A | Using Change Tables

You can use Attunity Replicate tasks to save the change events in change tables. This section describes how to use Attunity Replicate with change tables.

In this appendix:

Working with Change Tables

In addition to replicating changes from source endpoint tables to corresponding tables in a target endpoint, Attunity Replicate can also replicate changes to corresponding Change Tables in the target endpoint. This process occurs simultaneously when applying changes to the target tables. Attunity Replicate lets you determine whether to replicate the changes to the target only, store changes in the Change Tables, or both. See Using the Change Table Model below for more information.

Note UPDATEs applied to the source *that do not* change the source data will be applied to the target but will not be applied to the corresponding Change Table. For example, if an UPDATE operation on Column A in the source changes all values greater than 10 to 1, and one of the records in Column A is *already* 1, then the UPDATE for that record will not be written to the Change Table.

The Change Tables have the same names as the tables that are being replicated, however a suffix is added to each table name. By default the suffix is `__ct`, however you can change the suffix name in the Attunity Replicate Console. For more information on changing the suffix added to the Change Table names, see Store Changes Settings.

In addition to the selected columns from the source table, the Change Table also includes special header columns that provide more information on the change the row represents such as the operation, the transaction and the timestamp. This lets you use SQL Query Language to carry out various analysis of the change events, such as fraud detection, trend analysis, triggering of business processes, and Disaster Recovery. For more information about reading the Change Tables, see Reading the Change Tables.

Handling Truncate Operations

TRUNCATE operations will *not* truncate the Change Table. Instead, an additional record will be added to the table with `operation=TRUNCATE`. This also means that, when replicating to a Hadoop target, the HDFS files corresponding to the Change Table will *not* be deleted.

Regarding the actual target table, if both the Apply Changes *and* Store Changes replication options are enabled, the target table will be truncated. This also means that, when replicating to a Hadoop target, the HDFS files corresponding to the Change Table will also be deleted.

To apply TRUNCATE operations to both the Change Table and the Target Table (for sources that support TRUNCATE):

1. In the task settings' Store Changes Settings tab, make sure that **Apply to Change Table** (the default) is selected from the **DDL options** drop-down list.
2. In the task settings' Apply Changes Settings tab, make sure that **TRUNCATE target table** (the default) is selected from the **When source table is truncated** drop-down list.

Using the Change Table Model

When you work with Change Tables you can determine whether to store the changes in the Change Tables, apply the changes to the target tables, or both store and apply the changes. You determine this when you define the replication task. For more information on this setting, see Store Changes Settings.

In cases where you are both applying and storing the changes, the following is true:

- The target and Change Tables must be in the same endpoint, although they can have different schemas. For example, the Change Tables will contain the metadata headers.

  For further details about changing the schema, see Rename Change Table Schema .

- Changes applied to the Change Table will be handled exactly the same as the changes performed in the corresponding transaction in the source database. Therefore, when using **Transactional apply mode** or **Batch optimized apply** mode with the **Preserve transaction consistency** option selected, the changes will be processed as a single transaction.

  The exception to this is when an error is encountered and Replicate switches to "one-by-one" apply mode in order to determine which of the Change operations is responsible for the error.

- The same data columns are both applied and stored with the exception of the change header columns, which are only added to the stored Change Tables.

Reading the Change Tables

You can use the tools for your target endpoint to get information using the metadata in the change tables. This data is defined by the header columns added to the change table schema. For information on these headers, see Change Tables.

Change Tables

For every target table in the replication task, a change table with the corresponding name is maintained by Attunity Replicate in the endpoint with the target tables. For more information, see Working with Change Tables. A change table contains the original table columns, and header columns. The header columns contain a prefix so that the name does not conflict with the source table column names. The default prefix is `header__`. For information on how to change this prefix, see the **Change tables** listing under Metadata in Task Settings. The following table lists the default change table header columns.

Table A.1 | Change Table Headers

Column Name	Type	Description
[header_ _]change_seq	varchar (35)	A monotonically increasing change sequencer that is common to all change tables of a task. The change sequence has the following format: YYYYMMDDHHmmSShhxxxxxxxxxxxxxxxxxx Where: » YYYY is the four-digit year (such as 2012) » MM is the two-digit month (range from 01-12) » HH is the hour in the day (range from 00-23) » mm is the minute in the hour (range from 00-59) » SS is the second in the minute (range from 00-59) » hh is the hundredth of the second (range from 00-99) » xxxxxxxxxxxxxxxxxx is a 19-digit, zero prefixed change number (global per task). The time part usually refers to the commit time of the transaction that includes the change record. Attunity Replicate contains logic that maintains the monotonicity of the sequence number so modifying or adjusting the endpoint time may result in multiple changes to seem that they are within the same timestamp but with increasing change number. The xxx...xxx is usually the internal change number from the data record except that for BEFORE-IMAGE records it is the same as the change number of the matching UPDATE record (for example, if the change number of BEFORE-IMAGE is 1000 and that of the UPDATE is 1001, then both have 1001). This allows a simple left-outer-join between the table and itself where on the left we scan until the point in time but filter out `operation=before-image`, and on the right we join on the same `change_seq` with the `change_oper` being 'B' .

Table A.1 | Change Table Headers

Column Name	Type	Description
[header_ _]change_ oper	varchar (1)	The operation type. This can be one of the following: » I: INSERT » D: DELETE » U: UPDATE » B: Before Image

Table A.1 | Change Table Headers

Column Name	Type	Description
[header__] change_mask	varbinary (128)	The change mask indicates which data columns in the change table are associated with columns that changed in the source table. The bit position in the change mask is based on the column ordinal in the change table. This means that if there are 5 header columns, they occupy bits 0 to 4 and the first data column is bit 5 in the change mask. The change mask is a binary column (a byte array) representing the change mask in little-endian order: **Byte 0** bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0 **Byte 1** bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8 In this example, bit#N indicates that the change table column of ordinal N relates to a column that changed in the source table. If update mask is 11000 and the column ordinal is 3 the column did not change. The following describes the bit semantics: » For INSERT records, all the inserted columns have the associated bits set. » For DELETE records only primary-key (or unique index) columns have the associated bits set. This allows an applier to construct a DELETE statement without having to find the primary key fields from another source. » For BEFORE-IMAGE records, all bits are clear (the change mask can be empty). » For UPDATE records, each column whose value changed between the BEFORE-IMAGE and the UPDATE will have the associated bit set. For space and processing efficiency, the actual number of bytes stored in the change-mask can be null-trimmed. This means that trailing zeros do not need to be stored. The handling logic should take this into consideration.

Table A.1 | Change Table Headers

Column Name	Type	Description
[header__] stream_ position	varchar (128)	The source CDC stream position.
[header__] operation	varchar (12)	The operation associated with the change record. It can be one of the following: » INSERT » UPDATE » DELETE » BEFOREIMAGE
[header__] transaction_ id	varchar (32)	The ID of the transaction that the change record belongs to. The value is a hex-string of the 128-bit transaction ID.
[header__] timestamp	timestamp	The original change UTC timestamp (the value may be approximate). **Note:** With PostgreSQL source, the timestamp is only known after the commit occurs. Therefore, until the changes are committed to the source tables, the default date will be displayed (e.g. 1970-01-01).
[header__] partition_ name	string	The name of the partition created on the target when Change Data Partitioning is enabled. The partition name consists of the partition start and end time. **Example:** 20170313T123000_20170313T170000

Use Example

The following SQL statement returns a range of changes that starts after the last handled event (of change sequence "20120723144522110000000000000901203") and until the event committed on 23-Jul-2012 at 23:00:00.00. For update we also get the before image value (here for the salary before and after).

```
SELECT CHANGE.[header__change_seq]
```

```
   ,CHANGE.[header__stream_position]
   ,CHANGE.[header__operation]
   ,CHANGE.[header__transaction_id]
   ,CHANGE.[header__timestamp]
   ,CHANGE.[EMPLOYEE_ID]
   ,CHANGE.[FIRST_NAME]
   ,CHANGE.[LAST_NAME]
   ,CHANGE.[SALARY]
   ,BI.[SALARY]
FROM [Replication].[HR].[EMPLOYEES_ct] CHANGE LEFT OUTER JOIN
   [Replication].[HR].[EMPLOYEES_ct] BI  ON
      BI.[header__change_seq] = CHANGE.[header$__change_seq] AND
      BI.[header__change_oper] = 'B'
WHERE CHANGE.header__oper <> 'B' AND
      CHANGE.[header__stream_position] >
      '20120723144522110000000000000901203' AND
      CHANGE.[header__stream_position] <= '20120723230000000Z' AND
ORDER BY
      CHANGE.[header__stream_position], CHANGE.[header$__stream_oper]
```