

Contents

- [Overview](#)
- [Generating Objects](#)
- [Custom Code](#)
- [CRUD](#)
- [List Queries](#)
- [Caching](#)
- [Events](#)
- [Transfer Query Language](#)

Overview

Welcome to Transfer ORM!

When developing an Object Oriented web based application, it is normal to have a database with relational tables and a series of objects that represent that data. Often, the amount of time and effort it takes to manually map these objects back and forth from a database is large, and can be very costly.

Object Relational Mappers (ORM) were developed to cut down the amount of time this process takes, and automate the translation between a relational database and an Object Oriented system.

Transfer ORM's main focus is to automate the repetitive tasks of creating the SQL and custom CFCs that are often required when developing a ColdFusion application. Through a central configuration file Transfer knows how to generate objects, and how to manage them and their relationships back to the database.

So what does all that mean?

Generating Objects

First and foremost, Transfer generates TransferObjects for you according to the *object* elements that are set up in the [Transfer Configuration File](#). The *objectDefinitions* element allow you to map the objects in your application to tables in your relational database.

In addition to simple objects mapped to single tables, you can map composed objects to tables with *OneToMany*, *ManyToOne*, or *ManyToMany* relationships. The composed TransferObjects are generated for you.

For instance, if you have an object definition in your transfer configuration file that looks like this:

```
<object name="Subscriber" table="tblSubscriber">
  <id name="subscriberId" type="UUID" generate="true" />
  <property name="firstname" type="string" />
  ....
</object>
```

Then to create a new instance of a Subscriber, you'd simply call a method on the TransferFactory, passing in the *name* attribute of the object as defined above, like this:

```
newSubscriber = getTransfer().new("Subscriber");
```

Just to make it clear, you would not have to write a Subscriber.cfc, you'd only need to create a simple object configuration as above.

- In these examples, getTransfer() returns an instance of the TransferFactory. The TransferFactory (transfer.TransferFactory), should be instantiated as a singleton in your application at application startup. You can use ColdSpring to do that for you. An example of the configuration is given in the FAQ.

The generated TransferObjects have a wide variety of methods you can use, depending on their configuration, including getters and setters for all properties of the object, and a whole bunch of other useful stuff. See the documentation for details.

Custom Code

And if that ain't enough, and you need to add custom methods to a TransferObject, you can do so in 2 ways. Either you can use the function tag in the transfer configuration file OR you can use a *decorator*, which you can specify in the transfer configuration file like so:

```
<object name="Subscriber" table="tblSubscriber" decorator="path.to.SubscriberDecorator">
```

Any method added to the SubscriberDecorator in this case becomes available to Subscriber. So if for instance you needed a method that returned the full name of a Subscriber, you could add this to your SubscriberDecorator:

```
<cffunction name="getFullName">
  <cfreturn getfirstName() & " " & getlastName() />
</cffunction>
```

And call it like so

```
fullName = Subscriber.getFullName();
```

CRUD

In addition to generating TransferObjects, Transfer *very* conveniently handles create, read, update and delete operations between your TransferObjects and your database for you. Create is shown above already.

To get an existing Subscriber from the database and populate the Subscriber TransferObject:

```
Subscriber = getTransfer().get("Subscriber", subscriberId);
```

To persist a TransferObject, all you need to do is pass your TransferObject to the TransferFactory, like this:

```
getTransfer().save(Subscriber);
```

To delete a Subscriber from the database,

```
getTransfer().delete(Subscriber);
```

List Queries

If that weren't enough, Transfer also generates list queries for you. There are a variety [list methods](#), but as an example, to get a list of Subscribers ordered by date, you'd simply call the list method on the TransferFactory like so:

```
getTransfer().list("Subscriber", "dateSubscribed");
```

Caching

Transfer also provides, *believe it or not*, an extensive, highly configurable [caching system](#) that stores TransferObjects in memory so you don't have to fetch them from the database each time you need them. You can configure the cache to your heart's content within the objectCache section of the transfer configuration file. For example, this setting would cache Subscriber objects for the duration of a session:

```
<cache class="Subscriber">
```

```
<scope type="session"/>
</cache>
```

The cache is automatically synchronized with the database when you save() TransferObjects. Note that by default, Transfer's cache is configured to store objects in memory indefinitely. The cache automatically discards TransferObjects when the underlying Java engine requests memory to be freed, to ensure that the server's memory is not over run, and that the cache doesn't become a memory leak.

There are a group of methods you can use with Transfer's caching system, mainly to discard TransferObjects from the cache or recycle TransferObjects. See the [documentation](#) for details

Events

Transfer also provides ... yes, there is *still more* ... an [event model](#) that can be taken advantage of to notify other CFC's of updates, creates, deletes and new TransferObjects being created.

This can be used to clear query caches when data has been changed. For example, to clear the ColdFusion query cache after an update, I would write this sort of method on my an Observer of the Transfer Events:

```
<cffunction name="actionAfterUpdateTransferEvent" hint="Actions a Delete transfer Event" access="public" returntype="void" output="false">
  <cfargument name="event" hint="The event object" type="transfer.com.events.TransferEvent" required="Yes">
  <cfobjectcache action = "clear">
</cffunction>
```

Now every time an object is updated, the ColdFusion query cache is cleared.

Transfer Query Language

There is also a scripting language that allows you to perform database queries based on the information and naming scheme that you set up in your transfer configuration file called **T**ransfer Query Language ([TQL](#)). TQL is very similar to SQL, however since Transfer already knows about the relationships in your system, you don't have to write as much code to perform complicated queries against your database.

For example, if we wanted to perform a query to list all Posts in my Blog System, with their Author, and all the Categories they belonged to, ordered by the post date we would write this is TQL:

```
<cfsavecontent variable="tql">
  from
    post.Post as Post
      join system.Category
        join user.User
  order by
    Post.dateTime desc
</cfsavecontent>
```

And then we would create a Transfer Query Object by passing the TQL to transfer.createQuery(), and then create the actual query we need by passing the Transfer Query Object to transfer.listByQuery(), as in the example below.

```
<cfscript>
  query = transfer.createQuery(tql);
  qPosts = transfer.listByQuery(query);
</cfscript>
```

Since Transfer already has the relationships between the Post, Category and User, it can intelligently create the SQL joins for you! Saving you even more time!

There's another Database Management Method you can use with TQL besides transfer.listByQuery(), and that's transfer.readByQuery(). readByQuery returns a Transfer Object, whereas listByQuery() returns a query. readByQuery() requires TQL that will return only "one row" - if more than one row is returned, an exception is thrown - and it also requires that you specify the class, like all the other readBy* methods.

Basically, readByQuery() is a way to create a Transfer Object when you don't want to use the primary key, but specify some other condition that will return only one record.

For more information on these Transfer methods, see [Persisting and Retrieving Objects](#).

Categories:

- [Cache](#)
- [Clone](#)
- [Configuration](#)
- [Decorators](#)
- [Events](#)