Client/Server Applications with Visual FoxPro and SQL Server

Chuck Urwiler, Gary DeWitt, Mike Levy & Leslie Koorhan

Edited by Chaim Caron

Hentzenwerke Publishing
Client/Server Applications
with
Visual FoxPro and SQL Server

Chuck Urwiler
Gary DeWitt
Mike Levy
Leslie Koorhan

Hentzenwerke Publishing
To my wife, Michelle, for her patience and support.
— Chuck Urwiler

I dedicate this endeavor to Joan and Lou DeWitt.
— Gary DeWitt

To my wife, Heather, and children, Jacob and Megan, for having patience.
— Michael Levy

This is for my wife Sybille, who has put up with the most awful mess that a home office can be, with the excuse that “once I finish this, then I’ll clean it up.” And I did.
— Leslie Koorhan
Our Contract with You, 
The Reader

In which we, the folks who make up Hentzenwerke Publishing, describe what you, the reader, can expect from this book and from us.

Hi there!

I’ve been writing professionally (in other words, eventually getting a paycheck for my scribbles) since 1974, and writing about software development since 1992. As an author, I’ve worked with a half-dozen different publishers, and corresponded with thousands of readers over the years. As a software developer and all-around geek, I’ve also acquired a library of more than 100 computer and software-related books.

Thus, when I donned the publisher’s cap four years ago to produce the 1997 Developer’s Guide, I had some pretty good ideas of what I liked (and didn’t like) from publishers, what readers liked and didn’t like, and what I, as a reader, liked and didn’t like.

Now, with our new titles for the spring and summer of 2000, we’re entering our third season. (For those keeping track, the ’97 DevGuide was our first, albeit abbreviated, season, and the batch of six “Essentials” for Visual FoxPro 6.0 in 1999 was our second.)

John Wooden, the famed UCLA basketball coach, had posited that teams aren’t consistent—they’re always getting better—or worse. We’d like to get better… One of my goals for this season is to build a closer relationship with you, the reader.

In order to do this, you’ve got to know what you should expect from us.

• You have the right to expect that your order will be processed quickly and correctly, and that your book will be delivered to you in new condition.

• You have the right to expect that the content of your book is technically accurate and up to date, that the explanations are clear, and that the layout is easy to read and follow without a lot of fluff or nonsense.

• You have the right to expect access to source code, errata, FAQs, and other information that’s relevant to the book via our Web site.

• You have the right to expect an electronic version of your printed book (in compiled HTML Help format) to be available via our Web site.

• You have the right to expect that, if you report errors to us, your report will be responded to promptly, and that the appropriate notice will be included in the errata and/or FAQs for the book.

Naturally, there are some limits that we bump up against. There are humans involved, and they make mistakes. A book of 500 pages contains, on average, 150,000 words and several megabytes of source code. It’s not possible to edit and re-edit multiple times to catch every last
misspelling and typo, nor is it possible to test the source code on every permutation of
development environment and operating system—and still price the book affordably.

Once printed, bindings break, ink gets smeared, signatures get missed during binding. On
the delivery side, Web sites go down, packages get lost in the mail.

Nonetheless, we’ll make our best effort to correct these problems—once you let us know
about them.

And, thus, in return, when you have a question or run into a problem, we ask that you first
consult the errata and/or FAQs for your book on our Web site. If you don’t find the answer
there, please e-mail us at books@hentzenwerke.com with as much information and detail as
possible, including (1) the steps to reproduce the problem, (2) what happened, and (3) what you
expected to happen, together with (4) any other relevant information.

I’d like to stress that we need you to communicate questions and problems clearly.
For example…

• “Your downloads don’t work” isn’t enough information for us to help you.
  “I get a 404 error when I click on the Download Source Code link on
  www.hentzenwerke.com/book/downloads.html” is something we can help you with.

• “The code in Chapter 10 caused an error” again isn’t enough information. “I
  performed the following steps to run the source code program DisplayTest.PRG in
  Chapter 10, and received an error that said ‘Variable m.liCounter not found’” is
  something we can help you with.

We’ll do our best to get back to you within a couple of days either with an answer, or at
least an acknowledgment that we’ve received your inquiry and that we’re working on it.

On behalf of the authors, technical editors, copy editors, layout artists, graphical artists,
indexers, and all the other folks who have worked to put this book in your hands, I’d like to
thank you for purchasing this book, and hope that it will prove to be a valuable addition to your
technical library. Please let us know what you think about this book—we’re looking forward to
hearing from you.

As Groucho Marx once observed, “Outside of a dog, a book is a man’s best friend. Inside
of a dog, it’s too dark to read.”

Whil Hentzen
Hentzenwerke Publishing
August, 2000
Acknowledgements

First of all, I’d like to thank my wonderful wife Michelle for dealing with my extended working hours while I finished this book. As you all probably know, working a full-time job and then working on a book on top of it doesn’t leave room for a lot of “quality time” together. She’s been very supportive of me from the day I met her, and especially during the time I worked on this book. There were no complaints, only support. Thank you, Michelle... I love you.

Next I’d like to thank Whil for getting me involved with this book. We had talked in the past about doing some kind of book, so this was a great way for me to whet my appetite, since I didn’t have to write the whole thing <g>.

Which brings me to my third group of thank yous: Without Gary, Mike and Leslie, I would have had a lot more late nights of writing. I thank them for their contributions to the book. And Chaim did a great job of making sure my writing was technically complete as well as clear. I’d like to thank everyone at Micro Endeavors, both past and present. Without you, I would not have been able to learn so much about VFP and SQL Server. Thanks, everyone! The drinks are on me!

I must also thank my family and friends, who are all very excited that I’m finally going to have my name on the cover of a book. Thanks for your support and encouragement over the years as I tried to find my place in the world.

And finally, I’d like to thank all of “my” students, as well as those of you who support me at the DevCons and user groups where I’ve spoken. Without your questions, curiosity, feedback and willingness to share what you know with me, I wouldn’t be able to write this stuff.

— Chuck

I would like to thank Bonnie Berent and Kim Cameron for personal support and encouragement; Tamar Granor and Whil Hentzen for supporting and encouraging my writing; Chaim Caron for his insights and ideas; The Academy for...

— Gary

I don’t think I shall soon forget my experiences while participating in the development of this book. It has given me a better appreciation for those who give up family and free time to share their knowledge and experiences with the rest of us.

To Heather, thanks for your support, patience and willingness to proofread the early drafts, even though you did not understand the topics being discussed.

To Whil Hentzen, thanks for inviting me to participate in this project; to Gary DeWitt, thanks for heading up the project and taking on the bulk of the work; to my good friend Matt Tiemeyer, thanks for taking the time to discuss ideas and preview drafts; and to another good
friend, Leslie Koorhan (before you came to participate in the project), my thanks for providing ideas and the many long discussions.

—Mike

First of all, I thank Whil Hentzen for giving me the opportunity to make my contribution to this book, and for pushing. There’s someone else who I’d also like to thank for this chance, but Whil never told me who brought my name up in the first place. Second, there’s Mike Levy, who offered me a peek at his contributions long before I wrote one word. Mike has been a real friend as well, before and during this process. Third, there’s Gary DeWitt (I know these are other authors, but hey, they deserve all the praise that I can muster), who helped me shape my chapters. He gave me a lot of direction when I was just trying to formulate my thoughts. And after all, it was his outline of the book that started me. Fourth is Yair Alan Griver, who started me years ago with his client/server articles, and started me writing with his encouragement and inspiration. Finally, I also thank Dan Freeman, Paul Bienick, Ken Levy and Markus Egger.

—Leslie
About the Authors

Chuck Urwiler

Chuck Urwiler is a Senior Instructor, Consultant and Developer for Micro Endeavors, Inc., a Microsoft Solution Provider Partner and Certified Technical Education Center in Upper Darby, PA. He is a highly respected instructor who has trained thousands of developers nationwide in all versions of Visual FoxPro and in version 7 of SQL Server. He has authored numerous seminars and hands-on courses for Micro Endeavors, including introductions to the new features of SQL Server 7 and Visual FoxPro 6, Migrating FoxPro Applications to Visual FoxPro, and most recently, Mastering Microsoft Visual FoxPro Development. Chuck was also a technical contributor and editor for the Microsoft certification exams in Visual FoxPro. Currently, he holds several certifications: a Microsoft Certified Professional (MCP) in Visual FoxPro and SQL Server 7.0, a Microsoft Certified Trainer (MCT) and a Certified Technical Trainer (CTT), as well as being one of the first 1,000 Microsoft Certified Solution Developers (MCSD) in the world.

Not content to spend all of his time in the classroom, Chuck keeps his knowledge current by working with the Micro Endeavors development team on a variety of projects, many of which include both Visual FoxPro and SQL Server. He is a consultant for Micro Endeavors clients on their software projects, using his expertise and experience to solve business and technical problems with both SQL Server and Visual FoxPro.

Chuck can be reached at chuck@microendeavors.com.

Gary DeWitt

Gary DeWitt has been a frequent contributor to the FoxTalk and FoxPro Advisor journals for several years. Gary is the author of Client/Server Applications with Visual FoxPro and the technical editor of Internet Applications With Visual FoxPro 6.0, from Hentzenwerke Publishing, and has spoken at regional and national FoxPro conferences including Visual FoxPro DevCon. He is a Microsoft Certified Professional and a past Microsoft Most Valuable Professional. In addition to Visual FoxPro, Gary also works with C++, Java and Visual Basic, and has been in the computing industry in one way or another since 1976. Gary is currently senior software magician at Sunpro, Inc., the leader in fire service software, where he leads a team responsible for a large client/server COM application compatible with the National Fire Incident Reporting System.

Gary can be reached at gdewitt@garydewitt.com.

Michael Levy

Michael Levy is a consultant with G.A. Sullivan, where he specializes in SQL Server and database technologies. He is a Microsoft Certified Solution Developer (MCSD), Database Administrator (MCDBA) and Trainer (MCT). As an MCT, more than 400 students in 60 classes have benefited from his 10 years of FoxPro and four years of SQL Server experience. Mike is a well-known member of the Visual FoxPro community and donates his time helping others on various Internet newsgroups. In addition, he has spoken at multiple conferences and user groups and is a frequent contributor to various technical journals. Mike is a University of
Cincinnati graduate and lives with his wife, two kids, eight fish and an old dog in a house that’s still waiting for Mike to discover “landscaping.” Mike can be reached at ma_levy@hotmail.com.

**Leslie Koorhan**

Leslie Koorhan is an independent consultant and trainer who specializes in database applications. He has worked with nearly every version of FoxPro and Visual FoxPro as well as Microsoft SQL Server since the mid 1990s. He is also a Visual Basic developer. He has written numerous articles for several publications, most recently a series on Microsoft SQL Server OLAP Services for *FoxTalk*. Leslie has also written several training manuals.

Leslie can be reached at lkoorhan@earthlink.net.

**Chaim Caron**

Chaim Caron is the President of Access Computer Systems, Inc., in New York City, which, despite its name, specializes in software development using primarily Visual FoxPro. The firm also provides data integrity testing, data cleansing services, and various other services relating to data design and software development and support.

Chaim has specialized in FoxPro and Visual FoxPro since 1990. His articles have appeared in *FoxTalk* and *FoxPro Advisor*. He has spoken to technical and industry groups on technical and business issues for many years.

He spends his free time with his wife and daughter. His third love (after his wife and daughter) is the mandolin, which he plays with the New York Mandolin Quartet. Major mandolin influences were provided by Barry Mitterhoff, Carlo Aonzo, Ricky Skaggs, John Monteleone and, of course, Bill Monroe.

Chaim can be reached at ccaron@earthlink.net.
How to Download the Files

There are two sets of files that accompany this book. The first is the source code referenced throughout the text, and the second is the e-book version of this book—the compiled HTML Help (.CHM) file. Here’s how to get them.

Both the source code and the CHM file are available for download from the Hentzenwerke Web site. In order to obtain them, follow these instructions:


2. Look for the link that says “Download Source Code & .CHM Files.” (The text for this link may change over time—if it does, look for a link that references Books or Downloads.)

3. A page describing the download process will appear. This page has two sections:
   - **Section 1**: If you were issued a username/password from Hentzenwerke Publishing, you can enter them into this page.
   - **Section 2**: If you did not receive a username/password from Hentzenwerke Publishing, don’t worry! Just enter your e-mail alias and look for the question about your book. Note that you’ll need your book when you answer the question.

4. A page that lists the hyperlinks for the appropriate downloads will appear.

Note that the .CHM file is covered by the same copyright laws as the printed book. Reproduction and/or distribution of the .CHM file is against the law.

If you have questions or problems, the fastest way to get a response is to e-mail us at books@hentzenwerke.com.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction to Client/Server</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Visual FoxPro for Client/Server Development</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Introduction to SQL Server 7.0</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Remote Views</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>Upsizing: Moving from File-Server to Client/Server</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>Extending Remote Views with SQL Pass Through</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>Downsizing</td>
<td>125</td>
</tr>
<tr>
<td>8</td>
<td>Errors and Debugging</td>
<td>145</td>
</tr>
<tr>
<td>9</td>
<td>Some Design Issues for C/S Systems</td>
<td>159</td>
</tr>
<tr>
<td>10</td>
<td>Application Distribution and Managing Updates</td>
<td>177</td>
</tr>
<tr>
<td>11</td>
<td>Transactions</td>
<td>193</td>
</tr>
<tr>
<td>12</td>
<td>ActiveX Data Objects</td>
<td>209</td>
</tr>
<tr>
<td>A</td>
<td>Appendix A: New Features of SQL Server 2000</td>
<td>225</td>
</tr>
</tbody>
</table>
Table of Contents

Our Contract with You, The Reader v
Acknowledgements vii
About the Authors ix
How to Download the Files xi

Chapter 1: Introduction to Client/Server 1

<table>
<thead>
<tr>
<th>The PC revolution</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client/server to the rescue</td>
<td>2</td>
</tr>
<tr>
<td>Features of client/server databases</td>
<td>3</td>
</tr>
<tr>
<td>Data access</td>
<td>3</td>
</tr>
<tr>
<td>Security</td>
<td>4</td>
</tr>
<tr>
<td>Database backup</td>
<td>6</td>
</tr>
<tr>
<td>Point-in-time recovery</td>
<td>6</td>
</tr>
<tr>
<td>Triggers</td>
<td>7</td>
</tr>
<tr>
<td>Referential integrity</td>
<td>8</td>
</tr>
<tr>
<td>Indexes</td>
<td>8</td>
</tr>
<tr>
<td>Defaults</td>
<td>10</td>
</tr>
<tr>
<td>Rules</td>
<td>10</td>
</tr>
<tr>
<td>Primary key generation</td>
<td>10</td>
</tr>
<tr>
<td>Stored procedures</td>
<td>11</td>
</tr>
<tr>
<td>Views</td>
<td>12</td>
</tr>
<tr>
<td>User-defined data types</td>
<td>13</td>
</tr>
<tr>
<td>Replication</td>
<td>14</td>
</tr>
<tr>
<td>Transactions</td>
<td>14</td>
</tr>
<tr>
<td>Scalability</td>
<td>14</td>
</tr>
<tr>
<td>Reliability</td>
<td>15</td>
</tr>
<tr>
<td>Advantages of client/server</td>
<td>15</td>
</tr>
<tr>
<td>Performance</td>
<td>16</td>
</tr>
<tr>
<td>Cost</td>
<td>16</td>
</tr>
<tr>
<td>Security</td>
<td>16</td>
</tr>
<tr>
<td>Scalability</td>
<td>17</td>
</tr>
<tr>
<td>Summary</td>
<td>17</td>
</tr>
</tbody>
</table>

Chapter 2: Visual FoxPro for Client/Server Development 19

| Object-oriented programming (OOP) | 19 |
| Support for COM | 21 |
| Built-in client/server support | 23 |
| Built-in local data engine | 23 |
Support for other data-access technologies 24
Rapid Application Development (RAD) 25
Summary 25

Chapter 3: Introduction to SQL Server 7.0 27
Why move to SQL Server? 27
  Capacity 27
  Concurrency 28
  Robustness 28
  Security 28
Installation 29
  SQL Server editions 29
  Licensing 30
  Character sets 31
  Sort order 31
  Network libraries 31
Databases, database files and the transaction log 32
  Types of databases 32
  Database files 33
  Creating a database 33
  The transaction log 36
  How SQL Server allocates storage 36
Transactions and locking 37
  Implicit and explicit transactions 37
  Locking 37
Database objects 39
  SQL Server object names 39
  Tables 40
  Enforcing data integrity 41
  Indexes 46
  Views 48
  Stored procedures 49
  Triggers 52
Summary 55

Chapter 4: Remote Views 57
Connections 57
Remote views 63
Updatable views 65
Buffering 68
Committing and refreshing buffers 69
Other view properties 70
  FetchAsNeeded and FetchSize 70
  MaxRecords 71
Remote views vs. SQL pass through 118
  SQL pass through 118
  Remote views 119
Using remote views and SPT together 122
  Transactions 122
  Stored procedures 122
  Filter conditions 123
  Summary 123

Chapter 7: Downsizing 125
  The case for a single code base 125
  Interchangeable back ends 125
    Remote views of VFP data 126
    Substituting local views for remote views 128
    Abstracting data access functionality 130
Microsoft Data Engine (MSDE) 136
  What is MSDE? 136
  MSDE vs. SQL Server 137
  Distributing MSDE applications 141
  Migrating MSDE databases to SQL Server 142
  Summary 144

Chapter 8: Errors and Debugging 145
  Handling errors 145
    Trapping errors 145
    Reporting errors 146
    Conflict resolution 150
  View errors 151
  Debugging tools 152
    SQL Server Profiler 152
    The SQL Server Performance Monitor 155
    ODBC logs 156
  Summary 158

Chapter 9: Some Design Issues for C/S Systems 159
  SQL database design issues 159
    Data integrity mechanisms 160
    VFP developer vs. SQL Server DBA 168
  Client/server performance issues 169
    Choosing indexes 169
    Client/server division of work 171
    Data location 173
  Security 173
    Client application 173
### Chapter 10: Application Distribution and Managing Updates

- Client/server development
  - Development environment
- Deployment models
  - Traditional
  - Components
  - Server
- Distributing databases (creating)
  - Existence of SQL Server
  - SQL pass through
  - SQL scripts
  - SQL-DMO
  - Object transfer (DTS)
  - Backup/restore
  - `sp_Detach_DB` and `sp_Attach_DB`
- Managing updates
  - Application changes
  - Database updates
  - Version control coordination between client and server
- Local lookup data
  - Why
  - Managing updates
- Summary

### Chapter 11: Transactions

- Transaction basics
  - ACID properties
- Visual FoxPro transactions
  - The missing property
- SQL Server transactions
  - Implicit transactions
  - SQL Server isolation levels
  - Durable transactions
- Locking
  - Lock compatibility
  - Blocking
  - Viewing lock activity
  - Deadlocks
- Transaction gotcha!
- Summary
# Chapter 12: ActiveX Data Objects

<table>
<thead>
<tr>
<th>Why ADO?</th>
<th>209</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO benefits</td>
<td>209</td>
</tr>
<tr>
<td>ADO disadvantages</td>
<td>211</td>
</tr>
<tr>
<td>Installing and distributing ADO</td>
<td>211</td>
</tr>
<tr>
<td>Using ADO within Visual FoxPro</td>
<td>212</td>
</tr>
<tr>
<td>The Connection object</td>
<td>212</td>
</tr>
<tr>
<td>The RecordSet object</td>
<td>214</td>
</tr>
<tr>
<td>The Command object</td>
<td>221</td>
</tr>
<tr>
<td>Summary</td>
<td>224</td>
</tr>
</tbody>
</table>

## Appendix A: New Features of SQL Server 2000

| Feature list | 225 |
| Installation issues | 227 |
| Query Analyzer | 227 |
| Debugging stored procedures | 228 |
| User-defined functions | 229 |
| Referential integrity | 231 |
| Trigger enhancements | 232 |
| Indexing computed columns | 233 |
| New data types | 234 |
| Big integers | 234 |
| Variants | 234 |
| Tables as variables | 235 |
| Summary | 236 |
Chapter 1
Introduction to Client/Server

Client/server applications differ from file-server applications in many ways, but the key difference is that client/server applications divide the processing between two or more applications, a client and a server, which typically run on separate computers. In this chapter, you will learn a little of the history of client/server computing, as well as the features of client/server databases in general and Microsoft SQL Server in particular. You will also learn some of the advantages of client/server databases over file-server databases.

In the beginning, there were mainframes and minicomputers. All data resided on and was processed by these often room-filling machines. All bowed down before the mighty MIS department, as all information was in their hands.

Okay, so that might be an exaggeration, but not by much. In the late 1970s, there were plenty of data processing centers with raised floors, a sea of disk drives, a wall of tape drives and an army of operators. In this host-based model, all the processing was done by the mainframe, while data was entered on dumb terminals. Consider the example of Gary’s first programming project, in 1979: “The project involved trying to get data from our Data General minicomputer for a decision support system for our product managers. We wanted to give our incredibly powerful Apple IIIs, with two floppy disk drives, access to the corporate sales data so the product managers could make decisions on our product lines. We were completely at the mercy of the MIS department, who controlled all the data. No way were they going to let us have online access to it. They could be coerced into giving us a monthly dump of sales data, but that was the best we could do. Our project turned into a primitive data warehouse; modeling and reporting on the data turned out to be easy compared to getting access to it in the first place.”

Lest you think that this is merely a history lesson, there are many of these systems still in place. The most popular American hospital billing system uses an IBM AS/400 minicomputer-based DB2 database. Many systems like this are unlikely to be replaced in the near future, but to developers they are mostly a curiosity, as new development on such systems is rare, and the growth of the mainframe market is pretty flat.

The PC revolution

Then came the personal computer. PCs allowed departments and often entire corporations to dispense with their expensive, centralized host-based systems and replace them with networks of PCs sharing files on file servers. The pendulum had swung the opposite way. Rather than doing all of the processing on the central scrutinizer, it was done on the workstation. Dumb terminals had been replaced by dumb file servers.

This is where most of us come in. FoxPro, along with other systems like dBase, Paradox and Access, has a local data engine. All processing is done on the workstation, and the network is used for file storage only. Throughout this book, this model is referred to as a file-server database.
Because all the processing was performed locally and because the workstation could be a powerful computer in its own right, we developers were able to give users very sophisticated user interfaces. But we could not provide them with a secure, fault-tolerant database, and we used up a tremendous amount of network bandwidth. In fact, application server software such as Citrix WinFrame or Windows Terminal Server, which reduce network bandwidth by running applications on a server machine, became popular primarily because of file-server databases and their need for a big network pipeline. This is because all processing is performed on the local workstation, while only the files reside on a file server. To perform a query, all information necessary for finding the result set, such as index keys, must be downloaded in addition to the result set itself. Rushmore is very efficient about what it brings down, but whatever it needs still has to come to the local workstation.

Furthermore, improvements in database performance often require upgrades to each workstation running the application—a potentially expensive proposition when many users are involved.

Client/server to the rescue

The client/server database is an excellent solution to the problem of delivering sophisticated applications while maintaining security and fault-tolerance of the database and reducing network overhead. Client/server databases are so named because a system consists of at least two applications: a client application and a server application, or service. The client application typically runs on a workstation and can provide the sophisticated user interface that users have come to expect. To interact with the database, the client sends requests to the server. The server application typically runs as a service on an application server machine and manages the database and responds to requests from clients.

The concept of a client making a request and the server responding to that request is key to understanding client/server computing. The client has absolutely no control over the data on the server. It makes a request for everything. The client doesn’t open data files; it asks the server to log the user in to the database. This is the key to client/server security because there is no backdoor access to data. The client doesn’t download index keys to perform a query; it simply sends a SELECT statement to the server, and the server sends back only the matching records. This is the key to reducing network overhead, as only a minimum of network traffic is required.

In a file-server application, performing a query requires many round trips to the server. For example, opening a table requires sending low-level file access instructions to the server and returning a handle to the file. Then the workstation sends instructions to access the file addresses of the index keys, and the server returns those keys. The workstation then processes the keys to determine the result set and sends to the server the addresses of the records to retrieve. Finally, the data itself is downloaded. Use your network monitor and a modem connection sometime to perform a simple query against FoxPro tables, and you’ll get an excellent demonstration of just how slow and inefficient this process can be. The larger the database and the lower the network bandwidth, the worse this performance is.

In a client/server application, the client merely sends a SQL statement off to the server:

```
SELECT * FROM employees WHERE lastname LIKE 'King'
```
The server responds by sending back only the records that match. Not only has the quantity of transmitted data been reduced, but the number of network round trips has, too.

The problem of improving file-server performance is also partially resolved by client/server applications because database performance can be improved by upgrading a single machine, the server, rather than upgrading all the workstations. It is considerably less expensive to upgrade or replace a single, powerful application server than many lower-level workstations!

There are many client/server databases on the market today. Originally many of them, such as Oracle, Informix and Sybase, ran only on Unix. Several years ago, Microsoft and Sybase entered into an agreement whereby Microsoft would develop a version of Sybase SQL Server for the Windows NT platform, and Microsoft SQL Server was the result. Now many client/server database vendors, including the leader, Oracle, support Windows NT and/or Windows 9x.

Client/server databases are frequently referred to as SQL databases because they commonly support *Structured Query Language*, or SQL.

**Features of client/server databases**

This section summarizes the major features of client/server databases and, where appropriate, makes comparisons to similar features in file-server databases. Features and comparisons are illustrated using Microsoft SQL Server and Microsoft Visual FoxPro; however, most of these points apply to other client/server and file-server databases as well.

**Data access**

The key difference between client/server and file-server databases is in the way data is accessed. A client/server application always consists of two or more applications: a client and a server. The database server erects a wall around the physical data, and it can only be accessed by sending requests to the server application, which processes the requests and returns the results.

With a Visual FoxPro database, any machine that has VFP or the VFP ODBC driver and access to the data directory can process that data on the local workstation. All processing is actually performed on the local workstation, and all information required to perform that processing must be transmitted from the server to the workstation. After the server data is copied to memory on the workstation, the user can change the data and the changes are written directly to the database on the file server.

With a SQL Server database, the client workstation runs one or more applications that make requests of the database server and accept the results of those requests. The client can make changes to the data locally, but those changes are not made directly to the database. Instead, they are packaged as requests, typically a SQL INSERT, UPDATE or DELETE statement, and sent back to the server. Just as with a request for data, these change requests are handled by the server, which has the ultimate authority and control over how such requests are processed.

SQL Server includes a utility called Profiler that provides an excellent demonstration of just how this works. In Figure 1, you can see a trace in the Profiler. This trace was run while opening a VFP form that opens a couple dozen views. Each line in the trace shows the actual SQL statement sent to the server along with details on the number of disk reads and writes, duration of the processing, and so forth.
4 Client/Server Applications with Visual FoxPro and SQL Server

Security
A Visual FoxPro database has no security. A developer can write procedural code to enforce security, but this type of security can be circumvented.

By contrast, SQL Server databases are totally secure. All access to the database must be through the database server application. By default, no user has access to anything in SQL Server until the administrator has added the user to the system. Even then, the user has no access until the administrator specifically grants it. This system is called declarative security.

Any attempt to access the data causes the server to check for a user’s login ID and password. Figure 2 illustrates an attempt to access the Northwind database from Microsoft Visual InterDev. Note the login dialog.

Code you write in your application to access a SQL Server database will also require authentication by the server. Attempting to open a remote view of the Northwind employee table from the VFP Command Window, as shown in Figure 3, will also prompt the user with a login dialog.
Figure 2. An attempt to log in to the SQL Server Northwind database causes the user to be prompted for a login ID and password.

Figure 3. Attempting to open a remote view of SQL Server data also causes the user to be prompted for a login ID and password.
The preceding illustrations show the SQL Server login dialog, but there are actually many ways to handle logging in. For example, you can configure your ODBC connections to supply a login ID and password when connecting so that the login dialog doesn’t appear at all when the application runs.

SQL Server also offers a feature called Windows NT Integrated Security that can be used instead of the normal SQL Server authentication. With NT Integrated Security, SQL Server checks the name of the user logged in to NT rather than requiring a SQL Server user ID and password.

In addition to authenticating users for access to the database, SQL Server allows administrators to assign rights to any individual object in the database. For example, some users might have access to all columns in the employees table, while others might not be allowed to see addresses or salaries. See Chapter 3, “Introduction to SQL Server 7.0,” for more information on security in SQL Server.

**Database backup**

A friend recently described a client’s nightmare with a VFP database. They performed an automatic tape backup of their network every night. One day, the inevitable happened and the network went down. No problem, they simply went about restoring from backup. Well, not all the tables were backed up, as some were open when the backup was performed. So they went back to the previous night’s backup, but no dice. On and on they went, but no complete backup had been performed because every night somebody had some files open because they forgot to shut down their system or a developer was working late. They were in big trouble.

SQL Server eliminates this problem by allowing live backup of a database while it is in use. An administrator can schedule backups, or an application can periodically send a T-SQL BACKUP command to the server. The database is dumped to a backup file, which is closed as soon as the backup is completed, and this backup file is copied to the backup tape. If the server goes down, the client’s nightmare isn’t a problem. This backup capability permits both 24/7 operation and reliable backup.

**Point-in-time recovery**

SQL Server records every transaction in a transaction log in memory. Each time a transaction is completed, it is copied from the log in memory to the log on the disk. At various intervals, the transactions in the log are written to the physical database on disk. In the case of a crash, the data can be recovered as long as the transaction log is recoverable. Of course, any updates that had not yet been written to the physical transaction log would be lost.

The transaction log itself can also be backed up. Normally, the transaction log is not emptied when the database is backed up. However, when the transaction log itself is backed up, committed transactions are removed from it to keep the log size to a minimum. So if the database is backed up on Tuesdays and the transaction log is backed up every day, then the worst-case scenario even when the transaction log is destroyed is to restore the weekly backup and then each daily transaction log. Only part of a day’s transactions are lost, which is a substantial improvement over the aforementioned client’s nightmare.

Backups can be performed more often; however, backups affect performance of the system. This is one of the trade-off decisions you will have to make when designing a client/server system.
Triggers
Visual FoxPro databases support triggers. A trigger is a stored procedure that is triggered by an INSERT, UPDATE or DELETE of a record in a table. In VFP databases, triggers are used to enforce referential integrity and may be used for other purposes as well. One difficulty with VFP triggers is that the VFP ODBC driver only supports a limited subset of VFP syntax. So code in a trigger that works fine when running in a Visual FoxPro application may not work when accessing data via ODBC.

Although SQL Server can use triggers to enforce referential integrity, declarative referential integrity is the preferred method, simply because declarative integrity performs substantially better than trigger-based integrity.

Triggers are also frequently used to support business rules and are an excellent way to provide an audit trail. For example, a trigger might insert a record into an audit table containing the datetime of the change, the user making the change, and the old and new values.

Here is an example of a very simple auditing trigger. Suppose a fire department wants to keep track of any changes made to the alarm time, arrival time or cleared time for a fire incident. Although it is entirely possible that such a change is being made legitimately to reflect correct times, it is also possible that someone might change these times to make them look better or to cover up mistakes. Here’s the schema for a simple time logging table:

```
CREATE TABLE timelog (timelogkey int IDENTITY (1,1),
edittime datetime,
userid varchar(100),
columnname varchar(100),
oldtime datetime,
newtime datetime)
```

The timelogkey column is an identity column and will automatically enter unique integers, beginning with one and incrementing by one. Now an update trigger is created for the incident table:

```
CREATE TRIGGER utrIncidentUpdate
ON incident
FOR UPDATE
AS
DECLARE @oldtime datetime,
@newtime datetime,
@ikey udtKeyField
IF UPDATE (alarmdate)
BEGIN
SELECT @oldtime = alarmdate, @ikey = incidentkey
FROM deleted
SELECT @newtime = alarmdate
FROM inserted
INSERT timelog
(incidentkey, edittime, userid, columnname, oldtime, newtime)
VALUES
(@ikey, GETDATE(), USER, 'alarmdate', @oldtime, @newtime)
END
```

This trigger requires a bit of explaining. SQL Server stored procedures use temporary cursors that are visible only within the stored procedure. In the case of triggers, which are a
special type of stored procedure, there are two default cursors: deleted and inserted. In delete triggers, the deleted cursor holds the values of the row being deleted, while in update triggers it holds the values of the row prior to the update. In insert triggers, the inserted cursor holds the values of the row being inserted, while in update triggers it contains the new values of the row being updated.

The update trigger in the preceding code checks to see whether one of the three critical times—alarmtime—has been updated. This is done with the UPDATE() function. If so, a row is inserted into the timelog table. The row includes the current datetime (returned by the SQL Server GETDATE() function), the user making the change, and the name of the column being changed. It gets the old and new values from the deleted and inserted cursors, respectively, and inserts them as well.

By extending this technique, you can see that it is possible to create a complete audit trail of every change made in the database.

**Referential integrity**

Visual FoxPro databases support trigger-based referential integrity. When an application or user attempts to delete, modify or insert a record, the appropriate trigger is fired. The trigger determines whether the attempted delete, modification or insert can proceed. A deletion trigger may cause cascaded deletes of child records. Similar processing occurs when an attempt is made to change a primary key value. The change may be prevented by the trigger, or the change may be cascaded through the child tables. Although such trigger-based referential integrity is adequate for some purposes, it becomes less reliable as the schema becomes more complicated, as thousands of triggers could be firing for a single deletion.

While SQL databases also support the use of triggers for the purposes described in the previous paragraph, the preferred method is declarative referential integrity. Declarative referential integrity, supported by SQL Server since version 6.0, enforces referential integrity at the engine level. Deleting a record when children exist is simply prohibited. Instead of using triggers to cascade deletes, a stored procedure is typically written to delete records from the bottom up based on a given primary key for the top-level parent record. This technique is not only more reliable, but it typically provides better performance, too.

Declarative referential integrity is implemented through the use of foreign key constraints. Here is an example of how to create a foreign key constraint:

```
ALTER TABLE EMSAdvanced ADD CONSTRAINT
  fk_EMSAdvanced_incident FOREIGN KEY (incidentkey)
  REFERENCES incident (incidentkey)
```

**Indexes**

Indexes are used in Visual FoxPro databases to display data in a particular order, to improve query performance through Rushmore optimization, to enforce unique values, and to identify unique primary keys. SQL Server essentially uses indexes for the same purposes, but SQL Server does not use Rushmore. Instead, it uses its own optimization techniques designed specifically for the SQL Server query engine.
Clustered indexes
When a new record is added to a VFP table, it is typically appended to the end of the file, as this is much more efficient than writing a record in the middle of a file. If no index order is set, then browsing a table will show the records in this native order. Sometimes it makes sense for performance reasons to occasionally sort a table based on the value of some field, such as a primary key.

In SQL Server, the physical order of records can be controlled with a clustered index. Each table may have one clustered index, and a new record will be inserted into the table in the order determined by the clustered index. Clustered indexes can improve query performance when queries need to return a range of consecutive records. However, they tend to decrease insert or update performance, since these operations could force a reorganization of the table.

A clustered index on the customerid column of the Northwind customers table is created like this:

```sql
CREATE CLUSTERED INDEX idxcustomerid
ON customers (customerid)
```

Unique indexes
In a VFP table, a candidate index is used to enforce the uniqueness of a value in a table. They are called candidate indexes because the unique value is a likely candidate for a primary key. In SQL Server, the same thing is accomplished with a unique index. Don’t confuse this with a unique index in VFP (i.e., INDEX ON...TAG tagname UNIQUE), which is simply an index containing only a single key even when the table contains multiple records, each of which has a key of the same value. A unique index in SQL Server, like a candidate index in VFP, prevents duplication of the value in the table. A unique index on the employeeid column of the Northwind employees table is created like this:

```sql
CREATE UNIQUE INDEX pkemployeeid
ON employees (employeeid)
```

Primary keys
In a VFP database, you can specify one primary index per table like this:

```sql
ALTER TABLE mytable ADD PRIMARY KEY myfield TAG mytag
```

Behind the scenes, VFP actually creates a candidate tag in the index file and then adds a special entry in the DBC to indicate that it is the primary key.

Primary keys in SQL Server are very similar, using primary key constraints. This code creates a primary key constraint and a clustered index on the employeeid column of the employees table:

```sql
ALTER TABLE employee ADD CONSTRAINT pkemployeeid
    PRIMARY KEY CLUSTERED (employeeid)
```
Non-clustered indexes
Rushmore optimization in Visual FoxPro is effected with the use of index tags. Query performance is improved under most circumstances by having an index that matches the filter expression of a SELECT. Query optimization in SQL Server works in much the same way. Fields that are likely to be used in filter expressions should have non-clustered indexes. A non-clustered index on the lastname column of the Northwind employees table is created like this:

```
CREATE NON-CLUSTERED INDEX idxLastName ON employees (lastname)
```

Defaults
Both Visual FoxPro and SQL Server databases support defaults. Defaults allow you to specify a default value for a field. For example, a merchant in the state of Washington might want to assume that its customers are residents of Washington and automatically insert ‘WA’ in the state column.

```
CREATE DEFAULT uddWAState AS 'WA'
EXEC sp_bindefault uddWAState, 'customers.state'
```

Rules
Both Visual FoxPro and SQL Server databases support rules. Rules allow you to specify field-level validation in the database. Once specified, rules are enforced by the database engine.

A SQL Server rule is created using a variable, rather than a column name. Here’s a rule that requires Social Security numbers to consist of nine numeric values. Note the use of the @social variable (the “@” always denotes a local variable in SQL Server):

```
CREATE RULE udrSocial AS
    @social LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
```

By using a variable, rather than the name of a column, the same rule can be applied to any number of columns:

```
EXEC sp_bindrule udrSocial, 'employees.socialsecurity'
EXEC sp_bindrule udrSocial, 'drivers.license'
```

Primary key generation
In Visual FoxPro, you can generate primary key values through one of two common techniques. You can call a function or method that returns the next available primary key value, and then insert this value into the new record before it is committed. Alternatively, you can write a stored procedure that is invoked by the Default Value of a field, which automatically places the new value into the field. As with any other VFP stored procedure, you are restricted outside of a VFP application by the limitations of the VFP ODBC driver. The default values you so painstakingly create may not be generated at all when a record is inserted via ODBC.
SQL Server automates primary key generation by using identity columns. An identity column may not have a value inserted into it. Instead, it will automatically be set to the next available value for that column. The initial (or seed) value can be set, as well as the amount by which the value is incremented. The identity attribute can also be turned off for a column if you need manual control over the values inserted into the column. The attribute can be turned back on and set to increment from the highest existing value. This is all handled automatically by the engine and requires no code on the part of the developer.

However, identity columns are no panacea. Retrieving the last value is normally handled by checking the value of the @IDENTITY global variable. But it may be difficult to get the correct value of this variable, as insert triggers might have caused other identity columns to be incremented and thus would return the wrong value for @IDENTITY. Identity columns can also cause a problem when databases are replicated. If you have the choice, you should avoid identity columns when you design a database. Learn to use them, though, because you may find yourself working on databases that use them.

**Stored procedures**

Visual FoxPro databases support stored procedures. However, stored procedures that run in Visual FoxPro may not run via the VFP ODBC driver, so it is very difficult to create stored procedures that are of any value outside of a VFP application. Because SQL Server stored procedures are run by SQL Server, you never have this incompatibility issue.

One use for stored procedures in SQL Server is to provide parameterized record sets of data. VFP views support parameters that can be used to filter the rows returned by the view. For example, the customerorders view can be defined with a parameter to return only those records matching a particular customer ID:

```sql
CREATE SQL VIEW customerorders AS
SELECT customers.companyname, orders.orderdate,
    orderdetails.productid, orderdetails.quantity, orderdetails.unitprice
FROM customers JOIN orders
    ON customers.customerid = orders.customerid;
WHERE customers.customerid LIKE ?cCustomerID
```

The parameterized view is opened by setting the value of the parameter and opening the view with USE:

```sql
cCustomerID = 'ALFKI'
USE customerorders
```

Parameterized views are not supported by SQL Server, but they can be simulated with a stored procedure like this:

```sql
CREATE PROCEDURE usp_customerorders
    @cCustomerID nchar(5)
AS
    SELECT customers.companyname, orders.orderdate,
        orderdetails.productid, orderdetails.quantity, orderdetails.unitprice
    FROM customers JOIN orders
        ON customers.customerid = orders.customerid
```
JOIN orderdetails 
    ON orders.orderid = orderdetails.orderid 
WHERE customers.customerid LIKE @cCustomerID

In T-SQL code, one would access this stored procedure like this:

EXEC usp_customerorders 'ALFKI'

And from a VFP application, one would access this stored procedure like this:

SQLEXEC(lnHandle, "EXEC usp_customerorders 'ALFKI'")

Stored procedures in SQL Server are written using Transact-SQL, also known as T-SQL, SQL Server’s programming language. Although not as rich a language as Visual FoxPro and lacking any record-based data navigation (T-SQL is purely set-based), it is nonetheless a powerful procedural language and can be used for many purposes. It contains syntactical equivalents to such VFP constructs as IF..ELSE, DO WHILE, RETURN, PARAMETERS and so on.

One powerful feature of stored procedures in SQL Server is that they can be assigned security rights just like any other object in a database. One very good use for this is to use stored procedures for database updates rather than allowing direct access to the tables. The administrator takes away INSERT, UPDATE and DELETE rights for tables and/or specific columns, and the only way for a change to be made is to call the appropriate stored procedure and pass it the values necessary for the change.

Calling SQL Server stored procedures from Visual FoxPro applications will be examined in greater detail in Chapter 6, “Extending Remote Views with SQL Pass Through.”

Views
Visual FoxPro databases support views. A view is nothing more than a predefined SQL SELECT. Views can be used to determine which columns to include in a record set or to perform multi-table joins. A useful view limiting the number of rows and columns returned in a record set might look like this:

CREATE SQL VIEW customerorders AS ;
    SELECT customers.companyname, orders.orderdate, ;
        orderdetails.productid, orderdetails.quantity, orderdetails.unitprice ;
    FROM customers JOIN orders ;
    ON customers.customerid = orders.customerid ;
JOIN orderdetails ;
    ON orders.orderid = orderdetails.orderid

This view performs a three-way join and returns only five columns. This type of view may be ideal for reporting and can also be used for data entry, as views can be made updatable. Predefining the three-way join simplifies things for users who may otherwise have to write such a query themselves. However, the VFP ODBC driver only supports the calling of VFP views that are not parameterized through SQL pass through, so many of your views are only available in a VFP application.
SQL Server also supports views. Here is a T-SQL definition for the same view defined previously:

```sql
CREATE VIEW customerorders AS
SELECT customers.companyname, orders.orderdate,
    orderdetails.productid, orderdetails.quantity, orderdetails.unitprice
FROM customers JOIN orders
    ON customers.customerid = orders.customerid
JOIN orderdetails
    ON orders.orderid = orderdetails.orderid
```

As with VFP views, SQL Server views can be used to simplify access to the data and can be made updatable. However, SQL Server views are available to any application that can access SQL Server tables, making them more flexible than VFP views. Furthermore, security rights can be assigned to a view in SQL Server. Many developers and DBAs enforce security by withholding rights to table objects and allowing access to views instead.

**User-defined data types**

When you define a table in a VFP database, you are limited to the data types that VFP supports. In SQL Server, you can create a user-defined type, based on an intrinsic (that is, built-in to SQL Server) type, and use it when you define a table. For example, the US Fire Administration defines a set of codes, called 901 Codes, that are used for reporting fire incidents. When reporting a fire incident, there are literally hundreds of fields that could contain either 901 Codes or some other type of data. Here’s how we create a user-defined type for 901 Codes:

```sql
IF NOT EXISTS (SELECT * FROM systypes WHERE name = 'udtCode901')
    EXEC sp_addtype udtCode901, 'char(4)', 'NULL'
```

Now we have two ways to define a table:

```sql
CREATE TABLE mytable (myfield char(4) NULL)
```

or:

```sql
CREATE TABLE mytable (myfield udtCode901)
```

What’s the difference? What have you gained by using the user-defined type? When you look at the schema for this table, you now know not only the structure of the column, but also something about the nature, or business domain, of the data contained in it because you know it is designed to hold 901 Codes. Many developers spend a lot more time trying to figure out existing code than they do writing new code, and anything you can do to document your design will make somebody else’s (or your own future) work easier.

An important restriction on user-defined data types is that they do not provide inheritance. In other words, if the US Fire Administration changed the codes from four characters to five, you cannot simply modify the udtCode901 data type and expect the tables to pick up the change. Instead, you must first unbind or remove the data type from any columns where it is
used, and then make the modification to the data type. After the change is made, you can re-bind the data type to the appropriate columns.

**Replication**

Replication is the process of synchronizing multiple copies of a database. A company may have a database in the headquarters and a copy in each regional office. Each night these databases can be replicated so the headquarters and each regional office will have a copy of the latest data.

Visual FoxPro databases have no native support for replication. If you want to replicate a VFP database, you must write the code to do it yourself. While messages occasionally appear in online forums declaring how “easy” it is to do this, consider a group of conference attendees who were asked whether they’ve ever attempted this. Only a small percentage said yes, and of those, almost all gave up before completing the task. Those who completed it usually said they wouldn’t want to do it again.

SQL Server has built-in replication, which is handled as an administrative function. However, just because the replication is built in doesn’t mean it’s easy to get it to work. You still have to ensure that primary keys uniquely identify records, even across multiple copies of the database.

**Transactions**

Visual FoxPro supports limited transaction protection with BEGIN TRANSACTION, END TRANSACTION and ROLLBACK. SQL Server’s transaction protection is far more robust, as explained in the “Point-in-time recovery” section earlier in this chapter. In addition, by exposing its transaction process to Microsoft Distributed Transaction Coordinator (MS DTC), SQL Server can participate in transactions across databases, servers and even database systems. With MS DTC, multiple databases on multiple servers running SQL Server, Oracle and/or MSDE can all participate in the same transaction. Transactions are covered in greater detail in Chapter 11, “Transactions.”

**Scalability**

The term *scalability* is in vogue right now. Microsoft uses the word a lot because it’s hell-bent on overtaking Sun and Oracle in the enterprise market. Scalability is what the press has often claimed Windows NT and SQL Server lack in comparison to Oracle running on a Sun server. When an application is described as scaling well, it is typically meant that it can handle very high usage.

The term *scalability* can also be applied to applications written for high-usage environments that can be used in smaller systems as well. Chapter 7, “Downsizing,” addresses this downward scalability.

Visual FoxPro is capable of handling very large amounts of data, with the engine supporting tables up to 2GB. However, because the processing is handled by the workstation, really large tables usually cannot be handled efficiently.

SQL Server can handle terabytes of data. (A terabyte is a trillion bytes, or 1000GB.) To get an idea of just how big a terabyte of data really is, consider this: The entire 100+ year history of every transaction ever performed on the New York Stock Exchange is approximately 500GB, or one-half terabyte!