Object and Relational-Object Databases

Dr. Rabah Mokhtari

Department of Computer Science,
UNIVERSITY OF M'SILA

October 30, 2022

Content

Introduction

Why an Object Database?

Database for more complex applications

Traditional data models and systems, such as relational are not suitable for more complex database applications like

- Engineering design and manufacturing.
- Telecommunications.
- Geographic information systems.
- Multimedia

Object databases (**ODB**s) were proposed to meet some of the needs of these more complex applications.

Object Database Vs OO Programming Language (OOPL)

A key feature

A key feature of object databases is the power they give the designer to specify both the **structure** of complex objects and the **operations** that can be applied to these objects.

ODB Vs OOPL

- Another reason for the creation of ODBs is the vast increase in the use of OOPLs for developing software applications.
- ODBs are designed so they can be directly used to store and retrieve objects of those application.

Object-Relational DB (ORDB)

ORDB and ORDBMS

- The new version of Relational DBMS (RDBMS) vendors have also recognized the need for incorporating features that were proposed for object databases.
- This has led to database systems that are characterized as ORDBs (Object-Relational DB) and ORDBMSs (Object-Relational DBMSs).

SQL in ORDBMSs

- The latest version of the SQL standard (2008) for RDBMSs includes many features required to query ORDBs.
- Originally known as SQL/Object and they have now been merged into the main SQL specification, known as SQL/Foundation.

Prototypes and commercial ODB systems

ODBMSs

- **Orion system –** developed at MCC (Microelectronics and Computer Technology Corporation, Austin, Texas)
- **OpenOODB –** at Texas Instruments
- **Iris system –** at Hewlett-Packard laboratories.
- **InterSystems Cach**
- etc.

Object technology

Object Technology

Object characteristics

- Object technology was developed to represent the real world with software.
- The first OO language was Simula, developed in the 1960s in Norway.
- Objects in Simula appeared like actors possessing their own knowledge (i.e., data) and capabilities (algorithms or methods).
- Objects are modular. The Software system is segmented into easy understandable independent units.
- Objects are reusable, meaning that can be reused in applications that require the same units.
- through the named persistent objects users and applications can start their database access.

Object, Properties, Methods, and Messages

Object as an encapsulated unit

- The most important concept in object technology is the **object** itself.
- This describes an encapsulated unit with properties and methods.
- **Encapsulation** means that the internal implementation is covered by a **secrecy principle**.

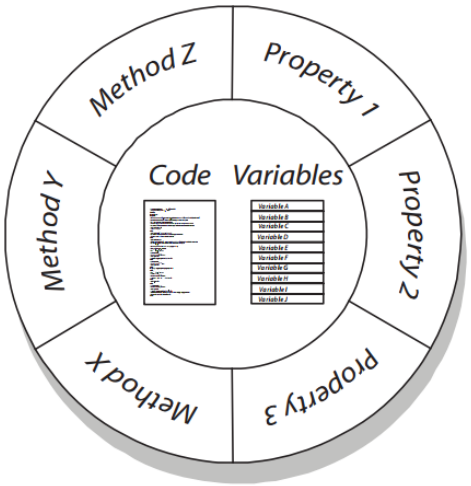


Figure: An object

Object, Properties, Methods, and Messages

Message for executing methods

- To execute a method, you cannot simply invoke its code, because this would violate the secrecy principle.
- Instead, you send the object a **message** requesting to execute a specific method.
- The message includes any parameters the method may require.
- The object itself decides what internal code to execute, invokes it, and passes parameter values appropriately.

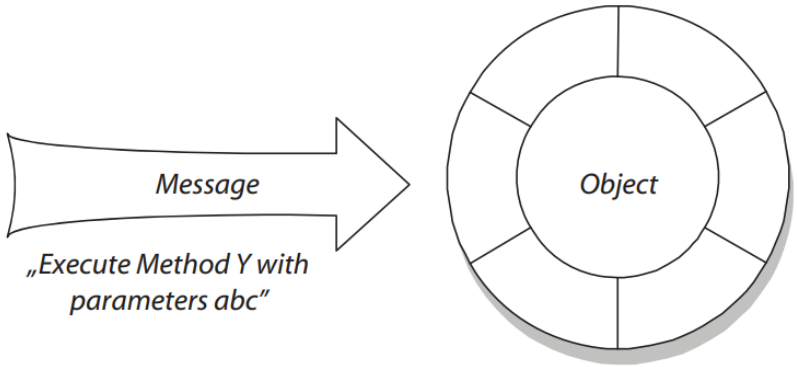


Figure: Sending a message to an object

Object Classes and Instances

Object Classes and Instances

- An object system usually contains a number of similar objects, which are grouped into so-called object **classes**.
- The incarnations of these object classes (the individual objects) are called **instances** of the class.
- The class defines properties and methods common to all instances.

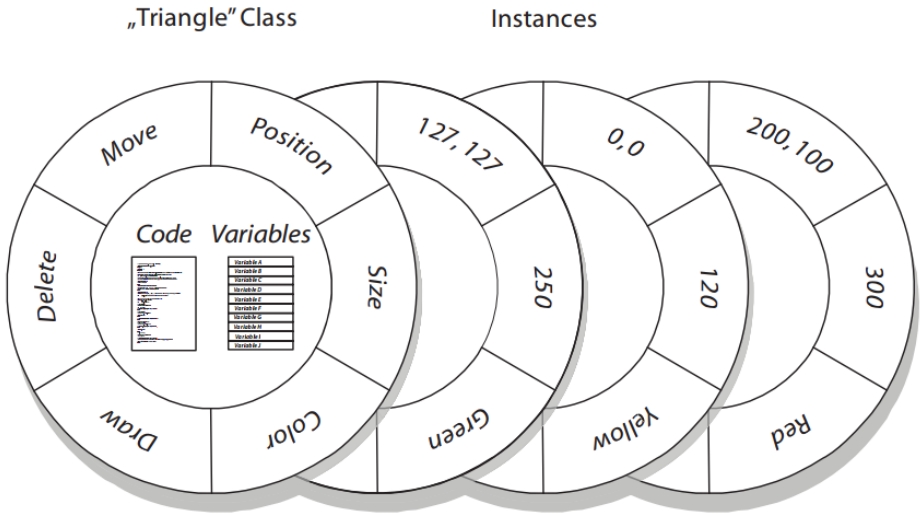


Figure: A “triangle” class and its instances

Class Hierarchies and Inheritance

Inheritance, Substitutability, and Specialization

- **Inheritance** means that one object class is derived from another.
- A subclass has an "**is a**" relationship to the super class from which it inherits all its properties and methods.
- **substitutability –** the subclass has the same interface as the super class and can be used in its place. It means that every message you can send to an instance of the super class is also valid for instances of the subclass.
- **Specialization of subclasses –** The subclass can be specialized by adding properties and methods that the super class does not have. Similarly, you can overwrite methods so the subclass behaves differently than the super class

Class Hierarchies and Inheritance (Example)

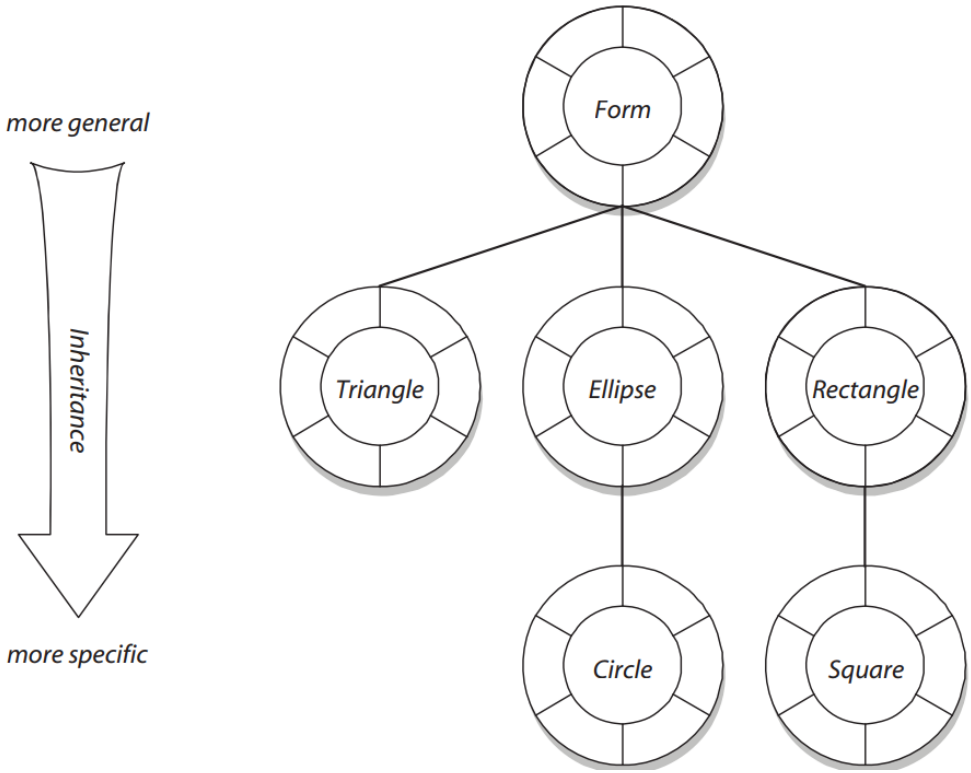


Figure: Inheritance in a class hierarchy

Abstract and Final Classes

Abstract class

- The super class "**form**" contains a general definition of geometric forms. It would not make much sense to create instances of this without knowing the associated form.
- You can designate a class as **abstract**, meaning that it can serve only as a template for defining subclasses.
- Abstract classes cannot have instances.

Final class

Similarly, there are some classes from which no further subclasses should be created. These are designated as **final**.

Persistence and Other ODB Requirements

Object Naming and reachability

Transient vs Persistent Objects

- Not all objects are meant to be stored permanently in the database (Two types).
  1. **Transient** objects exist in the executing program and disappear once the program terminates.
  2. **Persistent** objects are stored in the database and persist after program termination.
- The typical mechanisms for making an object persistent are naming and reachability.

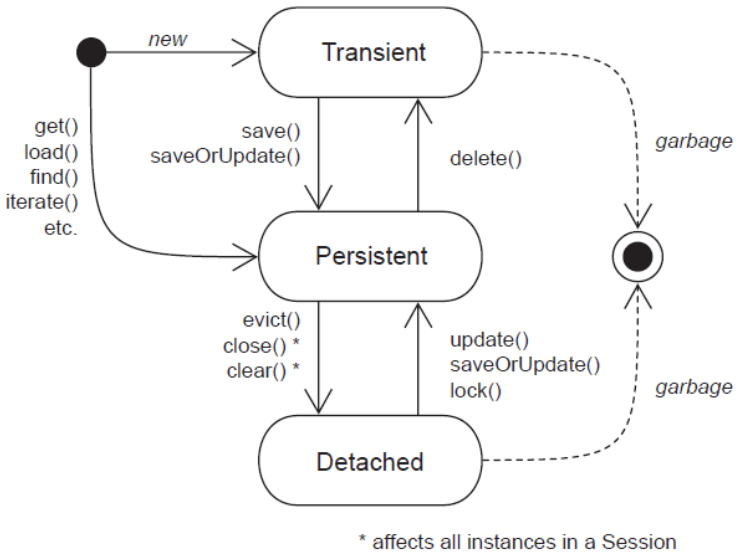


Figure: hibernate object lifecycle

Object Naming and reachability

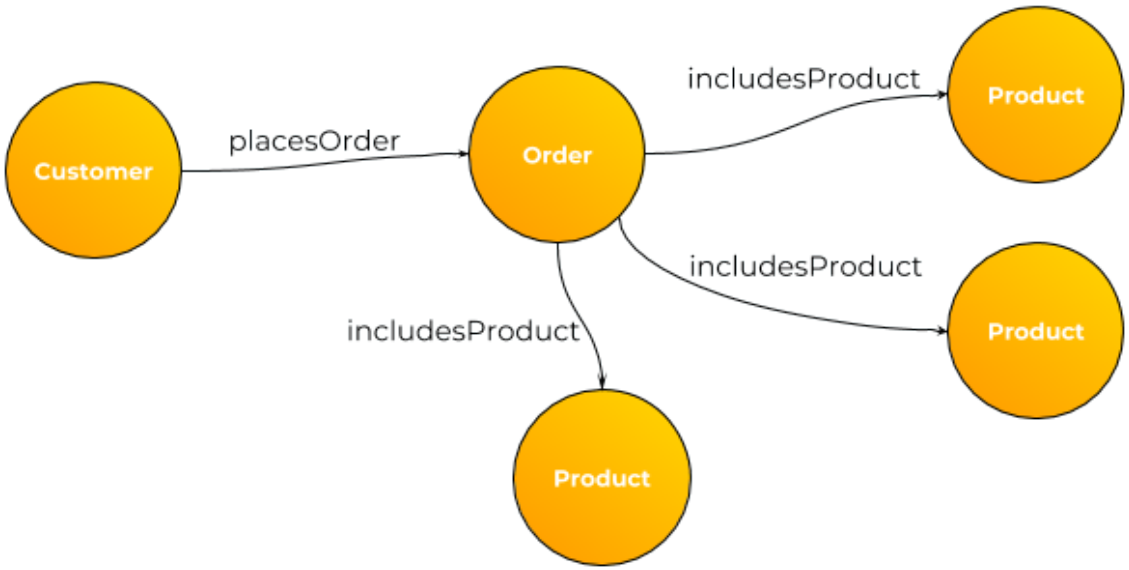
Naming

- The **naming mechanism** involves giving an object a unique persistent name within a particular database
- This persistent **object name** can be given via a specific statement or operation in the program

Object Naming and reachability

reachability

- The reachability mechanism works by making the object reachable from some other persistent object.
- An object B is said to be reachable from an object A if a sequence of references in the database lead from object A to object B.



Object Naming and reachability

Extent objects

- We can create a persistent Object of state **set** or **bag** to store object of class C.
- We can make objects of C persistent by adding them to the set, thus making them reachable from N.
- In the object model standard, N is called the **extent** of C.

Object-Relational Features

Object Database Extensions to SQL

Revision of SQL

- Starting with SQL3, features from object databases were incorporated into the SQL standard.
- At first, these extensions were known as SQL/Object.
- Later they were incorporated in the main part of SQL, known as SQL/Foundation.
- The **relational model** with object database enhancements is sometimes referred to as the **object-relational model**.
- Additional revisions were made to SQL in 2003 and 2006 to add features related to XML.

Object Database SQL-included features

Type constructors, Encapsulation of operations, and Inheritance

- Some type constructors have been added to specify complex objects including
  1. row type, which corresponds to the tuple (or struct) constructor
  2. An array type for specifying collections is also provided.
  3. Other collection type constructors, such as set, list, and bag constructors were later included in the standard.
- Encapsulation of operations is provided through the mechanism of user-defined types (UDTs) that may include operations as part of their declaration.
- Inheritance mechanisms are provided using the keyword UNDER.

User-Defined Types and Complex Structures for Objects

Complex structured objects

- To allow the creation of complex-structured objects SQL now provides user-defined types (UDTs).
- The user will create the UDTs for a particular application as part of the database schema.

User-Defined Types and Complex Structures for Objects

```
(a) CREATE TYPE STREET_ADDR_TYPE AS (
        NUMBER          VARCHAR (5),
        STREET_NAME     VARCHAR (25),
        APT_NO          VARCHAR (5),
        SUITE_NO        VARCHAR (5)
    );
    CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     STREET_ADDR_TYPE,
        CITY            VARCHAR (25),
        ZIP             VARCHAR (10)
    );
    CREATE TYPE USA_PHONE_TYPE AS (
        PHONE_TYPE      VARCHAR (5),
        AREA_CODE       CHAR (3),
        PHONE_NUM       CHAR (7)
    );

(b) CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR (35),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE ARRAY [4],
        ADDR            USA_ADDR_TYPE
    INSTANTIABLE
    NOT FINAL
    REF IS SYSTEM GENERATED
    INSTANCE METHOD AGE() RETURNS INTEGER;
    CREATE INSTANCE METHOD AGE() RETURNS INTEGER
        FOR PERSON_TYPE
        BEGIN
            RETURN /* CODE TO CALCULATE A PERSON'S AGE FROM
                      TODAY'S DATE AND SELF.BIRTH_DATE */
        END;
    );
```

Figure: User-Defined Types for Complex Structured Objects

Creating Tables Based on the UDTs

Generated Object Identifier

- Unique system-generated object identifiers can be created via the reference type in the latest version of SQL.
- **REF IS SYSTEM GENERATED** indicates that whenever a new **PERSON_TYPE** object is created, the system will assign it a unique system-generated identifier.
- It is also possible not to have a system-generated object identifier and use the traditional keys of the basic relational model if desired.

INSTANTIABLE class

For each UDT that is specified to be instantiable via the phrase **INSTANTIABLE** one or more tables may be created.

Encapsulation of Operations

SQL defined operations

- In SQL, a UDT can have its own behavioral specification by specifying operations.
- For example a declared method Age() that calculates the age of an individual object of type PERSON_TYPE.
- SQL provides certain built-in functions for UDTs. (eg. the constructor function TYPE_T( ) returns a new object of that type).
- An observer function A is implicitly created for each attribute A to read its value.
- Hence, A(X) or X.A returns the value of attribute A.

Inheritance and Overloading of Functions

Operation inheritance

- In general, both attributes and instance methods (operations) are inherited.
- The phrase **NOT FINAL** must be included in a UDT if subtypes are allowed to be created **under** that UDT.
- For example, PERSON_TYPE, STUDENT_TYPE, and EMPLOYEE_TYPE are declared to be **NOT FINAL**.

Inheritance rules

1. All attributes are inherited.
2. A subtype can redefine any function that is defined in its supertype, with the restriction that the signature be the same.

Objects Relationships

Specifying Relationships via Reference

- A component attribute of one tuple may be a reference (specified using the keyword **REF**) to a tuple of another (or possibly the same) table.
- The keyword **SCOPE** specifies the name of the table whose tuples can be referenced by the reference attribute.

```
(e) CREATE TYPE COMPANY_TYPE AS (
        COMP_NAME          VARCHAR (20),
        LOCATION           VARCHAR (20));
    CREATE TYPE EMPLOYMENT_TYPE AS (
        Employee REF (EMPLOYEE_TYPE) SCOPE (EMPLOYEE),
        Company REF (COMPANY_TYPE) SCOPE (COMPANY)  );
    CREATE TABLE COMPANY OF COMPANY_TYPE (
        REF IS COMP_ID SYSTEM GENERATED,
        PRIMARY KEY (COMP_NAME)  );
    CREATE TABLE EMPLOYMENT OF EMPLOYMENT_TYPE;
```

Figure: COMPANY_TYPE UDT

ODMG Object Model

ODMG Object Model and the Object Definition Language ODL

ODMG

- One of the reasons for the success of commercial relational DBMSs is the SQL standard
- The lack of a standard for ODMSs for several years may have caused some potential users to shy away from converting to this new technology.
- A consortium of ODMS vendors and users (Object Data Management Group) proposed a standard that is known as the ODMG-93 or ODMG 1.0 standard.
- The standard is made up of several parts, including the :
  1. object model, the object definition language (ODL)
  2. The object query language (OQL).

Objects and Literals

Objects Vs Literals

- Objects and literals are the basic building blocks of the object model.
- The main difference between the two is that an object has both an object identifier and a state (or current value) whereas a literal has a value (state) but no object identifier.
- In either case, the value can have a complex structure.
- The object state can change over time by modifying the object value. A literal is basically a constant value, possibly having a complex structure, but it does not change.

Objects and Literals

Objects features

An object has five aspects: identifier, name, lifetime, structure, and creation.

1. The object identifier is a unique system-wide identifier (or Object_id).
2. Some objects may optionally be given a unique name within a particular ODMS. Obviously, not all individual objects will have unique names.
3. The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates).
4. The structure specifies whether an object is atomic or not. An atomic object refers to a single object that follows a user-defined type, such as Employee or Department. If an object is not atomic, then it will be composed of other objects.
5. Object creation refers to the manner in which an object can be created.

Objects and Literals

Literals

- In the object model, a literal is a value that does not have an object identifier.
- However, the value may have a simple or complex structure.

Inheritance in the Object Model of ODMG

In the ODMG object model, two types of inheritance relationships exist:

1. behavior-only inheritance. and
2. State plus behavior inheritance.

Inheritance in the Object Model of ODMG

Behavior inheritance

- Behavior inheritance is also known as ISA or interface inheritance and is specified by the colon (:) notation.
- behavior inheritance requires the **supertype** to be an interface, whereas the **subtype** could be either a class or another interface.

EXTENDS inheritance

- It is used to inherit both state and behavior **strictly among classes**, so both the supertype and the subtype must be classes.
- Multiple inheritance via extends is not permitted.
- However, multiple inheritance is allowed for behavior inheritance via the colon (:) notation.
- Hence, an interface may inherit behavior from several other interfaces.
- A class may also inherit behavior from several interfaces via colon (:) notation.
- A class may inherit behavior and state from at most one other class via extends.

Built-in Interfaces and Classes in the Object Model

Basic Object Interface

All interfaces, such as **Collection**, **Date**, and **Time**, inherit the basic Object interface.

Collection objects

- Collection objects inherit the basic Collection interface which shows the operations for all collection objects.
  1. O.cardinality() operation returns the number of elements in the collection.
  2. O.is_empty() returns true if the collection O is empty, and returns false otherwise.
  3. The operations O.insert_element(E) and O.remove_element(E) insert or remove an element E from the collection O.
  4. Finally, the operation O.contains_element(E) returns true if the collection O includes element E, and returns false otherwise.

Atomic User-Defined Objects

Atomic objects

- Atomic objects are specified using the keyword class in ODL.
- In the object model, any user-defined object that is not a collection object is called an atomic object.

Atomic User-Defined Objects

Sample classes

We illustrate our discussion with the two classes EMPLOYEE and DEPARTMENT.

```
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: The attributes, relationships, and operations in a class definition.

Extents, Keys, and Factory Objects

Extent

- The database designer can declare an extent (using the keyword extent) for any object type that is defined via a class declaration.
- The extent is given a name, and it will contain all persistent objects of that class.
- Hence, the extent behaves as a set object that holds all persistent objects of the class.

Example

The EMPLOYEE and DEPARTMENT classes have extents called ALL_EMPLOYEES and ALL_DEPARTMENTS, respectively.

If two classes A and B have extents ALL_A and ALL_B, and class B is a subtype of class A (that is, class B extends class A) , then the collection of objects in ALL_B must be a subset of those in ALL_A at any point. This constraint is automatically enforced by the database system.

Extents, Keys, and Factory Objects

Key

- A class with an extent can have one or more keys.
- A key consists of one or more properties (attributes or relationships) whose values are constrained to be unique for each object in the extent.
- Example: the EMPLOYEE class has the **Ssn** attribute as key

Factory Object

- An object that can be used to generate or create individual objects via its operations.
- The built-in interface ObjectFactory has a single operation, new(), which returns a new object with an Object_id.
- By inheriting this interface, users can create their own factory interfaces for each user-defined (atomic) object type.
- Tthe programmer can implement the operation new differently for each type of object.

The Object Definition Language ODL

ODL Language

- ODL is used to define the object database schema.
- ODL is designed to support the semantic constructs of the ODMG object model.
- ODL is independent of any particular programming language.
- ODL is mainly used to create object specification - that is, classes and interfaces.

The Object Definition Language ODL

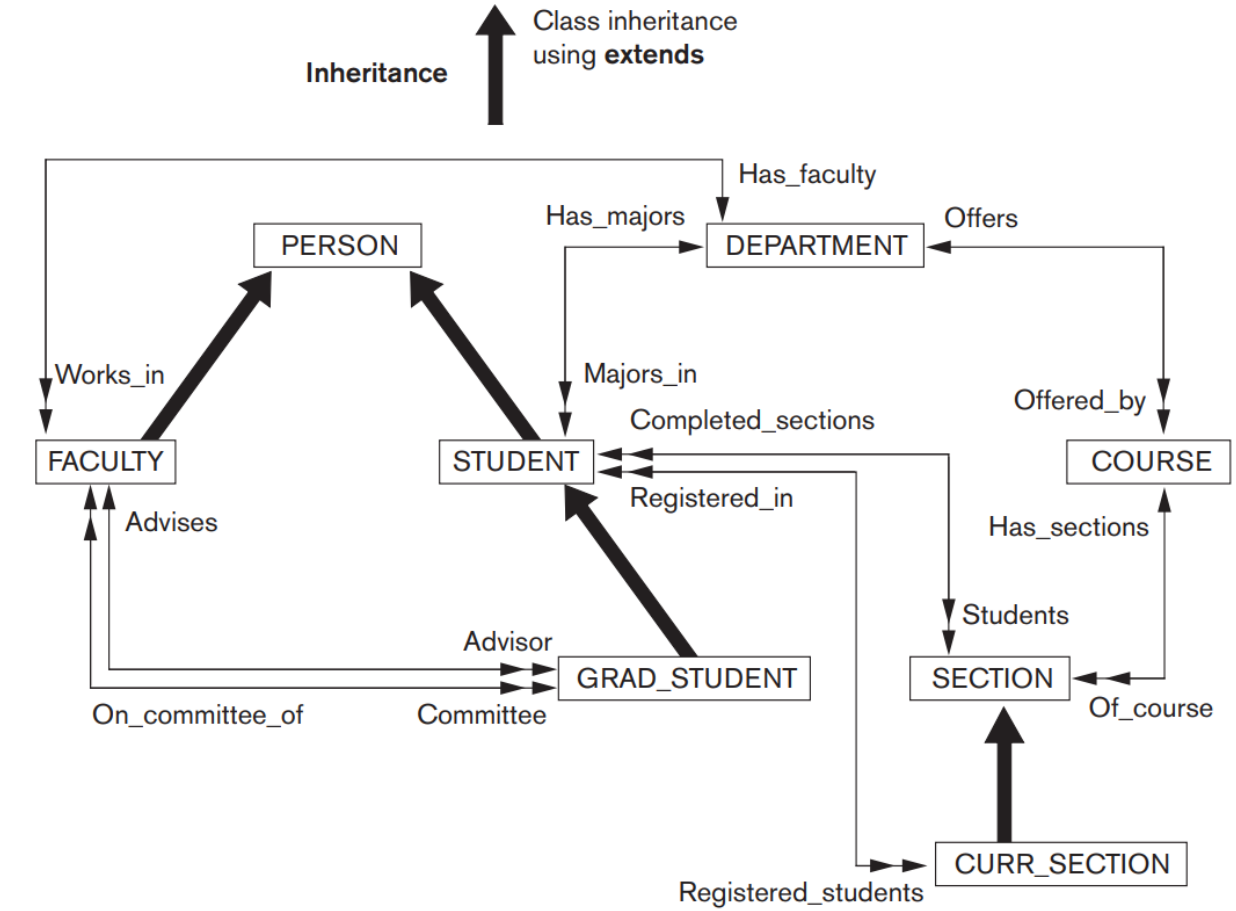


Figure: Part of UNIVERSITY DB

The Object Definition Language ODL

```
class PERSON
(   extent      PERSONS
    key         Ssn  )
{   attribute   struct Pname {  string   Fname,
                                string   Mname,
                                string   Lname  }   Name;
    attribute   string                              Ssn;
    attribute   date                                Birth_date;
    attribute   enum Gender{M, F}                   Sex;
    attribute   struct Address {  short   No,
                                  string  Street,
                                  short   Apt_no,
                                  string  City,
                                  string  State,
                                  short   Zip  }    Address;
    short       Age();    };
```

```
class FACULTY extends PERSON
(   extent          FACULTY  )
{   attribute       string              Rank;
    attribute       float               Salary;
    attribute       string              Office;
    attribute       string              Phone;
    relationship    DEPARTMENT          Works_in inverse DEPARTMENT::Has facul
    relationship    set<GRAD_STUDENT>   Advises inverse GRAD_STUDENT::A
    relationship    set<GRAD_STUDENT>   On_committee_of inverse GRAD_ST
    void            give_raise(in float raise);
    void            promote(in string new rank);  };
```

Figure: Person and Faculty classes

The Object Definition Language ODL

```
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<CURR_SECTION> Registered_in INVERSE CURR_SECTION::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);  };
```

Figure: Student class