Object Databases: Logical Data Modeling

Object Databases: Review
- Object Identity
- Behavioral modeling via methods
- Subclasses and inheritance
- Rich type system
- Encapsulation and information hiding

Object Databases: Advantages
- Better suited for many “new” applications
  - Non-tabular data
  - Large or variable-sized objects
- More realistic data structuring
- Explicit relationships
- Easier embedding in a host language (e.g. C++)
- Ease of design and querying (sometimes)
- “Support” for ordered data

Object Databases: Disadvantages
- Not as established as relational technology
- May be overkill for some systems
- Schema design not well understood
- Query processing still being researched

Logical Object Database Design
- We will focus on O2, an OODBMS
  - OO features generally subsume OR features
  - O2 is representative and mostly ODMG-compliant
  - Other OO/OR systems make other choices
- Can do first-level logical design directly:
  - Use ODL or other OO modeling languages
  - Can skip E/R entirely
  - One less level of translation (applause, please…)
  - No standards or theory for logical design (boo…)

Logical DB Design in O2
- Class concept captures
  - Object structure (type)
  - Object behavior (methods)
  - Inheritance (single and multiple)
  - Type extents
  - Many ICs
- Actual data created using named DB objects
  - like persistent global variables within a DB
  - “points of entry” into DB for browser, OQL
- Application programs model other aspects
**Classes: Attributes**

- Standard **scalar** types
  - integer, char, etc.
- **set** (really a multiset; allows duplicates)
- **unique set** (a “real” set)
- **list** (indexable)
- **object**
- **tuple**
- Object vs. tuple: different semantics, storage
- “types” not always tuples !!!

**Classes: Methods**

- Model object **behavior**
- Only way to access data of **private** types
- Used to update an object
  - Can also use browser directly if class is **public**
- The **init** method:
  - Like a C++ constructor
  - Invoked whenever a new object is created
  - Very handy for maintaining extents
  - Not inherited

**Classes: Inheritance**

- **Type** (structure) and **methods** (behavior) are both inherited by subclasses
- **Exception**: “init” method not inherited
- **Semantics**: substitutability
  - Example: a **Student** can appear wherever a **Person** is allowed to appear (students are people too!)
  - Collection types can participate in inheritance
- Multiple Inheritance:
  - Two or more direct superclasses (e.g. **WorkStudy**)
  - Must resolve naming conflicts (none in this case)

**Classes: Inheritance (cont.)**

- Can **override** method, attribute definitions in a subclass
  - Type of redefined attribute must be subtype of inherited attribute’s type
  - All types in a redefined method signature must be subtypes of corresponding types in inherited method
- **Problem**:
  - for (nextDept in Departments)
    nextDept->fire_emp;
  - Which **fire_emp** method is invoked on a nextDept?
  - Unknown until run-time: **late binding**

**Classes: Extents**

- An **extent** is a (unique, persistent) set containing (the OIDs of) every object of the class and its subclasses
- **Optional** for every class
- Similar to relations
- Built in to ODL
- In some actual systems, must create and maintain “manually”
- **Keep** can be specified for a class iff it has an extent

**Classes: Referential Integrity**

- OIDs can refer to an object anywhere in DB
- Objects can be referenced from anywhere in DB
- Insertion or deletion of a reference can never violate referential integrity
- Assume automatically maintained extents
- Deletion of a referenced object, default:
  - Same as SQL-92: disallow deletion from extent
- Methods can enforce other delete semantics
  - Cascading delete, set NULL, or set default
**Named DB Objects**

- These are the **roots of persistence**
- They are the only way to access data
- Can use browser to examine all our data:
  - Start at a named DB object and follow OIDs
  - Can do some limited updates via browser also
  - Can’t do “real” queries in browser
  - Can’t “link” two existing objects in browser
- OQL queries (next class) must use named DB objects to retrieve any data

**Object Deletion**

- Object physically removed when all sources of its persistence are removed
  - Thus deleting a named DB object doesn’t physically remove object unless nothing else in DB references that object
- One solution: reference counts
  - Remove object when reference count = 0
  - Performance problems
  - Must ensure that copying a ref. incr. ref. count
  - Used in early versions of O2
- Can use periodic garbage collection instead

**Object Deletion (cont.)**

- An alternative (not available in O2): Allow explicit object deletion at any time
- Replace the physical object with a tombstone
  - Tombstone is a special marker (similar to a NULL)
  - When some object follows an OID to the deleted object, it encounters the tombstone
  - The reference that was just followed can be set to NULL or some default value or other action taken
- This approach can make implementation of SET NULL semantics, etc. much easier!!
- Avoids the dangling references problem

**Application Programs**

- Associated with an O2 schema
- Used for **non-object-specific tasks**
  - Frequently-performed tasks that can’t or shouldn’t be coded as methods
  - Tasks that involve changing or examining more than one object
- **Examples:**
  - Prompt for a department name and display it
  - Display all departments
  - Move an employee from one dept. to another

**Physical Design: Indexing**

- Can index named list, set, unique set objects
- **Search key** must be:
  - An atomic value (e.g. integer)
  - An OID
  - A collection (list, set, unique set) of the above
- Elements of an index path must all be tuples, not OIDs (except possibly the last element)
- Sample indices:
  - **Companies:** name, address.Country, address.city.name
  - **Departments:** emps

**Physical Design: Clustering**

- When an object becomes persistent, by default it is clustered near its parent
- DBA can specify clusters based on **cluster trees**:
  - subsets of the schema composition graph
  - can be sorted
  - defined on classes or collection objects
  - deep cluster trees can impede performance
- **Examples:**
  - cluster Person /* all Person objects clustered */
  - cluster Department on (chair) /* Departments stored with chairy emps, majors stored elsewhere */
Summary

- **Classes** reflect behavior and complex structure
- **Inheritance** provides new semantics
- **Methods** and **OIDs** help enforce integrity
- Named DB objects are “entry points” into DB
- Method, application language has same type system as DBMS !!!
- Physical design options are numerous
- Much of this also applies, in a modified way, to Object-Relational DBMSs!!!

State of the Art (logical ODB modeling)

- Indexing techniques
- **Temporal** OODB modeling
- **Deductive** OODB modeling
- **Active** OODB modeling
- More sophisticated ordered type support (e.g. trees, graphs)
- Heterogeneous database integration
- Garbage collection techniques
- Storage and clustering techniques