Object Database Tutorial

Rick Cattell
Cattell.Net Consulting

International Conference on
Object-Oriented Databases
Zurich, 2009

© Copyright 2009: Use with permission please

Morning agenda

- What is an “object database”?
- Object database history and products
- Persistence models, object IDs
- Classes and inheritance
- Relationships and collections
- Encapsulation and methods
- Transaction models

Morning agenda continued

- Query languages
- Versioning and other features
- Implementation and performance issues
- Object database standards
- Object databases vs alternatives
- Q&A session

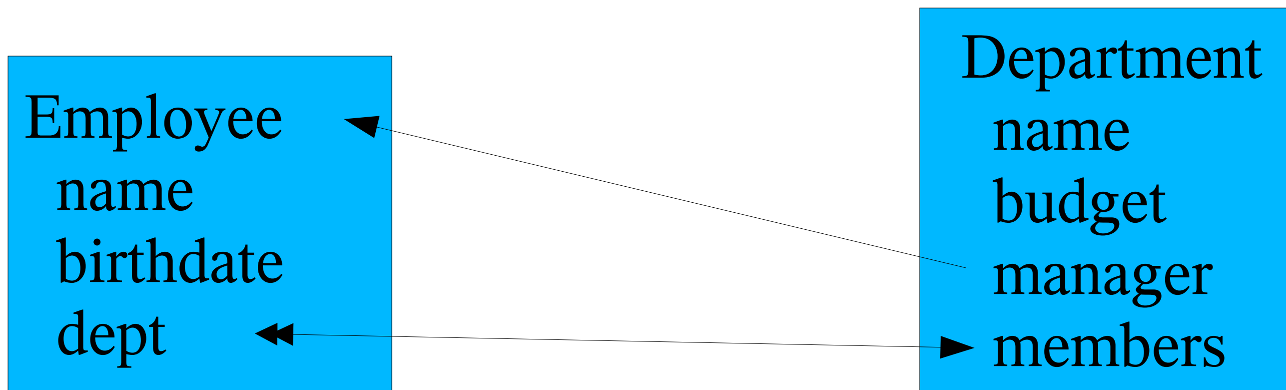
Afternoon agenda

- Versant: Robert Greene
- Objectivity: Leon Guzenda
- ObjectStore: Adrian Marriott
- db4o: Patrick Roemer
- Q&A panel session

What is an “Object Database”?

- Uses object-oriented data model
  - Simple and complex objects, attributes, methods, classes, inheritance, references
- Extends object-oriented programming language with database capabilities
  - Programming language objects are persistent
  - Supports essentially all of the language's data types, including references and collections
  - Supports database queries, transactions, etc.

Simple object schema example



Example declarations

```
class Employee {
  String name;
  Date birthdate;
  Department dept;
  ... methods ...
}
```

```
class Department {
  String name;
  float budget;
  Employee manager;
  Set<Employee> members;
  ... methods ...
}
```

Object databases history

- 1987: Early object database products: Small-talk GemStone, C++ Ontos Vbase
- 1991: ObjectStore, Objectivity/DB, O2, Ver-sant; engineering & specialized applications
- 1993: ODMG 1.0 Standard, distributed ob-jects, Poet, more applications
- 2000+: ODMG 3.0, JDO, wide range of ap-plications, db4o, consolidation

Object Database Manifesto*

- Object-Oriented features: Complex objects, object identity, encapsulation, classes, inheritance, overriding, overloading, late binding, computational completeness, extensibility
- Database features: Persistence, performance, concurrency, reliability, declarative queries

* M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik: The Object-Oriented Database System Manifesto, in *Building an Object-Oriented Database System, Morgan Kaufmann 1992.*

Object model and persistence

- Traditionally:
  - Programming language data is transient, database data is persistent
  - Programming language and database data types and data definition are completely different (an “impedance mismatch”)
- Object databases provide a single, unified data model and persistence model
  - Minimize work for “database” applications

Object database persistence

- Transparent: persistence orthogonal to type
  - Any object type may be persistent or transient
  - Need not write different code for persistence
- Some systems provide transitive persistence
  - “Persistence by reachability”: if an object is persistent, then objects it references are automatically persistent
  - Named root persistent objects and extents
- Some systems use explicit persistence

Source code example

```
public class Employee { ...
  /* ODBMS detects updates to fields & stores on commit */
  /* Also materializes referenced objects when needed */
  public void newAssignment (Department d, float raise) {
     this.department = d;
     this.boss = d.manager;
     this.salary = this.salary + raise;
     }
```

Object Identity

- Every object has a unique immutable object ID
- Objects are referenced by ID (note that pointers are bad in a persistent disk-based representation!)
- Complex objects can be created and an object can be “shared” across complex objects
- In contrast to relational, either Object IDs or keys can be compared for equality

Three kinds of “equal”

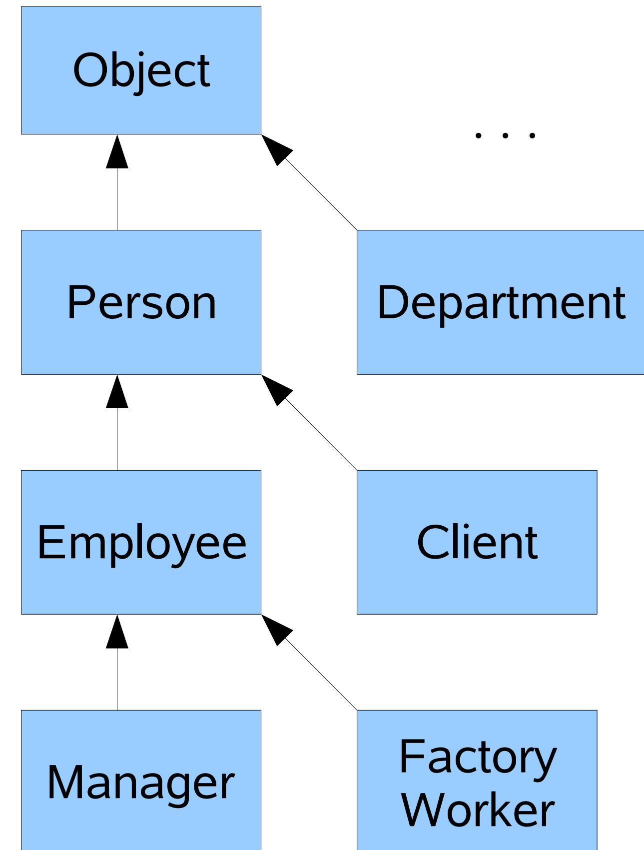
- Object identity: analogous to primary key equality, but allows primary key changes, and foreign keys need not be stored
- Shallow equality: roughly analogous to comparing a tuple's fields in relational
- Deep equality: compare an object and all referenced objects

Data Definition: Classes

- Programming language class definitions become the “DDL” of object database systems
- For “safe” languages like Smalltalk, Java, C#, and Python, most any declaration persistence-capable
- Class definition syntax is extended in some products and standards, e.g. by using Java 5 annotations to define relationship constraints
- ODMG defined language-independent DDL, can share data across programming languages

Class inheritance

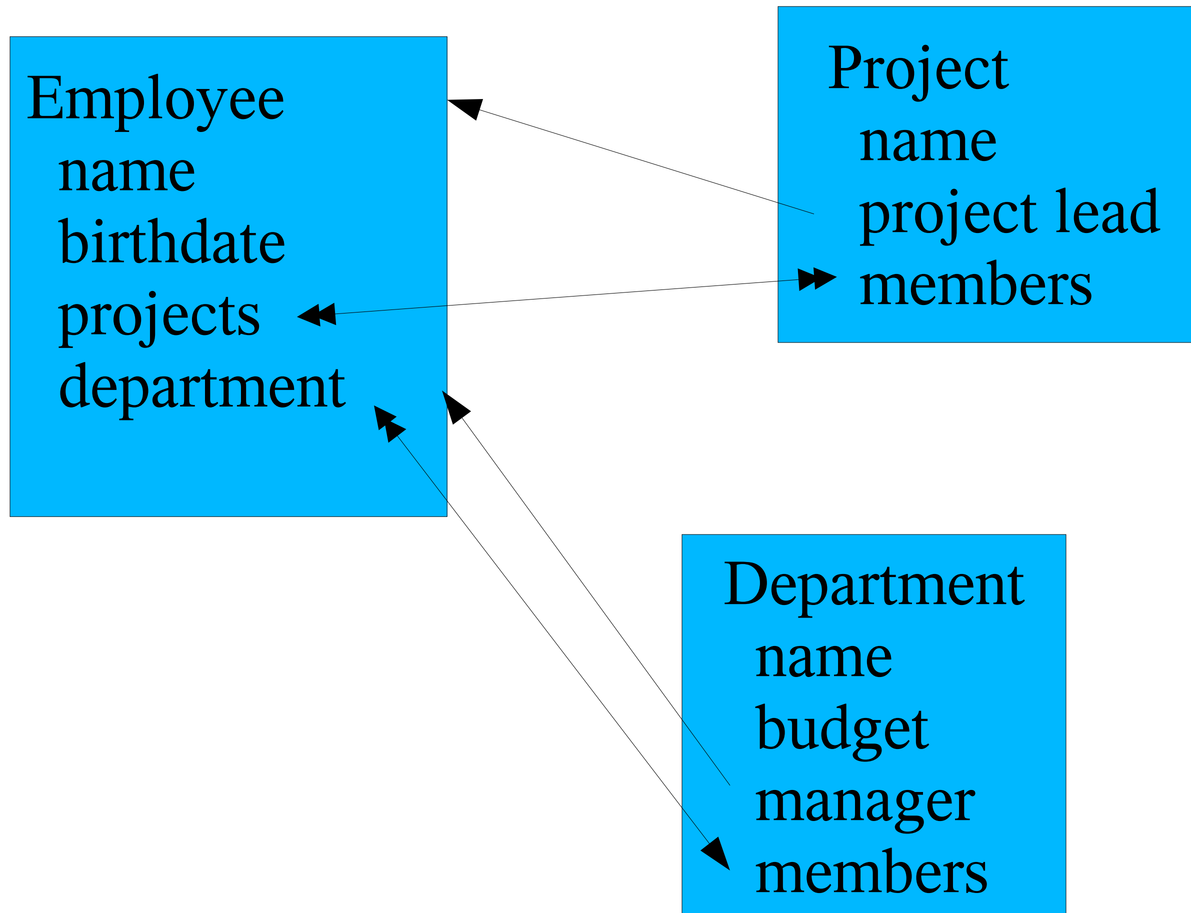
- Widely useful, can greatly simplify schema
- Not available in relational: must simulate w/tables
- Inherit both attributes and behavior
- Allows specific relationship typing, e.g. dept.mgr must be Manager



Object database relationships

- Define typed one-to-one, one-to-many, and many-to-many associations between classes
  - Both directions can be named as attributes
  - Some support “one-directional”
- More powerful than programming language:
  - Maintains bidirectional integrity
- More powerful than relational databases:
  - Independent of key values

Relationship examples



Relationship examples

```
class Employee {
  String name;
  Date birthdate;
  // Works in one Department
  Department dept;
  // Can work on many Projects
  Set<Project> projects;
```

(Note: inverse-attribute syntax omitted for brevity)

```
class Department {
  String name;
  float budget;
  manager Employee;
  Set<Employee> members;
  ... }
```

```
class Project {
  Set<Employee> members;
   ... }
```

Additional relationship features

- Cascade delete
  - Example: dependents deleted with employee
- Automatic pre-fetch
  - Example: fetch employee's department object when bring employee object into memory
- Distributed or federated objects
  - Example: reference to employee in a database at another company site

Encapsulation and methods

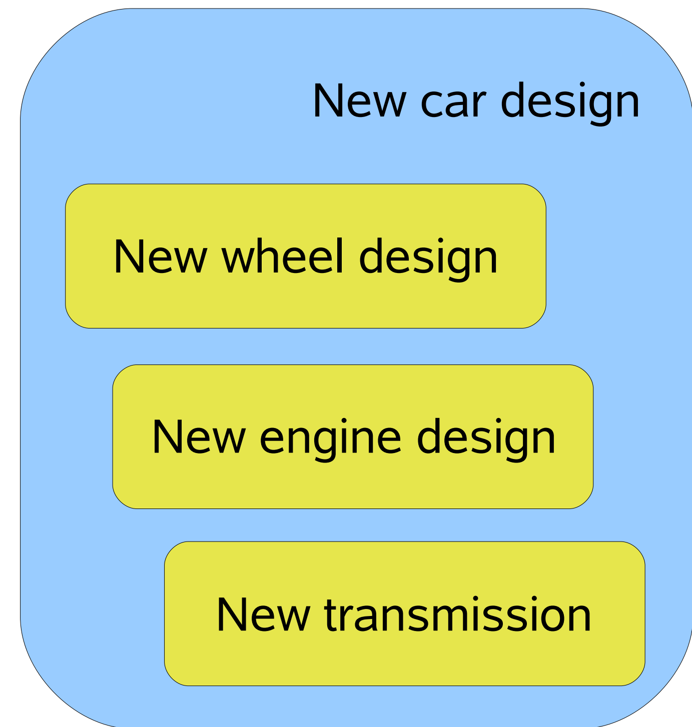
- Important paradigm shift from relational: programming and persistent data integrated
- Methods can execute in separate tier, caching data and offloading database server
- Encapsulation and private attributes allow representation to be hidden and evolved
- Full power of programming language, no need to learn another SQL extension

Object database transactions

- Support ACID properties of traditional transactions, with object-level locking
- Usually also support long transactions, e.g. for design session that lasts hours or days
- Typically independent nested transactions
- Transactions may support copy-on-write versioning of objects and references

Long and nested transactions

- Long transaction might be complete design update
- Nested transactions for individual mods
- Can abort mods w/o abandoning all design updates



Object database queries

- Same goal as original relational model: declarative queries (what you want, not how to find it)
- May require that updates be done through methods, for encapsulation
- Queries expressed in terms of programmer's classes, not an external schema
- Variables and class extent can be queried

Query example

(Syntax varies by product, and w/ EJBQL, JDOQL, ODMG, LINQ)

```
Query q = new Query (
  Employee.class, “manager.salary < salary”);
Collection result = q.select(employees);
```

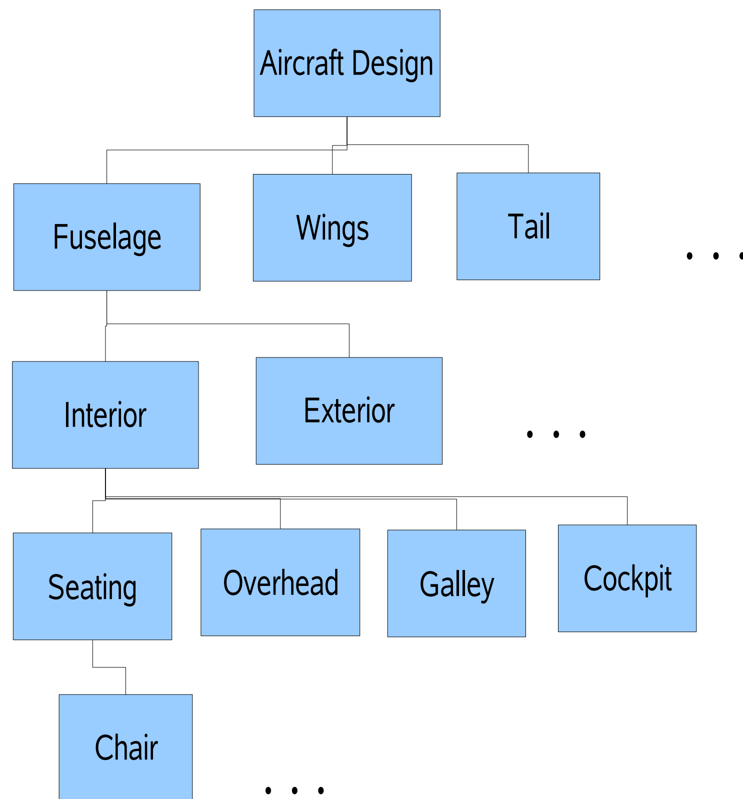
Note the equivalent SQL query is much more wordy, even without the JDBC part:

```
SELECT EMP.ID, EMP.NAME FROM EMPLOYEE EMP, EMPLOYEE BOSS
WHERE EMP.BOSS = BOSS.ID AND BOSS.SALARY < EMP.SALARY
```

Other object database features

- Schema evolution tools
- Database replication, failover
- Federated databases
- Transient fields
- Versioning

Design database example



- Object versions
- Design versions
- Complex objects
- Shared objects
- Multiple hierarchy

Storage level implementation

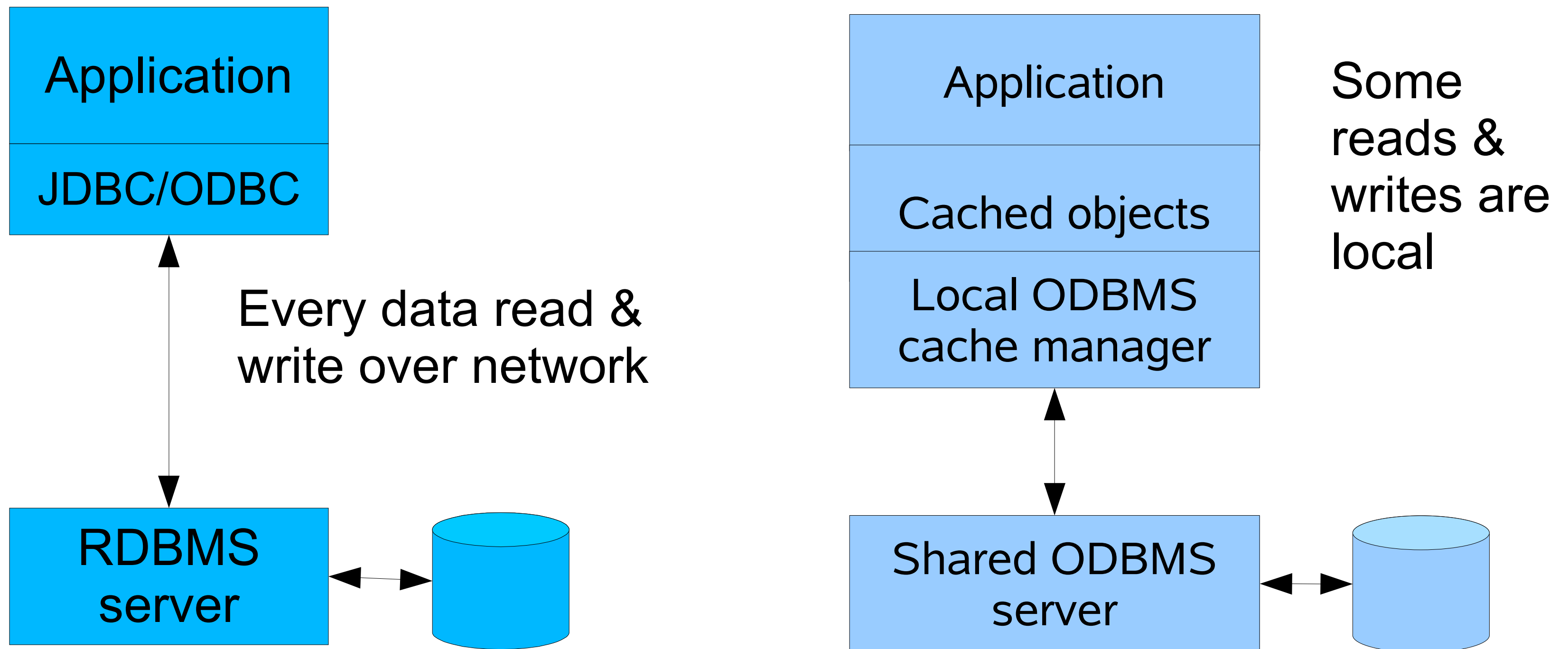
- Similar to relational, but must support class hierarchy, object IDs, etc.
- Indexes on class extents, subclass extents, and user-defined collections
- Indexes and clustering via path expressions, e.g. employee.dept
- Cache synchonization with server

Object ID implementation

Various options:

- Object IDs may be purely logical or partly physical (e.g., page or segment address in upper bits)
- Object ID fields may be “swizzled” into object pointers for efficiency when cached
- Object IDs may allow for “federated” links across databases (with level of indirection)

ODBMS vs traditional architecture



Important performance issues

- Object database can cache data in the same physical memory as the application
  - An inter-process or inter-machine call costs 1000x a local read or write
- Traditional DBMS/applications must convert data from on-disk representation to program representation, even if cached off disk
  - Object database can cache in program representation, and disk conversion is fast

Detecting object updates

Alternative ODBMS implementations:

- Explicit writebacks by programmer
- Before-and-after comparison
- Post-processing: bytecode enhancement pre-execution, set dirty bit on each putfield
- Page faults: detect first write (and read) by faulting on memory references

Cache synchronization

- Pessimistic: implicitly or explicitly lock all cached objects until transaction commit
- Optimistic: work on cache until commit, then fail if modified on re-fetch
- Flexible: application can accept “dirty reads” or other consistency level
- Lazy: defer fetch, use “hollow” objects
- Eager: pre-fetch based on relationships

ODMG standard components

- Standardized object model and ODL
- Standardized query language OQL
- Smalltalk binding
- C++ binding
- Java binding

Standardization history & future

- Note: standard came *after* products
  - Products also more diverse, feature-rich
  - Therefore less portability than with SQL
- ODMG created some commonality in 1993
  - Weaker, but RDBMS portability not great either!
- OMG has now taken the ball from ODMG
  - ODMG 3.0 was last version, except JDO
- For Java, now have JDO and EJBQL

Object databases vs alternatives

1. Relational database systems: access tables via ODBC, JDBC
2. Object-relational database systems with nested relations, abstract types, functions
3. Object-relational mapping: map relational data to OO language objects
4. Object serialization: persistence via external-ization as strings

Relational database example

Employee

ID	Name	Birthdate	Dept

Project

ID	Name	Lead

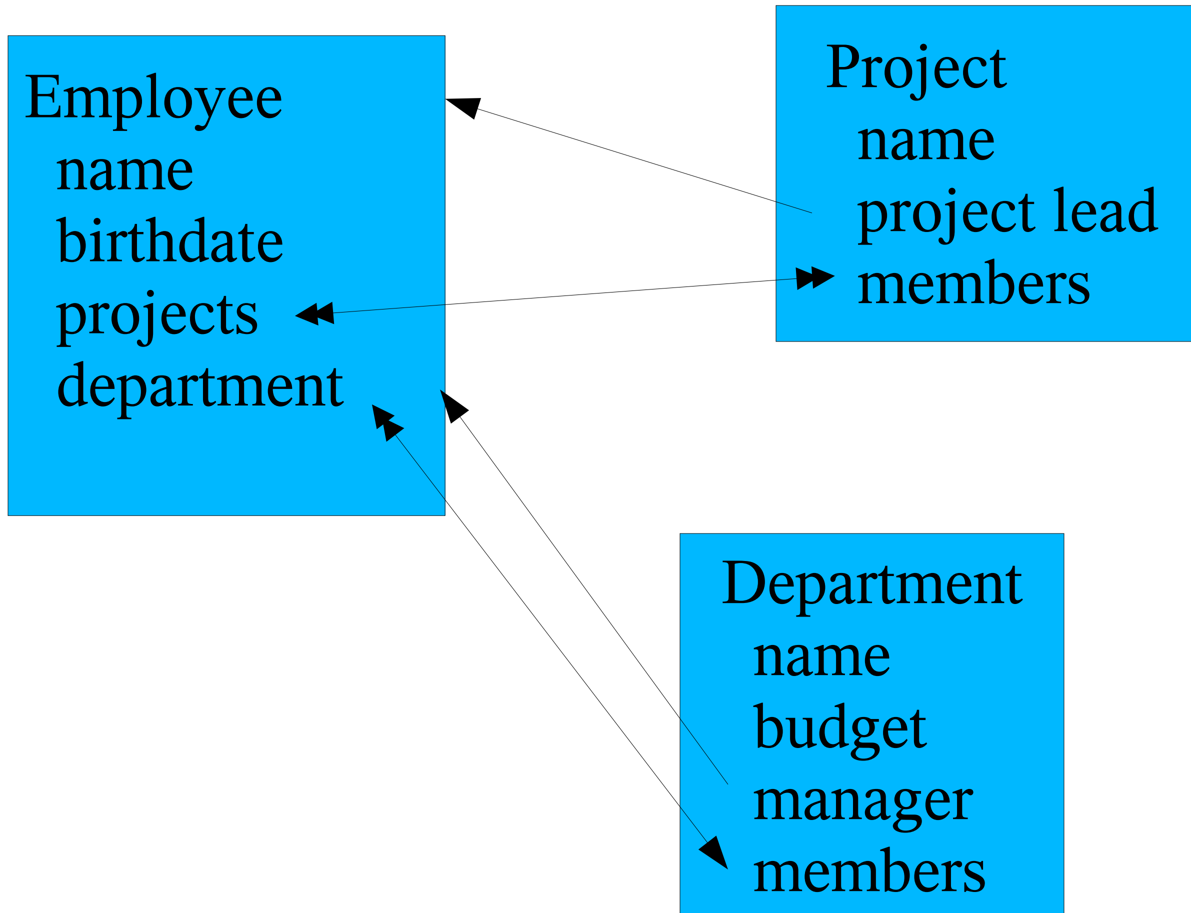
Employee-Projects

EmpID	ProjID

Department

ID	Budget	Mgr

Object model equivalent



Relational databases

- Advantages
  - Often corporate standard with existing data
  - Simpler model than objects, where tables work
  - Often more existing tools and applications
- Disadvantages
  - Impedance mismatch: must translate models, becomes much of application code/complexity
  - Lacks extensibility, inheritance, identity, path expressions, encapsulation, methods …
  - Performance: remote calls, translation

Object-relational databases

- Add new data functionality to relational:
  - User-Defined Types (UDTs): can be used in table cell or as row of a table
  - REF: pointer to the above via object IDs
  - Nested tables can be stored in a cell
- Computational completeness via methods on UDTs, and SQL stored procedures
- SQL-1999 and SQL-2003 standards

Object-relational databases

- Advantages
  - Easy migration from pure relational as subset
  - OO features appear as SQL extensions
- Disadvantages
  - Not integrated with programming language
  - Performance and conversion issues remain
  - OO features are awkward and complex add-on, e.g. querying nested tables, ADTs

Object-relational mapping

- Automates process of converting objects to and from relational representation
  - Define mapping between relational tables and programming language classes
  - Existing schema, existing classes, or both
- Various standards and products for this
  - Examples: Hibernate, EJB Persistence / TopLink, JDO / Kodo / JPOX, ADO.NET

Object-relational mapping

- Advantages:
  - Access existing relational data, translation code is automatically generated
- Disadvantages:
  - Lower performance than pure object database*
  - Must be aware of both data models, systems
  - Some functional limitations and restrictions due to mapping

*Zyl, Kourie, Broake: Comparing the Performance of Object Database and ORM Tools, Proceedings SAICSIT 2006.

Object serialization

- Serialization converts objects into a byte sequence that can be stored or transmitted
  - Object is reconstituted at a later time (or another place)
  - A tree or graph of referenced objects can be serialized/deserialized, following pointers
- Built into Java from the start
  - Implement the java.io.Serializable interface

Object serialization

- Advantages:
  - Simple, built-in implementation
- Disadvantages:
  - Manual calls to serialize/deserialize
  - Very slow compared to object database

Summary of alternatives

- Relational is widely used, but often maligned on impedance mismatch
- Object/Relational SQL extensions not used widely: complex and not integrated
- Object/Relational mapping is best and most popular compromise if must use RDBMS
- Serialization is used mostly for special purposes

Key points from this session

- Object database = OO PL + DB
- Integration avoids “impedance mismatch”
- Powerful model: relationships, collections, inheritance, versioning, long transactions
- Large performance advantage when working set can be cached in-process or when database load is reference-intensive

Conclusion

- Q&A session now
- Afternoon session logistics
- Good source for more info: odbms.org
- My contact info: see cattell.net