Database meets OOP

Chicago, May 17th 2007

php|tek

Lukas Kahwe Smith (lsmith@optaros.com)

http://pooteeweet.org/files/phptek07/database_meets_oop.pdf

optaros

About Myself

- Lukas Kahwe Smith
- Started using PHP in the late PHP3 days
- Joined PEAR sometime in 2002 to work on MDB
- Began posting on internals@lists.php.net in 2003
- Started the unofficial official PHP todo list in 2005
- Makes his living as a consultant for Optaros
    - OSS consulting and system integration firm
    - We are always hiring
        - http://optaros.com/en/company/careers
- Can be reached
    - Via Email: lsmith@optaros.com
    - On IRC: “lsmith” on efnet, freenode, ircnet ..
    - In Person: Here or at Ultimate Frisbee tournaments

© Optaros 2007. All rights reserved.

optaros

API and SQL Abstraction

- API Abstraction
  - PDO
    - Unified OO Client API (more or less) for all drivers
    - Exceptions, Iterators
    - Portability modes
- SQL Abstraction
  - MDB2 (PHP4/5), Creole (PHP5), ezc:Database (PHP5)
    - Unified OO Client API
    - Data type abstraction
    - Schema reading and writing
    - More portability modes
- Lots of manual work
  - Object => Array+SQL => Database
  - Database => SQL => Array => Object

© Optaros 2007. All rights reserved.

Object-relational impedance mismatch

- SQL is declarative and not object oriented (nor procedural)
  - Poor performance when not thinking in sets
- Most DBMS are poor at modeling domain logic and behavior
  - Data types (array's vs. tables), inheritance etc.
- Relational theory does not mandate identifiers
  - Primary keys are an afterthought to fit OO requirements
- OO traverses data along their relations vs. joining data sets
  - $forum = new Forum(1); foreach ($forum->threads) {..}
- Unclear who manages state, especially in the stateless web
- UI should prevent inputting incorrect data
  - Integrity constraints need to be known inside the UI
- DBA vs. Developer (aka cultural impedance mismatch)
  - Its hard to be good at both, even harder to think as both

© Optaros 2007. All rights reserved.

optaros

Object-relational impedance mismatch

- Department object with all employee's inside a property
  - `foreach ($dept->employees as $e) echo $e->name;`
  - Creating the instance of $department requires reading
    - Department data
    - Data about all employees
  - SELECT d.*, e.* FROM department d, employee e
    - Reads a lot of redundant data!
    - Gets even worse if you have more 1:many properties
      - Inventory
      - Sales records
      - etc.
    - One solution is to move the ”join“ into the middle tier
      - This by passes all the nice power of the DBMS and leads to a lot more queries

© Optaros 2007. All rights reserved.

Schema Challenges

- Simple example
  - Person Object
  - Student specialization of Person Object
  - Grad student specialization of Person Object
- Approach #1
  - Person table with fields for persons + person PK
  - Student table with fields for students + person PK
  - Grad student table with fields for grad students + grad student PK
    - Union needed when fetching data for multiple types
    - All data has to be moved when changing type
    - Issues when setting up FK's

© Optaros 2007. All rights reserved.

optaros

Schema Challenges

- Simple example
  - Person Object
  - Student specialization of Person Object
  - Grad student specialization of Person Object
- Approach #2
  - One table with all specific fields + type field
    - Potentially a lot of empty columns

© Optaros 2007. All rights reserved.

optaros

Schema Challenges

- Simple example
  - Person Object
  - Student specialization of Person Object
  - Grad student specialization of Person Object
- Approach #3
  - Person table with person specific fields + person PK
  - Student table with student specific fields + person PK
  - Grad student table with grad student specific fields + person PK
    - 3 Table (outer) join when trying to retrieve information
    - Gets worse when multiple types split off
      - Freshmen-, Senior-, Grad student

© Optaros 2007. All rights reserved.

optaros

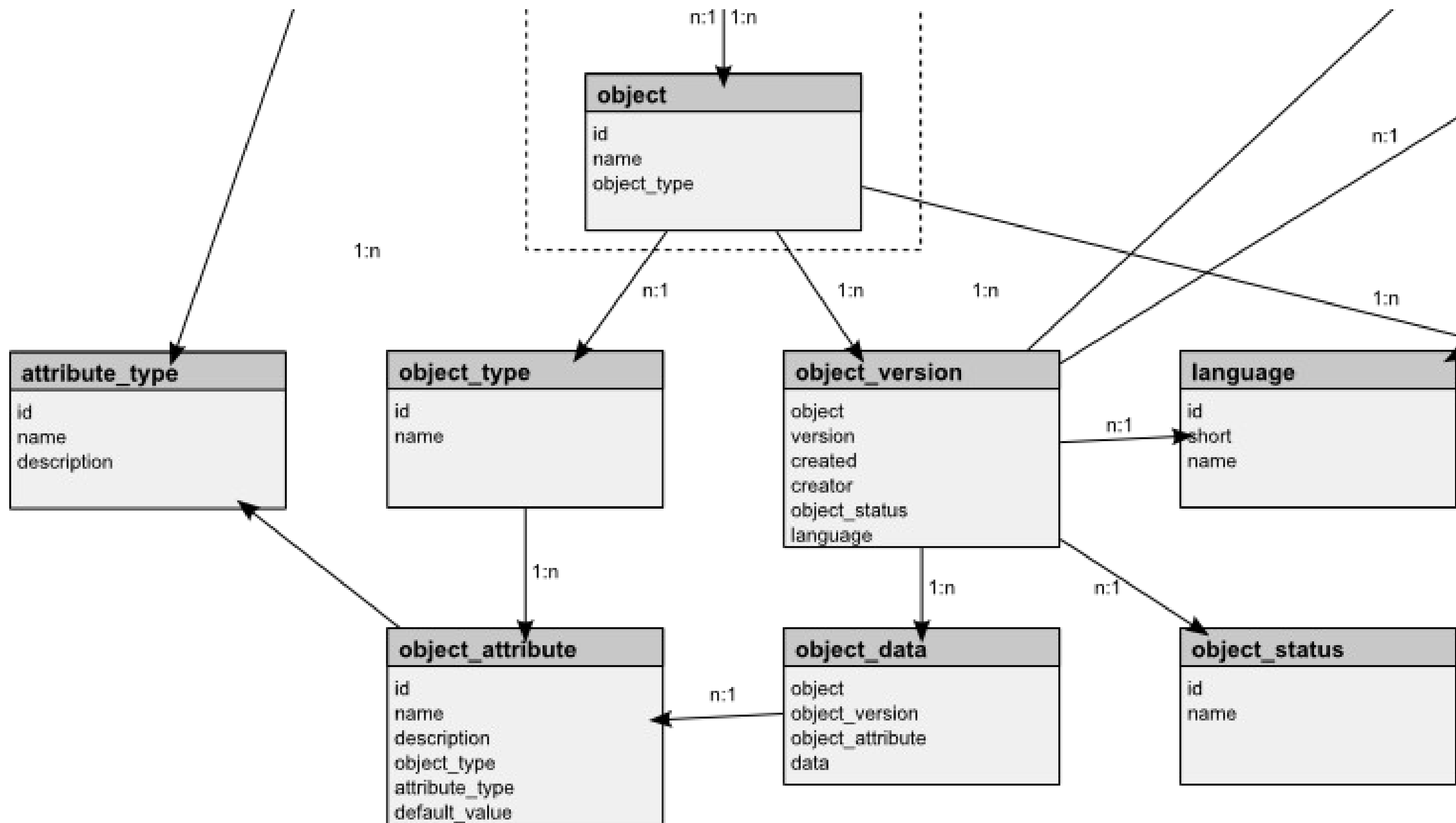
Schema Challenges

- Approach #3 - the academic approach
  - Enables proper leverage of relational tools (FKs etc.)
  - Requires more (join happy) queries in some cases
- Approach #2 - the pragmatic approach
  - Easy to implement and query
- Approach #1 – the sometimes useful approach
  - Can be more efficient than #2 and #3 in specific cases
- Look at your use cases
  - If you frequently need to search only using parent type fields, then you really want approach #3
  - Use #2 if you can evade performance issues
    - Remember best query is one you don't execute
  - If you usually only search one type at a time and you will maintain FK constraints in your application layer consider approach #1 as a performance optimization
    - Hope you never add another inheritance level

© Optaros 2007. All rights reserved.

optaros

Fancy OO-Structures



Kore Nordmann (eZ Systems)

© Optaros 2007. All rights reserved.

optaros

No more SQL

- Allows representing SQL queries and results in OO terms
  - Return nested instances of objects, rather than one dimensional arrays
  - Objects provide all the boring CRUD handling
  - Encapsulates many advance features such as pagination (some DBMS require complex subselects), profiling etc.
  - Tighter integration with middle tier language
    - Inheritance
    - Data types
    - Syntax checking/auto completion
- Facilitate writing database-type independent queries
  - That may not be using a DBMS at all (web services etc.)
- Popular approaches are ActiveRecord and ORM's
  - Most layers try to do best of all worlds
  - Usually build on top of a DBAL

© Optaros 2007. All rights reserved.

Oh No! No more SQL?

- Surrogate key hell
- Legacy schema hell
- Inefficiencies
  - More code to load
    - Somewhat solved through byte code caches
  - Memory consumption
  - Parsing time
  - More queries that tend to be less optimized
  - Dual schema management
- Overloading and delegation can become hard to grasp
  - Step-by-step debuggers are your friend
- Why bother?
  - Use an OO database like Cache and db4o

© Optaros 2007. All rights reserved.

optaros

Wrapping Frameworks aka ActiveRecord

- Schema ownership in the DBMS
  - Naming conventions (legacy databases require mapping)
  - Do not work well with all kinds of database objects
    - VIEWs, Stored Routines, etc.
- Meta programming with runtime analysis of DBMS schema
  - Use Caching and flexible triggers to lessen performance impact of runtime analysis
  - Use annotations to add new behaviors
  - PHP Implementations
    - CakePHP, Zend_Db_Table, PHP on Trax
- Class generation from the DBMS schema
  - Generates the base and custom class, only the later is filled with application specific business logic
  - Modify class templates to add new behaviors
  - PHP Implementations
    - Qcodo

© Optaros 2007. All rights reserved.

optaros

Mapping Frameworks aka O/R Mapper

- Schema is derived from the classes, annotations or XML
  - Extend from base classes, model classes generated or implement interface with a persistence manager
  - Often more limited in using DBMS specific schema magic
- Much more decoupled from DBMS
  - Do not require a database schema to begin coding
  - More more lax about naming conventions
- Different approaches:
  - Query-By-Example
  - Query-By-API
  - Query-By-Language
- PHP Implementations:
  - ezc::PersistentObject, Propel, Doctrine, PEAR::DB_DataObject

© Optaros 2007. All rights reserved.

optaros

Query-By-Example

- Example
  - $user = new User();
  - $user->name = 'Smith';
  - $all_smiths = Query::execute($user);
- Very straightforward and easy to use
- Cannot handle more complex queries
  - “find all Persons who's name is NOT Smith”
  - “find all Persons who's name is Smith or Johnson”
- Syntax highlighting/checking and auto completion
  - Except for columns (until IDE's learn virtual properties)
- PHP Implementations
  - PEAR::DB_DataObject

© Optaros 2007. All rights reserved.

optaros

Query-By-API

- Example
  - `$q = new Query();`
  - `$c = new EqCriteria('Name', 'Smith');`
  - `$q->Form('Person')->Where($c);`
  - `$all_smiths = $q->execute();`
- Can handle arbitrary complex queries
  - At the expense of being more verbose
  - More coupled to the schema
- Syntax highlighting/checking and auto completion
  - Except for identifiers (Person, Name, etc.)
- PHP Implementations
  - ezc::PersistentObject, Propel, Doctrine, PEAR::DB_DataObject etc.

© Optaros 2007. All rights reserved.

optaros

Query-By-API

- Looking even less like SQL
  - `// criteria API`
  - `$c = new Criteria();`
  - `$books = BookPeer::doSelect($c);`
- BookPeer::doSelect() can encapsulate additional logic or alternative methods can provide additional logic/optimizations
  - `$books = BookPeer::doSelectJoinAuthor($c);`
- Disadvantage is that to leverage the full potential of the DBMS the Peer class needs to be expanded
  - Increases the code size
  - Code can either auto generated or added manually

© Optaros 2007. All rights reserved.

Query-By-Language

- Example
  - `$q = "FROM Person p WHERE p.name = 'Smith'"`
  - `$all_smiths = $conn->query($q);`
- Derived from SQL, while being more concise and OO aware
  - Does need a certain amount of retraining
  - No syntax highlighting/checking or auto completion
    - C# LINQ provides a language level query language
- Can handle arbitrary complex queries
  - Only a subset of SQL
- More higher efficiency in what data is read without having to add specialized methods to base classes
- PHP Implementations
  - Doctrine with its own DQL syntax

© Optaros 2007. All rights reserved.

Query-By-Language

```
// using many sql queries for object population

$users = $conn->getTable('User')->findAll();

foreach ($users as $user) {
    print $user->name;
    // fetch phonenumbers one after another (*)
    foreach ($user->Phonenumber as $phonenumber) {
        print $phonenumber;

}  }

// using only one sql query for object population

$users = $conn->query('FROM User.Phonenumber');

foreach ($users as $user) {
    print $user->name;
    foreach ($user->Phonenumber as $phonenumber) {
        print $phonenumber;

}  }
```


© Optaros 2007. All rights reserved.

optaros

Aspects Galore

- Caching
- Optimistic/Pessimistic Locking
- Callbacks and overloading/iterator magic
  - Cascade on update/delete emulation
  - Versioning, auditing
- Multi-Language
- Hierarchies
  - Nested set, adjacency model, materialized path
- Permissions
- Managing state/work flows
- Inheritance
- Form generation/validation
- Abstract data types

© Optaros 2007. All rights reserved.

References

- Doctrine (PHP5)
  - http://www.phpdoctrine.net
- Propel (PHP5)
  - http://propel.phpdb.org/trac/
- ezc::PersistentObject (PHP5)
  - http://ez.no/community/articles/the_persistentobject_ez_component_putting_relations_where_relations_belong
- PEAR::DatabaseObject (PHP4/PHP5)
  - http://pear.php.net/package/DB_DataObject
- Zend_Database_Table (PHP5)
  - http://framework.zend.com/manual/en/zend.db.table.html
- Qcodo (PHP5)
  - http://qcodo.com/

© Optaros 2007. All rights reserved.

optaros

References

- Qcodo developers on ActiveRecord
  - http://www.qcodo.com/documentation/article.php/6
- Good overview of object-relational impedance mismatch
  - http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx
- These slides
  - http://pooteeweet.org/files/phptek07/database_meets_oop.pdf

© Optaros 2007. All rights reserved.