

Design Tip #63 Building a Change Data Capture System

By Ralph Kimball

The ETL data flow begins with transferring the latest source data into the data warehouse. In almost every data warehouse, we must transfer only the relevant changes to the source data since the last transfer. Completely refreshing our target fact and dimension tables is usually undesirable.

Isolating the latest source data is called “change data capture” and is often abbreviated CDC in high level architecture diagrams. The idea behind change data capture seems simple enough: just transfer the data that has been changed since the last load. But building a good change data capture system is not as easy as it looks.

Here are the goals I have for capturing changed data:

- Isolate the changed source data to allow selective processing rather than complete refresh
- Capture all changes (deletions, edits and insertions) made to the source data including changes made through non-standard interfaces
- Tag changed data with reason codes to distinguish error corrections from true updates
- Support compliance tracking with additional metadata
- Perform the change data capture step as early as possible, preferably before bulk data transfer to data warehouse

The first step in change data capture is detecting the changes! There are four main ways to detect changes:

- 1) **Audit columns.** In most cases, the source system contains audit columns. Audit columns are appended to the end of each table to store the date and time a record was added or modified. Audit columns are usually populated via database triggers that are fired off automatically as records are inserted or updated. Sometimes, for performance reasons, the columns are populated by the front end application instead of database triggers. When these fields are loaded by any means other than database triggers, you must pay special attention to their integrity. You must analyze and test each of the columns to ensure that is a reliable source to indicate changed data. If you find any NULL values, you must find an alternative approach for detecting change. The most common environment situation that prevents the ETL process from using audit columns is when the fields are populated by the front-end application and the DBA team allows “back-end” scripts to modify data. If this is the situation in your environment, you face a high risk that you will eventually miss changed data during your incremental loads.
- 2) **Database log scraping.** Log scraping effectively takes a snapshot of the database redo log at a scheduled point in time (usually midnight) and scours it for transactions that affect the tables you care about for your ETL load. Sniffing involves a “polling” of the redo log, capturing transactions on-the-fly. Scraping the log for transactions is probably the messiest of all techniques. It’s not rare for transaction logs to “blow-out,” meaning they get full and prevent new transactions from occurring. When this

happens in a production transaction environment, the knee-jerk reaction for the DBA responsible is to empty the contents of the log so the business operations can resume, but when a log is emptied, all transactions within them are lost. If you've exhausted all other techniques and find log scraping is your last resort for finding new or changed records, persuade the DBA to create a special log to meet your specific needs.

- 3) **Timed extracts.** With a timed extract you typically select all of the rows where the date in the Create or Modified date fields equal SYSDATE-1, meaning you've got all of yesterday's records. Sounds perfect, right? Wrong. Loading records based purely on time is a common mistake made by most beginning ETL developers. This process is horribly unreliable. Time-based data selection loads duplicate rows when it is restarted from mid-process failures. This means that manual intervention and data cleanup is required if the process fails for any reason. Meanwhile, if the nightly load process fails to run and misses a day, a risk exists that the missed data will never make it into the data warehouse.
- 4) **Full database "diff compare."** A full diff compare keeps a full snapshot of yesterday's database, and compares it, record by record against today's database to find what changed. The good news is that this technique is fully general: you are guaranteed to find every change. The obvious bad news is that in many cases this technique is very resource intensive. If you must do a full diff compare, then try to do the compare on the source machine so that you don't have to transfer the whole database into the ETL environment. Also, investigate using CRC (cyclic redundancy checksum) algorithms to quickly tell if a complex record has changed.

This design tip offers only a teaspoon sip of the issues surrounding change data capture. To dig deeper, here are two suggestions. Read the new *Data Warehouse ETL Toolkit* book I wrote with Joe Caserta for more detail on each of the above alternatives. Second, come to my class for ETL system designers. For change data capture, the students and I will build an aggregated set of requirements from around the room as if we were all part of one IT shop. Should be fun! The next ETL class is in San Jose in only three weeks. Go to our web site for details.