Building PHP Applications Using the ATK Framework

From Achievo Wiki

 Ivo is co-founder and currently CEO at Egeniq.com, and previously was CTO of Ibuildings, a PHP services company based in Europe, where Ivo was also designer and lead developer for the ATK framework project and the Achievo Project Management suite - created using ATK (both open source). Mr. Jansch is also the author of several books (http://www.amazon.com/gp/search/ref=sr_adv_b/?search-alias=stripbooks&unfiltered=1&fieldkeywords=&field-author=Ivo+Jansch&sort=relevanceexprank&Adv-Srch-Books-Submit.x=22&Adv-Srch-Books-Submit.y=7) .

The text from this article was originally published in the Oracle Technology Network (http://www.oracle.com/technetwork/articles/jansch-atk-085672.html) and published here with Ivo's permission (it has also been adapted for use with a MySQL database). You can also clone a complete project repository from GitHub at https://github.com/dalers/scottapp (commits roughly coorespond to the sections in the tutorial), and which includes an SQL file for creating a MySQL database with the SCOTT schema.

Contents

Introduction

Companies of all sizes are seeking a rapid but structured approach to "Webifying" their existing traditional database-type applications (e.g. Oracle). Increasingly, the combination of an existing database and PHP is an attractive option.

In this article, you will learn the power of this combination by using the open source ATK PHP application framework (http://www.atk-framework.com) to build a Web application on top of an existing database. Using the venerable Oracle *scott* schema as an example, you will see how to build a web application with full CRUD (create, read, update, delete) support for managing employees and departments - in just a few steps and in fewer than 100 lines of code.

About Application Frameworks

An application framework's foremost goal is to ease development of applications. Usually a framework consists of a set of classes that provide common functionality found in applications, such as authentication, authorization, and database connectivity. As an alternative to creating such functionality from scratch, a framework can significantly reduce the amount of time needed to develop an application.

The more standard functionality a framework can offer, the bigger the advantages. The downside is that the bigger the framework, the more it influences the architecture of the application. The ability to customize the framework is therefore essential. Fortunately, most object-oriented frameworks offer this possibility.

Lately, the term *business framework* has been gaining momentum. Whereas a "regular" framework targets application development in general, and as such offers functionality found commonly in applications and Web sites, a business framework is targeted at professional business applications. A business framework should help developers write a business application by letting them focus on the business logic. All application "plumbing," such as how to handle the user interface, how to interact with a database, how to handle sessions, and so on, should be dealt with by the framework (but remain customizable, should the need arise).

Business frameworks usually make use of reusable "business objects," or "entities," that define the data model of the application and contain all the business logic. These business objects form the core of an application. Some frameworks use XML to describe these objects; others, such as ATK, use code. The Oracle Application Development Framework (ADF) for Java is an example of a framework that uses a combination of XML and code.

About ATK

ATK is a business framework that has taken the goal of speeding up development to heart. All parts of the framework are intended to reduce the amount of code an application requires. Instead of providing a large set of utility classes to maximize functionality, it provides a common framework to minimize coding. All new features are carefully considered and are added only if they reduce development time in some area.

The addition of new ATK features is partly a matter of evolutionary development. No development team is responsible for new features; rather, the framework is extended based on the requirements of the applications that use it.

Prerequisites

To be able to work with the examples in this article, note the following:

- You need a working database server (MySQL, Oracle8*i*/9*i*/10*g*, PostgreSQL 7.1+, or Microsoft SQL Server).
- The scott/tiger example schema must be installed.
- You need a web server with the PHP scripting language that can connect to the database server.
- You need a Web browser.

Some knowledge of PHP will be helpful, but is not essential for understanding the examples in this article. In php.ini, make sure the setting for **error_reporting** is set to "E_ALL~E_NOTICE" (ATK contains a few notices that might show up; this is the default setting on most Unix PHP installations, but on Windows the default setting may be E_ALL).

Getting Started

The first thing to do when starting an ATK application is to download the library. The latest version can always be found on the ATK framework website (http://www.atk-framework.com) . ATK comes in two flavors: just the library, and the library bundled with a demo application. The latter is easier to set up. The plain library is particularly useful when upgrading to a newer version. In this example, you start with the demo application.

Download and Install

In this step, you will download and unpack the files in a place where the Web server can access them. Go to http://www.atk-framework.com/download, and under the "ATK ‘getting started' (library plus demo application)" heading, click either the .tar.gz file (for Linux/UNIX) or the .zip file (for Windows).

Download the file to your Web server's document root. On Unix systems this may be something like /var/www/html/ (for RHEL or CentOS GNU Linux), /usr/local/apache/htdocs (for other GNU Linux), or /usr/local/www/apache22/data/ (for FreeBSD), and on Microsoft Windows with IIS it might be C:\inetpub \wwwroot\. Consult your Web server documentation for the exact location if necessary.

Unzip/untar the file you downloaded. On a Linux command line, this would go like this:

```
[root@nikita /]$ cd /var/www/html
[root@nikita html]$ tar xzvf atkdemo-6.5.0.tar.gz
```

On Windows you would use a tool such as WinZip to unzip the .zip file.

By default, the directory is the same as the archive, atkdemo-6.5.0. In this case, for the sake of brevity, you want to call our application scottapp, so you rename the directory:

```
[root@nikita html]$ mv atkdemo-5.3.0 scottapp
```

On Windows you can do that via Windows Explorer.

The Web server needs only read-only access to most of the files and directories that are unzipped, but there's a directory called atktmp (where ATK stores its temporary files) that it also needs to write to. On Linux you can use the setup.sh script to change the permissions; it takes the name of your Web server user as the parameter (e.g. if your web server runs as the "apache" user, use apache as the parameter).

The snippet below assumes that the parameter is called www, which is common for many GNU Linux distributions.

```
[root@nikita html]$ cd scottapp
[root@nikita scottapp]$ ./setup.sh www
```

Configuration

In the scottapp directory you just created, you can find several files and directories. An important file is config.inc.php. It contains important configuration directives such as the database you are going to connect to.

Open the config.inc.php file in a text editor.

Find the configuration item called $config_db["default"]["driver"]. This is set to mysqli by default for connecting to a MySQL database, but ATK also includes drivers for Oracle 8.0.5/8*i*/9*i*/10*g*, PostgreSQL and Microsoft SQL Server (see config.inc.php for driver names).

Next, find the following directives:

```
$config_db["default"]["host"]
$config_db["default"]["db"]
$config_db["default"]["user"]
$config_db["default"]["password"]
```

You connect to a local server by using "localhost" for the host, and to a remote server by using the server's hostname or IP address.

MySQL Database

Suppose the database is running on IP 10.0.1.10 and the database name is "scott". In this case, your settings should look like this:

```
$config_db["default"]["mysqli"]
$config_db["default"]["host"] = "10.0.1.10";
$config_db["default"]["db"] = "scott";
$config_db["default"]["user"] = "scott";
$config_db["default"]["password"] = "tiger";
```

Oracle Database

If you are connecting to an Oracle database, specify driver "oci8" for an Oracle8*i* database or "oci9" for an Oracle9*i* or Oracle10*g* database, and the $config_db["default"]["db"] setting should be set to the service name of the database. Suppose the database is running on IP 10.0.1.10 and the service name is COMPANYDB. Your settings should look like this:

```
$config_db["default"]["oci9"]
$config_db["default"]["host"] = "10.0.1.10";
$config_db["default"]["db"] = "COMPANYDB";
$config_db["default"]["user"] = "scott";
$config_db["default"]["password"] = "tiger";
```

You can also connect to an entry from your tnsnames.ora file. Say there's an entry in the tnsnames.ora file called COMPANYDB that points to the database where our SCOTT schema lives. In this case, set $config_db["default"]["db"] to COMPANYDB. The $config_db["default"]["host"] is left blank as it will be determined by the tnsnames.ora file. Once configured, the configuration directives should resemble this:

```
$config_db["default"]["host"] = "";
$config_db["default"]["db"] = "COMPANYDB";
$config_db["default"]["user"] = "scott";
$config_db["default"]["password"] = "tiger";
```

Testing the Installation

Now you can test whether the application was properly set up. Access your web server with the appropriate URL for where you unpacked the application. For example, http://localhost/scottapp/index.php

If all went well, you will get a login prompt. Log in, using the default username/password combination administrator/demo (you can change the administrator password in the config.inc.php file).

You will see the ATK demo application menu, but accessing menu items will result in errors because you did not configure the database for the demo application.

Managing Departments

Let's start implementing the code to manage the departments in the dept table of the SCOTT schema. An ATK application is created using one or more modules. The demo application contains a number of modules for the demo lessons, which you will disable for now.

Creating a Module

Go to the modules subdirectory, and create a new directory there called scott. This will be your scott module, containing the functionality for managing departments and employees.

Next, edit the config.modules.inc file and remove the six default modules (or comment them by putting // in front of them), and add one line for the scott module:

```
module("scott");
```

In the modules/scott directory, create a file called module.inc, that indicates to the framework that this is a module containing functionality. Put the following code into the module.inc file:

```
<?php

  class mod_scott extends atkModule
  {
```

```
    function getMenuItems()
    {
    }
  }
?>
```

The **getMenuItems** method is called by the framework to determine which entries the application's menu should contain. You will implement this function later on.

The dept Node

A module contains one or more "nodes". A node, a representation of a database table, is a business object that defines the contents and behavior of a piece of information. Because this article shows you a way to manage employees and departments, you will be creating two nodes. Let's start with the easier one, the one for departments.

1. In the modules/scott directory, create a file called class.dept.inc, with the following contents:

```
<?php

  class dept extends atkNode
  {
    function dept()
    {
      $this->atkNode("dept");
      $this->add(new atkAttribute("deptno", AF_PRIMARY));
      $this->add(new atkAttribute("dname", AF_SEARCHABLE));
      $this->add(new atkAttribute("loc", AF_SEARCHABLE));

      $this->setOrder("dname");
      $this->setTable("dept");
    }

    function descriptor_def()
    {
      return "[dname] ([loc])";
    }
  }
?>
```

This essentially describes what the dept table looks like. There are three fields (called "attributes" in ATK jargon), one of which is the primary key.

The **dname** (department name) and **loc** (location) fields get the AF_SEARCHABLE flag, meaning that when managing departments, the user can search existing departments by using these fields.

The **setOrder** statement sets the default sort order for the administration screens, and the setTable statement tells the system which database table to use.

The **descriptor_def** method tells the system how a record can be represented to the user. In this case, you tell it that a department is represented by its name, with the location between brackets.

2. You might think that now you would have to write the code for adding new departments and editing or deleting existing ones. Actually, you won't have to. This definition is all ATK needs in order to generate an application for you. All you need to do now is add department management to the main menu. To do this, edit the modules/scott/module.inc file again and enhance the empty getMenuItems method with the third line below:

```
function getMenuItems()
{
  $this->menuitem("dept", dispatch_url("scott.dept", "admin"));
}
```

This basically tells the system to add a menu item called dept that points to the admin functionality of the dept node from the scott module.

Now browse the application. Log in, and when you click the menu dept link (you will change this text later on), you should see the following screen:

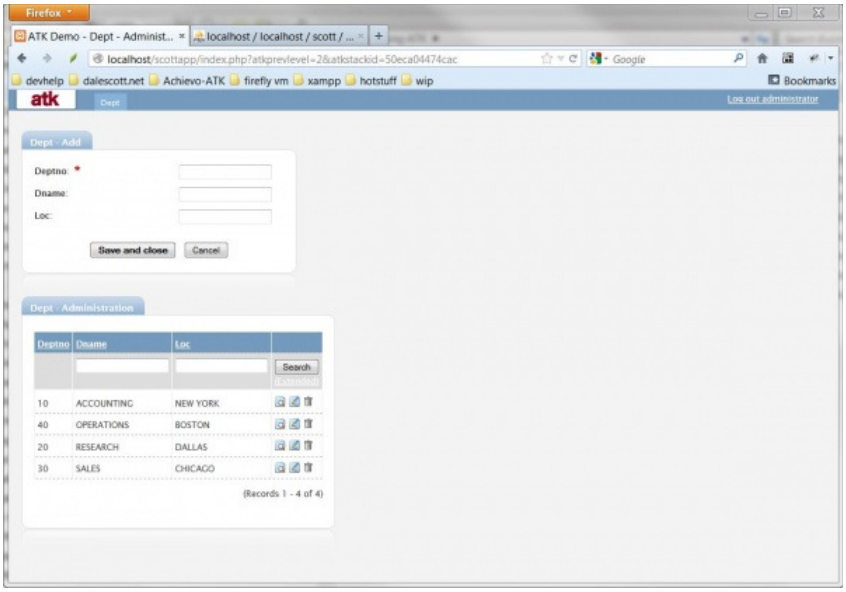

Figure 1 Managing Departments

The upper third of the main screen is a form for adding a new department. You can see that the Deptno field is required (it's the primary key). Note that the length of the input fields is adjusted to the size of the field in the database. (It uses database data dictionary to determine this.) The lower two thirds contains all existing records.

Note how it's sorted the way you wanted it, and note the effect of the AF_SEARCHABLE flags. The user can search through existing records or use the "Extended" search link to perform more-complex (and/or, substrings/wildcards) searches. Finally, the user can click one of the three icons next to each record to view, edit, or delete a record.

Note that when you edit the record with deptno 10, the title bar displays ACCOUNTING (NEW YORK). This is courtesy of the descriptor_def method you added in step 1.

Thus far, you have written only about 10 lines of code!

Managing Employees

Similar to what you did for departments, you're going to add functionality for managing employees. However, this involves some other features to keep things interesting.

The emp node

1. In the modules/scott directory, create a file called class.emp.inc with the following contents:

```
<?php

  useattrib("atkdateattribute");
  useattrib("atkcurrencyattribute");
  useattrib("atknumberattribute");
  useattrib("atklistattribute");

  class emp extends atkNode
  {
    function emp()
    {
      $this->atkNode("emp");
      $this->add(new atkAttribute("empno", AF_PRIMARY));
      $this->add(new atkAttribute("ename", AF_SEARCHABLE));
      $this->add(new atkListAttribute("job", array("CLERK", "SALESMAN",
                          "ANALYST", "MANAGER", "PRESIDENT"), array(),
                          AF_SEARCHABLE));
      $this->add(new atkDateAttribute("hiredate"));
      $this->add(new atkCurrencyAttribute("sal", AF_HIDE_LIST, 7, "USD"));
      $this->add(new atkNumberAttribute("comm", AF_HIDE_LIST, 7, 2));
      $this->setOrder("ename");
      $this->setIndex("ename");
      $this->setTable("emp");
    }

    function descriptor_def()
    {
      return "[ename] ([job])";
    }
  }
?>
```

The explanation of this code follows shortly. Let's first get the example in working condition.

2. Edit modules/scott/module.inc again, and use the code in the fourth line below to add a menu item for managing the employees:

```
function getMenuItems()
{
  $this->menuitem("dept", dispatch_url("scott.dept", "admin"));
  $this->menuitem("emp", dispatch_url("scott.emp", "admin"));
}
```

Now log in to the application again. You should now see a new menu item called menu emp. When you click it, you will see the following screen:

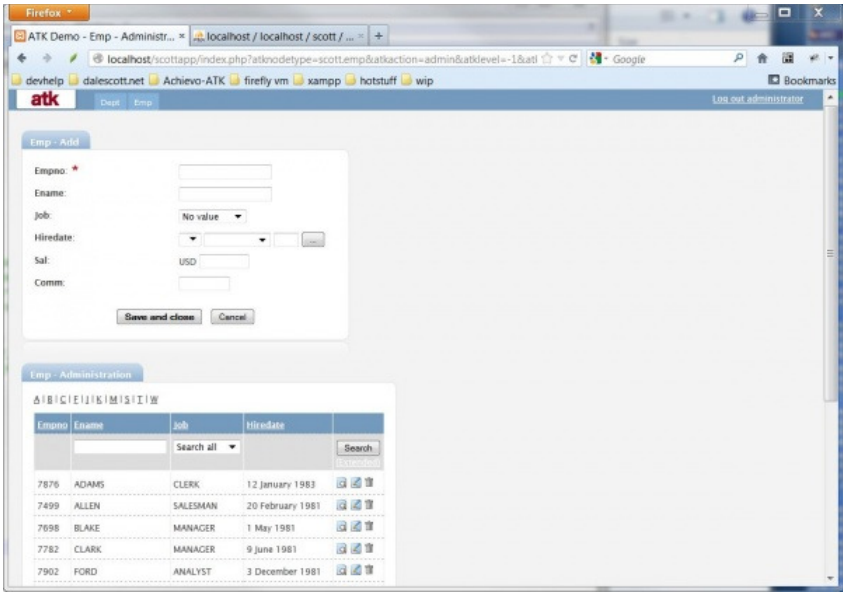

Figure 2 Managing Employees

Let's have a look at the code you added and see how it works.

Note that for several fields, you have used different types of attributes:

```
$this->add(new atkListAttribute("job", array("CLERK", "SALESMAN",
                    "ANALYST", "MANAGER", "PRESIDENT"), array(),
                    AF_SEARCHABLE));
$this->add(new atkDateAttribute("hiredate"));
$this->add(new atkCurrencyAttribute("sal", AF_HIDE_LIST, 7, "USD"));
$this->add(new atkNumberAttribute("comm", AF_HIDE_LIST, 7, 2));
```

Here you are telling the framework that for the Job field, you want the user to pick from a list of predefined options. As in the dept example, you use the AF_SEARCHABLE flag for fields you want to be able to search on.

For the Hiredate field, you use an atkDateAttribute. This is represented on-screen as a date picker, with a calendar pop-up (the "…" button). For the Sal (salary) of the employee, you use a currency attribute, with USD as the currency symbol. The AF_HIDE_LIST flag is introduced here. If you have a look at the application, you'll see that this flag causes these fields not to appear directly in the list of records. Finally, for the commission percentage (Comm), you use a number attribute.

When you play around with the application, you will notice that these attributes not only look different but also provide their own validation. If you try to enter non-numeric characters in the Comm field, for example, the application will tell you that this is invalid as soon as you try to save the value.

You may have noticed the useattrib calls at the start of the file. Much like includes, these tell the system that you are going to use certain attributes. For performance reasons, not all attributes are always loaded and you have to tell the system when you'll be using them (although there is an autoload mechanism, it is still good practice to declare attributes).

Another line of code that is different from the dept example is:

```
$this->setIndex("ename");
```

which creates an alphabetical index on top of the list of records, as you can see in the screen shot. Using the Ename field in this case, the user can page rapidly through the list of records.

With less than 30 lines of code, you can now manage employees and departments. You should be getting an idea about the power of combining an existing database with PHP and ATK.

Building Relations

There is one feature I want to show you in this brief introduction to ATK - the building block for creating complex applications with intricate data models.

Usually, an application does not consist of separate tables. There are almost always relations within the data. This is also the case in the SCOTT schema. In fact, the emp and dept tables demonstrate three types of relations. Here's how to implement these relations in your Web application.

Employee -> Department (N:1 relation)

First, there's a relation between an employee and that person's department. This is a many-to-one relation, because many employees can work in the same department but an employee cannot work in two departments at the same time (at least not in scott's world).

On the database level, this relation is represented by the Deptno field in the emp table. This field points to one of the departments in the dept table.

Let's have a look at how you implement this on the application level. Open the modules/scott/class.emp.inc file, and near the top, where the useattrib calls are, add the following line:

```
userelation("atkmanytoonerelation");
```

This tells the framework to include a relation, similar to what you did with the attributes.

In the body of the emp class, add the second line of code below the Comm field:

```
$this->add(new atkNumberAttribute("comm", AF_HIDE_LIST, 7, 2));
$this->add(new atkManyToOneRelation("deptno", "scott.dept", AF_SEARCHABLE));
```

This adds an atkManyToOneRelation to the node. The first parameter indicates that you use the Deptno field from the database, the second parameter tells ATK that the relation points to the dept node of the scott module. The AF_SEARCHABLE flag is used to allow the user to search employees by department.

This is all there is to it. If you now look at the application, you'll see what adding the relation does to the function of the application:

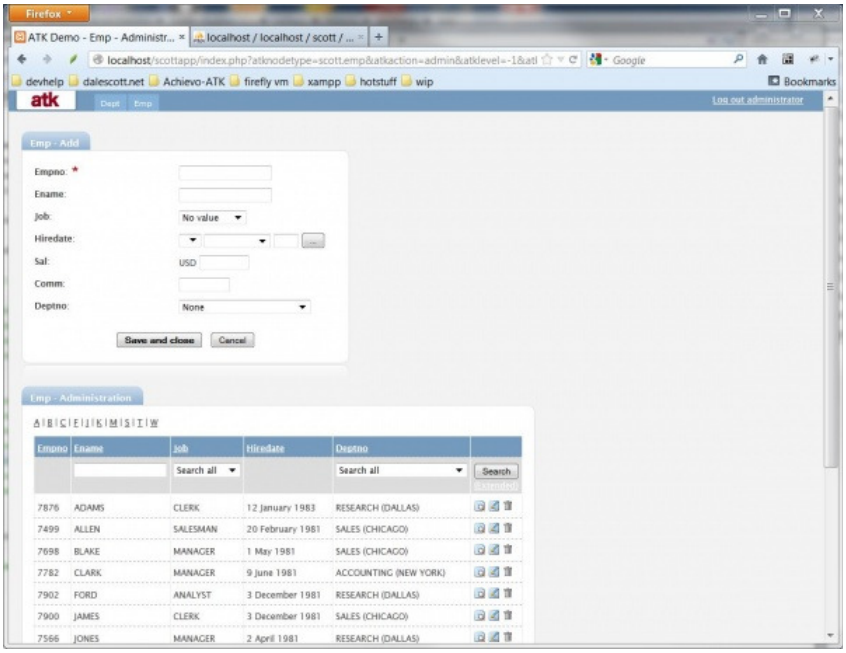

Figure 3 Employee-Department Relation

In the top part, you see that when adding an employee, you can now select the department the employee works in. Note how the descriptor_def method you've seen before is used by the framework to determine what a department looks like to the user. The bottom part shows that for each employee, you can see that person's department and that the user can search the employees based on the department.

Note how only two lines of additional code accomplished this, telling the framework only which node the relation points to and which field is used to store the current value. It figures out the rest by itself.

Employee -> Manager (N:1 relation)

The SCOTT schema features another N:1 relation, the relation between an employee and their manager. This is, in fact, the same kind of relation as the

employee/department relation, only this time, the relation is circular, pointing to the same table. Implementing this relation in the application is a matter of adding the following line of code to class.emp.inc:

```
$this->add(new atkManyToOneRelation("mgr", "scott.emp", AF_SEARCHABLE));
```

This links the Mgr field of the emp table to the emp node from the scott module. Browse the application to see the effect.

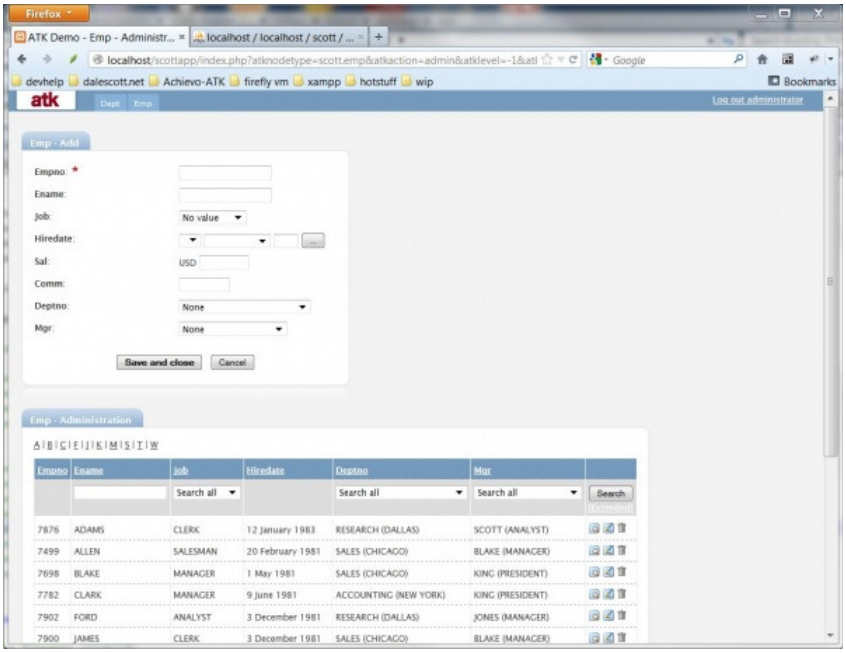

Figure 4 Employee-Manager Relation

Department -> Employees (1:N relation)

A different kind of relation is the emp/dept relation in reverse. If on the department side, you want to view and edit the employees in that department, you have a relation in the opposite direction. It is a 1:N relation, in that one department can have multiple employees. In the database, the relation is again represented by the Deptno field of the emp table. To implement the relation on the dept side, take the following steps:

Add the following line right above the dept class:

```
userelation("atkonetomanyrelation");
```

This is what you did in the emp class before, only this time you use atkonetomanyrelation.

Add the following code below the loc attribute:

```
$this->add(new atkAttribute("loc", AF_SEARCHABLE));
$this->add(new atkOneToManyRelation("emps", "scott.emp", "deptno",
                                    AF_HIDE_LIST));
```

This adds the relation to the node. The first parameter gives this relation a unique name within this node. Because in the dept table, there is no field to represent this relation, unlike the other attributes, emps is an arbitrary name. You could use anything here, as long as it is unique within the dept node. The second parameter is more important. Here you tell the framework what the relation points to. The third parameter is necessary to tell the framework that in the emp table, the Deptno field is used to point back to the department you are working on.

This example demonstrates that again with very few lines of code, relationships can be implemented. When you now use the application and click the edit icon for a department, you will see the following screen:

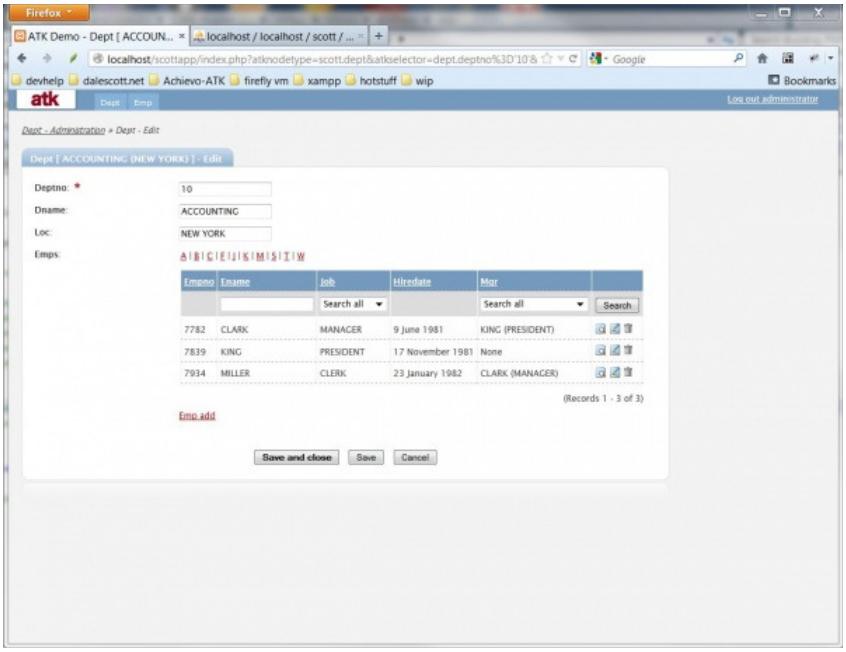

Figure 5 Department-Employee Relation

While altering the Accounting application, you can now see its employees. A very important thing to note is how the functionality of the emp node is reused here. The fields you made searchable are now searchable within a department too, without any additional coding effort.

The important thing to realize here is how this approach eases maintenance of the application. If a field gets added to the emp table, the only file you have to modify is the emp node in the class.emp.inc file. All other parts of the application that have relations with this node will automatically pick up the change.

Improving User-Friendliness

Although the framework does many things automatically, customizations are possible. One example is the use of translations. You will have noticed that by default, field labels and column headers take the name of the database field and that menu entries are prefixed by default with "menu." This behavior can easily be altered by used of language files.

Language Files

In the modules/scott directory, create a new subdirectory called languages.

In the languages subdirectory, create a file called en.lng, with the following contents:

```
<?php

  $en = array('menu_dept' => 'Manage departments',
              'menu_emp' => 'Manage employees',
              'scott_dept'=> 'Department',
              'scott_emp' => 'Employee',
              'deptno' => 'Department number',
              'dname' => 'Name',
              'loc' => 'Location',
              'ename' => 'Name',
              'emp_mgr' => 'Manager'
              );
?>
```

If you now browse the application, you'll see that certain fields are translated. You can add as many language files as you want; "en" stands for English; you can, for example, add a "de.lng," with German translations. The language used by the application is configured in the config.inc.php file.

Note how the Mgr field is prefixed with emp_. This is not necessary, but it's an example of how to add node-specific translations. The term ename is always translated to Name, regardless of which node it's used in, but mgr is translated to Manager only in the emp node.

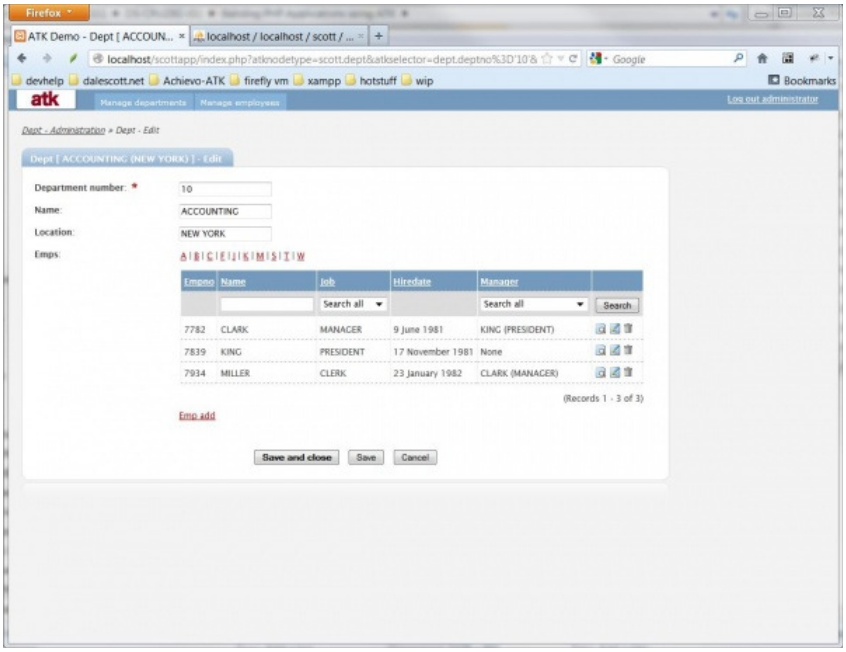

Figure 6 Language Translations

Conclusion

I hope that the examples in this article have demonstrated the ease with which PHP can help you make web applications on top of existing enterprise databases by using a framework. Approximately 30 lines of code are enough to let users manage departments and their employees.

With reusable building blocks (the dept and emp nodes), building an application of greater complexity does not mean more-complex code. As long as you have a good data model (tip: draw entity relationship diagrams of your database, and model the ATK nodes after that), creating a Web application on it is fairly straightforward.

Next, you will probably want to add more features, such as use authentication/authorization, or customize the user interface. Here are a few places to get more information:

http://www.atk-framework.com	The Web site for the ATK project. You can download the framework from here as well as view the online documentation and take part in the forums.
http://www.atkframework.com /documentation	The documentation section. Here you can find the API documentation as well as several useful how-tos.
http://forum.achievo.org	The members of the ATK community use this forum to communicate and help each other.
http://www.achievo.org/blog	My blog, where I occasionally discuss ATK and other PHP-related subjects.

There is much more to tell than there is room here to explain, so I encourage you to now try out the demo lessons to see more examples of what ATK can do for you. It will probably be simplest to unpack the atkdemo archive again (downloaded earlier), but this time don't change the directory name. You will also have to create and configure the demo database - see doc/INSTALL for more information. The demo application contains eight lessons, covering topics from creating a basic application to more advanced techniques such as using database metadata to create nodes and generating OpenOffice documents directly from nodes. You can even review the code from within the demo application itself.