Concepts for
Object-Oriented Databases

Chapter 20
Chapter Outline

- Overview of O-O Concepts
- O-O Identity, Object Structure and Type Constructors
- Encapsulation of Operations, Methods and Persistence
- Type and Class Hierarchies and Inheritance
- Complex Objects
- Other O-O Concepts
Introduction

Data Models:
- Hierarchical, Network (since mid-60’s),
- Relational (since 1970 and commercially since 1982)
- Object Oriented (OO) Data Models since mid-90’s

Reasons for creation of Object Oriented Databases
- Need for more complex applications
- Need for additional data modeling features
- Increased use of object-oriented programming languages
History of OO Models and Systems

- **Languages**: Simula (1960’s), Smalltalk (1970’s), C++ (late 1980’s), Java (1990’s)

- **DBMS**
  - **Experimental Systems**: Orion at MCC, IRIS at H-P labs, Open-OODB at T.I., ODE at ATT Bell labs, Postgres - Montage - Illustra at UC/B, Encore/Observer at Brown
  - **Commercial products**: Ontos, Gemstone, O2 ( -> Ardent), Objectivity, Objectstore ( -> Excelon), Versant, Poet, Jasmine (Fujitsu – GM)
    - Commercial OO Database products – several in the 1990’s, but did not make much impact on mainstream data management
Overview of O-O Concepts(1)

- **Main claim**: OO databases try to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon.

- **Object**: Two components: state (value) and behavior (operations). Similar to program variable in programming language, except that it will typically have a complex data structure as well as specific operations defined by the programmer.
In OO databases, objects may have an object structure of arbitrary complexity in order to contain all of the necessary information that describes the object.

In contrast, in 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.
Overview of O-O Concepts (3)

- The internal structure of an object in OOPLs includes the specification of **instance variables**, which hold the values that define the internal state of the object.

- An instance variable is similar to the concept of an attribute, except that instance variables may be encapsulated within the object and thus are not necessarily visible to external users.
Overview of O-Or Concepts (4)

Some OO models insist that all operations a user can apply to an object must be predefined. This forces a complete encapsulation of objects.

To encourage encapsulation, an operation is defined in two parts:

1. signature or interface of the operation, specifies the operation name and arguments (or parameters).
2. method or body, specifies the implementation of the operation.
Operations can be invoked by passing a message to an object, which includes the *operation name* and the *parameters*. The object then executes the method for that operation.

This encapsulation permits modification of the internal structure of an object, as well as the implementation of its operations, without the need to disturb the external programs that invoke these operations.
Overview of O-O Concepts (6)

- **Operator polymorphism**: It refers to an operation’s ability to be applied to different types of objects; in such a situation, an operation name may refer to several distinct implementations, depending on the type of objects it is applied to.

- This feature is also called *operator overloading*
Object Identity, Object Structure, and Type Constructors (1)

- **Unique Identity**: An OO database system provides a unique identity to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated **object identifier**, or **OID**.

- The main property required of an OID is that it be **immutable**; that is, the OID value of a particular object should not change. This preserves the identity of the real-world object being represented.
Object Identity, Object Structure, and Type Constructors (2)

- **Type Constructors**: In OO databases, the state (current value) of a complex object may be constructed from other objects (or other values) by using certain type constructors.

- The three most basic constructors are **atom**, **tuple**, and **set**. Other commonly used constructors include **list**, **bag**, and **array**. The atom constructor is used to represent all basic atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly.
Object Identity, Object Structure, and Type Constructors (3)

- **Example 1**, one possible relational database state corresponding to COMPANY schema

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>M</td>
<td>30000</td>
<td>333445555</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Voss, Houston, TX</td>
<td>M</td>
<td>40000</td>
<td>888665555</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td>999887777</td>
<td>1968-07-19</td>
<td>3321 Castle, Spring, TX</td>
<td>F</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td>987654321</td>
<td>1941-06-20</td>
<td>291 Berry, Bellaire, TX</td>
<td>F</td>
<td>43000</td>
<td>888665555</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
<td>M</td>
<td>38000</td>
<td>333445555</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>453453453</td>
<td>1972-07-31</td>
<td>5631 Rice, Houston, TX</td>
<td>F</td>
<td>25000</td>
<td>333445555</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jabbar</td>
<td>987987987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Borg</td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>M</td>
<td>55000</td>
<td>null</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Object Identity, Object Structure, and Type Constructors (4)

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>DNAME</th>
<th>DNUMBER</th>
<th>MGRSSN</th>
<th>MGRSTARTDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>1986-05-22</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
<td>1995-01-01</td>
<td></td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
<td>1981-06-19</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT_LOCATIONS</th>
<th>DNUMBER</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Stafford</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Bellaire</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Sugarland</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Houston</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>1</td>
<td>32.5</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>2</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>666884444</td>
<td>3</td>
<td>40.0</td>
<td></td>
</tr>
<tr>
<td>453453453</td>
<td>1</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>453453453</td>
<td>2</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>2</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>3</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>10</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>20</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>999887777</td>
<td>30</td>
<td>30.0</td>
<td></td>
</tr>
<tr>
<td>999887777</td>
<td>10</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>987987987</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
<tr>
<td>987987987</td>
<td>30</td>
<td>30.0</td>
<td></td>
</tr>
<tr>
<td>987654321</td>
<td>30</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>987654321</td>
<td>20</td>
<td>15.0</td>
<td></td>
</tr>
<tr>
<td>888665555</td>
<td>20</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>Houston</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
### Object Identity, Object Structure, and Type Constructors (5)

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th>ESSN</th>
<th>DEPENDENT_NAME</th>
<th>SEX</th>
<th>BDATE</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>333445555</td>
<td>Alice</td>
<td>F</td>
<td>1986-04-05</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>Theodore</td>
<td>M</td>
<td>1983-10-25</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>Joy</td>
<td>F</td>
<td>1958-05-03</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>987654321</td>
<td>Abner</td>
<td>M</td>
<td>1942-02-28</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Michael</td>
<td>M</td>
<td>1988-01-04</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Alice</td>
<td>F</td>
<td>1988-12-30</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Elizabeth</td>
<td>F</td>
<td>1967-05-05</td>
<td>SPOUSE</td>
<td></td>
</tr>
</tbody>
</table>
Example 1 (cont.)

We use $i_1, i_2, i_3, \ldots$ to stand for unique system-generated object identifiers. Consider the following objects:

\[
\begin{align*}
\sigma_1 &= (i_1, \text{atom}, \text{‘Houston’}) \\
\sigma_2 &= (i_2, \text{atom}, \text{‘Bellaire’}) \\
\sigma_3 &= (i_3, \text{atom}, \text{‘Sugarland’}) \\
\sigma_4 &= (i_4, \text{atom}, 5) \\
\sigma_5 &= (i_5, \text{atom}, \text{‘Research’}) \\
\sigma_6 &= (i_6, \text{atom}, \text{‘1988-05-22’}) \\
\sigma_7 &= (i_7, \text{set}, \{i_1, i_2, i_3\})
\end{align*}
\]
**Example 1 (cont.)**

\[ o_8 = (i_8, \text{tuple}, <\text{dname}: i_5, \text{dnumber}: i_4, \text{mgr}: i_9, \text{locations}: i_7, \text{employees}: i_{10}, \text{projects}: i_{11}>) \]

\[ o_9 = (i_9, \text{tuple}, <\text{manager}: i_{12}, \text{manager_start_date}: i_6>) \]

\[ o_{10} = (i_{10}, \text{set}, \{i_{12}, i_{13}, i_{14}\}) \]

\[ o_{11} = (i_{11}, \text{set} \{i_{15}, i_{16}, i_{17}\}) \]

\[ o_{12} = (i_{12}, \text{tuple}, <\text{fname}: i_{18}, \text{minit}: i_{19}, \text{lname}: i_{20}, \text{ssn}: i_{21}, \ldots, \text{salary}: i_{26}, \text{supervisor}: i_{27}, \text{dept}: i_8>) \]

\[ \ldots \]
Example 1 (cont.)

- The first six objects listed in this example represent atomic values. Object seven is a set-valued object that represents the set of locations for department 5; the set refers to the atomic objects with values {'Houston', 'Bellaire', 'Sugarland'}. Object 8 is a tuple-valued object that represents department 5 itself, and has the attributes DNAME, DNUMBER, MGR, LOCATIONS, and so on.
Representation of a DEPARTMENT complex object as a graph
Object Identity, Object Structure, and Type Constructors (10)

define type Employee:
  tuple (    fname: string;
    minit: char;
    lname: string;
    ssn: string;
    birthdate: 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;
                  startdate: Date;    );
    locations: set(string);
    employees: set(Employee);
    projects: set(Project);    );

- Specifying the object types Employee, date, and Department using type constructors
Encapsulation of Operations, Methods, and Persistence (1)

- **Encapsulation**

  - One of the main characteristics of OO languages and systems
  
  - Related to the concepts of *abstract data types* and *information hiding* in programming languages
Encapsulation of Operations, Methods, and Persistence (2)

- **Specifying Object Behavior via Class Operations:**
  - The main idea is to define the **behavior** of a type of object based on the **operations** that can be externally applied to objects of that type.
  - In general, the **implementation** of an operation can be specified in a **general-purpose programming language** that provides flexibility and power in defining the operations.
  - For database applications, the requirement that all objects be completely encapsulated is too stringent.
  - One way of relaxing this requirement is to divide the structure of an object into visible and hidden attributes (instance variables).
Encapsulation of Operations, Methods, and Persistence (3)

```plaintext
define class DepartmentSet:
    type set(Department);
    operations add_dept(d: Department): boolean;
        (* adds a department to the DepartmentSet object *)
    remove_dept(d: Department): boolean;
        (* removes a department from the DepartmentSet object *)
    create_dept_set: DepartmentSet;
    destroy_dept_set: boolean;
end DepartmentSet;

...

persistent name AllDepartments: DepartmentSet;
        (* AllDepartments is a persistent named object of type DepartmentSet *)

...

d := create_dept;
        (* create a new Department object in the variable d *)

...

b := AllDepartments.add_dept(d);
        (* make d persistent by adding it to the persistent set AllDepartments *)

...
```
Encapsulation of Operations, Methods, and Persistence (4)

- Specifying Object Persistence via Naming and Reachability:
  - Naming Mechanism: Assign an object a unique persistent name through which it can be retrieved by this and other programs.
  - Reachability Mechanism: Make the object reachable from some persistent object.
  - An object B is said to be reachable from an object A if a sequence of references in the object graph lead from object A to object B.
  - In traditional database models such as relational model or EER model, all objects are assumed to be persistent.
  - In OO approach, a class declaration specifies only the type and operations for a class of objects. The user must separately define a persistent object of type set (DepartmentSet) or list (DepartmentList) whose value is the collection of references to all persistent DEPARTMENT objects.
Type (class) Hierarchy

A type in its simplest form can be defined by giving it a type name and then listing the names of its visible (public) functions.

When specifying a type in this section, we use the following format, which does not specify arguments of functions, to simplify the discussion:

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

**Example:**

PERSON: Name, Address, Birthdate, Age, SSN
Type and Class Hierarchies and Inheritance (2)

- **Subtype:** when the designer or user must create a new type that is similar but not identical to an already defined type

- **Supertype:** It inherits all the functions of the subtype

**Example (1):**

EMPLOYEE: Name, Address, Birthdate, Age, SSN, Salary, HireDate, Seniority

STUDENT: Name, Address, Birthdate, Age, SSN, Major, GPA

OR:

EMPLOYEE subtype-of PERSON: Salary, HireDate, Seniority

STUDENT subtype-of PERSON: Major, GPA
Type and Class Hierarchies and Inheritance (3)

- **Extents**: In most OODBs, the collection of objects in an extent has the same type or class. However, since the majority of OODBs support types, we assume that extents are collections of objects of the same type for the remainder of this section.

- **Persistent Collection**: It holds a collection of objects that is stored permanently in the database and hence can be accessed and shared by multiple programs.

- **Transient Collection**: It exists temporarily during the execution of a program but is not kept when the program terminates.
Complex Objects (1)

- **Unstructured complex object:** It is provided by a DBMS and permits the storage and retrieval of large objects that are needed by the database application.
  
  - Typical examples of such objects are *bitmap images* and *long text strings* (such as documents); they are also known as *binary large objects*, or **BLOBs** for short.
  
  - This has been the standard way by which Relational DBMSs have dealt with supporting complex objects, leaving the operations on those objects outside the RDBMS.

- **Structured complex object:** It differs from an unstructured complex object in that the object’s structure is defined by repeated application of the type constructors provided by the OODBMS. Hence, the object structure is defined and known to the OODBMS. The OODBMS also defines methods or operations on it.
Other Objected-Oriented Concepts

- **Polymorphism (Operator Overloading):** This concept allows the same *operator name* or *symbol* to be bound to two or more different *implementations* of the operator, depending on the type of objects to which the operator is applied.

- **Multiple Inheritance and Selective Inheritance**
  Multiple inheritance in a type hierarchy occurs when a certain subtype $T$ is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both supertypes.

  For example, we may create a subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER. This leads to the creation of a type lattice rather than a type hierarchy.
END