[Creating a SQL Server audit using SQL Server Change Tracking (sqlshack.com)](https://www.sqlshack.com/creating-a-sql-server-audit-using-sql-server-change-tracking/#Enabling%20SQL%20Change%20Tracking)

Creating a SQL Server audit using SQL Server Change Tracking

March 27, 2019 by [Ahmad Yaseen](https://www.sqlshack.com/author/ahmad-yaseen/)

[](https://www.quest.com/solutions/sql-server/?utm_source=sql_shack&utm_medium=Direct-Outbound%20Marketing&utm_campaign=FY2024_Q2_AMER_SQLServer_SQLShack_ADV&utm_term=&utm_content=banner)

This continuation of our SQL Server auditing series is on SQL Server Change Tracking and will cover an overview, enabling, disabling, auditing DML changes including SQL updates and deletes.

If this is your first reading in the SQL Server Audit series, it is recommended to go through the previous articles of this series (see the [TOC](https://www.sqlshack.com/creating-a-sql-server-audit-using-sql-server-change-tracking/#SQLAuditToC) at the bottom), to build a solid background about the concept of the SQL Server Audit, the different reasons behind auditing the SQL Server instances and databases, and the different methods that can be used to audit the SQL Server databases. In this article, we will discuss how to perform a SQL Server audit using SQL Server Change Tracking.

Overview

SQL Server Change Tracking, also known as CT, is a lightweight tracking mechanism, introduced the first time in SQL Server 2008, that can be used to track the DML changes performed in SQL Server database tables. SQL Change Tracking can be configured in all SQL Server editions, including the free Express edition.

SQL Server Change Tracking is a synchronous tracking mechanism, in which the changes information will be available directly once the DML change is committed, without the need for a delay while reading the changes from the Transaction Log file, such as the Change Data Capture asynchronous mechanism. This means that, SQL Change Tracking does not require the SQL Server Agent service to be up and running as it has no dependencies on the SQL Agent jobs in capturing or writing the DML changes.

When SQL Server Change Tracking is enabled on a database table, the SQL Server Engine will create an internal table, under the name **Change\_Tracking\_<Object\_ID>**, to track the INSERT, UPDATE and DELETE statements on the tracked user table. What makes SQL Change Tracking lighter than the [Change Data Capture](https://www.sqlshack.com/change-data-capture-for-auditing-sql-server/) feature, is that, it will track the database table changes by mentioning that, there is a DML change performed on this row within the tracked table, providing the **Primary Key** column value of the modified row, the changed column and the modification type, without writing detailed information about the changed data, such as writing the inserted or deleted values or the values before and after update process, with the minimum storage requirements and overhead. This is the reason why the database table should have a Primary Key on it, in order to enable SQL Change Tracking on that table, as this key value will be used to identify the modified rows in the tracked tables.

SQL Server Change Tracking as an Audit Solution

SQL Server Change Tracking is considered a legacy SQL Server Audit solution, that can be used to track and audit the database table DML changes by answering simple auditing questions such as, which row is changed, by providing the Primary Key of that row, and what type of change performed on that row.

What makes SQL Change Tracking less desirable as a SQL Server database audit solution is that it records no information about the inserted data, the deleted data or the data before and after the update process. In addition, the tables that have no Primary Key constraints defined on it cannot be audited using SQL Change Tracking, as it is limited only for the database tables with Primary Key constraints.

On the other hand, SQL Server Change Tracking will record no history about the changes performed on a database table, where it will record the last change performed on that row, without retaining the version history. For example, if a row is inserted, then updated multiple times and finally deleted, SQL Change Tracking, as a SQL Server Audit solution, will only record the last delete statement, without considering the previous operations performed on that row.

To build a useful SQL Server Audit solution using SQL Server Chang Tracking, extra coding effort will be required to join the internal tables of SQL Change Tracking with the tracked source table, based on the Primary Key value of the changed row, that is stored in an internal table, to obtain complete information about the changed data. And due to the fact that the internal tables are not visible and cannot be queried directly, you can take benefits from the SQL Change Tracking functions, that uses the on-disk tables, built based on these internal tables, which we will discuss later in this article.

Another point to consider here is that the SQL Change Tracking internal tables will grow gradually with time. Although the purging process is controlled by the auto cleanup thread, that is responsible for purging old data from the internal on-disk tables based on a predefined retention period, with a default value of 2 days, you still need to set proper retention period to keep the changes data available for your SQL Server Audit solution. In addition, you can benefit from the [new stored procedure](https://support.microsoft.com/en-us/help/3173157/adds-a-stored-procedure-for-the-manual-cleanup-of-the-change-tracking) added in SQL Server 2016 to perform manual cleanup for the internal SQL Server Change Tracking table.

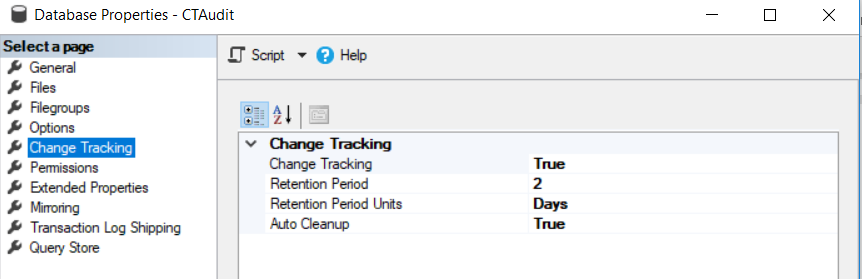
Let us see how we can use the SQL Change Tracking to audit the SQL Server DML changes.

Enabling SQL Change Tracking

In order to enable SQL Server Change Tracking on a database table for auditing purposes, you should enable it at the database level using the ALTER DATABASE T-SQL statement, by providing the retention period for the internal on-disk tables, and if you will enable the auto clean process, that will delete the internal on-disk table’s data older than the retention period automatically, as below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | USE master  GO  ALTER DATABASE [CTAudit]  SET CHANGE\_TRACKING = ON  (CHANGE\_RETENTION = 2 DAYS, AUTO\_CLEANUP = ON) |

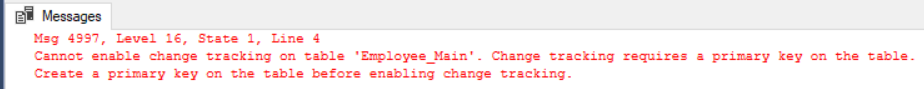
You can also enable it using the SQL Server Management Studio tool, from the **Change Tracking** tab of the Database Properties window, in which you can specify the retention period of the internal on-disk tables and enable the auto clean process, as shown below:



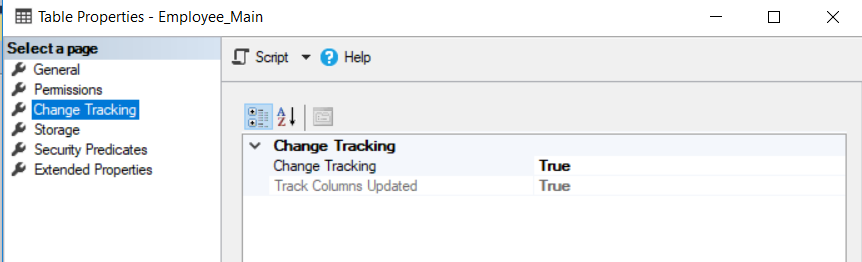
After enabling SQL Server Change Tracking at the database level, we need to enable it on each table that will track and audit the DML changes on it. This can be achieved using the ALTER TABLE T-SQL statement below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | USE CTAudit  GO  ALTER TABLE Employee\_Main  ENABLE CHANGE\_TRACKING  WITH (TRACK\_COLUMNS\_UPDATED = ON) |

If you try to enable SQL Change Tracking in a database table with no Primary Key defined on it, the ALTER TABLE statement will fail, showing that SQL Change Tracking requires creating a Primary Key on the table before enabling it, as in the error message below:

[](https://www.sqlshack.com/wp-content/uploads/2019/03/sql-server-audit-error-when-enabling-ct-on-table.png)

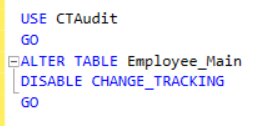
After adding a Primary Key constraint on the table, the ALTER TABLE statement will be executed successfully. You can also enable SQL Change Tracking using the SQL Server Management Studio, from the SQL Change Tracking tab of the Table Properties window, as shown below:



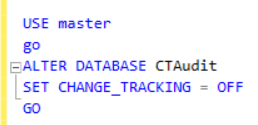
Disabling SQL Change Tracking

Enabling SQL Server Change Tracking on a database table will not prevent you from performing DDL changes at that table, except for the changes on the Primary Key that will fail unless the CT is disabled on that table.

Change Tracking can be disabled at the table level using the ALTER TABLE T-SQL statement below:



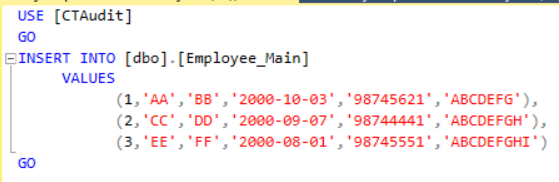
After disabling it at the table level, Change Tracking can be easily disabled at the database level, using the ALTER DATABASE T-SQL statement below:



Auditing DML Changes

INSERT Audit

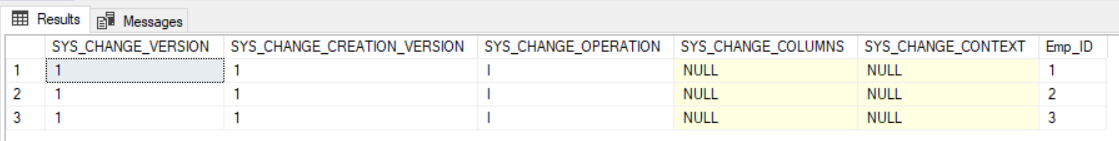
When SQL Server Change tracking is enabled on the database table for SQL Server database audit purposes, all DML changes that are performed on the table rows will be written to the CT internal tables. Assume that we perform the below INSERT statement into the Employe\_Main test table:



To get the Change Tracking data recorded in the internal table after the INSERT statement, you can use the [Change Tracking functions](https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/work-with-change-tracking-sql-server?view=sql-server-2017#about-the-change-tracking-functions), such as **CHANGETABLE** system function. The CHANGETABLE function returns all changes performed on the tracked table after the specified version number. The version number counter is associated with each changed row, that will be increased whenever there is a change on the tracked table is performed. The below T-SQL script can be used to retrieve the change information:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM CHANGETABLE  (CHANGES [Employee\_Main],0) as CT ORDER BY SYS\_CHANGE\_VERSION |

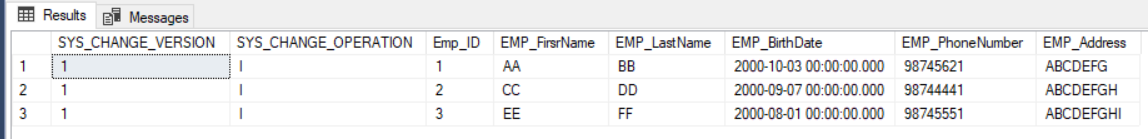
The data returned from the CT internal tables, after performing the INSERT statement will show, the version of the performed DML change, the type of the DML operation, which is I for INSERT in this case, the changed columns, which is NULL in the case of INSERT and finally the Primary Key value for the inserted rows in the SQL Server audited table, as shown below:



To get the complete inserted record, we can easily join the CHANGETABLE function with the tracked source table, based on the Primary Key value, as in the T-SQL script below:

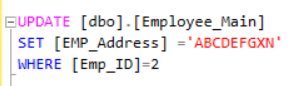
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT CT.SYS\_CHANGE\_VERSION,    CT.SYS\_CHANGE\_OPERATION, EM.\*    FROM CHANGETABLE  (CHANGES [Employee\_Main],0) as CT  JOIN [dbo].[Employee\_Main] EM  ON CT.Emp\_ID = EM.Emp\_ID  ORDER BY SYS\_CHANGE\_VERSION |

The returned result from joining the CT function and the tracked source table will show complete information about the inserted data, that can be useful to audit the database table, as below:

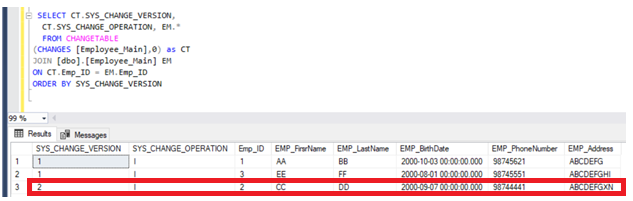


UPDATE Audit

If the below UPDATE statement is performed on the same tracked table:



Then execute the previous query that joins the CHANGETABLE function with the tracked source table, you will see that the change version number for the row with Emp\_ID value equal to 2 will be increased. It is clear also that only the last version of the change that is performed on that row will be recorded, missing the previous INSERT statement and keeping the last update statement, which will internally delete the previous record then insert the record with the new value, as shown below:

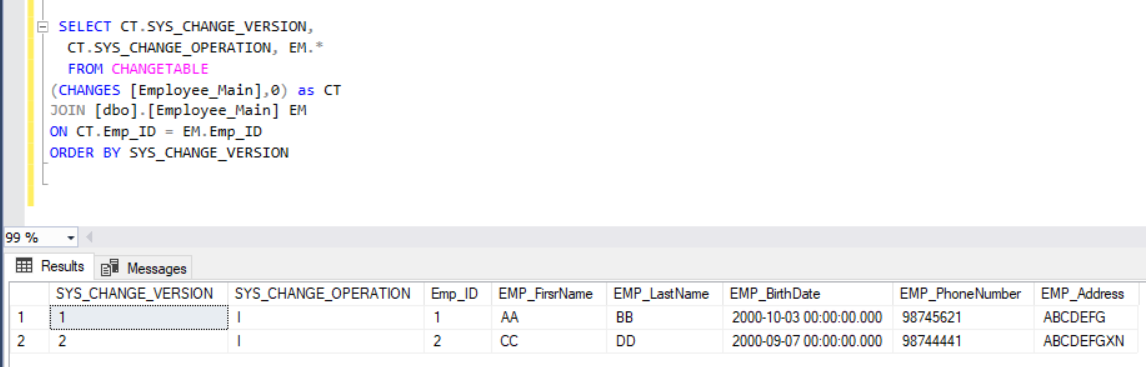


DELETE Audit

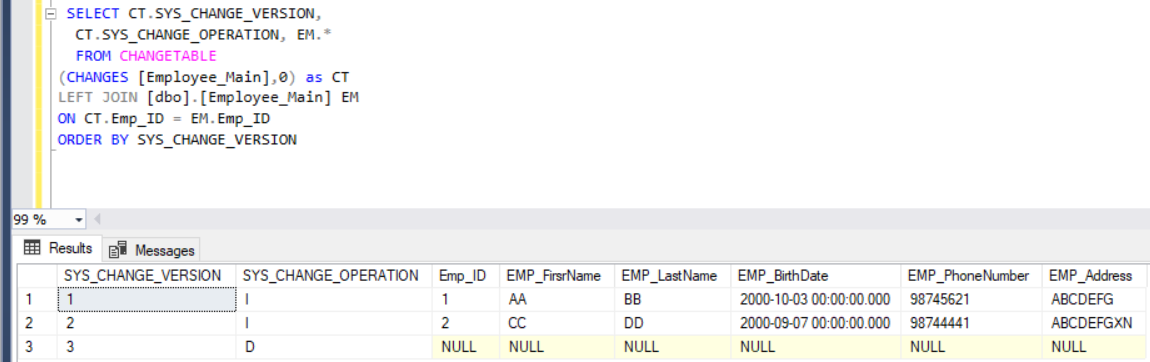
Assume that we execute the below DELETE statement to drop the third row from the tracked table:

Delete the third record

Then execute the same query that joins the CHANGETABLE function with the tracked source table to check the deleted record information. You will see that the deleted record data will not be shown, as it is not existing in the tracked source table, as shown below:



Changing the JOIN type in the previous query to LEFT OUTER JOIN, you will see that the deleted record information from the CHANGETABLE function will be retrieved. This information includes only the change version number and the change type, which is D for DELETE, with no information about the deleted record, as shown below:



It is clear from the previous results that, SQL Server Change Tracking can be used as a limited SQL Server database audit solution, to track the DML changes on the CT enabled table. This is due to the fact that it will return only the last change that is performed on the modified record with no historical information about the value before the update or delete operations.

In the next article of this series, we will discuss how to audit the SQL Server using the SQL Server Audit feature. Stay tuned.

Table of contents

|  |
| --- |
| [SQL Server Audit Overview](https://www.sqlshack.com/sql-server-audit-overview/) |
| [Implementing a manual SQL Server Audit](https://www.sqlshack.com/implementing-a-manual-sql-server-audit/) |
| [Creating a SQL Server audit using SQL Server Extended Events and Triggers](https://www.sqlshack.com/creating-a-sql-server-audit-using-sql-server-extended-events-and-triggers/) |
| [Auditing by Reading the SQL Server Transaction Log](https://www.sqlshack.com/auditing-by-reading-the-sql-server-transaction-log/) |
| [Change Data Capture for auditing SQL Server](https://www.sqlshack.com/change-data-capture-for-auditing-sql-server/) |
| Creating a SQL Server audit using SQL Server Change Tracking |
| [SQL Server Audit Feature Components](https://www.sqlshack.com/sql-server-audit-feature-components/) |
| [Using the SQL Server Audit Feature to Audit Different Actions](https://www.sqlshack.com/using-the-sql-server-audit-feature-to-audit-different-actions/) |
| [Performing a SQL Server Audit using System-Versioned Temporal Tables](https://www.sqlshack.com/performing-a-sql-server-audit-using-system-versioned-temporal-tables/) |
| [Perform a SQL Server Audit using ApexSQL Audit](https://www.sqlshack.com/perform-a-sql-server-audit-using-apexsql-audit/) |
| [SQL Server Auditing Best Practices](https://www.sqlshack.com/sql-server-auditing-best-practices/) |