Advanced Database Management System (CoSc3052)

Object Oriented Databases

Introduction

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

Introduction

- 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
- Commercial OO Database products –
  - Several in the 1990's, but did not make much impact on mainstream data management

History of OO Models and Systems

- Languages:
  - Simula (1960's)
  - Smalltalk (1970's)
  - C++ (late 1980's)
  - Java (1990's and 2000's)

History of OO Models and Systems (cont'd.)

- 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

History of OO Models and Systems (cont'd.)

- Commercial OO Database products:
  - Ontos
  - Gemstone
  - O2 (-> Ardent)
  - Objectivity
  - Versant
  - Object Store (Object Design)
  - Jasmine (Fujitsu – GM)

Overview of Object-Oriented Concepts

- 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)
  - Instance variables
    - Hold values that define internal state of object
  - Operation is defined in two parts:
    - Signature or interface and implementation

Overview of Object-Oriented Concepts (cont'd)

- 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 Object Database Concepts

- Origins in OO programming languages
- Object databases (ODB)
  - Object data management systems (ODMS)
  - Meet some of the needs of more complex applications
  - Specify:
    - Structure of complex objects
    - Operations that can be applied to these objects

Overview of Object Database Concepts (cont'd.)

- Object 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
  - Specifically, the OID value of a particular object should not change.
  - This preserves the identity of the real-world object being represented.

Overview of Object Database Concepts (cont'd.)

- Inheritance
  - Permits specification of new types or classes that inherit much of their structure and/or operations from previously defined types or classes
- Operator overloading
  - Operation's ability to be applied to different types of objects
  - Operation name may refer to several distinct implementations

Complex Type Structures for Objects and Literals

- Structure of arbitrary complexity
  - Contain all necessary information that describes object or literal
- Nesting **type constructors**
  - Construct complex type from other types
- Most basic constructors:
  - Atom
  - Struct (or tuple)
  - Collection

Complex Type Structures for Objects and Literals (cont'd.)

- Collection types:
  - Set
  - Bag
  - List
  - Array
  - Dictionary
- Object definition language (ODL)
  - Used to define object types for a particular database application

Complex Type Structures for Objects and Literals (cont'd.)

- Example
  - 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

Complex Type Structures for Objects and Literals (cont'd.)

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

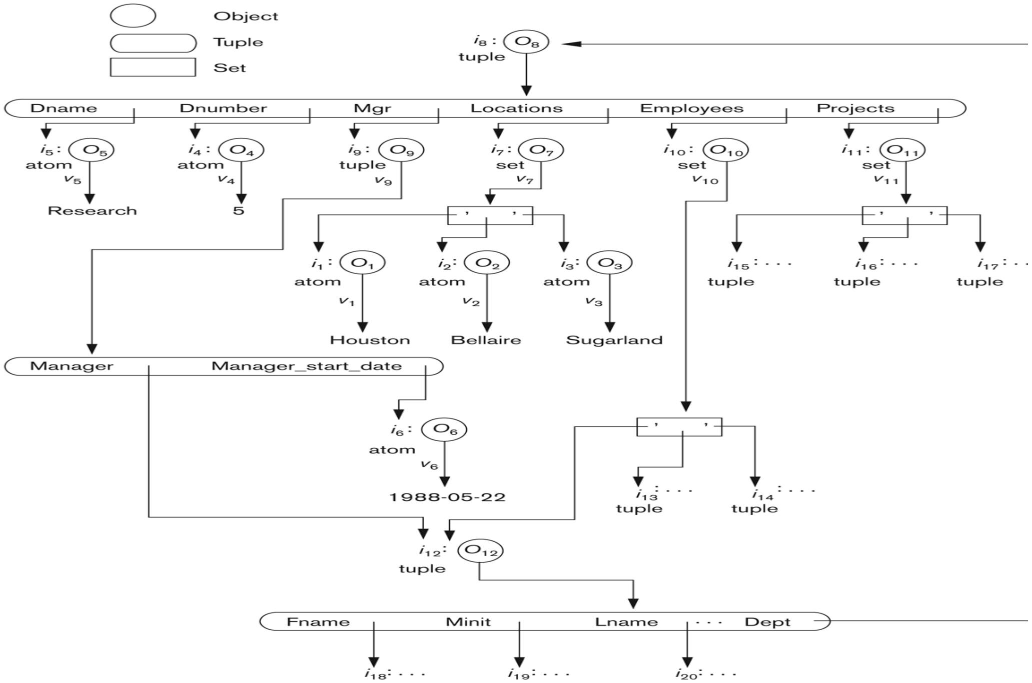
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

Complex Type Structures for Objects and Literals (cont'd.)

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



Complex Type Structures for Objects and Literals (cont'd.)

- 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.

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

```
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);  );
```


Encapsulation of Operations and Persistence of Objects

- Encapsulation
  - Related to abstract data types and information hiding in programming languages
  - Define **behavior** of a type of object based on operations that can be externally applied
  - External users only aware of interface of the operations
  - Divide structure of object into visible and hidden attributes

Encapsulation of Operations and Persistence of Objects

- 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:
  - **signature** or **interface** of the operation, specifies the operation name and arguments (or parameters).
  - **method** or **body**, specifies the implementation of the operation.

Encapsulation of Operations and Persistence of Objects

- Object constructor
  - Used to create a new object
- Destructor operation
  - Used to destroy (delete) an object
- Modifier operations
  - Modify the states (values) of various attributes of an object
- Retrieve information about the object
- Dot notation used to apply operations to object


```
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;
```

Adding operations tc
the definitions o
EMPLOYEE anc
DEPARTMENT

Encapsulation of Operations and Persistence of Objects

- Transient objects
  - Exist in executing program
  - Disappear once program terminates
- Persistent objects
  - Stored in database and persist after program termination
  - Naming mechanism
  - Reachability

Object Identity and Pointers

- A persistent object is assigned a persistent object identifier.
- Degrees of permanence of identity:
  - Intraprocedure – identity persists only during the executions of a single procedure
  - Intraprogram – identity persists only during execution of a single program or query.
  - Interprogram – identity persists from one program execution to another, but may change if the storage organization is changed
  - Persistent – identity persists throughout program executions and structural reorganizations of data; required for object-oriented systems.

Type Hierarchies and Inheritance

- Inheritance
  - Definition of new types based on other predefined types
  - Leads to **type** (or **class**) **hierarchy**
- Type: **type name** and list of visible (public) **functions**
  - Format:
    - `TYPE_NAME: function, function, ..., function`

Type Hierarchies and Inheritance (cont'd.)

- Subtype
  - Useful when creating a new type that is similar but not identical to an already defined type
  - Example:
    - `EMPLOYEE subtype-of PERSON: Salary, Hire_date, Seniority`
    - `STUDENT subtype-of PERSON: Major, Gpa`

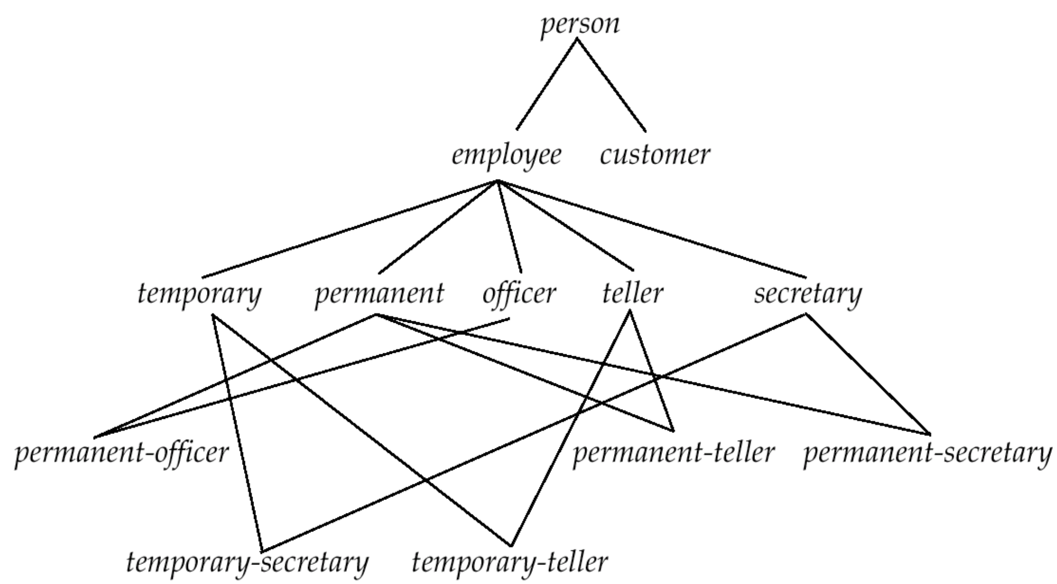
Type Hierarchies and Inheritance (cont'd.)

- Example:
  - Consider a type that describes objects in plane geometry, which may be defined as follows:
    - GEOMETRY_OBJECT: Shape, Area, ReferencePoint
  - Now suppose that we want to define a number of subtypes for the GEOMETRY_OBJECT type, as follows:
    - RECTANGLE **subtype-of** GEOMETRY_OBJECT: Width, Height
    - TRIANGLE **subtype-of** GEOMETRY_OBJECT: Side1, Side2, Angle
    - CIRCLE **subtype-of** GEOMETRY_OBJECT: Radius

Other Object-Oriented Concepts

- Polymorphism of operations
  - Also known as operator overloading
  - Allows same operator name or symbol to be bound to two or more different implementations
  - Depending on type of objects to which operator is applied
- Multiple inheritance
  - Subtype inherits functions (attributes and methods) of more than one supertype

Multiple Inheritance



Other Object-Oriented Concepts (cont'd.)

- Selective inheritance
  - Subtype inherits only some of the functions of a supertype

Summary

- Object identity
- Type constructor
- Encapsulation of operations
- Programming language compatibility
- Type hierarchies and inheritance
- Polymorphism and operator overloading