COS221
L18 - OO Models

(Chapter 11 in Edition 6 and Chapter 12 in Edition 7)

Linda Marshall

3 April 2023

Object Databases

- Object Databases (ODBs), previously referred to as Object-Oriented Databases (OODBs), were born out of the need to persist objects from the object-oriented model. That is, the need to store data from applications that required more complex structures and relationships.
- ODBs give the database users the power to manipulate structure as well as behaviour (operations).
- Relational database vendors realised the need to incorporate OO features into relational database systems resulting in the Object-Relational database systems.

Object Database Concepts

- An object typically has two components:
  - **State** - values defined using instance variables
  - **Behaviour** - operations
- **Inheritance** - promotes reuse
- **Object Identifier**
- **Persistence** - An object database persists objects, either simple or complex.

Object Database Concepts - State

Complex types are constructed from other types by nesting type constructors. There are 3 basic type constructors:-

- Atom - includes the built-in types. The types are single-valued or atomic types.
- Struct (or tuple) - used to create (generate) structured types which comprise of several components. Sometimes referred as a compound or composite type.
- Collection (or multivalued) - includes sets, lists, bags, arrays and dictionaries.

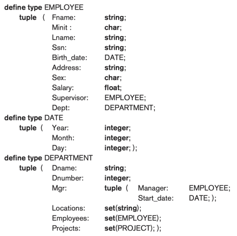
An Object Definition Language (ODL) can be used to define types for a particular database application.

Object Database Concepts - State

```
define type EMPLOYEE
     tuple  (  Fname:          string;
               Minit :         char;
               Lname:          string;
               Ssn:            string;
               Birth_date:     DATE;
               Address:        string;
               Sex:            char;
               Salary:         float;
               Supervisor:     EMPLOYEE;
               Dept:           DEPARTMENT;
define type DATE
     tuple  (  Year:           integer;
               Month:          integer;
               Day:            integer; );
define type DEPARTMENT
     tuple  (  Dname:          string;
               Dnumber:        integer;
               Mgr:            tuple  (  Manager:      EMPLOYEE;
                                         Start_date:   DATE; );
               Locations:      set(string);
               Employees:      set(EMPLOYEE);
               Projects:       set(PROJECT); );
```

Figure 12.1
Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

Object Database Concepts - State



```
define type EMPLOYEE
    tuple  (  Fname:        string;
              Minit :       char;
              Lname:        string;
              Ssn:          string;
              Birth_date:   DATE;
              Address:      string;
              Sex:          char;
              Salary:       float;
              Supervisor:   EMPLOYEE;
              Dept:         DEPARTMENT;
define type DATE
    tuple  (  Year:         integer;
              Month:        integer;
              Day:          integer; );
define type DEPARTMENT
    tuple  (  Dname:        string;
              Dnumber:      integer;
              Mgr:          tuple  (  Manager:      EMPLOYEE;
                                      Start_date:   DATE; );
              Locations:    set(string);
              Employees:    set(EMPLOYEE);
              Projects:     set(PROJECT); );
```

Figure 12.1
Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

- Instance variables can be defined as visible or hidden.
  - Visible instance variables are accessible from outside the object via a query language.
  - Hidden variables are only accessible through the operations defined.

Object Database Concepts - Behaviour

To promote encapsulation, operations are defined in two parts.

- A signature - operation name and arguments (parameters), and
- The method (body) providing the implementation.

Passing a message to an object means an operation is invoked by calling the operation name with parameters.

Object Database Concepts - Behaviour

- Operator overloading enables reuse through applying an operation to different types of objects.
- Unlike with relational databases where operations (SELECT, INSERT etc.) are defined and can access all variables, operations of ODBs are encapsulated in the object and provide a means to query hidden instance variables.
- A signature for an operation is defined when an object structure (class) is specified. The implementation is given elsewhere.

Object Database Concepts - Behaviour

Typical operations include:

- object constructor - creates the object
- destructor - destroys the object
- object modifier - change the values of instance variable of the object
- operations to retrieve information about the object

Object Database Concepts - Behaviour

```
define class EMPLOYEE
    type tuple (  Fname:          string;
                  Minit:          char;
                  Lname:          string;
                  Ssn:            string;
                  Birth_date:     DATE;
                  Address:        string;
                  Sex:            char;
                  Salary:         float;
                  Supervisor:     EMPLOYEE;
                  Dept:           DEPARTMENT; );
    operations    age:            integer;
                  create_emp:     EMPLOYEE;
                  destroy_emp:    boolean;
end EMPLOYEE;
define class DEPARTMENT
    type tuple (  Dname:          string;
                  Dnumber:        integer;
                  Mgr:            tuple ( Manager:       EMPLOYEE;
                                          Start_date:    DATE; );
                  Locations:      set (string);
                  Employees:      set (EMPLOYEE);
                  Projects        set(PROJECT); );
    operations    no_of_emps:     integer;
                  create_dept:    DEPARTMENT;
                  destroy_dept:   boolean;
                  assign_emp(e: EMPLOYEE): boolean;
                  (* adds an employee to the department *)
                  remove_emp(e: EMPLOYEE): boolean;
                  (* removes an employee from the department *)
end DEPARTMENT;
```

Figure 12.2
Adding operations to the definitions of EMPLOYEE and DEPARTMENT.

Object Database Concepts - Type Hierarchies and Inheritance

- Inheritance provides a definition of new types based on other predefined types. It leads to type (or class) hierarchy
- A type is defined by a type name and list of visible (public) functions:

  ```
      TYPE_NAME: function, function, ..., function
  ```

  For example:

  ```
  PERSON: Name, Address, Birth_date, Age, Ssn
  ```

- A subtype is useful when creating a new type that is similar but not identical to an already defined type, for example subtypes of the supertype person:

  ```
  EMPLOYEE subtype-of PERSON:
      Salary, Hire_date, Seniority
  STUDENT subtype-of PERSON:
      Major, Gpa
  ```

Object Database Concepts - Type Hierarchies and Inheritance

- The extent stores a collection of persistent objects for each type or subtype. Extents are subsets of the extent of class OBJECT.
- A *persistent collection* is stored permanently in the database.
- The *transient collection* exists temporarily during the execution of an application.

Object Database Concepts - Type Hierarchies and Inheritance

Polymorphism and Multiple Inheritance

- Polymorphism of operations (operator overloading), allows the same operator name or symbol to be bound to two or more different implementations. The operator called depends on the type of objects to which operator is applied
- Multiple inheritance is where a subtype inherits functions (attributes and methods) of more than one supertype.
- With selective inheritance a subtype inherits only some of the functions of a supertype.

Object Database Concepts - Object Identifier (OID)

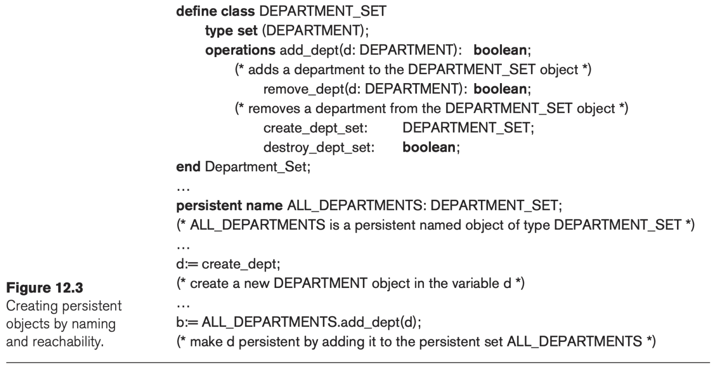
- To maintain a direct correspondence between real-world and database objects and not loose identity and integrity, a unique identifier to assigned to an object stored in a database.
- The Object Identifier (OID) is system generated and provides this unique identity.
- The main property of the OID is that it must be immutable, that is, the value linked to a specific object may not change.
- Once an OID has been assigned and used for an object, it may not used for another object.
- The OID may or may not be visible to user of the database system.
- ODBs allow for the representation of both objects and literals. Unlike objects, literals do not have id's.

Object Database Concepts - Persistence

- A distinction is made between transient (not stored in the database, only exists for the lifetime of the application) and persistant (stored in the database) objects.
- Two mechanisms exist for making an object persistent:
  - **naming** - assign a unique persistent name to an object within a database. Named persistent objects are used as entry points to the database by application programs. Naming is used for a select few objects and not all objects in the database.
  - **reachability** - objects are made available through another persistent object. An object B is reachable from a persistent object A, if there is a path (sequence of references) through the database from A to B.

"If we first create a named persistent object N, whose state is a set of objects of some class C, we can make objects of C persistent by adding them to the set, thus making them reachable from N. Hence, N is a named object that defines a persistent collection of objects of class C. In the object model standard, N is called the extent of C."

Object Database Concepts - Persistence



```
define class DEPARTMENT_SET
    type set (DEPARTMENT);
    operations add_dept(d: DEPARTMENT):   boolean;
        (* adds a department to the DEPARTMENT_SET object *)
            remove_dept(d: DEPARTMENT):  boolean;
        (* removes a department from the DEPARTMENT_SET object *)
            create_dept_set:        DEPARTMENT_SET;
            destroy_dept_set:       boolean;
end Department_Set;
...
persistent name ALL_DEPARTMENTS: DEPARTMENT_SET;
(* ALL_DEPARTMENTS is a persistent named object of type DEPARTMENT_SET *)
...
d:= create_dept;
(* create a new DEPARTMENT object in the variable d *)
...
b:= ALL_DEPARTMENTS.add_dept(d);
(* make d persistent by adding it to the persistent set ALL_DEPARTMENTS *)
```

Figure 12.3
Creating persistent objects by naming and reachability.

Object Model Standard

- Object Data Management Group (ODMG) standard
  - First proposed in 1993
  - Current version is ODMG 3.0 (2006)
- Comprises of the following parts:
  - Object model
  - Object definition language (ODL)
  - Object query language (OQL)
  - Bindings to object-oriented programming languages
- OMG has taken over the management of the standard and included in the work by ODBMS.org

ODMG - Object Model

An object has:

- Mandatory unique system-wide *Identifier* - `Object_id`
- Optional *name*
- A *lifetime*, either transient or persistent
- *Structure*, atomic or constructed out of other objects. Built in data-types are referred to as literals (Refer to Fig 12.5)
  - Atomic - basic types
  - Structured - created using STRUCT. Examples are Date, Time...
  - Collection - set, bag, list, array, dictionary
- Manner in which the object is *created*

All objects adhere to a standard interface as defined by `Object` that provides first class semantics (construction, assignment, copy and delete)

ODMG - Object Model

Inheritance is either

- Behaviour only (ISA). Specified with the colon (:) notation. Superclass is required to be an interface and the subclass either an interface or a class
- Behaviour and state (EXTENDS). Super- and sub-classes must be classes.

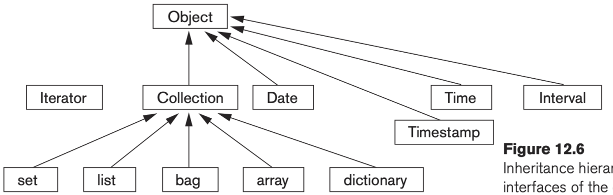


Figure 12.6
Inheritance hierarchy for the built-in interfaces of the object model.

ODMG - ODL

```
class EMPLOYEE
(     extent            ALL_EMPLOYEES
      key               Ssn   )
{
      attribute         string              Name;
      attribute         string              Ssn;
      attribute         date Birth_date;
      attribute         enum Gender{M, F}   Sex;
      attribute         short               Age;
      relationship      DEPARTMENT          Works_for
                            inverse DEPARTMENT::Has_emps;
      void              reassign_emp(in string New_dname)
                            raises(dname_not_valid);
};
class DEPARTMENT
(     extent            ALL_DEPARTMENTS
      key               Dname, Dnumber )
{
      attribute         string              Dname;
      attribute         short               Dnumber;
      attribute         struct Dept_mgr {EMPLOYEE Manager, date Start_date}
                            Mgr;
      attribute         set<string>         Locations;
      attribute         struct Projs {string Proj_name, time Weekly_hours)
                            Projs;
      relationship      set<EMPLOYEE>       Has_emps inverse EMPLOYEE::Works_for;
      void              add_emp(in string New_ename) raises(ename_not_valid);
      void              change_manager(in string New_mgr_name; in date
                            Start_date);
};
```

Figure 12.7
The attributes, relationships, and operations in a class definition.

ODMG - ODL

EER to ODL Mapping

1. Create ODL class for each EER entity type - Multivalued attributes: declared by using the set, bag, or list constructors
2. Add relationship properties for each binary relationship
3. Include appropriate operations for each class
4. ODL class that corresponds to a subclass in the EER schema - Inherits type and methods of its superclass in ODL schema
5. Weak entity types - Mapped same as regular entity types
6. Categories (union types) - Difficult to map to ODL
7. An n-ary relationship with degree $n > 2$ - Map into a separate class, with appropriate references to each participating class

ODMG - ODL

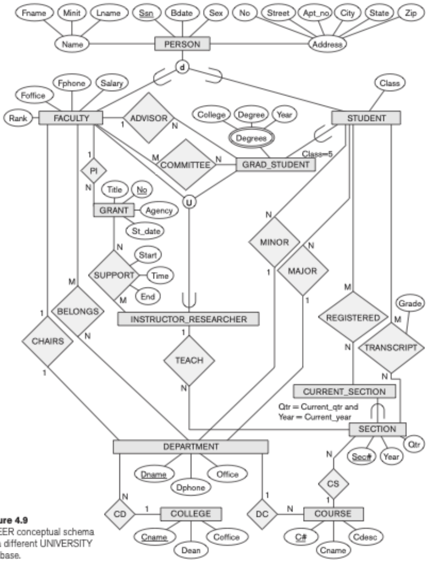


Figure 4.9
An EER conceptual schema for a different UNIVERSITY database.

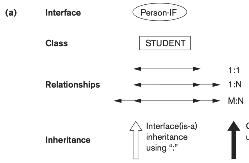
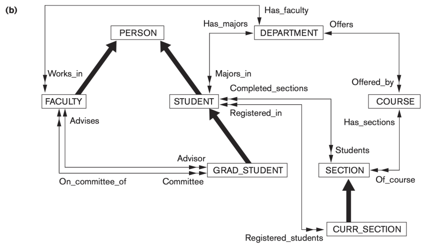

(a)
Interface
Person-IF
Class
STUDENT
Relationships
1:1
1:N
M:N
Inheritance
Interface(is-a) inheritance using ":"
Class inheritance using **extends**

Figure 12.9
An example of a database schema.
(a) Graphical notation for representing ODL schemas. (b) A graphical object database schema for part of the UNIVERSITY database (GRADE and DEGREE classes are not shown).

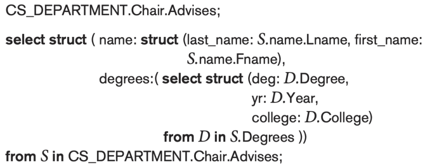


ODMG - OQL

- The OQL syntax mimics SQL.

  ```
  select D.Dname
  from D in DEPARTMENTS
  where D.College = 'Engineering';
  ```

- Returning complex structure



```
CS_DEPARTMENT.Chair.Advises;

select struct ( name: struct (last_name: S.name.Lname, first_name:
                              S.name.Fname),
                degrees:( select struct (deg: D.Degree,
                                         yr: D.Year,
                                         college: D.College)
                          from D in S.Degrees ))
from S in CS_DEPARTMENT.Chair.Advises;
```

The first query returns all the grad student who are advise by the CS chair, that is an object of type: `set<GRAD_STUDENT>`. The second query includes the names and the degrees of the grad students.

ODMG - Bindings

- For C++
  - a C++ library has been written in which template classes are specified
  - prefixed with `d_`
  - To create a persistent object:
    `D_Ref<STUDENT> S = new(DB1, 'John_Smith') STUDENT;`
  - Binding:
    `D_Extent<PERSON> ALL_PERSONS(DB1);`
- Java and Smalltalk bindings also exist