# Capture

**Contents**

## Description

Action **Capture** instructs HVR to capture changes from a location. Various parameters are available to modify the functionality and performance of capture.

For a database location, HVR gives you an option to capture changes using the location's **Capture_Method** property (log-based or trigger-based method). HVR recommends using the log-based data capture because it has less impact on database resources as it reads data directly from its logs, without affecting transactions, manages large volumes of data and supports more data operations, such as truncates, as well as DDL capture. In contrast, the trigger-based data capture creates triggers on tables that require change data capture, so firing the triggers and storing row changes in a shadow table slow down transactions and introduces overhead.

When defined on a file location this action instructs HVR to capture files from a file location's directory. Changes from a file location can be replicated both to a database location and to a file location if the channel contains table information. In this case any files captured are parsed (see action **FileFormat**).

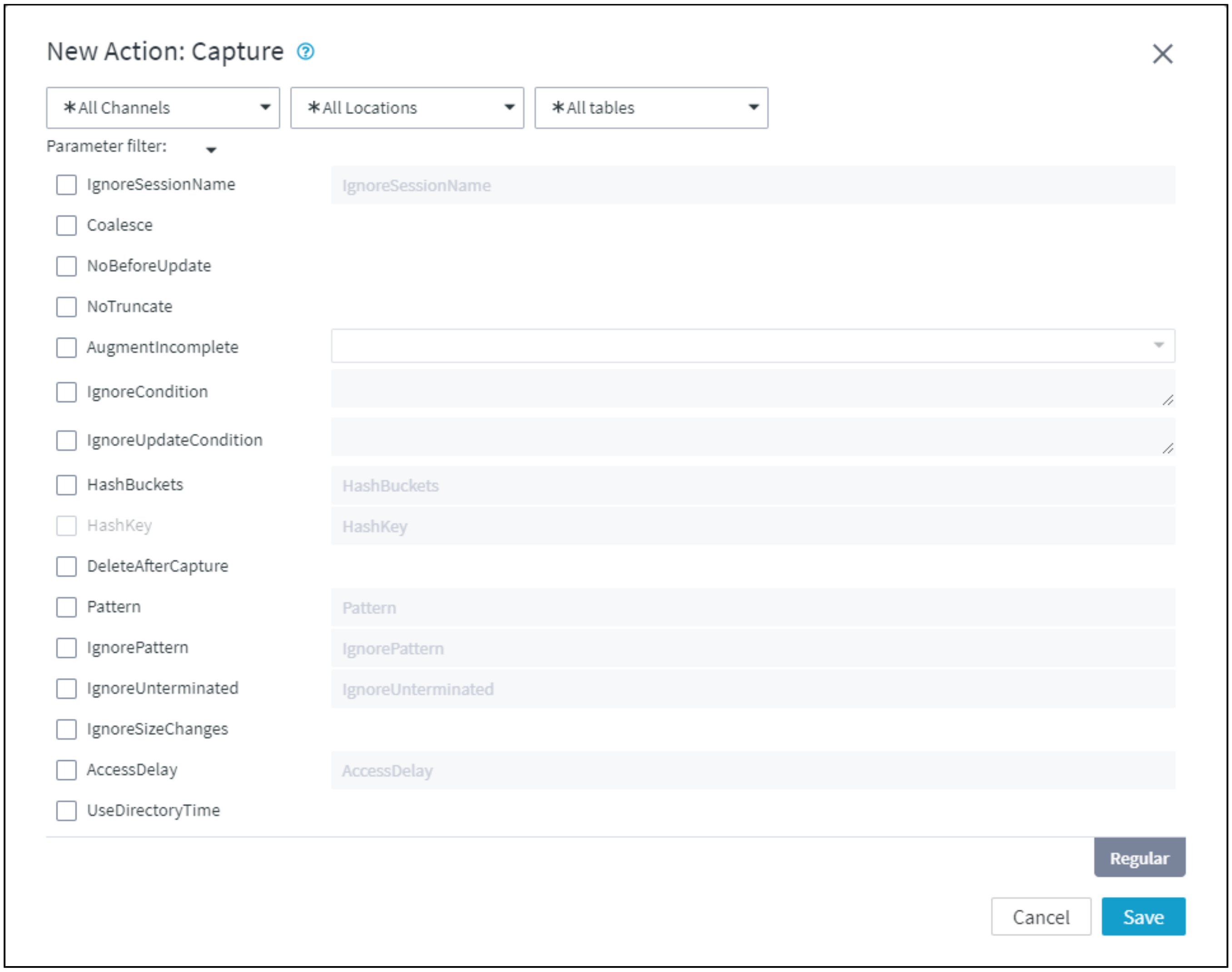
If **Capture** is defined on a file location without table information then each file captured is treated as a 'blob' and is replicated to the integrate file locations without HVR recognizing its format. If such a 'blob' file channel is defined with only actions **Capture** and **Integrate** (no parameters) then all files in the capture location's directory (including files in sub-directories) are replicated to the integrate location's directory. The original files are not touched or deleted, and in the target directory the original file names and sub-directories are preserved. New and changed files are replicated, but empty sub-directories and file deletions are not replicated.

Bidirectional replication (replication in both directions with changes happening in both file locations) is not currently supported for file locations. File deletion is not currently captured by HVR.

If **Capture** is defined on a file location without parameter **DeleteAfterCapture** and location property **File_State_Directory** is used to define a state directory outside of the file location's top directory, then HVR's file capture becomes read only; write permissions are not needed.

## Parameters

This section describes the parameters available for action **Capture**. By default, only the supported parameters for the selected location class are displayed in the **Capture** window.


New Action: Capture

*All Channels | *All Locations | *All tables

Parameter filter:

- IgnoreSessionName: IgnoreSessionName
- Coalesce
- NoBeforeUpdate
- NoTruncate
- AugmentIncomplete
- IgnoreCondition
- IgnoreUpdateCondition
- HashBuckets: HashBuckets
- HashKey: HashKey
- DeleteAfterCapture
- Pattern: Pattern
- IgnorePattern: IgnorePattern
- IgnoreUnterminated: IgnoreUnterminated
- IgnoreSizeChanges
- AccessDelay: AccessDelay
- UseDirectoryTime

| Parameter | Argument | Description |
|---|---|---|
| **IgnoreSessionName** | *sess_name* | This action instructs the capture job to ignore changes performed by the specified session name. Multiple ignore session names can be defined for a job, either by defining **IgnoreSessionName** multiple times or by specifying a comma separated list of names as its value.<br><br>Normally HVR's capture avoids recapturing changes made during HVR integration by ignoring any changes made by sessions named **hvr_integrate**. This prevents looping during bidirectional replication but means that different channels ignore each other's changes. The session name actually used by integration can be changed using action **Integrate** with parameter **SessionName**. For more information, see Managing Recapturing Using Session Names.<br><br>If this parameter is defined for any table with log based capture, then it affects all tables captured from that location.<br><br>An example for using this parameter is available in section Using IgnoreSessionName (below). |
| **Coalesce** | | Causes coalescing of multiple operations on the same row into a single operation. For example, an **INSERT** and an **UPDATE** can be replaced by a single **INSERT**; five **UPDATE**s can be replaced by one **UPDATE**, or an **INSE RT** and a **DELETE** of a row can be filtered out altogether. The disadvantage of not replicating these intermediate values is that some consistency constraints may be violated on the target database.<br><br>This parameter should not be used together with parameter **SapUnpack** in action **Transform**. |

| | | |
|---|---|---|
| **NoBeforeUpdate** | | Do not capture 'before row' for an update. By default when an update happens HVR will capture both the 'before' and 'after' version of the row. This lets integration only update columns which have been changed and also allows collision detection to check the target row has not been changed unexpectedly. Defining this parameter can improve performance, because less data is transported. But that means that integrate will update all columns (normally HVR will only update the columns that were actually changed by the update statements and will leave the other columns unchanged).<br><br>If this parameter is defined for any table with log based capture, then it affects all tables captured from that location. |
| **NoTruncate** | | Do not capture SQL truncate table statements such as **TRUNCATE** in Oracle and **modify** *mytbl* **to truncated** in Ingres.<br><br>If this parameter is not defined, then these operations are replicated using **hvr_op** value **5**.<br><br>For DB2 for z/OS, this parameter affects only **TRUNCATE IMMEDIATE**. HVR will always capture **TRUNCATE** if used without **IMMEDIATE** option (this will be replicated using **hvr_op** value **0**).<br><br>This parameter is not supported for Microsoft SQL Server. |
| **AugmentIncomplete** | *col_type* | During **capture**, HVR may receive partial/incomplete values for certain column types. Partial/incomplete values are the values that HVR cannot **capture** entirely due to technical limitations in the database interface. This parameter instructs HVR to perform additional steps to retrieve the full value from the source database, this is called augmenting. This parameter also augments the missing values for key updates.<br><br>• Defining this parameter can adversely affect the **capture** performance.<br>• If this parameter is not defined, and when a partial/incomplete value is received, the **capture** will fail with an error.<br><br>Valid values for *col_type* are:<br><br>• **NONE** (**default**): No extra augmenting is done.<br>• **LOB**: Capture will augment partial/incomplete values for all columns of a table, if that table contains at least one lob column. For key-updates, missing values are augmented too.<br>• **ALL**: Capture will augment partial/incomplete values for all columns of any table. For key-updates, missing values are augmented too.<br><br>In certain situations, the default behavior changes and defining **AugmentIncomplete** can only override the behavior with a 'stronger' value.<br><br>• For DB2 for Linux Unix and Windows, **LOB** should be selected to capture columns with **xml** data type.<br>• For DB2 for z/OS, the **default** *col_type* is **LOB** and can only be changed to **ALL**.<br>• For Oracle, when capturing using Logminer (**Capture_Method** = **LOGMINER**), the **default** *col_type* is **LOB** and can only be changed to **ALL**.<br>• For SQL Server, when capturing using SQL Access (**Capture_Method** = **SQL**), and tables contain non-key columns, the **default** *col_type* is **ALL** and can not be changed. |
| **IgnoreCondition** | *sql_expr* | Ignore (do not capture) any changes that satisfy expression *sql_expr* (e.g. **Prod_id < 100**). This logic is added to the HVR capture rules/triggers and procedures.<br><br>This parameter differs from the **Restrict /CaptureCondition** as follows:<br><br>• The SQL expression is simpler, i.e. it cannot contain subselects.<br>• The sense of the SQL expression is reversed (changes are only replicated if the expression is false).<br>• No 'restrict update conversion'. Restrict update conversion means if an update changes a row which did not satisfy the condition into a row that does satisfy the condition then the update is converted to an insert.<br><br>This parameter requires location property **Capture_Method** set to **DB_TRIGGER**. |
| **IgnoreUpdateCondition** | *sql_expr* | Ignore (do not capture) any update changes that satisfy expression *sql_expr*. This logic is added to the HVR capture rules/triggers and procedures.<br><br>This parameter requires location property **Capture_Method** set to **DB_TRIGGER**. |
| **HashBuckets**<br><br>Ingres | *int* | Identify the number *int* of hash buckets, with which the capture table is created. This implies that Ingres capture tables have a hash structure. This reduces the chance of locking contention between parallel user sessions writing to the same capture table. It also makes the capture table larger and I/O into it sparser, so it should only be used when such locking contention could occur. Row level locking (default for Oracle and SQL Server and configurable for Ingres) removes this locking contention too without the cost of extra I/O.<br><br>This parameter requires location property **Capture_Method** set to **DB_TRIGGER**. |

| | | |
|---|---|---|
| **HashKey**<br><br>**Ingres** | *col_list* | Identify the list of columns *col_list*, the values of which are used to calculate the hash key value.<br><br>The **default** hash key is the replication key for this table.<br><br>The key specified does not have to be unique; in some cases concurrency is improved by choosing a non-unique key for hashing.<br><br>This parameter requires location property **Capture_Method** set to **DB_TRIGGER**. |
| **DeleteAfterCapture**<br><br>**File/FTP/Sharepoint** | | Delete file after capture, instead of capturing recently changed files.<br><br>If this parameter is defined, then the channel moves files from the location. Without it, the channel copies files if they are new or modified. |
| **Pattern**<br><br>**File/FTP/Sharepoint** | *pattern* | Only capture files whose names match pattern.<br><br>The **default** pattern is **'**/*'** which means search all sub-directories and match all files.<br><br>Possible patterns are:<br><br>• **'*.c'** – Wildcard, for files ending with **.c**. A single asterisk matches all or part of a file name or sub-directory name.<br>• **'**/*txt'** – Recursive Sub-directory Wildcard, to walk through the directory tree, matching files ending with **txt**. A double asterisk matches zero, one or more sub-directories but never matches a file name or part of a sub-directory name.<br>• **'*.lis'** Files ending with **.lis** or **.xml**<br>• **'a?b[d0 9]'** Files with first letter **a**, third letter **b** and fourth letter **d** or a digit. Note that [**a f**] matches characters, which are alphabetically between **a** and **f**. Ranges can be used to escape too; [*****] matches ***** only and [**[**] matches character **[** only.<br>• '***.csv\|*.xml\|*.pdf**' Multiple patterns may be specified. In this case, all csv files, all xml files, all pdf files will be captured.<br>• **{***hvr_tbl_name***}** is only used when data is replicated from structured files to a database with multiple tables. If there are multiple tables in your channel, the capture job needs to determine to which table a file should be replicated and will use the file name for this. In this case, action **Capture** must be defined with parameter **Pattern**. This parameter is not required for channels with only 1 table in them.<br><br>*Example*: In your channel, you have a file **audit.csv** that needs to be replicated to a table called **file_a**, in which column names are the same as in csv file. To do this, the following actions should be defined on the source group - action **Capture** with parameter **Pattern={file_a}.csv** and action **FileFormat** with parameters **Csv** and **HeaderLine.**<br>• **{***hvr_address***}** When a file is matched with this pattern, it is only replicated to integrate locations specified by the matching part of the file name. Locations can be specified as follows:<br>  ◦ An integrate location name, such as **tgt1**.<br>  ◦ A location group name containing integrate locations, such as **TGTGRP**.<br>  ◦ An alias for an integrate location, defined using parameter **AddressSubscribe** in action **Restrict**. For example, **22** or **Alias7**.<br>  ◦ A list of the above, separated by a semicolon, colon or comma, such as **src**, **tgt1**.<br>• **{***name***}** is only used for replicating files between directories ("blob file" or "flat file"). An example of the **{***name***}** pattern is **{abc}.txt**. The value inside the braces is an identifier. Although the 'named pattern' works the same as a wildcard (*****), but it also associates the captured file with a property named **{abc}**. This property can be used in the 'named substitution' (see **Integrate** parameter **RenameExpression**).<br><br>*Example 1*: suppose a channel has capture pattern **{office}.txt** and rename expression **xx_{office}.data**. If file **paris.txt** is matched, then property **{office}** is assigned string value **paris**. This means it is renamed to **xx_paris.data**.<br><br>*Example 2*: suppose the **77-99.pdf** file on source needs to be renamed to **new-77-suff-99.pdf2** on target. In this case, the **Pattern** is **{a}-{b}.pdf**, and define action **Integrate** with parameter **RenameExpression**=**new-{a}-suff-{b}-pdf2**.<br><br>On Unix and Linux, file name matching is case sensitive (e.g. ***.lis** does not match file **FOO.LIS**), but on Windows and SharePoint it is case-insensitive. For FTP and SFTP the case sensitivity depends on the OS on which HVR is running, not the OS of the FTP/SFTP server. |
| **IgnorePattern**<br><br>**File/FTP/Sharepoint** | *pattern* | Ignore files whose names match *pattern*. For example, to ignore all files underneath sub-directory **qqq** specify ignore pattern **qqq/**/***. The rules and valid forms for **IgnorePattern** are the same as for **Pattern**, except that 'named patterns' are not allowed. |
| **IgnoreUnterminated**<br><br>**File/FTP/Sharepoint** | *pattern* | Ignore files whose last line does not match *pattern*. This ensures that incomplete files are not captured. This pattern matching is supported for UTF 8 files but not for UTF 16 file encoding. |

| | | |
|---|---|---|
| **IgnoreSizeChanges** File/FTP/Sharepoint | | Changes in file size during capture is not considered an error when capturing from a file location. |
| **AccessDelay** File/FTP/Sharepoint | *secs* | Delay reading file for *secs* seconds to ensure that writing is complete. HVR will ignore this file until its last create or modify timestamp is more than *secs* seconds old. |
| **UseDirectoryTime** File/FTP/Sharepoint | | When checking the timestamp of a file, check the modify timestamp of the parent directory (and its parent directories), as well as the file's own modify timestamp.<br><br>This can be necessary on Windows when parameter **DeleteAfterCapture** is not defined to detect if a new file has been added by someone moving it into the file location's directory; on Windows file systems moving a file does not change its timestamp. It can also be necessary on Unix/Windows if a sub-directory containing files is moved into the file location directory.<br><br>The disadvantage of this parameter is that when one file is moved into a directory, then all of the files in that directory will be captured again. This parameter cannot be defined with parameter **DeleteAfterCapture** (it is not necessary). |

## Writing Files while HVR is Capturing Files

It is often better to avoid having HVR capture from files while they are still be written. One reason is to prevent HVR replicating an incomplete version of the file to the integrate machine. Another problem is that if parameter **DeleteAfterCapture** is defined, then HVR will attempt to delete the file before it is even finished.

Capture of incomplete files can be avoided by defining **AccessDelay** or **IgnoreUnterminated**.

Another technique is to first write the data into a filename that HVR capture will not match (outside the file location directory or into a file matched with **IgnorePattern**) and then move it when it is ready to a filename that HVR will match. On Windows this last technique only works if **DeleteAfterCapture** is defined, because the file modify timestamp (that HVR capture would otherwise rely on) is not changed by a file move operation.

A group of files can be revealed to HVR capture together by first writing them in sub-directory and then moving the whole sub-directory into the file location's top directory together.

- If column **hvr_op** is not defined, then it default to **1** (insert). Value **0** means delete, and value **2** means update.
- Binary values can be given with the **format** attribute (see example above).
- If the **name** attribute is not supplied for the **<column>** tag, then HVR assumes that the order of the **<column>** tags inside the **<row>** matches the order in the HVR repository tables (column **col_sequence** of the **HVR_COLUMN** repository table).

## Examples

This section includes an example of using the parameter **IgnoreSessionName**.

### Using IgnoreSessionName

HVR allows to run a purge process on an Oracle source location without stopping active replication. Purging is deleting obsolete data from a database. To ensure that the deleted data does not replicate to a target location, the purge process must be started by a database user (e.g. **PurgeAdmin**) other than the user (e.g. **hvruser**) under which the replication process is running, and HVR must be configured to ignore the session name of the **PurgeAdmin**.

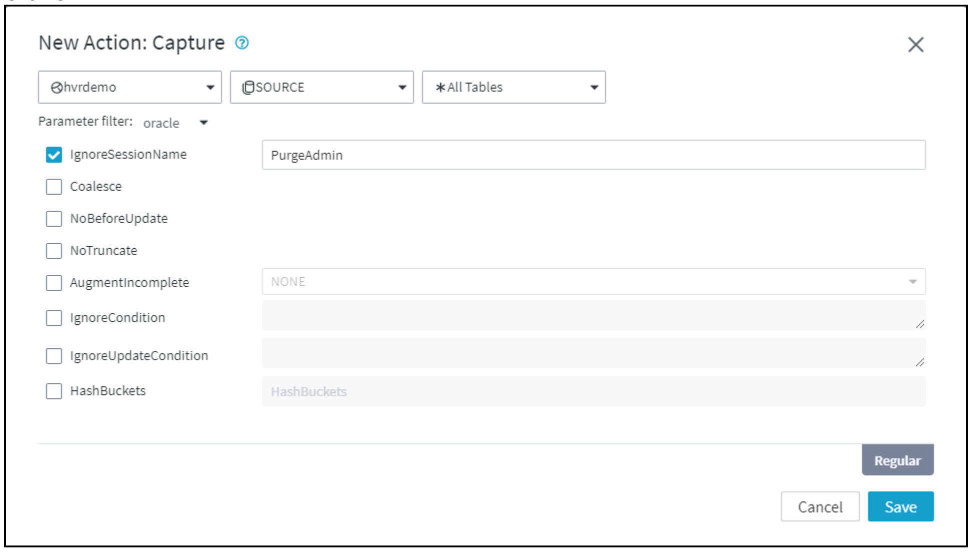
The steps for implementing this scenario are as follows:

1. In a source database, create a new user **PurgeAdmin** that will run a purge script against this database.
2. Grant the applicable permissions to user **PurgeAdmin**, e.g. a privilege to delete rows in another schema:

```
grant delete any table to PurgeAdmin;
```

3. In the UI, update action **Capture** defined on the existing channel by adding parameter **IgnoreSessionName**:
    a. In the **Actions** panel, click on the row containing action **Capture**.
    b. In the **Action: Capture** dialog, select parameter **IgnoreSessionName** and specify the user name **PurgeAdmin**.

c. Click **OK**.



New Action: Capture

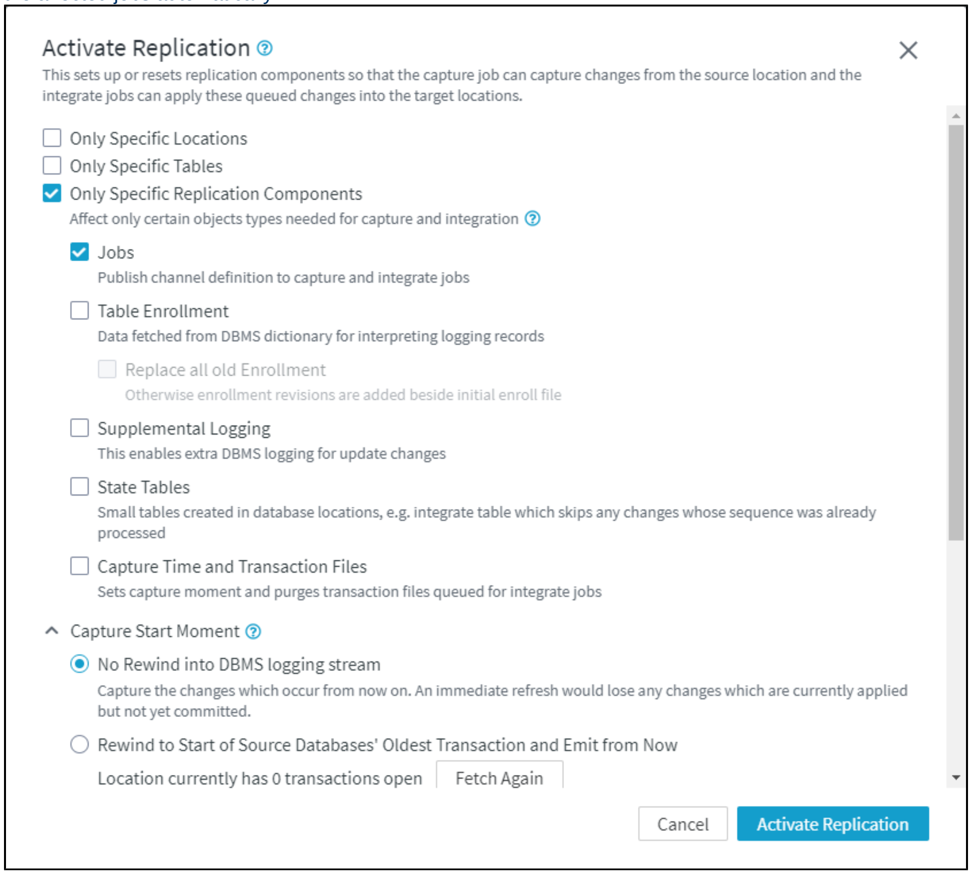
hvrdemo | SOURCE | All Tables

Parameter filter: oracle

- [✓] IgnoreSessionName: PurgeAdmin
- [ ] Coalesce
- [ ] NoBeforeUpdate
- [ ] NoTruncate
- [ ] AugmentIncomplete: NONE
- [ ] IgnoreCondition
- [ ] IgnoreUpdateCondition
- [ ] HashBuckets: HashBuckets

4. Perform **Activate Replication** to re-activate the capture job and apply the changes in action **Capture**:

a. Click **Only Specific Replication Components** and enable only **Jobs**. **Activating Replication** with option **Jobs** will suspend and restart the affected jobs automatically.



Activate Replication

This sets up or resets replication components so that the capture job can capture changes from the source location and the integrate jobs can apply these queued changes into the target locations.

- [ ] Only Specific Locations
- [ ] Only Specific Tables
- [✓] Only Specific Replication Components
  Affect only certain objects types needed for capture and integration
  - [✓] Jobs
    Publish channel definition to capture and integrate jobs
  - [ ] Table Enrollment
    Data fetched from DBMS dictionary for interpreting logging records
    - [ ] Replace all old Enrollment
      Otherwise enrollment revisions are added beside initial enroll file
  - [ ] Supplemental Logging
    This enables extra DBMS logging for update changes
  - [ ] State Tables
    Small tables created in database locations, e.g. integrate table which skips any changes whose sequence was already processed
  - [ ] Capture Time and Transaction Files
    Sets capture moment and purges transaction files queued for integrate jobs

Capture Start Moment

- (●) No Rewind into DBMS logging stream
  Capture the changes which occur from now on. An immediate refresh would lose any changes which are currently applied but not yet committed.
- ( ) Rewind to Start of Source Databases' Oldest Transaction and Emit from Now
  Location currently has 0 transactions open