ROHINI
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE and affiliated to Anna University,(An ISO Certified Institution)
Accredited by NAAC with A+ Grade

ACCREDITED WITH GRADE A+ NAAC

Recognized Under Section 2(f) of University Grants Commission, UGC Act 1956

Department of Management Studies

MBA – I Semester

BA4106 Information Management

Dr.Jackson Daniel
Professor/ECE Department

UNIT –III

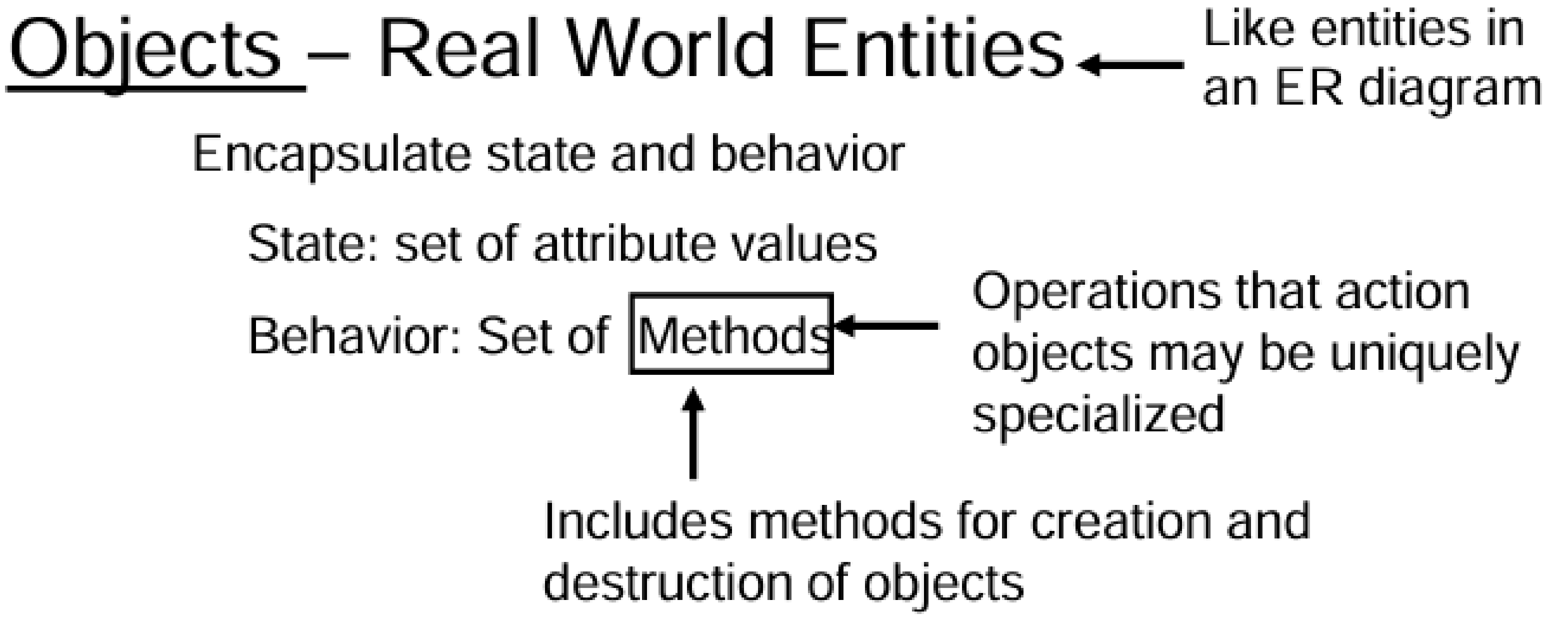
Database Management Systems

Object Oriented Database Management System – OODBMS

Object-Oriented Database Management System (OODBMS)

An Object-Oriented Database Management System (OODBMS) is a type of database management system that extends the principles of object-oriented programming to database management. In an OODBMS, data is represented as objects, similar to how objects are represented in object-oriented programming languages.

Object-Oriented DBS Concepts



Objects – Real World Entities

Like entities in an ER diagram

Encapsulate state and behavior

State: set of attribute values

Behavior: Set of Methods

Operations that action objects may be uniquely specialized

Includes methods for creation and destruction of objects

⇨ Objects offer **encapsulation** of both attributes and specialized operations/methods

OODBS Concepts (Continued)

CLASS — group of objects sharing the same attributes and methods

e.g. employee …

department …

INSTANCE — Individual, uniquely identified objects with attribute values

e.g. employee25 (‘John Smith’, M, 39, …)

(analogous to entity-scheme == class

entity tuple values == instance

Class	:	Object(instance)	(OO model)
Table	:	Tuple	(Rational model)
Entity Set	:	Entity	(ER model)

Object-Oriented Data Model

- Loosely speaking, an *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 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 an 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 implementor 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

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
  - message 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;
        int         salary;
    /* Messages */
        int         annual-salary();
        string      get-name();
        string      get-address();
        int         set-address(string new-address);
        int         employment-length();
};
```

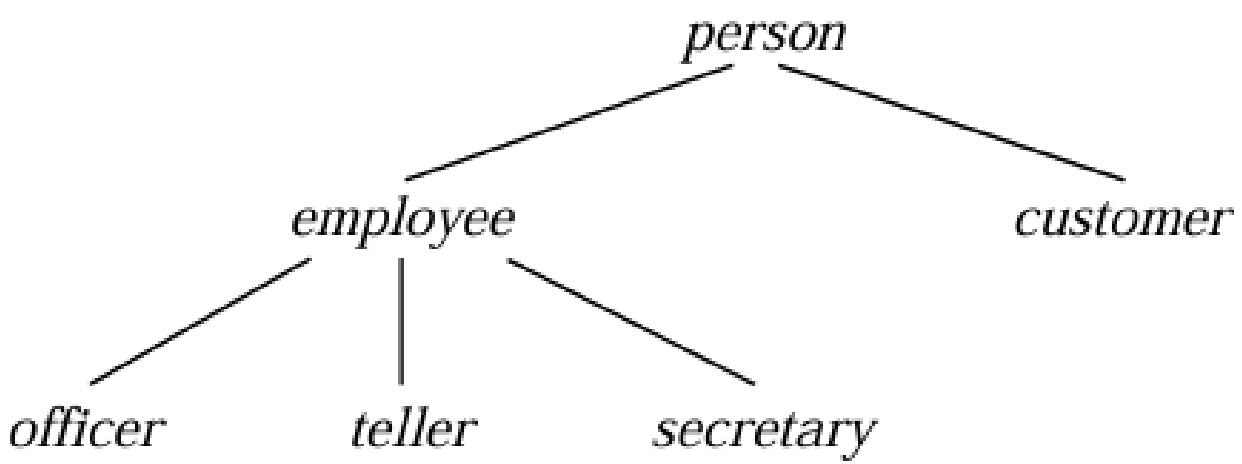
- For strict encapsulation, methods to read and set other variables are also needed
- *employment-length* is an example of a derived attribute

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 and *credit-rating* for customers
- Every employee is a person; thus *employee* is a specialization of *person*
- Similarly, *customer* is a specialization 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 specialization/IS-A hierarchy
  - variables/messages belonging to class *person* are *inherited* by class *employee* as well as *customer*
- Result is a class hierarchy



Note analogy with ISA hierarchy in the E-R model

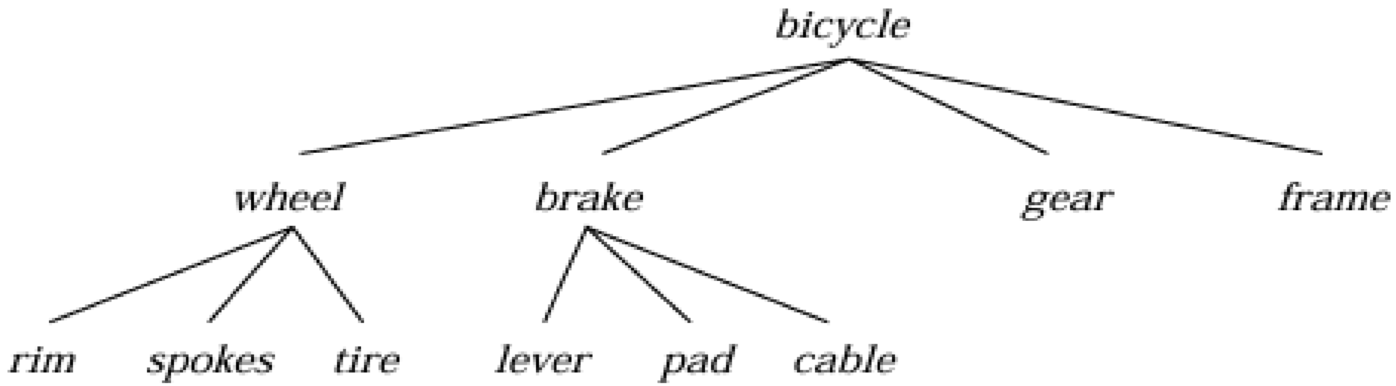
Object Identity

- An object retains its identity even if some or all of the values of variables or definitions of methods change over time.
- Object identity is a stronger notion of identity than 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 database) or external (such as social-security number)

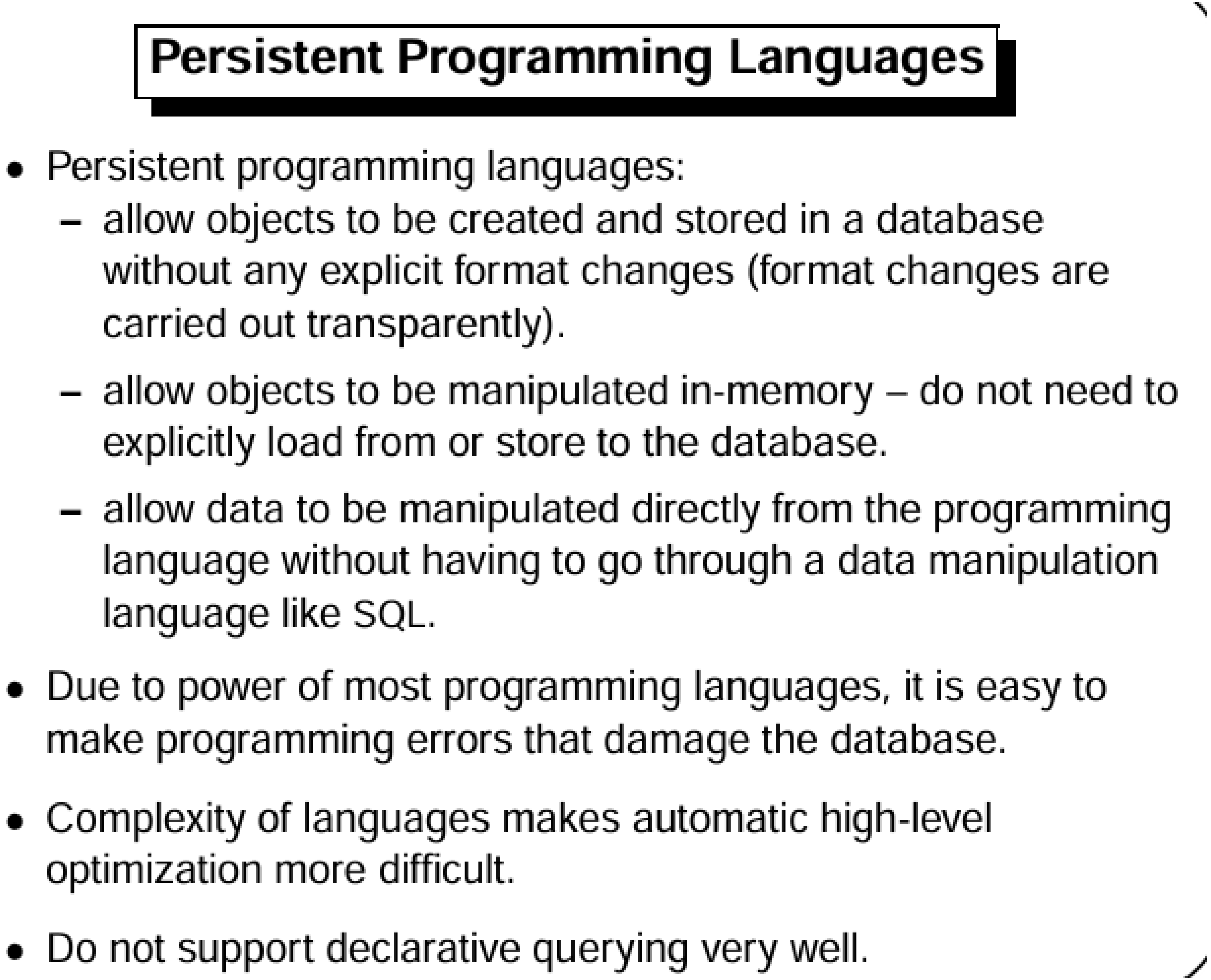
Object Containment



- Each component in a design may contain other components
- Can be modeled as containment of objects. 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

Object-Oriented Languages

- Object-oriented concepts can be used as a design tool, and be encoded into, for example, a relational database (analogous to modeling data with E-R diagram and then converting to a set of relations).
- The concepts of object orientation can be incorporated into a programming language that is used to manipulate the database.
  - Object-relational systems – add complex types and object-orientation to relational language.
  - Persistent programming languages – extend object-oriented programming language to deal with databases by adding concepts such as persistence and collections.



Persistent Programming Languages

- Persistent programming languages:
  - allow objects to be created and stored in a database without any explicit format changes (format changes are carried out transparently).
  - allow objects to be manipulated in-memory – do not need to explicitly load from or store to the database.
  - allow data to be manipulated directly from the programming language without having to go through a data manipulation language like SQL.
- Due to power of most programming languages, it is easy to make programming errors that damage the database.
- Complexity of languages makes automatic high-level optimization more difficult.
- Do not support declarative querying very well.

Persistence Of Objects

- Approaches to make transient objects persistent include establishing persistence by:
  - Class – declare all objects of a class to be persistent; simple but inflexible.
  - Creation – extend the syntax for creating transient objects to create persistent objects.
  - Marking – an object that is to persist beyond program execution is marked as persistent before program termination.
  - Reference – declare (root) persistent objects; objects are persistent if they are referred to (directly or indirectly) from a root object.

Object Identity and Pointers

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

Object Identity and Pointers (Cont.)

- In O-O languages such as C++, an object identifier is actually an in-memory pointer.

- Persistent pointer – persists beyond program execution; can be thought of as a pointer into the database.

Advantages / Disadvantages of OODB

Advantages

- Class inheritance
- Encapsulation of attributes/methods
- Extensible/flexible definition of complex data types and methods(support for complex objects)
- Much greater power given to the programmer to add or change databases semantics.

Disadvantages

- **Handling of relationships**
  - Cumbersome
  - Data duplicated
  - Consistency not enforced
- **Table based representation is often more**
  - Natural
  - Intuitive
  - Efficient
- **May give too much power to programmer**
- **Integrity/consistency poorly enforced**
  - More restrictive relational mode semantics makes integrity correctness enforcement easier.

Thank You