**Tracking Changes in Your Enterprise Database**

SQL Server 2008

Paul S. Randal

<http://technet.microsoft.com/en-us/magazine/cc987538(printer).aspx>

For developers, one difficult problem in SQL Server is tracking what data has changed in a database. An even greater challenge is architecting a simple solution that doesn’t heavily impact workload performance and isn't difficult to create, implement, and manage. So why go to all the trouble to track changes? Is tracking changes really worth all this effort? Two commonly cited examples are to support updates to a data warehouse and to support the synchronization of heterogeneous, occasionally connected systems.

A data warehouse usually has some representation of the tables in the Online Transaction Processing (OLTP) database, but the table schemas may actually be quite different. This means that there needs to be an ETL (extract, transform, load) process that moves data from the OLTP database to the data warehouse.

Video

Watch Paul Randal demonstrate how you can track changes in your database using the new Change Data Capture feature in SQL Server 2008.

I can think of three possibilities for doing this. The first is to refresh the entire data warehouse periodically. This is clearly impractical for large data volumes and also means that updates to the data warehouse are not continuous. The second method is to use a partitioning scheme in the OLTP database to allow the ETL process to work only on data that is new since the previous ETL process. This method only works for data inserts, not updates or deletes, and requires a complex mechanism to manage partition boundary definition and switching partitions. The third method is to track changes to the OLTP data and only perform the ETL process using the changed data. This is the most efficient method in terms of data volume.

Mobile devices are ubiquitous in today's business environment, which means that dealing with occasionally connected systems is a requirement. In terms of database systems, the problem is how to efficiently update a data store on a device that does not connect frequently, especially when the data store itself may be small and radically different in schema from the main database.

Consider a mobile sales rep who is responsible for a portion of a very large product catalog. Every night she connects her handheld device to the main database to download the latest data—all the changes to that portion of the product catalog, simplified for storage on a handheld device. The data transfer should be as efficient as possible.

Now, you could have the database system prepare the entire relevant portion of the product catalog for download to the device and have the device download it. In other words, all the data is downloaded every time the device connects, even if the data hasn't changed. This is obviously a rather inefficient approach.

Another approach is to have the database system track changes to the relevant portion of the product catalog. Then, when the handheld device connects, it asks for the data that has changed since the last time it connected. In this solution, the database system only has to prepare a subset of the data and the download is as efficient as possible.

Another reason for tracking changes is to support auditing, which is essential these days. Auditing tracks the changes being made as well as when the change occurred and who made the change. This really takes things to another level, with rigid constraints around durability, security, and correctness of a complete audit trail.

The technologies that were designed for tracking data changes in SQL Server 2008 were not intended to support auditing; however, SQL Server 2008 offers a new feature, which is called SQL Server Audit, that was designed specifically for auditing. Rick Byham discussed the SQL Server Audit feature in his "SQL Server 2008: Security" article from the April 2008 issue of *TechNet Magazine* (available at [technet.microsoft.com/magazine/cc434691](http://technet.microsoft.com/magazine/cc434691)).

As you can see, there are a number of compelling reasons to track changes to your data. So the important question is, how best can tracking be done?

**How You Track Changes in SQL Server 2005**

With SQL Server 2005 (and earlier versions of SQL Server) there is not a simple, canned solution. So for these platforms, developers have had to create custom solutions for their applications, usually involving timestamp columns, DML (Data Manipulation Language) triggers, and extra tables. These solutions, however, present a variety of potential problems. For instance:

* The addition of timestamp columns causes the table schema to change (with possible knock-on effects in stored procedures and other code).
* A DML trigger is implicitly part of the transaction containing the DML by which it is triggered, so its execution time will increase the length of the transaction. The more complex a trigger, the longer it will take to execute and so the higher the detrimental effect on workload performance. DML triggers used for tracking changes need to process the inserted and deleted tables to harvest all the changes and then insert them into another tracking table.
* The tracking table needs to be managed in some way to avoid its growing out of control, which may require you to create something like an Agent job to periodically trim old data.

**Easier Ways to Track Changes in SQL Server 2008**

SQL Server 2008 introduces two new technologies that make it much easier to track changes to data: change tracking and change data capture. Both features track data that has changed (as well as use the insert, update, or delete operations to track exactly how the data was changed), and they eliminate the need for custom solutions. Those similarities aside, their mechanisms and what exactly they track are actually quite different.

Change data capture uses an asynchronous mechanism that tracks all the changes that occur to a table (or a defined set of columns of the table), including the column values themselves. This is designed for scenarios like the data warehouse ETL process I described previously.

Figure 1 illustrates change data being consumed in time slices. The change data capture mechanism extracts the changed data into a set of tables, with the most recent changes being at the top of the table. The ETL process can then query the tables holding the change data for all changes that occurred within a set time period. This mechanism allows the ETL process to limit the amount of data that must be consumed in each batch.

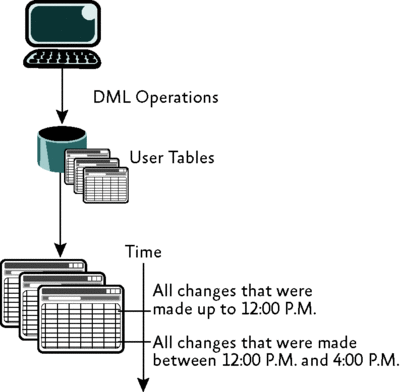


Figure 1 **Historical change data being consumed in time slices** (Click the image for a larger view)

Change tracking, on the other hand, uses a synchronous mechanism that tracks only that a particular row changed in a table (and optionally the list of columns that changed). This is designed to address such problems as the occasionally connected system scenario I described previously. This approach is illustrated in **Figure 2**.

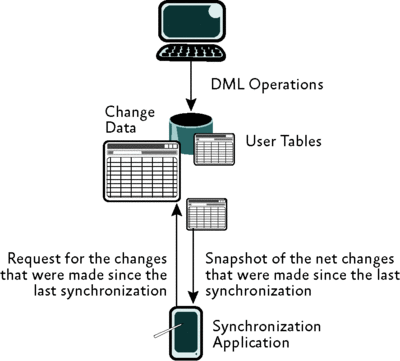


Figure 2 **An occasionally connected system using change tracking data** (Click the image for a larger view)

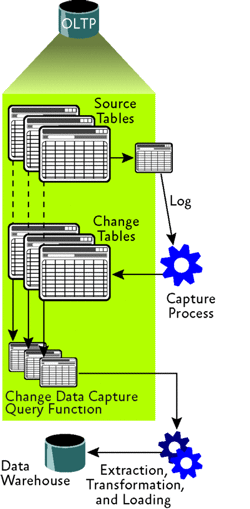
Both features introduce an increase in I/O and logging, but this is true with custom solutions as well—the change data has to be stored somewhere. What makes these two features potentially different from a custom solution is that the tables used to store the change data must be in the same database as the tables being tracked. This means all the change data will be included in backups and potentially transmitted over the network by log shipping or database mirroring.

In terms of development, these two features should remove a lot of the complexity from tracking changes. There are no table schema changes or triggers required for either technology. Both technologies have configurable, automatic cleanup processes, order changes by transaction commit times, and provide built-in functions to retrieve change information.

From the management perspective, there are pros and cons to each approach. As with any technology, there is a lot of information you must understand before you develop and deploy solutions that use these features. In the rest of this article I'll give an overview of each of these features, touching on how they work and the practical points to consider before using them in production.

**How Change Data Capture Works**

Change data capture doesn't do anything as part of the transactions that change the table being tracked. Instead, the insert, update, and delete operations are written to the transaction log, as normal, and periodically harvested from the log. Harvesting is performed by a SQL Agent log reader job, and the harvested operations are stored in a separate table called a change table. At some later point, the change table can be queried to obtain the change data using one of two functions. The combination of the change table and the two functions is called a capture instance. **Figure 3** shows the flow of data using change data capture to drive a data warehouse ETL process.



Enabling change data capture is a two-stage process. First a member of the sysadmin fixed server role must enable change data capture for the database using sys.sp\_cdc\_enable\_db. Then a member of the db\_owner fixed server role must enable change data capture on a specific table using sys.sp\_cdc\_enable\_table. These security requirements are due to the potential for high disk usage if change data capture is misconfigured. It makes perfect sense that a table owner cannot enable the feature and surprise a database administrator with extra disk usage.

When change data capture is enabled for a database, a few things are added to the database, including a new schema (called cdc), some metadata tables, and a trigger to capture Data Definition Language (DDL) events. (One capability that I think is cool is that you can get a list of the DDL changes to a table.)

Enabling change data capture also creates the capture instance for the table—the change table and up to two functions to return change data. The change table name is the same as the capture instance, with \_CT appended. The first function is always created and is the one used to return change data from the change table. The second function is created if the option to allow net changes is specified. This means that only the final result of all the captured changes is returned, rather than all the intermediate changes that the first function returns. The two function names are, respectively, fn\_cdc\_get\_all\_changes\_ and fn\_cdc\_get\_net\_changes\_, with the capture instance name appended. Note that (like the change tracking feature) this functionality requires the table to have a primary key or other unique index.

When you're dealing with the first table in the database to have change data capture enabled, two SQL Agent jobs may be created: the capture job and the cleanup job. I say "may be created" because the capture job is the same as the one used for harvesting transactions in transactional replication. If transactional replication is already configured, then only the cleanup job will be created and the existing log reader job will also be used as the capture job. This is good because having two log reader jobs would very quickly lead to contention problems with the log and hence performance problems. Either way, SQL Agent must be running if you want to use change data capture.

The logic inside the log reader automatically copes with tables being enabled and disabled for change data capture and alters what is harvested from the transaction log accordingly. One major point to note here is that once change data capture is enabled, the transaction log behaves just as it does with transactional replication—the log cannot be truncated until the log reader has processed it. This means a checkpoint operation, even in SIMPLE recovery mode, will not truncate the log unless it has already been processed by the log reader.

Also, if the BULK\_LOGGED recovery model is used to reduce logging, change data capture will force everything to become fully logged, except for index create/drop/rebuild operations. If you have never experienced such behavior, beware that this may cause transaction log size problems, especially if the capture job defaults are changed so the log isn't processed as frequently.

By default, the capture job runs continuously, scanning the log every five seconds and processing a maximum of 500 transactions from the log. Also by default, the cleanup job runs every day at 2 A.M. and removes all change data entries older than three days from the change tables. You can change these settings using the sys.sp\_cdc\_change\_job procedure, and then changes will not take effect until you restart the jobs using sys.sp\_cdc\_stop\_job and sys.sp\_cdc\_start\_job.

Although the log reader process will usually have a low impact on system performance, it is possible on heavily loaded OLTP systems with a lot of changing data that the addition of just one log reader process can cause contention on the transaction log. The actual contention would be caused by the disk heads having to move back and forth between the point at which the log is being written to by transactions and the point at which it is being read by the log reader process. In this case, it may be necessary to change the frequency at which the capture job runs to ensure OLTP performance will not suffer. This, however, creates a classic disk space versus performance trade-off—the log will continue to grow until the capture job processes it.

The same problem occurs if the cleanup job frequency or change data retention periods are changed—the change tables will continue to grow until the change data is cleaned. This leads to a wholesale design consideration of what gets tracked and how long it is retained. Important things to consider here include:

* The required column list for the capture instance. When more columns are captured, more change data is inserted into the change tables.
* The amount of disk space used by the change tables.
* The frequency at which the process runs that consumes the change data. Keep in mind that the data can't be deleted if it hasn't been used yet.
* The frequency that the cleanup process runs—there may be so much change data generated that the cleanup process that deletes it can only run on the weekend, for example, because otherwise it generates too much transaction log.

Change data capture can be set up to simply track all changes to a table or to track a subset of the columns in a table. Using a subset can be useful if some of the unimportant columns are very wide varchar columns or Binary Large Object (BLOB) columns (such as text, image, or XML); otherwise, the space used by the change table could grow unwieldy very quickly.

Given the increased potential for disk space usage, the filegroup location of the change table can be set when change data capture is enabled. This allows easier management of the underlying disk space and means that all the change data can potentially be stored on a volume with a less expensive RAID level than the main database. Also, while the cleanup job settings apply to all capture instances, an individual capture instance can be separately cleaned at any time if disk space becomes an issue. You can easily monitor disk space usage using sp\_spaceused on the capture tables.

The actual row written to the change table contains metadata about the transaction (the commit log sequence number, or LSN), as well as the order within the transaction that the change occurred, what the operation was, a bitmask of which columns were changed, and the actual column values.

DDL changes are unrestricted while change data capture is enabled. However, they may have some effect on the change data collected if columns are added or dropped. If a tracked column is dropped, all further entries in the capture instance will have NULL for that column. If a column is added, it will be ignored by the capture instance. In other words, the shape of the capture instance is set when it is created.

If column changes are required, it is possible to create another capture instance for a table (to a maximum of two capture instances per table) and allow consumers of the change data to migrate to the new table schema. But care should be taken when doing this because two capture instances for a tracked table mean twice the amount of disk space, I/Os, and logging.

Without going into too much depth, changes are retrieved from the change tables using the functions I've described. The functions take a start LSN and end LSN, and there are other functions supplied that allow you to convert a regular time into an LSN. When retrieving updates, you can even specify whether to see the before and after values, or just the before. There is a screencast of me using change data capture available at [www.technetmagazine.com/video](http://www.technetmagazine.com/video).

**How Change Tracking Works**

As I've mentioned, change tracking is a synchronous process and is a lot less complex than change data capture. It is part of the transaction that makes a change to a row in a table being tracked, and the fact that the row has changed is tracked in a separate table. The table is what's called an internal table, and there is no control over its name or where it is stored. I don't consider this an issue, as there should be a lot less data in this table than in a change table used for change data capture. But there can still be disk space problems, as I'll explain in a moment.

The fact that change tracking is done synchronously means that there is extra processing done inside each transaction that changes the table being tracked. The effect on performance is similar to that when a non-clustered index exists on the table and must be updated with each change to the table. Transactions themselves are also tracked when they commit by a row in the internal sys.syscommittab table.

Change tracking is enabled and disabled using regular ALTER DATABASE and ALTER TABLE syntax, and it follows the same model as change data capture, where it must be enabled at the database level before the table level. The sequence of operations would look something like this:

|  |
| --- |
|  |
| ALTER DATABASE AdventureWorks2000 SET CHANGE\_TRACKING = ON  (CHANGE\_RETENTION = 2 DAYS, AUTO\_CLEANUP = ON);  GO  USE AdventureWorks2000;  GO  ALTER TABLE Person.Person ENABLE CHANGE\_TRACKING  WITH (TRACK\_COLUMNS\_UPDATED = ON);  GO |

The required permissions for enabling change tracking at the database and table levels are also different from those for enabling change data capture: db\_owner and the table owner, respectively. When change tracking is enabled at the database level, the retention period can be set as well as whether change data is automatically cleaned up. The default retention period is 2 days, with a maximum of 90 days and a minimum of one minute.

Automatic cleanup is also turned on by default. When making changes to these settings, you need to evaluate the same trade-offs as I discussed with change data capture—basically disk space and performance against application needs.

By default, what is captured for each row is just that it changed. This is done by making a note of the primary key of the row that changed (which means that change tracking on a table requires that it has a primary key), along with a version number (once a database is enabled for change tracking, a version number is instituted, which allows ordering of operations), and the type of operation that made the change. You can also optionally track which columns changed; this requires 4 bytes per changed column.

Disk space monitoring is slightly different with change tracking, since the change data is stored in internal tables. To find the names of the internal tables being used, simply use the sys.internal\_tables system catalog view:

|  |
| --- |
|  |
| SELECT [name] FROM sys.internal\_tables  WHERE [internal\_type\_desc] = 'CHANGE\_TRACKING';  GO |

Then pass the name into sp\_spaceused to see how much disk space is being used.

Unlike with change data capture, when change tracking is enabled, there are restrictions on the DDL that can be performed on a table being tracked. The most notable restriction is that the primary key cannot be altered in any way. The other restriction worth calling out here is that an ALTER TABLE SWITCH will fail if either table involved has change tracking enabled. This is most likely due to the fact that it doesn't make sense to automatically start or remove change tracking for a partition being switched out of a change tracked, partitioned table, or a change tracked table being switched into a partitioned table, respectively.

Changes are retrieved from the internal change tables using a new CHANGE­TABLES (CHANGES …) function. This takes the name of the change tracked table plus the version number from the previous time it was used and returns information about all rows that have changed since that previous time. There are various functions to find the current and oldest valid version. The application can then use the information that is returned to query the table being change tracked to get the actual column values. This, of course, is a multi-step process—you get the current version, use that version to query change tracking, and then query the actual tables for the column data corresponding to that version.

On a constantly changing system, it is possible to get inconsistent or incorrect results unless some kind of unchanging view of the version, change data, and actual column data is maintained. To do this, you can use snapshot isolation and wrap the multi-step process in an explicit transaction. This works well but has potential drawbacks. Snapshot isolation can affect the workload performance, and it affects the performance and space usage of tempdb. You can find more details about this at [technet.microsoft.com/library/cc280358](http://technet.microsoft.com/library/cc280358).

**Wrapping Up**

**Figure 4** provides a side-by-side comparison of change tracking and change data so you can get a better idea of the major differences that DBAs will care about. You can see from the table that change data capture is much heavier weight than change tracking. It requires more care when deciding what to track because of the potential for fast growth in the size of the tracking table if, say, the table that is tracked contains BLOB columns or very wide rows. There is also the potential for transaction log management issues, as the log will not be truncated until the log reader has harvested records from the log.

Figure 4 Comparison between change tracking and change data capture

|  |  |  |
| --- | --- | --- |
| **Feature** | **Change Tracking** | **Change Data Capture** |
| Synchronous | Yes | No |
| Requires SQL Agent | No | Yes |
| Forces full logging of some bulk operations | No | Yes |
| Prevents log truncation | No | Yes, until log records harvested |
| Requires snapshot isolation | Recommended | No |
| Requires separate tables to store tracking data | Yes | Yes |
| Requires primary key | Yes | Not by default |
| Allows placement of tracking tables | No | Yes |
| Potential for space consumption issues | Some | Lots |
| Automatic cleanup process | Yes | Yes |
| Restrictions on DDL | Yes | No |
| Permission required to enable | Sysadmin | Database owner |
|  |  |  |

Change tracking is not without its own requirements, however. For instance, a primary key is required, and it is strongly recommended that you use snapshot isolation when change tracking is enabled. Snapshot isolation itself can add significant workload overhead and requires much more careful management of tempdb.

There is one further issue that developers and DBAs need to cope with: disaster recovery. Although discussing this in depth is beyond the scope of this article, the topic of disaster recovery is too important to not at least mention here.

Both features play well with BACKUP and RESTORE. The problem comes when a database gets restored and is essentially taken back in time. How should the overall application/system behave? Custom solutions designed for tracking changes also face this problem, and it still needs to be considered when using SQL Server 2008.

As always, make sure you read through all the available documentation ([technet.microsoft.com/library/bb418491](http://technet.microsoft.com/library/bb418491)) and any existing white papers before embarking on a design and deployment project that involves the new features to track changes. You need to first find out if any potential issues I haven't covered here may apply to you. You should also get details on the new monitoring SPs and Dynamic Management Views (DMVs).

Overall these new features are a huge advance over previous methods for tracking data changes. Now that they exist, you can be sure developers will want to use them in solutions that you manage.

There are important configuration and management issues to consider, and I hope this article has given you a solid overview of the technologies so you can anticipate and prepare for some of the issues I have discussed. If you have any feedback on this article or questions, go ahead and drop me a line at [Paul@SQLskills.com](mailto:Paul@SQLskills.com).

**Paul S. Randal** is the Managing Director of [SQLskills.com](http://SQLskills.com) and a SQL Server MVP. He worked on the SQL Server Storage Engine team at Microsoft from 1999 to 2007. Paul wrote DBCC CHECKDB/repair for SQL Server 2005 and was responsible for the Core Storage Engine during SQL Server 2008 development. Paul is an expert on disaster recovery, high availability, and database maintenance and is a regular presenter at conferences around the world. He blogs at [SQLskills.com/blogs/pau](http://SQLskills.com/blogs/paul)