SQL-Retriever 4

The Connection Methods

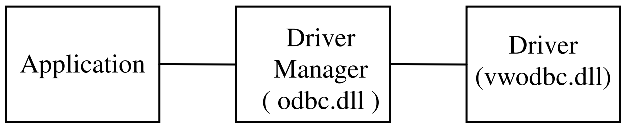
Before we discuss the connection methods of SQL-Retriever 4 we must first take a look at the methods used by version 3.2, these been:

There are three methods used by the old product:

1) RPC / TCP
2) Rlogin / TCP
3) Serial line

Lets looks at RPC /TCP

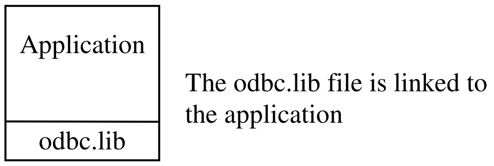
In order to form a connection the application has to issue either an SQLConnect or an SQLDriverConnect (ODBC API calls), before these the application has to call SQLAllocenv (which allocates an environment handle) then it calls SQLAllocConnect (allocates a connection handle). These two functions allocate storage (structures) for connections and statements, it then calls the SQLConnect (allows you to connect to an existing datasource) or SQLDriverConnect (allows you specify a complete datasource on the fly).



How does the application contact the driver manager

There are two ways:

1) It either links statically to an import library (odbc.lib), static references are linked into the import library then at run time the dynamic linker links odbc.dll and the entry points are mapped into it. This is done so you do not have to load the dll itself, it saves you having to resolve symbols and finally the link has a very small memory footprint



2) The other way, is to load the library directly and resolve functions symbols into addresses, these functions could then be called by address. However, this would have to be done for each function and is therefore not a valid solution.

In applications such as Visual Basic you can declare a function prototype in a dll , VB will then load the dll that you reference and resolve the symbols in to an addresses.

What happens if the application cannot find the driver manager ?

Well it will report:

'Cannot find ODBC.DLL'

This will be because:

a) it did not find ODBC.DLL in the Windows\system directory
b) it has run out of memory or
c) it is corrupt.

After these functions have been called the driver manager will then look at ODBC.INI for the datasource as specified in the SQLDriverConnect function.
(All of the functions above have parameters (handles) that have been missed off , these are not really relevant to this discussion and would confuse this discussion).
The Driver manager then looks for the 'Driver =' line and uses a load library call to load our driver (vwodbc.dll). In the driver manager there is an entry point (symbol) for every ODBC function available, there is also the same set of entry points in our dll.
The load library allows the dll just to be loaded onto memory, the driver manager says to Microsoft Windows (via SDK) look up for me 'symbol SQLConnect' this then returns the address of SQLConnect in vwodbc.dll in memory, the driver manager links this to a name, after which the actual function is called by address and therefore there is never a symbol clash.
When the driver manager tries to resolve a symbol not exported in the driver it returns 'function not supported from the wrapper function.
The driver manager therefore is responsible for loading the driver on behalf of the application, it's responsible for handling requests, it's responsible for primitive parameter checking (it checks if it is in the right context). It is also contains support for functions that cannot be supported by the driver, such as SQLDataSources because only the driver manager can tell what datasources are available and SQLFunctions, again only the driver manager knows what functions are available, as well as checking individual function exist.

There is another possible cause for problems, our driver may not be found:

Problems:

1) File not there
2) Memory
3) file corrupt
4) Driver = line is wrong in ODBC.INI.

The error that is returned is:

[Microsoft][ODBC.DLL]Cannot load specified driver.

If all goes well the driver manger then loads up VWCOMMS, which in turn loads up :

NMTCPRPC.DLL
NLWINSOCK.DLL (NMTRMRPC.DLL for Rlogin)

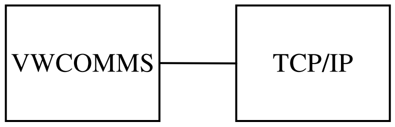
WINSOCK.DLL (If using the Windows Socket 1.1 interface)

Now VWCOMMS.DLL and NMTCPRPC.DLL swap place and the driver makes calls directly to the NMTCPRPC.DLL and this holds the connection open.

If we cannot find any of the above you will receive something like:

Cannot load library 'DLL name'

Let's say for the use of this discussion that all the above DLL's are the 'VWCOMMS'.



When the application calls SQLConnect in the driver, the driver looks into the ODBC.INI file and searches for the host name, database name and perhaps the username and password (Oracle), initially only the hostname is extracted then more information is extracted when needed. When the RDBMS is extracted it then looks at the vwodbc.ini for the RPC number for the database that has been specified (391436 for INFORMIX, originally supplied by SUN for us to use).

What are Remote Procedure Calls (RPC's) ?

Let's look firstly at what you may understand. You have no doubt written a Local Procedure Call (LPC) when you have written a program:

```
Main()
{
        Function();
}

/* Then further down the listing you would have the code that implements the function */

Function()
{
*********
}
```

An RPC is really the same, the only difference is that the code for the function itself is on the remote machine and then returns the value to the PC.
This fits into our model quite nicely, we have the equal of main() in our driver (vwodbc.dll) and the function module thus underneath could be the equivalent of how we want to do the work on the database. This two ends are tied together using two factors; an RPC number (391436) and an RPC version number (2), these together uniquely identify what we call a CLIENT and a SERVER in RPC's.

How do these get written ?

Well you have a definitions file (a .x file), we took the source code for SUN's RPC and ported it to all the UNIX platforms we support and also to the Windows environment. We also added a security module called **vwauth** (this gives you the Invalid User Authorisation Specification) because when we looked a SUN's

RPCs we found they were very insecure, we therefore redeveloped the security and made it UNIX security aware. Once the RPC's have been ported, in essence there are two parts to it; the RPC development kit and the RPC generator which is called RPCGEN (libvwrpc.a on UNIX), you would then write the definitions file, this says I want this particular function with this argument of this type (it'ss a function prototyping file) you then run the function prototyping file through RPCGEN (called vwrpcgen so it supported Windows), it then writes two files, a client and a server which are run on the PC and UNIX machines, by using the RPC number we can then talk to each side.

To Create a connection the ODBC Driver calls 'create client' in the VWCOMMS, for this to work we need three things:

1) The IP address
2) RPC number
3) RPC Version number.

However at the UNIX side something is happening, we need something called the PORTMAPPER (/etc/portmap or rpcbind) which helps identify the program we want to talk to down to one, you need an IP address and a port number to identify a specific program. Therefore in order to talk to our server we need to know which port it is listening on, how's this done ?
Well we could have a 'well known port' in /etc/services, but at the time of writing SQLR we did not want to place a entry in the services file. We therefore had to contact the portmapper which does have an well known port, (Port 111 used for RPC's), this now means we can send a request (connectionless UDP datagram, it does not guarantee things will arrive and in the correct order, but it does guarantee the contents will be correct) to the portmapper on port 111. Another protocol we can use is TCP, this is like telephoning someone but this has to physically form the connection beforehand therefore we do not use this later protocol (as portmapper does not support it)
However before we get this far the Portmapper itself has to know what these numbers are, we start the portmapper as ROOT and it binds to port 111 and the listens on that port (it registers itself with itself), you will therefore always get an entry in the mapping table for portmapper if 'rpcinfo -p' is used. Seconds later the SQL-Retriever demon is started, it forms a TCP style connection and then uses 'bind to any port' where by it binds to the next available port (say 1024) it then presents the RPC number, version number and service name to the portmapper, and also tells the portmapper it is listening on port 1024.

We can now send a UDP to the portmapper and ask say 'for this RPC number and the version number what port do I contact ?'. The portmapper then returns port '1024'. If we do not get a reply after a specific time we assume the UDP got lost, if this happens the original request is sent again, this is done six times then if the final UDP is lost we present the following to the user:

```
Cfstat 12 reerrno 3 'No such process , cannot contact portmapper'
```

(You would then look for the portmapper running on the host box with the PS command)

The driver now knows the port number of the server, it therefore makes a TCP connection to that IP address and that particular port number (a telephone call).

Note: There are two things that can go wrong at this point:

1) The portmapper may not know about the server program, you then see 'Server program not registered with portmapper'

2) The server could be running but may have tried to register with the portmapper while the portmapper was registering with itself (called the quiescent stage). You should place a sleep between starting the portmapper and our server help resolve this issue (on fast machines this may not be necessary). Your startup script would look something like:

```
/etc/portmap &
sleep 2
nohup /etc/visionware/sqlr.inf32.d &
```

It now passes the username and password for UNIX and the server program uses a C function called ‘Getspnam’ to verify the password. SQL-Retriever does part of this checking and so does the C function..

What happens while this is going on ?

First it opens the /etc/passwd file and looks for the username and find the passwrd entry, it then encrypts the password given and compares it with that already in the file and sees if they are the same. However, if the Shadow password is in use it opens the file in /etc/shadow and compares that. The reason why /etc/shadow is used is because only ROOT can read and write this file whereas the /etc/password file can be read by normal users. If this fails then you will get the IUAS error message.

We then check to see if the user has a working directory, again if this fails you receive the IUAS message. After that we look for the initial program in the password file, we trust /bin/csh/, /bin/sh and /bin/ksh but they have to be in the exact locations. If you have the file /etc/shells then no programs are trusted unless they are entered into this file.

You can also get the IUAS error message if the server program is not running as ROOT as it cannot access the /etc/shadow file.

When the server got a ‘select’ on port 1024 it accepted the request and then formed another socket and forked (a child process) itself so two copies of the server are running at the same time, the child now inherits all that was owned by its parent and the accepted socket is copied onto the child. The server again goes on listening on the original socket but the child is now working on the new one. Finally the child process sets its UID to the user who requested the initial conversation, again without the server program running as ROOT the setuid could not be done. Now the server uses ESQLC to connect to the database and run a copy of the engine (depending which database you run).
The problem we have now is that the engine / engine interface that is run needs environment variables, and because we go over RPC none of the users login scripts are read (.cshrc, .login) we therefore have a configuration file called xxxsqld.conf (depending on the database used the x’s are replaced with the first three characters of the RDBMS name, such as inf for Informix) file, the environment variables are put into the environment from this file.

Once we have got the database via ESQLC you will start getting database errors if anything goes wrong.

The connection method Rlogin over TCP.

The connection method is the same as RPC until we get to ‘create client’, as the port number is known for Rlogin it uses a TCP connection over the wire on port 512 (rlogind), the username and password are sent and then rlogind returns the login program back (we need a shell, and vwauth is not used via Rlogin because rlogind performs this).
We have a problem with Rlogin as it is a terminal source or teletype connection, we therefore have to set ‘RAW’ mode and this is done using input / output controls. We therefore needed to write a program that has a configuration file that contains the RPC number, the version and program server name to run and set raw mode. VWCOMMS now types :

‘vwportmap - prog, <rpc number>, - ver <version number>

the stdin and stdout which were connected to the shell are now inherited by the vwportmap program This now has a look in the configuraton file to see what it should be running and sets the terminal to a transparent connection (raw) and the program is exec'd (sqlr.inf32.d). The only problem now is that you will get a 'cannot register with portmapper' error message, we therefore need a flag to tell our server 'Do not register with the portmapper, do not use any socket operations, purely use stdin / stdout for communication', this flag is called -vwTerm. The connection is then made to database via ESQLC.

The connection method Serial line

To make things simple lets imagine you do not need PC-Connect

The communications port is opened on the PC and the UNIX getty prompts for the username and password, we supply these and a shell comes back to the PC. The vwportmap -vwTerm is issued and the connection is formed. However, a lot more can go wrong there may be data corruption due to noise on the wire, we also need to have more than one connection down the wire. Thus the reason we require PC-Connect to provide the multichannel error correcting protocol.

Stored Procedures

Stored procedures in databsases are like functions in a programming language, you may have a function like:

C = ReturnValue(a,b)

In a and b you can have two numbers and the added value is returned into C, previously before SQL-Retriever 4 we could not return C. With SQL-R 4 we can now return C and we also support outputs:

ReturnValue(a,b,c)

The letter (c) in the function call above would be the output. We therefore now have FULL stored procedure support with SQL-Retriever 4

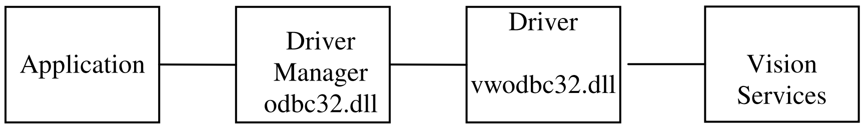
SQL-Retriever 4

Two versions are going to be written, a 16 bit for Windows 3.11 and 32 bit one for Windows 95 & Windows NT, these will now support the new Vision communications. The connection method used in the 16 bit (SQLR 4) version will work the same as version 3.5 (RPC over TCP).

NOTE: The 16 bit version will work in the 32 bit environments.

These new PC side components talk to a new server module which is also backwards compatible with 3.5.

The new 32 bit SQLR will use the new connection methods.



NOTE: As an aside there is something called DBSERV, this is a interface for accessing the communications database and it serialises requests to use the communications database.

Before we describe what happens on the PC further we need to look at the server end.

USP over TCP connection

There are now several processes running on the server:

1) Local Name Agent (LNA) this is like a portmapper and an inetd process combined.
2) In /usr/local/etc there is a file called servers, it contains:

```
# ( Hashes are comments but be aware there are special comments )
#BOOT_START ( this is a special entry )

# Any items between these two special comments gets started when the LNA boots, entries outside these
#special comments get started on demand.

#BOOT_END
```

The lines in the file have the following format:

<Service name> <program, <M / S>,<flags>

The service name is the name of the service, such as sqlrinf, the program is the name of the program that provides this service and the flags are any flags that you wish to place on the program. The third field has (M) Multi-threaded or (S) single threaded, the M stands for a server that can start other servers and the S stands for a server that cannot start other server and you would talk to a single instance of it. SQL-Retriever would have an 'M' in this field. There would be one entry for each database you have on the system and they must be in between the special entries in the servers file.

NOTE: for internal use only, the server can be also started from the command line.

LNA has to be in a boot file, therefore there is no longer the need for INETD or RC2 for the server itself.

The LNA has on it '-b' (BOOT), LNA goes through the services file at boot time and starts the program in between the commented lines in the service file. Therefore the first thing that should be run for our software is 'LNA -b' , it then starts the name space manager (NSM), this acts like an NIS server, therefore it resolves nodenames and services into IP addresses and port numbers. It uses the universal naming (UNC) conventions (\\hostname\service) to resolve the IP address and port.

On your network there will be only one master NSM, this manages all the name spaces, other secondary NSM's will elect to become the master if the present master dies. When you install SQLR on the server you will be prompted to elect this machine as the master NSM, you can say yes or no. On one of the hosts you would say yes to this question and no on any other machines you install SQLR on.

The LNA also starts VWAUTH, we will look at this later. The last thing that is started (optional) is licsrv, the license server. There is another process called logsrv for debugging

Therefore for trouble shooting you should look for the following processes running:

LNA

NSM
SQLR.xxx40.d

NOTE: If you are using our 16 bit server it must be in the boot section of the service file, as there is no other way to start the server other than by this method.

When the server is run what happens ?

The server is now in /usr/local/vision/bin/sqlr.xxx40.d (and the config files are in vision/etc), when run it creates a tcpip connection and binds to anyport (perhaps 1024), it still has an RPC number, version number, service name and a port number. But it now does the following:

a) It registers with the portmapper (for backwards compatibility with 3.2 / 3.5 PC sides).
b) Tells the LNA its IP address, port number and service information. The LNA then tells the NSM this information (NSM may be running on another machine thus this process is required).

If anything breaks at this point the error is logged into the on-going log file produced by logsrv.

When our ODBC driver is loaded into memory the Vision Services are also loaded and they do a broadcast on port 16024 to try to find the NSM on a host machine (the NSM has already bound to port 16024 and is listening). The listening master NSM then returns it's IP address saying 'I am the NSM'

NOTE: Port 16024 is not well known to others, therefore other programs maybe wanting to use this port, if this is the case then our software can be configured to use another port by editing the Vision services on the PC (from the control panel).

The above process is only carried out once (providing the naming request has not failed).

The reply from the NSM is stored in the communication database and the registration file.

The application usually calls the following functions at connect time:

SQLAllocenv()
SQLAllocConnect()
SQLDriverConnect()

Now the driver manager goes to the registration file (with SQLR 3.2 / 3.5 it would have gone to the ODBC.INI) to find out the 'driver =' line.

NOTE: In the 32 bit ODBC world you have user and system datasources., the difference is that system datasources can be seen by everyone whoever logs onto the Windows machine, if it is a user datasource then only that user can see it. These are placed into different areas of the registry, for a system DSN then it is placed into the HKEYLOCAL_MACHINE (This predefined handle is the root of the per machine configuration data, regardless of what user is logged on to the system) part of the registry (Reg diagram 1)

All user DSN's are located in the HKEY_CURRENT_USER (This predefined handle is the root of the currently logged on user's profile and contains all of the information necessary to set up a specific user's environment. This includes such items as, program groups, screen colors, etc) and has a similar format to the system entries. You will not see an entry for the ODBC.INI as this has to be seen by all users.

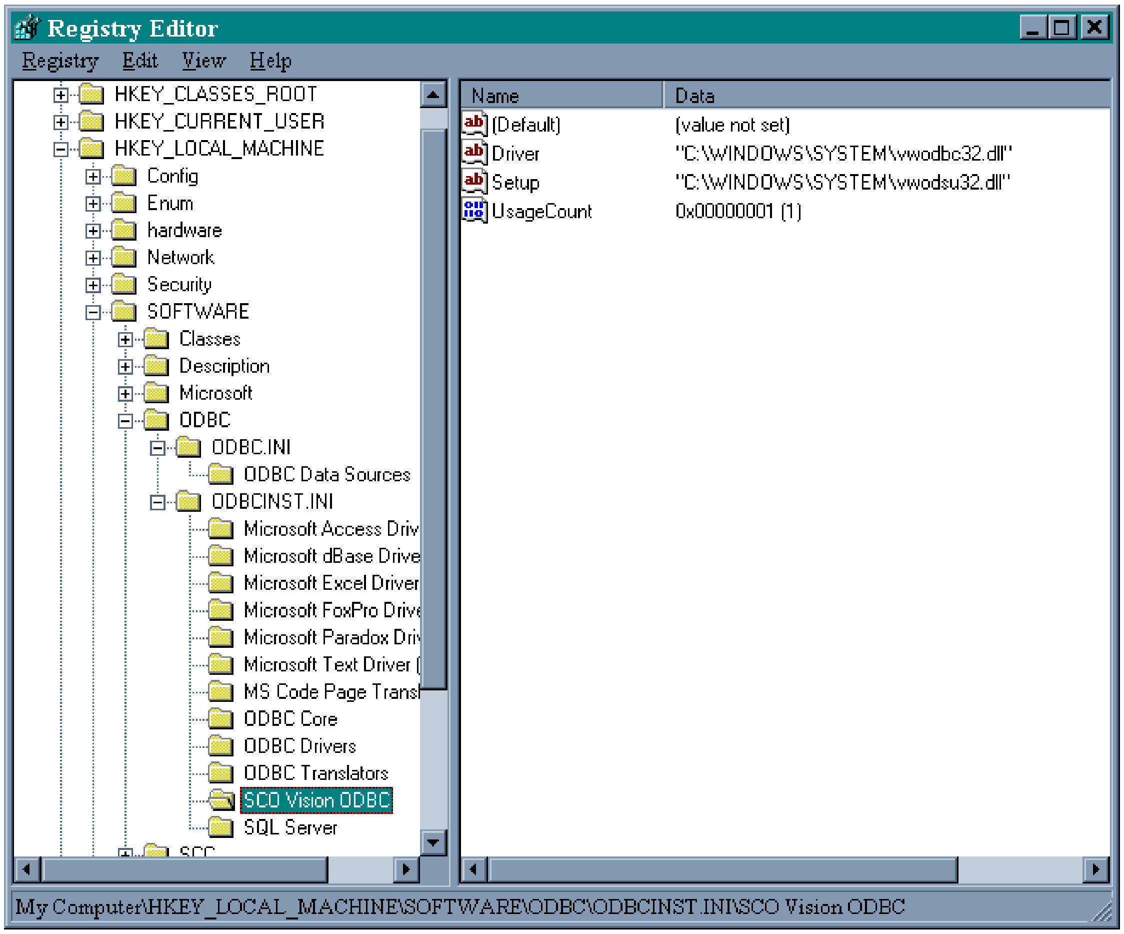
Under the HKEY_LOCAL_MACHINE - Software - SCO - VWODBC.INI you will see all the RPC numbers etc as in the old VWODBC.INI file.

Once the driver has been found it gets loaded into memory and then looks for the RDBMS you are using ans then extracts service name from the registry (for Informix this would be sqlrinf). In now knows the nodename and the service name therefore it goes to the NSM and asks were it can talk to such a service.

Problems that can be encountered at this stage :

If the NSM cannot supply this information then the NSM may have gone down, we would then say 'Who is the NSM ?' . If there were no secondary NSM's running thus there is now no NSM's running you would receive the error message 'No NSM manager'. You would then ask the customer where their NSM is supposed to be running and start it. Another reason for it to fail is the customer may not have run 'LNA -b' or run the server (sqlr.inf40.d) on the UNIX machine.

If all is well the NSM returns the IP address and port number where the service can be found.



Name	Data
(Default)	(value not set)
Driver	"C:\WINDOWS\SYSTEM\vwodbc32.dll"
Setup	"C:\WINDOWS\SYSTEM\vwodsu32.dll"
UsageCount	0x00000001 (1)

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\SCO Vision ODBC

Reg diagram 1

The next thing that happens is the client forms a TCP connection to the IP address and port number that was returned by the NSM, which is our server running in ROOT mode. Our server then accepts the connection on the bound port (1024). The server then forks copying the accepted socket to the child, and continues to listen on the original

The above connection description is USP (universal session protocol) over TCP, the net thing that happens is authorisation (checking of the username and password)

Authorisation

This is done differently from SQLR 3.2, as our server now only forks children (However when used with SQLR 3.2 it still does the authorisation) . We say to the NSM , ‘Where is \\Poohbear\authserv ?’, this returns the IP address and port number of the authserver the client then connects to authserv and send it username, password and service name to verify (Note: No /etc/shells cheet now !) The authorisation server now sends back a token (a pass) if the user details are correct and if they fail you will get the IUAS error message. Once the PC has the token the connection is closed down to the authorisation server.

NOTE: With SQLR 4 the password is fully encrypted over the network.

The PC now presents the root server its token together with the username (sent for correctly setting the UID for the child server process , the password is also sent but I not used), this set of credentials is also sent with every RPC call making sure each is secure. The child now sets it’s usersid to the connecting user, the requested database is then connected using ESQLC.

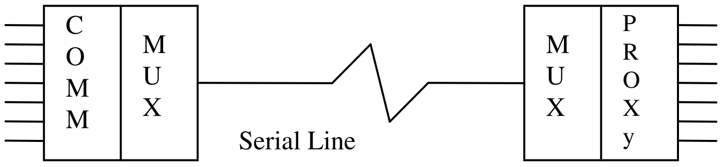
The server is now talking to the PC.

USP over Serial Line connection method

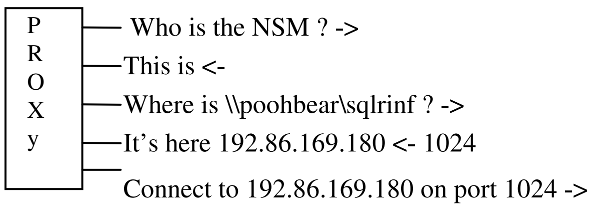
We do not need the old PC-Connect host module, however you do need TCP on the UNIX machine (explained below) !

The number of connections are now only limited by speed and bandwidth.

The PC opens the communication port and the UNIX getty sends the prompt back to the PC for the uername and PC, we send these and a shell is sent back we then run a proxy (something that does something on your behalf) ‘usptcppxy’ (The proxy has to run on the machine with the serial line attached but the database and SQLR can be running on another machine, but they have to be connected via a network)



Lots of connections are multiplexed down one serial line and then open up on the UNIX side, the proxy device then copies what the TCP did in terms of asking ‘who is the NSM’ (see diagram below)


PROXy
Who is the NSM ? ->
This is <-
Where is \\poohbear\sqlrinf ? ->
It’s here 192.86.169.180 <- 1024
Connect to 192.86.169.180 on port 1024 ->

The big drawback with this type of method is that the proxy does what the PC did when using a USP over TCP connection by sending out a network request for who is the NSM. By doing this you must have a NETWORK because the NSM listens on a network as does LNA, authserv etc.

You cannot simply get round this by just loading the TCP stack on the UNIX machine as when the kernal boots it will try to initialise the network adapter. If you get a network adapter you will need to create a network with at least a T piece and some terminators. Therefore you will have to configure TCP/IP on you UNIX machine when you will not even use it.

The only way round this is by writing you own TCP/IP kernal interface !! NOT.

The Security Manager

The security manager uses a configuration file which is located in /usr/local/vision/etc/sqlrsecure.conf

The manager is built into the server and gives us user / application security.

It is enabled by :

‘sqlr.xxx40.d -r’

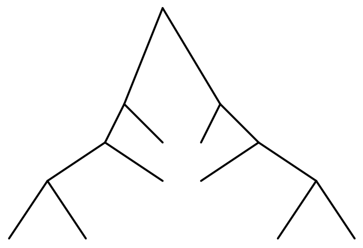
NOTE: If the server is running with the -r flag when using the ‘ps’ command it means it has been enabled with the security manager. By default when this has been enabled no one can do anything to the database.

If the .conf file above is updated with new entries after the server has been started, the ROOT server has to be restarted. This is because the ROOT server only reads the file into memory and the child parses the file for the security permissions etc. This has the good effect that the .conf file is now only readable by ROOT, therefore adding to the security.

The server can also read in user defined configuration files, these can be specified as an argument after the ‘-r’ flag has been used. An example of which is :

sqlr.xxx40.d -r <userdefined security file>

How does the security manager work ?



The Grammar Parser Tree

It uses the parse tree created by the grammar translator . This is used to translate one syntax into another . We examine the leafs on the tree to find out what components of SQL are been used. These are then checked against the security manager file, for example we can find out what operation is been done on what table.

If the security broker is enabled the parse tree is created even if the ‘Pass RDBMS grammar’ is chosen, (however this is not translated).

If the credentials are not passed you will get a security violation.

The format of the security manager file is :

Usergroup	<Groupname>	=	<Item, item >
For Example:			
	Support	=	mac, mattsc, allang, markst
DBGROUP	<groupname>	=	<item, item >
TABLEGROUP	<groupname>	=	<item, item >
APPGROUP	<groupname>	=	<item, item >
SQLGROUP	<groupname>	=	<item, item >
Example	readonly	=	select
Groups	<groupname>	=	<item, item > (Only usergroups can be grouped in here)

NOTE: You cannot have groups within groups (except for the one above) , for example :

	Marketing	=	mac, support

were support is a group, is not valid

Comments can be inserted using the ‘#’ sign.

What can you do with all these entries ?

Well you can have Grant or Revoke the permissions:

user database qualifier owner table app = privledges

You can use the ‘*’ or ‘ALL’ to mean everything.

Therefore to give everybody permission to alter anything in the database you would have:

Grant * : * : * : * : * : * = ALL

NOTE: You must have the permission already on the database to be able to use these permission, if you do not then you will receive an error back for the database.

Let’s say we want the following user:

The user mac when using SQLGold with any database, any owner, any qualifier and using sales can only select. The table format would be:

Grant mac ALL ALL * sales SQLGold = select

Anywhere you can use and item you can use a groupname.

Enhancements to SQLR 4:

Access uses parameter markers and putdatas, we now cache these putdatas and then submit them in one RPC call (we used to send each one down the line separately), thus this speeds up MS Access considerably.