Chapter 11

Object-Oriented Databases (from E&N and my editing)

Introduction

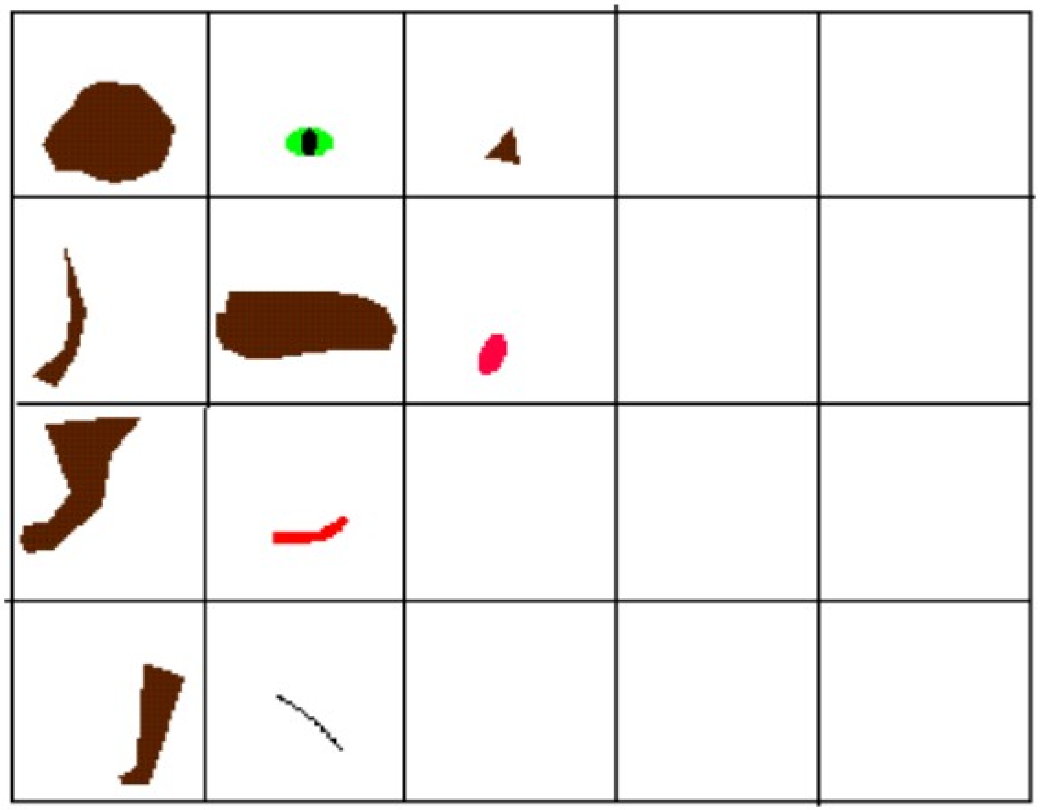
- Traditional Data Models : Hierarchical, Network (since mid-60's), Relational (since 1970 and commercially since 1982)
- Object Oriented (OO) Data Models since mid-90's
- Reasons for creation of Object Oriented Databases
  - Need for more complex applications
  - Need for additional data modeling features
  - Increased use of object-oriented programming languages
  - Unsuitability of RDBMSs for advanced database applications.
- Commercial OO Database products – several in the 1990's, but did not make much impact on mainstream data management
- Basics of object-oriented database analysis and design.

Summary by me

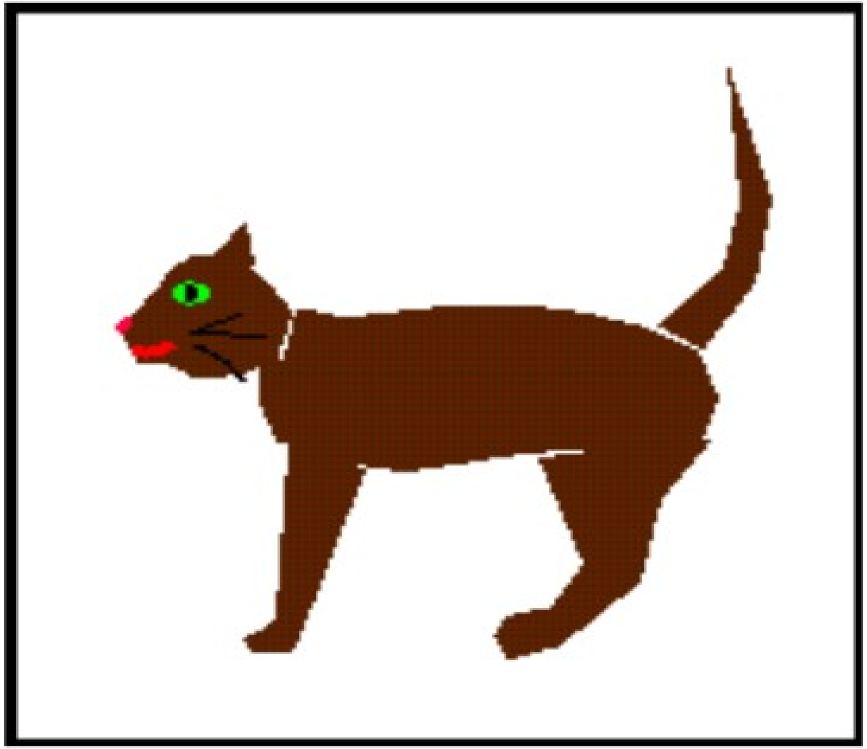
- OODB mencoba menjadi solusi atas kekurangan data model yg lain terutama Relational Model, motivasi yang lain adalah perkembangan OOP perlu diimbangi dengan OODB
- Saat ini OOP namun masih menggunakan Relational Model
- Untuk memodelkan banyak problem yg kompleks
- Dalam ER/ EER satu konsep object dipisah2 dalam beberapa relasi, dengan pendekatan OODB satu object tidak akan dipecah2

- OODB memiliki kelebihan konsep OO dibanding RDB
- Secara theory sudah berusia lebih dari satu decade tapi implementasi enginenya lambat, masih jauh lebih banyak Relational Model.
- Salah satunya karena OODB terlalu kompleks, tidak sederhana seperti RDB
- Untuk menjembatani RDB dengan OODB dibuatlah ORM (Object-Relational Model)
- Bidang yg mengawali pemakaian OODB adalah Multimedia dan GIS → kompleks, baru kemudian bidang yang lain

- RDB of cat → komponen object terpisah dalam banyak relasi (terlalu kompleks bagi RDB untuk memodelkan object yang kompleks)



- OODB of cat → uinified



History of OO Models and Systems

- Languages: Simula (1960's), Smalltalk (1970's), C++ (late 1980's), Java (1990's)
- 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 OO Database products: Ontos, Gemstone, O2 (-> Ardent), Objectivity, Objectstore (-> Excelon), Versant, Poet, Jasmine (Fujitsu – GM)

OODBMS

- Khoshafian and Abnous:
  - OODBMS = OO + Database capabilities.
- Parsaye:
  - High-level query language with query optimization.
  - Support for persistence, atomic transactions: concurrency and recovery control
  - Support for complex object storage, indexes, and access

    methods.
  - OODBMS = OO system + (1), (2), and (3)

- Complex objects must be supported.
- Object identity must be supported.
- Encapsulation must be supported.
- Types or Classes must be supported.
- Types or Classes must be able to inherit from their ancestors.
- Dynamic binding must be supported.
- The DML must be computationally complete (general

  purpose programming language)

Overview of Object-Oriented 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 Object-Oriented 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.

Abstraction

- Process of identifying essential aspects of an entity and ignoring unimportant properties.
- Concentrate on what an object is and what it does, before deciding how to implement it
- Encapsulation: Object contains both data structure and set of operations used to manipulate it.
- Information Hiding: Separate external aspects of an object from its internal details, which are hidden from outside

Object

- Uniquely identifiable entity that contains both the attributes that describe the state of a real-world object and the actions associated with it
- Definition very similar to that of an entity, however, object encapsulates both state and behavior; an entity only models state.

Attributes

- Contain current state of an object
- Attributes can be classified as simple or complex.
- Simple attribute can be a primitive type such as integer, string, etc., which takes on literal values.
- Complex attribute can contain collections and/or references.
- Reference attribute represents relationship.
- An object that contains one or more complex attributes is called a complex object

Object Identifier

- Object identifier (OID) assigned to object when it is created that is:
  - System-generated.
  - Unique to that object.
  - Invariant
  - Independent of the values of its attributes (that is, its state).
  - Invisible to the user (ideally).

- In RDBMS, object identity is value-based: primary key is used to provide uniqueness.
- Primary keys do not provide type of object identity required in OO systems:
  - key only unique within a relation, not across entire system;
  - key generally chosen from attributes of relation, making it dependent on object state.

Methods and Massage

- Method
  - Defines behavior of an object, as a set of encapsulated functions.
- Message
  - Request from one object to another asking second object to execute one of its methods.

Class

- Blueprint for defining a set of similar objects.
  - Objects in a class are called instances.
  - Class is also an object with own class attributes and class methods.
  - Class different from type

Inheritance

- Inheritance allows one class of objects to be defined as a special case of a more general class.
  - Special cases are subclasses and more general cases are superclasses.
  - Process of forming a superclass is generalization; forming a subclass is specialization.
  - Subclass inherits all properties of its superclass andcan define its own unique properties.
  - Subclass can redefine inherited methods.

- All instances of subclass are also instances of superclass.
- Principle of substitutability states that instance of subclass can be used whenever method/construct expects instance of superclass.
- Relationship between subclass and superclass known as A KIND OF (AKO) relationship.
- Four types of inheritance: single, multiple, repeated, and selective.

Polymorphism

- Overriding
  - Process of redefining a property within a subclass.
- Overloading
  - Allows name of a method to be reused with a class or across classes.
- Polymorphism
  - Means ‘many forms’. Three types: operation, inclusion, and parametric.

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

- Example 1 (cont.):

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

- Example 1 (cont.)

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

- Example 1 (cont.) → DEPARTMENT

  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})

- 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>)

  . . .

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.

Example 2:

This example illustrates the difference between the two definitions for comparing object states for equality.

$o_1 = (i_1, \text{tuple}, <a_1:i_4, a_2:i_6>)$

$o_2 = (i_2, \text{tuple}, <a_1:i_5, a_2:i_6>)$

$o_3 = (i_3, \text{tuple}, <a_1:i_4, a_2:i_6>)$

$o_4 = (i_4, \text{atom}, 10)$

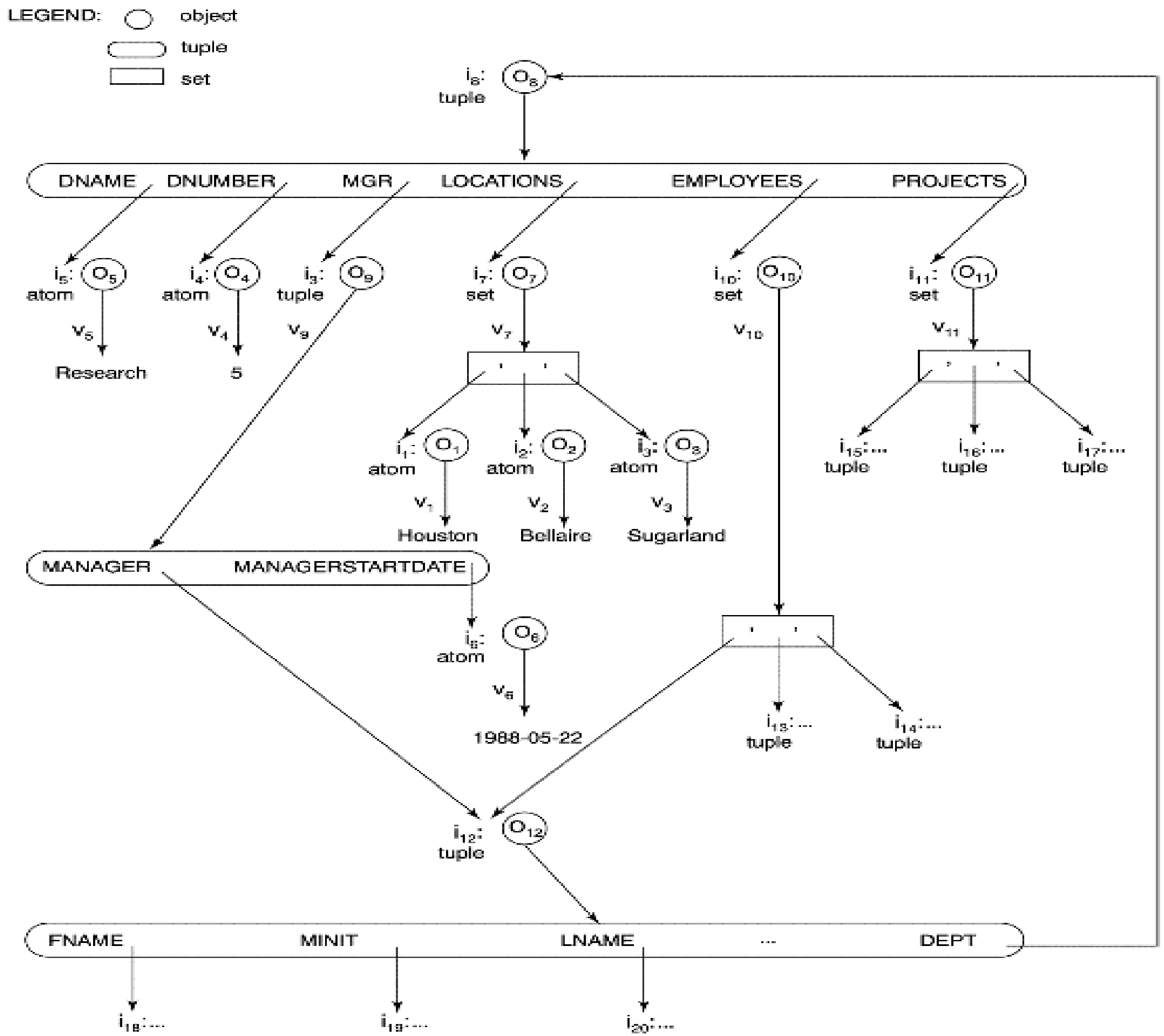
$o_5 = (i_5, \text{atom}, 10)$

$o_6 = (i_6, \text{atom}, 20)$

Example 2 (cont.):

In this example, The objects o_1 and o_2 have ***equal*** states, since their states at the atomic level are the same but the values are reached through distinct objects o_4 and o_5.

However, the states of objects o_1 and o_3 are ***identical***, even though the objects themselves are not because they have distinct OIDs. Similarly, although the states of o_4 and o_5 are identical, the actual objects o_4 and o_5 are equal but not identical, because they have distinct OIDs.


LEGEND:
object
tuple
set
i8: tuple
O8
DNAME
DNUMBER
MGR
LOCATIONS
EMPLOYEES
PROJECTS
i5: atom
O5
i4: atom
O4
i9: tuple
O9
i7: set
O7
i10: set
O10
i11: set
O11
v5
v4
v9
v7
v10
v11
Research
5
i1: atom
O1
i2: atom
O2
i3: atom
O3
i15:... tuple
i16:... tuple
i17:... tuple
v1
v2
v3
Houston
Bellaire
Sugarland
MANAGER
MANAGERSTARTDATE
i6: atom
O6
v6
1988-05-22
i13:... tuple
i14:... tuple
i12: tuple
O12
FNAME
MINIT
LNAME
...
DEPT
i18:...
i19:...
i20:...

```
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);   );
```

Summary (1)

- *Object identity:* Objects have unique identities that are independent of their attribute values.
- *Type constructors:* Complex object structures can be constructed by recursively applying a set of basic constructors, such as tuple, set, list, and bag.
- *Encapsulation of operations:* Both the object structure and the operations that can be applied to objects are included in the object class definitions.

Summary (2)

- *Programming language compatibility:* Both persistent and transient objects are handled uniformly. Objects are made persistent by being attached to a persistent collection.
- *Type hierarchies and inheritance:* Object types can be specified by using a type hierarchy, which allows the inheritance of both attributes and methods of previously defined types.

Summary (3)

- *Extents:* All persistent objects of a particular type can be stored in an extent. Extents corresponding to a type hierarchy have set/subset constraints enforced on them.
- *Support for complex objects:* Both structured and unstructured complex objects can be stored and manipulated.
- *Polymorphism and operator overloading:* Operations and method names can be overloaded to apply to different object types with different implementations.

Summary (4)

- *Versioning:* Some OO systems provide support for maintaining several versions of the same object.