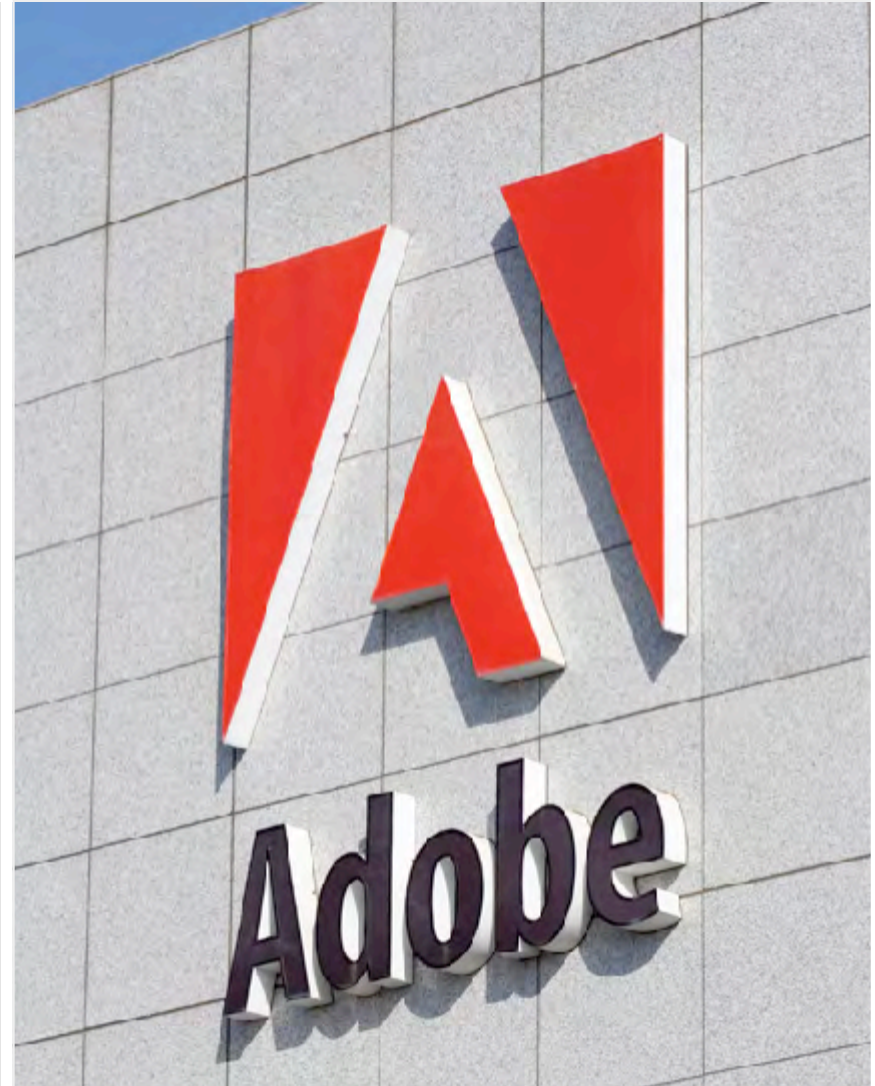


Objects & Persistence

Sean A Corfield

Senior Computer Scientist

Adobe Systems, Inc.



What is this about?

- ▣ As we move from procedural code to object-oriented code we have to deal with the issue of loading and saving objects
- ▣ One of the biggest stumbling blocks for many developers is the mapping between objects and database tables
- ▣ This talk looks at the mapping issue and some of the tools that can make the whole loading / saving objects thing easier

Who am I?

- ▣ Team Lead, Hosted Services, Adobe Systems Incorporated
- ▣ Formerly Senior Architect, Macromedia IT
- ▣ 25 years of IT (I'm persistent!)
- ▣ 14 years of object-oriented development
- ▣ 8 years of web development
- ▣ 5 years of ColdFusion development

Overview

- ▣ Definitions & Basic Persistence – The boring bit!
- ▣ Persisting Objects with Objects
- ▣ Objects, Tables, Relationships
- ▣ Frameworks and Other Animals
- ▣ Code – The exciting bit!

The Basics – Definitions

- Persistence
- CRUD
- Bean
- DAO
- Gateway aka Data Gateway or ...
- ORM

Persistence

- The process of making data persistent
- Persistent data is still there after you switch the machine off and back on!
- Persistent data therefore lives on disk, not in memory
- Note: application scope is shared memory – not persistent!

CRUD

- Acronym for the four basic persistence operations
- Create / Read / Update / Delete
- Often (mis)used to include “list” (bulk query) operations as well

Bean

- Loosely encapsulates a set of properties (data)
- Has getXxx() methods (“getters”) for each “public” property xxx
- Typically has setXxx(value) methods (“setters”) for each property xxx
- Setters may be public or private – a private setter represents a property that is essentially readonly to the outside world
- The term comes from Java

DAO

- Data Access Object
- An object that provides CRUD operations for a specific object type
- Load an object, persist an object
- Often create / update operations are combined in a save() method
- e.g., PersonDAO provides create / read / update / delete for a Person object

Gateway

- a.k.a. Data Gateway and a few other names
- An object that provides various bulk query operations (usually just selects but sometimes updates / deletes)
- Methods usually return a query object (record set)
 - Converting to an array of objects (common in Java) can be expensive and often unnecessary
 - ColdFusion has native language support for query type – which Java does not

ORM

- Object Relational Mapping
- The process of translating between object structures in memory and (relational) tables and columns in a database
- We map types, names, relationships...
- As we shall see shortly: Object NEQ Table!

Persistence in Action

- Cover a variety of techniques:
 - Basic “Old School” database interactions
 - Introducing CFCs to organize code
 - Data Access Objects, Beans, Gateways
- Look at objects, tables and the relationships between them

Persistence: Old School

- ❑ `<cfquery>` etc inline in `.cfm` pages

- ❑ `edit.cfm` (form) submits to `save.cfm` (`<cfquery>` / `<cfupdate>`) redirects to confirmation page

- ❑ `edit.cfm`:

```
<cfquery name="item" ..>  
  SELECT .. FROM ..  
  WHERE id = <cfqueryparam ..>
```

```
</cfquery>
```

```
<cfform action="save.cfm" ..>  
  .. #item.name ..
```

```
</cfform>
```

- ❑ `save.cfm`:

```
<cfquery ..>  
  UPDATE .. SET name = <cfqueryparam ..>
```

```
</cfquery>
```

```
<cfrelocate ..>
```

Persistence with CFCs (1)

▣ Move all SQL into a CFC

- ▣ Maybe one CFC, maybe one per “concept”
- ▣ Create in Application.cfc (onApplicationStart)

▣ Call from action page

▣ Benefits

- ▣ Encapsulates database activity
- ▣ Isolates change (e.g., from inline SQL to stored procs, from MS Server to MySQL, from database to file system)

▣ ItemStore.cfc:

```
<cffunction name="getItem"
  returnType="query" ..>
  <cfargument name="id" ..>
  <cfset var items = "">
  <cfquery name="items" ..> .. </cfquery>
  <cfreturn items >

</cffunction>
```

▣ edit.cfm:

```
<cfset item =
  application.itemStore.getItem(id)>

<cfform action="save.cfm" ..>
  .. #item.name# ..

</cfform>
```

Persistence with CFCs (2)

- ❑ Create beans to contain object data and data access objects for each one
 - ❑ Simple partitioning of queries
 - ❑ Typically one object / DAO per table at this point
 - ❑ We'll expand on this in the next few slides!
- ❑ What about aggregate operations? (list() etc)
 - ❑ Should really create Gateway CFCs
 - ❑ OK to have a simple list() method in your DAO if you have no other methods that would need a Gateway...

❑ Item.cfc - **bean**

❑ ItemDAO.cfc:

```
<cffunction name="getItem"  
    returntype="Item" ..>  
    ..  
    <cfreturn createObject("component",  
        "Item").init(items.name, ..) >  
</cffunction>
```

❑ edit.cfm:

```
<cfset item =  
    application.itemDAO.getItem(id)>  
<cfform action="save.cfm" ..>  
    .. #item.getName()# ..  
</cfform>
```

One DAO per table?

- Usually but not always!
- A DAO works with a single object
- An object may span multiple tables
- e.g., “member” object on macromedia.com has core profile table and extended profile table for optional attributes
- A single table may also contain multiple object types

One Object : One Table

- ❑ Customer table (FIRSTNAME, LASTNAME, BILLTO_STREET, BILLTO_CITY, BILLTO_STATE, BILLTO_ZIP etc)
- ❑ Customer object (firstName, lastName, billToStreet, billToCity, billToState, billToZip etc)
- ❑ CustomerDAO is a simple 1:1 mapping

Multiple Objects : One Table

- Customer table as before
- Customer object (firstName, lastName, billToAddress, shipToAddress etc)
- Address object (street, city, state, zip etc)
- CustomerDAO loads / saves Customer and both Address objects from / to single table (since Address objects are dependent on Customer)
- AddressDAO is optional here (but you could write one that operates “per Customer”)

One Object : Multiple Tables

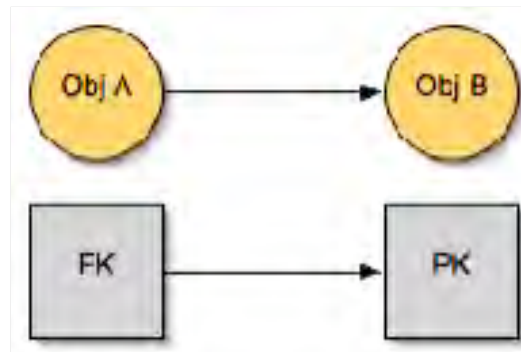
- Customer object with optional preference attributes (e.g., newsletter: HTML, text – absence of attribute means no preference)
- Customer table has core attributes
- Preference table has customer ID, preference name, preference value
- CustomerDAO loads / saves Customer from / to multiple tables

Objects vs Tables

- ❑ Objects exist in memory and form a graph
 - ❑ By reference: Person has-a home Address
 - ❑ `setHomeAddress()` / `getHomeAddress()`
- ❑ Tables exist in database
- ❑ Need primary / foreign keys
 - ❑ PERSON table has ID, ADDRESS table has ID
 - ❑ PERSON table has HOME_ADDRESS_ID column
- ❑ Simplest solution is to maintain keys in-memory in objects

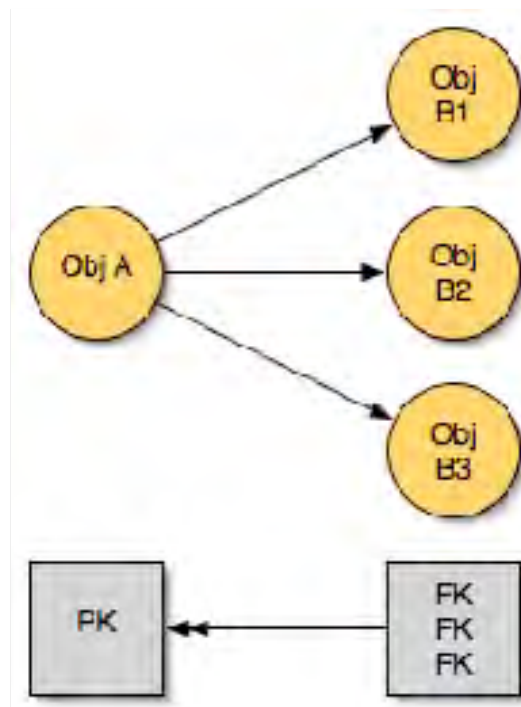
Object Relationships (1)

- 1:1 – just a variable reference
- Simple Foreign Key / Primary Key relationship in database
 - e.g., CUSTOMER has ADDRESS_ID (FK) to ADDRESS's ID (PK)



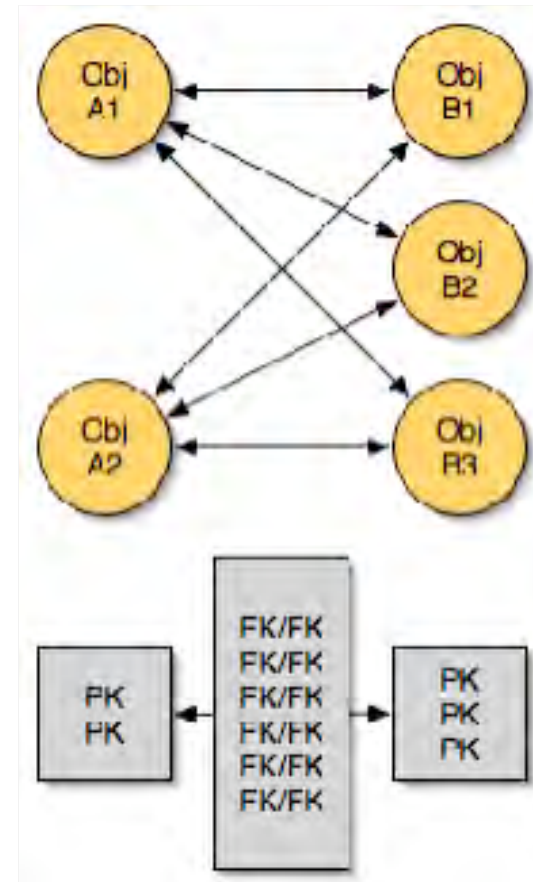
Object Relationships (2)

- 1:n – array or struct (of references)
- Multiple rows in the database – the PK / FK relationship goes the opposite way to how 1:1 relationships are usually done



Object Relationships (3)

- n:m – array or struct (in both objects)
- Relationship may be key-based (PK / FK) or a separate lookup table in database (pairs of FKs)



SQL is just so tedious...

- ...there has to be a better way!
- What if you didn't have to write any SQL?
- What if you could just point some code at a database and have it write the SQL for you?

Frameworks can help!

- They can introspect the database structure and generate SQL automagically
- You specify relationships between the tables (hasOne / hasMany / belongsTo etc)
- They manage creation and persistence of bean-like “record” objects (and cascades)
- Some let you extend the “record” objects to act as full-blown business objects
- Note: these are typically table-based so you still need to map complex objects (to multiple tables)

What frameworks?

- Arf! – Joe Rinehart - <http://clearsoftware.net/>
- ObjectBreeze – Nic Tunney - <http://nictunney.com/objectBreeze.cfm>
- Transfer – Mark Mandel - <http://compoundtheory.com/transfer/>
- Reactor – Doug Hughes - <http://www.doughughes.net/>

Arf!

- ❑ ActiveRecord Factory – Rails-inspired
- ❑ Basic DAO / gateway (CRUD and list) methods, QueryIterator, minimal validation support (error collection, XML dictionary)
- ❑ User defines skeleton CFC per table
 - ❑ CFC defines relationships (and can override CRUD / list() operations)
- ❑ Generates CFCs / injects methods on first run
- ❑ Introspects database using ServiceFactory
 - ❑ Generic JDBC so database support is “automatic”
 - ❑ Assumes certain naming conventions (singular table names, “tableId” or “id” for primary key)
- ❑ Video tutorials on website, good QuickStart included with download, no sample application
- ❑ Available via SVN and download (no active development is planned)

objectBreeze

- ❑ Basic DAO / gateway (CRUD and list) methods, QueryObject (iterator), no validation support
- ❑ Relationships defined at runtime – no skeleton CFCs, no CFC generation
- ❑ Explicit object collections (assumes user defined gateways)
- ❑ Introspects database using DB-specific commands / tables (MS SQL Server, MySQL, Oracle, PostGreSQL)
- ❑ Some documentation on website, simple sample application
- ❑ Download only (currently v0.9.6 beta – although 1.0 is coming very soon)

Transfer

- Basic DAO (CRUD) methods – user defines their own gateways, no validation support
- XML configuration, CFC generation
 - XML can specify CFML (methods) to inject into generated business objects
- Event model – before/after observers for CRUD operations
- Allows aliasing of tables and columns
- Documentation on website and with downloaded files, sample application (blog) available separately
- Download only (currently v0.3 beta)

Reactor

- ❑ Separate DAO and gateway objects, built-in validation (against database types)
- ❑ XML configuration, CFC generation using XSLT, generates “shell” CFCs to contain custom methods
- ❑ Introspects database using DB-specific commands / tables (MS SQL Server, MySQL with support for Oracle planned)
- ❑ Event model – before / after listeners for DAO operations
- ❑ Allows aliasing of tables and columns
- ❑ Supports multi-table linking relationships
- ❑ Full object representation of “prepared query” via gateways
- ❑ Some documentation on website, sample applications (blog, contact manager), unit tests (based on cfcUnit)
- ❑ Available via SVN and download (very active development!)

Framework USPs

□ Unique Selling Points:

- Arf! is by far the simplest to use: “quick'n'dirty”, broadest database support (ServiceFactory) – no active development being done
- objectBreeze: explicit object collection management – just gone 1.0
- Transfer: custom methods can be added to XML config (and are generated into the “record” objects)
- Reactor: full-featured & robust, full metaquery and gateway support, has killer sample app

Another approach

- Code generators:
 - Explicitly generate a CFC for each table
 - You take the generated code and use it (and modify it) in your application
 - Many examples out there, e.g., HomeSite CRUD CFC Generator – Stan Winchester - <http://www.aftershockweb.net/forums/messages.cfm/ThreadId/50>
- Some ORM frameworks generate CFCs containing SQL that you can build on anyway (e.g., Reactor)

Code Examples

- Simple wiki application
- “Traditional” persistence
- Persistence via CFC
- Persistence via DAO and “bean”
- Persistence with each of four frameworks
 - I will only show the Reactor version
- Notes:
 - DAO in these examples includes list() method (because there is no other need for a gateway)
 - The example uses just one table (so no example of relationship management via the frameworks)

Persistence with Arf!

- DAO wraps Arf!
- “bean” replaced by ActiveRecord named for the table in the database and exposes column names
- DAO has no save() method – save.cfm calls ActiveRecord save () method directly – good!

Persistence with objectBreeze

- ❑ DAO wraps objectBreeze service
- ❑ No “bean” – but getXxx() methods become getProperty(“xxx”) (leaks implementation) – bad!
- ❑ Alternative: keep “bean” to hide objectBreeze MasterObject type – extra work required in DAO and “bean” would just delegate to MasterObject

Persistence with Transfer

- ❑ DAO wraps Transfer service
- ❑ “bean” replaced by TransferObject and uses property names instead of column names – good!
- ❑ DAO readById() method needs to wrap Transfer get() method in try/catch
- ❑ DAO readByTerm() requires hand-coded SQL / create object / populate from query idiom
- ❑ DAO list() is a hand-coded gateway method

Persistence with Reactor

- DAO wraps Reactor
- “bean” replaced by Reactor's AbstractRecord and can use property names instead of column names – good!
- DAO has no save() method – save.cfm calls AbstractRecord save() method directly – good!
- DAO list() method uses Reactor's gateway to retrieve and sort the records – no SQL so that's good too!

Summary

- ❑ 1:1 object-to-table mapping is fine in many cases
- ❑ Frameworks let you avoid writing SQL
 - ❑ Reactor handles complex queries as well as CRUD!
- ❑ Frameworks don't really do any ORM
 - ❑ Transfer / Reactor map table and column names
 - ❑ They do however manage cascading CRUD
 - ❑ Great for applications with a 1:1 object-to-table mapping
- ❑ Frameworks are a very useful tool but you still need to think about object design, separate from relational table design (object model vs data model)

Q&A

Objects & Persistence

Sean A Corfield, Adobe Systems, Inc.

sean.corfield@adobe.com

sean@corfield.org

An Architect's View – <http://corfield.org>

Better by Adobe.™