What's New in Sybase SQL Server™ Release 11.0?

Sybase SQL Server Release 11.0.x
Document ID: 36440-01-1100-02
Last Revised: December 15, 1995

Principal author: Server Publications Group

Document ID: 36440-01-1100

This publication pertains to Sybase SQL Server Release 11.0.x of the Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

APT-FORMS, Data Workbench, DBA Companion, Deft, GainExposure, Gain *Momentum*, Navigation Server, PowerBuilder, Powersoft, Replication Server, SA Companion, SQL Advantage, SQL Debug, SQL Monitor, SQL SMART, SQL Solutions, SQR, SYBASE, the Sybase logo, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. Adaptable Windowing Environment, ADA Workbench, AnswerBase, Application Manager, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, Bit-Wise, Client-Library, Client/Server Architecture for the Online Enterprise, Client/Server for the Real World, Client Services, Configurator, Connection Manager, Database Analyzer, DBA Companion Application Manager, DBA Companion Resource Manager, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, DirectCONNECT, Easy SQR, Embedded SQL, EMS, Enterprise Builder, Enterprise Client/Server, Enterprise CONNECT, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Gain Interplay, Gateway Manager, InfoMaker, Interactive Quality Accelerator, Intermedia Server, IQ Accelerator, Maintenance Express, MAP, MDI, MDI Access Server, MDI Database Gateway, MethodSet, Movedb, Navigation Server Manager, Net-Gateway, Net-Library, New Media Studio, OmniCONNECT, OmniSQL Access Module, OmniSQL Gateway, OmniSQL Server, OmniSQL Toolkit, Open Client, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open Solutions, PC APT-Execute,

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

Preface

1. New Features in SQL Server Release 11.0

2. Changes in SQL Server Release 10.0

3. Changes That May Affect Existing Applications

List of Tables

List of Figures

Preface

What's New in Sybase SQL Server Release 11.0? is an introduction to the new Sybase SQL Server™ features and the commands, system procedures, system tables, and documentation that support them.

Audience

This manual is intended for customers who are upgrading to SQL Server release 11.0 from release 10.0 or earlier.

How to Use This Book

This manual consists of the following chapters:

- Chapter 1, "New Features in SQL Server Release 11.0," describes the features that were added for release 11.0. All customers upgrading to SQL Server 11.0 should read this chapter.
- Chapter 2, "Changes in SQL Server Release 10.0," describes the features that were added for release 10.0. Customers upgrading from releases earlier than 10.0 should read this chapter.
- Chapter 3, "Changes That May Affect Existing Applications," describes the possible effects of product changes on existing applications. All customers upgrading to SQL Server 11.0 should read this chapter.

Related Documents

Other manuals that you may find useful are:

- SQL Server installation and configuration guide, which describes the installation procedures for SQL Server and documents operating system-specific system administration, security administration, and tuning tasks.
- *SQL Server Reference Manual*, which contains detailed information about Transact-SQL® commands and system procedures.
- *SQL Server Performance and Tuning Guide*, which explains how to tune SQL Server for maximum performance. The book includes information about database design issues that affect performance, query optimization, how to tune SQL Server for

very large databases, disk and cache issues, and the effects of locking and cursors on performance.

- *SQL Server Reference Supplement*, which contains a list of Transact-SQL reserved words, definitions of system tables, a description of the *pubs2* sample database, a list of SQL Server error messages, and other reference information that is common to all the SQL Server manuals.
- *SQL Server Security Administration Guide*, which explains how to use the security features provided by SQL Server to control user access to data. The manual includes information about how to add users to the server, give them controlled access to database objects and procedures, and manage remote servers.
- *SQL Server Security Features User's Guide*, which explains how to use the security features of SQL Server.
- *SQL Server System Administration Guide*, which provides in-depth information about administering servers and databases. The manual includes instructions and guidelines for managing physical resources and user and system databases and for specifying character conversion, international language, and sort order settings.
- SQL Server utility programs manual, which documents the Sybase utility programs such as **isql** and **bcp** that are executed at the operating system level.
- *Transact-SQL User's Guide*, which documents Transact-SQL, Sybase's enhanced version of the relational database language. It serves as a textbook for beginning users of the database management system.

1 New Features in SQL Server Release 11.0

Introduction

This chapter describes the new features for SQL Server release 11.0, the new commands, system procedures, and system tables added to support these features, and the reorganization of the product documentation set.

Features Added for Release 11.0

Following are the features added for release 11.0:

Changes to Commands, System Procedures, and System Tables

The last section of this chapter, “Changes to Commands, System Procedures, and System Tables” on page 1-18, provides a summary of changes as follows:

Documentation Reorganization

This section describes the new manuals added for release 11.0 and the changes to existing manuals.

User-Defined Caches

Release 11.0 allows System Administrators to split the SQL Server data cache into separate named data caches and to bind databases or database objects to those caches. Also, System Administrators can create pools within caches that perform large I/O to disk, improving performance for many queries.

Configuring Caches

Named data caches are created with the new system procedure **sp_cacheconfig**. Memory pools for large I/O are configured within data caches with **sp_poolconfig**. Configuring caches and pools can also be done by editing a configuration file.

Entities are bound to caches with **sp_bindcache** and unbound with **sp_unbindcache** or **sp_unbindcache_all**. The system procedure **sp_helpcache** provides information about caches and cache bindings. **sp_cacheconfig** also provides information about caches and the pools within caches.

See Chapter 9, “Configuring Data Caches,” in the *System Administration Guide* for instructions on configuring named caches and binding database objects to them.

Binding Objects to User-Defined Caches

In general, you can increase performance by binding the following objects to their own named caches:

- *sysindexes* table and its index

  Bind these two objects to the same named cache. This is the most frequently used table and its index. However, you do not have to make its named cache very large.

- *syslogs* table

  This helps reduce the contention on the buffer manager spinlocks. You should also configure a 4K memory pool in its named cache to benefit from the larger log I/O size, as defined by **sp_logiosize**.

- *tempdb* database

  This helps performance if many temporary or work tables are generated by your applications.

In addition to the above objects, the most frequently used tables and indexes (based on your database and application designs) are good candidates for binding to their own caches. However, if you have only one table with no index, binding that table to a named cache will not improve performance.

➤ ***Note***

Having too many named caches without increasing the total memory available to your server is generally not a good idea. Each additional named cache reduces the amount of memory available in the default cache.

Cache Strategies

SQL Server release 11.0 provides new optimization strategies and new commands to control the use of those strategies for objects, sessions, and queries. Figure 1-1 shows the two cache strategies.

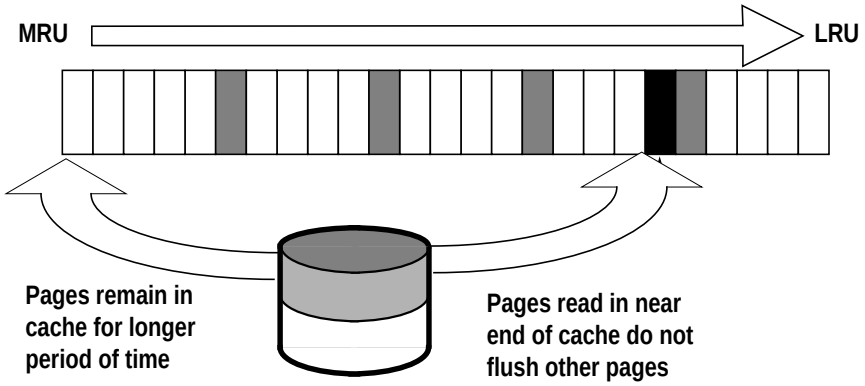

Figure 1-1: Cache strategy choices

SQL Server can:

- Read pages in at the start of the MRU (most recently used) chain in order to keep the pages in cache so that they can be accessed many times without performing additional I/O
- Read pages in near the end of the cache so that they will not force other pages out of the cache

See Chapter 15, "Memory Use and Performance," in the *Performance and Tuning Guide* for performance information.

Large I/O

When caches are created, all space is assigned to a 2K pool. This pool can be split to allow the cache to perform large I/O, reading up to eight data pages at once. Since most I/O time is spent performing queueing, seeking, and positioning operations, large I/O can greatly increase performance for queries that scan entire tables or ranges of tables.

SQL Server can also perform large I/O on the transaction log.

Changing the Log I/O Size

Using the new **sp_logiosize** system procedure, you can change the log I/O size used by SQL Server. This can improve performance by reducing the number of times that SQL Server writes transaction log pages to disk. The value you specify for **sp_logiosize** must correspond

to an existing memory pool configured for the cache used by the database's transaction log.

By default, SQL Server defines the log I/O size of a database as 4K. If the transaction log for a database is bound to the default cache or a user-defined cache that does not contain a 4K memory pool, SQL Server sets the log I/O size to 2K (a 2K memory pool is always present in any cache). For most workloads, a log I/O size of 4K performs much better than one of 2K, so each cache used by a transaction log should have a 4K memory pool.

New Query Tuning Options

New query tuning options support the capabilities of named caches and provide additional control when tuning queries:

- The **set forceplan on** command forces SQL Server to join the tables in a query in the order in which they are listed in the **from** clause.
- The **set table count** command allows you to specify the number of tables that are considered at once as joins are optimized.
- The new system procedure **sp_cachestrategy** disables and re-enables cache strategies and large I/O for individual objects.
- New clauses for **select**, **delete**, and **insert** commands allow you to specify the index, cache strategy, or I/O size for a query.
- The **set prefetch** command allows you to enable or disable large I/O for a session.

See Chapter 9, "Advanced Optimizing Techniques," in the *Performance and Tuning Guide* for information on query processing options that control caching strategies and I/O size.

Data Storage Changes

This section describes changes to the way SQL Server stores data.

Maximum Number of Rows per Page

You can now configure the maximum number of rows stored on a data page or index leaf page. The **max_rows_per_page** value specified in a **create table**, **create index**, or **alter table** command restricts the number of rows allowed on a data page, a clustered index leaf page, or a

nonclustered index leaf page. This reduces lock contention and improves concurrency for frequently accessed tables.

The **max_rows_per_page** value applies to the data pages of an unindexed table or the leaf pages of an index. The value of **max_rows_per_page** is stored in the *maxrowsperpage* column of *sysindexes*. (The *maxrowsperpage* column was named *rowpage* in previous releases of SQL Server.)

Unlike **fillfactor**, which is not maintained after creating a table or index, SQL Server retains the **max_rows_per_page** value when adding or deleting rows. For information about how to use **max_rows_per_page** to reduce lock contention and improve concurrency for your server, see the *Performance and Tuning Guide*.

Page Allocation

SQL Server 10.0 and earlier releases searched the OAM page chain for unused pages before allocating new extents to objects. Release 11.0 provides a system-wide configuration parameter, **page utilization percent**, that allows SQL Server to allocate a new extent to an object without searching the OAM page chain, depending on the ratio of used and unused pages in the table to the table's allocated extents.

Partitioned Tables

By default, SQL Server stores a **heap table**'s data in one doubly linked chain of database pages. When a transaction inserts a row into the table, it holds an exclusive page lock on the last page of the page chain while inserting the row. As multiple transactions attempt to insert rows into the same table at the same time, performance problems can occur. Because only one transaction at a time can obtain an exclusive lock on the last page, other, concurrent insert transactions block, as shown in Figure 1-2.

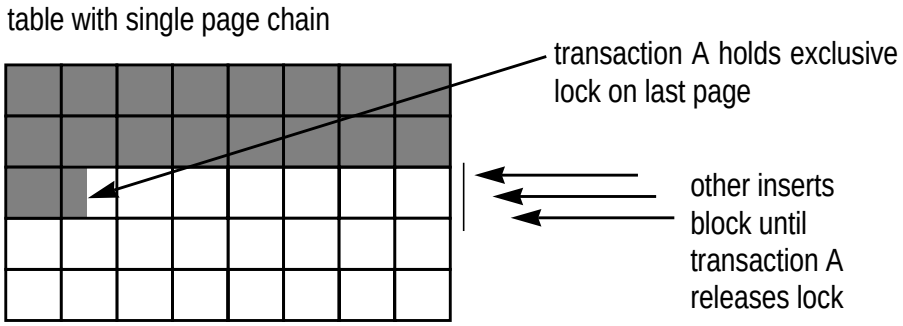

Figure 1-2: Page contention during inserts

In release 11.0, SQL Server provides the ability to partition heap tables. A **partition** is simply another term for a page chain. Partitioning a table creates multiple page chains (partitions) for the table and, therefore, multiple last pages for insert operations.

When a transaction inserts data into a partitioned table, SQL Server randomly assigns the transaction to one of the table's partitions. Concurrent inserts are less likely to block, since multiple last pages are available for inserts, as shown in Figure 1-3.

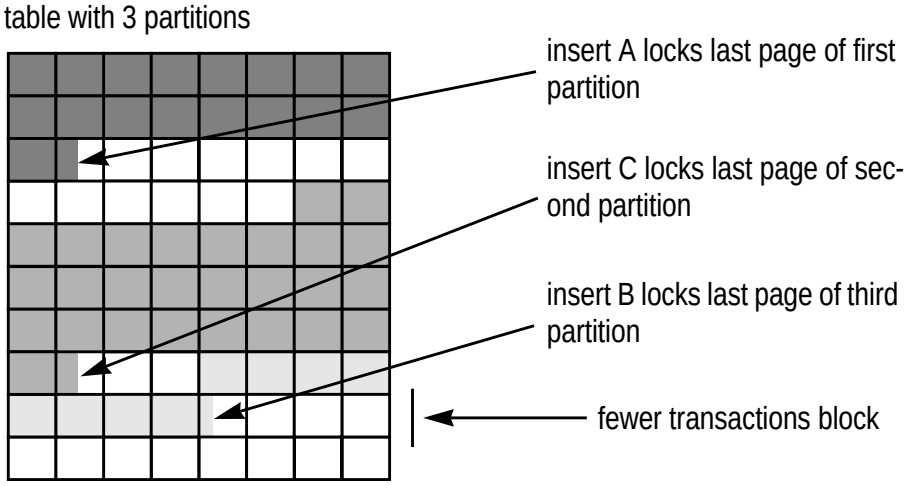

Figure 1-3: Addressing page contention with partitions

If a table's segment spans several physical disks, SQL Server distributes the table's partitions across those disks when you create the partitions. This can improve I/O performance when SQL Server

writes the table's cached data to disk, since the I/O is distributed over several devices.

SQL Server release 11.0 manages partitioned tables transparently to users and applications. Partitioned tables appear exactly like unpartitioned tables, except when accessed via the **dbcc checktable** and **dbcc checkdb** commands or when viewed with the new **sp_helpartition** procedure.

See "Partitioning and Unpartitioning Tables" in Chapter 13, "Controlling Physical Data Placement" of the *Performance and Tuning Guide* for information about how to create and administer partitioned tables.

Transaction Log Changes

This section describes the changes made to the transaction log for release 11.0.

User Log Caches

There is one user log cache for each configured user connection. SQL Server uses these user log caches to buffer the user transaction log records, which reduces the contention at the end of the transaction log. When a user log cache becomes full or when another event occurs (such as when the transaction completes), SQL Server "flushes" all log records from the user log cache to the database transaction log. You can configure the size of all user log caches for your server using the **user log cache size** parameter of **sp_configure**; the default size is 2048 bytes.

Since more than one process can access the contents of a user log cache when determining a possible "flush," SQL Server uses spinlocks to protect the user log caches. A spinlock is an internal locking mechanism that prevents a second process from accessing the data structure being used by another process. SQL Server uses the **user log cache spinlock ratio** parameter of **sp_configure** to specify the ratio of user log caches per user log cache spinlock; the default is 20 user log caches for each spinlock.

For information about configuring **user log cache size** and **user log cache spinlock ratio**, see Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide*.

syslogshold Table

You can query the new *syslogshold* system table to determine the oldest active transaction in each database. *syslogshold* exists in the *master* database, and each row in the table represents either:

- The oldest active transaction in a database or
- The Replication Server® truncation point for the database's log.

A database may have no rows in *syslogshold*, a row representing one of the above, or two rows representing both of the above. For information about how Replication Sever truncation points affects the truncation of a database's transaction log, see your Replication Server documentation.

Querying *syslogshold* can help you when the transaction log becomes too full, even with frequent log dumps. The **dump transaction** command truncates the log by removing all pages from the beginning of the log up to the page that precedes the page containing an uncommitted transaction record (the oldest active transaction). The longer this active transaction remains uncommitted, the less space is available in the transaction log, since **dump transaction** cannot truncate additional pages.

For information about how to query *syslogshold* to determine the oldest active transaction that is holding up your transaction dumps, see Chapter 19, "Backing Up and Restoring User Databases," in the *System Administration Guide*.

Isolation Level 0

You can specify isolation level 0 for the queries in transactions along with the isolation levels 1 and 3 supported in release 10.0. Isolation level 0 prevents other transactions from changing data that has already been modified by an uncommitted transaction. The other transactions are blocked from modifying that data until the transaction commits. However, other transactions can still read the uncommitted data (known as **dirty reads**).

Queries executing at isolation level 0 do not acquire any read locks for their scans, so they do not block other transactions from writing to the same data (and vice versa). Applications that are not impacted by dirty reads may see better concurrency and reduced deadlocks when accessing the same data using isolation level 0. However, transactions that require data consistency should **not** use isolation level 0.

You can specify isolation level 0 for the transactions of a session as follows:

```
set transaction isolation level 0
```

Since this command makes all queries execute at isolation level 0, you should be wary of using it for any transaction that requires data consistency. Instead, you can selectively choose isolation level 0 for a query in a transaction using the **at isolation** clause as follows:

```
select *
from titles
at isolation read uncommitted
```

For information about transactions and isolation levels, see Chapter 17, "Transactions: Maintaining Data Consistency and Recovery," in the *Transact-SQL User's Guide* and "Transactions" in the *SQL Server Reference Manual*.

Lock Manager Changes

This section describes the changes to locking behavior.

Table, Page, and Address Lock Tables

SQL Server manages the acquiring and releasing of table locks by using an internal hash table with 101 rows (known as **hash buckets**) and of page and address locks by using internal hash tables with 1031 rows each. In release 11.0, these tables use one or more spinlocks to serialize access between processes that are running on different engines. A spinlock is an internal locking mechanism that prevents a second process from accessing the resource currently being used by another process. All processes trying to access the resource must wait (or **spin**) until the lock is released.

For SQL Servers running with multiple engines, you can set the ratio of spinlocks protecting each hash table using the **table lock spinlock ratio**, **page lock spinlock ratio**, and **address lock spinlock ratio** configuration parameters. The table locks hash table default ratio is 20 rows for each spinlock, and the page or address locks hash table default ratio is 100 rows for each spinlock. A SQL Server configured with only one engine has only one spinlock for each hash table, regardless of the values specified for these parameters.

For information about configuring the **table lock spinlock ratio**, **page lock spinlock ratio**, and **address lock spinlock ratio** parameters, see Chapter 11,

"Setting Configuration Parameters," in the *System Administration Guide*.

Deadlock Checking

SQL Server performs deadlock checking after a minimum period of time for any process that is waiting (or sleeping) for a lock to be released. By default, this minimum period of time is 500 ms. Previous releases of SQL Server perform this deadlock check at the time the process begins to wait for a lock. This deadlock checking is a time-consuming overhead for applications that wait without a deadlock.

You can change the minimum amount of time (in milliseconds) that a process must wait before it initiates a deadlock check using the **deadlock checking period** configuration parameter. If you expect your applications to deadlock infrequently, you can delay deadlock checking even further and reduce the overhead cost. However, configuring **deadlock checking period** to a higher value produces longer delays before deadlocks are detected.

For information about deadlock checking, see Chapter 11, "Locking on SQL Server," in the *Performance and Tuning Guide*.

New Engine Freelock Lists

In release 11.0, when a process that is running on a multi-engine SQL Server requests a lock, it looks for one in its engine's freelock list. If the engine freelock list is out of locks, SQL Server moves a certain number of locks from its global freelock list to the engine freelock list. For single-engine SQL Servers, the entire global freelock list is moved to the engine freelock list at server start-up time.

After an engine completes a process, all locks held by that process are released and returned to that engine's freelock list. This reduces the contention of each engine accessing the global freelock list. However, if the number of locks released to the engine exceed the maximum number of locks allowed in the engine's freelock list, SQL Server moves a number of locks to the global freelock list. This replenishes the number of locks that are available to other engines from the global list.

You can configure the maximum number of locks available to the engine freelock lists as a percentage of the total number of locks available to your server with the **max engine freelocks** configuration parameter. You can also configure the number of locks transferred

back and forth between the engine and global freelock lists using the **freelock transfer block size** parameter. For information about these parameters, see Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide*.

Increasing the Engine Freelock Lists

Each engine's freelock list is a portion of the global freelock list allocated to the engines, defined by the **max engines freelocks** configuration parameter as a percentage of the total number of locks. By default, SQL Server assigns 10 percent of the total number of locks to all the engines' freelock lists; the remainder stays in the global freelock list. For example, if your server is configured with 5000 locks, 500 locks are distributed evenly between each engine freelock list, and 4500 remain in the global freelock list.

For most SQL Server configurations, you can double the percentage of locks assigned to the engine freelock lists and improve the performance of your applications.

Housekeeper Task

When SQL Server has no user tasks to process, a housekeeper task automatically begins writing dirty buffers from cache to disk. Because these writes are done during the server's idle cycles, they are known as **free writes**.

The benefits of the housekeeper task are:

- Improved CPU utilization
- Decreased need for buffer washing during transaction processing
- Faster checkpoints
- Shorter recovery time

In applications that repeatedly update the same database page, the housekeeper task may initiate some database writes unnecessarily. System Administrators can use the **housekeeper free write percentage** configuration parameter to disable the housekeeper task or to control its side effects.

For more information about the housekeeper task, see Chapter 17, "Using CPU Resources Effectively," of the *Performance and Tuning Guide*.

SQL Server Configuration

In previous releases, SQL Server was configured using **sp_configure, reconfigure,** and a number of undocumented **dbcc tune** and **buildmaster -y** parameters. In order to provide a single entry point for all SQL Server configuration (except for Buffer Manager), **sp_configure** has been redesigned to incorporate the configuration parameters previously implemented with **dbcc tune** and **buildmaster -y.** In addition, a number of new configuration parameters have been implemented.

For information about which old configuration names have new names, see "Changes in SQL Server Configuration" on page 3-2.

The **sp_configure** interface has been redesigned to include these new features:

- Display levels, which allow the user to select one of three views of the configuration parameters:
  - Basic level, appropriate for general tuning of SQL Server
  - Intermediate level, somewhat more inclusive than the Basic level
  - Comprehensive level, appropriate for the most detailed level of SQL Server tuning
- Parameter hierarchy, which organizes the configuration parameters according to the area of SQL Server behavior to which they pertain.
- Configuration files, which allow users to replicate specific configurations, validate configurations before setting them, and create multiple configurations that can be easily switched.

For information about SQL Server configuration, see Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide*.

Lock Escalation

In previous versions of SQL Server, once a statement accumulated more than 200 page locks on a table, SQL Server tried to issue a table lock on that object. If the table lock succeeded, the page locks were no longer necessary and were released.

The **lock promotion HWM**, **lock promotion LWM**, and **lock promotion PCT** configuration parameters and the **sp_setpglockpromote** system procedure configure the number of page locks that SQL Server

acquires on a table before it attempts to escalate to a table lock on a server-wide, per database, and per table basis.

For more information, see Chapter 11, "Locking on SQL Server," in the *Performance and Tuning Guide*.

Multiple Network Engines

If your symmetric multiprocessing (SMP) system supports network affinity migration, each SQL Server engine handles the network I/O for its connections in release 11.0. During login, SQL Server migrates the client connection task from engine 0 to the engine that is servicing the smallest number of connections. The client's tasks run network I/O on that engine until the connection is terminated.

By distributing the network I/O among its engines, SQL Server can handle more user connections. The per-process limit on the maximum number of open file descriptors no longer limits the number of connections. Adding more engines linearly increases the maximum number of file descriptors. For more information about how multiple network engines work in an SMP environment, see Chapter 10, "Managing Multiprocessor Servers," in the *System Administration Guide*.

Improvements to *showplan*

The output for the **set showplan** command has been improved as follows:

- Indents and delimiters have been added to help readability.
- Line numbers and statement numbers help debug long batches and procedures.
- Additional messages include the keys used on indexes and messages that provide more information about access methods.
- Subquery types, nesting levels, and other subquery information help check subquery performance.
- Messages have been added for the new release 11.0 features to display the I/O size and the caching strategy.

In the following example, there is an index named *title_ix* on the *titles* table:

```
select title_id, price
    from titles
    where title = "Computers and Privacy"
```

This is the pre-11.0 output from **showplan**:

```
STEP 1
The type of query is SELECT.
FROM TABLE
titles
Nested iteration
Index : title_ix
```

This is the release 11.0 output from **showplan**:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).


    STEP 1
        The type of query is SELECT.

        FROM TABLE
            titles
        Nested iteration.
        Index : title_ix
        Ascending scan.
        Positioning by key.
        Keys are:
            title
        Using I/O Size 2 Kbytes.
        With LRU Buffer Replacement Strategy.
```

All **showplan** messages are documented in Chapter 8, "Understanding Query Plans," of the *Performance and Tuning Guide*.

Query and Data Modification Changes

This section describes the changes made to subquery processing and update behavior.

Subquery Changes

The behavior of certain subqueries has changed. See Chapter 3, "Changes That May Affect Existing Applications," for detailed information about these changes and how they may affect your existing applications.

Update Changes

Update strategies for SQL Server have improved. More updates are now made in place or directly, with fewer writes to the transaction log. See the *Performance and Tuning Guide* for more information on performance and update operations.

Upgrading Database Dumps

When a SQL Server installation is upgraded to a new release, all databases associated with that server are automatically upgraded.

As a result, the database and transaction log dumps created with any previous SQL Server must be upgraded before they can be used with the current SQL Server.

SQL Server release 11.0 provides an automatic upgrade mechanism, on a per-database basis, for upgrading a database or transaction log dump from any SQL Server release 10.0 to the current SQL Server, thus making the dump compatible for use. This mechanism is entirely internal to SQL Server release 11.0 and requires no external programs. It provides the flexibility of upgrading individual dumps as needed.

The following tasks are not supported by this automatic upgrade functionality:

- Loading a SQL Server release 10.0 *master* database onto release 11.0.
- Installing new or modified stored procedures. Continue to use **installmaster.**
- Loading and upgrading dumps prior to SQL Server release 10.0.

Associated with this feature is a new **online database** command. This command marks a database available for public use after a normal load sequence, and if needed, upgrades a loaded database and/or transaction log dumps to the current version of SQL Server.

For more information on upgrading database dumps, see Chapter 19, "Backing Up and Restoring User Databases," in the *System Administration Guide* and the **load database** and **dump database** commands in the *SQL Server Reference Manual.*

For more information on the **online database** command, see the *SQL Server Reference Manual.*

Tape Device Determination by Backup Server

To become less device dependent and more flexible, Backup Server™ provides a method for determining the tape device characteristics (tape positioning for read, close, append, I/O size, file marks, and ability to overwrite a tape mark) for a dump operation.

When you issue a **dump database** or **dump transaction** command, Backup Server checks to see whether the device type of the specified dump device is known (supplied and supported internally) by SQL Server. If the device is not a known type, Backup Server checks the tape configuration file (default location is *$SYBASE/backup_tape.cfg)* for the device configuration. If the configuration is found, the **dump** command proceeds.

If the configuration is not found in the tape device configuration file, the **dump** command fails with the following error message:

```
Device not found in configuration file. INIT needs
to be specified to configure the device.
```

This means the device needs to be configured. Issue the **dump database** or **dump transaction** with **init** qualifier to configure the device. Using operating system calls, Backup Server attempts to determine the device's characteristics, and if successful, stores the device characteristics in the tape configuration file.

If Backup Server cannot determine the dump device characteristics, it defaults to one dump per tape. The device can not be used if the configuration fails to write at least one dump file.

Tape configuration by Backup Server applies only to UNIX platforms.

For more information on tape device determination and the tape configuration file, see Chapter 19, "Backing Up and Restoring User Databases," in the *System Administration Guide.*

IDENTITY Column Changes

The IDENTITY column feature allows you to create a column with system-generated values that uniquely identify each row in a table. Release 11.0 provides the following enhancements to IDENTITY columns:

- For tables with no unique indexes, the **identity in nonunique index** database option allows you to create a unique index by including

an IDENTITY column in the index key. Unique indexes are required for isolation level 0 reads and updatable cursors.

- The **identity grab size** configuration parameter allows each SQL Server process in a multiprocessor environment to reserve a specified number of IDENTITY column values when you add rows to tables that contain IDENTITY columns.
- The **size of auto identity** configuration parameter sets the precision of the IDENTITY columns that SQL Server generates for tables created in databases where the **auto identity** database option is set to **true**.

For more information about the enhancements to IDENTITY columns, see "IDENTITY Columns" in the *SQL Server Reference Manual*.

New *text* and *image* Global Variables

The following five global variables have been added to release 11.0 to facilitate inserting or updating *text* and *image* data through Open Client™ applications:

- *@@textcolid*
- *@@textdbid*
- *@@textobjid*
- *@@textptr*
- *@@textts*

For information about these global variables, see "Variables (Local and Global)" in the *SQL Server Reference Manual*. For information about how to use these variables in a procedure for Open Client applications, see the *Client-Library/C Reference Manual*.

Changes to Commands, System Procedures, and System Tables

This section describes the changes to commands, system procedures, and system tables that were made to support new release 11.0 features.

New *online database* Command

The following table summarizes the new **online database** command:

Table 1-1: New command

Name	Function
online database	Makes a database available for public use after a normal load sequence and, if needed, upgrades a loaded database and transaction log dumps to the current version of SQL Server.

Changes to Existing Commands

The following table summarizes the changes to existing commands:

Table 1-2: Changes to existing commands

Name	Change
alter table	The new **partition** clause allows you to create additional page chains for a table with no clustered index. The new **unpartition** clause allows you to concatenate all page chains for a partitioned table.
buildmaster	The **buildmaster -r** flag is no longer supported.
delete	New clauses allow specifying the index, cache strategy, and I/O size for a query.
reconfigure	The **reconfigure** command is no longer required to activate changes made with **sp_configure**. **reconfigure** is now non-operational. Scripts using **reconfigure** will still run, but should be changed at your earliest convenience, as **reconfigure** may not be supported in future releases.
select	New clauses allow specifying the index, cache strategy, and I/O size for a query. The new **at isolation** clause allows you to specify the isolation level for the query.
set showplan	Changes to output.
set transaction isolation level	New isolation level option: 0 (zero)
update	New clauses allow you to specify the index, cache strategy, and I/O size for a query.

New *set* options

The following table summarizes the new **set** options:

Table 1-3: New set options

Option	Function
forceplan	Forces the optimizer to perform joins in the order in which tables are named in the **from** clause of a query.
prefetch	Disables or enables large I/O (data prefetch) for a session.
statistics subquerycache	Prints statistics about the use of the subquery cache during the execution of a subquery.
table count ***integer***	Specifies the number of tables that are to be optimized at the same time in a join query.

New System Procedures

The following new system procedures provide some of the functionality for the features described earlier in this summary:

Table 1-4: New system procedures

Procedure Name	Function
sp_bindcache	Binds databases, tables, indexes, or *text* or *image* chains to data caches.
sp_cacheconfig	Configures named data caches and provides information about caches.
sp_cachestrategy	Enables and disables caching strategies and large I/O for specific tables and indexes.
sp_chgattribute	Changes the **max_rows_per_page** value for future space allocations of a table or index.
sp_displaylevel	Sets and displays a user's display level. The display level determines which SQL Server configuration parameters are displayed in **sp_configure** output.
sp_dropglockpromote	Removes lock promotion values from a table or database.
sp_helpcache	Provides information about cache overhead requirements and provides information about caches and cache bindings.

Table 1-4: New system procedures (continued)

Procedure Name	Function
sp_helpartition	Lists the first page and the control page for each partition of a partitioned table.
sp_logiosize	Changes the log I/O size used by SQL Server to a different memory pool when doing I/O for the transaction log of the current database.
sp_poolconfig	Configures pools within named caches to enable large I/O.
sp_procqmode	Reports the subquery processing mode of an object.
sp_setpglockpromote	Sets or changes the lock promotion thresholds for a database, table, or for SQL Server.
sp_unbindcache	Unbinds a specific database, table, index, or *text* or *image* object from a data cache.
sp_unbindcache_all	Unbinds all objects bound to a cache.

Changes to System Procedures

The following system procedures have been changed in this release:

Table 1-5: Changes to system procedures

Procedure Name	Change
sp_dboption	Has a new database option, **identity in nonunique index,** that allows you to create a unique index by including an IDENTITY column in the index key.
sp_configure	Includes new parameters and new optional subcommands, and displays new output.

New System Tables

The following system tables are new in this release:

Table 1-6: New system tables

Table Name	Function
sysattributes	Defines attributes for objects such as databases, tables, indexes, users, logins, and procedures.

Table 1-6: New system tables (continued)

Table Name	Function
syslogshold	Stores information about the oldest active transaction and the Replication Server truncation point for each database
syspartitions	Stores internal information about table partitions.

Changes to Existing System Tables

The following system tables have been changed in this release:

Table 1-7: Changes to system tables

Table Name	Change
sysindexes	Changed column: *rowpage* to *maxrowsperpage* The *root* column entry for a table becomes obsolete when the table is partitioned. Root pages for individual partition are derived from information in the new *syspartitions* table.
sysconfigures	Added new columns: • *name* contains the configuration parameter name • *parent* contains the group to which the parameter belongs • *value2* is used for configuration parameters whose values are character strings • *value3* is not currently used

Table 1-7: Changes to system tables (continued)

Table Name	Change
syscurconfigs	Added new columns: • *value2* is used for configuration parameters whose values are character strings • *defvalue* contains default values for configuration parameters • *minimum_value* contains the minimum legal value for a configuration parameter • *maximum_value* contains the maximum legal value for a configuration parameter • *memory_used* contains the per-unit amount of memory used by a configuration parameter • *display level* contains the display level associated with a configuration parameter • *datatype* and *message_num* are for internal use
sysdatabases	Added new column: • *offline* status bit is used by the upgrading database dumps feature

Documentation Reorganization

The SQL Server documentation set has been reorganized for release 11.0. This reorganization has resulted in the following changes:

- The new *SQL Server Performance and Tuning Guide* explains how to tune SQL Server for maximum performance. The book includes information about database design issues that affect performance, query optimization, how to tune SQL Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance. This information was previously in the *System Administration Guide.*
- The new *SQL Server Reference Supplement* includes the Transact-SQL reserved words, definitions of system tables, a description of the *pubs2* sample database, a list of SQL Server error messages, and other reference information. Much of the material in this book was previously included as appendixes in the individual manuals.
- The new *SQL Server Security Administration Guide* explains how to use the security features provided by SQL Server to control user

access to data. This manual includes information about how to add users to the server, give them controlled access to database objects and procedures, and manage remote servers. Some of its contents previously appeared in the *System Administration Guide.*

- The new *SQL Server Security Features User's Guide* explains how to use the security features of SQL Server. It includes information about permissions that was formerly in the *Transact-SQL User's Guide.*
- The *System Administration Guide Supplement* is no longer available. The information that was in this book is now part of an expanded, platform-specific SQL Server installation and configuration guide.
- Volume 1 of the *SQL Server Reference Manual* has been reorganized. Chapter 1, "SQL Server Roadmap," tells you which commands, system procedures, and functions to use to accomplish particular tasks. Information about datatypes has been consolidated into Chapter 2, "System and User-Defined Datatypes." The "Transact-SQL Commands," "Transact-SQL Functions," and "Transact-SQL Topics" chapters follow.

2 Changes in SQL Server Release 10.0

Introduction

This chapter describes the features that were introduced in Sybase SQL Server release 10.0. If you are upgrading from a pre-10.0 release of SQL Server, these features will be new to you in release 11.0:

The last section of this chapter provides a summary of release 10.0 changes, as follows:

New Installation and Upgrade Utility

A new installation program, **sybinit**, provides a flexible, menu-driven tool for managing configuration of SQL Server and Backup Server. It can be used interactively or with a resource file, if you need to install or upgrade many servers.

sybinit allows you to:

- Install a new release 10.0 SQL Server or Backup Server
- Upgrade an existing SQL Server from release 4.8 or later to release 10.0
- Modify an existing SQL Server 10.0 to install and activate auditing
- Install languages and character sets
- Configure your default language, sort order, or character set
- Install other System 10™ products

It also helps manage your system's interfaces file, providing a convenient tool for adding, deleting or changing interfaces file entries.

See your SQL Server installation and configuration guide for information on running **sybinit.**

Addition to System Databases

Release 10.0 provides many new features, including more than 25 new system procedures. In addition, the catalog stored procedures (formerly installed separately) are now included in the default installation procedures.

Because the new system procedures and system tables nearly double the size of the *master* database, release 10.0 stores system procedures in a new database, *sybsystemprocs*. This saves customers the trouble of repartitioning disks and moving the *master* database to ensure that the entire database resides on a single device.

When your installation or upgrade is complete, Sybase system procedures reside in the new *sybsystemprocs* database. Special search routines for executing any procedure name beginning with "sp_" allow SQL Server to locate the system procedures, even if the procedure call is fully qualified with the database name "master." For example, **master..sp_who** finds the procedure.

Benefits of Moving System Procedures

The new structure provides several benefits:

- It dramatically shrinks the space used in *master*. You will not have to change the size of the *master* database for a long time.
- It creates greater flexibility for users. Users can store larger numbers of local stored procedures in *sybsystemprocs* without worrying about space considerations on the master device.
- It can speed recovery of the *master* database. Under the pre-10.0 structure, if the master device was damaged, a significant portion of recovery time involved the re-creation of the system procedures. Now, if the system procedures are on a separate database device, *master* can be recovered much more quickly.
- The new *sybsystemprocs* database can be restored quickly and easily in case of disk failure without affecting the *master* database.
- The *sybsystemprocs* database can safely use an operating system file, if necessary, as a database device. It is rarely updated and can be easily restored.
- It reduces the size of backups of the *master* database. Due to system restrictions, backups of *master* must fit on a single tape or file.

The Upgrade Process

When you upgrade an existing SQL Server or install a new SQL Server, you specify the device where you want to install *sybsystemprocs*. **sybinit** creates the device. For upgrades, it drops all existing Sybase system procedures from *master* without affecting any user-created procedures stored in *master*.

If necessary, the upgrade process increases the number of open databases allowed on your SQL Server.

See Chapter 3, "Changes That May Affect Existing Applications," in this manual for information on moving your local procedures from *master* to *sybsystemprocs*, and other system administration information relating to this change.

Backup and Space Management

Backup and space management changes are as follows.

Backup Server

The Open Server™-based Backup Server utility handles all dumps and loads for SQL Server. Dumps and loads now work more quickly and with significantly less effect on other SQL Server activities. New dump and load syntax provides more flexibility and more options:

- Dump striping allows you to use up to 32 dump devices in parallel to dump or load a single database or transaction log.
- Multifile and multivolume dumps allow one dump to span multiple tapes or allow multiple dumps to be made to a single tape.
- Network dumps allow dumping and loading over the network to or from a device on another machine.
- Backup Server complies with ANSI standard tape labeling and handling.
- Platform-specific tape handling options support dump and load command syntax specification for volume naming, dismount and load control, tape density, block size, tape capacity, days to retain, initialization, file naming for multidump volumes, and listing header or file information instead of loading the files.
- The system procedure **sp_volchanged** signals volume changes during backups. (OpenVMS users do this with the **REPLY**

command.) It replaces the **console** utility program. **sp_volchanged** can be submitted from any ordinary Sybase client to signal volume changes to the Backup Server.

- Multiple dumps and loads can be managed from one or more local or remote servers.

Installing a Backup Server

If you do not want to use the new functionality immediately, you can continue to use your existing dump and load routines. Pre-10.0 syntax is compatible with 10.0 syntax. The only exception is that **dump database** and **dump transaction** commands cannot be included in a user-defined transaction in release 10.0.

Dump and load commands **must** use Backup Server. It must be installed and running, and SQL Server must be configured to connect to Backup Server.

◆ ***WARNING!***

You cannot load dumps made with earlier releases using 10.0 dump and load commands: pre-10.0 dumps are not compatible with SQL Server 10.0.

The SQL Server installation and configuration guide contains information about installing Backup Servers, supported dump devices, and other platform-specific issues.

Additional Sources of Information

The new syntax for the **dump database**, **load database**, **dump transaction**, and **load transaction** commands and the **sp_volchanged** system procedure is documented in the *SQL Server Reference Manual*.

Information about starting and stopping Backup Servers during routine operations can be found under **startserver** in the SQL Server utility programs manual, under **shutdown** in the *SQL Server Reference Manual*, and in your SQL Server installation and configuration guide. Also, see **backupserver** in the SQL Server utility programs manual for information about flags for the **backupserver** executable.

See Chapter 18, “Developing a Backup and Recovery Plan,” in the *System Administration Guide*, for a complete discussion of Backup Server.

Threshold Manager

Thresholds monitor how much free space remains on a particular segment. When the amount of free space falls below a threshold, SQL Server automatically executes the associated stored procedure. For example:

- You can create a threshold and a stored procedure to dump the transaction log when space runs low.
- You can create a threshold and a stored procedure that detects when space for the data segment of a database runs low so that messages in the error log warn you before space runs out.

The system procedures **sp_addthreshold**, **sp_dropthreshold**, **sp_modifythreshold**, and **sp_helpthreshold** create, drop, change, and monitor free-space thresholds in a database. See the *SQL Server Reference Manual* for specific syntax information. Also see Chapter 21, "Managing Free Space with Thresholds," in the *System Administration Guide* for additional examples.

In databases that store data and logs on separate segments, SQL Server installs a last-chance threshold on the log segment. This threshold calls a procedure named **sp_thresholdaction**. Or you can use **sp_modifythreshold** to call a different stored procedure.

Since user requirements for threshold procedures vary so widely, Sybase does **not** provide **sp_thresholdaction**; you must write it yourself. A special system function, **lct_admin**, creates last-chance thresholds on databases created in pre-10.0 SQL Server.

The new **abort tran on log full** option for **sp_dboption** allows you to choose whether to suspend or kill processes that attempt to write to the log when there is not enough log space.

See **sp_addthreshold**, **sp_dboption**, and Chapter 21, "Managing Free Space with Thresholds," in the *System Administration Guide* for examples.

Variables Allowed in Dump and Load Commands

As part of the release 10.0 threshold improvements, you can now use parameters or variables in dump and load commands for database and device names. This allows a single stored procedure (such as **sp_thresholdaction**) to dump multiple databases or transaction logs. See **dump database, dump transaction, load database,** and **load transaction** in the *SQL Server Reference Manual*.

New Security Features

SQL Server release 10.0 provides features targeted at the C2 level of trust. Many of the security features provide new flexibility for SQL Server users who do not need all of the C2 functionality. The major features are:

- New system administration roles
- New user identification and authentication features, including password encryption
- A complete system for auditing activity on SQL Server

System Administration Roles

In previous releases, SQL Server allowed only one login, "sa", to perform most system administration tasks. SQL Server release 10.0 recognizes the following security-related operational and administrative roles:

- System Administrator: required for tasks such as manipulating disk space and memory usage, granting and revoking permission to execute **create database** statements, running diagnostic and repair functions that read data pages or recover data or indexes in a controlled manner, granting and revoking the System Administrator role, and modifying, dropping, and locking server login accounts.
- System Security Officer: required for security-related tasks such as creating server login accounts, changing passwords, granting and revoking roles, configuring the password expiration interval, and managing the audit system.
- Operator: used for backing up and restoring databases.

These roles can be granted to and revoked from individual login accounts in SQL Server. A login account can have more than one role. Roles can be granted permissions on objects and commands, similar to groups. Details on roles can be found in Chapter 5, "Roles in SQL Server," in the *Security Administration Guide*.

When SQL Server release 10.0 is installed, it still has the default "sa" account, which has the System Administrator, System Security Officer, and Operator roles enabled. For greater accountability for the highly privileged users in your system, it is recommended that you create individual login accounts for users who are to be granted these privileges, grant them their roles, and then lock the "sa" account. If

you have automated scripts or programs that log into SQL Server as "sa", see Chapter 5, "Roles in SQL Server," in the *Security Administration Guide* for more information.

User Identification

SQL Server uses a variety of mechanisms to positively identify an individual user and to enforce accountability and security:

- Login account locking makes it possible to lock a user's account without dropping the user and the user's objects from all databases.
- A minimum password length of 6 bytes makes passwords more secure. (This does not affect existing user passwords, but all new or changed passwords must be at least 6 bytes.)
- Passwords are stored in *master..syslogins* in encrypted form.
- You can enable optional system-wide password expiration.
- You can enable optional client-side password encryption.
- Recovery mechanisms are provided in the case of lost or expired passwords.

New sections, "Roles" and "Login Management" in the *SQL Server Reference Manual*, provide an overview of the system procedures and other commands used to manage these features.

Auditing

SQL Server can be configured to provide an audit trail for events such as:

- Server logins and logouts
- Use of any commands that require a special role
- Use of commands that reference a specified object or database
- Deletion of objects
- Execution of stored procedures and triggers
- Any actions performed by a specified user

The audit system includes a new database, *sybsecurity*, which contains the audit trail in a system table called *sysaudits*. Auditing is described in Chapter 8, "Auditing," in the *Security Administration Guide*.

Cursors

SQL Server release 10.0 provides full support for cursors. The following cursor commands are documented in the *SQL Server Reference Manual*:

declare cursor
fetch
open
close
deallocate

A new topic, "Cursors," in the *SQL Server Reference Manual* provides an overview of cursors. New keywords that allow updating and deleting at cursor positions are described on the **delete** and **update** pages. A new **set** option controls the number of rows returned by a **fetch** command. A new system procedure, **sp_cursorinfo**, provides information about the active cursors in a database.

Chapter 16, "Cursors: Accessing Data Row by Row," in the *Transact-SQL User's Guide* provides examples of cursor use, including cursors in stored procedures.

Data Definition Enhancements

Enhancements are as follows.

Integrity Constraints

In addition to the rules, triggers, defaults, and unique indexes already provided by SQL Server, you can now specify integrity constraints in the **create table** statement. These include:

- Unique and primary key constraints to ensure unique values in a column
- Referential integrity constraints to guarantee that a primary key exists in another table when you are inserting or updating a foreign key table
- Check constraints to limit the values that can be inserted into a table
- Defaults to specify default values for a column

Constraints can be changed or dropped with **alter table**. You can get information about the constraints defined for a table using the new **sp_helpconstraint** system procedure.

See **create table** and **alter table** in the *SQL Server Reference Manual* for the new syntax. Chapter 7, "Creating Databases and Tables," in the *Transact-SQL User's Guide* describes how to use the constraints and provides additional examples.

Changes to Views

The **distinct** keyword can now be used in view definition, allowing you to create views that do not contain duplicate rows.

The **with check option** clause has been added to the **create view** statement. When a view is created **with check option**, all rows that are inserted or updated through the view must meet the view criteria.

create view statements can be included in batches with other Transact-SQL statements.

See **create view** in the *SQL Server Reference Manual* and Chapter 9, "Views: Limiting Access to Data," in the *Transact-SQL User's Guide* for more information and examples.

Schemas

The new **create schema** command allows you to create several objects and to grant permissions on those objects in a single statement, which can be committed or rolled back as a unit. See **create schema** in the *SQL Server Reference Manual*.

Datatypes

Release 10.0 of SQL Server provides these new datatypes:

- *numeric* and *decimal* (*dec*) specify exact numeric values, with specified precision and scale to indicate the total number of digits and the number of digits to the right of the decimal point.
- *double precision* specifies an approximate numeric type.

The following changes have been made to existing datatypes:

- *float* now accepts an optional precision specification.

- The default length for the *char*, *varchar*, *nchar*, and *nvarchar* columns is 1 in **create table** statements, variable declarations, and parameter specifications.

All system datatype names are case-insensitive. SQL Server treats numeric constants in queries (such as **select 2* *colvalue***) as exact numeric types unless they are entered using E notation. Values between 2^{31} - 1 and -2^{31} without decimal points are treated as *int*. Values that fall outside the range for integers, or that include a decimal point, are treated as *numeric*. See Chapter 3, "Changes That May Affect Existing Applications," for information on how this can affect existing applications.

The following list shows synonymous datatype names. Names that are new in 10.0 are in bold type.

- *char, character*
- *varchar, char varying, character varying*
- *nchar, national character, national char*
- *nvarchar, national character varying, national char varying, nchar varying*
- ***double precision***, *float*, ***real***
- ***dec***, ***decimal***, ***numeric***
- *int, integer*

See "System and User-Defined Datatypes" in the *SQL Server Reference Manual* and Chapter 6, "Using and Creating Datatypes," in the *Transact-SQL User's Guide* for more information.

Automatic Sequential Values Using the IDENTITY Property

Each table in a database can have a single IDENTITY column. This column is used to store numbers, such as invoice numbers or employee numbers, that are automatically generated by SQL Server. The value of the IDENTITY column uniquely identifies each row in a table.

The IDENTITY column must be of type *numeric* and scale 0. By default, IDENTITY columns cannot be updated and do not allow null values. Only the table owner, the Database Owner, or a System Administrator can explicitly insert a value into an IDENTITY column after setting **identity_insert on** for the table.

Each time you insert a row into a table, SQL Server assigns a unique, sequential value to the IDENTITY column. To retrieve the last value

inserted into an IDENTITY column, use the *@@identity* global variable. To select a table's IDENTITY column, use the **syb_identity** keyword, qualified by the table name where necessary.

For more information about IDENTITY columns, see **create table**, **alter table**, **insert**, **select**, **create view**, and "IDENTITY Columns" in the *SQL Server Reference Manual*.

Automatic Creation of IDENTITY Columns

System Administrators can use the new **auto identity** database option to automatically include a 10-digit IDENTITY column in new tables. Each time a user creates a table in the database without specifying either a **primary** key, a **unique** index, or an IDENTITY column, SQL Server automatically defines an IDENTITY column for the table. For more information see "IDENTITY Columns" in the *SQL Server Reference Manual*.

Changes to Transactions, Triggers, and Stored Procedures

Changes are as follows.

Data Definition Language in Transactions

Certain data definition language commands are now allowed in user-defined transactions. These commands include:

alter table	**create table**	**drop procedure**
create default	**create trigger**	**drop rule**
create index	**create view**	**drop table**
create procedure	**drop default**	**drop trigger**
create rule	**drop index**	**drop view**
create schema		

grant and **revoke** commands are now allowed in transactions. However, some commands such as **dump database** and **dump transaction** are no longer allowed.

A new option for **sp_dboption**, called **ddl in tran**, allows enabling or disabling the use of data definition language in transactions. Be sure to read the cautionary statements about using data definition

language in transactions, since this activity requires locking system tables.

For information about which commands you can use in transactions and about the **ddl in tran** option, see "Transactions" in the *SQL Server Reference Manual*.

Transaction State Information: @@*transtate*

SQL Server can now return information about the state of a transaction. This status information indicates whether the transaction succeeded, whether a single statement was rolled back, or whether an entire transaction was rolled back. The information is available in the new global variable, *@@transtate*.

See "Transactions" in the *SQL Server Reference Manual.* Also see "Variables" in the *SQL Server Reference Manual* for a list of the status values.

New *rollback trigger* Command

The rollback feature has been expanded to include **rollback trigger.** Now, you can roll back an entire transaction using **rollback transaction**, or roll back statements up to the first data modification that fired a trigger using **rollback trigger.** See **rollback trigger** and "Transactions" in the *SQL Server Reference Manual* for more information.

Trigger Self-Recursion

A new **set** option, **self_recursion**, allows triggers to fire self-recursively; that is, to refire as a result of data modifications made by the trigger itself. By default, when a trigger causes data modifications that would also cause the trigger to fire, the trigger is prevented from firing. See **set** in the *SQL Server Reference Manual* for more information.

Stored Procedure Size Limits Removed

Release 10.0 eliminates the size limit for compiled stored procedures and batches. To accommodate this change, the maximum amount of text allowed for a stored procedure has been increased to 16MB. This increase also applies to objects whose text is stored in the

syscomments system table, such as triggers, views, defaults, rules, and constraints.

Query and Data Modification Changes

Changes are as follows.

Datatype Conversion Changes

Changes were made to datatype conversion rules for release 10.0 to provide greater consistency across all Sybase products.

- Formerly disallowed conversions to and from binary datatypes are now allowed.
- Conversion from integer to character now returns an error instead of "**" if the character value is not long enough to store the converted value.
- Some changes were made to the "datatype hierarchy" that controls the output datatype of mixed-mode datatype operations.

See Chapter 3, "Changes That May Affect Existing Applications," for information on how these changes may affect existing applications. The new hierarchy is included in "System and User-Defined Datatypes" in the *SQL Server Reference Manual*.

Additional information about datatype conversions is provided in "Datatype Conversion Functions" in the *SQL Server Reference Manual* and in Chapter 10, "Using the Built-In Functions in Queries," in the *Transact-SQL User's Guide*.

New Hex Conversion Functions

Two new functions provide platform-independent conversions between integer values and hexadecimal strings. See "Datatype Conversion Functions" in the *SQL Server Reference Manual* and Chapter 10, "Using the Built-In Functions in Queries," in the *Transact-SQL User's Guide* for information about **inttohex** and **hextoint**.

Subquery Changes

The behavior of certain subqueries has changed. See Chapter 3, "Changes That May Affect Existing Applications," for detailed

information about these changes and how they may affect your existing applications.

Grouping on *bit* Columns Allowed

You can now use **group by** on *bit* columns.

Random Number Generator Improvements

The **rand** function now uses the output of a 32-bit pseudo-random integer generator. See "Mathematical Functions" in the *SQL Server Reference Manual* for more information.

between Predicate Changes

In some cases, the **between** predicate in Transact-SQL returned results regardless of whether the values supplied were in the proper order (lower number first after the predicate) or were swapped (higher number first). In release 10.0, a query with swapped **between** values returns no rows.

select into Column Headings

In a **select into** statement, column headings must be provided for any select list item that includes aggregate functions or expressions. See Chapter 3, "Changes That May Affect Existing Applications," for examples of expressions that require headings and for examples and syntax.

Column Alias

You can now use the keyword **as** in **select** statements between the select expression and the alias name. For example:

```
select au_lname as lastname from authors
```

Full Dynamic SQL/Precompiler Support

SQL Server release 10.0 provides full support for host variables and Dynamic SQL. For complete information, see the *Embedded SQL/C* and *Embedded SQL/COBOL Programmer's Guides.*

Right Truncation of Character Strings

In previous releases, SQL Server silently truncated *char*, *nchar*, *varchar*, and *nvarchar* strings when an **insert** or **update** entered strings that were longer than the specified column length. A new **set** option, **string_rtruncation**, controls silent truncation of character strings. Set this option **on** to prohibit silent truncation and enforce SQL92 behavior. See **set** in the *SQL Server Reference Manual* for more information.

SQL Standards Compliance

In addition to major features such as declarative integrity constraints and cursors, which are described elsewhere in this chapter, the following changes are included in release 10.0 for compliance to SQL standards. Since certain standard behaviors are not compatible with existing SQL Server applications, Transact-SQL provides **set** options that allow you to toggle these behaviors.

set Commands Required for SQL92 Compliance

Compliant behavior is enabled by default for all Embedded SQL™ precompiler applications. Other applications that need to match standard behavior can use the option settings in Table 2-1 for entry level SQL92 compliance.

Table 2-1: set options for compliance to SQL standards

Option	Setting
ansi_permissions	on
ansinull	on
arithabort	off
arithabort numeric_truncation	on
arithignore	off
chained	on
close on endtran	on
fipsflagger	on
quoted_identifier	on
string_rtruncation	on
transaction isolation level	3

The following sections describe the differences between standard behavior and the default Transact-SQL behavior. For more information on setting these options, see **set** in the *SQL Server Reference Manual*.

FIPS Flagger

For customers who write applications that must conform to standard SQL, SQL Server provides a **set fipsflagger** option to flag incompatible syntax. When this option is turned on, all commands containing Transact-SQL extensions that are not allowed in entry level SQL92 generate an informational message. See **set** in the *SQL Server Reference Manual* for more information.

SQLSTATE Messages and Codes

By default, SQL Server returns SQLSTATE values to embedded SQL applications, as required by entry level SQL92. SQLSTATE values are included in a new column in *sysmessages*, when SQLSTATE codes are defined in the standard. Some of the **set** commands listed in Table 2-1 on page 2-16 cause the associated message text to be delivered to client applications such as **isql**.

The *escape* Clause in the *like* Predicate

SQL Server now supports the SQL standard's **escape** clause, which allows you to specify an escape character in the **like** predicate to search for wildcard characters. This is in addition to Transact-SQL's use of square brackets for the same purpose. See "Wildcard Characters" in the *SQL Server Reference Manual*.

Standard-Style Comments

In Transact-SQL, comments are delimited by /* and */ and can be nested. Transact-SQL now also supports SQL standards-style comments, which consist of any string beginning with two consecutive minus signs, a comment, and a terminating newline character:

```
select "hello" -- this is a comment
```

This syntax conflicts with the subtraction of a negative number. See Chapter 3, "Changes That May Affect Existing Applications," for

information on how this can affect existing applications. Transact-SQL's /**/ comments are still fully supported, and the "--" within Transact-SQL comments is not recognized.

Changes to *set* Options *arithabort* and *arithignore*

Changes were made to the **set** options **arithabort** and **arithignore** to allow compliance with the SQL92 standard:

- **arithabort arith_overflow** determines how SQL Server handles arithmetic overflows and divide-by-zero errors during implicit and explicit conversions. Set this option **on** to roll back the entire transaction or batch in which the error occurs. Set this option **off** to abort the statement that contains the error and continue processing other statements in the transaction or batch.
- **arithabort numeric_truncation** determines how SQL Server handles loss of scale during implicit conversions to a *numeric* or *decimal* datatype. Set this option **on** to abort the statement that contains the error and continue processing other statements in the transaction or batch. Set it **off** to truncate the results as necessary and continue processing.
- **arithignore arith_overflow** determines whether SQL Server ignores arithmetic overflow and divide-by-zero errors. Set this option **off** to print a warning message when these errors occur. Set it **on** to ignore these errors and print no messages.

If you have used these options in your applications, examine them to be sure that they are still producing the desired behavior.

Synonymous Keywords

Several keywords have been added for SQL standard compatibility that are synonymous with existing Transact-SQL keywords.

Table 2-2: SQL standard-compatible keyword synonyms

Pre-Release 10.0 Transact SQL	Additional Syntax
tran transaction	work
any	some
grant all	grant all privileges
revoke all	revoke all privileges

Table 2-2: SQL standard-compatible keyword synonyms (continued)

Pre-Release 10.0 Transact SQL	Additional Syntax
max (*expression*)	**max** ([all | distinct]) *expression*
min (*expression*)	**min** ([all | distinct]) *expression*
user_name (built-in function)	**user** keyword

The new **work** keyword is synonymous with **tran** and **transaction** only with the **commit transaction** and **rollback transaction** commands, not with the **begin transaction** command.

See "Datatypes" on page 2-10 for a list of synonymous datatypes.

Chained Transactions and Isolation Levels

SQL Server now provides SQL standard-compliant **chained** transaction behavior as an option. In chained mode, all data retrieval and modification commands (**delete, insert, open, fetch, select,** and **update**) implicitly begin a transaction. Since such behavior is incompatible with many Transact-SQL applications, Transact-SQL style (or **unchained**) transactions remain the default.

◆ ***WARNING!***

Before you change the default transaction mode or isolation level, be sure to read the sections in the *Transact-SQL User's Guide* and the *SQL Server Reference Manual* that describe how these changes can affect existing applications and stored procedures.

Chained transaction mode can be initiated with a new option to the **set** command. Another **set** option controls transaction isolation levels. See "Transactions" in the *SQL Server Reference Manual* for more information.

Delimited Identifiers

SQL Server now supports delimited identifiers for table, view, and column names. Delimited identifiers are object names enclosed in double quotation marks. Using them allows you to avoid certain restrictions on object names.

Delimited identifiers can begin with non-alphabetic characters, including characters that would not otherwise be allowed, or even be Transact-SQL reserved words. They cannot exceed 28 bytes.

Set the new **quoted_identifier** option **on** to allow delimited identifiers. While the option is on, do not use double quotes around character or date strings; use single quotes instead. Delimiting strings with double quotes causes SQL Server to treat them as identifiers.

➤ ***Note***

Delimited identifiers cannot be used with some system procedures, cannot be used with **bcp**, and may not be supported by all client software.

Treatment of Nulls

A new **set** option, **ansinull**, determines whether or not evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons and in aggregate functions is SQL standard-compliant. This option does not affect how SQL Server evaluates NULL values in other kinds of SQL statements, such as **create table**.

Right Truncation of Character Strings

A new **set** option, **string_rtruncation**, controls silent truncation of character strings for SQL standard compatibility. Set this option **on** to prohibit silent truncation and enforce SQL standard behavior.

Permissions Required for *update* and *delete* Statements

A new **set** option, **ansi_permissions**, determines what permissions are required for **delete** and **update** statements. When this option is **on**, SQL Server uses SQL92's more stringent permissions requirements for these statements. Because this behavior is incompatible with many existing applications, the default setting for this option is **off**.

Enhancements to Permissions

Object and command permissions in SQL Server have been enhanced with the following new options:

grant with grant Option

A new clause for the **grant** command, **with grant option**, allows a permission holder to pass **grant** capability, along with the specific permission, to other users, but not to groups, roles, or "public." A corresponding clause for the **revoke** command, **cascade**, allows the privilege holder to revoke permissions from all users who obtained it via **with grant option**. See **grant** and **revoke** in the *SQL Server Reference Manual* for syntax information. Also see Chapter 6, "Managing User Permissions," of the *Security Administration Guide* for a full discussion and examples of this new option.

Granting to Roles

You can grant and revoke object and command permissions to all users who have been granted a specific role. The roles are **sa_role**, **sso_role**, and **oper_role**. Permissions granted to roles override permissions granted to individuals or groups.

Configurable Packet Size

New **sp_configure** options allow System Administrators to reset the network packet size used by SQL Server. Larger packet sizes can yield performance improvements when large amounts of data are being transferred into or out of SQL Server. See Chapter 11, "Setting Configuration Parameters," of the *System Administration Guide* for full information.

New command line options for **isql** and **bcp** allow you to set the packet size for an individual session. See the SQL Server utility programs manual for information about using these options. Also see the Open Client Client-Library™ documentation for information on using variable packet sizes.

Chargeback Accounting

Chargeback accounting provides a mechanism for tracking CPU and I/O usage for each server user. This feature was previously available only on OpenVMS.

See **sp_clearstats** and **sp_reportstats** in the *SQL Server Reference Manual* for information about getting reports and restarting the accounting interval. See **sp_configure** for information about the **cpu accounting flush**

interval and **i/o accounting flush interval** configuration parameters that affect chargeback accounting.

New *dbcc* Options

These changes have been made to **dbcc**, the database consistency checker:

- A **fix** option has been added to **dbcc checkalloc**, so this command can now fix some allocation errors.
- A **skip_ncindex** option has been added to **dbcc checkdb** and **dbcc checktable**. When you use this option, **dbcc** skips checking the nonclustered indexes. Depending on the number of indexes on your table or in your database, this can speed up the performance of the command by up to 40 percent.

kill Command Enhancements

The **kill** command, used to terminate SQL Server sessions, can now terminate more kinds of sessions. In previous releases, some of these sessions could not be terminated (except by rebooting SQL Server), or would be terminated only when the processes stopped sleeping. Now, these sessions can be terminated immediately. The **sp_who** system procedure displays different status values for these sessions. For more information, see **kill** in the *SQL Server Reference Manual* and Chapter 4, "Diagnosing System Problems," in the *System Administration Guide.*

shutdown Command Enhancements

The **shutdown** command, used to shut down a SQL Server, can now be used to shut down a Backup Server. The Backup Server must be listed in your *sysservers* table and in the *interfaces* file for the SQL Server from which you execute the **shutdown** command.

Unless you use the **with nowait** option, **shutdown** waits for active dumps and loads to complete. Once you issue a **shutdown** command to a Backup Server, no new dumps or loads to the Backup Server can start.

tempdb Changes

By default, all users now have **create table** permission in *tempdb*. See "Temporary Tables" in the *SQL Server Reference Manual* for more information.

Additional Information on Space Usage

System procedures now display additional information about space available in databases:

- **sp_helpdb *database_name*** displays information about how much space is left on each disk piece assigned to the database
- **sp_helpsegment *segment_name*** displays information about the amount of free space left on the segment

Error Handling

raiserror can now return the names of the tables and columns that are causing an error, along with the error number and text. See **raiserror** in the *SQL Server Reference Manual*.

create index Performance Enhancements

Internal improvements have enhanced the speed of the **create index** command. Also, you can increase the speed of creating indexes with a new **sp_configure** option.

When SQL Server creates an index, it reads and writes pages to disk for intermediate sort results and writes out the final index pages one page at a time. With release 10.0, a System Administrator can allocate buffers so that **create index** can perform these reads and writes one extent at a time, thereby making more efficient use of the disk and increasing the speed of the **create index** command. If you want to use this option, there are two important considerations:

- The buffers are **added to** the memory allocated by SQL Server. Setting the value too large can make it impossible for SQL Server to start if it cannot acquire enough memory.
- The first user who starts a **create index** command acquires all of the allocated buffers. All subsequent **create index** commands that are started before the first one completes will use regular one-page-at-a-time disk I/O. For best performance, either create indexes

when other users are not likely to contend for the resource or coordinate the creation of large indexes among table owners.

See Chapter 11, "Setting Configuration Parameters," of the *System Administration Guide* for information about the new **sp_configure** option, **number of extent i/o buffers.** The section contains suggestions for choosing the correct buffer size.

Improvements to the Query Optimizer

Improvements to the query optimizer have increased SQL Server's accuracy in evaluating the cost of using indexes. You must run **update statistics** on your indexes in order to take advantage of these changes, although any previously created indexes continue to work.

This change involves the way that the query optimizer assesses the cost of using indexes with compound keys on the inner table of a join. Prior to release 10.0, the optimizer computed a value for the entire key and often underestimated the cost of using an index if the query used only one or a few of the columns. In release 10.0, SQL Server maintains statistics for each prefix of columns in the index, that is, for the first column, for the first and second columns, for the first, second, and third columns, and so on.

The statistics used by the optimizer are generated by the **create index** command and are updated by the **update statistics** command. Indexes created in earlier releases of SQL Server still work in release 10.0, but in order to take advantage of the new behavior described here, you must run the **update statistics** command on the table or index.

Bulk Copy Performance Enhancements

Internal improvements, which are transparent to users, result in faster bulk copy speeds. In addition, configurable packet sizes can increase bulk copy throughput. See **bcp** in the SQL Server utility programs manual and Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide* for information about configuring packet size.

Spanish Collating Orders

Release 10.0 supports these Spanish dictionary sort orders:

- Case-sensitive
- Case-insensitive
- Case- and accent-insensitive

To change the sort order on an existing SQL Server to use a Spanish collating sequence, follow the directions in Chapter 12, "Configuring Character Sets, Sort Orders, and Message Language," in the *System Administration Guide*. Instructions for installing a new SQL Server using a Spanish collating sequence are in the SQL Server installation and configuration guide.

Documentation Reorganization

The SQL Server documentation has been reorganized for this release.

- The *Commands Reference Manual* has been split into two volumes:
  - Transact-SQL commands, functions, and special topics (such as "Locking" and "Transactions") are now documented in Volume 1 of the *SQL Server Reference Manual*.
  - The Sybase system procedures and catalog stored procedures are now documented in Volume 2 of the *SQL Server Reference Manual*.
- The Sybase SQL Server utility programs manual now documents the utility programs for your operating system.
- The *Transact-SQL User's Guide* has been reorganized, with basic concepts for new SQL users presented in the first section and the more advanced and complex concepts presented in the second section.

Summary by Command Name, Type, and Tables Affected

Affected commands, types, and tables are as follows.

set Options

The following table summarizes new and changed **set** options:

Table 2-3: New set options

Option	Function
set ansinull {on | off}	Toggles SQL standard treatment of NULL-valued operands in equality (=) and inequality (!=) comparisons. When **on**, treatment is SQL92-compliant.
set ansi_permissions {on | off}	Determines whether SQL92-compliant permissions requirements for **update** and **delete** statements are checked.
set arithabort [arith_overflow | numeric_truncation] {on | off}	Determines whether SQL Server aborts a query when an arithmetic overflow or numeric truncation error occurs.
set arithignore [arith_overflow] {on | off}	Determines whether SQL Server displays a warning message after any query that results in arithmetic overflow.
set chained {on | off}	Toggles the chained transaction mode in SQL Server.
set close on endtran {on | off}	Forces all open cursors to be closed when the transaction ends (**on**) or to remain open across transactions (**off**).
set cursor rows *number* for *cursor_name*	Sets the number of cursor rows returned to the host on each **fetch** for the specified client cursor. The default is 1.
set dup_in_subquery {on | off}	Controls whether a subquery using the **in** clause returns duplicate values. The default is **off**; duplicate rows are not returned. In prior SQL Server releases, a row was returned for each matching row in the subquery. SQL92 specifies removing duplicate values from the result set.
set fipsflagger {on | off}	Toggles the FIPS flagger. When **on**, the option prints warning messages for the Transact-SQL extensions to SQL92.
set identity_insert *table_name* {on | off}	Determines whether a user can explicitly insert a value into *table_name*'s IDENTITY column. Users can turn this option **on** for only one table at a time per database.

Table 2-3: New set options (continued)

Option	Function
set quoted_identifier {on | off}	Determines whether SQL Server treats strings enclosed in double quotes (") as identifiers.
set role {"sa_role" |"sso_role" | "oper_role"} {on | off}	Toggles the specified role during the current SQL Server session.
set self_recursion {on | off}	Switches trigger self-recursion on or off so that triggers that modify data cause the trigger to fire again.
set string_rtruncation {on | off}	Determines whether silent truncation of character strings is allowed. When **on**, treatment is SQL92-compliant.
set transaction isolation level {1|3}	Sets the transaction isolation level. An isolation level of 3 is the equivalent of **select with holdlock**.

set showplan now prints full command names instead of abbreviations.

New and Changed Built-In Functions

The following table lists new and changed built-in functions:

Table 2-4: New and changed built-in functions

Name	Function
col_name	Changed to accept a database name as an optional third parameter.
curunreservedpgs	New system function. Returns the number of free pages in adisk piece.
inttohex	New datatype conversion function. Returns the platform-independent hexadecimal equivalent of an integer.
hextoint	New datatype conversion function. Returns the platform-independent integer equivalent of a hexadecimal string.
index_col	Changed to accept a user name as an optional fourth parameter.
lct_admin	New system function. Adds a last-chance threshold on the log segment of a pre-10.0 database, or reports on the last-chance threshold status, or wakes up processes waiting for a threshold, depending on the parameter you use.

Table 2-4: New and changed built-in functions

Name	Function
object_name	Changed to accept a database name as an optional second parameter.
proc_role	New system function. Checks to see if the user possesses the specified role. Useful for restricting execution of procedures.
show_role	New system function. Shows the user's current, active roles.
user	New SQL92-compatible system function returns the user name.

In addition, many of the mathematical functions have been changed to accept the new numeric datatypes, as appropriate.

See the *SQL Server Reference Manual* for information on these functions.

New System Procedures

The following new system procedures provide some of the functionality for the features described earlier in this summary. The table shows the procedure name and its associated function:

Table 2-5: New system procedures

Procedure	Function
sp_addauditrecord	Allows users to enter user-defined audit records (comments) into the audit trail.
sp_addthreshold	Creates a free-space threshold to monitor space remaining on a database segment.
sp_auditdatabase	Establishes auditing of different types of events within a database or of references to objects within that database from another database.
sp_auditlogin	Audits a user's attempts to access tables and views, and/or the text of a user's commands.
sp_auditobject	Establishes auditing of accesses to tables and views.
sp_auditoption	Enables and disables system-wide auditing and global audit options.
sp_auditsproc	Audits the execution of stored procedures and triggers.

Table 2-5: New system procedures (continued)

Procedure	Function
sp_bindmsg	Binds a user message to an integrity constraint.
sp_checkreswords	Checks for reserved words used as identifiers. The procedure is run as part of pre-upgrade.
sp_configurelogin	Initializes the security-relevant information for a new Secure SQL Server™ login.
sp_cursorinfo	Reports information about a specific cursor or all cursors that are active.
sp_dbremap	Forces changes made by **alter database** to be recognized by SQL Server. Run this procedure only if instructed by SQL Server messages.
sp_displaylogin	Displays information about a login account.
sp_dropthreshold	Removes a free-space threshold from a segment.
sp_estspace	Estimates the amount of space needed for a table and its indexes.
sp_helpconstraint	Reports information about any integrity constraints specified for a table.
sp_helpthreshold	Reports information about all thresholds in the current database or all thresholds for a particular segment.
sp_locklogin	Prevents a user from logging in, or displays a list of all locked accounts.
sp_modifylogin	Modifies default database, default language, and full name information for a SQL Server™ login account.
sp_modifythreshold	Changes parameters for existing thresholds in a database.
sp_procxmode	Displays or changes the transaction modes associated with stored procedures.
sp_remap	Upgrades a release 4.8 or higher stored procedure, trigger, rule, default, or view to be compatible with release 10.0. Use this on objects that the **upgrade** procedure failed to remap.
sp_role	Grants or revokes roles to a SQL Server login account.
sp_thresholdaction	Default threshold procedure that is executed automatically when remaining space on the log segment falls below the last-chance threshold. This procedure is not supplied by Sybase. By creating it yourself, you ensure that it is tailored to your own needs.
sp_unbindmsg	Unbinds a user-defined message from a constraint.

Table 2-5: New system procedures (continued)

Procedure	Function
sp_volchanged	Notifies SQL Server that the operator has performed the requested volume handling during a **dump** or **load**.

The following procedures are used for chargeback accounting. The chargeback accounting feature is now available on all platforms. In earlier releases, it was available only on OpenVMS.

Table 2-6: Chargeback accounting system procedures

Procedure	Function
sp_clearstats	Initiates a new accounting period for all server users, or a specified user. Prints out statistics for the previous period by executing **sp_reportstats**.
sp_reportstats	Reports statistics on system usage.

See the *SQL Server Reference Manual* for information on these procedures.

Changes to System Procedures

The following system procedures have been changed in this release:

Table 2-7: Changes to system procedures

Procedure	Change
sp_addsegment, **sp_dropsegment,** **sp_extendsegment**	These procedures now require a database name as the second argument. This change prevents users from accidentally affecting segments in the wrong databases.
sp_defaultdb **sp_defaultlanguage**	These procedures have been superseded by **sp_modifylogin**, which can also change or add "full name" information about a login account. These procedures are still provided by Sybase, but are no longer documented. Use **sp_modifylogin** instead.
sp_helpdb	Now displays information about how much space is left on each disk piece assigned to the database.
sp_helpsegment	Now displays information about the amount of free space left on the segment.

Table 2-7: Changes to system procedures (continued)

Procedure	Change
sp_lock	Now displays information whether a lock is associated with a cursor.
sp_who	Displays different status values for sleeping processes: recv sleep, send sleep, lock sleep, and alarm sleep.

New Configuration Parameters

These new configuration parameters are available for use with **sp_configure**:

Table 2-8: New configuration parameters

Option	Function
additional netmem	Memory added for use with variable packet sizes.
audit queue size	Number of audit records in the audit queue.
cpu accounting flush interval (formerly **cpu flush**)	Used to configure chargeback accounting (new on non-OpenVMS platforms).
default network packet size	Default packet size.
identity burning set factor	Determines the percentage of potential IDENTITY column values made available in memory. When assigning a new IDENTITY column value, SQL Server chooses the next value from this set. If the server fails before assigning these values, it discards any remaining values in the set, leading to gaps in IDENTITY column values.
i/o accounting flush interval (formerly **i/o flush**)	Used to configure chargeback accounting (new on non-OpenVMS platforms).
max network packet size (formerly **maximum network packet size**)	Maximum allowable packet size.
number of extent i/o buffers (formerly **extent i/o buffers**)	Allocates additional memory to be used for buffering data pages during **create index** and **order by** reads and writes to disk.

Table 2-8: New configuration parameters (continued)

Option	Function
systemwide password expiration (formerly **password expiration interval**)	Sets password expiration time.

The serial number field, configuration value 114, has been removed.

New Database Options

These new database options are available:

Table 2-9: New sp_dboption database options

Option	Function
abort tran on log full	Determines whether user processes are suspended (the default) or aborted when the last-chance threshold on a database's log segment is reached.
allow nulls by default	Changes the default null type for **create table** statements from **not null** (the Transact-SQL default) to **null** (the SQL92 default.)
auto identity	Automatically defines a 10-digit IDENTITY column, **syb_identity_col**, in each new table that does not specify a **primary** key, a **unique** index, or an IDENTITY column. The column is not visible with a **select *** statement; to retrieve it, you must include the column name in the select list.
ddl in tran	Allows data definition language in user transactions. The allowed commands are all **create** and **drop** commands, except **create/drop database**, plus **grant** and **revoke** commands.
no free space acctg	Suppresses free-space accounting and the execution of threshold actions on non-log segments of a database.

You can change these options with **sp_dboption.**

Also, the **trunc. log on chkpt.** option can now be used with or without the periods, that is, **trunc log on chkpt** is also supported.

New System Tables

There are four new system tables in all databases:

- *sysconstraints*
- *sysreferences*
- *sysroles*
- *systhresholds*

The *master* database includes the following new system tables:

- *syslisteners*
- *sysloginroles*
- *syssrvroles*

If you install auditing, the *sybsecurity* database is automatically created with two tables: *sysaudits* and *sysauditoptions* (as well as the default system tables for all user databases).

These are fully documented in Chapter 2, "The System Tables," of the *SQL Server Reference Supplement*.

Changes to Existing System Tables

The following system tables have been changed in this release:

Table 2-10: Changes to existing system tables

Table	Change
master..sysdatabases	New columns: *status2, audflags, deftabaud, defvwaud, defpraud; mode* removed.
master..syslocks	New column: *class*.
master..syslogins	New columns: *pwdate, audflags, fullname*. Formerly reserved columns now used: *status, accdate, totcpu*, and *totio*. (The last three columns are used by chargeback accounting, formerly available on OpenVMS only.)
master..sysmessages	New column: *sqlstate*.
master..sysprocesses	New columns: *tran_name, time_blocked, network_pktsz*.
master..sysusages	New columns: *pad and unreservedpgs*.
syscolumns	New columns: *prec, scale*.
syscomments	New column: *colid2*.
sysindexes	New column: *status2*. Changed columns: *spare1* to *oampgtrips*, *spare2* to *ipgtrips*.

Table 2-10: Changes to existing system tables (continued)

Table	Change
sysobjects	Changed columns: *schema* to *schmacnt*, *refdate* to *sysstat2*, *category* to *ckfirst*. New columns: *objspare* and *audflags*.
sysprotects	New column: *grantor*. Changed columns: values for the *protecttype* column are different. 0 indicates **grant with grant**, 1 indicates regular **grant**, and 2 indicates **revoke**.
systypes	New columns: *prec*, *scale*, *ident*, *hierarchy*.

3 Changes That May Affect Existing Applications

Introduction

This chapter describes the changes to release 11.0 and release 10.0 (for users upgrading to release 11.0 from a release prior to 10.0) of SQL Server that may affect existing applications.

The SQL Server Release 11.0 Changes

The SQL Server Release 10.0 Changes

Changes Related to SQL Server Release 11.0

This section describes the changes to release 11.0 from release 10.0 of SQL Server that may affect your applications. If you upgrade to release 11.0 from a pre-release 10.0 of SQL Server, read this section **and** the subsequent section, "Changes Related to SQL Server Release 10.0", which describes the changes that are still relevant for release 11.0.

New Transact-SQL Keywords

The following keywords are new "reserved" words in SQL Server 11.0. They cannot be used as object names or column names.

Table 3-1: New release 11.0 reserved words

max_rows_per_page
online
partition
unpartition

You must change all database names that are new reserved words before you can upgrade an earlier version of the server. You can change table, view, and column names or use delimited identifiers. Once you upgrade to release 11.0, you cannot use database objects whose names are new reserved words until you modify your procedures, SQL scripts, and applications.

Changes in SQL Server Configuration

The following changes to the SQL Server configuration interface will affect existing applications.

The *reconfigure* Command

In previous releases, changes made with **sp_configure** had to be activated with the **reconfigure** command. This is no longer required. **reconfigure** is no longer operational. Existing scripts should continue to run, but should be changed at your earliest convenience because **reconfigure** may not be supported in future releases.

buildmaster -r No Longer Supported

The **buildmaster -r** flag, which was used to rewrite the configuration block with default values for SQL Server configuration parameters, is no longer supported.

To run SQL Server with its built-in default values:

1. Rename, move, or delete your configuration file.
2. Shut down and restart SQL Server.

When you do this, SQL Server uses the built-in default configuration values, and creates a configuration file *server_name.cfg* in the directory from which SQL Server was started. However, these values do not get written to *sysconfigures*. In order to have them written to *sysconfigures*, you need to shut down and restart SQL Server once again.

New Names for Existing Configuration Parameters

The following SQL Server configuration parameters (formerly called "configuration variables") have new names:

Table 3-2: New configuration parameter names

Old Name	New Name
T1204 (trace flag)	print deadlock information
T1603 (trace flag)	allow sql server async i/o
T1610 (trace flag)	tcp no delay
T1611 (trace flag)	lock shared memory
allow updates	allow updates to system tables
calignment	memory alignment boundary
cclkrate	sql server clock tick length
cfgcprot	permission cache entries
cguardsz	stack guard size
cindextrips	number of index trips
cmaxnetworks	max number network listeners
cmaxscheds	i/o polling process count
cnalarm	number of alarms
cnblkio	disk i/o structures

Table 3-2: New configuration parameter names (continued)

Old Name	New Name
cnlanginfo	number of languages in cache
cnmaxaio_engine	max async i/os per engine
cnmaxaio_server	max async i/os per server
cnmbox	number of mailboxes
cnmsg	number of messages
coamtrips	number of oam trips
cpreallocext	number of pre-allocated extents
cpu flush	cpu accounting flush interval
cschedspins	runnable process search count
csortbufsize	number of sort buffers
csortpgcount	sort page count
ctimemax	cpu grace time
database size	default database size
default language	default language id
devices	number of devices
extent i/o buffers	number of extent i/o buffers
fillfactor	default fill factor percent
i/o flush	i/o accounting flush interval
language in cache	number of languages in cache
locks	number of locks
maximum network packet size	max network packet size
memory	total memory
mrstart	shared memory starting address
nested trigger	allow nested triggers
open databases	number of open databases
open objects	number of open objects
password expiration interval	systemwide password expiration
pre-read packets	remote server pre-read packets
procedure cache	procedure cache percent
recovery flags	print recovery information

Table 3-2: New configuration parameter names (continued)

Old Name	New Name
recovery interval	recovery interval in minutes
remote access	allow remote access
remote connections	number of remote connections
remote logins	number of remote logins
remote sites	number of remote sites
sql server code size	executable code size
tape retention	tape retention in days
user connections	number of user connections

New Configuration Parameters

In release 11.0, there are a number of new SQL Server configuration parameters:

Table 3-3: New SQL Server configuration parameters

address lock spinlock ratio
configuration file
deadlock checking period
deadlock retries
event buffers per engine
freelock transfer block size
housekeeper free write percent
identity grab size
lock promotion HWM
lock promotion LWM
lock promotion PCT
max engine freelocks
o/s async i/o enabled
o/s file descriptors
page lock spinlock ratio
page utilization percent

Table 3-3: New SQL Server configuration parameters (continued)

partition groups
partition spinlock ratio
size of auto identity column
table lock spinlock ratio
total data cache size
user lock cache size
user lock cache spinlock ratio

New *deadlock checking period* Parameter

Previous releases of SQL Server performed deadlock checking when the process began to wait for a lock. In release 11.0, SQL Server performs deadlock checking for a process only after it has waited a minimum of 500 ms. for a lock to be released. To return to the previous way of deadlock checking, set **deadlock checking period** to 0.

New *page utilization percent* Parameter

The default behavior of previous releases of SQL Server is to search the OAM page chain for unused pages before allocating a new extent. To keep this behavior, set the **page utilization percent** to 100.

Compiled Object Sizes Have Grown

The size of compiled objects is larger in System 11, and was significantly larger in System 10. You may need to resize your procedure cache to maintain the same performance.

SQL Server Code Size Has Grown

More memory is used for the kernel and other internal structures. You may need to add more memory to maintain the same performance as your previous release.

Subquery Changes

Several problems relating to subqueries have been fixed in SQL Server release 11.0. This section describes old and new behavior in

detail. Examine applications that use subqueries to determine if your application relies on behavior that has been corrected.

Subquery processing in previous releases of SQL Server used "outside-in" processing. Release 11.0 subqueries are processed "inside-out" for greater efficiency. Upgrading to release 11.0 does not automatically change the processing mode of the subquery.

After upgrading to release 11.0, you must drop and re-create objects to take advantage of the new processing style. Use **sp_procqmode** to identify objects which include subqueries. After upgrading to SQL Server release 11.0, you cannot create an object which uses System 10 processing.

It is best to test procedures that contain subqueries and make any changes that are necessary for your application before upgrading your production system.

Different Results

Subqueries in objects created on a release 11.0 SQL Server may return different results than subqueries in objects created on a previous SQL Server. To test performance after upgrade, create the objects on a release 11.0 SQL Server, test the results and performance, and make any changes that are necessary for your application.

Determining Query Processing Mode

A new system procedure, **sp_procqmode**, reports on the subquery processing mode of an object.

Duplicates in Subquery Results

As of release 10.0, subqueries using **in** or **any** predicates no longer return duplicates.

set dup_in_subquery on was provided as an upgrade path to SQL Server release 10.0. It is no longer supported. If your application depends on the duplicate rows in the result set, rewrite the subquery as a join.

For example,

```
select a from s where b in
    (select c from t)
```

would become

```
select a from s, b where s.b = t.c
```

Procedures created in SQL Server release 10.0 that use **dup_in_subquery** will continue to run until the procedure is dropped and re-created in release 11.0.

Changes in Subquery Restrictions

The following subquery restrictions have been removed:

- You can use **distinct** with **group by** in a subquery.

  The **distinct** keyword suppressed duplicate rows in the output of pre-System 10™ subqueries. This keyword is no longer necessary in subqueries and is ignored.

- You can reference a correlated variable in the select list of a subquery. For example, the following expression subquery is now supported:

```
select t1.c1 from t1
    where t1.c2 = (select t2.c2 + t1.c3 from t2)
```

The following subquery restriction has been added:

- There is a maximum of 16 subqueries within a single side of a union.

Handling NULL Results

Prior to release 11.0, a correlated expression subquery in the **set** clause of an update returned 0 instead of NULL when there were no matching rows. Release 11.0 correctly returns NULL when there are no matching rows, and an error is raised if the column does not permit nulls.

If you have applications that depend on the pre-release 11.0 behavior, you will need to rewrite them.

For example, the following trigger tries to update a column that does not permit NULL values:

```
update t1
    set c1 = (select max(c1)
    from inserted where t1.c2 = inserted.c2)
```

The correct trigger is:

```
update t1
    set c1 = (select isnull(max(c1), 0)
    from inserted
    where t1.c2 = inserted.c2)
```

or:

```
update t1
    set c1 = (select max(c1)
    from inserted
    where t1.c2 = inserted.c2)
where exists (select * from inserted
    where t1.c2 = inserted.c2)
```

The **where** clause updates table *t1.c1* to 0, if the subquery does not return any correlation values from the outer table *t1*.

Another example of this is the *totalsales_trig* in the *pubs2* sample database. In previous versions, the trigger read as follows:

```
create trigger totalsales_trig
on salesdetail
for insert, update, delete
as
/* Save processing:  return if there are no rows affected */
if @@rowcount = 0
        begin
                return
        end

/* add all the new values */
/* use isnull:  a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
        set total_sales = isnull(total_sales, 0) +
        (select sum(qty)
                from inserted
                where titles.title_id = inserted.title_id)

/* remove all values being deleted or updated */
update titles
        set total_sales = isnull(total_sales, 0) -
        (select sum(qty)
                from deleted
                where titles.title_id = deleted.title_id)
```

sum(qty) is NULL if no row is returned from the table, so when a statement changes the *total_sales* column, the trigger changes to NULL all the rows in *titles* that do not qualify.

To guarantee that the subquery in the expression for the update returns a non-null value, the corrected trigger is:

```
create trigger totalsales_trig
on salesdetail
for insert, update, delete
as
/* Save processing:  return if there are no rows affected */
if @@rowcount = 0
        begin
                return
        end

/* add all the new values */
/* use isnull:  a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
        set total_sales = isnull(total_sales, 0) +
        (select sum(qty)
                from inserted
                where titles.title_id = inserted.title_id)
        where title_id in (select title_id from inserted)

/* remove all values being deleted or updated */
update titles
        set total_sales = isnull(total_sales, 0) -
        (select sum(qty)
                from deleted
                where titles.title_id = deleted.title_id)
        where title_id in (select title_id from deleted)
```

Improved Performance

Most subqueries will perform better after being dropped and re-created.

However, an expression subquery containing an aggregate where the outer table is very large and has few duplicate correlation values, and the inner table is small, may not perform as well under System 11. For example:

```
select * from huge_table where x=
    (select sum(a) from tiny_table
    where b = huge_table.y)
```

The workaround is:

```
select huge_table.y, s=sum(a)
    into #t
    from huge_table, tiny_table
    where b=huge_table.y
    group by huge_table.y

select huge_table.*
    from huge_table, #t
    where x=#t.x
    and huge_table.y=#t.y
```

Changes to *showplan* Output

The output for the set option showplan has changed. If you have any applications that use this generated output, they need to be changed to use the new output. See Chapter 8, "Understanding Query Plans" in the *Performance and Tuning Guide*.

New Caching Strategies May Affect Performance

The optimizer may now choose a new caching strategy called "fetch and discard" strategy. Standard caching strategy reads pages in at the head of the MRU/LRU (most recently used/least recently used) chain in cache. This flushes other pages out of cache. Fetch and discard (or mru) strategy reads pages into cache much closer to the end of the cache, so it does not flush other pages. But these pages remain in cache a much shorter time, so subsequent user queries that might get a "cache hit" by finding these pages in cache under the standard strategy will be less likely to find the page in cache.

When the optimizer estimates that a particular query will read a significant number of pages, and only need those pages once during a query, it may choose fetch and discard strategy. If your disk I/O increases in System 11, check the I/O strategy for queries using **showplan** to see the cache strategy in use. If the queries are using fetch and discard or "MRU" strategy, you can add a clause to **select**, **update** and **delete** commands to specify standard caching. See Chapter 9, "Advanced Optimizing Techniques" in the *Performance and Tuning Guide*.

Upgrading Database Dumps

If you use scripts to perform database loads using the **load database** and **load transaction** commands, the scripts will fail unless you include the **online database** command at the end of your load sequence. You should also remove the following **sp_dboption** options from your scripts: **no chkpt on recovery**, **dbo use only**, and **read only**. These options are no longer needed with release 11.0, as **load database** sets the database status to offline, thus making it inaccessible by users.

For more information on upgrading database dumps, see Chapter 19, "Backing Up and Restoring User Databases," in the *System Administration Guide* and the **load database** and **dump database** commands in the *SQL Server Reference Manual*.

Partitions and Physical Data Placement

In release 10.0, a heap table's data was always inserted at the end of a single page chain. This meant that a heap table (for example, a history table) would physically store its newest entries at the end of the page chain, and its oldest entries at the beginning. A simple **select** statement or cursor scan against such a table could return rows in roughly the same order in which they were inserted.

Release 11.0 enables you to create partitioned tables that have multiple page chains. Inserts into partitioned tables can occur at the ends of many separate page chains. Data in such a table **is not** physically stored in sequential order. If you want to view rows from a partitioned table in sequential order, use the **order by** clause in your **select** statement.

See "Improving Insert Performance with Partitions" on page 13-12 in the *Performance and Tuning Guide* for more information about partitions.

Changes Related to SQL Server Release 10.0

If you upgrade to release 11.0 from a pre-release 10.0 SQL Server, this section describes the changes in SQL Server release 10.0 that are still relevant for release 11.0.

New Transact-SQL Keywords

The following keywords are new "reserved" words in SQL Server 10.0. They cannot be used as object names or column names.

Table 3-4: New release 10.0 reserved words

arith_overflow	identity_insert	references
at	isolation	replace
authorization	key	role
cascade	level	rows
check	mirror	schema
close	national	shared
constraint	noholdlock	some
current	numeric_truncation	stripe
cursor	of	syb_identity
deallocate	only	syb_restree
double	open	user
endtran	option	user_option
escape	precision	varying
fetch	primary	work
foreign	privileges	
identity	read	

You must change all database names that are new reserved words before you can upgrade an earlier version of the server. You can change table, view, and column names or use delimited identifiers. Once you upgrade, you cannot use database objects whose names are new reserved words until you modify your procedures, SQL scripts, and applications.

A new system procedure, **sp_checkreswords**, checks for reserved words used as identifiers and reports the object names. For information on running this procedure before you upgrade, see your SQL Server installation and configuration guide.

If you choose to leave some of these reserved words as identifiers until after you upgrade, you can run the procedure **sp_checkreswords** at any time. See **sp_checkreswords** in the *SQL Server Reference Manual*

for information about running this procedure and for detailed information about upgrading your applications.

Password Changes

SQL Server no longer allows NULL passwords. All passwords must be at least 6 bytes in length. Existing passwords containing less than 6 bytes are not changed during upgrade, but all future uses of **sp_password** will require a new 6-character password.

You can globally force users to change passwords within a specified time period using the **systemwide password expiration** option to **sp_configure**. This feature provides last-chance warning messages when a user logs in with a password that is about to expire. Users whose passwords have expired can log in, but they can execute **sp_password** only.

If you have applications that include embedded passwords, you will need to update them each expiration period if you turn on password expiration.

See Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide* for more information.

Addition of *sybsystemprocs* Database

As of release 10.0, all Sybase-provided system procedures are stored in a database called *sybsystemprocs* rather than in the *master* database. Possible effects on user applications are:

- Permissions on system procedures need to be changed in *sybsystemprocs*.
- System Administrators need to decide where to place their own procedures.
- Upgrade may increase the **open databases** configuration parameter, if adding *sybsystemprocs* requires it. You should verify the value for your needs on your server.
- Scripts that perform backups and **dbcc** may need to be changed.
- Recovery procedures of the *master* database and system databases have been changed.

Changing Permissions on System Procedures

If you have changed the default permissions on any system procedures, you must **grant** or **revoke** the permissions on those procedures in *sybsystemprocs* after upgrade.

Moving Your Own Procedures

To move your own procedures to *sybsystemprocs*, you must:

- Change all references to system tables or other objects that exist only in *master* to reference the *master* database explicitly.
- Add checks to be sure that the procedure is not being run from within a transaction.

Once these steps have been performed, you can create the procedure in the *sybsystemprocs* database and drop it from *master*.

Changing References to master's System Tables

If you want to move your locally created system procedures to *sybsystemprocs*, and they reference system tables in *master*, you must qualify all the table names with the database name.

For example, if your procedure references *syslogins*, you must change the reference to *master..syslogins*. When you create a stored procedure, SQL Server checks to see whether all the tables that are referenced in the procedure exist. You cannot create the procedure in *sybsystemprocs* without the explicit database reference.

Checking for the Existence of Transactions

A stored procedure in *sybsystemprocs* should never modify the tables in *master* inside a transaction, because this could create problems with recovery.

Sybase system procedures include a check, similar to this:

```
if @@trancount > 0
begin
    print "Can't run this procedure from within a transaction"
    return 1
end
```

Configuring Open Databases

By default, SQL Server is configured to allow up to 10 open databases. The upgrade process will reconfigure the **open databases** variable if the addition of *sybsystemprocs* requires an additional open database.

Changing Backup and *dbcc* Procedures

If you add your own procedures to *sybsystemprocs* or change the default permissions on the system procedures, you should add *sybsystemprocs* to your regular backup schedule.

You should also include *sybsystemprocs* in your regular **dbcc** checks.

Changes in Master Recovery

Due to the addition of *sybsystemprocs* and also Backup Server changes in system tables and system procedures, the process of recovering system databases in case of a failure of the master device has changed. See Chapter 20, "Backing Up and Restoring the System Databases," in the *System Administration Guide* for more information.

Remote Access and the Backup Server

As of release 10.0, all dump and load operations are executed through remote procedure calls to the Backup Server. To allow SQL Server to communicate with the Backup Server, the configuration parameter **allow remote access** must be turned on. Leave this parameter turned on unless you have security concerns. You must be a System Security Officer to set **allow remote access**.

Turning **allow remote access** on does not affect any of the other conditions needed to execute remote procedure calls between servers. This setting, in itself, does not represent a security concern, because server-to-server communication requires additional system administration actions to enable remote procedure calls. These actions include making entries for the remote servers in *master..sysservers*; and making entries for remote users in *master..sysremotelogins*.

If you want to leave **allow remote access** disabled except during a dump or load, you must issue the following command before issuing the dump or load command:

```
sp_configure "allow remote access", 1
```

After you complete the dump, turn **allow remote access** off again with the command:

```
sp_configure "allow remote access", 0
```

The **allow remote access** configuration parameter is dynamic: you do not have to restart SQL Server for the **sp_configure** command to take effect.

Changes to Dump Scripts

Backup facilities have been changed as of SQL Server release 10.0. This section describes the minimum changes needed to existing dump scripts.

The single feature that can most affect your current use of tapes and dump commands is Backup Server's ability to make multiple dumps to a single tape. The new dump is placed after the last existing file on the current tape volume.

➤ ***Note***

All dumps to the "null device" (*/dev/null* on UNIX or any device starting with "NL" on OpenVMS) are now prohibited.

Here are some guidelines for backing up your databases immediately after upgrading:

- If you use new tapes, or tapes without ANSI labels, existing dump scripts overwrite the entire tape.
- If you use single-file media (for example, 1/4-inch cartridge) with ANSI labels, and the expiration dates on the tapes have expired, existing dump scripts will overwrite the tapes.
- If the expiration date on a single-file tape has not been reached, you will be asked to confirm the overwrite. A positive response overwrites the existing tape; a negative response initiates a request for a volume change, and tests are repeated on the new volume.
- If you use multifile tape media, and you do not change your dump scripts, the dump is appended to the existing files on the tape.
- If you want to overwrite existing tapes that have ANSI labels, you must append the **with init** clause to existing dump commands:

```
dump databasse mydb
    to datadump1
    with init
```

You can also use operating system commands to erase or truncate the tape.

The following diagram shows the logic used in tape label checking on a dump command used without the **init** option:

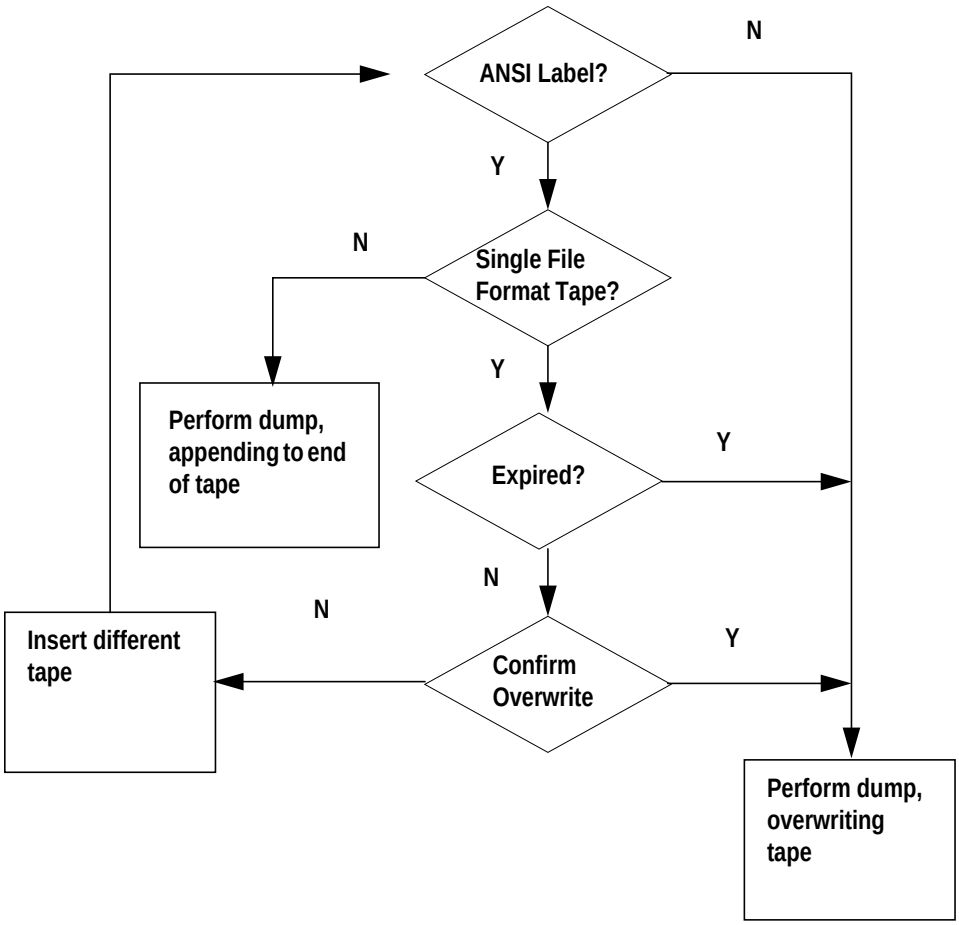

Figure 3-1: Checking tape format and expiration dates for dump commands

Changes to Renaming Databases

If any table in the database references—or is referenced by—a table in another database, **sp_renamedb** cannot rename the database. It produces the following error message:

```
Database 'database_name' has references to other
databases. Drop those references and try again.
```

Execute the following query to determine which tables and external databases have foreign key constraints on primary key tables in the current database:

```
select object_name(tableid), db_name(frgndbname)
from sysreferences
where frgndbname is not null
```

Execute the following query to determine which tables and external databases have primary key constraints for foreign key tables in the current database:

```
select object_name(reftabid), db_name(pmrydbname)
from sysreferences
where pmrydbname is not null
```

Before renaming the database, you must use **alter table** to drop the cross-database constraints in these tables. See **sp_renamedb** in the *SQL Server Reference Manual* for more information about renaming databases.

Datatype and Conversion Changes

Changes to datatypes and conversions affect the following areas.

Display Format Changes

The **isql** display format for approximate numeric datatypes now displays additional digits of precision. Maximum units of precision for storage and display are machine dependent. *real* values now display up to 9 digits of precision*; float* values, up to 17 digits. Values are rounded to the last digit on display. Previously, only 6 places to the right of the decimal point were displayed.

All values requiring more digits than the maximum are displayed in scientific notation, that is, a *float* value "1e18" is displayed as such, rather than 1000000000000000000.000000. Note that the new exact numeric types, *decimal* and *numeric*, display the entire number.

Default Type Changes

Numeric literals in SQL statements are now treated as exact numeric types, *int* or *numeric*, unless they are entered using scientific notation. Here are examples of differences you might see:

Table 3-5: Changes in output using numeric literals

SQL Statement	Pre-10.0 SQL Server	10.0
`select 2147483648`	Integer overflow	2147483648.
`select 2*1.12345678`	2.246914	2.24691356.
`select 2.8 * 5`	14.000000	14.0

Datatype Hierarchy Changes

Each system datatype has a **datatype hierarchy**, stored in the *systypes* system table. User-defined datatypes inherit the hierarchy of the system types on which they are based. The following query ranks the system datatypes by hierarchy:

```
select name,hierarchy
from systypes
order by hierarchy

name                           hierarchy
------------------------------ ---------
floatn                                  1
float                                   2
datetimn                                3
datetime                                4
real                                    5
numericn                                6
numeric                                 7
decimaln                                8
decimal                                 9
moneyn                                 10
money                                  11
smallmoney                             12
smalldatetime                          13
intn                                   14
int                                    15
smallint                               16
tinyint                                17
bit                                    18
varchar                                19
```

```
sysname                                  19
nvarchar                                 19
char                                     20
nchar                                    20
varbinary                                21
timestamp                                21
binary                                   22
text                                     23
image                                    24
(28 rows affected)
```

The datatype hierarchy determines the results of computations using values of different datatypes. The result value is assigned the datatype that is closest to the top of the list.

In earlier releases of SQL Server, there were certain exceptions to this rule that have been eliminated as of release 10.0. Specifically, *float* and *numeric* computations have been combined with *money* or *smallmoney*.

Table 3-6: Changes to datatype hierarchy

Datatypes	Pre-10.0 SQL Server	10.0
money and *numeric*	N / A	*numeric*
money and *float*	*money*	*float*

Workarounds include:

- If you are combining *money* and literals or variables, and need results of *money* type, use *money* literals or variables:

  ```
  select moneycol * $2.5 from mytable
  ```

- If you are combining *money* with a *float* or *numeric* column, use the **convert** function:

  ```
  select convert (money, moneycol * percentcol)
      from debts, interest
  ```

Overflow Changes

More information on overflow conditions is provided, and overflow behavior has changed for some datatypes. Additional error messages provide more information about the specific cause of the overflow problem.

Floating Point-to-Character Conversions

In previous releases, conversion from floating point to character in some cases allowed some truncation of floating point decimal data without warning, and in other cases generated warning messages.

Now, all conversions to character data of any length succeed only if no decimal digits are lost. The length of the character string varies depending upon the number being converted and the accuracy of floating point numbers supported by the platform. To guarantee success, use a target of 25 characters.

Numeric-to-Numeric Conversions Requiring Truncation

All conversions to money datatypes round to four places. When an explicit conversion of one numeric value to another results in a loss of scale, the results are truncated without warning. For example, when you explicitly convert a *float*, *numeric*, or *decimal* type to an *integer*, SQL Server assumes you really want the result to be an integer and truncates all numbers to the right of the decimal point.

During implicit conversions to *numeric* or *decimal* types, loss of scale generates a scale error. Use the **arithabort numeric_truncation** option to determine how serious such an error is considered. The default setting, **arithabort numeric_truncation on**, aborts the statement that causes the error but continues to process other statements in the transaction or batch. If you set **arithabort numeric_truncation** to **off**, SQL Server truncates the query results and continues processing.

Integer-to-Character Conversions

Conversions from integer types to character types now return a buffer overflow error (instead of "*") on overflow.

Changes to Conversion Error Messages

The text of the messages for unsupported datatype conversions (message 529) now reads:

```
Explicit conversion from datatype 'type' to 'type'
is not allowed.
```

New error messages provide additional information about datatype conversion errors:

Table 3-7: New type conversion error messages

Error Number	Message
241	Scale error during [explicit | implicit] conversion of *datatype* value '*value*' to a *datatype* field.
245	Domain error during [explicit | implicit] conversion of *datatype* value '*value*' to a *datatype* field.
247	Arithmetic overflow during [explicit | implicit] conversion of *datatype* value '*value*' to a *datatype* field.
249	Syntax error during [explicit | implicit] conversion of *datatype* value '*value*' to a *datatype* field.
265	Insufficient result space for [explicit | implicit] conversion of *datatype* value '*value*' to a *datatype* field.

Change to *between*

The **between** predicate is used in **where** clauses such as this:

```
where colx between val1 and val2
```

val1 must be less than *val2*. In a few situations, the **between** predicate returned correct values, even when *val2* < *val1*. As of release 10.0, these values must be given in the correct order; check your applications that use **between** if you think they may be dependent on the incorrect behavior.

Standard-Style Comments

To comply with SQL92, SQL Server supports standards-style comments, which consist of two minus sign characters. All characters after "--" are ignored. This could cause problems in any SQL query that uses the subtraction of a negative number, such as this:

```
select 5--2
```

This query no longer returns the value 7; it now returns 5. All characters from "--" to the end of the line are ignored.

You can correct any constructions like these by inserting a space between the minus signs or by enclosing the "-2" in parentheses:

```
select 5-(-2)
```

SQL Standards Permissions Requirements for *update* and *delete*

SQL Server supports SQL92 permissions requirements for **update** and **delete** statements. The following table summarizes these requirements:

Table 3-8: SQL92 permissions for update and delete statements

	Permissions Required with *set ansi_permissions off*	Permissions Required with *set ansi_permissions on*
update	**update** permission on columns where values are being set	**update** permission on columns where values are being set And **select** permission on all columns appearing in **where** clause **select** permission on all columns right side of **set** clause
delete	**delete** permission on columns where values are being set	**delete** permission on the table from which rows are being deleted And **select** permission on all columns appearing in **where** clause

If you attempt to **update** or **delete** without having all the required permissions, an exception is generated and the transaction is rolled back. In this case, you will see the following message:

permission_type permission denied on object *object_name* database *database_name*, owner *object_owner*

If this occurs, you need to be granted **select** permission on all relevant columns by the column owner.

SQL Standards Treatment of Nulls

SQL Server supports SQL92-compliant treatment of NULL value operands in equality (=) and inequality (!=) comparisons.

When **set ansinull** is **on**, SQL Server returns an error message if a NULL value operand is eliminated from equality (=) and inequality (!=) comparisons and aggregate functions.

Switching to Chained Transaction Mode

Stored procedures written to use regular Transact-SQL transaction mode may be incompatible with the chained transaction mode required by SQL92. As of release 10.0, all transactions are tagged to identify which mode they use. If you intend to convert existing SQL Server applications to use chained transaction mode, see "Transactions" and **sp_procxmode** in the *SQL Server Reference Manual*.

Using Roles in SQL Server

See "System Administration Roles" on page 2-7 for a description of the new roles. There are two ways to use roles in SQL Server:

- As before, use the "sa" account for system administration tasks. When SQL Server is installed, this account has the System Administrator, System Security Officer, and Operator roles enabled.
- After installing SQL Server, use the "sa" account to create new server logins for users who are to be granted roles. Grant the correct roles to these users, and then lock the "sa" account.

The second method increases user accountability for these highly privileged users, as they perform system administration tasks under their own logins.

However, if you decide to lock the "sa" account, be sure to check all scripts that may contain the "sa" login name and password. These can include scripts that perform backups, run **bcp**, or perform **dbcc** checking. The scripts cannot run if they are meant to run as "sa" and that account is locked. Change the logins in those scripts to the name of one of the users with the correct role.

Repeated Table Names in *from* Clauses

In SQL92, the following construction is not allowed:

```
select * from table1, table1
    [where_clause]
```

In earlier releases, SQL Server handled the **select** statement by returning only the first table and ignoring the self-join. As of release 10.0, this syntax returns an error message.

The correct syntax uses table correlation names:

```
select * from table1 t1, table1 t2
    [where_clause]
```

A similar construction using repeated table names in an **update** statement also returns an error message. The construction is:

```
update table1
    set a = a + 1
    from table1
    where b = 5
```

The **update** statement also creates a self-join on the table named after **update** and the table in the **from** clause. It returns unexpected results because it updates all rows from the first-mentioned table, seemingly ignoring the restriction in the **where** clause. The correct syntax also uses table correlation names in the **from** clause:

```
update table1
    set a = a + 1
    from table1 t1
    where t1.b = 5
```

Note that the SQL92 standard does not include the use of the **from** clause in an **update** statement.

Column Names Required for *select into* with Expressions

Previous releases of SQL Server allowed NULL column headings in tables created by **select into**. These NULL headings resulted from any use of an expression in the select list: an aggregate function, constant, built-in function, arithmetic expression, concatenation, and so on. As of release 10.0, you must provide a column heading. The heading must be a valid identifier, that is, it must begin with an alphabetic character or an underscore, and contain only letters, numbers, and a limited set of symbols (#, @, _, $, ¥, or £). It cannot contain embedded spaces.

Check all applications that use **select into**. Examples of select list items that require headings are:

- An aggregate function:

  ```
  avg(advance)
  ```

- Arithmetic expression:

  ```
  colname * 2
  ```

- String concatenation:

  ```
  au_lname + ", " + au_fname
  ```

- Built-in function:

  ```
  substring(au_lname,1,5)
  ```

- Constant:

  ```
  "Result"
  ```

There are three ways to specify the column heading:

- Before the expression or aggregate function, with "=":

  ```
  select title_id, avg_advance = avg(advance)
      into #tempdata
      from titles
  ```

- After the expression or aggregate function:

  ```
  select title_id, avg(advance) avg_advance
      into #tempdata
      from titles
  ```

- After the expression or aggregate function, with **as**:

  ```
  select title_id, avg(advance)as avg_advance
      into #tempdata
      from titles
  ```

The column names must be valid identifiers, unless you are using the delimited identifier feature.

Changes to Correlation Name Handling

As of release 10.0, queries that include correlation names conform to the requirements of SQL92. In earlier releases, statements that specified correlation names but did not use them consistently still returned results. These statements now return errors. For example, this statement is incorrect:

```
select title_id
    from titles t
    where titles.type = "trad_cook"
```

The correct query is:

```
select title_id
    from titles t
    where t.type = "trad_cook"
```

When this same type of correlation is included in a subquery, no error is reported, but queries may return different results. Here is an example:

```
select *
from mytable
where columnA =
    (select min(columnB) from mytable m
         where mytable.columnC = 10)
```

In earlier releases, the reference to *mytable.columnC* in the subquery referred to the *mytable* in the subquery. Now, this query is a correlated subquery, and *mytable.columnC* refers to the outer table *mytable*.

If the query needs to refer to the *mytable* in the subquery, the correct statement is:

```
select *
from mytable
where columnA =
    (select min(columnB) from mytable m
        where m.columnC = 10)
```

Subquery Changes

Several problems relating to subqueries have been fixed as of SQL Server release 10.0. This section describes old and new behaviors in detail. Examine applications that use subqueries to determine if your application relies on a behavior that has been corrected. This may be especially true of the first two items below.

The following types of subqueries have changed:

- Correlated subqueries using **in** and **any**
- Subqueries using **not in** when the subquery returns NULL values
- **in** and **any** subqueries combined with **or**
- >**all** and <**all** with subqueries that return no rows

- Subqueries that suppressed duplicate values from the outer query
- Aggregate queries with **exists** when there are duplicate values
- Correlated subqueries with **distinct** when used with **in**

In addition, support has been added for:

- The use of =**all** to introduce subqueries
- The use of aggregates in **where** clauses, if the following conditions are true:
  - The aggregate appears in a subquery that is in a **having** clause
  - The aggregated value refers only to tables named in the **from** clause of the outer query
  - The aggregated value does not refer to tables named in the **from** clause of the subquery itself

An **in** subquery (or a subquery using **any**) performs an existence check. The subquery evaluates to TRUE or FALSE. Correlated subqueries return a single TRUE or FALSE value for each row in the outer query. A subquery should not cause any row from the outer query to be returned more than once.

Subqueries using the **in** predicate used to return duplicates. The same is true for subqueries using **any**. For example, using the *pubs2* database:

```
select pub_name
    from publishers
    where pub_id in
        (select pub_id
             from titles)
```

Or:

```
select pub_name
    from publishers
    where pub_id = any
        (select pub_id
             from titles)
```

In pre-10.0 releases of Transact-SQL, these queries both returned a *pub_name* for each row in the inner query:

```
pub_name
----------------
New Age Books
New Age Books
New Age Books
New Age Books
New Age Books
Binnet & Hardley
Binnet & Hardley
Binnet & Hardley
Binnet & Hardley
Binnet & Hardley
Binnet & Hardley
Binnet & Hardley
Algodata Infosystems
Algodata Infosystems
Algodata Infosystems
Algodata Infosystems
Algodata Infosystems
Algodata Infosystems
```

They now return:

```
pub_name
------------------------
New Age Books
Binnet & Hardley
Algodata Infosystems
```

For more information, see "Duplicates in Subquery Results" on page 3-7.

Subqueries Using *not in* or *all* with NULL

A subquery using the **not in** or **all** predicate returns a set of values for each row in the outer query. If the value from the outer query is not in this set, the **not in** returns TRUE. If the set returned by the subquery contains no matching value, but contains a NULL, the **not in** returns UNKNOWN. This is because NULL stands for "value unknown," and it is impossible to tell whether the value you are looking for is in a set containing an unknown value. In earlier releases, SQL Server erroneously considered this case to be TRUE.

For example, using the *pubs2* database:

```
select pub_id
    from publishers
    where $100.00 not in
        (select price
         from titles
         where titles.pub_id = publishers.pub_id)
```

In previous releases, this query returned:

```
pub_id
------
0736
0877
1389
```

It now returns the correct result because publishers 0877 and 1389 have rows where the price is NULL:

```
pub_id
------
0736
```

in and *any* Subqueries Using *or*

When a subquery using the **any** or **in** predicate is combined with **or** in a logical expression, the query should return rows if the subquery evaluates to FALSE, when the other clause of the logical expression evaluates to TRUE. The same is true of subqueries using **any.** For example, using the *pubs2* database:

```
select pub_name
    from publishers
    where pub_id in
        (select pub_id
             from titles
             where title = 'No Such Book')
    or pub_id = '1389'
```

Or:

```
select pub_name
    from publishers
    where pub_id = any
        (select pub_id
             from titles
            where title = 'No Such Book')
    or pub_id = '1389'
```

These queries both returned no result rows in pre-10.0 SQL Server.

They now return:

```
pub_name
----------------------------
Algodata Infosystems
```

>*all* and <*all* When the Subquery Matches No Rows

When a subquery using the **>all** or **<all** predicate matches no rows, the subquery should evaluate to TRUE. Instead, it evaluated to FALSE in pre-10.0 SQL Server. For example, using the *pubs2* database:

```
select title
    from titles
    where advance > all
        (select advance
         from publishers, titles
        where titles.pub_id = publishers.pub_id
    and pub_name = 'No Such Publisher')
```

This query returned no rows. It now returns all of the rows, the correct result.

Correlated Subqueries Suppressing Duplicates from Outer Query

A correlated subquery returning a data value to the outer query sometimes caused the outer query to fail to return duplicate rows if all of the columns in the outer query are also used in the subquery. For example, using the *pubs2* database:

```
select pub_id,
     (select count(*)
     from publishers
    where publishers.pub_id = titles.pub_id)
from titles
```

In previous versions, this query returned these results:

```
pub_id
------ -----------
0736            5
0877            7
1389            6
```

It now returns:

```
pub_id
------ -----------
1389             6
1389             6
0736             5
1389             6
0877             7
0877             7
0877             7
1389             6
1389             6
1389             6
0877             7
0736             5
0736             5
0736             5
0736             5
0877             7
0877             7
0877             7
```

Aggregate Queries with *exists* Subqueries

A query that selects an aggregate and has a correlated subquery that uses the **exists** predicate produced the wrong answer if the table in the subquery's **from** list had duplicates. For example, using the *pubs2* database:

```
select count(*)
    from publishers
    where exists
        (select *
         from titles
        where titles.pub_id = publishers.pub_id)
```

In pre-10.0 SQL Server, this query returned the value 18. It now returns the correct response, 3. It is similar to the first problem in this discussion, as it has to do with duplicates in an "existence-type" subquery to cause duplicates in the outer query.

Correlated Subqueries Using *distinct* and the *in* Predicate

A correlated subquery introduced with the **in** predicate and having a **select distinct** in the subquery previously returned no rows. For example, using the *pubs2* database:

```
select pub_name
    from publishers
    where pub_id in
        (select distinct pub_id
         from titles
         where pub_id = publishers.pub_id)
```

This query returned no rows in pre-10.0 SQL Server. It now returns:

```
pub_name
----------------------------------------
New Age Books
Binnet & Hardley
Algodata Infosystems

(3 rows affected)
```

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W