MSDA101 – Missing Data

1. Introduction

Missing events are one of the more serious problems that a user can encounter, and usually require a complex process of analysis, in order to find its root cause. As opposed to other Replicate problems (like task crash or latency), missing events are typically found a long time after they actually occured. This is for two reasons, one there is no direct indication for missing events in the Replicate Console or log files. The second, is that missing events are usually found by the customer following his usage or analysis of the data in the target database, a process that takes time.

2. Missing Events

2.1 Missing Events

- A “missing” event is usually one or more rows that are found on source table but not on target table (usually caused by a missing insert)
- You can also miss updates or deletes – a missed update can cause the target row to not have the current data like source
- A missing delete can cause a duplicate or redundant row on target
- One of the ways to know you have a missing event is getting a “0 rows affected error” on an update
- But you might not know if you have a missing events if you will not get an error (e.g. an insert was missed but no update came after it to give “0 rows affected” error) or will not show if customer will not compare source and target (missing update)

3. Zero - 0 Rows affected

3. (Zero) 0 rows affected

These error messages can happen on UPDATE, DELETE and occasionally on INSERT operations.

3.1 0 rows affected during UPDATEs

There is a high probability that they indicate missing events, but this is not always applicable.

Different case scenarios

Missing events. Example:

- A WHERE clause in an UPDATE.

They can also indicate duplicates. Example:

- Starting from a specific timestamp could do the UPDATE twice.

- When applying a PK change more than once.

Or when it is not applicable. Example:

- MySQL can output this message if an UPDATE was applied twice.

3.2 0 rows affected during DELETEs

These messages are often caused by a missing record on target but can also happen on some cases.

They can indicate duplicate events. For instance, when applying the same DELETE more than once.

***They can happen in cached events in Bulk processing mode or when using UPSERT or MERGE options, in which case the logs would not print any errors (]E:) but instead they will appear only in debug.

Note: Within the Apply Conflicts Handling Policy the default setting for No record found for applying DELETE is to ignore the record.

3.3 Example with Redshift as a target endpoint

When you see the following error:

[...]E: 0 rows affected [122510] Zero rows affected (redshift_bulk.c:418)

It means CDC failed to UPDATE a record on the target, and this will cause the target to switch to one by one mode which will slow the task significantly.

In this case we will need to check the following:

1. Check the "attrep_apply_exceptions" table on the target, it should show the exact record on the target which is missing – please make sure that this record is indeed missing.

Please send us the statement from the "attrep_apply_exceptions".

2. Assuming the record is missing on the target, please check if it's missing on the source, please send us the result of the select command showing that record.

We'd like to compare the source and target statements, maybe we'll see something unusual at this stage.

3. Assuming the record exist on the source but not on the target, we suspect it was missed by the full_load, in this case, we'll need to run full load again with the following settings:

a. Set "Source_capture" and "Target_apply" detailed debug.

b. Add to the target DB connection string the following:

keepcsvfiles=true

this will keep the CSV file under:

~\Attunity\Replicate\data\tasks\<task name>\redshift\applied_files

c. In the task settings:

Change Processing->Store change settings

Click the icon at the top of the page so you'll see the message:

Store changes processing is ON

After we check the logs, we might ask for information from the "change tables".

4. After the full load completes, please send us the task logs and the CSV files, we'll check them and UPDATE you with our findings.

4. Missing Events during Full Load

4. Missing Events during Full Load

Missing events during Full load won't happen often, but the main case scenario would be:

- An INSERT operation on the source was not captured. For instance, when there's a timeout in ODBC and the Full load continues executing.

- An INSERT operation on the source was captured but not applied to target. For instance, when no commit happens after a certain amount of changes.

- An UPDATE to a PK was moved to a different partition.

- Filters were set on the table but were not configured properly.

- Source/target had an error but didn't report it and the Full load continued.

- Parallel Full load/parallel CSV load can be a cause of issues.

Debugging:

Add SOURCE_UNLOAD and TARGET_LOAD logging on Trace and check for any issue, as each endpoint may print different information.

For ODBC set to Trace for both ODBC source/target.

On CSV targets (like Redshift, MySQL, Postgres, snowflake, Postgres, Hadoop , etc.) you can also use keepCSVFiles internal parameter.

Example of source issue:

MySQL ODBC driver sometimes didn't deliver all the records if some timeout was received during full load, but it didn't report any error.

This can be checked from the source unload report row count.

Example of an issue in the log – if the table has 1000 rows, but the source only reads 500 rows:

00033332: 2019-07-02T12:20:35 [SOURCE_UNLOAD]I: Unload finished for table 'F'.'TBFJ91' (Id = 141). 500 rows sent. (streamcomponent.c:3498)

00033333: 2019-07-02T12:20:35 [TARGET_LOAD]I: Load finished for table 'F'.'TBFJ91' (Id = 141). 500 rows received. 0 rows skipped. Volume transfered 183128. (streamcomponent.c:3787)

00033297: 2019-07-02T12:20:35 [TASK_MANAGER]I: Loading finished for table 'F'.'TBFJ91' (Id = 141) by subtask 7. 500 records transferred. (replicationtask.c:2290)

5. Missing Events during CDC

5. Missing Events during CDC

Events can be missing for different reasons:

- An INSERT/UPDATE/DELETE was not captured from the source during CDC.

- The source was not able to read the events from the transaction log.

- The event was captured but the source has read a rollback instead of commit.

- The event was captured but didn't begin transaction read.

- Transactional consistency.

- Data object id missing, because of a partition operation/truncate.

- Filtered by sorter in cached events incorrectly, due to an internal issue or time difference on the source and on the replicate server.

- Filtered by sorter due to another internal issue.

- Target apply issue, caused by the loss of context/event id after a recoverable error.

- Target apply issue, due to bulk apply in order of events.

- Filters were set on the table but not configured properly.

6. When Missing Events are Expected

- In bulk apply, if target has more than one PK/UI, changes might get lost
- A PK/UI column is of float or double datatype – these are not accurate datatypes
- When there is a CDC only task that is not started with the correct timestamp

  Example: A Full Load task is started at 8:00 am and completes at 10:00 am followed by a CDC only task that starts at 9:55 am will miss all changes that happened between the start 8:00am and the completion of the Full Load

7. Trouble Shooting Missing Events

7.1 Need to get:

Important to have (as much as possible)

- **Diagnostic package**
- **Full task logs (as Diagnostic package may trim logs to 10MB)**
- **Example of data lost and time it was lost/Content of apply_exceptions table from target (could be filtered to actual problem task/table/time).**
- Full DDL of table on source (oracle script)
- Target table DDL
- Log with verbose – see other slide for more details
- Database versions of source and target
- Anything changed/ significant event like error or maintenance job happened on source database/replicate server /target database

7.2 Nice to have:

- Time zones on source database , replicate server and target database , including daylight savings settings,
- Are the clocks on source database and replicate server are synced (e.g. time difference of 1 minutes regardless of time zone)

7.3 Trouble-Shooting Missing Events – CDC

7.3.1 You can check first for obvious reasons for missing events

1. **What are the primary key/unique index columns of the table**? Anything usual , e.g.

float/double datatype – not accurate

, string datatype – compare of string in where clause can be different on source and target (i.e case sensitive),

2. **Check if source and target tables have same primary** key/unique index:. Having one unique index on source and one on target but with different columns on target, can cause duplicate errors that will actually not be duplicates

- **Also If target table has more than one PK/UI constrain**t , that can be a problem in bulk apply causing missed events.
- **Any filters on table?** They might be causing a missed data. Also filters should be on columns that don't change ever

7.3.2 Initial things to check:

1. Get at least one example of a missed event from task log/apply exceptions. Try to get the insert time in case of update
2. Is record found on target/source – If it is on target it is not a missing event but something else. If on source - get full record, if not on source – there was a delete
3. Check on any Warning/errors/assertions in the logs in general and in apporx. Time of insert
4. Was there any of these around time of issue:
    1. Task crash
    2. Task stop/resume
    3. error
    4. Detach/attach
    5. Going to One by one
5. Does the table of missing event contain a "create date"/insert date/update date etc. column? If so, try to see when was the missing event – during full load , during cached events , during regular CDC? Did anything special happened around time of missed event – an error, a connection issue, a detach/attach, a crash, a task stop/resume.

7.3.3 More things to check

- Check if target table contains a column that contains the insert time of the record to the table
- Did this happened during full load or regular CDC?
- What happened around that time – stop task or target detach/attach 3. if the events missed during full load
- If possible, try to reproduce the missing events issue with SOURCE_CAPTURE, SORTER , TARGET_APPLY and PERFROMANCE verbose logging
- If you find an insert with the same primary key value, then the issue may be related to the sorter or target. You will then need to get the transaction id for that insert, and check if there was a commit on the transaction or not.
- If the event was missing during full load , we need to isolate the issue some more and see if the event was inserted before full load started and was not replicated by the full load process itself or event happened on source during full load , and so this event was a “cached event”.
- If the event happened before full load started, then we need to understand why the record was not captured in full load process. Please set the task to TARGET_LOAD verbose logging, and if the target is a CSV target also set KeepCSVFiles and KeepErrorFiles internal properties and do a reload table again (you can also create a separate task only for that table).try to look for errors during full load that are hiding in debug. Try to locate the missing record in the CSV files. Known issues – duplicates on MySQL , postgres autocommit
- If the event was missing during cached events, check any timezone difference or clock difference between source and target. Check if the source is Oracle standby – standbydelay time

7.3.4 Things that can be a problem:

1. Time zone and clock difference between source database and replicate server
2. Full load done by a different tool than replicate – different translation of pk values

7.3.5 If nothing of the obvious reasons is relevant

- If possible, try to reproduce the missing events issue with SOURCE_CAPTURE, SORTER , TARGET_APPLY and PERFROMANCE verbose logging,
- Also would be good if you can add “Store changes” to the task setting to get all data in CT tables as well, and use KeepCSVFiles and KeepErrorFiles internal parameters for CSV targets (like Redshift, snowflake, MySQL, Postgres, Hadoop etc.). This is not a must, at least at first stage, but be can help.
- If issue doesn’t happen all the time, or in order to save log file size, you can also setup a test dummy task with only problem table , going to some other target (like CT table or NULL) with verbose logging and start that task from timestamp or around the time of missing event
- In the verbose log, if you have source capture verbose , try to find the event with the same primary key value. If the source was able to read this record. Check the entire transaction of the change – did we capture the begin, did we capture the commit (or there was a rollback) if we capture the begin and commit, then issue might be in sorter/target, then the issue may be related to the sorter or target. Follow the SORTER verbose logs if transaction was passed to target and the TARGET_APPLY verbose log if target got it.
- If sorter didn’t pass it – check why it was filtered (cached event before full load time, table is suspended etc.) if it got to target, then see why it was not applied

7.3.6 Found the insert event with same PK in source capture?

- Look at transaction id – is there a begin? An commit/rollback?
- If there is begin and commit, did the sorter got the event?
- If sorter got the event, was it sent to target stream? (on Oracle binary/SAP etc. it will need to first though differed constructor)
- If it was not sent to target stream – why?
    - Table suspended?

- Commit before full load?
- Detach mode

- If it was sent to target stream, did target get it?
- If so did it run the apply statement with no error?

7.4 Troubleshooting CDC Missing Data using Change Tables

Problem Description

Data quality testing is showing missing data on the Target. The customer compared the source and the target confirming the missing data.

In this article we will suggest several approaches to debugging the missing events using CT tables.

The Task may already have CT tables enabled, and at some cases it makes sense to add Change Tables to the existing Task. However, if the user has a large number of tables, or if the storage demand is so high, another approach is to create a temporary Task adding only a subset of tables.

OPTION 1 – Task already has Change Tables

Having Change Tables may give us more information and enable faster analysis of the missing data. If the missing INSERT for example appears in the CT and not in the Target table, it could mean that the missing data is not related to the SOURCE.

In this case it may be related to the apply process, for example the Bulk or the Target itself.

The Change Tables by default have the "__ct" suffix which is configurable. Change Tables include header fields and the data for each change (or event):

- INSERTs
- DELETES
- UPDATES in the form of BEFORE and AFTER images (when supported).

The header fields enable the research of the data by providing the information about each change including the operation type, transaction ID and the timestamp of the original change (review the manual for more details).

OPTION 2 – Existing Task

Replicate allows you to add the Change Tables to an existing Task. If this option is feasible, you need to careful modify the Task:

- Make sure the Task is not processing a large batch of transactions.
- Select a quiet period to Stop the Task and perform the change.

- We always recommend testing this approach using a TEST machine, and if that is not possible go to OPTION 3.

To add CT to an existing table, follow these steps:

- Stop the Task.
- Click on the Task Settings -- Change Processing -- Store Changes Settings.
- Enable the Store Change Processing by click on the Icon.

The text near the icon should read: Store changes processing is ON.

- Create the missing tables.

Click on Run -- Advanced Run Options then select the “Metadata only. Create missing tables and stop”

This will create all the __ct tables in the Target. Review the target database and confirm that all the __ct tables are generated.

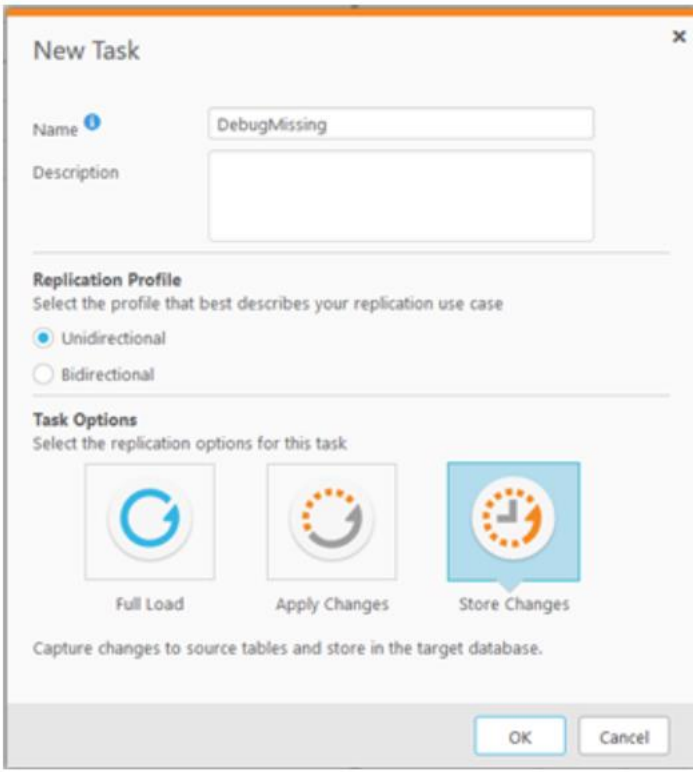
- Resume the Task.

OPTION 3 – Create a Temporary CT Task

This option requires the user to create a new Task to debug the missing data. This Task will help analyze the missing data and determine if it always occurs or if it’s intermittent.

To create a temporary CT Task, perform the followings:

- Create a CDC only Task with Store Changes:



- Add the tables you want to investigate.

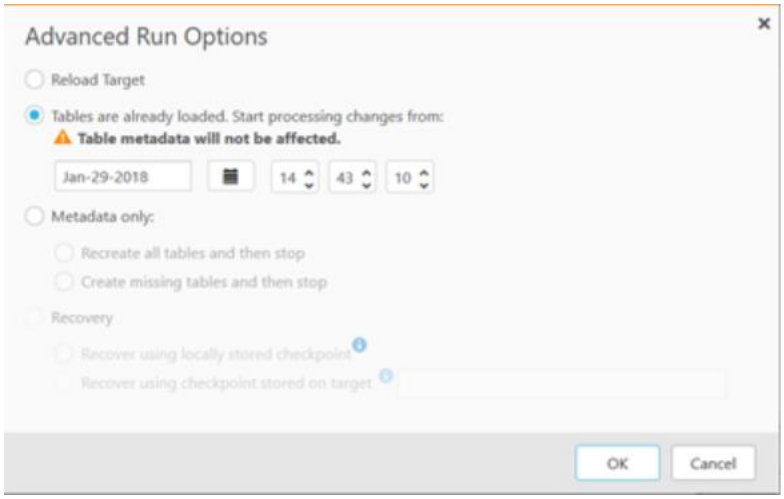
If the missing change is part of a transaction that includes several tables, you may want to add these tables.

- You may set the target schema to be different than the production tables. This will isolate these CT tables if you don't plan to add them as part of the normal work flow.

To modify the owner:
Task Settings -- Metadata -- Target Metadata

- Start the processing from a specific timestamp. This timestamp should be around the time of the suspected missing events. Usually you would want to go back more in time, the reason if this change or changes are part of a long running transaction, we would want to capture the transaction from the start.

To start from timestamp, click on the Run drop-down then Advanced Run Options and select the "Tables are already loaded. Start process changes from":



Analyzing the Change Tables

The resulting CT tables will allow you and Attunity's support to analyze the missing data. Always attach these results to the case in the support portal in parallel to your analysis.

Questions to ask:

- If the missing data is in the CT and not in the target table. We need to focus on the apply process.
- If the missing data does not exist in the CT and the target, then we need to focus on the Source.

7.5 Troubleshooting Missing Data during Full Load

Problem Description

The missing data problem in Full Load is not common, and can be divided into two categories:

1. Full Load only Tasks
2. Full Load and CDC

The Full Load only Tasks are very straightforward and the main issue here if the "missing data" is related to actual failures or data changes which occurred during the Full Load. When adding CDC, the cached changes should also be taken into account.

In many systems some of the tables may include one timestamp field or more, this identifies the actual creation date/time of the record, and at times the last modified date/time. This actually depends on the design of the tables, and may exists when the database serves a software package like CRMs, ...

These fields will make it easier debug such issues, giving us a clear indication of time.

Suggested approach:

- Select one of the simple table(s) for the debugging purpose, make sure it includes timestamp fields.
- Review the start time of the Full Load from the Task's Full Load Monitor view, or review the log file.
- Review the log file of any failures related to load process.

Hints:

- Search for errors or warnings related to the Load.
- Search for data and truncation errors.
- Review the Source and Target table DLLs:
    - Data type differences.
    - Nullability issue.
    - Primary Key and Unique Indexes differences.

Cached Events

When the Task is both Full Load and CDC, an intermediate process called "Cached Events" will be in use. Cached events are those events captured during the Full Load. These will be applied immediately after the Full Load finishes and just before starting the CDC process.

If the timestamp of the records indicate that the missing data is related to the cached events. It's important to review the log file for any failures.

Information Required by Support

To help R&D and Support to debug such problem, please a package with the following information:

1. Export of the Task(s).
2. Log file(s) of the Full Load process.
3. Metadata of the table(s) from the Source and the Target.
4. Example(s) of missing data records.

If the problem is reproducible add the following logging and attach new log:

- SOURCE_UNLOAD to Verbose.
- TARGET_LOAD to Verbose.

7.6 Important Notes about missing events

7.6.1 Oracle Source Endpoint Enhancements

Replicate 6.5 introduces the following changes in functionality:

In previous versions, Replicate would detect open transactions on the RAC master (primary) node only, which would sometimes result in changes not being captured.

From 6.5 version, Replicate is now able to detect open transactions on all RAC nodes.

Customers wishing to benefit from this new functionality need to grant the Replicate user the following permissions:

Regardless of which method is used to access the redo logs:

Grant SELECT ON GV_$TRANSACTION (for detecting open transactions on all RAC node)

When using Attunity Log Reader to access the redo logs:

Grant SELECT ON V_$DATABASE_INCARNATION (for capturing RESETLOGS operations)

Note If Replicate detects that the user (specified in the endpoint settings) does not have the requisite permissions, a warning will be written to the log and the task will continue as normal. However, when replicating from a RAC environment and/or wish to capture RESETLOGS operations, not setting these permissions may result in data loss and/or unpredictable behavior.

2DON'T use SCN, it behaves much like start from timestamp and you are almost guaranteed to either lose or duplicate events.As for Checkpoint, we cannot assure prevention of data lose or duplicates by the following rules:It depends on the target only:

Transactional DB

1. Am I to expect missing events? – No

2. Am I to expect duplicate events? – Yes

Big Data (Other than Kafka)

1. Am I to expect missing events? – Yes

2. Am I to expect duplicate events? – No

Kafka

1. Am I to expect missing events? – No

2. Am I to expect duplicate events? – Yes

Explanation (Assuming there were events, if not the checkpoint would contain only timestamp):

The checkpoint contains two “numbers”:

1. The change position on the longest running transaction i.e. SCN,LSN,….or the stream position (In case of log stream)

2. The last confirmed committed record position.

Now on a transactional DB there is sometime a delay of the confirmation

e.g.1. The “temporary tables” on the target still have data that has not been delivered, on the next execution, Replicate would pile the same data on these tables and thus “duplicate” the events piled on these tables.

2. The confirmation happens only when data from the sorter has been deleted as well, the crash might happen before that happens (and after the data was committed on the target).

On a **Big Data DB** the confirmation is generated once the local files have been created, before the data actually went to the target.

So in a case of crash, it is most likely that we lose events (unless all the files have been delivered).

On a **Kafka target** we have a tool called Message Tracker (MT).

We treat a record as delivered only when we get the acknowledgement that it has indeed delivered (no acknowledgement is therefore not supported and would always result data loss).

So it is unlikely that we have missing events.

Since we wait for the acknowledgement the message might have went through but, if we have crashed before the confirmation has been received or the MT is malfunctioning, we might send the same event/s again.

8. Oracle Source Missing Events

8.1 Oracle Source Missing Events

- All the general reasons for missing events can also happen when source is Oracle
- **What is special in each endpoint like Oracle, in the items that are endpoint specific**:
    - How to translate values of data that we know of (values of PK columns or other columns) to values printed in verbose logs and vice versa
    - The stream position of Oracle
    - CDC flow that is unique to Oracle e.g:
        - RAC thread used , the merger ,
        - the default constructor,
        - Binary errors/assrtions
        - CDB/PDB
    - Supplemental logging not always
    - trouble-shooting tools : export REDO log dump, log miner queries

8.1.1 Translating values

- Oracle keeps the data on REDO in the format of RAW datatype
- To translate from RAW to the actual datatype and vice versa you can use any conversion you can find on the web for data type conversion from RAW to the datatype
- Here are some know converstions
- **From RAW to number:**

  select utl_raw.cast_to_number ('C30F1F06') from dual;
- **From RAW to varchar:**

  select utl_raw.cast_to_varchar2('410142') from dual;

- **From NUMBER to RAW**

  select utl_raw.cast_from_number(54) from dual;

From VARCHAR to RAW

select UTL_RAW.CAST_TO_RAW ('BMG6620C1038') from dual;

Raw to timestamp –use script

9. Upsert Merge

9.1 Upsert Merge – Introduction

The Upsert Merge settings for a task are configured in the Apply Conflicts Screen

Upsert: Change an update to an insert if the row doesn't exist on the target

Merge: Change an insert to an update if the row already exists on the target

9.2 Upsert Merge – How it is implemented

The way the Upsert Merge works is dependent on the task Apply Changes setting

When in: Batch Apply: mode – The task will do an unconditional Delete of all rows in the batch, followed by an Insert of all rows in the batch. In batch apply mode the task will actually issue a pair of transactions (1st a delete of the record and then 2nd an insert) this pair of transactions is unconditional and will result in a "newly inserted row every time the record is updated on the source.

When in: Transactional Apply: mode – The source transaction, Insert or Update, statement is run against the target and if it errors out then the switch is done (try and catch). If the task is in Transactional apply mode, then the update will be performed in a "try / catch" fashion. The update will be run and only if it fails will it be inserted. This way the original insert date/time will be preserved for records that are not missing.

9.2 Upsert Merge – Gotcha’s

1) If the table has added fields that are used/modified by a downstream process then there is a good chance that this will cause a problem. In example: A task, running in Batch Apply mode, that has added columns to the target that capture Insert Date “for use in a downstream process” will have the Insert Date overwritten when the Merge option is selected, because the target row will be deleted and then reinserted.

2) Oracle table must have supplemental logging all column set

3) Masking of possible underlying issue - why was the record not there for an update or why was it there already for and insert.

4) This setting will not bring over all missing rows from source to target. Note that there may be other missing records (from and insert) that was not captured, and this setting will only bring over missing records if they are updated on the source.

9.2 Upsert Merge – Possible Use Cases

1) Missing values in some fields/missing rows on target. With Upsert/Merge set a "fake" update can be issued on the source (Update

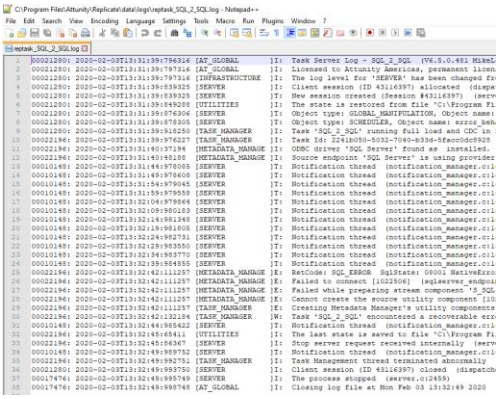
field1 = field1) without a where clause will force every row in the source to refreshed to the target (without losing the target data like a reload would)

2) Prevent a task from switching out of bulk apply due to error (appliance targets could benefit from this)

3) Could help the CDC only task of split task configuration – (one task is Full Load and one task is CDC only) by allowing an overlap of the CDC timestamp used to run the CDC task. The upsert merge setting will prevent the task from error out of batch mode.

10. Helpful Tools

10.1 Helpful Tools – Viewing Logs



Viewing Log Files:

One of the best things you can do to make troubleshooting missing data is to have good tools available for viewing the log files. Depending on the log levels set the task logs can grow very large very quickly.

• Large file viewer – V editor, Total commander/lister

•Grepping strings in big files – grep, findstr

•Many logs – script to find keywords

•Notepad++ - if file not too big

10.2 Helpful Tools – Log Miner Queries - 1

Log miner queries to find events in REDO log

1. add required REDO log

.begin

sys.DBMS_LOGMNR.ADD_LOGFILE('**/attlogs/oradata/archivelogs/miwayarch_1_390628_745673251.dbf**', sys.DBMS_LOGMNR.NEW); end;

2. Then we will start a Log miner session , with the start SCN and EndSCN values. Example command:

begin

SYS.DBMS_LOGMNR.START_LOGMNR(

startScn => 8352283449700, endScn => 8352283477350);

end;

10.3 Helpful Tools – Log Miner Queries - 2

Log miner queries to find events in REDO log

3. Logminer will read all the logs that we have added, parse query the v$logmnr_content view.for changes

```
select operation, NVL(sql_undo, ' '), xidusn, xidslt, xidsqn, NVL(seg_name, ' '),

SCN,

RBASQN, RBABLK, RBABYTE, NVL(seg_owner, ' '), NVL(sql_redo, ' '), CSF, rollback,

NVL(row_id, ' '), timestamp, NVL(username, ' ')

from v$logmnr_contents

where

SCN >= 8352283449700

and ((operation IN ('INSERT','DELETE','UPDATE','DIRECT INSERT'))

or operation IN ('START','COMMIT', 'ROLLBACK', 'DDL'))
```

10.4 Helpful Tools – REDO log dump

REDO log dump instructions:

1. Copy the relevant log file to a directory on the Oracle machine.

2. Login to Oracle the run the following commands:

a. first command to increase size of REDO log dump to unlimited

alter session set max_dump_file_size='unlimited';

b. (optional, but will help you to find the dump file) - give an identifier to the trace file name

alter session set TRACEFILE_IDENTIFIER='replicate_REDO_dump1';

c. Then , run the command to dump the log file:

alter system dump logfile '<PATH_TO_REDO_LOG>';

(replace <PATH_TO_REDO_LOG> with the path to the REDO log file. For example: ' / arch_1_151764_582990794.arc

3. Now, the dump file will be found in the directory pointed by user_dump_dest, and if you TRACEFILE_IDENTIFIER in step 2(a), it will be part of the file name