Managing Recapturing Using Session Names

Contents

In HVR, session name is the name of a transaction performed by the integrate job. In all DBMSes, for which capture is supported, transactions performed by the integrate job also contain an update to the integrate state table (named **hvr_stin*** or **hvr_stis***) which has column **session_name** containing the session name. This allows log-based capture jobs to recognize the session name defined by the integrate job, but this is ignored by the trigger-based capture jobs.

The default session name is **hvr_integrate**. Normally, HVR capture avoids recapturing changes made on the integration side by ignoring any changes made by sessions named **hvr_integrate**.

Replication recapturing is when changes captured on a source location are captured again on the integration side. Recapturing may be controlled using session names and actions **Capture /IgnoreSessionName** and **Integrate /SessionName**. Depending on the situation recapturing can be useful or unwanted.

The following are typical cases of how recapturing is managed using session names.

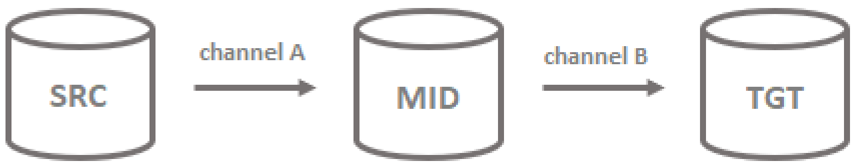
Cascade Replication

Cascade replication is when changes from one channel are captured again by another channel and replicated to a different group of databases.

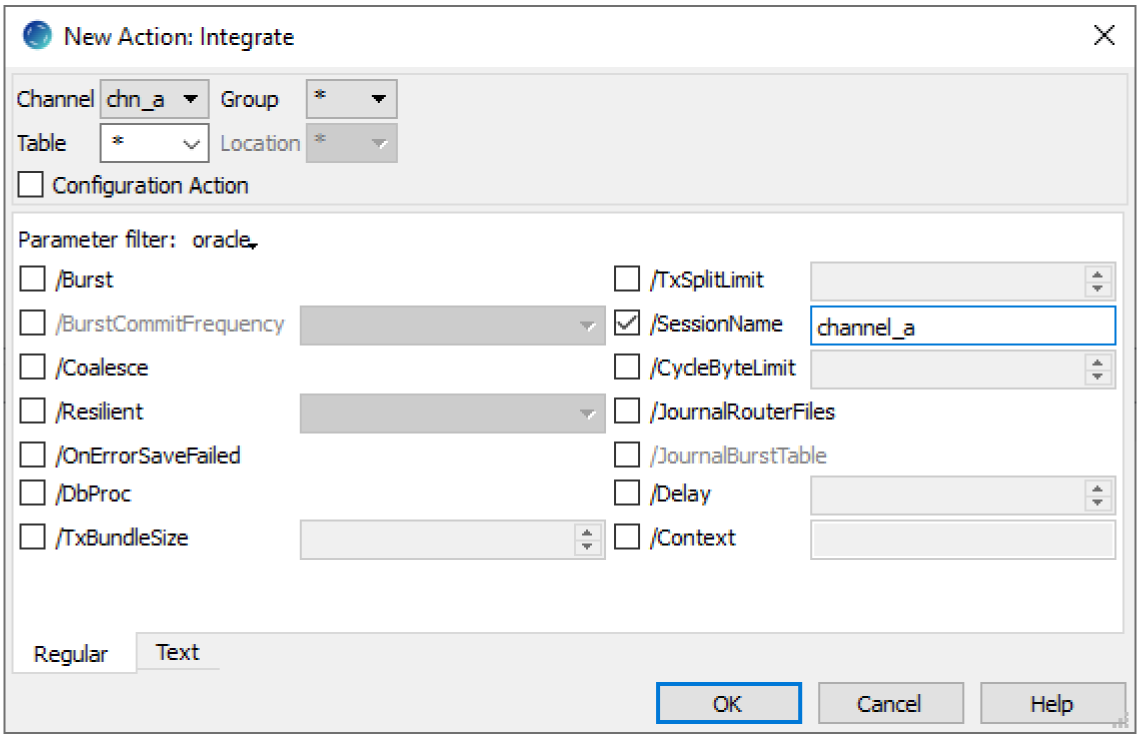
Recapturing is necessary for cascade replication. In this case, recapturing can be configured using action **Integrate /SessionName** so that integration is not recognized by the capture triggers in the cascade channel. This action allows to integrate changes with specific session name.

Example

Changes made by **channel A** do not get captured by **channel B** by default. The default session name of **channel A** in database **MID** is **hvr_integrate** and the capture job of **channel B** ignores sessions with the **hvr_integrate** name.



To capture changes, the default session name of **channel A** must be changed from the default value to a different name. Do this by setting the **/SessionName** parameter to another value, for example **channel_a**.



New Action: Integrate

Channel chn_a Group *
Table * Location *
☐ Configuration Action

Parameter filter: oracle

☐ /Burst		☐ /TxSplitLimit	
☐ /BurstCommitFrequency		✓ /SessionName	channel_a
☐ /Coalesce		☐ /CycleByteLimit	
☐ /Resilient		☐ /JournalRouterFiles	
☐ /OnErrorSaveFailed		☐ /JournalBurstTable	
☐ /DbProc		☐ /Delay	
☐ /TxBundleSize		☐ /Context	

Regular Text

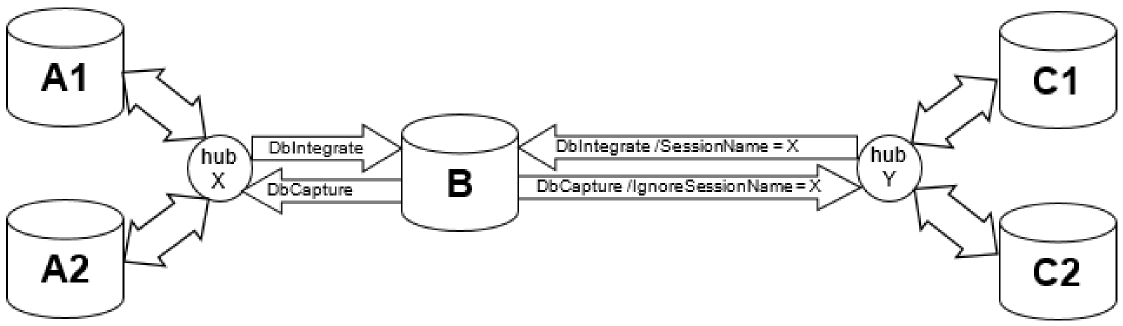
OK Cancel Help

Bi-directional Replication

During bidirectional replication, integrated changes can be accidentally captured again boomeranging back to the original capture database. This will form an infinite loop. For this reason, action Capture **/IgnoreSessionName** should be used to check the session name and avoid recapturing on the integration side. This action instructs the capture job to ignore changes performed by the specified session name.

Example

Changes made to databases **A1** and **A2** must be replicated via **hub X** to database **B** and then cascade via **hub Y** to databases **C1** and **C2**. Changes made to **B** must replicate to **A1** and **A2** (bi-directional replication) and to **C1** and **C2**, but must not boomerang back to **B**. Normally changes from **A1** and **A2** to **B** would not be cascade replicated onto **C1** and **C2** because they all use the same session name (**X**). This is solved by adding parameters **/SessionName** and **/IgnoreSessionName** to the channel in **hub Y**.


A1
A2
hub X
DbIntegrate
DbCapture
B
DbIntegrate /SessionName = X
DbCapture /IgnoreSessionName = X
hub Y
C1
C2

Bi-directional Replication using MySQL

By default, during HVR Bulk Refresh, a target table is truncated and then a bulk copy of data from the source table is transmitted to the target table.

For a MySQL database, HVR does not associate a truncate table operation with a session name, i.e. HVR does not distinguish between the truncate table operation initiated by the HVR Bulk Refresh and the truncate table operation initialed by a third-party user application.

Therefore, in MySQL bi-directional setup, the HVR Bulk Refresh may cause looping truncates on either side of the bi-directional setup, which in turn will result in all data getting deleted.

One possible workaround to this is to define action **Restrict** with parameter **/RefreshCondition** set to ‘1=1’ on a channel. This will delete all data from the table not truncate the table. Deleting the data will generate more transaction log output and take longer than truncating, but in the bi-directional setup, deleting will not cause the loopback issue.

Another approach for this would be to suspend the capture job when truncates happen (e.g. as part of the B ulk Refresh) and reset the capture time (using option Capture Rewind) beyond the time when the truncates happened before restarting the capture job.

Batch Work to Purge Old Data

Sometimes replication of large blocks of batch work is too expensive. Capturing of the batch work can be disabled by setting the session name so that the capture job ignores the changes done by that batch job. For log-based capture, the session name must be set using the 'current user name' approach (see below), as other approaches are not recognized for this.

Application Triggering During Integration

HVR has multiple implementations for session names because of variations in the DBMS features and because some implementations can only be recognized by trigger-based capture jobs and others can only be recognized by log-based capture jobs.

Sometimes an application will have database triggers on the replicated tables. These triggers have already been fired on the capture database so firing them again during HVR integration is unnecessary and can cause consistency problems. For Ingres and SQL Server databases, this rule firing can be avoided with action **Integrate /NoTriggerFiring**.

Current user name

In all DBMSes, for which capture is supported, trigger-based capture jobs also recognize the current DBMS user name as the session name. Therefore, end user sessions can make changes without activating the trigger-based capture by connecting to the database using a specially created user name (e. g. making database changes as user name **hvr_integrate**). An application database trigger can be prevented from firing on the integration side by changing it to include the following clause: **where user <> 'hvr_integrate'**.

Ingres role

For Ingres trigger-based capture, the integrate job always connects to the target database using an Ingres role, which is the name of the session (e.g. **hvr_integrate** or **hvr_refresh**). This is recognized by the trigger-based capture jobs. Therefore, end-user sessions can make changes without activating the trigger-based capture by connecting to the database with **SQL** option **–R**. An application database trigger can be prevented from firing on the integration side by changing it to include the following clause: **where dbmsinfo ('role') != 'hvr_integrate'**.

SQL Server marked transaction name (log-based capture)

For SQL Server, log-based capture jobs also recognize the marked transaction name as the session name. Therefore, end user sessions can make changes without activating log-based capture by executing their DML statements inside a marked transaction with a specially created name (e.g. **hvr_integrate** or **hvr _refresh**). The following SQL statement can be used to start a marked named transaction:

```
begin transaction hvr_integrate with mark;
```

SQL Server application name (trigger-based capture)

For SQL Server trigger-based capture only, the integrate job always connects to the target dataset using an application name which is the session name (e.g. **hvr_integrate**). End user sessions can therefore make changes without trigger-based capture being activated by also connecting to the database using a specific application name. An application database trigger can be prevented from firing on the integration side by changing it to include the following clause: **where app_name() <> 'hvr_integrate'**.