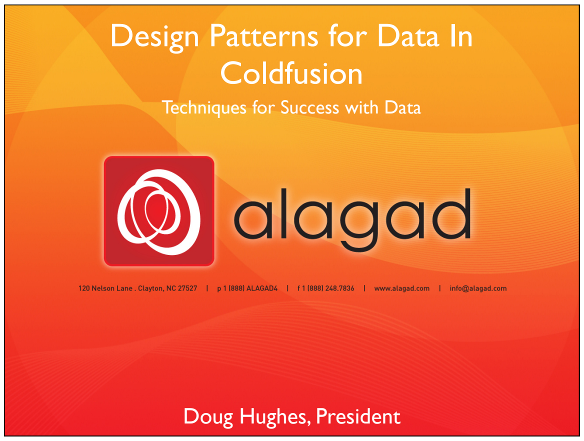
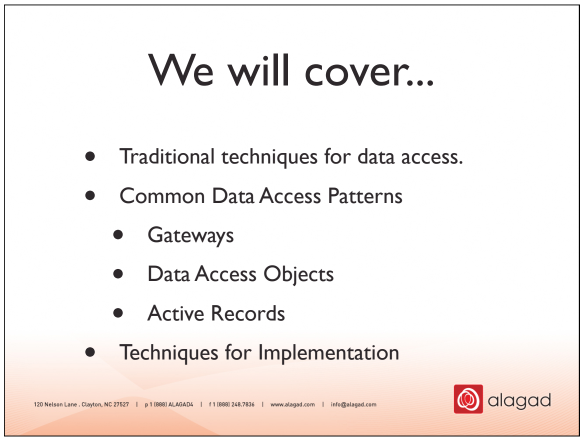


Design Patterns for Data In Coldfusion

Techniques for Success with Data

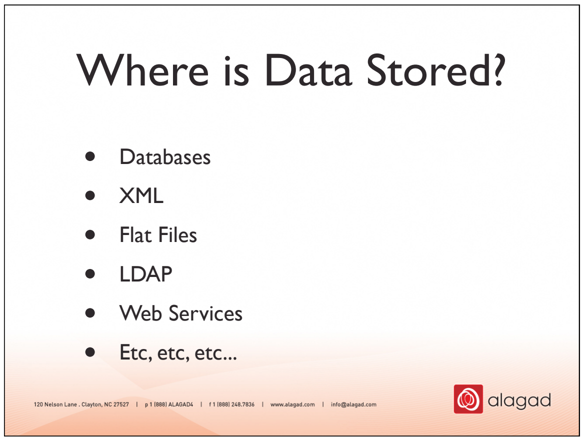
alagad

Doug Hughes, President



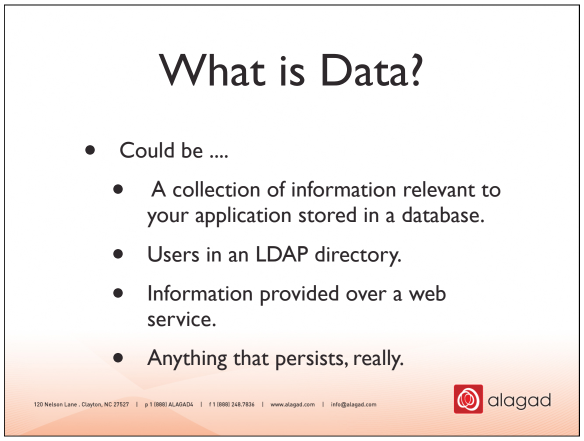
We will cover...

- Traditional techniques for data access.
- Common Data Access Patterns
  - Gateways
  - Data Access Objects
  - Active Records
- Techniques for Implementation



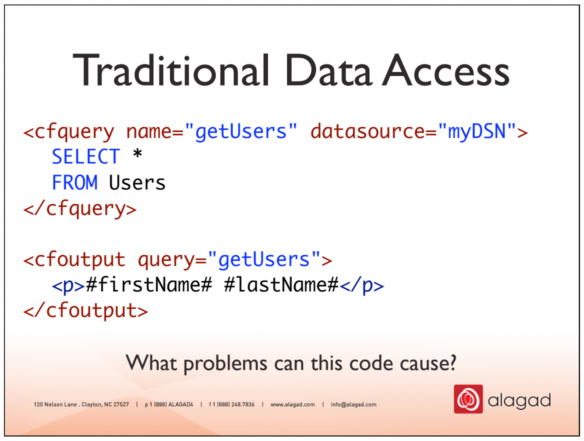
Where is Data Stored?

- Databases
- XML
- Flat Files
- LDAP
- Web Services
- Etc, etc, etc...



What is Data?

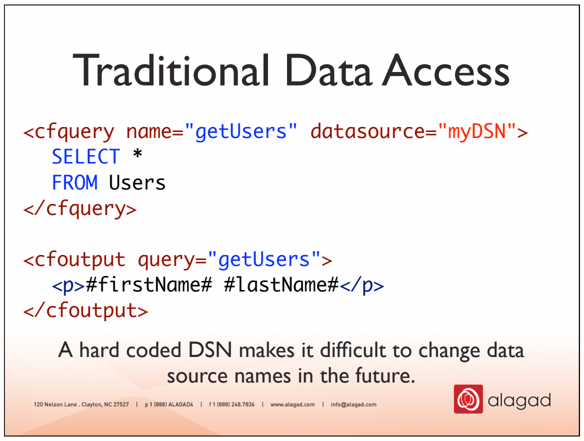
- Could be ....
  - A collection of information relevant to your application stored in a database.
  - Users in an LDAP directory.
  - Information provided over a web service.
  - Anything that persists, really.



Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

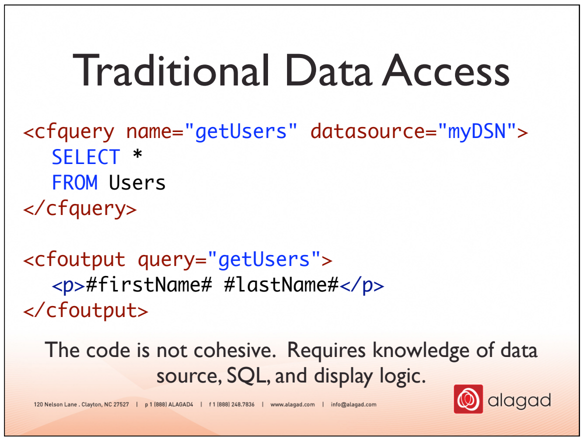
What problems can this code cause?



Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

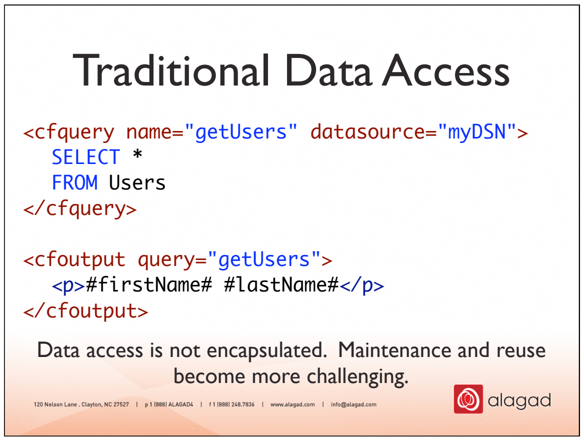
A hard coded DSN makes it difficult to change data source names in the future.



Traditional Data Access

```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

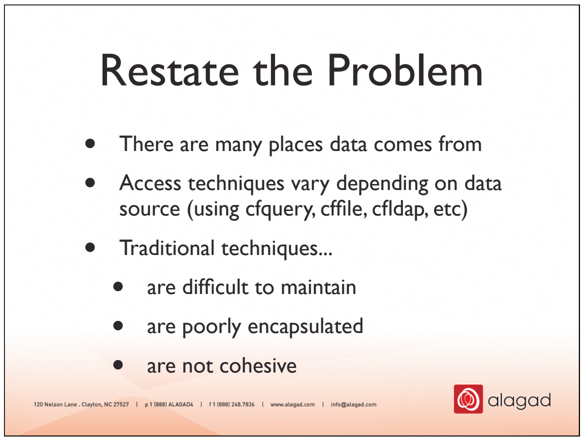
The code is not cohesive. Requires knowledge of data source, SQL, and display logic.



Traditional Data Access

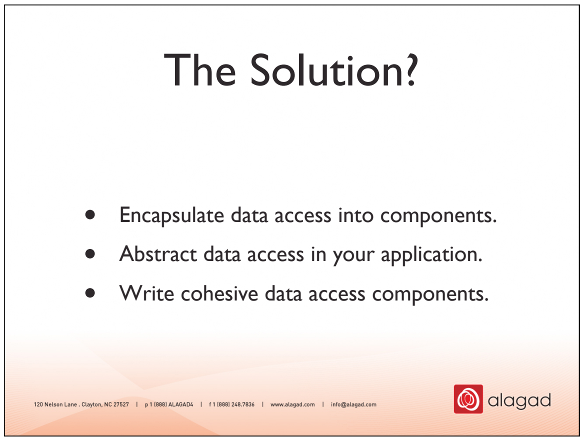
```
<cfquery name="getUsers" datasource="myDSN">
   SELECT *
   FROM Users
</cfquery>

<cfoutput query="getUsers">
   <p>#firstName# #lastName#</p>
</cfoutput>
```

Data access is not encapsulated. Maintenance and reuse become more challenging.



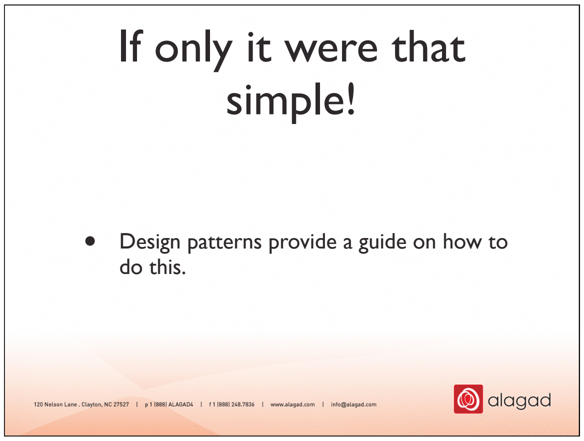
Restate the Problem

- There are many places data comes from
- Access techniques vary depending on data source (using cfquery, cffile, cfldap, etc)
- Traditional techniques...
  - are difficult to maintain
  - are poorly encapsulated
  - are not cohesive



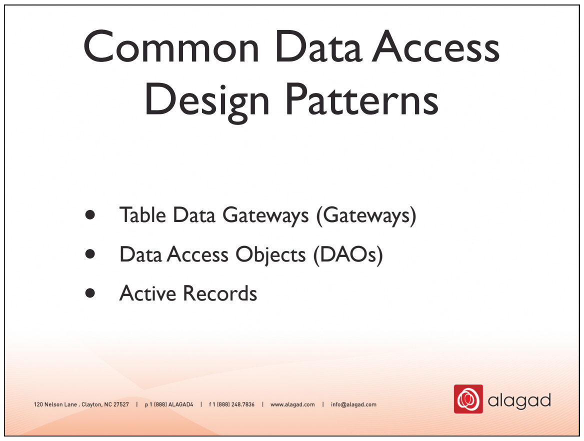
The Solution?

- Encapsulate data access into components.
- Abstract data access in your application.
- Write cohesive data access components.



If only it were that simple!

- Design patterns provide a guide on how to do this.



Common Data Access Design Patterns

- Table Data Gateways (Gateways)
- Data Access Objects (DAOs)
- Active Records

Related Patterns

- Beans
- Transfer Objects
- Factories
- Service Layers
- Presumably, many others.

Table Data Gateways

- Defined by Martin Fowler in Patterns of Enterprise Application Architecture.
- A Gateway holds all the SQL (or whatever) used to access a specific table or view in a database (or whatever).
- Other components use Gateway methods to get data.

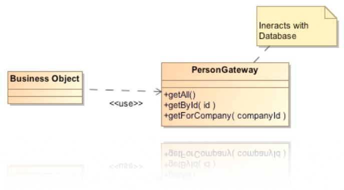


Table Data Gateways

Important Notes on Gateways

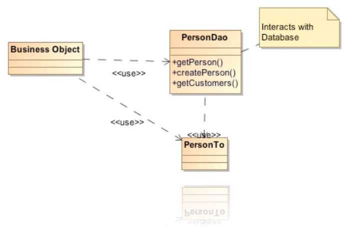
- The pattern doesn't define the API, function arguments or return values.
- The pattern doesn't define how the Gateway is used.
- The pattern doesn't define how the Gateway is configured.
- It's quite simple.

Gateways In ColdFusion

- Traditionally Gateways return ColdFusion queries.
- Could also perform bulk actions such as bulk inserts and bulk deletes.

Data Access Objects

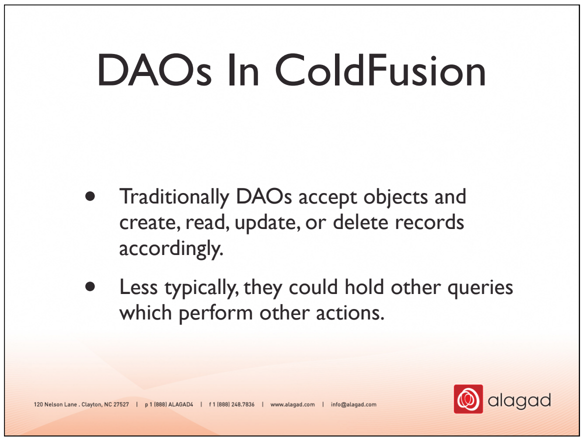
- Defined in Core J2EE Patterns
- A DAO holds all of the SQL (or whatever) used to obtain and store data in a database (or whatever).
- The DAO creates or uses "Transfer Objects" to hold data.
- Other components use Gateway methods to get data.



Data Access Objects

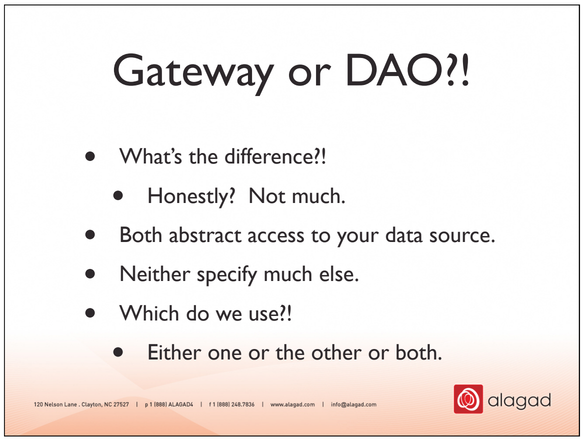
Important Notes on Data Access Objects

- The pattern doesn't define the API, function arguments or return values.
- The pattern doesn't define how the DAO is used.
- The pattern doesn't define how the DAO is configured.
- The pattern requires the use of a Transfer Object... somehow.



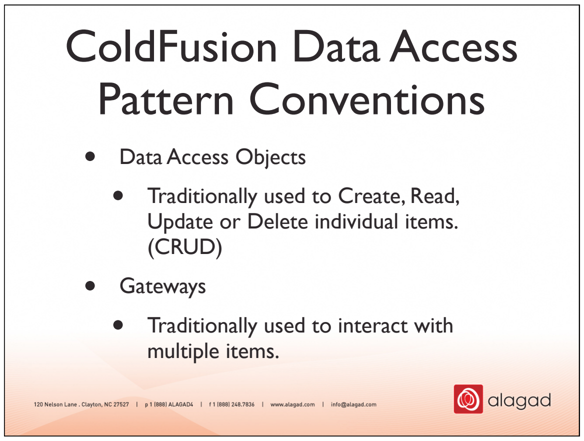
DAOs In ColdFusion

- Traditionally DAOs accept objects and create, read, update, or delete records accordingly.
- Less typically, they could hold other queries which perform other actions.



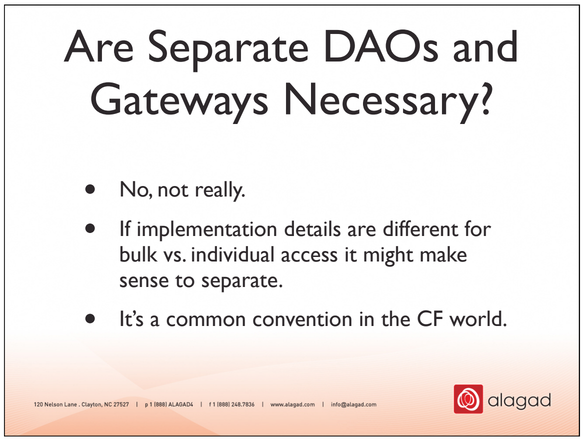
Gateway or DAO?!

- What's the difference?!
  - Honestly? Not much.
- Both abstract access to your data source.
- Neither specify much else.
- Which do we use?!
  - Either one or the other or both.



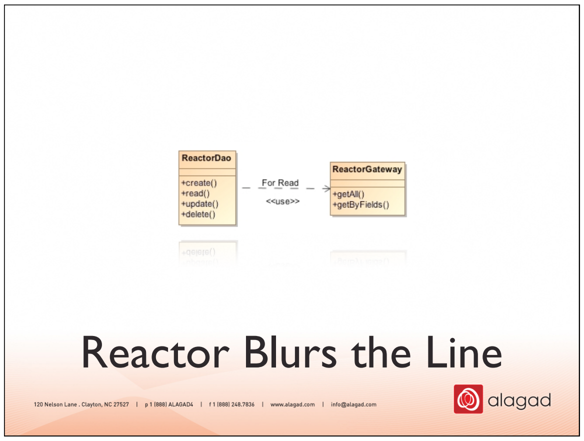
ColdFusion Data Access Pattern Conventions

- Data Access Objects
  - Traditionally used to Create, Read, Update or Delete individual items. (CRUD)
- Gateways
  - Traditionally used to interact with multiple items.

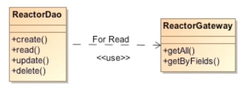


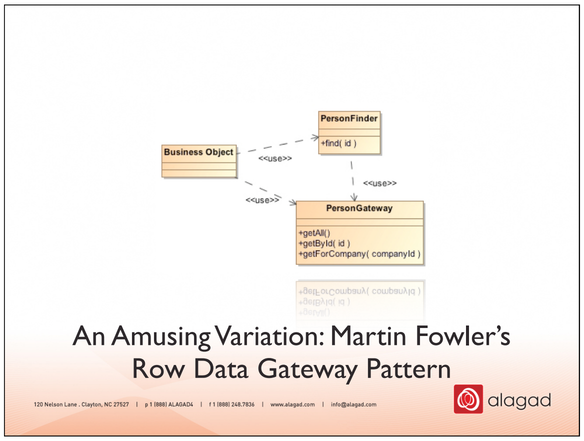
Are Separate DAOs and Gateways Necessary?

- No, not really.
- If implementation details are different for bulk vs. individual access it might make sense to separate.
- It's a common convention in the CF world.

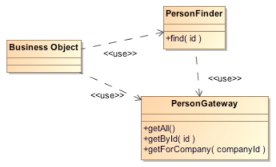


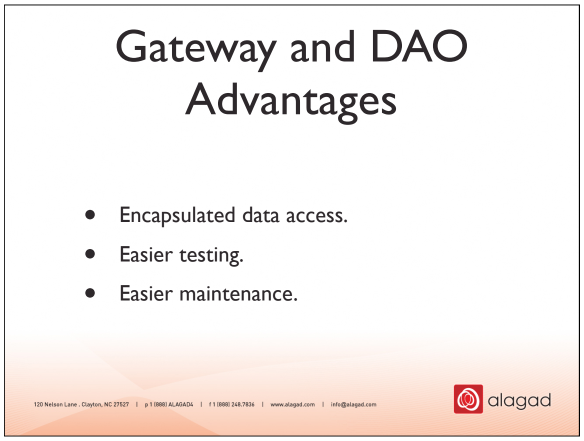
Reactor Blurs the Line





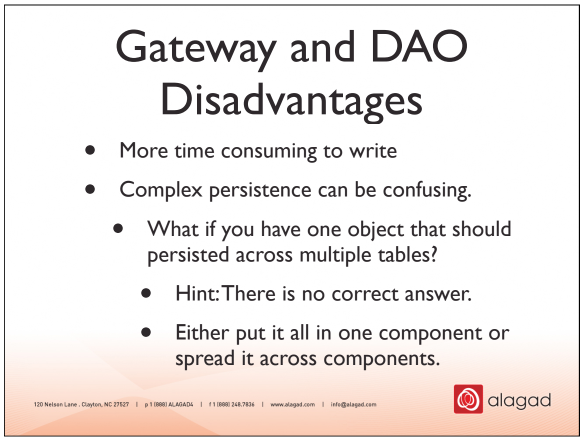
An Amusing Variation: Martin Fowler's Row Data Gateway Pattern





Gateway and DAO Advantages

- Encapsulated data access.
- Easier testing.
- Easier maintenance.

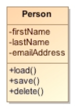


Gateway and DAO Disadvantages

- More time consuming to write
- Complex persistence can be confusing.
  - What if you have one object that should persisted across multiple tables?
    - Hint: There is no correct answer.
    - Either put it all in one component or spread it across components.

Active Record

- Defined by Martin Fowler in Patterns of Enterprise Application Architecture.
- A business object holds all the SQL (or whatever) used to persist itself in a database (or whatever).
- Other components use business object.



Active Record

Important Notes on Active Records

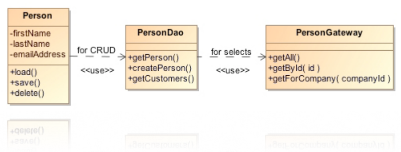
- The pattern doesn't define the API, function arguments or return values.
- The pattern doesn't define how the business object interacts with the database.
- The pattern doesn't define how the business object is configured or used.
- It's extremely simple.

Active Record Advantages

- Encapsulated data access.
- Easy to use
  - Fewer objects to interact with.
- Easy to write
  - Fewer objects to write.

Active Record Disadvantages

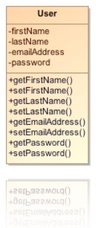
- Business objects are less cohesive.
- Reusing persistence code is potentially difficult.
- Persisting complex object hierarchies is difficult.



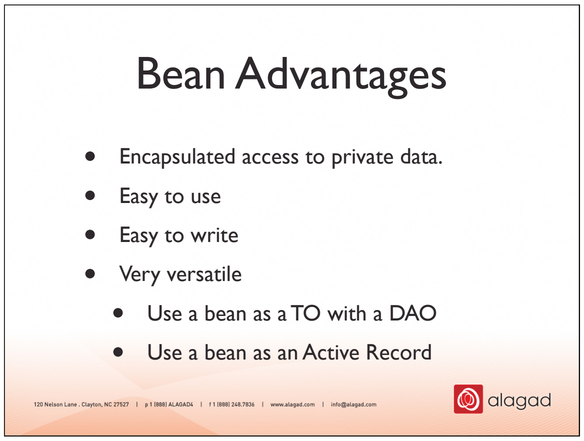
Active Records using DAOs and Gateways.

Beans

- The term "bean" comes from the Java world
  - *Aren't they funny people at Sun?*
- Beans typically keep all their data private
- Control access via getters and setters
- Similar to a structure



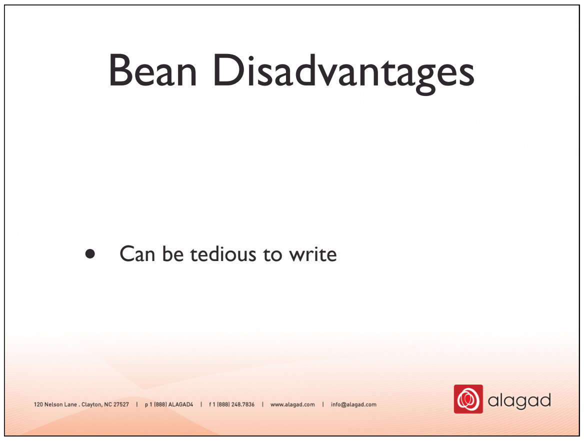
Bean



Bean Advantages

- Encapsulated access to private data.
- Easy to use
- Easy to write
- Very versatile
  - Use a bean as a TO with a DAO
  - Use a bean as an Active Record

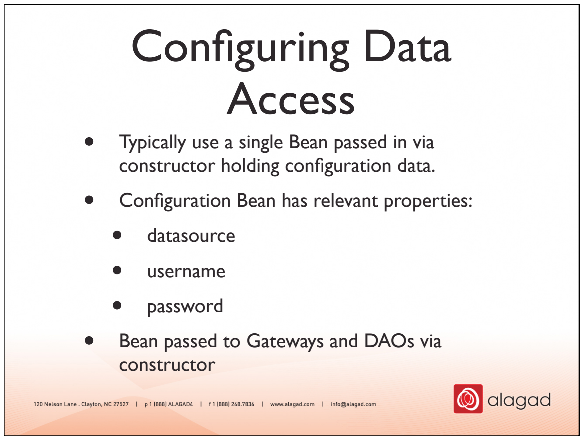
alagad



Bean Disadvantages

- Can be tedious to write

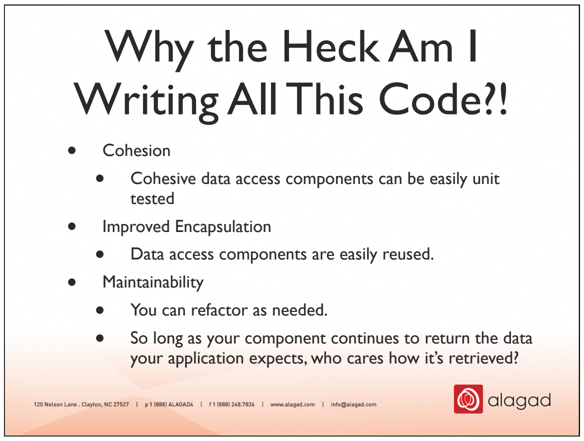
alagad



Configuring Data Access

- Typically use a single Bean passed in via constructor holding configuration data.
- Configuration Bean has relevant properties:
  - datasource
  - username
  - password
- Bean passed to Gateways and DAOs via constructor

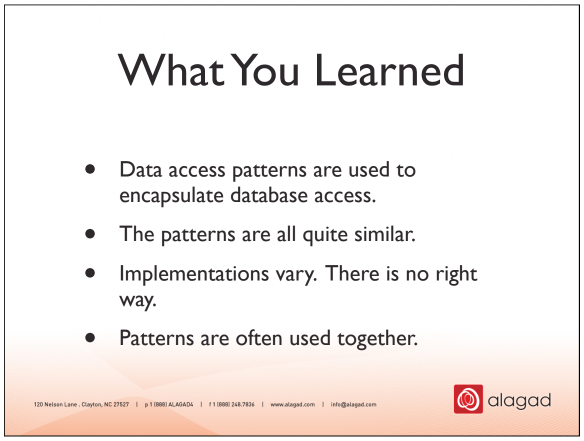
alagad



Why the Heck Am I Writing All This Code?!

- Cohesion
  - Cohesive data access components can be easily unit tested
- Improved Encapsulation
  - Data access components are easily reused.
- Maintainability
  - You can refactor as needed.
  - So long as your component continues to return the data your application expects, who cares how it's retrieved?

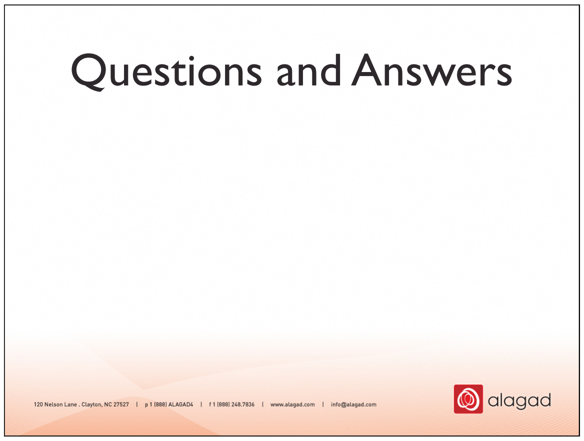
alagad



What You Learned

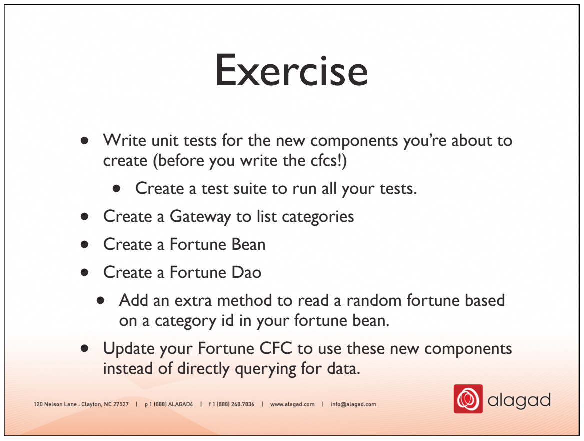
- Data access patterns are used to encapsulate database access.
- The patterns are all quite similar.
- Implementations vary. There is no right way.
- Patterns are often used together.

alagad



Questions and Answers

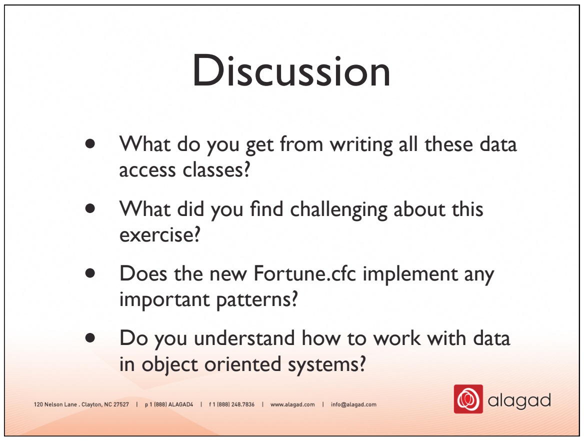
alagad



Exercise

- Write unit tests for the new components you're about to create (before you write the cfcs!)
  - Create a test suite to run all your tests.
- Create a Gateway to list categories
- Create a Fortune Bean
- Create a Fortune Dao
  - Add an extra method to read a random fortune based on a category id in your fortune bean.
- Update your Fortune CFC to use these new components instead of directly querying for data.

alagad



Discussion

- What do you get from writing all these data access classes?
- What did you find challenging about this exercise?
- Does the new Fortune.cfc implement any important patterns?
- Do you understand how to work with data in object oriented systems?

alagad