CRUDhelper documentation

René Verschuren

CRUDhelper

Create|Read|Update|Delete with ease

Date 18 november 2013

Contents

1 About

1.1 Features

With the help of the CRUDhelper you can keep your Qvd files in shape. You can perform create, read, update and delete actions with ease. Incremental loads (and of course full loads) will become very easy now.

All actions are time stamped and affected rows will be counted. This way you can monitor your ETL-process in great detail. The metadata is saved in a batch control file. This file also contains the parameters for the next incremental load.

The CRUDhelper consists of one batch and one or more workflows. A batch starts at the beginning of the loading script. Each reload will get its unique batch id. The batch will execute one or more workflows.

Each workflow consists of a number of tasks:

- Backing up the old Qvd file.
- Counting the number of valid rows in the source.
- Reading the (new or updated) source data.
- Creating new rows.
- Updating existing rows.
- Deleting invalid rows.
- Rebuilding the data if an incremental load fails.
- Storing the data to the Qvd file.

The goal of each workflow is storing data in a Qvd file.
After executing all workflows without any errors the batch will be ended successfully.

1.2 CRUDhelper code prefix

All variables, tables, fields and subroutines used by the CRUDhelper are prefixed with an underscore. This is done to prevent collisions between CRUDhelper and your own code. So, do not use prefixed variables, tables, fields and subroutines in your own code, unless you really want to hook into the CRUDhelper framework.

2 Warning

Each workflow must contain all the following subroutines:

- _GetSourceRows
- _GetSourceStats
- _GetTargetRows
- _RemoveInvalidRows
- _KeepValidRows
- _SetIncrementalLoadParameters
- _SetFullLoadParameters
- _SetNextLoadParameters
- _RunExtensions

If a subroutine is missing or the order has been changed, then the subroutine of the previous workflow will be called. Of course with potentially devastating results!

Subroutines must exist in the specified order, but they can contain zero code. So, the minimum code is:

```
SUB _RunExtensions
ENDSUB;
```

3 Setup batch

3.1 Initial setup

Before you begin you have to setup the CRUDhelper by changing some variables and database connections, see loading script tab CRUDhelper.

3.2 Setup variables

Set the directory for Qvd repository into the variable _vTargetDir.
Example: 'D:\Qvd\'

Set the directory for backing up Qvd files into variable _vTargetBackupDir.
Example 1: 'D:\Qvd\Backup\'
Example 2: 'D:\Qvd\Backup\bk_' (the file name will be prefixed with bk_)

Set the directory for batch control file into variable _vBatchControlDir.
Example: 'D:\Qvd\BatchControl\'

Set the batch control file name into variable _vBatchControlFile. Do not add the extension .qvd to the file name.
Example: 'CRUDhelper_batch_control'.

3.3 Setup database connections

Set the database connection(s) in the subroutine _DatabaseConnection. Add a named connection for any connection you need.

3.4 How to use database connections

Call _DatabaseConnection with the name of the connection as parameter field. The subroutine _DatabaseConnection can be called anywhere from your own code. Calling a not existing named connection (for better clarity use Close) will close the active connection. So, CALL _DatabaseConnection ('Close'); will execute the disconnect command;

4 Setup workflow

4.1 Initial setup

Each new workflow must contain workflow variables, subroutines and a call to _StartWorkflow. Omitting these variables and subroutines or changing the order is a bad thing to do. It is a best practice to put each workflow in its own tab.

4.2 Setup variables

- Set the target file name into variable _vTargetFile. The .qvd extentions will be added by the CRUDhelper.
- Set the parameter datatype into variable _vParameterType. The only valid values are Timestamp, Date, Num or Text.
- Set the backup strategy into variable _vBackupStrategy. The only valid values are Never, Overwrite or Append.
- Set the load strategy into variable _vLoadStrategy. The only valid values are Incremental or Full.
- Set the delete strategy into variable _vDeleteStrategy. The only valid values are TooManyRows, Always or Never.
- Set the rebuild strategy into variable _vRebuildStrategy. The only valid values are AsNeeded or Never.
- Set the indicator drop data table into variable _vDropDataTable. The only valid values are Yes or No.

4.2.1 _vParameterType

- Use Timestamp when start and end parameters consist of dates and time.
- Use Date when start and end parameters are dates only.
- Use Num when start and end parameters are numeric.
- Use Text when start and end parameters contain characters.

4.2.2 _vBackupStrategy

- Use Never to skip backing up the old target.
- Use Overwrite to replace the previous backup file.
- Use Append to keep all previous backup files (warning: can cost a lot of disk space).

4.2.3 _vLoadStrategy

- Use Incremental for loading recently changed data only.
- Use Full for loading all the source rows each reload.

4.2.4 _vDeleteStrategy

- Use TooManyRows if deletion must be performed only when row counts (stats source count vs new target count after create, read and update) does not match.

- Use Always to perform deletions regardless of any row count.
- Use Never to skip deletions.

4.2.5 _vRebuildStrategy

- Use AsNeeded to rebuild the target by performing a full load if the stats count and new target count does not match (something went wrong during the CRUD process).
- Use Never to skip the rebuilding task.

4.2.6 _vDropDataTable

- Use Yes to drop table _tblData after saving the data to the target file.
- Use No to keep _tblData in memory.

4.3 Setup subroutines

Add the following subroutines to the workflow (do not change the order):

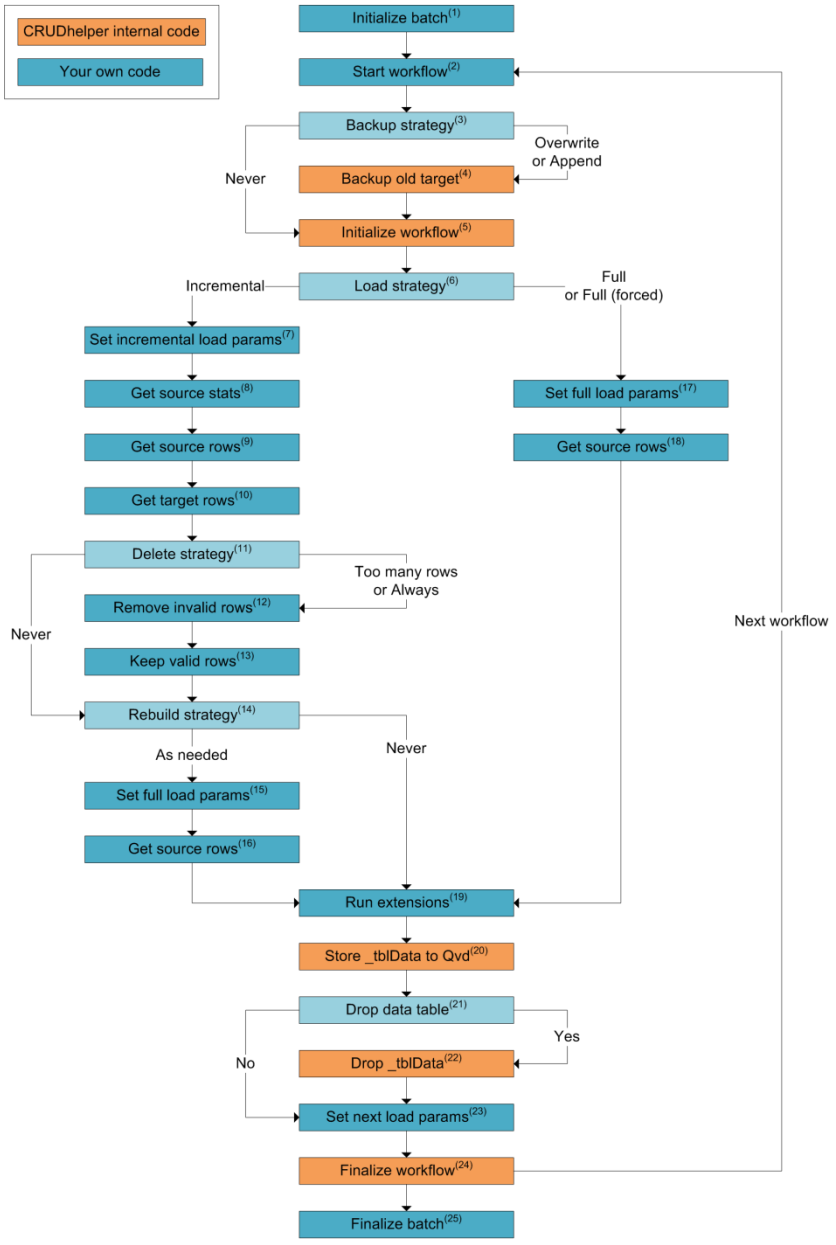
- _GetSourceRows
- _GetSourceStats
- _GetTargetRows
- _RemoveInvalidRows
- _KeepValidRows
- _SetIncrementalLoadParameters
- _SetFullLoadParameters
- _SetNextLoadParameters
- _RunExtensions

4.4 Start workflow

Within each workflow the last row contains the command to start the workflow. This is done by calling subroutine _StartWorkflow from the CRUDhelper.

5 Framework

5.1 Schema



5.2 Framework step by step

5.2.1 Step 1 - Initialize batch

Call _InitializeBatch from your code. In this step the batch control file _vBatchControlFile will be read from _vBatchControlDir. Based on the read data the new batch id will be created.

If the batch control file does not exist, then a new file will be created. In this case all incremental loads are forced to full loads, because the incremental load parameters are unknown.

5.2.2 Step 2 - Start workflow

Call _StartWorkflow for executing the workflow.

5.2.3 Step 3 - Backup strategy

Decide to backup (Overwrite or Append) or not (Never). The decision is based on the variable _vBackupStrategy.

5.2.4 Step 4 - Backup old target

Backup the old target _vTargetFile from _vTargetDir to _vTargetBackupDir. The old target will be read into the table _tblBackup. After saving this table to the backup directory the table will be dropped.

5.2.5 Step 5 - Initialize workflow

Lookup the target file name (_vTargetFile) in the batch control file and get the next start and end parameters from the previous reload.
If variable _vLoadStrategy is Incremental then the target file must exist with a row count of one or more and the next start parameter must have a value (not null).
If this is not the case then the variable _vLoadStrategy will be set to 'Full (forced)' indicating a full load must be performed.

5.2.6 Step 6 - Load strategy

Decide to perform an incremental load (see steps 7 to 16) or a (forced) full load (see steps 17 and 18). The decision is based on the variable _vLoadStrategy.

5.2.7 Step 7 - Set incremental load parameters

This is the first step when performing an incremental load (see step 6).
Subroutine _SetIncrementalLoadParameters assigns a value to the variables _vParameterStart and _vParameterEnd. Setting these variables is mandatory.
The datatype must match variable _vParameterType.
Both variables can be used in steps 8 and 9; getting the row count and the new and updated source data.
During the execution of step 5 (initialize workflow) two variables (_vParameterNextStart and _vParameterNextEnd) got their value based on the previous reload.
Use _vParameterNextStart and _vParameterNextEnd to set the mandatory variables _vParameterStart and _vParameterEnd.

- Example 1: LET _vParameterStart = Date (Floor ('$(_vParameterNextStart)') - 1, 'DD-MM-YYYY'); // Previous start date (no time) minus 1 day.
- Example 2: LET _vParameterEnd = ; // Use this code if you do not need this variable.

5.2.8 Step 8 - Get source stats

The subroutine _GetSourceStats contains the logic to count the total number of valid source rows.
Always load to the table _tblStats.
Use this line of code: NoConcatenate _tblStats:
Assign the row count result to the field _StatsCount.

Only the first loaded row will be used by the CRUDhelper. The valid number of source rows is stored in the variable _vStatsCount.

5.2.9 Step 9 - Get source rows

Subroutine _GetSourceRows reads new and updated rows from the source. Load the source data to the mandatory table _tblData. You are free to load to one of more additional tables.
Use this line of code: NoConcatenate _tblData:
Use variables _vParameterStart and _vParameterEnd to get only new and updated source rows.

- Example Oracle SQL: WHERE last_update_date >= TO_DATE ('$(_vParameterStart)', 'DD-MM-YYYY')

If you need one SQL for incremental loads and another SQL for full loads, then you could work with an IF statement for testing the variable _vLoadStrategy. This variable can hold two user defined values: Incremental and Full. And there are two system defined values: Full (forced) and Full (rebuild).

5.2.10 Step 10 - Get target rows

In step 9 new and updated rows are loaded into the table _tblData. Subroutine _GetTargetRows combines these rows with unchanged rows from the old target file. Combine the rows with this code: Concatenate (_tblData)
Get the unchanged rows from the target with this condition: Where not Exists (primary key).

5.2.11 Step 11 - Delete strategy

Decide to delete invalid rows or not. The decision is based on the variable _vDeleteStrategy. In case of _vDeleteStrategy is Never nothing will happen. Deleting invalid rows is started when _vDeleteStrategy is Always. Or when _vDeleteStrategy is TooManyRows and the _tblData row count is greater than the counted number of valid source rows (see step 8).

5.2.12 Step 12 - Remove invalid rows

Subroutine _RemoveInvalidRows gets invalid primary keys from the data source. These keys must be left joined to _tblData.
Use this code: Left Join (_tblData)
To mark the invalid rows you have to add the mandatory field _RemoveThisInvalidRowFromTable and assign a value to this field. Use 1 as value (or any other value you like), but do not use null.
Supply only the primary key and the field _RemoveThisInvalidRowFromTable. Any additional field will end up in _tblData.
Removing invalid rows is the fastest way to delete rows. But the data source must keep a record of all deleted rows.
If you cannot retrieve deleted rows, then leave this subroutine empty.

5.2.13 Step 13 - Keep valid rows

Subroutine _KeepValidRows is executed if _vDeleteStrategy is Alway or when _vDeleteStrategy is TooManyRows and the _tblData row count is (still) greater than the number of valid source rows (see step 8).
To delete invalid rows you have to retrieve all valid primary keys from the data source and inner join them to _tblData. Do not supply additional fields.
Use this code: Inner Join (_tblData)

Retrieving all valid keys can take a long time. When a table has a lot of rows and many fields, then it could be wise to perform a rebuild (full load). In this case you can leave this subroutine empty.

5.2.14 Step 14 - Rebuild strategy

Decide whether to perform a rebuild (full load) or do nothing. This is based on variable _vRebuildStrategy.
A rebuild is performed when _vRebuildStrategy is AsNeeded and the _tblData row count is not equal to the number of valid source rows (see step 8).

5.2.15 Step 15 - Set full load params

Subroutine _SetFullLoadParameters assigns a value to the variables _vParameterStart and _vParameterEnd. Setting these variables is mandatory.
The datatype must match variable _vParameterType.
Both variables can be used in step 16; getting all the rows from the source data.

- Example 1: LET _vParameterStart = Date (MakeDate (1900, 1, 1), 'DD-MM-YYYY'); // Get all the rows starting from 1 january 1900.
- Example 2: LET _vParameterEnd = ; // Use this code if you do not need this variable.

5.2.16 Step 16 - Get source rows

Subroutine _GetSourceRows should be flexible, so it can be used to perform incremental loads and rebuilds (full load). For more information about _GetSourceRows see step 9.

5.2.17 Step 17 - Set full load params

This is the first step when performing a (forced) full load (see step 6).
Subroutine _SetFullLoadParameters assigns a value to the variables _vParameterStart and _vParameterEnd.
In case of a real full load you can leave this subroutine empty or use the code below:

- LET _vParameterStart = ;
- LET _vParameterEnd = ;

When an incremental load is forced to a full load the SQL in subroutine _GetSourceRows expects the variables _vParameterStart and/or _vParameterEnd.

- Example 1: LET _vParameterStart = Date (MakeDate (1900, 1, 1), 'DD-MM-YYYY'); // Get all the rows starting from 1 january 1900.
- Example 2: LET _vParameterEnd = ; // Use this code if you do not need this variable.

5.2.18 Step 18 - Get source rows

In case of a real full load you use subroutine _GetSourceRows to load all the rows from the source. Load the source data to the mandatory table _tblData. You are free to load to one of more additional tables.
Use this line of code: NoConcatenate _tblData:
In case of a forced full load the subroutine _GetSourceRows should be flexible, so it can be used to perform incremental loads and forced full loads (for more information see step 9).

5.2.19 Step 19 - Run extensions

The incremental or full load tasks (see steps 7 to 18) will affect table _tblData.

With help of subroutine _RunExtensions you can run extra code just before saving _tblData to the target file.
You can perform additional modifications to _tblData or whatever you like to do.

5.2.20 Step 20 - Store _tblData to Qvd

If there are no scripting errors the table _tblData will be saved to the target file.

5.2.21 Step 21 - Drop data table

Decide to drop table _tblData or not. The decision is based on the variable _vDropDataTable.
If you keep _tblData the next workflow must handle the existing table _tblData correctly. Keep subroutine _GetSourceRows (see steps 9, 16 and 18) empty or add a condition (_tblData exists or not).

5.2.22 Step 22 - Drop _tblData

CRUDhelper will drop table _tblData.

5.2.23 Step 23 - Set next load params

With help of subroutine _SetNextLoadParameters the start (_vParameterNextStart) and end (_vParameterNextEnd) parameters can be set by your own business logic.
The next reload will use these parameters. You can use timestamps, dates, numbers or text as parameter values, but the datatype must match variable _vParameterType.

- Example: LET _vParameterNextStart = '$(_vBatchStartedOn)'; // _vWorkflowStartedOn could be a nice alternative.

5.2.24 Step 24 - Finalize workflow

Save the workflow metadata to the batch control file. Go to the next workflow (step 2) or finalize the batch (step 25).

5.2.25 Step 25 - Finalize batch

Call from your own code _FinalizeBatch to end the batch properly. Batch end date will be set and the batch control file is saved. If there are script errors then the batch end date will be null.