Craig S. Mullins

Return to Home Page

Winter 1994

The open enterprise survival guide

An Introduction to the Architecture of Sybase SQL Server

By Craig S. Mullins

Sybase SQL Server is unique among the most popular RDBMS products in that it was planned and designed to operate within a client/server architecture. Sybase, Inc., built SQL Server with the network in mind. Each client process establishes a connection to SQL Server over the network. Information is sent from client to server and back again over the network, using standard application programming interfaces (APIs).

Other similar RDBMS products were ported to a networked environment only after the client/server "boom" began. Having the network as an integral component of the SQL Server environment aids in its ability to support robust, production-level client/server development efforts.

Single-process, multi-threaded architecture
SQL Server is a multi-user, relational database server built on an open, client/server architecture providing high performance, high availability, and scalability for robust, production-level application development. Now what exactly does that imply?

DBMS software can be architected in one of two ways:

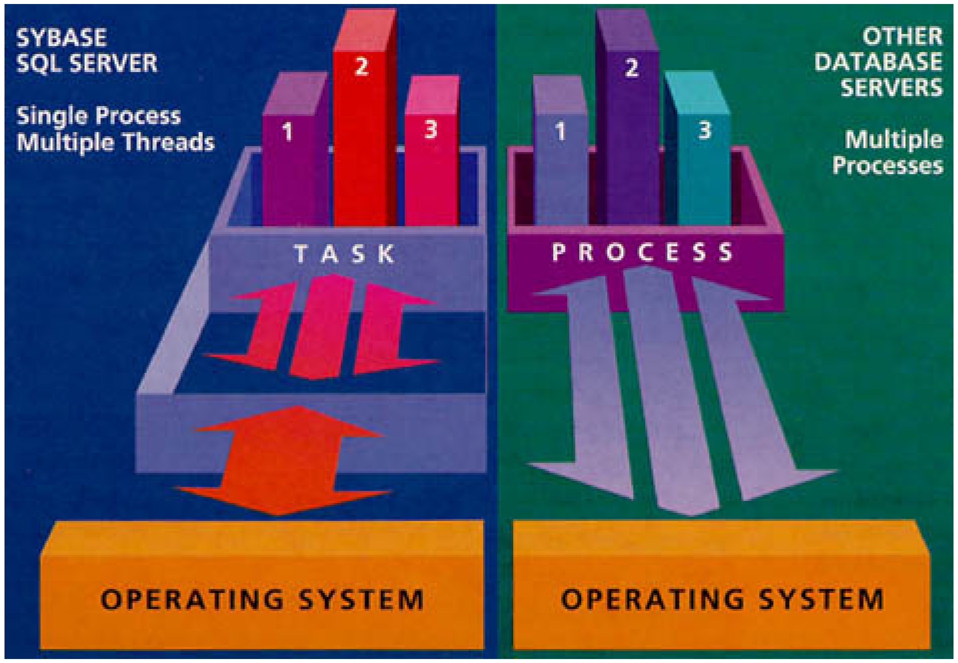
> **Single-process:** A product designed to use a single-process architecture single threads requests through the DBMS. This can produce bottlenecks in a multi-user environment. Single-process architectures are typical for single-user workstation DBMS products.
>
> **Multi-process:** A product designed to use a multi-process architecture creates new processes to handle each new request. Quick exhaustion of system resources (such as memory) can occur as more processes are initiated. Many multi-user DBMS products are multi-process architected.

SQL Server uses a single-process, multi-threaded architecture, employing multiple threads within a single process to service multiple users. Only one process will ever run on the server regardless of how many clients are connected. Employing this type of architecture reaps two benefits:

1. Context switching is more efficient because SQL Server handles it internally, instead of allowing it

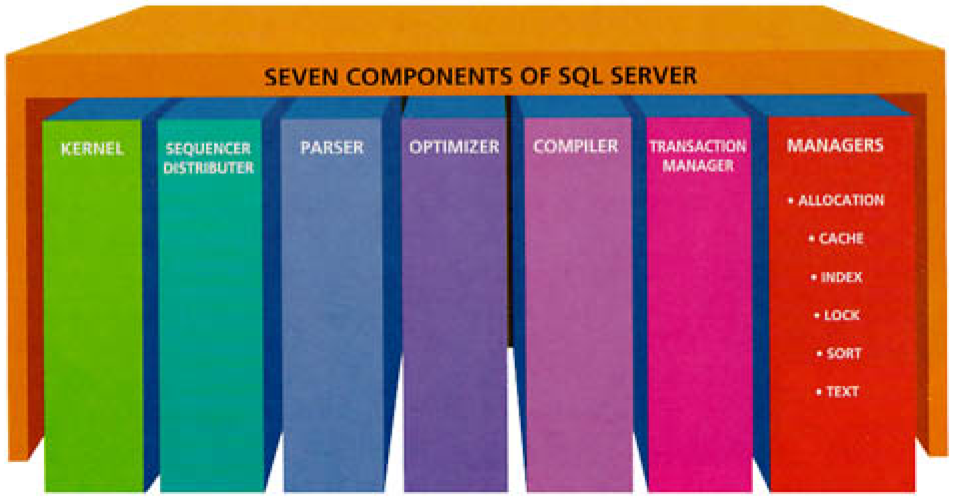
to be handled at the operating system level.

2. System resources are conserved and response time is minimized because a single process consumes fewer resources (i.e., less memory) than multiple processes.



Sybase has indicated, however, that future implementations may be architected differently in order to take better advantage of alternate platforms and their inherent threading capabilities. SQL Server manages data and executes queries against the data that it administers. Clients access the server, over the network, to request data. Sybase provides Open Server and Open Client to enable this type of processing. These are two collections of libraries containing APIs that enable the client and the server to communicate.

Open Client is required to enable client applications to send requests with SQL Server. Some SQL Server users still refer to Open Client by its older name, DB-Library. Open Server, on the other hand, is not required. Clients can communicate directly with SQL Server, or can interface with Open Server APIs to communicate to other DBMSs, data sources, and/or services.



SYBASE architecture components
To understand the architecture of SQL Server, it is helpful to decompose it into seven key components. These components operate in an integrated manner to provide support for the client tasks:

1. The *kernel* performs many jobs normally done by an operating system. It provides for such tasks as context switching, scheduling, I/O, and network

communications. Placing this type of functionality into the kernel yields two benefits:

- Supporting operations that are typically associated with the operating system within SQL Server eliminates operating system differences between platforms. This enables SQL Server to function (more or less) the same in each different environment, thereby promoting scalability.

- Inherent inefficiencies in each operating system can be overcome by providing very efficient routines specifically engineered to the manner in which SQL Server operates.

2. The *sequencer/distributor* controls application execution. When any application task requests database services, the sequencer/distributor analyzes the request, determines how it should be handled, and passes the request to the appropriate component for processing.

   Additionally, the sequencer/distributor controls the sequence of procedural execution. Because Transact-SQL can contain looping constructs and multiple SQL statements, the sequence in which statements are executed is important. As requests are received by the sequencer/distributor, it will do three things:

- perform authorization checking to ensure that the requester is authorized to access the data as requested
- sequence the request into internal structures called command trees
- pass the request on to the parser for subsequent processing.

The sequencer/distributor also manages the execution of stored procedures and sends portions of distributed queries to the appropriate database for processing.

3. The *parser* examines the SQL requests that are sent to SQL Server for processing. This is necessary to ensure that each statement is correct and can actually be executed. The parser examines SQL in two phases:

    Phase 1. The SQL is checked for syntax. The parser will ensure that all appropriate keywords are used and spelled accurately, that commas, periods, and punctuation are placed correctly; it also ensures the general syntactical validity of the entire SQL statement.

    Phase 2. The SQL statement is checked for semantic correctness. This is basically a final check asking, in essence, "Does this request make sense?" Semantic issues include:

- Do the objects that are being requested for access or modification exist?
- Do any objects that are being created not exist?
- Can the column be set to null (i.e., is it nullable)?

4. The *optimizer* is the inference engine that determines the best possible access strategy to satisfy SQL requests. The SQL Server optimizer is cost-based. Using statistical input (stored by SQL Server) on the tables being accessed, the optimizer chooses the best access method/join strategy to accommodate a given query. The optimizer makes several estimates based upon the statistics generated by the update statistics utility. If statistics have not been gathered, then the optimizer uses default statistics.

5. The *compiler* accepts the parsed and optimized query and compiles it into executable code based on the instructions forwarded to it by the optimizer.

6. The *transaction manager* component oversees all data modification operations. It is responsible for managing data inserts, updates, and deletes. The decision whether to use immediate or deferred update processing is made by the transaction

manager.

In order to ensure data validity and recoverability, the transaction manager maintains a write-ahead transaction log which can be used to recover data in the event of an error. The transaction manager also administers begin-and-end transaction requests.

7. Six managers within SQL Server perform the following tasks:

    The Allocation Manager manages disk allocation tasks such as handling inserts to full pages and adding space to a database.

    The Cache Manager, sometimes called the Buffer Manager, manages the data and procedure caches. Caching is used to retain information in memory so that when it is needed later, disk I/O can be avoided.

    The Index Manager manages the upkeep of SQL Server indexes. This includes index searching, modification, and space allocation.

    The Lock Manager administers locks for SQL Server resources thereby ensuring data integrity as users concurrently access and update SQL Server objects.

The Sort Manager is used to sort data as required by SQL Server. Operations that perform a sort include:

- the ordering of query results (ORDER BY)
- aggregation (GROUP BY)
- duplicate (UNION/DISTINCT)
- index creation

The Text/Image Manager manages the tasks of retrieving, searching, and manipulating text and data types.

Synopsis

The architecture of SYBASE SQL Server enables developers to create robust, production-level client/server applications. A basic understanding of the components of that architecture will aid developers to create optimally performing applications.

EDGE Magazine Online, published by PLATINUM *technology, inc.*

Home.