# IIUG Tech Talk

# Migrating your database under the worst conditions with Fail-back!

Art Kagel
President & Principal Consultant
ASK Database Management Corp.

ASK Database Management

# Challenge

- Migrate a large database to the cloud over slow and unreliable WAN connections.
- Maintain fail-back strategy:
  - Keep the target database in sync with the source until turnover.
  - After turnover keep the source database in sync with the target in case of unexpected need to revert!

# Options

- Enterprise Replication
  - Won’t work if the applications do not support Primary Keys.
  - Will not work if tables have no primary key and cannot take the downtime to add them.
- HDR or RSS to the Cloud server?
  - No good if the WAN connection is too slow.

# Alternatives

- Any form of export/import
  - Lots of lost time/updates while export files are copied to the target and loaded.
- Direct database to database table copies
  - Same problem

# Alternatives

- Archive and restore?
  - Ontape
  - Onbar
  - Ifxclone

All of the above alternatives are fine for the initial copy (assuming identical versions and platform). Still you need a way to keep the source and target in sync until the source can safely be taken offline after it is clear that there will not be a need to fail back to the source server!

# Deliberation

As noted, any method of initially migrating some state of the data to the target is acceptable. But how to maintain sync?

I fell back on a method I developed for a client back in 2013 to migrate from v7.31 to v11.50 since ER would not work between such an old release and v11.50!

# Method

I had developed a set of scripts that creates an audit table for each table in the database and adds a set of ON EVERY ROW triggers on the parent table – an insert trigger, an update trigger, and a delete trigger – each calling a generated stored procedure that copied the appropriate data from the transaction into the audit table.

# Method

- INSERT – save the entire new row
- DELETE – save the entire deleted row
- UPDATE – save both the pre-update version and the new version of the row.

The audit tables contained all of the columns of the parent table but also had a SERIAL type column and an operation character (i,d,b,a) as the primary key and a timestamp that could be used to restart a sync operation at a given point in time. As inferred above, an update operation wrote two rows to the audit table.

Once the audit table and triggers were installed on the source table, which only required a brief lock on each table, data from the audit tables, known as “deltas” could be exported periodically, and transported to the Cloud copy of the server.

Another set of scripts read through the deltas and generated SQL to reapply the changes to the target database table on the Cloud server.

# What about failing back?

All that had to be done, was when the applications were finally taken offline to redirect them to the Cloud server, drop the triggers on the original on-prem server, add them to the Cloud server, and reverse the processing of the delta records to maintain the on-prem server in sync until we either had to fail back to on-prem or determined that the original on-prem server was no longer needed as a backup.

# Recently had an opportunity to use this methodology again.

A new client had a similar need, so, I dragged out the audit scripts and began to customize them to the specific needs of this client.

ASK Database Management

# Not everything was rosey!

I had built the original scripts to read the output from myschema to generate the DDL to create the audit tables. The new client does not have my utilities on their servers and time constraints did not permit us the luxury of waiting for security concerns to approve installing them.

Recode the scripts to use dbschema?

Problem: dbschema's output formatting is inconsistent. The scripts were a mess.

# What to do?

I had a “WTF” moment when I realized that I had done it the hard way back in 2013 in the first place! Why not just read the %$^@# database catalog tables to generate the DDL for the audit tables?

Although, for that 2013 client, sourced in v7.31, I probably could have written the whole thing in AWK & ksh, today in v14.10 I could not! There were no extended types in v7.31. However, there are several “built-in” commonly used types in v14.10 that are actually extended types. Too complex for shell scripting!

So, I had to write a stored procedure to generate the list of columns with their types and feed that into the scripts that generate the DDL for the audit tables.

# Issues that came up

- Coping and auditing tables with encrypted data in columns
- Compare source and target tables and data
- Cleanup junk data prior to replicating
- Cleanup audit records to prevent the audit tables from getting too large using small intervals to avoid long transaction. That restricts how long between processing and cleanup sessions.
- Set same serial number and timestamp on before/after pair for update statement audit records
- Update large tables on known unique columns only (the original processing code updated by matching all columns from the pre-image record.
- Don’t cleanup audit table on error

# Issues that came up

Several issues affected processing the unloaded delta data from the audit tables:

- Embedded delimiters in the data
- Embedded quote characters in the data
- Embedded garbage characters in the data
- Embedded web and word processing characters in the data

The solution for this client was to process the data in code space directly from the audit tables instead of unloading it and processing with scripting.

# Quick Demo

ASK Database Management

# Questions?

Art Kagel
ASK Database Management Corp.
www.askdbmgt.com

ASK Database Management