

Ron's Story

In 1988, Ron Soukup was working for Covia, the United Airlines subsidiary that provided the Apollo Reservation System and related systems for airport and travel agency use. He had spent the previous five years working with the new breed of relational database products that had appeared on the minicomputer and mainframe computer scene. He had worked with IBM's DB/2 running on MVS; IBM's SQL/DS running on VM; and Oracle, Informix, and Unify running on UNIX. Ron viewed PC databases of the day essentially as toys that were good for storing recipes and addresses but not much else. He used a PC for word processing, but that was about it. At work, they were beginning to use more and more LAN-based and PC-based systems, and Ron had begun doing some OS/2 programming. When he heard of the NDK, with this new SQL Server product that had been mentioned in the trade press, he ordered it immediately.

The NDK was very much a beta-level product. It didn't have a lot of fit and finish, and it crashed a couple of times a day. But from practically the first day, Ron knew that this product was something special. He was amazed that he was using a true DBMS—on a PC!—with such advanced features as transaction logging and automatic recovery. Even the performance seemed remarkably good. Having used mostly minicomputer and mainframe computer systems, he was most struck by the difference in PC response time. With the bigger systems, even a simple command resulted in an inevitable delay of at least a couple of seconds between pressing the Enter key and receiving a reply. PCs seemed almost instantaneous. Ron knew PCs were fast for local tasks such as word processing, but this was different. In this case, at one PC he entered an SQL command that was sent over the network to a server machine running this new SQL Server product. The response time was a subsecond. He had never seen such responsiveness.

Ron's initial kick-the-tires trial was encouraging, and he received approval to test the product more thoroughly. He wanted to get a feel for the types of applications and workloads for which this interesting new product might be used. For this, Ron wanted more substantial hardware than the desktop machine he originally tested on (a 10MHz 286 computer with 6 MB of memory and a 50-MB hard drive). Although SQL Server ran reasonably well on his desktop machine, he wanted to try it on one of the powerful new machines that used the Intel 80386 processor. Ron was able to procure a monster machine for the day—a 20-MHz 386 system with 10 MB of memory and two 100-MB disk drives—and was the envy of his division!

In 1987, Ron and a colleague at Covia had developed some multiuser database benchmark tests in C to help them choose a UNIX minicomputer system for a new application. So Ron dusted off these tests and converted the embedded C to the call-level interface provided in SQL Server (DB-Library) and ported these benchmarks to the PC. Ron hoped that SQL Server would handle several simultaneous users, although he didn't even consider that it could come close to handling the 15 to 20 simulated users they had tried in the earlier minicomputer tests. After many false starts and the typical problems that occur while running an early beta of a version 1.0 product, he persevered and got the test suite—2000 lines of custom C code—running on a PC against the beta version of SQL Server.

The results were amazing. This beta version of SQL Server, running on a PC that cost less than \$10,000, performed as well and in many cases better than the minicomputer systems that they had tested a few months earlier. Those systems cost probably 10 times as much as the PC that Ron was using, and they needed to be managed by a professional UNIX system administrator. Ron knew the industry was in for a big change.

In May 1989, Ashton-Tate/Microsoft SQL Server version 1.0 shipped. Press reviews were good, but sales lagged. OS/2 sales were far below what had been expected. Most users hadn't moved from MS-DOS to OS/2, as had been anticipated. And about the only tool available to create SQL Server applications was C. The promised dBASE IV Server Edition from Ashton-Tate was delayed, and although several independent

software vendors (ISVs) had promised front-end development tools for SQL Server, these hadn't materialized yet.

During the preceding six months, Ron had come to really know, respect, and admire SQL Server, and he felt the same about the people at Microsoft with whom he had worked during this period. So in late 1989, Ron accepted a position at Microsoft in the SQL Server group in Redmond, Washington. A few months later, he was running the small but talented and dedicated SQL Server development team.

Kalen's Story

Kalen Delaney saw the first version of Microsoft SQL Server at about the same time that Ron did. She started working for Sybase Corporation's technical support team in 1987. And in 1988, she started working with a team that was testing the new PC-based version of Sybase SQL Server, which a company in the Seattle area was developing for OS/2. Sybase would be providing phone support for customers purchasing this version of the product, and Kalen, along with the rest of the technical support team, wanted to be ready.

Up to that point, she had never actually worked with PC-based systems and had pegged them as being mere toys. Sybase SQL Server was an extremely powerful product, capable of handling hundreds of simultaneous users and hundreds of megabytes of data. Kalen supported customers using Sybase SQL Server databases to manage worldwide financial, inventory, and human resource systems. A serious product was required to handle these kinds of applications, and Kalen thought PCs just didn't fit the bill. However, when she saw a PC sitting on a coworker's desk actually running SQL Server, Kalen's attitude changed in a hurry. Over the next few years, she worked with hundreds of customers in her product support position and later in her position as Senior Instructor, using Microsoft SQL Server to manage real businesses with more than toy amounts of data, from a PC.

In 1991, Kalen and her family left the San Francisco Bay Area to live in the Pacific Northwest, but she continued to work for Sybase as a trainer and courseware developer. A year later, Sybase dissolved her remote position, and she decided to start her own business. A big decision loomed: should she focus primarily on the Sybase version or the Microsoft version of SQL Server? Because Microsoft SQL Server would run on a PC that she could purchase at her local Sears store with low monthly payments, and because she could buy SQL Server software and the required operating system in one package, the decision was simple. Microsoft had made it quite easy for the sole proprietor to acquire and develop software and hone his or her skills on a powerful, real-world, relational database system.

Microsoft SQL Server Ships

By 1990, the co-marketing and distribution arrangement with Ashton-Tate, which was intended to tie SQL Server to the large dBASE community, simply wasn't working. Even the desktop version of dBASE IV was quite late, and it had a reputation for being buggy when it shipped in 1989. The Server Edition, which would ostensibly make it simple to develop higher-performance SQL Server applications using dBASE, was nowhere to be seen.

As many others have painfully realized, developing a single-user, record-oriented application is quite different from developing applications for multiple users for which issues of concurrency, consistency, and network latency must be considered. Initial attempts at marrying the dBASE tools with SQL Server had

dBASE treating SQL Server as though it were an indexed sequential access method (ISAM). A command to request a specific row was issued for each row needed. Although this procedural model was what dBASE users were accustomed to, it wasn't an efficient way to use SQL Server, with which users could gain more power with less overhead by issuing SQL statements to work with sets of information. But at the time, SQL Server lacked the capabilities to make it easy to develop applications that would work in ways dBASE users were accustomed to (such as browsing through data forward and backward, jumping from record to record, and updating records at any time). Scrollable cursors didn't exist yet.

NOTE

The effort to get dBASE IV Server Edition working well provided many ideas for how scrollable cursors in a networked client/server environment should behave. In many ways, it was the prime motivation for including this feature in SQL Server version 6.0 in 1995, six years later.

Only two years earlier, Ashton-Tate had been king of the PC database market. Now it was beginning to fight for its survival and needed to refocus on its core dBASE desktop product. Microsoft would launch OS/2 LAN Manager under the Microsoft name (as opposed to the initial attempts to create only OEM versions), and it needed SQL Server to help provide a foundation for the development of client/server tools that would run on Microsoft LAN Manager and Microsoft OS/2. So Microsoft and Ashton-Tate terminated their co-marketing and distribution arrangements. The product would be repackaged and reintroduced as Microsoft SQL Server.

Microsoft SQL Server version 1.1 shipped in the summer of 1990 as an upgrade to the Ashton-Tate/Microsoft SQL Server version 1.0 that had shipped in 1989. For the first time, SQL Server was a Microsoft-supported, shrink-wrapped product, and it was sold through the newly formed Microsoft Network Specialist channel, whose main charter was to push Microsoft LAN Manager.

NOTE

When version 1.1 shipped, Microsoft didn't see SQL Server as a lucrative product in its own right. SQL Server would be one of the reasons to buy LAN Manager—that's all.

SQL Server 1.1 had the same features as version 1.0, although it included many bug fixes—the type of maintenance that is understandably necessary for a version 1.0 product of this complexity. But SQL Server 1.1 also supported a significant new client platform, Microsoft Windows 3.0. Windows 3.0 had shipped in May 1990, a watershed event in the computer industry. SQL Server 1.1 provided an interface that enabled Windows 3.0-based applications to be efficiently developed for it. This early and full support for Windows 3.0-based applications proved to be vital to the success of Microsoft SQL Server. The success of the Windows platform would also mean fundamental changes for Microsoft and SQL Server, although these changes weren't yet clear in the summer of 1990.

With the advent of Windows 3.0 and SQL Server 1.1, many new Windows-based applications showed up and many were, as promised, beginning to support Microsoft SQL Server. By early 1991, dozens of third-party software products used SQL Server. SQL Server was one of the few database products that provided a Windows 3.0 dynamic link library (DLL) interface practically as soon as Windows 3.0 shipped, which was now paying dividends in the marketplace. Quietly but unmistakably, Microsoft SQL Server was the leading database system used by Windows applications. Overall sales were still modest, but until tools beyond C existed to build solutions, sales couldn't be expected to be impressive. It was the classic chicken-and-egg situation.

Development Roles Evolve

Microsoft's development role for SQL Server 1.0 was quite limited. As a small porting team at Sybase moved its DataServer engine to OS/2 and moved the DB-Library client interfaces to MS-DOS and OS/2, Microsoft provided testing and project management. Microsoft also developed some add-on tools to help make SQL Server 1.0 easy to install and administer.

Although a number of sites were running OS/2 as an application server with SQL Server or as a file server with LAN Manager, few were using OS/2 for their desktop platforms. Before Windows 3.0, most desktops remained MS-DOS based, with a well-known limit of 640 KB of addressable real memory. After loading MS-DOS, a network redirector, network card device drivers, and the DB-Library static link libraries that shipped with SQL Server version 1.0, developers trying to write a SQL Server application were lucky to get 50 KB for their own use.

For SQL Server 1.1, rather than shipping the DB-Library interface that Sybase had ported from UNIX to MS-DOS, Microsoft wrote its own, from scratch. Instead of 50 KB, developers could get up to 250 KB to write their applications. Although small by today's standards, 250 KB was a huge improvement.

NOTE

The same source code used for DB-Library for MS-DOS also produced the Windows and OS/2 DB-Library interfaces. But the MS-DOS RAM cram is what motivated Microsoft to write a new implementation from scratch. The widespread adoption of Windows 3.0 quickly eliminated the MS-DOS memory issue—but this issue was a real problem in 1989 and 1990.

With SQL Server 1.1, Microsoft provided client software and utilities, programming libraries, and administration tools. But the core SQL Server engine was still produced entirely by Sybase; Microsoft didn't even have access to the source code. Any requests for changes, even for bug fixes, had to be made to Sybase.

Microsoft was building a solid support team for SQL Server. It hired some talented and dedicated engineers with database backgrounds. But with no access to source code, the team found it impossible to provide the kind of mission-critical responsiveness that was necessary for customer support. And, again, fixing bugs was problematic because Microsoft depended entirely on Sybase, which had become successful in its own right and was grappling with its explosive growth. Inevitably, some significant differences arose in prioritizing which issues to address, especially when some issues were specific to the Microsoft-labeled product and not to Sybase's product line. Bugs that Microsoft deemed of highest priority sometimes languished because Sybase's priorities were understandably directed elsewhere. The situation was unacceptable.

It was a great day in the SQL Server group at Microsoft when, in early 1991, Microsoft's agreement with Sybase was amended to give Microsoft read-only access to the source code for the purpose of customer support. Although Microsoft's SQL Server group still couldn't fix bugs, they at least could read the source code to better understand what might be happening when something went wrong. And they could also read the code to understand how something was expected to work. As anyone with software development experience knows, even the best specification is ambiguous at times. There's simply no substitute for the source code as the definitive explanation for how something works.

As a small group of developers at Microsoft became adept with the SQL Server source code and internal workings, Microsoft began to do virtual bug fixes. Although they still weren't permitted to alter the source code, they could identify line-by-line the specific modules that needed to be changed to fix a bug.

Obviously, when they handed the fix directly to Sybase, high-priority bugs identified by Microsoft were resolved much more quickly.

After a few months of working in this way, the extra step was eliminated. By mid-1991, Microsoft could finally fix bugs directly. Because Sybase still controlled the baseline for the source code, all fixes were provided to Sybase for review and inclusion in the code. The developers at Microsoft had to make special efforts to keep the source code highly secured, and the logistics of keeping the source code in sync with Sybase was sometimes a hassle, but all in all, this was *heaven* compared to a year earlier. Microsoft's team of developers was becoming expert in the SQL Server code, and they could now be much more responsive to their customers and responsible for the quality of the product.

OS/2 and Friendly Fire

In 1991, Microsoft released SQL Server 1.11, a maintenance release. SQL Server was slowly but steadily gaining acceptance and momentum—and a long list of ISV supporters. Client/server computing wasn't widely deployed yet, but new converts appeared every day. Customer satisfaction and loyalty were high, and press reviews of the product were all favorable. Sales were generally disappointing, but this was hardly a surprise because OS/2 had continued to be a major disappointment. Windows 3.0, however, was a runaway hit. Rather than move their desktop platforms from MS-DOS to OS/2, huge numbers of PC users moved to Windows 3.0 instead. OS/2 hadn't become a widespread operating system as anticipated, and it was now abundantly clear that it never would be.

SQL Server's Limitations and the Marketplace

~~Microsoft SQL Server 1.11 clearly had a scalability limit. It was a 16-bit product because OS/2~~ could provide only a 16-bit address space for applications, and its throughput was hampered by OS/2's lack of some high-performance capabilities, such as asynchronous I/O. Even though an astonishing amount of work could be performed successfully with SQL Server on OS/2, there would come a point at which it would simply run out of gas. No hard limit was established, but in general, SQL Server for OS/2 was used for workgroups of 50 or fewer users. For larger groups, customers could buy a version of Sybase SQL Server for higher-performance UNIX-based or VMS-based systems.

This was an important selling point for both Microsoft and Sybase. Customers considering the Microsoft product wanted to be sure they wouldn't outgrow it. The large number of ISV tools developed for Microsoft SQL Server worked largely unchanged with Sybase SQL Server, and applications that outgrew OS/2 could be moved quite easily to a bigger, more powerful, more expensive UNIX system. This relationship still made sense for both Microsoft and Sybase.

The need for compatibility and interoperability made it especially important for Microsoft SQL Server to be based on the version 4.2 source code as soon as possible. Furthermore, a major features version hadn't been released since version 1.0 in 1989. In the rapidly moving PC marketplace, the product was in danger of becoming stale. Customers had begun to do serious work with Microsoft SQL Server, and new features were in great demand. Microsoft's version 4.2 would add a long list of significant new features, including server-to-server stored procedures, UNION, online tape backup, and greatly improved international support that would make SQL Server more viable outside the United States.

development team to *not* develop a full 32-bit version for OS/2. One of the joys of working for Microsoft is that senior management empowers the people in charge of projects to make the decisions, and this decision was left with Ron Soukup.

In early 1992, however, the development team faced some uncertainty and *external* pressures. On one hand, the entire customer base was, by definition, using OS/2. Those customers made it quite clear that they wanted—indeed, expected—a 32-bit version of SQL Server for OS/2 2.0 as soon as IBM shipped OS/2 2.0, and they intended to remain on OS/2 in the foreseeable future. But when OS/2 2.0 would be available was unclear. IBM claimed that it would ship by the fall of 1992. Steve Ballmer, Microsoft senior vice president, made a well-known pledge that he'd eat a floppy disk if IBM shipped the product in 1992.

The SQL Server team was now under pressure to have a version of SQL Server running on Windows NT as soon as possible, with prerelease beta versions available when Windows NT was prereleased. The SQL Server team members knew that Windows NT was the future. It would be the high-end operating system solution from Microsoft, and from a developer's perspective, Windows NT would offer a number of technical advantages over OS/2, including asynchronous I/O, symmetric multiprocessing, and portability to reduced instruction set computing (RISC) architectures. They were champing at the bit to get started.

Although Microsoft had decided in 1991 to fall back to a 16-bit version of SQL Server, Microsoft's developers had continued to work on the 32-bit version. By March 1992, just after shipping version 4.2, they saw that both the 16-bit and 32-bit versions ran slower on the beta versions of OS/2 2.0 than the 16-bit version ran on Tiger (OS/2 1.3). Perhaps by the time OS/2 2.0 actually shipped, it might run faster. But then again, it might not—the beta updates of OS/2 2.0 didn't give any indication that it was getting faster. In fact, it seemed to be getting slower and more unstable.

For a product with the scope of SQL Server, there's no such thing as a small release. There are big releases and bigger releases. Because resources are finite, the developers knew that working on a product geared toward OS/2 2.0 would slow down the progress of Windows NT development and could push back its release. Adding more developers wasn't a good solution. (As many in the industry have come to learn, adding more people is often the cause of, and seldom the solution to, software development problems.) Furthermore, if Microsoft chose to develop simultaneously for both OS/2 2.0 and Windows NT, the developers would encounter another set of problems. They would have to add an abstraction layer to hide the differences in the operating systems, or they'd need substantial reengineering for both, or they'd simply take a lowest-common-denominator approach and not fully use the services or features of either system.

So Microsoft developers decided to stop work on the 32-bit version of SQL Server for OS/2 2.0. Instead, they moved full-speed ahead in developing SQL Server for Windows NT, an operating system with an installed base of zero. They didn't constrain the architecture to achieve portability back to OS/2 or to any other operating system. They vigorously consumed whatever interfaces and services Windows NT exposed. Windows NT was the only horse, and the development team rode as hard as they could. Except for bug fixing and maintenance, development ceased for SQL Server for OS/2.

Microsoft began to inform customers that future versions, or a 32-bit version, of SQL Server for OS/2 2.0 would depend on the volume of customer demand and that the primary focus was now on Windows NT. Most customers seemed to understand and accept this position, but for customers who had committed their businesses to OS/2, this was understandably not the message that they wanted to hear.

At this time, Sybase was working on a new version of its product, to be named System 10. As was the case when they were working on version 4.2, the developers needed Microsoft SQL Server to be compatible with and have the same version number as the Sybase release on UNIX. OS/2 vs. Windows NT was foremost at Microsoft, but at Sybase, the success of System 10 was foremost.

Although System 10 wasn't even in beta yet, a schedule mismatch existed between the goal of getting a version of Microsoft SQL Server onto Windows NT as soon as possible and getting a version of System 10 onto Windows NT, OS/2 2.0, or both as soon as possible. The development team decided to compromise and specialize. Microsoft would port SQL Server version 4.2 for OS/2 to Windows NT, beginning immediately. Sybase would bring Windows NT under its umbrella of core operating systems for System 10. Windows NT would be among the first operating system platforms on which System 10 would be available. In addition, Microsoft would turn the OS/2 product back over to Sybase so those customers who wanted to stay with OS/2 could do so. Although Microsoft hoped to migrate most of the installed customer base to Windows NT, they knew that this could never be 100 percent. They were honestly pleased to offer those customers a way to continue with their OS/2 plans, via Sybase. The SQL Server development team truly didn't want them to feel abandoned.

This compromise and specialization of development made a lot of sense for both companies. The development team at Microsoft would be working from the stable and mature version 4.2 source code that they had become experts in. Porting the product to a brand new operating system was difficult enough, even *without* having to worry about the infant System 10 code base. And Sybase could concentrate on the new System 10 code base without worrying about the inevitable problems of a prebeta operating system. Ultimately, System 10 and SQL Server for Windows NT would both ship, and the two companies would again move back to joint development. That was the plan, and in 1992 both sides expected this to be the case.

The SQL Server team at Microsoft started racing at breakneck speed to build the first version of SQL Server for Windows NT. Time to market was, of course, a prime consideration. Within Microsoft, the team had committed to shipping within 90 days of the release of Windows NT; within the development group, they aimed for 30 days. But time to market wasn't the only, or even chief, goal. They wanted to build the best database server for Windows NT that they could. Because Windows NT was now SQL Server's only platform, the developers didn't need to be concerned about portability issues, and they didn't need to create an abstraction layer that would make all operating systems look alike. All they had to worry about was doing the best job possible with Windows NT. Windows NT was designed to be a portable operating system, and it would be available for many different machine architectures. SQL Server's portability layer would be Windows NT itself.

The development team tied SQL Server into management facilities provided by Windows NT, such as raising events to a common location, installing SQL Server as a Windows NT service, and exporting performance statistics to the Windows NT Performance Monitor. Because Windows NT allows applications to dynamically load code (using DLLs), an open interface was provided to allow SQL Server to be extended by developers who wanted to write their own DLLs.

This first version of Microsoft SQL Server for Windows NT was far more than a port of the 4.2 product for OS/2. Microsoft essentially rewrote the *kernel* of SQL Server—the portion of the product that interacts with the operating system—directly to the Win32 API.

Another goal of SQL Server for Windows NT was to make it easy to migrate current installations on OS/2 to the new version and operating system. The developers wanted all applications that were written to SQL Server version 4.2 for OS/2 to work unmodified against SQL Server for Windows NT. Because Windows NT could be dual-booted with MS-DOS or OS/2, the team decided that SQL Server for Windows NT would directly read from and write to a database created for the OS/2 version. During an evaluation phase, for instance, a customer could work with the OS/2 version, reboot the same machine, work with the Windows NT version, and then go back to OS/2. Although it would be difficult to achieve, the developers wanted 100 percent compatibility.

The developers reworked SQL Server's internals and added many new management, networking, and

extensibility features; however, they didn't add new external core database engine features that would compromise compatibility.

The Windows NT-Only Strategy

Many have questioned the Windows NT-only strategy. But in 1992, the UNIX DBMS market was already overcrowded, so Microsoft developers felt they wouldn't bring anything unique to that market. They recognized that SQL Server couldn't even be in the running when UNIX was a customer's only solution. Microsoft's strategy was based on doing the best possible job for Windows NT, being the best product on Windows NT, and helping to make Windows NT compelling to customers.

The decision to concentrate on only Windows NT has had far-reaching effects. To be portable to many operating systems, the Sybase code base had to take on or duplicate many operating system services. For example, because threads either didn't exist on many UNIX operating systems or the thread packages differed substantially, Sybase had essentially written its own thread package into SQL Server code. The Windows NT scheduling services, however, were all based on a thread as the schedulable unit. If multiprocessors were available to the system and an application had multiple runnable threads, the application automatically became a multiprocessor application. So the Microsoft SQL Server developers decided to use native Windows NT threads and not the Sybase threading engine.

The development team made similar choices for the use of asynchronous I/O, memory management, network protocol support, user authentication, and exception handling. (In [Chapter 3](#), we'll look at the SQL Server architecture in depth and delve into more of these specifics. The point here is that the goals intrinsic to portability are in conflict with the goal to create the best possible implementation for a single operating system.)

For example, the developers ensured that there were no differences in the SQL dialect or capabilities of the Windows NT and OS/2 versions. The plan was for the future System 10 version to implement many new features. This release would be fundamentally a platform release. To emphasize compatibility with the OS/2-based 4.2 product and with the Sybase product line, Microsoft decided to call the first version of SQL Server for Windows NT *version 4.2*. The product was referred to as simply *Microsoft SQL Server for Windows NT* and often internally as *SQL NT*, a designation that the press and many customers also were beginning to use.

In July 1992, Microsoft hosted a Windows NT developers' conference and distributed a prebeta version of Windows NT to attendees. Even though SQL Server didn't exist yet at a beta level, Microsoft immediately made available via CompuServe the 32-bit programming libraries that developers needed for porting their applications from OS/2 or 16-bit Windows to Windows NT. Just as it had enjoyed success back in 1990 by providing the NDK to prospective Windows 3.0 developers, Microsoft sought the same success by providing all the necessary tools to would-be Windows NT developers.

Commitment to Customers

Incidentally, at approximately the same time they were delivering the NDK to developers, Microsoft also shipped a maintenance release of SQL Server for OS/2 (and they shipped another the following year). In porting to Windows NT, the development team found and fixed many bugs that were generic to all platforms. Even though they wouldn't create new SQL

wouldn't be excited about it. As is always the case in development, many features *could* be implemented for every feature that is actually implemented: features specific to Windows NT didn't tend to interest Sybase as much as those that would benefit its UNIX products.

Sybase engineers had to confront the issue of portability to multiple operating systems. In fact, Microsoft's implementation of version 4.2 for Windows NT was already causing friction because Sybase was progressing with System 10 for Windows NT. Sybase was understandably implementing System 10 in a more portable manner than Microsoft had done. This was entirely rational for Sybase's objectives, but from Microsoft's perspective, it meant a looser bond with Windows NT. System 10 would not, and *could not*, perform as well on Windows NT as the product that had been designed and written exclusively for Windows NT.

Because of the economics involved, as well as the changing competitive landscape, the Microsoft/Sybase agreement of 1987 was no longer working. Microsoft SQL Server was now a viable competitive alternative to Sybase SQL Server running on UNIX, Novell NetWare, and VMS. Far from seeding the market for Sybase, Microsoft SQL Server was now taking sales away from Sybase. Instead of choosing a UNIX solution, customers could buy Microsoft SQL Server at a fraction of the cost of a UNIX solution, run it on less expensive PC hardware, and install and administer it more easily. Although Sybase earned royalties on sales of Microsoft SQL Server, this amounted to a small fraction of the revenue Sybase would have received if customers had bought Sybase products for UNIX in the first place. Microsoft and Sybase were now often vigorously competing for the same customers. Both companies recognized that a fundamental change in the relationship was needed.

On April 12, 1994, Microsoft and Sybase announced an end to joint development. Each company decided to separately develop its own SQL Server products. Microsoft was free to evolve and change Microsoft SQL Server. Sybase decided to bring its System 10 products (and subsequent versions) to Windows NT—the first time that the Sybase-labeled SQL Server would be available for a Microsoft operating system. (The original agreement gave Microsoft exclusive rights on Microsoft operating systems.) Both companies' products would be backward-compatible with existing SQL Server applications—however, the products would diverge in the future and would have different feature sets and design points. Sybase's product would be fully compatible with its UNIX versions. Microsoft's product would continue to be integrated with Windows NT as much as possible. In short, the products would directly compete.

SQL Server Performance: No Secret Formulas

Although Microsoft SQL Server is designed for and optimized for Windows NT, it uses only publicly documented interfaces. From time to time, articles or speakers suggest that Microsoft SQL Server uses private, undocumented APIs in Windows NT to achieve its performance. But this assumption is false, without exception. SQL Server's performance results from using the available and published Windows NT services without any compromise. This is something that other products could also do if developers were willing to dedicate themselves to doing the best possible job for Windows NT without making compromises for portability to other operating systems.

NOTE

Although the Sybase product competed with Microsoft SQL Server, Microsoft encouraged and provided support for getting Sybase System 10 shipping on Windows NT as soon as possible because it was important to the acceptance and success of Windows NT. Such collaboration is typical; many such relationships are competitive on one level and cooperative on another.

Although version 6.0 was released to manufacturing in June 1995, on December 15, 1995, Microsoft shipped a feature-complete beta version of 6.5 to 150 beta sites. The production version of 6.5 was released to manufacturing in April 1996, a scant 10 months after 6.0 was released. The SQL Server team was not about to slow down.

[34](#)

The Secret of the Sphinx

Slowing down was never really an option. Even before SQL Server 6.5 was released, an entirely separate team of developers was hard at work on a brand new vision for Microsoft SQL Server. In 1993, Microsoft had decided that databases would be a core technology in the product line, and in late 1994, it began hiring expertise from DEC and other major vendors to work with Microsoft's Jet and SQL Server teams to plan components for a whole new generation of database technology. During 1995, the period when SQL Server 6.0 and SQL Server 6.5 were being released, this team was building a new query processor that was planned as a component for what was to become the Microsoft Data Engine (MSDE).

Development of the MSDE was enhanced by the simultaneous development of OLE DB, which allowed elements of the SQL Server core product to be developed as independent components. Components could then communicate with each other using an OLE DB layer. At the end of 1995, the new query-processing component was integrated into the SQL Server code base, and development of a brand new SQL Server, with the code name *Sphinx*, began in earnest. The team working on the query processor joined the rest of the SQL Server development team.

The development of this new generation of SQL Server had one overriding theme: to re-architect the entire database engine so that the SQL Server product could scale as much as its users wanted it to. This would mean upward growth to take full advantage of more and faster processors and as much memory as the operating system could handle. This growth would also have to allow for new functionality to be added to any of the components so that, for example, the query processor code could add an entirely new join algorithm to its repertoire as easily as a new hard drive could be added. In addition to the upward growth, SQL Server would also be expanded outward to support whole new classes of database applications. It would also need to scale downward to run on smaller systems such as desktops and laptops.

There were two immediate goals for the first version of this new re-architected system:

- Full row-level locking with an extremely smart lock manager.
- A brand-new query processor that would allow such techniques as distributed heterogeneous queries and efficient processing of ad hoc queries. (Ad hoc queries are a fact of life when you deal with data warehousing, Internet-based applications, or both.)

This brand-new SQL Server version 7.0 debuted in a limited beta 1 release in the fall of 1997. Beta 2 was made available to several hundred sites in December 1997. Because of the new architecture, all databases and the data structures they contained needed to be completely rebuilt during the upgrade process. Microsoft and the SQL Server development team were absolutely committed to making sure all customers would be successful in moving from SQL Server 6.5 to SQL Server 7.0. A program called the 1K Challenge was instituted, in which 1000 customers were invited to send copies of their databases to the SQL Server development team to be upgraded to version 7.0. A porting lab was set up in the building on the Redmond campus where the entire development team worked. Each week from February to August of 1998, four or five ISVs sent a team of their own developers to Microsoft to spend an entire week in the lab, making sure their products would work out of the box with the new SQL Server 7.0. The core SQL Server development engineers made themselves available to immediately isolate and fix any bugs encountered and to spend time

Finally, there was competitive pressure to make the next release bigger and better than originally planned. The Million Dollar Challenge issued by Larry Ellison of the Oracle Corporation pointed out a major piece of functionality that "they" had and "we" didn't. Adding materialized views to the product, which would allow SQL Server to meet this challenge, would be more than just a simple fix.

So the decision was made to have Shiloh be a full-blown release with at least an 18-month development cycle, but the version number would still be 7.5. The magnitude of the changes that would eventually find their way into this release was not immediately clear because the only absolute "must-have" at the outset was the cascading updates and deletes. It soon became clear that the release would grow beyond initial expectations. The team itself had grown substantially, spilling out of their original building on the main Microsoft campus into parts of two adjacent buildings. With a larger group of developers, a greater number of small to medium-size new features could be added to the product without seriously jeopardizing the planned release date. This book describes many of these new features.

In addition to having to add a substantial amount of new functionality, the development team also imposed upon themselves something they called "stretch goals." They declared an objective of a 20 percent increase in performance across the board, on all types of applications, but to give themselves something concrete to aim for, they extended this for specific applications. The main example of this was their goal to improve performance on the SAP R/3 Sales and Distribution benchmark by at least 40 percent. To do this, they had to make very specific changes in the optimizer that would directly affect the SAP queries but also benefit many other applications. At the Windows 2000 launch event in San Francisco on February 17, 2000, benchmark results were announced that demonstrated 6700 users for the Sales and Distribution benchmark, compared to 4500 users in SQL Server 7.0 for the same benchmark on the same hardware (an eight-way Pentium III-550). This represented a 48 percent improvement, and the goal had been well met.

After the development period was extended to a full 18 months, the decision was made to add another new feature. This decision was held in the strictest secrecy and wasn't even discussed with many high-level executives at Microsoft. The feature wasn't even addressed until after the release of Beta 1 in November 1999, and it was not announced publicly until the February launch of Windows 2000. Referred to internally by the code name *Coyote*, this secret project allowed SQL Server 2000 to support *distributed partitioned views* to allow unprecedented scalability in data management operations. It was this new feature that allowed the world-record-breaking benchmark results to be achieved and announced in San Francisco in February 2000.

Originally, these scalability features were scheduled for the version after Shiloh, but a close investigation revealed that many of the necessary components were already in place. These features included extensions to the optimizations of union views and the ability to update union views. Details of distributed partitioned views will be discussed later in the book.

The initial Beta 1 of Shiloh was released to a very small group of Early Adopters and serious beta testers in September 1999. Shortly after that, Microsoft announced that the official name of the product would be SQL Server 2000. There were two main reasons for this change. First, because of everything that was changing about the product, it was unrealistic to make it just a dot release (7.5); it really needed a whole new number. Second, if the product had gone with the name 8.0, it would be the only Microsoft BackOffice product that didn't use the name 2000. To conform to this companywide naming standard, the name SQL Server 2000 was announced. However, if you look at the internal version number using the @@VERSION function, you'll see that it returns the number 8.00.194.

It turns out that from the user's perspective, SQL Server 2000 introduces even more new functionality than did its immediate predecessor. SQL Server 7 had a completely rewritten engine, including brand-new storage structures, data access methods, locking technology, recovery algorithms, transaction log architecture, memory architecture, and optimizer. These, of course, are the things that this book describes in

detail. But for the end user, developer, or part-time database administrator (DBA), the external changes and language enhancements in SQL Server 7 were minimal. SQL Server 2000 adds dozens of new language features as well as drastic changes to existing objects such as table constraints, views, and triggers that all developers and most DBAs will need to know about.

Because the internal engine changes are minimal, only two betas were planned. Beta 2, released in April 2000, was the widespread public beta and was sent to thousands of interested users, conference attendees, ISVs, and consultants. The SQL Server 2000 development team froze the code at build 8.00.194.01, on August 6, 2000, and released the product to manufacturing on August 9, 2000.

Internal SQL Server development is a never-ending story. Just as after the release of SQL Server 7, new capabilities were already well past the design stage, ready to be implemented. Features that didn't make it into SQL Server 2000, as well as continued improvements in scalability, availability, interoperability, and overall performance, continue to be addressed. The development team is using feedback from the beta forums and from early users of the released version of SQL Server 2000 to shape the future of Microsoft SQL Server.