


Conformance QC Test Framework

Created by Mike Timchenko

Last updated: Jan 03, 2022 • 5 min read •  15 people viewed

Clusters: **Hemonc, Hemonc-PTD**

Schema: **QC**

Each procedure saves results into the table **QC.TEST_RESULTS**



Column name	Datatype	Description
test_type	varchar(50)	Possible values: foreign key, is not null, unique, allowed increment, window match, data match
test_group	varchar(200)	Name of a group of tests
test_action	varchar(100)	Possible values: stop, warn, email, wiki
test_time	timestamp	Start time of a group of tests
starttime	timestamp	time of the beginning and the end of each test
endtime	timestamp	
order_run	varchar(50)	ID of order run on DK Platform
schema_name	varchar(100)	Key table information. The field column_names is empty after test allowed-increment
table_name	varchar(100)	
column_names	varchar(200)	
skip_errors	int	The test will fail if the amount of test errors is greater then "skip_errors" value. Use 0 if every error is critical
input_parameters	varchar(1000)	Values of additional parameters
result_code	int	1: done/success; 0: wrong data/error; -1: wrong parameters/fail
result_message	varchar(200)	Contains common parameters and message about wrong parameters/data
result_row_count	int	The amount of rows in the table. It is used in allowed-increment

		test
result_error_count	int	The total amount and examples of wrong data
result_error_data	varchar(4000)	

QC Rule-based procedures

Template of procedure names: **proc_qc_<type_of_rule>**

Common parameters:

Parameter name	Datatype	Description
p_test_group	varchar(200)	Name of a group of tests
p_test_action	varchar(100)	Actions: stop, warn, email, wiki
p_test_start	timestamp	{{now}}
p_order_run	varchar(50)	{{CurrentOrderRunId}}
p_schema_name	varchar(100)	Key table information to select and test a set of data
p_table_name	varchar(100)	
p_column_names*	varchar(200)	
p_skip_errors*	int	The number of rows with incorrect data a test will ignore

*The test **allowed-increment** does not use values of the parameter *p_column_names* and *p_skip_errors*.

Not-null test (CALL QC.proc_qc_not_null)

The test checks null-values in the table. This test uses only common parameters.

Uniqueness test (CALL QC.proc_qc_unique_key)

The test checks duplicate values in the table. This test uses only common parameters.

Primary-key test (CALL QC.proc_qc_primary_key)

The test checks presence of null-values and duplicate values. This test uses only common parameters.

Allowed-increment test (CALL QC.proc_qc_allowed_increment)

The test compares the amount of row in the table with the amount of row in the same table from previous test run. Negative value of **p_delta_record_count** allows to check allowed decrease. This test does not use values of the parameter *p_column_names* and *p_skip_errors*.

Parameter name	Datatype	Description

p_delta_record_count	int	Value could be negative, zero or positive
----------------------	-----	---

Window-match test (CALL QC.proc_qc_window_match)

The test compares values of the current period with values from the previous period. This test uses data from the same table.

Parameter name	Datatype	Description
p_window_column	varchar(100)	The column name with time/date datatype that allows to select data for a period
p_window_days	int	The amount of days to compare current and previous periods
p_mode	varchar(20)	<p>"same" - identify different values of datasets;</p> <p>"no-drops" - identify deleted values</p>

Data-match test (CALL QC.proc_qc_data_match)

The test compares values of two tables (p_table_name and p_match_table_name).

Parameter name	Datatype	Description
p_match_schema_name	varchar(100)	Key information about table to match values
p_match_table_name	varchar(100)	
p_match_column_names	varchar(200)	
p_mode	varchar(20)	<p>"same" - identify different values of datasets;</p> <p>"no-drops" - identify deleted values</p> <p>"foreign key" - checks existence of all values in the match-table</p>

Prior-match test (CALL QC.proc_qc_prior_match)

The test compares values of the current table and the table from previous build (usually is stored in <schema_name>_last).

Parameter name	Datatype	Description
p_prior_schema_name	varchar(100)	Key information about table to match values

p_mode	varchar(20)	“same” - identify different values of datasets; “no-drops” - identify deleted values
--------	-------------	---

Value-match test (CALL QC.proc_qc_value_match)

The test compares values in the table with values of the parameter **p_value_set**. Each value should be separated by **comma**. Text and date values should be used with **apostrophes**. In case of several columns, every values set should be separated by **semicolon**. The column *result_error_count* of the table *qc.test_results* stores amount of unique wrong values.

Parameter_name	Datatype	Description
p_value_set	varchar(500)	Set of values
p_mode	varchar(20)	IN or NOT IN

Condition-check test (CALL QC.proc_qc_condition_check)

The test runs a custom’s condition. The condition might include **AND, OR** and subqueries. In case of mistakes in the condition, the test procedure fails, generates an error message and stops the variation. Before release of changes, all mistakes of condition must be eliminated. In order to track data not passed the test, all columns used in condition should be listed in the parameter **p_column_names**.

Parameter_name	Datatype	Description
condition	varchar(1000)	SQL condition

Aggregate-match test (CALL QC.proc_qc_aggregate_match)

The test compares aggregate values of two tables (p_table_name and p_match_table_name).

Parameter name	Datatype	Description
p_groupby_names	varchar(200)	
p_having_condition	varchar(500)	
p_match_schema_name	varchar(100)	Key information about table to match values
p_match_table_name	varchar(100)	
p_match_column_names	varchar(200)	
p_match_groupby_names	varchar(200)	
p_match_having_condition	varchar(500)	
p_mode	varchar(20)	“same” - identify different aggregate function results between two data sets;

		"no-drops" - identify deleted groups and any changes of existing groups "numeric increment" - indicates decrease of aggregate function results (applies only for numeric data)
--	--	---

Custom-query test (CALL QC.proc_qc_custom_query)

The test runs a custom's condition. The condition might include **AND**, **OR** and subqueries. In case of mistakes in the condition, the test procedure fails, generates an error message and stops the variation. Before release of changes, all mistakes of condition must be eliminated. A custom query should return one column with a name "**ERROR_DATA**". In order to return data from several columns, all columns should be concatenated into one column.

Parameter_name	Datatype	Description
query	varchar(4000)	SQL query

Example of usage

1. Create an additional node at the beginning of a variation to save a value of the jinja variable `{{CurrentOrderRunId}}` as a Scalar Result. In this case a value of **LocalOrderRunId** would be accessible at any node if you resume an order run. **Do not include other steps in this node.**

⤵

2. Create one or several steps to call auto-test procedures in a new or existing node. The example of the way to call procedures:

```

call qc.proc_qc_foreign_key('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t', 'raw_car_t_gps_treatments_flat_file', 'record_id',0,
'car_t', 'raw_car_t_gps_process_steps_flat_file', 'treatment');

call qc.proc_qc_primary_key('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t', 'raw_car_t_gps_summary_flat_file', 'id_enrollment_id,record_id,join_id',0);

call qc.proc_qc_allowed_increment('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t', 'raw_car_t_gps_summary_flat_file', 0);

call qc.proc_qc_window_match('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t',raw_shs_demand_car_t,'product,period_unit',0, 'period', 7,'same');

call qc.proc_qc_prior_match('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t', 'ref_ide_cel_target_list_12m', 'treatment_site_id',0, 'car_t_last','same');

call qc.proc_qc_value_match('car_t_auto_qc','warn','{{now}}','{{LocalOrderRunId}}',
'car_t', 'raw_car_t_site_list', 'tier', 0, "'1'", NULL, "", 'in');

```

3. Create a step to get all result messages with result code = -1 and save it into a jinja variable

```

select listagg(test_type||': '||result_message,' ') within group (order by starttime)
from qc.test_results
where order_run = '{{LocalOrderRunId}}'
and result_code = -1;

```

4. Create a step to get all result messages with result code = 0 and save it into a jinja variable

5. Create tests to check rows with result_code = 0 or -1

—

Result of usage

1. Run a variation and open results. It contains messages about all errors and warnings.



2. Use DataGrip to access results and examples of wrong data in the table **QC.TEST_RESULTS**

```
1 select result_code, test_type, result_message, result_error_data
2 from qc.test_results
3 where order_run = '<YourOrderRunId>'
4 and result_code in (-1, 0)
```



 Like Be the first to like this

No labels 

