HOW TO USE A DESKTOP VERSION OF A DBMS FOR CLIENT-SERVER APPLICATIONS

Julian Vasilev

Abstract: DBMS (Data base management systems) still have a very high price for small and middle enterprises in Bulgaria. Desktop versions are free but they cannot function in multi-user environment. We will try to make an application server which will make a Desktop version of a DBMS open to many users. Thus, this approach will be appropriate for client-server applications. The author of the article gives a concise observation of the problem and a possible way of solution.

Keywords: Database management systems (DBMS), Information technology, parallel processing, Cache, client-server applications, application server, sockets.

ACM Classification Keywords: H.2.8 Database Applications, H.4 information systems applications.

Introduction

Single user versions of some DBMS are also called Desktop versions. They are usually free for commercial, home and office use. We can give Intersystem's Cache as an example [1]. The license fee for the use of the multi-user version of this DBMS is 245 EUR per process excluding VAT (Value Added Tax). If we calculate it with VAT then price is 294 EUR. If a company buys accounting software for 200 EUR and wants to use it as a client-server application for 4 computers, it has to pay 200 EUR for the software and 1176 EUR for license fees just for the right to use the multi-user version of the DBMS. This fact obstructs many small and middle enterprises in buying software products. That is why we have a possible solution. We will build an application server which will receive queries from workstations and redirect them to a single-user database. After receiving the answer from the database it will be redirected to the appropriate workstation. In this way, end users cannot feel that they use a single-user database. Moreover there is no need in changing the existing software, for instance the accounting software.

Background of the problem

Form a technological point of view this idea can be realized in Delphi, C#, Visual Basic. Moreover, there is a version of Delphi for Linux, called "Kylix". In this way the future application server can be compiled for another operating system. The program implementation consists of two parts: a server application and a client application. According to Cantu [2, 749] the idea can be realized by using the communication interface DCOM.

"DCOM is directly available in Windows NT/2000 and 98/Me, and it requires no additional run-time applications on the server. You still have to install it on Windows 95 machines. DCOM is basically an extension of COM technology that allows a client application to use server objects that exist and execute on a separate computer. The DCOM infrastructure allows you to use stateless COM objects, available in the COM+ and in the older MTS (Microsoft Transaction Server) architectures. Both COM+ and MTS provide features such as security, component management, and database transactions, and are available in Windows NT/2000 and in Windows 98/Me. Due to the complexity of DCOM configuration and of its problems in passing through firewalls, even Microsoft is abandoning DCOM in favour of SOAP-based solutions."

We made several experiments and few basic problems occurred. First, computers with different version of operating systems (for instance Windows 98 and Windows XP) cannot communicate. Second, DCOM does not keep alive several connections. Third, there is a limit in the number of connections. That is why our research continues. We want to find a better solution. Let us examine the work of a multi-user database (fig. 1).

The server part is usually installed on a computer, named "server" and the client part of the DBMS – on workstations. When we use a different DBMS the installation process is similar.

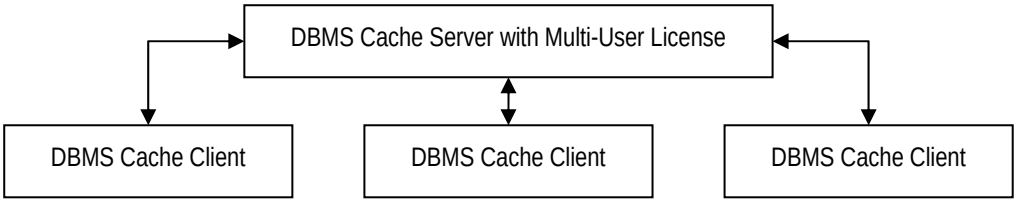

Figure 1: Technology of using a multi-user DBMS

A possible solution

We have to use another information technology to solve this problem. The communication can be realized using Transmission Control Protocol/Internet Protocol (TCP/IP for short). In a local area network (LAN) each computer has a unique IP Address. Connections between computers can be implemented by TCP ports. Each TCP connection takes place though a port. Some TCP ports have a standard usage for specific high-level protocols and services. In other words, you should use those port numbers when implementing those services and stay away from them in any other case. Here is a short list (table 1).

Table 1 Ports for some protocols

Protocol	Port
HTTP (Hypertext Transfer Protocol)	80
FTP (File Transfer Protocol)	21
SMTP (Simple Mail Transfer Protocol)	25
POP3 (Post Office Protocol, version 3)	110
Telnet	23

HTTP, SMTP and FTP are standard protocols. If we want a custom communisation between server and workstations we have to define custom protocol. A set of communication rules is generally indicated as a protocol. Basically, the server can receive different requests and, depending on the type of request and whether it can be accomplished, replies to the client. The server will respond to many requests. Transfer protocols are at a higher level than transmission protocols. That is why protocols are independent not only from the operating system and the hardware but also from the physical network. Communication can be started only if we launch a server program which accepts client connections. The client requests a connection indicating the server it wishes to connect to. When the client sends the request, the server can accept the connection, starting a specific server side socket, which connects to the client-side socket.

Methodology of implementation

Delphi 5 ships with three sets of socket components. Newer versions of Delphi also support Socket components. They can be used to read and write information over a TCP/IP connection. The Internet page of the palette hosts the Client Socket and Server Socket components. Sending text to server can be done by issuing method "SendText".

*ClientSocket.Socket.SendText('Select * From Customers Where CustNo = 1394')*

In this way we can send to the server a SQL (structured query language) statement. The server will receive the sent message as simple text. The Server Sockets reads the text by calling the method "Client Read". The text is actually contained in the property "Receive Text".

SQL_to_execute := ServerSocket.Socket.ReceiveText;

The server can use blocking or non-blocking connections. When the server uses blocking connections requests are processed in sequence. Huge information systems cannot scale by blocking connections. One of the possible solutions is the use of non-blocking connections. If we build a large system a good idea is to use threads to communicate with the database.

For the starting of the server we have to do the following:

ServerSocket.Port := 1974;

ServerSocket.Active := True;

On the client side, to connect to the server we have to connect to the server:

```
ClientSocket.Port := 1974;
ClientSocket.Host := '192.168.23.117'; // This is the IP address of the server
ClientSocket.Active := True;
```

As we mentioned we send a SQL statement to the server.

ClientSocket.Socket.SendText(SQL_to_execute);

The server receives request. This event fires the method OnClientRead of the Server Socket.

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
  i:integer;
  st : string;
begin
  for i := 0 to ServerSocket.Socket.ActiveConnections-1 do
  begin
    with ServerSocket..Socket.Connections[i] do
    begin
      st := ReceiveText;
      if st <> '' then
      begin
        Memo1.Lines.Add(RemoteAddress + ' sends :') ;
        Memo1.Lines.Add(str);
      end;
    end;
  end;
end;
```

In this way received text is added in a Memo.

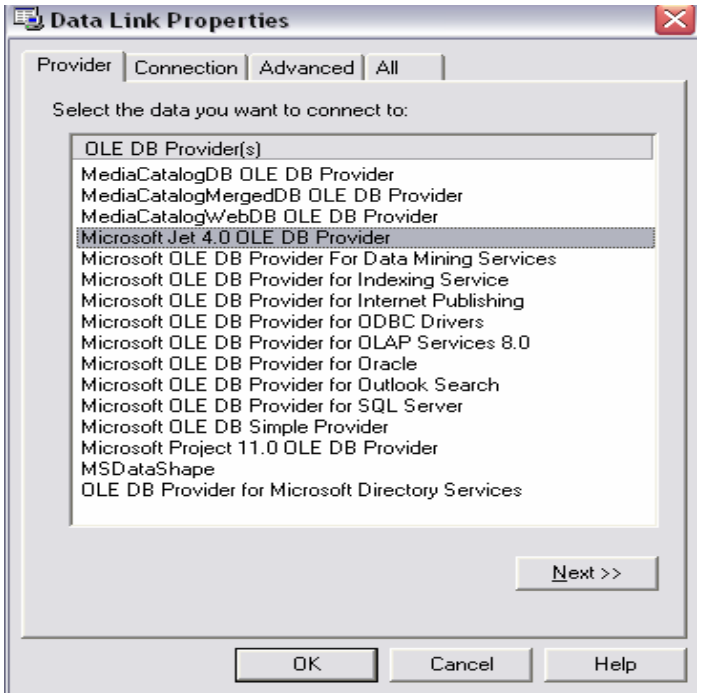

Figure 2 Provider for Data Link Properties

We have to redirect it to a database. We can use ADOConnection and a Windows UDL file to access different types of databases (for instance MS Access, Oracle, MS SQL Server, Informix, Sybase, Cache, DB2 and others).

To make an UDL file we just make a simple text file and we change its extension from "txt" to "udl". We open the file. On the "Provider" tab we set the type of DBMS we want to use (fig. 2).

The marked choice is for MS Access. If we use Oracle, we have to choose Microsoft OLE DB Provider for Oracle. If we use MS SQL Server, we have to choose Microsoft OLE DB Provider for MS SQL Server. In the next step we choose the database, username and password for login and we can test the connection. In this way our server application is DBMS independent. Moreover, the connection to the database is initialized through a text file which looks like an "ini" file. It is a stand-alone file and can be modified without the need of compilation.

As we highlighted the server receives requests. They are redirected to a DBMS using "ADOConnection" and "ADOQuery". The result of the execution of a query is in the form of DataSet. This dataset is two-dimensional. It consists of rows and columns. To be send back to the server it has to be represented as a simple string. That is why we have to use 2 delimiters: one - for rows and another one – for columns. They can be "tab character" – ASCII code "9" for field delimiter and "line feed and carriage return" – ASCII codes "10", followed by "13" – for record delimiter (table 2).

Table 2 Simple dataset returned as a result of execution a query

Order_number	Order_date	Cust_code
12345	03.04.2007	1395
12336	04.04.2007	1391

This tabular data will be transformed in one string as follows:

Result_string := '12345'+#9+'03.04.207'+#9+'1395'+#10#13+'12346'+#9+'06.04.207'+#9+'1391';

Actually, the result string is formed by using two "for" cycles. The sample code is too simple. That is why we skipped it. The next step is sending the result dataset to the client.

ServerSocket.Socket.Connections[nConnection].SendText(Result_string);

The variable "nConnection" indicates the unique number of connection. The client socket receives the result dataset. The Client Socket fires the event "OnRead". To read the incoming message from the server we have to write the following:

Received_text := socket.ReceiveText;

The next step is to convert the string into a two-dimensional array in order to visualize the dataset in a tabular format. This operation is simple. That is why we go on. A corner-stone can be the size of the string send over socket connection. A possible solution is to send the result dataset in several parts. OLE fields are another corner-stone. If we have to send a file or multimedia fields or BLOBs (binary large objects) we can use streams over socket connections.

Software development in brief

To realize the idea of using a desktop DBMS in a local area network (multi-user environment) we need to do the following. Firstly, we have to use the technology of communication by using the built-in port in Windows. For instance Trojan horses and some DBMS (Cache uses port 1972) use them for communication. Likewise, web servers use this port to communicate with end users. The default port is 80. This fact determines the use of sockets. We can use Client Sockets and a Server Socket. Secondly, we need an application server which will stay one level above the DBMS and will dispatch queries. Its kernel is based upon a Server Socket. Thirdly, a small application is needed on workstations with built-in Client Socket. Fourthly, we need a standard for communication between end-users' workstation and the application server. An example of such organization of work is described in fig. 3.

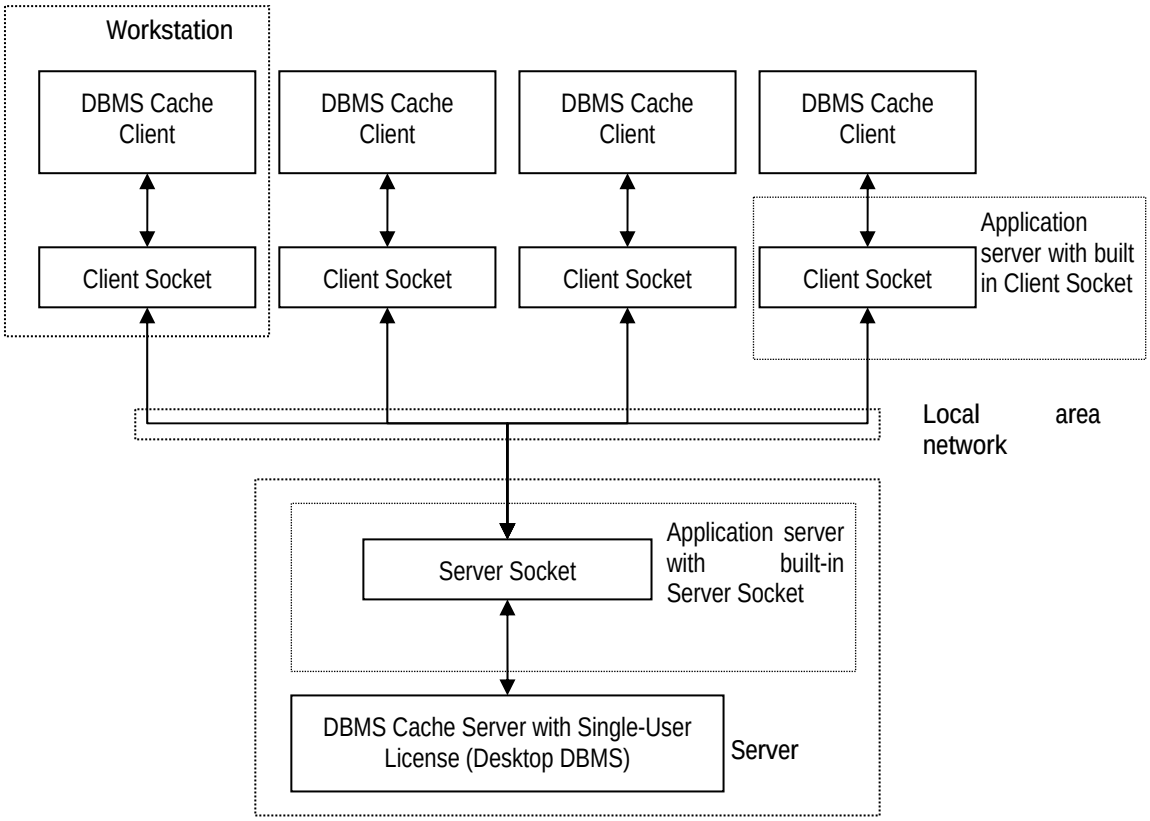

Figure 3: Technology of using a single-user DBMS in multi-user environment

In another aspect this application server can be used as a dispatching server for cluster applications. This server can have several subordinate sub-servers which process users' queries. In this way we get better parallel processing of queries. The result is similar to that in search engines such as google.com, yahoo.com, live.com, ask.co.uk.

Conclusions

As the reader sees, we succeeded in using a single user DBMS in multi-user environment. We gave a methodology of implementation of this idea. It was realized in a software product installed in 2005 at the Varna University of Economics – graphic user interface software for testing students' knowledge. The system is used in 7 disciplines. The achieved positive results are obvious evidence for the positive aspects of issued concepts. Moreover, software companies in Bulgaria can easily adapt this idea and make their products accessible to many small and middle enterprises.

Bibliography

[1] www.e-dbms.com.

[2] Cantu, M. Mastering Delphi 6, Sybex, Alameda, CA, 2001.

Authors' Information

Julian Vasilev – Chief assistant professor, Department of Informatics, Varna University of Economics; 77, Kniaz Boris I str.; Varna; Bulgaria; e-mail: vasilev@ue-varna.bg