

# Using LDAP as an Oracle Professional, Part I : The Snowball Effect

It often seems that new Oracle releases include features you're not likely to use in the near future, like AQ, Oracle Wallet, iFS, analytics, etc. You make a mental note and move on. One new feature, LDAP integration, should not be dismissed so quickly.

Unless you've been comatose for the last five years, you've heard of LDAP already. Most documents that describe its meteoric rise use words like "mushroom," "explode," and "snowball." The snowball effect is the best metaphor for LDAP's brief history. It also sums up how it will catch on in your company once given life in at least one application. All it takes is a gentle nudge. Beyond that, due to its inherent qualities and usefulness, it will quickly grow (in a good way), almost as if under its own momentum.

Oracle 9i is the terminal release of Oracle Names, and the preferred method for Oracle naming is now in an LDAP directory. Oracle is currently migrating many of their applications to begin using an LDAP directory server for security and application metadata purposes. Part of this push is seen in their new C and PL/SQL LDAP APIs. Even Microsoft, the "ole grim reaper" of the computing world's standards, has joined in. They wrote the core of Windows 2000 around an LDAP v3 server.

LDAP isn't over-hyped vaporware or another ignored effort from the IETF or W3C. It's a simple, widely accepted, useful piece of your technology pie. Much like ASCII, HTML, TCP/IP and XML, it's here to stay. Time to sink your teeth in!

We will look at how LDAP began, its best qualities, basic terminology, LDAP URLs, searches and filters, as well as where Oracle is headed with LDAP. A case study is presented where the author converted a custom, table-based, web security framework to one using an LDAP directory server. There is a great library of resources on the 'net and in Oracle documentation if you need to dig into more advanced LDAP concepts. See the partial list at the end of the article.

## How Did LDAP Get Started?

Most IT environs in the mid-nineties were a heterogeneous mess: proprietary this, distributed that, out-of-sync and closed-off everything. Client-server computing was the real catalyst to this horror, where every application had become an island unto itself, often referred to as a "silo." This created real problems when a user or application needed up-to-date information about human or physical resources, or if one of those resources needed to be updated or removed. The data was scattered in many systems and spreadsheets. Instead of getting your data from the main pot, it was like the pot had exploded; you now had to scurry around, looking for the juicy chunks if you wanted any lunch.

Rather than deal with the mess, IT shops usually wound up creating yet another independent silo of critical data, or outsourcing another million-dollar contract to tie some systems together. Within a short time, users were being overwhelmed by passwords and how to access and use scores of mainframe and PC systems. IT managers were also being overwhelmed supporting so many disparate applications and their frustrated users. Then network computing burst onto the scene! The problems of distributed data and computing multiplied ten-fold as all these silos were wired together. Everything intensified. In an age blessed with technology, it still took forever to get anything done. For example, from 1995 to 1998, the world's largest long distance provider had managed to create 38 separate, and largely unique, web application infrastructures (my team was hired to consolidate them into one). Things were just plain nuts.

Standardization and some centralization were needed, where critical data about departments, customers, employees, credentials, access controls, offices, etc. was kept in one place. Since it would be used by different applications and potentially thousands of internal and external users, the solution needed to be

flexible, robust, scalable, secure and fast. It also had to handle internationalization, delegated administration and replication out of the box (to accommodate language, time-zone and latency issues).

Enter X.500 and the directory servers based upon it. Unfortunately, X.500 was a decision-by-committee standard. It was bloated, resource intensive and complex. Its method of accessing the directory server was called the Directory Access Protocol (DAP), which required the 7-layer OSI stack and more computing power than early 90's PCs had. Directory server technology was the answer, but X.500 and DAP, were not. So a University of Michigan team developed a lightweight version of DAP that could operate over the TCP/IP protocol. Originally called DIXIE (Directory Interface to X.500 Implemented Efficiently), it was transformed into an IETF standard by Tim Howes, Steve Kille and Wengyik Yeong in 1995. LDAP was born. You can read Tim's story at [http://home.netscape.com/columns/techvision/innovators\\_th.html](http://home.netscape.com/columns/techvision/innovators_th.html).

Much like Apache and Linux, LDAP was provided to the community free of charge. LDAP had many early adopters from 1995 and 1996 and not just for pilot projects either. Netscape was a big contributor as well, hiring Mr. Howes in 1996 and releasing their C and Java LDAP APIs to the public. Clayton Donley wrote PerlLDAP a short time later. Finally, any application or system that could speak TCP/IP, or make use of the free APIs, could be directory-enabled and realize the huge simplicity, cost, and peace-of-mind benefits that directories offered.

In two short years, LDAP gained amazing footholds, solving problems and saving a lot of pain and cash at even the biggest corporations in the world (Ford, Citibank and Home Depot, to name a few). On the server side, Netscape dominated mindshare, followed by Novell, IBM, lesser-knowns, Oracle and open source offerings. Even Microsoft joined the fray, basing the core of NT 5 (now Windows 2000) on an LDAP directory server. Things were no less busy on the client side. In fact, Netscape, IE, Eudora, Exchange and Lotus Notes have had LDAP features for some time. Nowadays, vendors of all classes of software have announced support for LDAP, or plans for such.

## Where Is Oracle Going With LDAP?

*Note: Some of this section comes from 8i and 9i documentation. Some I took from a PR-filtered reply from the directory team at Oracle.*

From our vantage point as the customer, Oracle's first foray into LDAP showed up as an optional naming method in 8.1.6. Instead of storing the connection descriptors in your local tnsnames.ora file, or your Oracle Names server, you could store them in a directory server, namely, NDS, AD or OID.

Starting in version 8.1.7, Oracle provided an API to work with data in a directory server. The API came in both C and PL/SQL flavors. Unfortunately, the PL/SQL version, a supplied package named DBMS\_LDAP, only had a subset of the C API, and there were a number of minor bugs associated with it

With 9i, NDS is no longer supported for centralized naming, the PL/SQL API is fairly complete, and with 9.0.1.2, the private memory leak in DBMS\_LDAP is fixed. Along with the API, Oracle provides some sample programs, a developer's guide, and the usual LDAP command line tools to form the "OID SDK release 3.0.1." If you need to access OID or another directory server from Java, you turn to the JNDI classes, which are part of the J2SE and J2EE from Sun.

The 9i docs note that in future versions Oracle Names will no longer be supported as a centralized naming method. Oracle strongly recommends migrating to directory naming instead (using OID). I also noticed that Oracle's support for third-party directories was getting narrower, rather than wider. Anticipating that customers who had investments in Oracle Names or a non-Oracle directory server would be a little upset, I contacted Oracle to show me what's inside their crystal ball.

In short, they're totally committed to the benefits of "centralizing and standardizing" on an LDAP v3 directory server, in this case, OID. Although OID **can** be purchased separately, it is **included** as a

necessary part of 9iAS EE or 9i DBMS EE. In the future, OID will be needed for most applications that Oracle sells. Apparently, great forces are stirring to achieve this.

A directory server is good for storing anything that is updated infrequently and should be shared, not just user, group and network information. Oracle seems to be doing just that, using OID to centralize application configuration and metadata as well. Their PR response read

“...we have now marshaled all of our development teams across Oracle to build and leverage whatever shared application data they need directory storage for into OID...DO expect the number of disparate user silos in the Oracle stack to drop drastically starting with Oracle9i.”

They also mentioned their Directory Integration Platform (DIP), pieces of which are starting to appear now in 9iAS R2. The DIP will allow companies to leverage prior investments in LDAP and non-LDAP directories like Oracle HR, ERP systems, or in our case, the iPlanet Directory Server. In summary, Oracle is totally committed to the LDAP standard and directory technology. OID and the LDAP protocol will become almost as essential to your skill set as SQL. It will become very hard for you to ignore LDAP as an Oracle professional.

## What Does An LDAP Server Do?

*A directory service is the main switchboard of a network operating system. It manages the identities and brokers relationships between distributed resources so they can work together. Further, it is a place to store information about corporate and organizational assets, such as applications, files, printers, and users. It provides a consistent method for naming, describing, locating, accessing, managing, and securing information about the resources. It simplifies administration, strengthens security and enables interoperability. – Microsoft AD Overview*

Well-said. However, Microsoft's view is NOS-centric. A directory service can also be the main switchboard for B2B systems and enterprise application integration, providing the same administration, security and interoperability benefits to these widening, complex networks.

Some of the first applications you may see, or help develop, are those that a directory server “does best.” These would be Single-Sign On frameworks and HR-oriented applications. After using your directory server for data on humans, you soon realize that it can store other things, like information on offices, districts, assets of all sorts, rooms, conference bridges, etc.

Then things start to sweeten: network-aware devices, like routers, firewalls, and other ‘net gear start using the directory to store configuration info, addresses and mappings which make them more dynamic and manageable from one interface. Then the application developers get involved and realize there's no need for .ini or desktop registry entries anymore. Their applications can now gain location transparency by dipping into the directory for runtime configuration, as well as saved user profiles and personal preferences.

At this point, your company is thoroughly hooked on having a simple, high-speed, robust directory server at the heart of your IT world, because it has simplified life so much for you. It is at this point that you need to be cautious. The LDAP technology is your hammer and everything looks like a nail. For this reason, you should be aware of when NOT to use LDAP.

## How Does An LDAP Server Differ From Oracle?

In Part II of this series, I'll be delving into Oracle's LDAP directory. But now is a good time to define an LDAP directory server in general (a DS), in light of our focus on Oracle (an RDBMS). The goal is to understand when it's right to use a directory server and when you should stick to Oracle.

First, to clear a common confusion: LDAP is **not** the same as the directory server, but it is shorter and more fun to say. So you will often hear, “Let’s store that in LDAP.” LDAP is a protocol, a simple string-oriented means of reading or writing data to a directory server. You can’t store data away in a protocol. But in everyday parlance, “LDAP” is used interchangeably with “directory server.”

A DS is a hierarchical database, not relational (although some DS’s are able to maintain data integrity). This means several things, one of which is you can’t use SQL against it. LDAP and LDAP filters are the “query language” for DS’s. It will be a new, but short, learning curve.

A DS is optimized for fast, frequent reads, but very few writes and updates. RDBMS’s are great at I/Os of all sorts, but they’ll probably never reach the read speeds of DS’s.

A DS functions as a repository for fairly static, heavily-read, shared information. You should never put transactional data in a DS. That’s reserved for an RDBMS, which is great at heavy insert, update and delete activity.

A DS is flexible. It is very easy to extend existing objectclasses or create new ones. You can store information about **anything** in a DS, including binary data. Up until recently, RDBMS’s were incapable of allowing user-defined datatypes, and getting binary data into and out of an RDBMS was not at all easy.

A DS is usually small, simple to install, very robust and easily optimized. Our DS at NGT is only a few megabytes in size, takes very little RAM, and hasn’t gone down once since we turned it on in November 2000. One cannot, with a straight face, say any of the same things about large RDBMS’s.

A DS is meant to run both inside and outside an internal network. This allows a company’s external partners and customers to participate in processes and services that use the centralized information in the DS. You would never even think of putting an enterprise RDBMS outside the firewall!

A DS usually comes with all sorts of built-in mechanisms to address security concerns, from third party authentication, to SSL or TLS encryption, to fine-grained and policy-based access control, even down to the individual attribute level. If an RDBMS even has these features, it will probably be extra cost and difficult to configure. Oracle does have some equivalent functionality, but I have yet to hear of a DBMS that can handle access control down to the individual column in a row.

Since the read activity can get intensive for large enterprises using LDAP technology, many DS’s are needed with a mirror of the centralized data. So most DS’s come with push/pull replication that is easy to use and configure. In the relational database world, if this feature is offered at all, it is expensive and takes a guru to install and configure.

Hopefully, you now understand what a directory server is good at. If not, or you deem my writing to be pure tripe, check out [http://www.ldapzone.com/why\\_ldap.html](http://www.ldapzone.com/why_ldap.html). It’s a good start when you need to form a business case around migrating functionality to an LDAP server.

*Note: Since Oracle’s LDAP directory, OID, stores its data in an RDBMS, many of the points above are questionable or foggy. Some LDAP purists are unhappy about this mainly for performance and replication/scalability reasons. For a decent paper on why you wouldn’t use an RDBMS to do a directory server’s job, see <http://www.openldap.org/faq/data/cache/378.html>.*

## How Do I Start Learning LDAP?

When learning a foreign language, like Spanish, you find yourself translating everything you’re reading and hearing in your head. It’s frustrating and time-consuming, but it gets the job done. Occasionally you come across a familiar word, like *computadora* and *comunicación*. Others throw you for a loop, like *embarazada*, which means pregnant, not embarrassed. So you need to exercise care in your assumptions. However, if you immerse yourself in the language, you eventually find you think and dream with it.

Learning to think in LDAP is similar. I can't bring you to the second stage of learning the LDAP language, but it's easier if you have a "map" that translates LDAP terminology into things you're already familiar with, like relational design concepts or OO programming. Table 2 at the end of this section, and the following discussion attempts to give you that map:

First, a directory is a hierarchical structure, much like the listing of your hard drive or a Unix file system. A directory is composed entirely of entries. A collection of entries, and the hierarchy they fall into, is called the Directory Information Tree (DIT). See Listing 1 for a simple DIT. Even the items that look like folders, such as groups and people, are also just another type of entry.

The entry, like the object in OO systems, is the key element of LDAP technology. An entry contains data about an object (like a department, person, or router). This data is found in the entry's attributes (like my direct line, fax number, and title). See Listing 2 for the sample *IT* and *Bill Coulam* entries that are in LDIF format (used for exporting and importing entries and schemas into an LDAP server).

Attributes have a type and one or more values. The "ou" in my personal entry is an attribute type. The string following the colon is the value. As you can see, I have several attributes that are repeated. Each attribute type has a syntax that may specify type-checking (e.g., does this attribute store a DN, a string, or binary values?) and behavior during directory operations (e.g., does the compare operation use case-sensitivity against this attribute or not?).

Each entry is uniquely named in relation to the other entries at its level. This is its Relative Distinguished Name (RDN). My RDN is [uid=bill.coulam@ngt.com](#). It is unique among all the employees at NGT whose entries are at my level under ou=people. Each entry also has a globally unique name, called a Distinguished Name (DN). Moving up the hierarchy from bottom to top, the DN is composed of the RDN at each level. My DN is [uid=bill.coulam@ngt.com.ou=people.dc=ngt.dc=com](#). Like DNS addresses, the DN should guarantee uniqueness among all LDAP servers. For this reason, most DITs today have a base DN that mimics their DNS domain, e.g., our domain is ngt.com; therefore, our base DN is dc=ngt,dc=com.

An LDAP objectclass, much like an OO class, is a template that specifies the data members that fully describe an entry. Part of the objectclass's job is to enforce which attributes are required and which are allowed (or optional) in an entry. If designed well, this template can be used by other objects. Looking at my entry above, you can see that it is based on a combination of five templates, or objectclasses. This is roughly equivalent to multiple inheritance in C++, or using multiple interfaces in Java (except that no methods are ever involved). Most standard directory schemas stop at inetOrgPerson. However, we had some special attributes that we needed to record. So we added a new objectclass, ngtPerson, to our LDAP schema, and added it to all our "people" entries. This sort of modification is very easy to do, which is another reason why directory servers are so flexible.

An LDAP schema is a collection of objectclasses, attribute definitions and security constraints that describe the parameters of your directory server, who has access to what, what attributes certain entries must have, where certain entries must be placed, etc. If you were to reverse engineer all the DDL for a given Oracle database, you'd have a rough analog to the LDAP schema.

When you need to find something in a directory, you tell the server where you want to start searching in the hierarchy with a scope identifier (whole tree, from this branch down, or just this entry), and what criteria you want to use to limit the search, which is known as the search filter.

LDAP filters require their own RFC (2254). We'll simplify a little. You may search for any entry or set of entries. You may search both by attribute types or their values. You have the usual =, >= and <= operators, but you're also given the \*= operator for partial string, or wildcard comparisons, and the ~= operator for comparing values that are similar (equivalent to the Oracle SOUNDEx function). You also have Boolean operators available, the usual | for OR, & for AND and ! for NOT. However, LDAP uses a non-intuitive way of constructing Boolean phrases. In this case, expressions are enclosed in parentheses, with the Boolean operators preceding the expressions, which are read from left to right, innermost to outermost. See Table 1 for examples.

**Table 1**

Filter	Paraphrased Query Expression
(!(ou=Executives))	everyone that isn't an executive
(&(ou=IT)((title=*developer*)(title=*analyst*)))	everyone in IT that is also a developer or analyst
((sn~=christiansen)(sn~=jensen))	everyone whose surname sounds like Christiansen or Jensen. Would return Christensen, Jenson, etc.
(&(objectclass=ngtOffice)(c=US))	Contact info on every office in the USA

**Table 2**

LDAP Concept	Close Oracle or OO Equivalent
Directory Server (DS)	Database Server (DB)
LDAP <b>protocol</b>	SQL *Net/Net8/Oracle Net or RMI/IIOP
LDAP <b>API</b> in C, Java or Perl	SQL DML or JDBC statements
DIT	Table list and their relationships
Organizational entry	Table Name
Object entry	Row with nested table columns
Schema	DDL for tables and ADTs + portions of the data dictionary
Objectclass	Table structure (poor comparison) or OO class (w/o methods)
Attribute type	Column Name (nested table column being the best comparison)
Attribute value	Column value
Attribute syntax	Column datatype and constraints
base DN	DNS domain
DN	<DNS domain>.<Oracle SID>.<Schema>.<Table>.<ROWID/PK>
RDN	UK or UROWID
Filter	WHERE clause
Scope	<i>None</i>
LDAP URL	<i>None</i>
Abandon operation	Killing SQL client
Search and Compare ops	Select
Add, Modify, Delete ops	Insert, Update, Delete
Bind, Unbind ops	Log into/out of DB with account and password

## Do Try This At Home

Now let's put some of this into practice. If you have IE or Netscape, and either a real directory server, or Microsoft Exchange/Windows 2000 AD in place, you can try some LDAP searches right at work. If you've got serious time to kill, you could download, install and populate the openLDAP server, or use Novell's public test server. The examples below use "host" in place of whatever your directory server's hostname or IP address is. You'll have to get the "baseDN" from your directory admin. For some reason, trying these against Exchange, I had no need for the baseDN, but against iPlanet, it was required.

You may execute LDAP searches in your browser using LDAP URL syntax. The syntax is as follows:

```
ldap[s]://<host>[:<port>]/<baseDN>?<attributes>?<scope>?<filter>
```

[s] operates over SSL

baseDN is where in the DIT you want the search to start; the default is the root

attributes is a comma-separated list of attribute types; if left out, the default brings back all attributes

scope is either base|one|sub; the default is base, which isn't too useful, so most URL's use "sub"

The simplest LDAP URL is to search for yourself by last name:

```
ldap://<host>/<baseDN>??sub?(sn=<lastname>)
```

Return the name and members of every department in the company (assuming the standard schema):

```
ldap://host/ou=groups,<baseDN>?cn,uniquemember?one?(objectclass=groupOfUniqueNames)
```

Return the enterprise and application group DNs you belong to:

```
ldap://host/<baseDN>?dn?sub?(&(objectclass=groupOfUniqueNames)(uniquemember=<yourDN>))
```

## Case Study

Before we dive into the code, a little more background is in order. We were doing a project for a large telco back in 1997. After considering numerous alternatives, we chose the Oracle Web Server and PL/SQL web toolkit to serve up the application. The client had very specific security requirements that the built-in security mechanisms of OWS 3 could not meet. So I designed a PL/SQL-based web application framework, with services for users, profiles, dynamic menuing, sessions, UI abstraction, form validation, cookies, etc. A primary feature was fine-grained A&A services where users, groups, objects and their associations were rows in tables, as opposed to clumsy Oracle accounts. My next employer was another Oracle-centric shop. I selected OAS 4 again and rewrote the framework. It was very useful having “lightweight” users (a feature that finally showed up in WebDB/Portal 3.0).

I didn’t realize it at the time, but I had built another isolated silo. That would have been fine if all our applications had used the same framework, but our needs quickly changed. Teams started doing their own thing, and uncontrolled purchasing brought third-party systems and databases inside our walls. Without the power to enforce adherence to the enterprise architecture, the continuity fell apart. In two short years we had eight different silos of A&A data. The future looked messy. If we kept producing silos at that rate, our little IT department would soon be crushed under the weight of supporting our own systems.

Our Joo Janta Peril-Sensitive sunglasses went totally black (they’re a Douglas Adams thing ;-). Sensing the impending doom, we bought SilverStream (mainly for its fast, visual servlet designer and built-in LDAP support) in order to rewrite our web applications, all going to a single directory server for A&A functionality. It’s one of the best moves we ever made. It’s a beautiful thing when you can give your users a single login for all the systems your shop produces. Well, all except the OAS-based applications. They’d been running almost non-stop since June 1999 and no one saw the need to rewrite them. I migrated the OAS-based systems to 9iAS last July. It went smoothly, but the issue of a separate login remained; that is, until the release of 8.1.7, which came with the DBMS\_LDAP package. That explains how I arrived at this case study for RMOUG. I’ll briefly cover the few A&A functions that I modified, and how dbms\_ldap operations flow. Although dbms\_ldap covers all the LDAP operations (including entry modification and deletion), the code conversions only cover binding, searching and comparing.

## DBMS\_LDAP

With dbms\_ldap, it was now possible to get A&A data from our iPlanet directory server, the same source all our other applications were using. All I had to do was change the implementation of my A&A procedures, bypassing the local tables I’d been using. If you are storing user or group names in any Oracle table, hard-coding them, or sending e-mail to users or groups from PL/SQL, it is likely that you too could benefit from using dbms\_ldap, assuming you now have, or will soon have, an LDAP directory in place.

You’ll encounter a few enigmas in dbms\_ldap, but reading RFC 2251 (if nothing else, to cure your insomnia) will clear up most of them for you. Please glance at the dbms\_ldap API documentation now. If it’s not loaded in the SYS schema on your 8.1.7 or 9i DB, with a public synonym, have your DBA run <ORACLE\_HOME>/rdbms/admin/catldap.sql as SYS. Then open the dbms\_ldap spec using TOAD or similar tool. Alternatively, you can read about it at [http://download-west.oracle.com/otndoc/oracle9i/901\\_doc/network.901/a90152/plsql\\_pk.htm#1017684](http://download-west.oracle.com/otndoc/oracle9i/901_doc/network.901/a90152/plsql_pk.htm#1017684). Notice the provided constants, exceptions, and data types. Read a few of the comments for the init, bind and search functions. Then return here.

Our custom 9iAS framework also included presentation and business logic that allowed users to manage their own profile and password, and superusers to create and manage other accounts. But they all go away with the migration to LDAP, since we have a separate JNDI web application for that functionality.

That left three routines that needed to use `dbms_ldap`: one that authenticated a user's login attempt, one that determined if the user belonged to a named group, and one that got a list of an application's user groups for drop-down display. There was a fourth, but it should be left for another article.

*Note: All three routines belong to my "ia\_aa" package. So if the code refers to a seemingly undeclared constant or PL/SQL routine, it's because the item is found elsewhere in the ia\_aa package spec/body, or in another package. The "msg" calls are to a custom package that replaces `dbms_output`, handles logging/notification, manages error messages, and controls debugging statements. Also, our naming standards follow, for better or worse, a pseudo-Hungarian notation. Variables that start with 'l' or 'g', are local or global. Functions begin with 'f'. Types start with 't', etc. The letter(s) following the first identifier indicate the datatype.*

## ***fb\_authenticates***

The original function is shown in Listing 4 (old). As you can see, given the values from the login page, it attempts to find a match in the user table. Although 8i has a cryptography toolkit, the encrypt/decrypt functions you see work with older versions of Oracle. I got the original code for them off the usenet. Feel free to email if you'd like a copy.

Listing 4 (new) simply removes the TYPE dependency on the tables and replaces the SELECT statement with an LDAP bind attempt. If the bind is successful, then the user's authentication credentials were valid. The function is longer now thanks to the overhead of creating and dropping a "session" with the LDAP server.

Since `init`, `bind` and `unbind` must be called for every LDAP operation, I'd extract them into separate, modular functions in the A&A package, thus hiding the detail and error checking, and eliminating redundancy (see Listing 6 new). But I wanted to clearly show the steps of using `dbms_ldap`, so I chose to include them inline to prevent the reader from having to mentally jump around.

`LDAPHOST` and `LDAPPOR`T are constants set in the specification of my A&A package. You would replace these with the name or address of your directory server, and port 389 (636 if doing SSL LDAP).

Everything about this conversion works well in practice. However, I chose to give the new routine a little stress test. The overhead of stepping out of Oracle and performing packet operations over the network to the directory server meant that we would probably see a performance hit. And the metrics bear that out. It took about two seconds to perform 1,000 authentications the old way, and about six seconds to do 1,000 with `dbms_ldap`. But since our user base is less than 200, with maybe five concurrent, this was not a concern.

## ***fb\_belongs\_to\_grp***

The original code is Listing 5 (old). My apologies for the triple-nested functions in the first statement, but I find it an elegant way to code that seems to come naturally when using a modularized framework. The "NP" in the session cookie name stands for Non Persistent (it disappears when the browser closes).

Listing 5 (new) makes use of the LDAP compare operation. Give it the entry's DN, the attribute to check, and what value the attribute should have, and it returns true or false. Although I could have used a search and checked the resulting entries for a filled or empty result set, compare is custom-built for a query like this, i.e., Does this user belong to that group? How I wish Oracle had an existence operation so I could bypass SELECTING COUNT(\*) into a local and then checking it for equality to zero!

## ***get\_groups***

The old way of getting a list of groups for a given application was very straightforward (see Listing 6 old ). Using dbms\_ldap (Listing 6 new) is obviously more involved, but well worth it.

By default, dbms\_ldap raises exceptions as its error handling mechanism. If you want more control over exception handling, set the USE\_EXCEPTION toggle to FALSE (commented out in first statement). This prevents the dbms\_ldap package from raising exceptions that halt your program. Instead, you will need to check the return codes from all operations and branch your logic from there. You may want to use the exceptions and err2string function provided by dbms\_ldap to customize your error handling.

The get\_groups routine is a good demonstration of the steps required to do a “select” against an LDAP directory. The basic idea is you: init, bind, compose a list of attributes you’re searching for, search, iterate through the entries in the result set and pull the attributes desired, and for each attribute, pull its values. If you were dealing with rows in an Oracle table that had several nested table columns, your SELECT statement would be almost as complex, so not much is lost in the conversion here.

Since get\_groups just needs a list of group names, not entire DNs, I use dbms\_ldap.explode dn to break apart the DN, and then skim off the first token, the group name. Be aware that explode\_dn puts its tokens into the receiving PL/SQL index-by table starting at subscript 0. I’d forgotten that index-by tables can have subscripts using negative integers and zero. So my first version didn’t work until I replaced the hard subscript (1) with (last\_vals.FIRST). I skip the step of getting the attributes and their values, since the DN contained the group’s name. See Listing 7 for the additional code needed to get attributes.

The calls to first\_entry and next\_entry are interesting. It’s not possible to just call next\_entry in a loop. You **must** call first\_entry first. After processing the first entry, you may now call next\_entry in a loop where its result is used as the input for the next call. The entry variable serves as a sort of pointer, telling the LDAP API where to go in the result set.

## ***Migrating Directory Data from Oracle to an LDAP Server***

When we first set up our directory server, we had to migrate our user/group info out of Oracle. Gurmeet Singh of DCC (divcomp.com) was my LDAP mentor at the time. He pointed out the use of UTL\_FILE to read from our custom tables and write LDIF-formatted records for easy importation into the LDAP server. This worked flawlessly.

## ***Drawbacks***

By moving my user/group info out of the database, and into the directory where it belonged, a minor drawback dawned on me: there is a disconnect between the two. Now I no longer have DB-managed data integrity. If a user is modified or deleted from the DS, I currently have no way of synchronizing with the DB. For example, if I delete user X from the DS, but I’ve kept a record of every order that user X has placed, the rows associated with X will be “orphaned” in the DB. So far, it hasn’t been a concern. The orphans we’ve created in Oracle by deleting LDAP users are things like sessions, work orders and trouble tickets that I don’t want a cascading delete to affect anyway since they’re valuable history. So I’m putting off writing any synchronization subsystem for later.

## ***Bio***

Bill spent the last 7 years building n-tier OSS and intranet applications, first for large telcos as an Andersen Consulting senior, and now as the app architect and Oracle specialist for New Global Telecom in Golden. Once in a while he leaves his cube to explore Denver's mountains on his bike or with his 6 year old.  
bill.coulam@ngt.com.

## Code Listings

### Listing 1

```
com
  ngt
    offices
    groups
      IT (see Listing 2)
    people
      Bill Coulam (see Listing 3)
    apps
    extranet
      customer1
        groups
        people
      partner1
      vendor1
```

### Listing 2

```
dn: ou=IT,ou=groups,dc=ngt,dc=com
objectclass: top
objectclass: organizationalUnit
objectclass: groupOfUniqueNames
uniquemember:uid=bill.coulam@ngt.com,ou=people,dc=ngt,dc=com
uniquemember:uid=fozzy.bear@ngt.com,ou=people,dc=ngt,dc=com
uniquemember:uid=suzi.consultant@bigbucks.net,ou=people,
ou=vendor1,ou=extranet,dc=ngt,dc=com
```

### Listing 4 (old)

```
FUNCTION fb_authenticates
(
  is_user_id IN w_user.user_id%TYPE,
  is_password IN w_user.password%TYPE
) RETURN BOOLEAN
IS
  ln_rowcount PLS_INTEGER := 0;
BEGIN
  -- Use simple count to see if user exists in
  -- table where the given password matches
  SELECT COUNT (*)
  INTO ln_rowcount
  FROM w_user
  WHERE user_id = is_user_id
  AND password = fs_encrypt(is_password)
  AND user_status <> 'I';

  IF (ln_rowcount > 0) THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END fb_authenticates;
```

### Listing 3

```
dn:uid=bill.coulam@ngt.com,ou=people,dc=ngt,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: ngtPerson
cn: Bill Coulam
sn: Coulam
uid: bill.coulam@ngt.com
mail: bill.coulam@ngt.com
title: Lead Pizza-Fetcher
mgr:
uid=jon.smith@ngt.com,ou=people,dc=ngt,dc=com
ou: IT
ou: Software Development
telephonenumber: 303.239.1078
facsimiletelephonenumber: 303.999.9999
mobile: 303.555.5555
jpegPhoto: <binary data>
ngtprimaryoffice:
cn=Golden,ou=offices,dc=ngt,dc=com
ngtalternatemail: 30399999998@mobile.att.net
ngthiredate:
ngtreleasedate:
ngtemployeestatus:
<address attributes snipped>
```

### Listing 4 (new)

```
FUNCTION fb_authenticates
(
  is_user_id IN VARCHAR2 - should be full DN
  ,is_password IN VARCHAR2
) RETURN BOOLEAN
IS
  l_session dbms_ldap.SESSION;
  lb_authenticatd BOOLEAN := FALSE;
  lx_session_err EXCEPTION;
BEGIN
  -- must establish a valid session with LDAP host
  -- before attempting any LDAP operations
  l_session := dbms_ldap.init(LDAPHOST,LDAPPOR);

  -- authenticate to LDAP host using DN and password
  IF ( dbms_ldap.simple_bind_s(
    l_session
    ,is_dn
    ,is_password
  ) = dbms_ldap.SUCCESS) THEN
    lb_authenticatd := TRUE;
  ELSE
    lb_authenticatd := FALSE;
  END IF;

  -- unbind, destroying the session handle to the
  -- LDAP host and its services
  IF ( dbms_ldap.unbind_s(l_session) <> dbms_ldap.SUCCESS) THEN
    RAISE lx_session_err;
  END IF;

  RETURN lb_authenticatd;
EXCEPTION
  WHEN lx_session_err THEN
    RAISE APPLICATION_ERROR (-20500,
      'Communicating with '||LDAPHOST||' failed.'||c.LF||
      c.CONTACT_SYSADMIN);
    RETURN lb_authenticatd;
  WHEN dbms_ldap.init_failed THEN
    RAISE APPLICATION_ERROR (-20501,
      'Unable to establish session with '||LDAPHOST||c.LF||
      c.CONTACT_SYSADMIN);
    RETURN FALSE;
END fb_authenticates;
```

**Listing 5 (old)**

```

FUNCTION fb_belongs_to_group
(
  is_group_nm IN VARCHAR2
  ,is_user_id IN VARCHAR2 DEFAULT NULL
) RETURN BOOLEAN
IS
  ls_user_id w_user.user_id%TYPE;
  ln_rowcount PLS_INTEGER := 0;
BEGIN
  -- retrieve user_id from session table using
  -- session cookie as the key for lookup
  ls_user_id :=
    NVL(is_user_id
      ,ia_sess.fs_get_sess_user(
        ia_cook.fs_get_cookie(ia_cook.NPSESSION)
      )
    );

  -- determine if user is a member
  SELECT COUNT(*)
  INTO   ln_count
  FROM   w_user_group
  WHERE  user_id = ls_user_id
  AND    group_nm = is_group_nm;

  IF (ln_rowcount > 0) THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END fb_belongs_to_group;

```

**Listing 6(old)**

```

PROCEDURE get_groups
(
  is_app      IN   VARCHAR2 -- system owner of group list
  ,oas_groups OUT t.tas80 -- pl/sql table of varchar2(80)
) IS
BEGIN
  FOR lr_cur IN (
    SELECT group_nm FROM w_group
    WHERE  app_owner = is_app
  ) LOOP
    oas_groups(oas_groups.COUNT+1) := lr_cur.group_nm;
  END LOOP;
END get_groups;

```

**Listing 5 (new)**

```

FUNCTION fb_belongs_to_group
(
  is_group_nm IN VARCHAR2 -- should be DN of group OU
  ,is_user_id IN VARCHAR2 DEFAULT NULL
) RETURN BOOLEAN
IS
  ls_user_id   VARCHAR2(60);
  l_session    dbms_ldap.SESSION;
  ln_rc        PLS_INTEGER := 0;
  ...<other vars snipped>
BEGIN
  ...<code to get session and bind snipped>
  ls_user_id :=
    NVL(is_user_id
      ,ia_sess.fs_get_sess_user(
        ia_cook.fs_get_cookie(ia_cook.NPSESSION)
      )
    );

  ln_rc := dbms_ldap.compare_s(
    l_session
    ,is_group_nm -- DN of group to check
    ,'uniquemember' -- attribute to check
    ,ls_user_id -- attribute value to check
  );

  ...<code to unbind snipped>

  IF (ln_rc = dbms_ldap.COMPARE_TRUE) THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;

  ...<code to handle exception snipped>
END fb_belongs_to_group;

```

**Listing 6 (new)**

```
PROCEDURE get_groups
(
  is_dn      IN    VARCHAR2 -- should be DN of group OU
  ,oas_groups OUT  t.tas80 -- pl/sql table of varchar2(80)
)
IS
  -- heavily-used vars for most dbms_ldap functions
  l_session   dbms_ldap.SESSION; -- handle to LDAP server
  l_results   dbms_ldap.MESSAGE; --handle to LDAPMessage envelope
  las_vals    dbms_ldap.STRING_COLLECTION; --for breaking apart the DN

  -- vars for handling entry iteration
  l_entry     dbms_ldap.MESSAGE;
  ln_entry_idx PLS_INTEGER := 0;
  las_attrs   dbms_ldap.STRING_COLLECTION;

  ln_rc       PLS_INTEGER := 0;
  lx_session_failure EXCEPTION;

BEGIN
  --dbms_ldap.USE_EXCEPTION := FALSE;
  -- call ia_aa func to init session and bind anonymously to dir server
  start_ldap_sess(l_session); -- session var is IN/OUT

  las_attrs(1) := 'dn'; -- list of attributes desired

  -- execute our search to find all group entries under a given DN.
  handle_rc(
    dbms_ldap.search_s(
      l_session -- handle to LDAP server comm session
      ,is_dn -- starting point of search in DIT
      ,dbms_ldap.SCOPE_SUBTREE -- how deep to search
      ,'(objectclass=groupOfUniqueNames)' -- filter
      ,las_attrs -- array of attributes desired in the result
      ,0 -- 0=attributes and their values; non-zero=attributes only)
      ,l_results
    ), 'search');

  IF (dbms_ldap.count_entries(l_session, l_results) > 0) THEN -- iterate through any entries

    -- MUST use first_entry func to get the required arg for later call to next_entry
    l_entry := dbms_ldap.first_entry(l_session, l_results);

    WHILE (l_entry IS NOT NULL) LOOP
      ln_entry_idx := ln_entry_idx + 1;
      -- get DN from entry, and tokenize the parts...
      las_vals := dbms_ldap.explode_dn(
        dbms_ldap.get_dn(l_session, l_entry)
        ,1 -- "0" leaves attribute in token, e.g. 'ou=', "1" skims off the attributes types
      );
      -- ...to place group name from first RDN into outbound result
      oas_groups(ln_entry_idx) := las_vals(las_vals.FIRST);

      l_entry := dbms_ldap.next_entry(l_session, l_entry);
    END LOOP;
  ELSE
    NULL; -- leave OUT array empty
  END IF; -- if entries returned from search

  stop_ldap_sess(l_session); -- private pkg func that unbinds
END get_groups;
```

**Listing 7**

If I'd needed the attributes as well, you would have seen additional variables...

```
-- vars for handling attribute iteration
ls_attr_nm  VARCHAR2(80);
ls_attr_val VARCHAR2(500);
ln_attr_idx PLS_INTEGER := 0;
--handle to a BER structure used for decoding incoming messages
l_ber_handle dbms_ldap.BER_ELEMENT;
```

...and another WHILE loop with calls to first\_attribute, next\_attribute and get\_values:

```
ln_attr_idx := 0; --reset attribute counter
ls_attr_nm := dbms_ldap.first_attribute(l_session, l_entry, l_ber_handle);
WHILE (ls_attr_nm IS NOT NULL) LOOP
    ln_attr_idx := ln_attr_idx + 1;

    -- there may be more than one instance of an attribute for each entry
    -- get all the values for the current attribute
    las_attrs := dbms_ldap.get_values (l_session, l_entry, ls_attr_nm);

    <Do stuff with the attributes you now have in las_attrs>

    -- try to get another attribute
    ls_attr_nm := dbms_ldap.next_attribute(l_session, l_entry, l_ber_handle);
END LOOP;

IF (ln_attr_idx = 0) THEN
    msg.p('No attributes for this entry!');
END IF;
```

## Resources

<http://www.openldap.org> (took over for U. of Michigan site as free LDAP Mecca)

<http://developer.{netscape|planet}.com/tech/directory/index.html>

<http://www.slapped.net/>

<http://www.nldap.com/NLDAP/> Novell's public LDAP server for testing purposes.

<http://www.cygsoft.com/> - free LDAP browser, for Windows

Oracle9i Directory Service Integration and Deployment Guide

Oracle Internet Directory Administrator's Guide

**Oracle Internet Directory Application Developer's Guide – best place to start**

Oracle9i Net Services Administrator's Guide

Oracle LDAP FAQ (Doc ID 135696.1 on Metalink)

LDAP is defined by RFCs 2251-2256,2829,2830. Here are the most relevant:

[RFC 2251 - LDAP \(v3\)](#)

[RFC 2252 - LDAP \(v3\): Attribute Syntax Definitions](#)

[RFC 2253 - LDAP \(v3\): UTF-8 String Representation of Distinguished Names](#)

[RFC 2254 - The String Representation of LDAP Search Filters](#)

[RFC 2255 - The LDAP URL Format](#)

*LDAP Programming with Java*, by Rob Weltman and Tony Dahbura

*Understanding and Deploying LDAP Directory Services*, by Howes, Smith and Good

<http://www.networkmagazine.com/article/DCM20000502S0039/3> - why LDAP, by Tim Howes

<http://www.ldapman.org/articles/index.html> - articles on designing/deploying a directory