Object-Oriented Database Systems (OODBS)

- **Motivation**
  - Relational Models not adequate for the need of
    - CAD (Computer-Aided Design)
    - CASE (Computer-Aided Software Engineering)
    - Geographic Databases
    - Multimedia Databases
  
  - **Richer data types needed** (images, audio, video, geographical data, text)
  - Need to model complex objects (design for engineering of car in CAD, newtal documents)
  - Need for better generalizational inheritance
  - Long-duration transactions (problem of waits/locks)
  - Temporal evolution of data (version control)
Object-Oriented DBS Concepts

**Objects** – Real World Entities

- **Encapsulate state and behavior**
  - State: set of attribute values
  - Behavior: Set of **Methods**

- **Like entities in an ER diagram**
  - 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)

---

<table>
<thead>
<tr>
<th>Class</th>
<th>Object(instance)</th>
<th>(OO model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Tuple</td>
<td>(Rational model)</td>
</tr>
<tr>
<td>Entity Set</td>
<td>Entity</td>
<td>(ER model)</td>
</tr>
</tbody>
</table>
CLASS INHERITANCE

Classes organized in a

TYPE HIERARCHY (Tree or Directed Acyclic Graph)

Like ER Diagram

ACCOUNT

ISA

SAVINGS ACCOUNT

CHECKING ACCOUNT

Subclass inherits attributes and methods from Superclass
CLASS INHERITANCE

PERSON: Name, Address, Birthday, Age, SSN → Method

EMPLOYEE: , Salary, Hiredate, Manager, Seniority → Method

STUDENT: , Major, GPA Method, ...

FULLTIME EMPLOYEE: , HealthPlan, ...

PART EMPLOYEE: , Hours per week
CLASS INHERITANCE

PERSON: Name, Address, Birthday, Age, SSN → Method

EMPLOYEE: , Salary, Hiredate, Manager, Seniority ← Method

STUDENT: , Major, GPA method, ...

FULLTIME EMPLOYEE: , Hours per week, ...

PARTTIME EMPLOYEE: , Hours per week, ...

TELLER

SECRETARY

FULLTIME TELLER

PARTTIME TELLER
Problem of Multiple Inheritance

• Inherit Attributes, methods from:
  • Both/ALL superclasses
  • Only dominant superclass (all cases or only where conflict)
  • Ad hoc

  Which is most appropriate for the employee case?

• SELECTIVE INHERITANCE

  EXCEPT (attribute/method)

  Listed in subclass to indicate attributes not to be inherited
OBJECT INDENTITY

Remains invariant once object is created

→ Unique permanent object ID number

Unlike Relational Model where

Object ID is derived exclusively from data values

(Tuples with the same values \(\equiv\) same)
Advanced Concepts

• **Polymorphism** (Operator overloading)
  
  Geometric database with multiple methods for computing ‘area’ dependent on implementation
  
  (late binding of method code)

• **VERSION CONTROL**

  (e.g. Attribute values change over time,
  
  Maintain version graph to capture multiple states)

  (like software version control)
SAMPLE OODBS

- O2 (O2 Technology) -- 1991
- ObjectStore (Object Design Inc.) -- 1991

Data Definition in O2

- Atomic data types (char, int, …)
- Type constructors (tuple, list, set, unique set)
- Class definitions
  - Attributes (have type)
  - Methods (have defining case)
  - Inherit <supertype> ➯ multiple inheritance of all superclass attributes
- Objects (member of a class)
  - Persistent or transient ➯ temporary
  - Remain after database shutdown
OODBS QUERY LANGUAGES

• Convert/Export O2 DBs into persistent object store for use with C++

• **O2SQL** Language

  Select tuple (fname: s.name.firstname, lname: s.name.lastname)
  from S in Student
  where S.majors_in.dname = "Computer Science"

  Select firstname, lastname
  from Student, majors_in
  where majors_in.dname = "Computer Science"
  And majors_in.student_id = student.student_id -- Join Condition
DOT NOTATION \approx \text{Join}

S: \begin{align*}
\text{STUDENT:} \\
\text{Majors_in: Department}
\end{align*}
\quad \rightarrow \quad \begin{align*}
\text{DEPARTMENT} \\
dname: \text{string}
\end{align*}

S.majors_in.dname = “Computer Science”

\\\\
Nested dot resolution \rightarrow \text{pointer following}
(\text{like network DBS})

Takes place of \text{Join}.
Relations are poorly captured in OODBS. One of their greatest weaknesses is the inability to represent relationships effectively.

**BINARY RELATIONSHIPS**

- Class student inherit person
  - majors in: department,
  - end

- Class department
  - majors: set(student),
  - end

Major_in relation captured by “pointers” from one object to another.
Problems with Relation Representation

• Information duplicated: 
  Majors: set(student) 
  Duplicates info contained in each 
  Majors_in:department listing.

• Consistency not enforced by DBS
  - Programmer writing change-major method responsible for ensuring that changes to student: major_in also reflected/updated in department: majors: set(student)
  - This consistency NOT guaranteed so different methods of extracting a list of majors may yield different results.

  Jones: majors_in = ‘Compute science’ but
  Jones not in ‘ComputerScience.majors’ set
Problems with Relational Representation (cont)

- Relationships can’t have attributes in pointer representation

```plaintext
Class employee
    
    works_on: set(project)

end
```

```
Class project
    
    members: set(employee),

end
```

How to represent hours worked by each employee on each project is problematic.
Problems with representing relations

Class project_work

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>participant: employee,</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>project_on: project,</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>hours_worked: integer,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
</table>
| end

Class employee

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>works_on: set(project)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
</table>
| end

Class project

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>members: set(employee),</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
</table>
| end

No explicit database support or consistency enforcement - left to programmer.

Can simulate a relation on DB relation turned objects, but not in spirit of OODBs.

Each employee/project pairing is its own object.

Probably also set lists in employee and project.
## Advantages / Disadvantages of OODB

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Class inheritance</td>
<td>• Handling of relationships</td>
</tr>
<tr>
<td>• Encapsulation of attributes/methods</td>
<td>➢ Cumbersome</td>
</tr>
<tr>
<td>• Extensible/flexible definition of complex data types and methods</td>
<td>➢ Data duplicated</td>
</tr>
<tr>
<td>(support for complex objects)</td>
<td>➢ Consistency not enforced</td>
</tr>
<tr>
<td>• Much greater power given to the programmer to add or change databases</td>
<td><strong>Table based representation is often more</strong></td>
</tr>
<tr>
<td>semantics.</td>
<td>➢ Natural</td>
</tr>
<tr>
<td></td>
<td>➢ Intuitive</td>
</tr>
<tr>
<td></td>
<td>➢ Efficient</td>
</tr>
<tr>
<td></td>
<td><strong>May give too much power to programmer</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Integrity/consistency poorly enforced</strong></td>
</tr>
<tr>
<td></td>
<td>➢ More restrictive relational mode semantics makes integrity correctness</td>
</tr>
<tr>
<td></td>
<td>enforcement easier.</td>
</tr>
</tbody>
</table>

- 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.