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

Overview of O-O Concepts (2)

- 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.

Overview of O-Or Concepts (5)

- 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 <u>operator overloading</u>

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

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

Object Identity, Object Structure, and Type Constructors (4)

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

Object Identity, Object Structure, and Type Constructors (5)

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Object Identity, Object Structure, and Type Constructors (6)

- **Example 1 (cont.)**

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

  o_1 = (i_1, atom, ‘Houston’)
  o_2 = (i_2, atom, ‘Bellaire’)
  o_3 = (i_3, atom, ‘Sugarland’)
  o_4 = (i_4, atom, 5)
  o_5 = (i_5, atom, ‘Research’)
  o_6 = (i_6, atom, ‘1988-05-22’)
  o_7 = (i_7, set, {i_1, i_2, i_3})

Object Identity, Object Structure, and Type Constructors (7)

- **Example 1(cont.)**

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

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

  o_{10} = (i_{10}, set, {i_{12}, i_{13}, i_{14}})

  o_{11} = (i_{11}, set {i_{15}, i_{16}, i_{17}})

  o_{12} = (i_{12}, tuple, <fname:i_{18}, minit:i_{19}, lname:i_{20}, ssn:i_{21}, . . ., salary:i_{26}, supervi-sor:i_{27}, dept:i_8>)

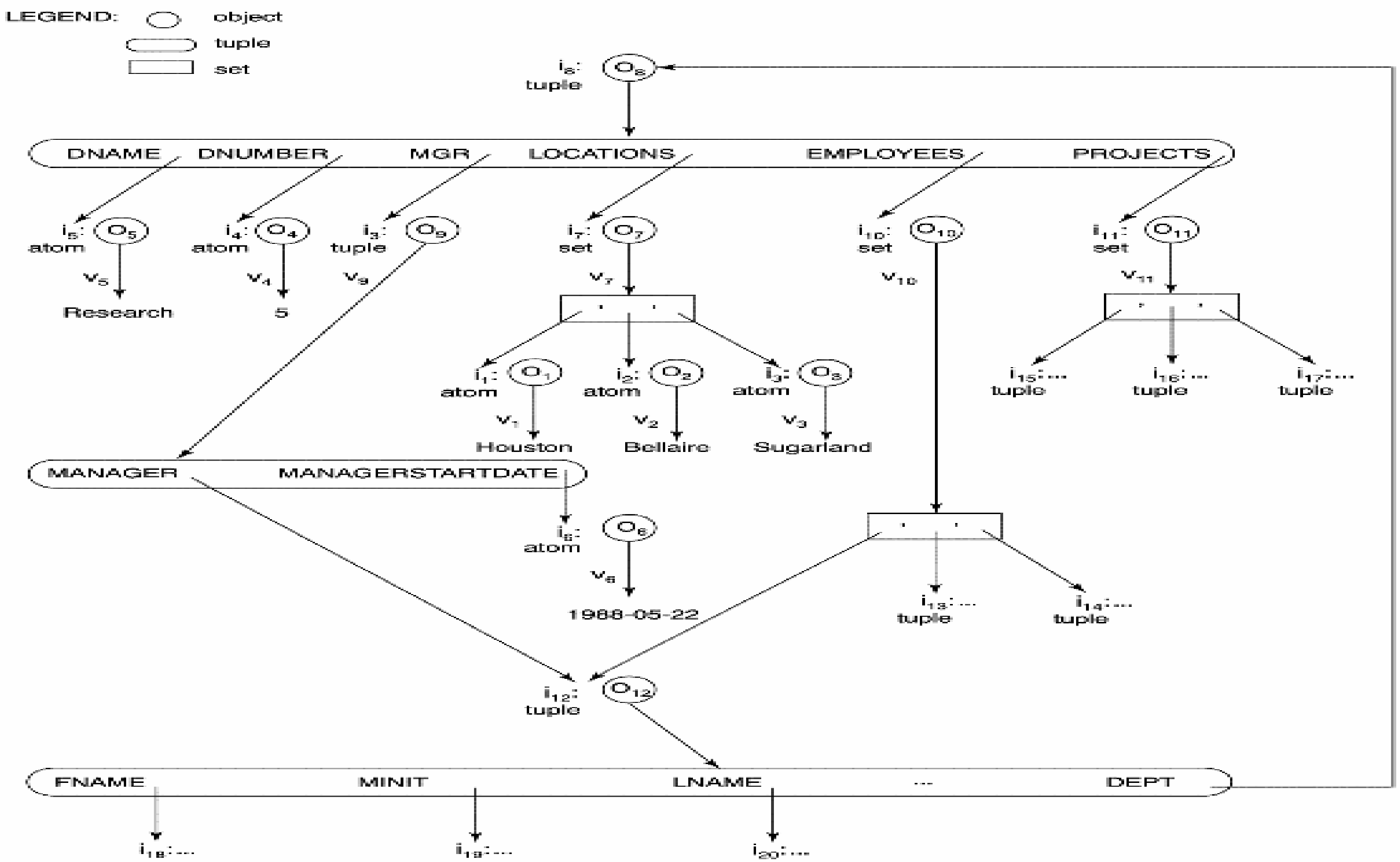
  . . .

Object Identity, Object Structure, and Type Constructors (8)

Example 1 (cont.)

- The first six objects listed in this example represent atomic values. Object seven is a <u>set-valued object</u> 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.

Object Identity, Object Structure, and Type Constructors (9)



- 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)

```
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 <u>separately</u> define a <u>persistent object</u> of <u>type set</u> (DepartmentSet) or <u>list</u> (DepartmentList) whose value is the collection of references to all persistent DEPARTMENT objects

Type and Class Hierarchies and Inheritance (1)

- 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