New Database Applications

Data models designed for data-processing-style applications are not adequate for new technologies such as computer-aided design, computer-aided software engineering, multimedia and image databases, and document/hypertext databases.

These new applications require the database system to handle features such as:

- complex data types
- data encapsulation and abstract data structures
- novel methods for indexing and querying
**Weaknesses of RDBMSs**

- **Representation of ‘real world’ entities:** The process of normalisation generally leads to the creation of relations that do not correspond to entities in the ‘real world’.

- **Semantic overloading:** The relational model has only one construct for representing data and data relationships: the relation.

- **Homogeneous data:** The relational model assumes both horizontal and vertical homogeneity. Also, intersection of a row and column must be an atomic value => this structure is restrictive for many ‘real world’ objects with a complex structure.

**Weaknesses of RDBMSs (cont.)**

- **Limited operations:** The relational model has a fixed set of operations (provided in SQL). => does not allow new operations to be specified.

- **Recursive queries:** It is extremely difficult to produce recursive queries (queries about relationships that a relation has with itself).

- **Impedance mismatch:** Result of mixing different programming paradigms (e.g., SQL is a declarative language that handles rows of data whereas a high-level language such as ‘C’ is a procedural language that can handle only one row at a time).
Object-Oriented Data Model

Loosely speaking, and object corresponds to an entity in the E-R model.

The object-oriented paradigm is based on encapsulating code and data related to an object into a single unit.

The object-oriented data model is a logical model (like the E-R model).

Adaptation of the object-oriented programming paradigm (e.g., Smalltalk, C++) to the database systems.

Object Structure

An object has associated with it:
- A set of variables that contain the data for the object. The value of each variable is itself and object.
- A set of messages to which the object responds; each message may have zero, one, or more parameters.
- A set of methods, each of which is a body of code to implement a message; a method returns a value as the response to the message.

The physical representation of data is visible only to the implementer of the object.

Messages and responses provide the only external interface to an object.
Messages and Methods

The term message does not necessarily imply physical message passing. Messages can be implemented as procedure invocations.

Methods are programs written in a general-purpose language with the following features:
- only variables in the object itself may be referenced directly
- data in other objects are referenced only by sending messages

Strictly speaking, every attribute of an entity must be represented by a variable and two methods, e.g., the attribute address is represented by a variable address and two messages get-address and set-address.
- For convenience, many object-oriented data models permit direct access to variables of other objects.

Encapsulation

In Programming Languages
- Definition of Object: Specification + Implementation
- Only specification visible to user
- Implementation is seen only by the programmer

In Databases
- Definition of Object: data + operations
- Signatures of operations are the only visible part of the object

Encapsulation implies data independence
Data and Operations in a RDBMS

- Data stored in the database (designed first)
- Programs stored in a file system (designed after)
- Example:
  - a database file: EMPLOYEE(name, age, SSN, location, salary)
  - application program
    » Change_Location (EMPLOYEE, new_location)
    » Change_Salary (EMPLOYEE, amount)

Encapsulation in an OODBMS

- Data and operations are encapsulated in the same object
- Example

  EMPLOYEE --> (OiD)

  Specification (visible)
  - Change_Location (EMPLOYEE, new_location)
  - Change_Salary (EMPLOYEE, amount)
  end_specification

  Implementation (invisible)
  - Name, age, SSN, location, salary (Data)
  - Change_Location (...) 
  - Change_Salary (...) 
  end_implementation
Encapsulation: Relational vs OO Systems

- Data and operations are designed at the same time in an OO database system
- Data and operations are stored in the same system
- Only the signature of operations is visible to the users

Object Classes

- Similar objects are grouped into a class; each such object is called an instance of its class.
- All objects in a class have the same
  - variable types
  - messages interface
  - methods
  They may differ in the values assigned to variables
- Example: Group objects for people into a person class
- Classes are analogous to entity sets in the E-R model
Class Definition Example

class employee {
    /* Variables */
    string name;
    string address;
    date start-date;
    /* Messages */
    int annual-salary();
    string get-name();
    string get-address();
    int set-address(string new-address);
    int employment-length();
};

// For strict encapsulation, methods to read and set other variables are also needed.
// employment-length is an example of a derived attribute.

Inheritance

// E.g., class of bank customers similar to class of bank employees: both share some variables and messages, e.g., name and address. But there are variables and messages specific to each class e.g., salary for employees and credit-rating for customers.
// Every employee is a person; thus employee is a specialisation of person
// Similarly, customer is a specialisation of person.
// Create classes person, employee and customer
  // variables/messages applicable to all persons associated with class person.
  // variables/messages specific to employees associated with class employee; similarly for customer
Inheritance (cont.)

Place classes into a specialisation/IS-A hierarchy
– variables/messages belonging to class person are inherited by class employee as well as customer

Result is a class hierarchy

```
person
    employee
        officer
teller
    customer
        secretary
```

Note analogy with ISA hierarchy in the E-R model

Class Hierarchy Definition

```cpp
class person {
    string name;
    string address;
};

class customer isa person {
    int credit-rating;
};
class employee isa person {
    date start-date;
    int salary;
};
class officer isa employee {
    int office-number;
    int expense-account-number;
};
...
Class hierarchy Example (cont.)

// Full variable list for objects in the class officer:
- office-number, expense-account-number: defined locally
- start-date, salary: inherited from employee
- name, address: inherited from person
// Methods inherited similar to variables.
// Substitutability - any method of a class, say person, can be invoked equally well with any object belonging to any subclass, such as subclass officer of person.
// Class extent: set of all objects in the class. Two options:
1 Class extent of employee includes all officer, teller and secretary objects
2 Class extent of employee includes only employee objects that are not in a subclass such as officer, teller or secretary

Multiple Inheritance

// The class/subclass relationship is represented by a directed acyclic graph (DAG) - a class may have more than one superclass.
// A class inherits variables and methods from all its superclasses
// There is potential for ambiguity. E.g., variables with the same name inherited from two superclasses. Different solutions such as flag error, rename variables, or choose one.
// Can use multiple inheritance to model “roles” of an object.
- A person can play the roles of student, a teacher or footballPlayer, or any combination of the three (e.g., student teaching assistants who also play football).
- Create subclasses such as student-teacher and student-teacher-footballPlayer that inherit from multiple classes
Example of Multiple Inheritance

Class DAG for banking example

Object Identity

- An object retains its identity even if some or all the values of variables or definitions of methods change over time.
- Object identity is a stronger notion of identity that in programming languages or data models not based on object orientation.
  - Value - data value; used in relational systems.
  - Name - supplied by user; used for variables in procedures.
  - Built-in - identity built into data model or programming language.
    - no user-supplied identifier is required.
    - form of identity used in object-oriented systems.
Object Identifiers

//Object identifiers used to uniquely identify objects

– can be stored as a field of an object, to refer to another object.
– E.g, the spouse field of a person object may be an identifier of another person object.
– can be system generated (created by the database) or external (such as social-security number).

Object Containment

//Each component in a design may contain other components
//Objects containing other objects are called complex or composite objects.
//Multiple levels of containment create a containment hierarchy: links interpreted as IS-PART-OF, not IS-A.
//Allows data to be viewed at different granularities by different users.
Polymorphism

- Same operation is defined on objects of different types

- Example: Suppose we sell a variety of Items, but the rules for computing their total price differs. For instance, medical items might be exempt from VAT, whereas clothing items are not. We still want all Items to have a function called TotalPrice(), but instead of one TotalPrice() function we now want different functions depending on the class.

Polymorphism is also known by other names such as generic functions, functions instances and operator overloading.

Overriding, Overloading and Late Binding

- Overriding: the possibility of re-defining methods in the class hierarchy

- Overloading: using the same name to indicate different codes

- Late Binding (Dynamic Binding): delaying the association between the name of a method and its implementation until run-time (rather than compile time)