition language (ODL)
- Object query language (OQL)
- Object and object-relational DBs

3

---

---

---

---

---

---

---

---

## Problems with relational DBs

- Focused on traditional business applications
- Limited data types and operators
  - limited data abstraction
- Definition of additional specific types and operators is not straightforward
  - need to be embedded in application programs
- Insulation between data and operations

4

---

---

---

---

---

---

---

---

continued

## Problems with relational DBs

- Integration with OO languages
  - storage of complex objects from a Java program is not straightforward
- Need for
  - abstract data types
  - encapsulation of operations
  - inheritance
  - better integration with programming languages

5

---

---

---

---

---

---

---

---

## OODBs – introduction

- Integration of two technologies:  
OO programming and DBs
- Main ideas
  - raise the level of abstraction
  - provide direct “correspondence” between real-world and database objects
    - objects do not lose their integrity and identity and can easily be identified and operated upon

6

---

---

---

---

---

---

---

---

## OODBs – objects

- An object belongs to a specific class
- Have structure, state (value), behavior (operations), identity, name(s)
- Object structure of *arbitrary complexity*
  - in order to contain all of the necessary information
- Contrast to traditional database systems
  - information about a complex object is often scattered over many relations or records, leading to loss of direct correspondence between a real-world object and its database representation

7

---

---

---

---

---

---

---

---

## OODBs – operations

- Applied to objects of a particular type
  - includes non-standard application-specific operations
- Operations can be invoked by passing a *message* to an object
  - includes the operation name and the parameters
- Operator *polymorphism/overloading*
  - ability to be applied to different types of objects
  - operation name may refer to several distinct implementations, depending on the type of objects it is applied to

8

---

---

---

---

---

---

---

---

## Brief overview of OO concepts

- Classes and objects
- Object identity
- Encapsulation and information hiding
- Inheritance
- Polymorphism

(see additional materials on the Web)

9

---

---

---

---

---

---

---

---

**OODB standards, models and languages**

10

---

---

---

---

---

---

---

---

**Why OO standards?**

- One of the reasons for success of the relational database paradigm was the SQL standard
  - portability, interoperability
- *Object Data Management Group* proposed a standard ODMG that includes
  - Object Model (OM)
  - Object Definition Language (ODL)
  - Object Query language (OQL)
  - programming language bindings

11

---

---

---

---

---

---

---

---

**Note: ODMG**

- The ODMG is a non-profit consortium of database vendors and interested parties who collaborated to define data storage portability standards for object-oriented applications. Since its inception in 1991, the ODMG has continued to grow and today includes database vendors, tool vendors, consulting firms, and corporate end users.
- Member companies include Advanced Language Technologies, Andersen Consulting, Ardent Software, Baan, CERN, Computer Associates, Ericsson, GemStone Systems, Hitachi, IBEX Computing SA, JavaSoft/Sun Microsystems, Lockheed Martin, Microsoft, NEC Corporation, Object Design, The Object People, Objectivity, POET Software, Sybase, Telenor R&D, Versant Object Technology, and Watershed Technologies.
- The ODMG group completed its work on object data management standards in 2001 and was disbanded. The final release of the ODMG standard (ODMG 3.0, including the overall data model) can be found at <http://www.odmg.org>

12

---

---

---

---

---

---

---

---

## ODMG object model

- Provides a standard data model for object databases
- Supports object definition via ODL
- Supports object querying via OQL
- Supports a variety of data types and type constructors

13

---

---

---

---

---

---

---

---

continued

## ODMG object model

- basic concepts
  - objects
    - collections
    - atomic objects (user-defined)
  - literals
  - interface [see additional materials]
  - class
  - extent, key, factory objects

14

---

---

---

---

---

---

---

---

## ODMG objects

- Object has four characteristics
  - **identifier (OID)**: unique system-wide identifier
  - **name**: unique within a particular database and/or program; [optional]
  - **lifetime**: persistent vs. transient
  - **structure**: specifies how object is constructed by the type constructor and whether it is
    - **collection** object
    - **atomic** object

15

---

---

---

---

---

---

---

---

## ODMG built-in collections

- **Set** – unordered collection that does not allow duplicates
- **Bag** – unordered collection that allows duplicates
- **List** – ordered collection that allows duplicates
- **Array** – one-dimensional array of dynamically varying length, with direct access
- **Dictionary** – unordered sequence of key-value pairs with no duplicate keys

Notes:

- 1) all objects in a collection must be of the same type
- 2) OM defines built-in interfaces for each collection (see later)

---

---

---

---

---

---

---

---

## ODMG atomic objects

- Any object that is *not* a collection
  - typically structured complex objects, such as tuples (struct)
- Contains
  - properties: attributes and relationships
  - operations
- Attribute values can be literals or OIDs
- Note
  - atomic objects are defined via keyword *class* in ODL (see later)

17

---

---

---

---

---

---

---

---

## ODMG literals

- Literal has a current *value* but not an *identifier*
  - embedded in objects
- Three types of literals
  - **atomic**: basic (predefined) data type values (e.g., *short*, *float*, *boolean*, *char*)
  - **structured**: values that are constructed by type constructors (e.g., *date*, *time*, *struct* variables)
  - **collection**: a collection (e.g., *set*, *array*) of values or objects

18

---

---

---

---

---

---

---

---

## ODMG class

- Class is a specification of *abstract behaviour* and *abstract state* of an object
  - classes are “*instantiable*”
- Inheritance through classes allows both state and behavior inheritance, but (strictly) among classes
- Two concepts related to classes: **extent** and **key**

19

---

---

---

---

---

---

---

---

continued

## ODMG class

- **Extent** of a class
  - contains all persistent objects of that class
  - similar to creating an object of type `Set<class_name>` and making it persistent
- **Keys**
  - one or more properties whose **values** are unique for objects in an extent
  - e.g. national insurance number

20

---

---

---

---

---

---

---

---

## ODMG factory object

- Used to generate or create individual objects
- Basically, provides constructor operations for new objects
- There is an interface called `ObjectFactory`

```
interface ObjectFactory {  
    Object new ();  
};
```

`new()` returns new objects with an `object_id`

21

---

---

---

---

---

---

---

---

## ODMG database

- ODMG Object Model supports the concept of databases as storage areas for persistent objects of a given set of types (defined by OM)
- A database has a *schema* that contains a set of type definitions (see ODL)
- ODMG defines interfaces for **Database** and **DatabaseFactory** objects

22

---

---

---

---

---

---

---

---

continued

## ODMG database

- Each OO database has
  - its database **name**
  - **bind** operation (assigns individual unique names to persistent objects)
    - *named objects* are entry points to the database
  - **unbind** (removes bound names)
  - **lookup** (retrieves an object with the specified name)

23

---

---

---

---

---

---

---

---

24

---

---

---

---

---

---

---

---

## Object Definition Language (ODL)

25

---

---

---

---

---

---

---

---

## ODL

- Supports semantics constructs of ODMG
- Independent of any programming language
  - but can be easily mapped to a OO language using the specific language bindings
- Used to create object specifications (classes and interfaces)
- Not used for database manipulation

26

---

---

---

---

---

---

---

---

## ODL – class definition

```
class Name, [ extends Name2 ]
(
  extent name_of_extent
  key name_of_key
)
{
  attribute struct s_name {...} name;
  attribute string;
  ....
  relationship class_name r_name inverse name;
  ....
  void a_method();
  ...
}
```

27

---

---

---

---

---

---

---

---

## Class definition – example 1

```
class Person
( extent persons
  key ssn )
{
  attribute struct Pname (string fname, string lname) name;
  attribute string ssn;
  attribute date birthdate;
  attribute enum Gender(M, F) sex;
  attribute struct Address (short no, string street, string city,
                           string post_code) address;
  short age();
}
```

28

---

---

---

---

---

---

---

---

## Class definition – example 2

```
Class FacultyMember extends Person
(extent faculty)
{
  attribute string rank;
  attribute float salary;
  attribute string phone;

  relationship Dept works in inverse Dept::has_facultyMember;
  relationship set<GradStu> advises inverse GradStu::advisor;

  void give_raise (in float raise);
  void promote (in string new_rank);
};
```

29

---

---

---

---

---

---

---

---

## Class definition – inheritance

- Inheritance is done using **extends**
  - individual objects inherit both the properties (attributes and relationships) and operations
- Multiple class inheritance is **not** supported! (but multiple behaviour inheritance is)

30

---

---

---

---

---

---

---

---

## Class definition – relationships

- Relationships: a property that specifies that two objects are related
- Only binary relationships are explicitly represented in the object model
- **Inverse** relationship used to specify a single conceptual relationship (possibly in both ways)

```
relationship Dept works_in inverse Dept::has_facultyMember;
```

```
FacultyMember works_in Dept  
Dept has_facultyMember FacultyMember
```

31

---

---

---

---

---

---

---

---

continued

## Class definition – relationships

- Similar construction for Dept class

```
Class Dept  
( extent departments key dname )  
{  
  attribute string dname;  
  attribute FacultyMember head;  
  ....  
  relationship set<FacultyMember> has_facultyMember inverse  
    FacultyMember::works_in;  
};
```

- inverses are used to maintain referential integrity

32

---

---

---

---

---

---

---

---

continued

## Class definition – relationships

- If a relationship is to be represented only in one direction – use attributes

```
attribute FacultyMember chair;
```

- Syntax for specifying collections: collection<class>

```
Class FacultyMember extends Person  
{  
  ...  
  relationship set<GradStu> advises inverse GradStu::advisor;  
};  
A FacultyMember advises a set of GradStus
```

33

---

---

---

---

---

---

---

---

## Class definition – operations

- Operation signatures
  - name, argument types and return value
- Operation names are unique within the class/interface
- Can specify **exceptions**
  - raising exceptions during operation execution

```
void register_student(in string ssn)  
    raises(student_no_valid, section_full);
```

34

---

---

---

---

---

---

---

---

## Note: language bindings

- Specify how ODL constructs are mapped to constructs in a specific OO language
  - Smalltalk and C++ bindings defined by ODMG (see Ch 21.4 for C++)
  - ODMG Java binding has been superseded by Java Data Objects (JDO)

35

---

---

---

---

---

---

---

---



36

---

---

---

---

---

---

---

---

**Object Query Language  
(OQL)**

37

---

---

---

---

---

---

---

---

**OQL**

- ODMG's query language
- Syntax is **similar to SQL** with additional features for objects
- Works closely with programming languages such as C++
  - embedded OQL statements return objects that are compatible with the type system of the host language

38

---

---

---

---

---

---

---

---

**Basic syntax**

```
SELECT d.name
FROM d in departments
WHERE d.college = 'Engineering';
```

- An entry point to the database (e.g. named persistent object) is needed for **each** query
- Typically, an extent name (e.g., *departments*) may serve as an entry point

Recall: an extent is a persistent set of all persistent objects from a class

39

---

---

---

---

---

---

---

---

## Basic syntax: iterators

- Iterator variables (*d*) are defined whenever a collection is referenced in an OQL query
  - in the previous example, *d* ranges over each object in the collection (*departments*)
- Syntactical options for specifying an iterator:
  - d in departments*
  - departments d*
  - departments as d*

40

---

---

---

---

---

---

---

---

## Basic syntax: data type of result

- **bag**: *select...from...where...*
  - previous example: *bag<string>*
- **set**: *select distinct ...from...where...*
- **list**: *select ...from...where...order by*

41

---

---

---

---

---

---

---

---

continued

## Basic syntax: data type of result

- However, a query does not have to follow the *select...from...where...* format
- The data type of a query result can be any type defined in the ODMG model
- E.g., a persistent name on its own can serve as a query whose result is a reference to the persistent object (e.g., *departments*) whose type is *set<Departments>*

42

---

---

---

---

---

---

---

---

## OQL syntax summary

```
SELECT [DISTINCT] <expression>
FROM      <fromList>
[WHERE    <expression>]
[GROUP BY <attribute1:expression1, attribute2:expression2,...>]
[HAVING  <predicate>]
[ORDER BY <expression>]
```

where:

```
<fromList> ::= <variableName> IN <expression>|
<variableName> IN <expression>,<fromList>|
<expression> AS <variableName>|
<expression> AS <variableName>,<fromList>
```

43

---

---

---

---

---

---

---

---

## Basic syntax: path expressions

- A path expression is used to specify a path to attributes and objects in an entry point
- A path expression starts at a persistent object name (or its iterator variable)
- The name will be followed by zero or more dot connected relationship or attribute names

informatics\_dept.chair

(assuming that there is a persistent object named informatics\_dept)

44

---

---

---

---

---

---

---

---

continued

## Basic syntax: path expressions

- examples

```
select f.rank
from f in informatics_dept.has_facultyMember
```

```
select distinct f.rank
from f in informatics_dept.has_facultyMember
```

FacultyMember

set<FacultyMember>

45

---

---

---

---

---

---

---

---

## Example class (Department)

```
Class Department
( extent departments key dname )
{
  attribute string dname;
  attribute FacultyMember head;
  ....
  relationship set<FacultyMember> has_facultyMember inverse
    FacultyMember::works_in;
  relationship set<Students> has_major inverse
    Student::majors_in;
};
```

46

---

---

---

---

---

---

---

---

## Example class (Student)

```
class Student extends Person
( extent students )
{
  attribute string class;
  attribute Department minors_in;
  relationship Department majors_in inverse Department::has_majors;
  relationship set<Grade> completed_sections inverse Grade::student;
  relationship set<CurSection> registered_in
    inverse CurSection::registered_students;
  void change_major(in string dname) raises(dname_not_valid);
  float gpa();
  void register(in short secno) raises(section_not_valid);
  void assign_grade(in short secno; in GradeValue grade)
    raises(section_not_valid,grade_not_valid);
};
```

47

---

---

---

---

---

---

---

---

## Example

```
select struct( last_name: s.name.lname,
              first_name: s.name.fname,
              gpa: s.gpa),
from s in informatics_dept.has_majors
where s.class = 'senior'
order by gpa desc, last-name asc, first_name asc;
```

Retrieve last and first names plus GPA (average) of all 'senior' (i.e. final year) students in the InformaticsDept; order the results by gpa, and then by names

48

---

---

---

---

---

---

---

---

## Example

```
select struct( last_name: s.name.lname,  
              first_name: s.name.fname,  
              gpa: s.gpa),  
from s in students  
where s.majors_in.dname = 'Informatics' and  
      s.class = 'senior'  
order by gpa desc, last-name asc, first_name asc;
```

Retrieve last and first names plus GPA (average) of all 'senior' (i.e. final year) students in the InformaticsDept; order the results by gpa, and then by names

---

---

---

---

---

---

---

---

## Example – for you

```
class Student  
(extent students key student_id)  
{  
  attribute string name;  
  attribute int student_id;  
  attribute int year;  
  relationship <Dept> studies_in inverse Dept::has_students;  
  relationship <Staff> has_tutor inverse Staff::tutees;  
}
```

---

---

---

---

---

---

---

---

## Example – for you

```
class Staff  
(extent staff key staff_id)  
{  
  attribute string name;  
  attribute int staff_id;  
  attribute string position;  
  
  ...  
  relationship set<Student> tutees  
    inverse Student::has_tutor;  
}
```

---

---

---

---

---

---

---

---

## Example – for you

- 1) Create a Dept class.
- 2) Retrieve names of all students in the InformaticsDept whose tutor is a senior lecturer.

```
select s.name
from s in informaticsDept.has_students
where ...
```

52

---

---

---

---

---

---

---

---

## Views as named objects

- The *define* keyword in OQL is used to specify an identifier for a named query
  - name should be unique; if not, the results will replace an existing named query
- Once a query definition is created, it will persist until deleted or redefined
- A view definition can include parameters

53

---

---

---

---

---

---

---

---

continued

## Views as named objects

- A view to include students in a department who have a minor (in that department):

```
define has_minor(dept_name) as
select s
from s in students
where s.minor_in.dname = dept_name
```

```
has_minor can now be used in queries:
has_minor('Informatics') – returns bag of students
```

54

---

---

---

---

---

---

---

---

## Aggregate operators

- OQL supports a number of aggregate operators (that operate over a collection) that can be applied to query results
- *min*, *max*, *count*, *sum*, and *avg*
- *count* returns an integer; others return the same type as the collection type

```
count (s in has_minors('Informatics'));  
count (select f.name from f in faculty where f.sex = F);
```

55

---

---

---

---

---

---

---

---

continued

## Aggregate operators

```
avg (select s.gpa  
from s in students  
where s.class = 'senior' and  
s.majors_in.dname = 'Business');
```

56

---

---

---

---

---

---

---

---

## Membership and quantification

- (**e in c**) is true if e is in the collection c

```
'Informatics' in (select d.dname from d in departments)
```

```
select s.name.fname, s.name.lname  
from s in students  
where 'COMP67321' in  
(select c.cname  
from c in s.completed_sections.section.of_course);
```

57

---

---

---

---

---

---

---

---

## Membership and quantification

- **(for all e in c: b)** is true if all e elements of collection c satisfy b
- **(exists e in c: b)** is true if at least one e in collection c satisfies b

```
for all g in (select s from s in grad_students
             where s.majors_in.dname = 'Informatics')
: g.advisor in InformaticsDept.has_faculty;
```

---

---

---

---

---

---

---

---

## Grouping operator

```
select deptname, avg_gpa : avg (select p.s.gpa
                               from p in partition)
from s in students
group by deptname: s.majors_in.dname
having count (partition) > 100;
```

attribute expression

Retrieve average GPA of majors in each department having >100 majors

---

---

---

---

---

---

---

---

## Examples – embedded query

```
select struct (name: struct( last_name: s.name.lname,
                           first_name: s.name.fname),
             degrees: (select struct (deg: d.degree,
                                     yr: d.year,
                                     col: d.college)
                       from d in s.degrees)
             )
from s in informatics_dept.chair.advises;
```

Retrieve last and first names plus degree information for all students advised by the head of InformaticsDept

---

---

---

---

---

---

---

---

**Designing Object Databases**

61

---

---

---

---

---

---

---

---

**OODBs vs. RDBs: relations**

- OODB
  - relationships are handled by reference attributes that include OIDs of related objects
  - single and collection of references are allowed
  - references for binary relationships can be expressed in a single direction or both directions via *inverse* operator
    - if in both directions: redundancy, and hence inconsistency
    - sometimes better to create a separate class to represent the relationship (cf. Grade, Student, Section classes)

62

---

---

---

---

---

---

---

---

continued

**OODBs vs. RDBs: relations**

- RDB
  - relationships among tuples are specified by attributes with matching values (via *foreign keys*)
  - foreign keys are single-valued
  - M:N relationships must be presented via a separate relation (table)

63

---

---

---

---

---

---

---

---

## OODBs vs. RDBs: inheritance

- Inheritance structures are built in OODB (via ":" and `extends` operators)
- RDB has no built-in support for inheritance relationships
  - object-relational databases

64

---

---

---

---

---

---

---

---

## OODBs vs. RDBs: design

- the specification of operations
  - OODB: operations specified during design (as part of class specification)
  - RDB: may be delayed until implementation

65

---

---

---

---

---

---

---

---

## Mapping EER to OODB schema

- Mapping EER schemas into OODB schemas is *relatively* simple especially since OODB schemas provide support for inheritance relationships
- Once mapping has been completed, **operations must be added** to OODB schemas since EER schemas do not include an specification of operations

66

---

---

---

---

---

---

---

---

**Current status and conclusions**

67

---

---

---

---

---

---

---

---

**Current status**

- OODB market around \$250M as opposed to the relational DB revenue of \$20-50B
- OODBMS were originally thought of to **replace** RDBMS because of their better fit with object-oriented programming languages. However, high switching cost and the inclusion of object-oriented features in RDBMS to make them "object-relational" have made RDBMS successfully defend their dominance.

68

---

---

---

---

---

---

---

---

continued

**Current status**

- Object databases are now established as a **complement** rather than a replacement for relational databases.
- OO ideas are being used in a large number of applications, without explicitly using the OODB platform to store data
  - OO tools for modelling and analysis, OO programming languages
  - object relational DBMS

69

---

---

---

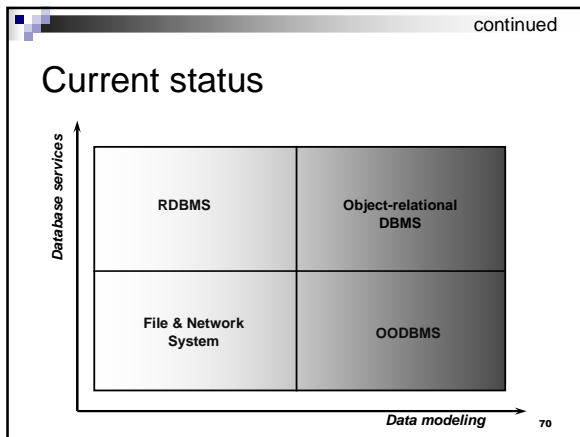
---

---

---

---

---




---

---

---

---

---

---

---

---

## Current status – ORDBs

- An object-relational database (ORDB) is a relational database that allows developers to integrate the database with their own custom data types and methods.
- Also, software running over traditional DBMSs to provide similar features
- The goal of ORDBMS technology is to allow developers to raise the level of abstraction at which they view the problem domain.
- Object-oriented features added to SQL-99

71

---

---

---

---

---

---

---

---

continued

## Current status – ORDBs

- SQL-99 includes
  - some type construction (*row, array, set, list*)
  - object identity (through reference types)
  - encapsulation of operations
  - inheritance
  - new data types (BLOBs, CLOBs)
- Many vendors provide specific type extensions for newer data type (e.g. **cartridges** in Oracle)
- Other vendors: Informix Universal Server, IBM's UDB, DB2/II

72

---

---

---

---

---

---

---

---

## Current status – OODB products

### ■ Commerical

db4objects (db4o)  
GemStone (GemStone/S, Facets)  
InterSystems (Caché)  
Matisse (Matisse)  
ObjectDB (ObjectDB)  
Objectivity (Objectivity/DB)  
Progress (ObjectStore, PSE Pro)  
Versant (merged with Poet; VDS, FastObjects)  
W3apps (Jeevan)

### ■ Open source

db4o  
Ozone  
Perst  
Zope (ZODB)

73

---

---

---

---

---

---

---

---

## Summary: OODB approach

### ■ User-defined **data types** and **operations**

- this is part of the database definition
- direct “correspondence” between real-world and database objects
- abstract data types, inheritance, encapsulation

### ■ Integration and compatibility with OO languages

- OODBs provide **persistent** and **straightforward storage** for program objects and data structures (not straightforward in RDBs)
- objects do not lose their integrity and identity

74

---

---

---

---

---

---

---

---

continued

## Summary: OODB approach

### ■ main concepts/advantages (summary)

- object identity
- support for complex object structures
- encapsulation of properties and operations
- type hierarchies and behaviour inheritance
- polymorphism
- programming language compatibility (storage of objects and ODL/OQL bindings)

75

---

---

---

---

---

---

---

---

## Summary: OODB approach

- Issues
  - OM is more complex than relational model
    - can represent more complex data structures than the simple relational representation, but more difficult to design
    - representation of non-binary relationships not trivial
  - no early standards, so limited use, popularity, portability
    - low investment in non-standard technologies
    - RDBs are traditionally much more used
  - OQL needs named entities
- Object-relational databases as a compromise

---

---

---

---

---

---

---

---

## Summary

- Overview of OO methodology
- Object model and various constructs and built-in types of the ODMG model presented
- ODL and OQL languages were presented
- Current state of OODBs/ORDBs, advantages and disadvantages discussed

---

---

---

---

---

---

---

---

## Reading for this lecture

- Chapters 20 and 21 in the textbook
- Further information/optional reading:
  - Object relational DBs: Chapter 22
- <http://www.odbms.org/>, <http://www.odmg.org/>

---

---

---

---

---

---

---

---