

Identity Gaps

Revisited: The Long-Awaited Solution Has Arrived

By Rob Verschoor

*A new solution for repairing
identity gaps in ASE 12.5.0.3
and ASE 12.5.1*

In 1998, I wrote my first article for the *ISUG Technical Journal* about one of my favorite ASE topics: identity columns. The article, entitled “A Better Method for Dealing with Identity Gaps,” (still available at www.sypron.nl/idgaps.html) presented a design trick to avoid identity gaps: those huge gaps in identity columns where the column values jump enormously (typically, some five billion units) after a *shutdown with nowait*. These huge identity values often cause problems for applications that do not expect, and often cannot handle, such large values.

In this article, I assume the user is at least a bit familiar with identity columns and identity gaps. If not, you may want to read my earlier article first.

At the time of the first article, the main problems with identity columns were that (1) identity gaps simply could not be avoided, but mostly that (2) once they had occurred, the gaps could not be repaired, or only in a clumsy, time-consuming way at best. My article offered a workaround for these problems, but since it required data model changes (albeit minor ones), this approach was not always feasible.

In ASE 12.0, these problems were partly addressed with the introduction of the *identity_gap* setting. With this table attribute, the size of an identity gap could

be limited for a specific table, potentially reducing the problem. However, explicit DBA action remained necessary, since *identity_gap* is undefined by default. Therefore, the possibility of getting huge identity gaps could still not be precluded completely (in practice, I've often seen DBAs forget to define *identity_gap*).

However, I am glad to report that the solution has finally arrived: ASE 12.5.0.3 and ASE 12.5.1 have delivered the long-awaited pieces of missing functionality, making repairing identity gaps as easy as it should always have been.

Why Identity Gaps Occur

The reason that identity gaps occur lies in the purpose of the identity column feature. It was designed as a high-performance mechanism for assigning unique, sequential data table values for multiple concurrent users. To achieve this goal, the identity column counter is cached in memory, and written to disk in the table's OAM page only periodically instead of after each assignment. Furthermore, identity column assignments are not transactional, providing a high level of concurrency. Indeed, an identity column far outperforms a functionally equivalent “key counter table” algorithm, where the last assigned key value is stored in a database table.

The downside of not writing the memory-based identity counter to disk



Rob Verschoor is a freelance consultant in the Netherlands, specializing in Sybase ASE, and the author of several books. He can be reached at rob@sypron.nl.

continuously is that a *shutdown with nowait* or a system crash will not record the last value of the identity counter on disk. To avoid duplicate identity values after an ASE restart, the identity counter on disk takes a big jump forward by default, creating the identity gap. When ASE shuts down properly, this disk-based counter value is overwritten with the actual identity counter from memory, so no gaps occur after restarting ASE. However, this does not happen in an ungraceful shutdown and can lead to identity gaps. Again, the DBA should always define the *identity_gap* setting to limit the size of these gaps and therefore the seriousness of the problem.

Against the background of a performance-oriented design trade-off, this mechanism makes sense, and identity gaps can be regarded as the price one has to pay for high-speed sequential number generation by identity columns. However, the real problem for DBAs has always been that it is next to impossible to repair identity gaps once they have occurred. It is in this last area where significant improvements have now been made to ASE.

Identity Gaps Due to Other Causes

The most common cause of identity gaps is a *shutdown with nowait*. It should also be mentioned that there are other ways in which an identity gap can occur, although these are relatively rare. One scenario involves making a database dump for a database containing identity tables. When this dump is loaded, and a row is inserted into the identity table in the just-loaded database, an identity gap may occur. Something similar can happen after loading a transaction log dump.

It is also possible to get identity gap look-a-likes when using the *identity grab size* feature (although this is not a real identity gap as defined in this article).

Much more can be said about the technical background of the different scenarios for getting identity gaps, but due to space limitations, they are not included in this article. Interested readers may refer to Chapter 11 of my book *Tips, Tricks & Recipes for Sybase ASE* (www.sypron.nl/ttr) for a full and detailed discussion.

Repairing an Identity Gap

When stuck with an identity gap, a DBA needs to perform two actions to repair it: Correct the insert identity column values at the upper side of the gap, and reset the identity counter downwards for future insertions (not necessarily in this order). In the following example, an identity gap has occurred between values 10031 and 5000002:

```
1> select invoice_nr from invoices order by 1
2> go

invoice_nr
-----
...
10028
10029
10030
10031
5000002 ← identity gap: value 10032 was expected instead!
5000003

(10033 rows affected)
```

To repair this gap, the DBA should do the following:

1. Update the identity column values at the upper side of the gap; in this example, update 5000002 and 5000003 to 10032 and 10033, respectively;
2. Reset the identity counter downwards, so that a correct value is generated when the next row is inserted; in this example, the identity counter should be set to 10034.

Unfortunately, neither of these actions was possible prior to 12.5.0.3, at least not in a supported or DBA-friendly way. We'll examine the options available to a DBA in different versions of ASE.

Repairing an Identity Gap in ASE Pre-12.5.0.3/12.5.1

In ASE versions earlier than 12.5.0.3, prevention is historically the only supported remedy against identity gaps. This consists of setting *identity_gap* for each identity table in ASE 12.0, and possibly using my earlier data model design trick mentioned in the introduction of this article.

Once an identity gap occurs, the only supported way to repair it is to drop and recreate the table and copy the data rows. For large tables, this approach is often not acceptable because it requires too much downtime.

This does not mean there is no solution at all. In fact, both repair actions can be performed in pre-12.5.0.3/12.5.1, albeit in a nonstandard and unsupported manner, as described below.

Updating Identity Column Values in ASE Pre-12.5.0.3

To update the identity column values at the upper side of the identity gap, in ASE pre-12.5.0.3, a regular **update** statement won't work since ASE does not allow identity columns to be updated:

```

1> update invoices
2> set invoice_nr = 10032
3> where invoice_nr = 5000002
4> go
Msg 7744, Level 16, State 1
Server 'SYB120', Line 1
Illegal attempt to update identity field 'invoice_nr'.

```

However, a tricky and unsupported workaround can be performed by manually removing some status bits from system tables. Specifically, this concerns bit 128 in `syscolumns.status` and bit 64 in `sysobjects.sysstat2`. By setting these bits to 0, the identity property is temporarily suppressed and the column, now behaving as a regular numeric column, can be updated normally. After restoring these bits, the identity column behaves normally again.

I have written two stored procedures, `sp_identity_remove` and `sp_identity_restore`, to properly enable and disable these status bits. These procedures can be downloaded from www.sypron.nl/idgaps.html.

It should be noted that this workaround, as well as these stored procedures, are undocumented and unsupported by Sybase. Great care should be taken when using them; for example, there must be no user access to the table while performing these operations. Despite these drawbacks, this workaround can be a real lifesaver when faced with identity gaps.

Updating Identity Column Values in ASE 12.5.0.3

In ASE 12.5.0.3, the first part of repairing an identity gap has been fixed. Using the new command `set identity_update MyTable on`, identity column values can now be updated with a regular `update` statement:

```

1> set identity_update invoices on
2> go
1> update invoices
2> set invoice_nr = 10032
3> where invoice_nr = 5000002
4> go
1> set identity_update invoices off
2> go

```

Note that `set identity_update` is a session-specific setting similar to `set identity_insert`. ASE does not check whether the update causes duplicate identity values unless there is a unique index on the identity column. Per session, only one table can be marked for `identity_insert` or `identity_update` (or both) at a time.

Resetting the Identity Counter in ASE Pre-12.5.1

The ability to reset the identity counter downwards has long been on the wish list of many DBAs who have been hit by identity gaps. Unfortunately, there was no practical way of doing this until ASE 12.5.1 arrived (see below). In pre-12.5.1, the only supported solution is to completely rebuild the table, but especially for large tables this is not practical.

Fortunately, an unsupported workaround exists for pre-12.5.1. But before we proceed, be warned. This method is convoluted, difficult to use, and may cause problems itself!

This workaround involves the undocumented command `dbcc object_atts`, which can set the identity counter on the table's OAM page to any desired value. Unfortunately, this command is poorly documented and difficult to use. It also requires a `shutdown with nowait` immediately afterwards, which can lead to identity gaps in other identity tables (this operation must therefore be performed after ASE has been stopped and restarted without any user activity).

Some of the pain caused by `dbcc object_atts` can be avoided by using the stored procedure `sp_identity` (downloadable from www.sypron.nl/idfix.html), but this method is still nowhere near user-friendly. Yet, it can be the only option available to a DBA in pre-12.5.1.

Resetting the Identity Counter in ASE 12.5.1

In ASE 12.5.1, perhaps the most-wanted identity-related feature was finally implemented: It is now possible to reset a table's identity counter to any desired value with an easy-to-use, supported command. Technically, this is a new option, `identity_burn_max`, for the existing procedure `sp_chgattribute`:

```
sp_chgattribute invoices, 'identity_burn_max', 0, '10034'
```

Under the covers, `sp_chgattribute` calls `dbcc set_identity_burn_max` to set the identity counter in the table's OAM page to the specified new value (here, 10034). Note that the new value must be specified as a quoted string, even though it is an integer, because an identity column can be up to 38 digits long.

While `sp_chgattribute` is modifying the table's OAM page, no ASE process should be accessing the table to avoid possibly damaging interference. For this reason, `sp_chgattribute` first takes an exclusive-table lock on the table before calling `dbcc set_identity_burn_max`. In fact, this `dbcc` command will do the same should it be called directly.

Thanks to this new functionality, it is no longer necessary to mess around with `dbcc object_atts` in ASE 12.5.1 or later.

Fine-Tuning *sp_chgattribute*

While the new *identity_burn_max* option for *sp_chgattribute* is very welcome indeed, it actually contains a small hidden problem. It appears to contain a built-in check to ensure the identity counter is not set to a value lower than the highest identity column value currently in the table. In other words, when the highest value of the identity column for all data rows is 100, *sp_chgattribute* will not allow the user to set the identity counter to anything lower than 100 (so the value assigned to the next inserted row would be 101).

This check is designed to protect a DBA against himself. Should the identity counter be set to 99, the next generated value will be 100; since this value already exists in the table, we'd have identical identity column values. This check by *sp_chgattribute* guarantees such a scenario will not occur.

Unfortunately, the situation described here may be exactly what many DBAs will encounter in practice. When there has been a large identity gap (say, the classical case of a 5-billion units jump), the very first thing a DBA might need to do is reset the identity counter so that regular processing (i.e., inserts) can proceed. The DBA can then start to update the identity column values at the upper side of the gap. However, the check built into *sp_chgattribute* would make this scenario impossible. The DBA would then be forced to perform these steps in reversed order, while new rows keep being inserted with high identity values; or to suspend all application activity on the table until the repair has been completed.

Fortunately, there is a simple way to bypass this check. By editing the source code of *sp_chgattribute* and recreating the procedure, an additional option can be added that does not perform this check. I have placed an example of such edited SQL source code on my website (with the consent of Sybase Engineering) at www.sypron.nl/idgaps.html.

When installed, this modified version of *sp_chgattribute* accepts another option called *identity_burn_max_force* to reset the identity column value, regardless of the currently existing identity column values in the table. Of course, by using this approach, you accept all responsibility for avoiding duplicate identity values!

This modified version of *sp_chgattribute* is intended as a suggestion for modifying the code yourself, rather than as a replacement to be installed blindly. Please inspect all code changes (about ten of them) before installing this code in your server.

Other Identity Enhancements in 12.5.0.3

Apart from *set identity_update*, ASE 12.5.0.3 introduced two new identity-related built-in functions:

- ◆ *next_identity(table_name)* returns the identity column value that will be assigned to the next row inserted into this table. This function can be used to determine whether there is an identity gap for a table (for example directly after an ASE restart), but without actually inserting a row. Note that there is no guarantee that this next identity value is assigned to a row inserted by your session; it goes to the user who comes first.
- ◆ *identity_burn_max(table_name)* returns the next value that the identity column would contain if a *shutdown with nowait* occurred now.

Automatically Detecting and Repairing Identity Gaps

With the recent new identity-related features, it is possible to create a stored procedure to automatically detect whether a table may be hit by an identity gap, and if so, to repair the gap automatically. An example of such a procedure (named *sp_idgap_repair*) can be found at www.sypron.nl/idgaps.html.

Identity Gap Recap

Let's just summarize the main points about identity gaps:

- ◆ Identity gaps can result from a *shutdown with nowait* or, less common, by certain database dump/load sequences.
- ◆ In ASE 12.0, *always* define the *identity_gap* setting for all permanent tables with an identity column; this will limit the size of potential identity gaps. Not defining *identity_gap* should be considered an error!

To detect whether there is an identity gap for a certain table:

- ◆ In pre-12.5.0.3, this can only be determined by inserting a row and comparing the generated identity value with the highest value in the table prior to this insert.
- ◆ In ASE 12.5.0.3, no row needs to be inserted, since *next_identity()* provides the same information.

When faced with an identity gap, it usually needs to be repaired. Take the following steps:

- ◆ To update existing identity column values at the upper side of the gap:
 - ◆ In ASE 12.5.0.3 or later, use *set identity_update* to update the identity column values as needed.
 - ◆ In pre-12.5.0.3, the only option is to temporarily suppress the identity behavior by resetting status bits in *syscolumns* and *sysobjects*, assuming you don't want to delete those data rows and re-insert them after resetting the identity counter downwards. Stored procedures to toggle these status bits can be found at www.sypron.nl/idgaps.html.

- ◆ To reset the identity counter downwards:
In ASE 12.5.1 or later, use `sp_chgattribute 'identity_burn_max'` to instantly reset the identity counter. In fact, you may want to download the improved version from www.sypron.nl/idgaps.html to override a built-in check that is likely to be in your way.

In pre-12.5.1, the only feasible option is to use `dbcc object_atts`, as described at www.sypron.nl/idfix.html (assuming you don't want to drop and recreate the entire table). To avoid identity gaps completely, consider the two-table design trick described at www.sypron.nl/idgaps.html.

Avoiding Identity Gaps After All

In 2002, four years after writing my first article about identity gaps, I discovered a way to avoid identity gaps in case of a *shutdown with nowait*, even in ASE 11.0. This works as follows:


1. Before executing *shutdown with nowait*, run the command `dbcc cacheremove(database_name, table_name)` for every permanent identity table.

2. Next, run a `checkpoint` command in each database where the above action was performed;
3. Finally, issue `shutdown with nowait`.

Steps 1 and 2 have the effect of forcing the memory-based identity counter to be written to the table's OAM page on disk, thus avoiding identity gaps. Note that this will work only when there are no further inserts into the table after step 1. Also note that `dbcc cacheremove` is an undocumented and unsupported command, so all disclaimers apply. Nevertheless, I am sure that many hours of DBA desperation could have been avoided if this trick had been known earlier. `dbcc cacheremove` can also help to avoid identity gaps caused by a database dump&load, but not for dump&load of a log dump.

Conclusion

There are no reasons to avoid identity columns for fear of identity gaps. Since ASE 12.0, the size of identity gaps can be limited, and the introduction of new features in ASE 12.5.0.3 and ASE 12.5.1 makes it easy to repair identity gaps, should they occur anyway. Lastly, an undocumented dbcc command can be used to avoid identity gaps even in ASE pre-12.0. ■



Got a great tip for DBAs or developers? Some great advice, a new method, or excellent syntax?

Consider writing an article for the ISUG Technical Journal!

Becoming an author for the Journal helps build your resume, establish your reputation, and share your great concepts with fellow ISUG members.

For more information, check out the *ISUG Technical Journal* Online to see sample articles and guidelines for authors. Then contact Journal Director Eric Van Patten at eric@isug.com or Managing Editor Mary Freeman at freemancomm@yahoo.com to discuss your article concept.

www.isug.com