



[TechNet Home](#) > [Script Center](#) > [Learn to Script](#)

Tales from the Script

May 2005

Published: May 19, 2005



By [The Scripting Guys](#)

For a list and additional information on all Tales from the Script columns, [click here](#).

On This Page

- ↓ [Dude, Where's My Printer: Using Scripts to Search Active Directory \(Part 2 of 2\)](#)
- ↓ [The Basic Parts of a SQL Query](#)
- ↓ [Specifying the Attributes to be Returned](#)
- ↓ [Returning Multiple Attributes: The Strange Case of SELECT * FROM](#)
- ↓ [Specifying the Search Starting Location](#)
- ↓ [Specifying an Optional WHERE Clause](#)
- ↓ [Searching Object Classes and Object Categories](#)
- ↓ [Using AND and OR in Queries](#)
- ↓ [Specifying Values in a WHERE Clause](#)
- ↓ [Using Variables in a WHERE Clause](#)
- ↓ [Using Operators in WHERE Clauses](#)
- ↓ [Using Wildcards in a WHERE Clause](#)
- ↓ [So Now We Know Everything About Searching Active Directory, Right?](#)



Dude, Where's My Printer: Using Scripts to Search Active Directory (Part 2 o

We learn more by looking for the answer to a question and not finding it than we do from learning the answer itself

-- Lloyd Alexander

Lloyd Alexander is a writer of fantasy books for children, so it's understandable that he doesn't speak for the typical system administrator. And it's probably true that, in the fantasy world, there is great value in searching for – and not finding – the answer to a question. For system administrators, however, that usually isn't the case:

“Did you find a list of all the printers owned by the Finance Department?”

"No, I didn't. But I really think I learned some valuable lessons in the process of searching for the answer."

We can think of one valuable lesson this system administrator probably learned: in the real world it's usually better answer than to *not* find the answer. (As we Scripting Guys all know from painful personal experience.)

Fortunately many of the answers we need are in Active Directory: all we have to do is ask and Active Directory will happy to help. And that's the purpose of this month's *Tales from the Script* (and yes, contrary to popular belief, sor column *does* have a purpose). In **Part 1** of this two-part series we provided you with the fundamental principles be searching Active Directory. In Part 2 we're going to get down to the real nuts-and-bolts behind searching Active Dir going to show you how to write SQL queries that will help you return the exact information you're looking for. No r less.

Note. There's actually a second type of syntax that can be used to search Active Directory: the LDAP syntax. We f syntax to be far easier to understand, so we're going to focus on that. If you're interested in the LDAP syntax take so out of your busy day to watch this [Scripting Guys webcast](#) on searching Active Directory. Or don't. After all, : you can learn more from not watching a webcast than you can by watching a webcast. Or so they say.

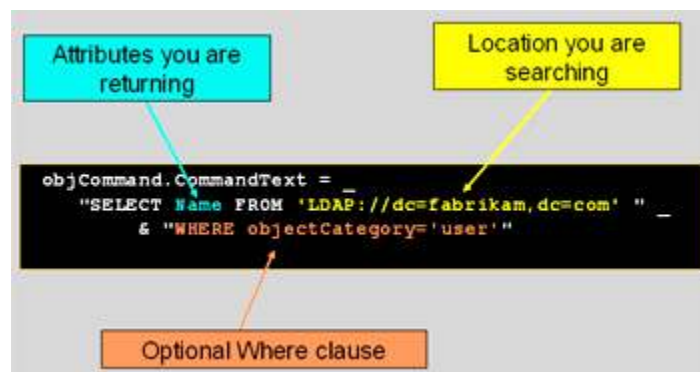
[↑ Top of page](#)

The Basic Parts of a SQL Query

One of the Scripting Guys has a hazy memory of a movie where a young man approaches a young woman and asks kiss. She promptly slaps him. The young man's friend goes up to the same woman and asks for a kiss, and she pro Why did he succeed where the first man failed? "It's not *what* you ask," said the successful suitor. "It's *how* you ask

The same thing is true of Active Directory. Active Directory won't slap you if you ask a question the wrong way (alth being considered for future versions); however, if you ask the wrong question it's a pretty safe bet that Active Dire return the wrong information (if it returns any information at all). What does that mean? That means that the secre successful Active Directory searching hinges not only on asking the right question but also in asking that question t And asking a question the right way means carefully crafting your SQL query.

In this month's column we'll explain the mysteries and the nuances behind writing SQL queries to search Active Dir that end we've divided a typical SQL query into the three pieces shown in the following diagram:



For a full-sized version of this diagram click [here](#).

What we'll do now is walk you through each of these items:

- Specifying the Attributes to be Returned
- Specifying the Search Starting Location
- Specifying an Optional WHERE Clause

Let's have at it.

[↕ Top of page](#)

Specifying the Attributes to be Returned

Part 1 in our SQL query is pretty straightforward: all you do is specify the name of the attribute you want returned. do you want to return the name of all the objects in Active Directory? (Don't worry; we'll show you how to limit return a particular category – such as user accounts – in a bit.) Then use this query:

```
"SELECT NAME FROM 'LDAP://dc=fabrikam,dc=com'"
```

How about the ADsPath for all the objects in Active Directory? Here you go:

```
"SELECT ADsPath FROM 'LDAP://dc=fabrikam,dc=com'"
```

Pretty easy, huh? But what if you want to return multiple attributes; in fact, what if you want to get back *all* the attributes that case you probably use a query like this:

```
"SELECT * FROM 'LDAP://dc=fabrikam,dc=com'"
```

Is that right, Scripting Guys? Do I use SELECT * FROM to return multiple attributes? Hello, Scripting Guys?

Listen, we need to talk....

[↕ Top of page](#)

Returning Multiple Attributes: The Strange Case of SELECT * FROM

If you've had any experience with writing SQL queries (including writing queries using the WMI Query Language, a subset of SQL) then you're no doubt familiar with using the SELECT * FROM shortcut to select all the records in a table. In the case of the properties of an object. For example, in WMI this query selects all the properties of all the instances of the Win32_Process class:

```
"SELECT * FROM Win32_Process"
```

If you're familiar with that use of SELECT * FROM then sooner or later you're bound to write an Active Directory script that does this, figuring this is a quick and easy way to get back all of the properties of an object or group of objects:

```
Const ADS_SCOPE_SUBTREE = 2

Set objConnection = CreateObject("ADODB.Connection")
Set objCommand = CreateObject("ADODB.Command")
objConnection.Provider = "ADSDSOObject"
objConnection.Open "Active Directory Provider"
Set objCommand.ActiveConnection = objConnection

objCommand.Properties("Page Size") = 1000
objCommand.Properties("Searchscope") = ADS_SCOPE_SUBTREE

objCommand.CommandText = _
    "SELECT * FROM 'LDAP://dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user'"
Set objRecordSet = objCommand.Execute

objRecordSet.MoveFirst
Do Until objRecordSet.EOF
    Wscript.Echo objRecordSet.Fields("Name").Value
    objRecordSet.MoveNext
```

Loop

We would expect this script to return all the attributes of all the user accounts in the domain; we'd then expect the dutifully echo the value of the **Name** attribute for each user. But there's a problem: instead of getting back the names of the users, you're going to get back this error message:

```
ADODB.Recordset: Item cannot be found in the collection corresponding to the requested name o
```

Huh? The only thing we tried to echo back is the Name. Are we saying that Name isn't a valid Active Directory attribute?

Listen, don't panic: Name is perfectly valid. The problem is that our query didn't return the Name attribute; that's why it can't be found in the recordset. But *why* didn't our query return the Name attribute? Didn't we use SELECT * FROM

Yes, we did, and *that's* the problem. In ADSI SELECT * FROM doesn't work the way you might expect it to. In WMI SELECT * FROM returns all the properties of an object. When querying Active Directory, however, SELECT * FROM returns only one attribute. That's it. This script – which echoes the ADsPath attribute for all the user accounts – works just fine:

```
Const ADS_SCOPE_SUBTREE = 2

Set objConnection = CreateObject("ADODB.Connection")
Set objCommand = CreateObject("ADODB.Command")
objConnection.Provider = "ADsDSOObject"
objConnection.Open "Active Directory Provider"
Set objCommand.ActiveConnection = objConnection

objCommand.Properties("Page Size") = 1000
objCommand.Properties("Searchscope") = ADS_SCOPE_SUBTREE

objCommand.CommandText = _
    "SELECT * FROM 'LDAP://dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user'"
Set objRecordSet = objCommand.Execute

objRecordSet.MoveFirst
Do Until objRecordSet.EOF
    wscript.Echo objRecordSet.Fields("ADsPath").Value
    objRecordSet.MoveNext
Loop
```

Substitute any other attribute for ADsPath in the Echo statement and the script will fail. Guaranteed.

Now, we can't say for sure why SELECT * works this way, although it likely has to do with the fact that you probably don't want to return all the properties of all the objects. If you have Microsoft Exchange installed (and who doesn't?) then each mailbox has some 300 attributes. Take 300 attributes times, say, 10,000 users and you've done two things: you've put a stonewall across the network (after all, it takes system resources to bundle up all that information), and you've sent a ton of data across the network. And all that just so you can echo back the Name of each user? Changing the behavior of SELECT * is actually a *good* thing; it keeps you from taking a shortcut that could dramatically interfere with your network.

And that *is* a good thing. It's just that, although we have nothing against ADsPath, this is hardly the only attribute we want returned from a search. For example, if we were putting together a phone directory we'd need – at minimum – first name, last name, and phone number. We've already seen that we can query Active Directory for a single attribute, but what if we want to return multiple attributes, and if SELECT * FROM won't work, does that mean we have to run separate queries for each attribute?

Relax; we wouldn't make things *that* hard on you. If you want to return more than one attribute in your query you can list each of those attributes in your SELECT statement, using commas to separate the individual attributes. For example, here's a query that returns three attributes – givenName, SN, and telephoneNumber – for all the users in the fabrikam domain:

```
"SELECT givenName, SN, telephoneNumber " _
```

```
& "FROM 'LDAP://dc=fabrikam,dc=com' WHERE objectCategory='user'"
```

Note. Whoa, whoa, wait a second there Scripting Guys! We asked for first name and last name, and yet the query uses givenName and SN. (SN?) What's the deal?

Well, here's the deal: Active Directory doesn't always use the same terminology as we do. For example, although we use something a user's last name, Active Directory calls that same thing the user's SN (surname). The user's first name in Active Directory, that's his or her givenName. So how are you supposed to know that? Well, one way to determine the terminology Active Directory uses for attributes is to check out the [Active Directory Schema](#); when you do this, make sure you use the Display Name for each attribute (the ldapDisplayName for a class represents the name we use in our scripts). For example,

Given-Name

Contains the given name (first name) of the user.

CN	Given-Name
Ldap-Display-Name	givenName
Size	-
Update Privilege	Domain administrator or account owner.
Update Frequency	When the user's record is created.
Attribute-Id	2.5.4.42
System-Id-Guid	f0f8ff8e-1191-11d0-a060-00aa006c33ed
Syntax	String(Unicode)

Now, where were we? Oh, right: retrieving multiple attributes in a query. Want to get the user's Title and Department, their first name, last name, and telephone number? Then just add those attributes to the SELECT statement (put them in the order you want; the order doesn't matter):

```
"SELECT givenName, SN, telephoneNumber, title, Department " _
& "FROM 'LDAP://dc=fabrikam,dc=com' WHERE objectCategory='user'"
```

See how easy that is? If you want to do something with an attribute (like echo back the value) then make sure that attribute is included in your SELECT query.

OK, good point: that's easy as long as you need just a few attributes. But what if you need 47 attributes, or 93 attributes, or heaven forbid – what if you really *do* need all the attributes for all your users? Are you going to have to type 300 attributes in some humungous SELECT statement stretching from here to Katmandu?

No. A much better way to do this is to return just the ADsPath for each user. After you have the ADsPath you can then individually bind to each user account and echo back as many attribute values as you want. In theory, this might be slower than returning all the values at the same time; in reality, however, you'll typically find it faster simply because it's less resource-intensive. For example, here's a sample script that returns just the ADsPath for all the users in the fabrikam domain. The script then uses the value of ADsPath to individually bind to each user account and echo back a number of properties, none of which were specified in the search criteria:

```
Const ADS_SCOPE_SUBTREE = 2

Set objConnection = CreateObject("ADODB.Connection")
Set objCommand = CreateObject("ADODB.Command")
objConnection.Provider = "ADsDSOObject"
objConnection.Open "Active Directory Provider"
Set objCommand.ActiveConnection = objConnection

objCommand.Properties("Page Size") = 1000
objCommand.Properties("Searchscope") = ADS_SCOPE_SUBTREE

objCommand.CommandText = _
```

```

        "SELECT ADsPath FROM 'LDAP://dc=fabrikam,dc=com' WHERE " _
        & "objectCategory='user'"
Set objRecordSet = objCommand.Execute

objRecordSet.MoveFirst
Do Until objRecordSet.EOF
    strPath = objRecordSet.Fields("ADsPath").Value
    Set objUser = GetObject(strPath)
    Wscript.Echo objUser.givenName
    Wscript.Echo objUser.initials
    Wscript.Echo objUser.SN
    Wscript.Echo objUser.organization
    Wscript.Echo objUser.department
    Wscript.Echo objUser.title
    Wscript.Echo objUser.telephoneNumber
    objRecordSet.MoveNext
Loop

```

When we issue the query we get back a recordset consisting of the ADsPath for each user in fabrikam.com. We walk the recordset and grab the ADsPath for the first user and store it in a variable named strPath; we then use the value of strPath to bind to the user account for that user. That's what we do in these two lines of code:

```

strPath = objRecordSet.Fields("ADsPath").Value
Set objUser = GetObject(strPath)

```

After we've bound to the user account we can echo back the values of any attributes of that account. We then loop back to repeat the process for the next user whose ADsPath is in the recordset.

Incidentally, this is the same process you need to use if you want to modify a value using a search script. The ADO Active Directory is read-only; it doesn't support UPDATE queries or any other query that modifies data. Therefore when you search for a specific user you'll have to then bind to that user account to make any updates or modifications. For example, a script that retrieves all the users in the Accounting department and changes the name of that department to Finance. The query returns only the ADsPath attribute; the script then takes the value of that attribute, connects to each user returned, and individually changes the name of the department.

```

Const ADS_SCOPE_SUBTREE = 2

Set objConnection = CreateObject("ADODB.Connection")
Set objCommand = CreateObject("ADODB.Command")
objConnection.Provider = "ADsDSOObject"
objConnection.Open "Active Directory Provider"
Set objCommand.ActiveConnection = objConnection

objCommand.Properties("Page Size") = 1000
objCommand.Properties("Searchscope") = ADS_SCOPE_SUBTREE

objCommand.CommandText = _
    "SELECT ADsPath FROM 'LDAP://dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user' AND Department='Accounting'"
Set objRecordSet = objCommand.Execute

objRecordSet.MoveFirst
Do Until objRecordSet.EOF
    strPath = objRecordSet.Fields("ADsPath").Value
    Set objUser = GetObject(strPath)
    objUser.Department = "Finance"
    objUser.SetInfo
    objRecordSet.MoveNext
Loop

```

[↩ Top of page](#)

Specifying the Search Starting Location

One of the true benefits of Active Directory searching is that it, well, enables you to search all of Active Directory. Do you want to search all your users, all your computers, or all your published printers? You can get that information just by searching Act

and specifying your domain root as the starting point for the search.

Those of you familiar with WMI are used to writing a query such as this, which selects a property (Name) from a WMI class (Win32_Process):

```
"SELECT Name FROM Win32_Process"
```

When searching Active Directory you use a similar syntax: you simply substitute an ADSI attribute for the WMI property and the ADsPath of the starting search location for the WMI class. For example, the following query returns the Name attribute of user objects in fabrikam.com (we're assuming that the search scope has been set to the default value of **ADS_SCOPE_BASE**):

```
"SELECT Name FROM 'LDAP://dc=fabrikam,dc=com'"
```

How do we know that this query will search the entire domain? That's easy: we've specified the ADsPath of the domain as the starting point:

```
'LDAP://dc=fabrikam,dc=com'
```

Because we're using the default search scope our script will not only search the domain root but also any other OUs found in the domain root as well as their subcontainers. In effect, that's everything in the domain.

By the way, take careful note of the single quote marks that surround the ADsPath, leave those out and your script

Note. What if you wanted to search all your domains? Well, assuming the attributes you're looking for are replicated in the global catalog then you can just connect to the GC: (global catalog) provider rather than the LDAP: provider:

```
'GC://dc=fabrikam,dc=com'
```

Searching the entire domain is extremely useful. However, there will be times when you want to do a more targeted search. For example, you might want to limit the returned data to users in the Finance OU (and any sub-OUs within the Finance OU). That's the case then all you have to do is specify a different starting point for your search: instead of starting the search at the domain root, start it in the OU of interest. For example, here's a SQL statement that returns the ADsPath for all the user objects in the Finance OU (and any sub-OUs of the Finance OU). Note that we don't specify the domain root this time around; instead we specify the ADsPath of the Finance OU:

```
"SELECT ADsPath FROM 'LDAP://ou=Finance,dc=fabrikam,dc=com' WHERE objectCategory='user'"
```

Suppose the Finance OU doesn't have any sub-OUs. In that case you wouldn't need to use a search script to get a list of all the users; you could simply bind to the OU and list all the user accounts. If the Finance OU *does* have sub-OUs, though, it becomes a true godsend: otherwise you'd have to write a recursive function to get at all the sub-OUs (and any sub-objects contained within the sub-OUs). See how confusing that can be just trying to talk about it? Scripting it can be even more confusing.

But what if you *do* want to search only one OU, ignoring any sub-OUs? For example, maybe you want a list of all the user objects in the Finance OU who aren't actually members of the Finance department. Doing a search of that OU will probably be faster than binding to each user account and checking the value of the Department attribute.

In a case like that, you should still specify LDAP://ou=Finance,dc=fabrikam,dc=com. In addition to that, however, you should set the search scope to ADS_SCOPE_BASE; that instructs the script to search *only* the target container (the Finance OU).

ignore any sub-containers (that is, any sub-OUs). Here's a complete script that searches just the Finance OU:

```
Const ADS_SCOPE_BASE = 0

Set objConnection = CreateObject("ADODB.Connection")
Set objCommand = CreateObject("ADODB.Command")
objConnection.Provider = "ADsDSOObject"
objConnection.Open "Active Directory Provider"
Set objCommand.ActiveConnection = objConnection

objCommand.Properties("Page Size") = 1000
objCommand.Properties("Searchscope") = ADS_SCOPE_BASE

objCommand.CommandText = _
    "SELECT Name FROM 'LDAP://ou=Finance,dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user'"
Set objRecordSet = objCommand.Execute

objRecordSet.MoveFirst
Do Until objRecordSet.EOF
    wscript.Echo objRecordSet.Fields("Name").Value
    objRecordSet.MoveNext
Loop
```

Scripting Guys Tip. If you decide to do a targeted search, keep in mind that neither the Users container nor the container is an OU; to specify either of those locations as the starting point for a search you need to write a query this, one that uses **cn** in the ADsPath instead of **ou**:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user'"
```

[↑ Top of page](#)

Specifying an Optional WHERE Clause

You're right: saying that the WHERE clause is optional is a bit like saying water is optional. Although you can survive – without water, it isn't long before this "optional" accessory becomes a requirement. The same thing is true of the clause in Active Directory. For example, here's a perfectly valid SQL statement, one that returns the name of all the Active Directory:

```
"SELECT ADsPath FROM 'LDAP://dc=fabrikam,dc=com'"
```

See? No WHERE clause, and we have a perfectly valid SQL query. Optional.

Of course, it won't be long before you tire of getting back a list of every single object in Active Directory, especially wanted was a list of the users in the Finance Department or the printers capable of printing in color. Pretty soon you create more efficient and better-targeted searches; that's the moment the WHERE clause becomes a necessity. So should spend a few minutes walking through WHERE clauses and how they are used.

[↑ Top of page](#)

Searching Object Classes and Object Categories

Because we talked about object classes and object categories in [Part 1 of this series](#) we won't spend any time discussing these concepts here. Suffice to say that you'll probably want to include one or the other (most likely an object category) in your Active Directory search scripts. After all, including classes or categories in your search will enable you to do so return just the printers that belong to the Finance department, not the printers and the users and the contacts and

computers and.... (Hint: See Part 1 if you have no idea what we're talking about.)

So how *do* you return just the published printers found in the domain? Use a query like this one:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='printQueue'"
```

All we've done here is specify that the objectCategory – that is, the kind of objects to be returned – should be restricted to printQueue objects, the Active Directory name for published printers. Here are some of the more common object categories you might find yourself searching for:

- computer
- contact
- group
- inetOrgPerson
- organizationalUnit
- printQueue (printers published in Active Directory)
- site
- siteLink
- subnet
- user
- volume (shared folders published in Active Directory)

[↑ Top of page](#)

Using AND and OR in Queries

To be honest, there's no reason that you *have* to search Active Directory. For example, suppose you wanted to get information about a user with the sAMAccountName kenmyer. You don't have to search Active Directory; instead you could run a recursive script that methodically retrieves all the user names from all the containers in Active Directory and checks each account to see if the sAMAccountName attribute happens to be kenmyer. This would be extremely slow and extremely cumbersome, but it would work.

But who wants to be extremely slow and extremely cumbersome? (We didn't say we *weren't* slow and cumbersome didn't want to be.) We want to be extremely fast and extremely efficient. And one good way to do that is to write queries when searching Active Directory. For example, do you want a list of all the users who have the sAMAccountName kenmyer? Then use this code:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
    & "objectCategory='user' AND sAMAccountName = 'kenmyer'"
```

Notice what we've done here: we've specified two separate criteria in our WHERE clause: we want a list of objects where objectCategory is equal to user *and* where the sAMAccountName is equal to kenmyer. For data to be returned both criteria must be true; the object must be a user *and* it must have the sAMAccountName kenmyer. That's why we use the WHERE clause. What if we wanted a list of all the users in the Finance department who happen to be accountants? Our query might look like this:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
```

```
& "objectCategory='user' AND department = 'Finance' AND title = 'Accountant'"
```

See how that works? We could continue to string additional items onto this clause, making sure we separate each item with AND.

Of course, sometimes we want to be a little more inclusive. For example, suppose we want to return a list of users *either* the Accounts Payable department or the Accounts Receivable department? We can't use AND in this case; after all, AND all the conditions have to be true, meaning our user would need to belong to both Accounts Payable and Accounts Receivable. Instead we want data returned if *either* condition is true. Thus we use OR instead:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
& "department='Accounts Payable' OR department='Accounts Receivable'"
```

With this query we'll get information back from every object that belongs to either Accounts Payable *or* Accounts Receivable.

Why did we say "every object" instead of "every user?" Well, to keep this first example as simple as possible we removed the **objectCategory='user'** portion of the query. We'll add that back in now, but be sure to take a careful look at how the WHERE clause is constructed; after all, this time we're using both AND *and* OR:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
& "objectCategory='user' AND " _
& "(department='Accounts Payable' OR department='Accounts Receivable')"
```

Hopefully you noticed that we used parentheses to keep the two conditions separate: we only want objects that are members of either the Accounts Payable or Accounts Receivable departments. If you're using both AND and OR in the WHERE clause parentheses will help guard against unexpected results. For example, if we remove the parentheses from the query is difficult to decipher:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
& "objectCategory='user' AND " _
& "department='Accounts Payable' OR department='Accounts Receivable'"
```

After all, this query *can* be read like this: select all the users who are members of the Accounts Payable department *and* everything (not just users) from the Accounts Receivable department. That's probably not what we want, and the best guard against misinterpretation (either on our part or on the part of the computer running the script) is to use parentheses.

Here's another example: this returns either users *or* computers that belong to either the Accounts Payable or the Accounts Receivable departments. Again, note the use of parentheses:

```
"SELECT Name FROM 'LDAP://cn=Users,dc=fabrikam,dc=com' WHERE " _
& "(objectCategory='user' OR objectCategory='computer') AND " _
& "(department='Accounts Payable' OR department='Accounts Receivable')"
```

[↑ Top of page](#)

Specifying Values in a WHERE Clause

Any time you write a WHERE clause you need to know – at a minimum – two things: the name of the attribute (prepending the data type of the attribute). Active Directory attributes come in different flavors: some hold string values, some hold numeric values, some hold Boolean values. You need to know the data type for an attribute because the data type determines the syntax of your WHERE clause.

What do we mean by that? Take a look at a simple WHERE clause:

```
WHERE objectCategory='user' AND department='Finance'
```

Notice that the specified values for `objectCategory` ('user') and `department` ('Finance') are surrounded by single quotes because `objectCategory` and `department` are string attributes. Any time you're working with string attributes the space must be enclosed in single quote marks. Want to return a list of all the users with the surname (SN) Smith? Then your `WHERE` clause will look like this:

```
WHERE objectCategory='user' AND SN='Smith'
```

This is *not* the case when dealing with attributes that have either numeric or Boolean data types. For example, suppose you want to search for all the global groups in a domain. Global groups have a `groupType` value of 2; thus your `WHERE` clause will look like this:

```
WHERE objectCategory='group' AND groupType=2
```

See? No single quotes around the value 2. The same thing is true for attributes with a Boolean data type. Need a list of all the printers? Any printer capable of printing in color will have a value of `TRUE` for its `printColor` attribute:

```
WHERE objectCategory='printQueue' AND printColor=TRUE
```

Again, note the absence of single quote marks around the value `TRUE`. Single quotes are used only for string data.

True confession. OK, so we're being a tad bit untruthful here: things can get a little messier when dealing with bitmask attributes. Because this is an introductory article, however, we're simply pretending that some of these other data types don't exist. We'll deal with those other data types in a future article.

[↑ Top of page](#)

Using Variables in a WHERE Clause

We had a feeling someone was going to ask that. All our examples use hard-coded values like this, with the department explicitly included in the script:

```
WHERE objectCategory='user' AND department='Finance'
```

That's useful for educational purposes but, as we know, when does education ever have anything to do with real life? The answer is create a script that can search for only one thing: users in the Finance department. In real life you'd probably want a script that could search for users in *any* department, a department you might specify as a command-line argument. Let's go on to talk about command-line arguments today; if you'd like to know more about those see this [Tales from the column](#). Let's just assume that, one way or another, you managed to store the department you're searching for in a variable named `strDepartment`. Now, how do you use that variable in a `WHERE` clause?

No, sorry, this *won't* work:

```
WHERE objectCategory='user' AND department='strDepartment'
```

Why? Well, in this case the script will search for a department with the *name* `strDepartment`, a script that probably

anything. This won't work either:

```
WHERE objectCategory='user' AND department=strDepartment
```

How come? Remember, department has a string data type; thus it's looking for a string value – something surrounded by quotes. No single quotes here; thus you get back this cryptic error message:

```
Provider: One or more errors occurred during processing of command.
```

Yes, remarkably helpful, isn't it? What it comes down to, though, is the fact that the script has no idea what **department=strDepartment** means. And so it just plain gives up.

We won't keep you in suspense any longer; here's a WHERE clause that *will* work:

```
WHERE objectCategory='user' AND department='" & strDepartment & "'
```

Believe it or not this actually makes sense. We expect a string value to be passed to department, and we expect that to begin with a single quote mark. And that's exactly what we have:

```
WHERE objectCategory='user' AND department='
```

If we were hard-coding a value we'd tack that value on right after this single quote. However, we *aren't* hard-coding; we're using a variable. Therefore we temporarily terminate the hard-coded text; that's what the double quote mark then use the ampersand to add the value of the variable strDepartment to the clause. When the script gets processed (assuming strDepartment equals Finance) the computer will read our WHERE clause like this:

```
WHERE objectCategory='user' AND department='Finance
```

Why? Because it has substituted in the actual value **Finance** for the variable strDepartment. We then simply need a closing single quote mark and we're done; that's what we do with the second ampersand, the second double quote mark means, "OK, now that we've substituted in the variable let's pick up with the hard-coded text again"), and the second quote.

Don't worry too much if you didn't follow the logic; you can get a slightly-more detailed explanation [here](#). Just follow what's shown above and you'll be fine.

If you're working with Boolean values or numeric values, do the exact same thing, but leave out the single quotes.

```
WHERE objectCategory='printQueue' AND printColor=" & blnColor & "
```

[↑ Top of page](#)

Using Operators in WHERE Clauses

So far we've only shown you SQL queries that used the equals operator in a WHERE clause; for example:

```
WHERE objectCategory='user' AND department='Finance'
```

Very handy if you want to get back a list of all the users in the Finance department. But what if you wanted to get the users who *aren't* in the Finance department. Can you write a query that doesn't use the equals operator?

You're right: we wouldn't have asked the question if we didn't have an answer for you. Sure you can; in fact, WHERE use any of these operators:

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Want a list of all the users *not* in the Finance department? Then just use a query that says "Show me all the users whose department attribute is not equal to Finance." The WHERE clause for that query will look a lot like this:

```
WHERE objectCategory='user' AND department<>'Finance'
```

[↑ Top of page](#)

Using Wildcards in a WHERE Clause

If you're thinking, "Man, if they show me any more things about querying Active Directory I'm going to *explode!*", you might want to stop reading at this point. But, listen, there's no reason to explode: using wildcards in a WHERE clause is the last thing we talk about today.

Yes, we know: we have covered a lot of ground in this one column. Fortunately this section will go fast, because the only one wildcard character to deal with: the asterisk (*), which simply stands for "anything." For example, suppose you want to search for a user who has the last name – well, to be honest, you don't really *remember* the last name. It was Smith or Smithson, something like that. What do you do in that case?

Well, in that case you use a wildcard character to search for any user whose SN begins with the letters *SM*. Here's a query that returns all the users whose SN begins with the letters *SM*:

```
WHERE objectCategory='user' AND SN='Sm*'
```

Note the position of the wildcard character. If we wanted to limit the returned data to Smith, Smithson, or anything with *SMITH* we'd use this query:

```
WHERE objectCategory='user' AND SN='Smith*'
```

And that's really all there is to it. You can also use the wildcard at the beginning of the string. This query returns Sr Johnson, Thompson, and any other user whose last name ends in *SON*:

```
WHERE objectCategory='user' AND SN='*son'
```

You could even stick the wildcard character in the middle of the string; however, that can seriously degrade performance on a domain controller so we don't recommend it.

The wildcard character is particularly useful when you're just trying to find out if something exists. For example, this query returns a list of all users who have a telephone number, regardless of what that phone number might be:

```
WHERE objectCategory='user' AND telephoneNumber='*'
```

A search like this – which simply specifies the * – can be very handy, especially if you're auditing Active Directory for a list of all the users who don't have a department or a title or something. For example, this query returns a collection of users without a job title:

```
WHERE objectCategory='user' AND title<>''
```

Note that this time we used the <> (not equal to) operator.

See, that wasn't so bad, was it?

[↑ Top of page](#)

So Now We Know Everything About Searching Active Directory, Right?

Well, not exactly. You now have enough information to get started and to carry out most searches; however, that's not all we've covered everything. We haven't talked about searching bitmasked attributes or multi-valued attributes or doing recursive searches or – well, you get the idea. If you're interested in the additional topics write to us at scripter@microsoft.com. If there's enough interest we'll see what we can do. Until then, and to paraphrase Lloyd Alexander, sometimes you can learn more by not learning everything than you can by learning everything.

OK, so we don't know what that means, either. But we had to have *some* kind of catchy slogan with which to end this column and that was the best we could do. Hey, it's not our fault: if Lloyd Alexander would say a few more clever things than we would, we wouldn't be forced to make these slogans up ourselves!

[↑ Top of page](#)

[Manage Your Profile](#)

© 2008 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#) | [Contact Us](#)

Microsoft