**SQL Server Vs Sybase**

| �  **Feature** | **Sybase T-SQL** | **Microsoft T-SQL** |
| --- | --- | --- |
| **Indexes** | **B-Tree indexes** | **B-Tree indexes** |
| **Tables** | **Relational tables,**  **Temporary tables** | **Relational tables,**  **Temporary tables** |
| **Triggers** | **login triggers,**  **AFTER triggers** | **AFTER triggers,**  **INSTEAD OF triggers** |
| **Procedures** | **Sybase T-SQL statements,**  **SQLJ stored procedure (SQL**  **wrapper to a Java static method)** | **Microsoft T-SQL statements** |
| **User-defined functions** | **SQLJ functions (SQL wrapper**  **to a Java static method)** | **Scalar functions,**  **Inline table-valued functions,**  **Multistatement table-valued functions** |
| **Indexed Views** | **Not Supported** | **Supported** |
| **Multiple Triggers** | **Not Supported** | **Supported** |
| **Arrays** | **Not Supported** | **Not Supported** |

## **SQL Server 2000 and Sybase ASE 12.5 limits**

Here you can find some SQL Server 2000 and Sybase ASE 12.5 limits:

| **Feature** | **SQL Server 2000** | **Sybase ASE 12.5** |
| --- | --- | --- |
| **database name length** | **128** | **30** |
| **table name length** | **128** | **30** |
| **view name length** | **128** | **30** |
| **column name length** | **128** | **30** |
| **index name length** | **128** | **30** |
| **triggers name length** | **128** | **30** |
| **procedures name length** | **128** | **30** |
| **defaults name length** | **128** | **30** |
| **rules name length** | **128** | **30** |
| **cursor name length** | **128** | **30** |
| **nested triggers levels** | **32** | **16** |
| **columns referenced by a view** | **4096** | **1024** |
| **columns per SELECT statement** | **4096** | **1024** |
| **max indexes per table** | **250** | **250** |
| **bytes per index** | **900** | **600 for 2K page,**  **1250 for 4K page,**  **2600 for 8K page,**  **5300 for 16K page** |
| **columns per index** | **16** | **31** |
| **parameters per stored procedure** | **1024** | **255** |
| **text of a stored procedure** | **250MB** | **16MB** |
| **nested stored procedure levels** | **32** | **16** |
| **nested subqueries** | **32** | **16** |
| **tables per SELECT statement** | **256** | **50** |
| **columns in GROUP BY, ORDER BY** | **Limited only by number**  **of bytes (8060)** | **31** |

# Advanced Topics

## Recovery Issues

Sybase databases can be backed up with users still accessing the database. The users might notice some degradation in performance.

Sybase databases are dumped (backed up) via two different methods. The first method is called a "full backup". It represents a total snapshot of the database.

The second method is called a transaction log backup. This is an incremental backup. Put another way, transaction log backups only save changes made since the last transaction log dump.

The advantages to the transaction log dumps is that they contain much less data and are therefore smaller and faster.

In some cases, transaction log backups can be done once every few minutes.

To recover a database, it is necessary to load the database from the last full dump, followed by loads of the transaction log dumps in the order that they were originally performed.

Transaction logs are also an integral part of the Sybase data caching mechanism. A committed update to data will always generate an entry in the transaction log. However, the actual data may be held in cache for a while before it's written back to the hard disk. If the server crashes and reboots, the transaction logs are used to recreate the data that was in cache and had not yet been written to a hard disk.

## Transaction Logs Filling Up

Sybase keeps the transaction log information in a special table in each database called "syslogs". Every write to a database is recorded in the syslogs table.

If too many writes occur before a transaction dump occurs, the syslogs table can fill up. At this point, it is no longer possible to dump the transaction logs. No further write operations can occur in that database until the transaction log is cleared by the system administrator.

It is possible to set the database so that the syslogs table is automatically cleared on a periodic basis. This will help prevent the transaction logs from filling up, but prevents the use of transaction log dumps. Full database dumps would have to be used instead.

## **Holdlock** verses Browse Mode

What about the case were a client GUI program pulls up a row's worth of information, lets the user modify the row, and then writes the row of data back.

The problem here is that a 2nd client may have changed the data after the 1st client read it, but before the 1st client had a chance to write it back.

We have previously talked about the "holdlock" option on the select statement. Remember that this causes a selected row to be locked until the end of the current transaction. Holdlock could be used here to prevent the 2nd client from modifying the holdlocked row.

Sometimes holdlock is too restrictive. Specifically, what if 99.99% of the time your application ends up not modifying the row it just read? What if your end user fetches the row to his screen and then goes out to lunch. In this case you would be needlessly holding up other transactions. This is especially true if there are other transactions which need to scan large sections of the table.

As an alternative to holdlock, Sybase has a feature called browse mode.

Basically browse mode works as follows:

* There is a extra timestamp field in the table indicating the time the row was last updated.
* When the client reads the row, he also reads the timestamp field.
* When the client updates the row, he also updates the timestamp value he previously read.
* Sybase automatically compares the timestamp value written back by the client to the timestamp value in the row in the table.
* If they're different, then someone else has updated this row since it was read by us. Sybase aborts the update operation. The client should be sent an error message politely saying please try again.
* If they're the same, Sybase performs the update and automatically updates the timestamp field with a new value.

Browse mode should be used when the frequency of update collision is low. You don't want to blow off too many client update operations.

Sybase only officially supports browse mode at the dblib C programming level. However, it does seem to work when used at the Transact-SQL level.

Sybase uses a special datatype called "timestamp" to implement browse mode. This datatype does not translate to normal date/time format, instead it is used as sort of a unique identifier. (In fact, the name timestamp is a misnomer. It's actually a counter).

The following is an example of browse mode operations:

     create table mytable( f1 int, f2 char(10), timestamp )

     /\* execute create table command \*/

     create unique index f1\_idx on mytable(f1)

     /\* execute create index command \*/

     declare @f1 int, @f2 char(10)

     declare @timestamp timestamp

     /\* The timestamp field is automatically filled in \*/

     insert into mytable values( 1, "abc" )�

     insert into mytable values( 2, "def" )

     select @f1=f1, @f2=f2, @timestamp=timestamp

     from mytable

     where f1=1

     for browse

     /\* simulate user processing \*/

 waitfor delay "00:01:00" /\* waitfor 60 seconds \*/

     /\* let somebody else go in and update row 1 \*/

     /\* While this does lock the the page, it does so only

     \*\* briefly, just before the update occurs.

     \*\* Note that the tsequal will raise an error if a match

     \*\* is not found.

     \*/

     begin transaction

     if exists ( select \* from mytable holdlock

                      where f1=@f1

                      and tsequal( timestamp, @timestamp)

                   )

       begin

          select old\_timestamp=timestamp from mytable

          where f1=@f1

          update mytable    /\* this changes the timestamp \*/

          set f2="ghi",

          where f1=@f1

          select old\_timestamp=timestamp from mytable

          where f1=@f1

    end

     commit transaction

## **tempdb**

The tempdb database is a special Sybase supplied database that comes in each server. Data in tempdb is not permanent. The tempdb database is cleared each time the server reboots.

Any queries which do any sort of sort operation implicitly use tempdb. These queries include select statments with group by's or order by's.

Be careful, if you select a large set of rows and use an order by or group by clause, you query may fail because tempdb isn't big enough.

The system administrator can extend the size of tempdb.

## Sybase Temporary Tables

Any table name that starts with a "#" is automatically put into the tempdb database.

Furthermore that temporary table is only seen by the current client database login. If someone else creates a temporary table with the same name, it will be a different one from yours.

For example:

     create table #mytable(

               title\_id char(6),

               total\_money\_made money )

     insert into #mytable

          select title\_id,

                total\_money\_made=sum( ytd\_sales\*price )

          from titles

          group by title\_id

     select \* from #mytable

     drop table #mytable

If you do not explicitly drop the temporary table, it will go away when the current database login session is ended.

### Pro's & Con's of Temporary Tables

Temporary table are very useful for breaking down complicated selects with multiple subqueries into a series of easily understood steps.

It takes a second or two to create a temp table. Also, each time a temp table is loaded with inserted data, there are additional writes that might not have occured in a single select.

Be careful when referring to temp tables. Only use the phrase "temp table" with tables that start with a "#". Specifically, if you have a regular table that you temporarily use and then drop, refer to it as a working table.

## **bulkcopy**

Sybase supplies operating system utilities for importing/exporting data between a Sybase database and a flat file. The utility is called "bcp" (for bulkcopy).

In addition, the Sybase kernel supports special functionality to help speed the loading of data via the bulkcopy utility.

If a database has the bulkcopy option set, and a table in that database has no indexes, then a bulkcopy operation into that table will work in a special fast mode. Data rows inserted this way will not be transaction logged.

See the Commands Reference Manual, Utilities Section. Also see the System Administration Guide.

## Select/into verses Insert/select

Sybase supports a special way to copy data from one table to another in the same server.

The normal way to copy a table in SQL is to use the insert..select statement. For example:

     create table #mytable( au\_id char(11), zip char(5) )

     {execute create table}

     insert into #mytable (au\_id, zip )      /\* normal way \*/

       select au\_id, zip from authors

       where zip like '9%'

     drop table #mytable

This method performs transaction logging for each row inserted. Also the "inserted to" table must exist before the insert..select statement is run.

Sybase also supports an "select..into" statement which behaves slightly differently. The significant features of a select..into are as follows:

* Transaction logging is not performed.
* The "inserted to" table is created by the select..into table.

The select..into statement is often used in conjunction with Sybase temporary tables.

For example:

     select au\_id, zip

     into #mytable

     from authors

     drop table #mytable /\* good software hygiene \*/

Note that #mytable is automatically created on the fly with a set of columns whose names and datatypes match the output of the select statement.

The select..into statement can only be used if the system administrator has turned off transaction logging for the database. Remember that turning off transaction logging effectively prevents the use of incremental dumps (transaction log dumps).

### When Should Select**..**Into Be Used

#### Minimizing Transaction Log Usage

Any operation that move a large amount of data between tables is a candidate for select..into operations. In some cases, the transaction logs are just not big enough to perform a regular insert..select.

Also, transaction logging takes time. If performance is an issue and incremental backups are not needed, then select..into makes sense.

### Duplicating Table Definitions

If you perform a select..into with a where clause which is always false, the new table is created with a set of columns which matches the select list, but the new table will be empty. For example:

          select \*

          into #mytable

          from authors

          where "I win" = "the lottery" /\* never true \*/

This will create an empty Sybase temporary table #mytable with all the columns from the authors table.

## System Tables

In the Commands Reference, appendix B, there is a definition of internal database tables used by Sybase.

Most of these table are just normal Sybase tables with access limited to select only operations.

At the end of the appendix is a two page block diagram of the system tables.

The two pages in the block diagram correspond to the two classes of system tables: Tables which pertain to the whole server and tables pertain to a individual databases.

An example of a table which pertains to a whole server is the syslogins table. This table defines server logins and associated passwords. Remember a person has one login to a server, but can be added as a database user to specific databases within that server.

System tables which pertain to the whole database are kept in the "master" database.

Here's are some sample queries which access system tables:

     select dbid, name from master..sysdatabases

     /\* You can't select the password field from syslogins \*/

     select suid, name from master..syslogins

An example of a table which occurs in each database is the sysusers table. Each database contains a sysusers table which defines the server logins that have corresponding database user entries in that database.

Here are some sample queries for system tables which occur in each database:

     select uid, name from sysusers

     select id, name, type

     from sysobjects

     where type="U" or type="P"    /\* user tables or procs\*/

Some system tables which occur in each database join to the central "master" database. Here's an example of an SQL query which performs a join against the sysusers table in the current database against the syslogins table in the master database:

     select    login\_id=master..syslogins.name,

               database\_user=sysusers.name

     from master..syslogins, sysusers

     where master..syslogins.suid = sysusers.suid

There is no way, in Transact-SQL, to perform this query for all databases in a server.

## Text and Image Fields

Sybase has two special datatypes called "text" and "image".

The text datatype can store arbitrary length text strings with lengths from 0 to 2 gigabytes long.

The image database can store arbitrary length binary values whose size varies from 0 to 2 gigabytes.

For both text and image datatypes, internal storage is allocated in 2k bytes (2048). Even if you only put one character into a text field, 2k bytes are allocated internally.

A text field can be used to hold large portions of textual material, eg. chapters from a book.

Here's an example of a select on a text field:

     select au\_id, copy

     from blurbs

     where au\_id="486-29-1786"

The only way text fields can be used in a "where" clause is with the "like" statement. For example:

     select au\_id from blurbs

     where copy like '%Etiquette%'

There are special rules for using text/image datatypes. See the following sections for more information on text/image datatypes:

     Users Guide

          Chp2, Selecting Text and Image Values

          Chp9, Changing Data with Writetext

     Commands Reference

          Chp2, Text/Image Datatypes

          Chp2, Text/Image Functions

          Chp2, Readtext

          Chp2, Writetext

## **Showplan**/**noexec** Query Tuning

Sybase physically accesses tables in one of two ways. The first way involves Sybase reading each row of a table and throwing away the rows it doesn't want. This is referred to as "scanning" a table.

The second way involves Sybase using an index to access only select rows in a table.

Query performance can vary dramatically depending on whether a table is being scanned or read via an index. For example, with a 500 megabyte table, scanning a table to find a particular row can take half an hour. The same query using an index can take only a few seconds.

You can use the "showplan" feature to analyze the query plan Sybase intends to use. Additionally, you can also use the "noexec" feature to indicate you don't want to actually run the query, just evaluate.

For example:

     set showplan on

     {execute or go}

     set noexec on

     {execute or go}

     select \* from {large\_table}

     where ....

The showplan output will have the form:

     {lots of garbage}

     Type of query is {select/insert/update/delete}

     FROM TABLE

     {table name}

     Nested Iteration {or something else obscure}

     Table Scan {or} Using Index {or} Using Clustered Index

To turn off noexec type the following:

     set noexec off

     {execute or go}

To turn off showplan, first turnoff noexec, then type the following:

     set showplan off

     {execute or go}

The End.