CA DataMinder

iConsole Review Queue Configuration Guide

Release 14.6

CA Technologies Product References

This document references the following CA Technologies products:

- CA DataMinder™

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 5: Customizations 25

Chapter 1: About iConsole Review Queue Configuration

This guide describes how Database Administrators configure, manage and customize the Review Queue feature. It also includes a 'quick setup' for configuring review queues against Oracle and SQL Server databases.

The Review Queue (RQ) feature enables reviewers to generate lists of events that they need to review or audit. Specifically, it provides an iConsole search that reviewers can run to retrieve all unreviewed events in their review queue (unreviewed events are assigned to reviewers based on their management groups). It also includes various reports that provide administrators with technical information about RQ database searches.

This section contains the following topics:

Review Queue Search and Reports (see page 7)

Review Queue Search and Reports

The Review Queue feature includes the following iConsole search and reports:

Reviewer Search

Retrieves events in a user's personal review queue. These are events waiting to be reviewed.

Admin Reports

These include the following reports:

Review Queue Configuration

Shows the event selection rules for your management groups when a review queue job runs.

Review Queue Diagnostics

Provides details about the most recent review queue run. For each step, it shows the execution time, the actual query statement and the status.

Review Queue History

Shows summary details for previous database review queue runs, including run times, status and event counts.

Note: The Review Queue is not designed for use with Policy security models.

Chapter 2: Deploy the Review Queue

This section contains the following topics:

Requirements

RQ is compatible with Oracle and SQL Server CMS databases. This necessitates a distinct code set for each platform. Although the functionality provided in each case is the same, there are slight architectural differences. In particular, the core product code for Oracle is contained within multiple routines residing in a single stored database package called WGN_RQ, while for SQL Server each routine is an individual stored database procedure or function.

The CMS and iConsole requirements are described in the *Platform Deployment Guide*.

Installation

The required RQ database components are installed automatically when you install or upgrade your CMS.

You install the RQ search and administrative reports when you install the iConsole standard searches and reports. Full installation instructions are in the *Platform Deployment Guide*.

Configuration: Quick Setup

The RQ framework can be customized in many ways. This section focuses on one example of how to deploy the RQ and describes the minimum steps required to get the RQ working.

Custom Metric Definition

The most common method of customizing the RQ is by defining the review metrics. These metrics define how events are sampled in order to be placed onto review queues. Customizing the metric definition is described below.

Note: This customization is not needed if you want to use the default review metric. The default review review metric adds *all* events that triggered policy for *all* reviewers onto review queues.

Advisors and Managers

Example: An organization has two types of users and groups in the system, Advisors and Managers. In the user hierarchy, all Advisors are in the ADV group (or its subgroups) and all Managers are in the MGR group (or its subgroups).

Which Events Are Queued for the ADV Group?

- 5% sample of messages are selected from a pool of violations
- If the resulting number is between 10 and 50, only that number is applicable for review by Advisors
- If the resulting number is more than 50, only 50 messages are applicable for review by Advisors
- If the resulting number is less than 10, then we top it off for a total of 10 messages

Which Events Are Queued for the MGR Group?

- 2% sample of messages are selected from a pool of violations
- No minimum or maximum number is set for this group

Logically, we can define these metrics as follows:

Group	Metric Category	Metric Value	Metric Type	Metric Target	Data Target	Data Grouping
ADV	0	5	0	P	P	-
ADV	1	10	1	-	A	-
ADV	2	50	1	-	-	-
MGR	0	2	0	P	P	

To implement these metrics

Using an appropriate SQL interface such as SQL*Plus or TOAD:

1. Determine the internal GroupIDM and GroupID values for the group. This is done in SQL with the following:
   ```
   SELECT GROUPIDM,GROUPID FROM WGNGROUP WHERE GROUPNAME='ADV'
   ```
2. The stored procedure WGN_RQ_METRIC_INSERT_1 (SQL Server) or WGN_RQ.METRIC_INSERT_1 (Oracle) inserts the metric values into the table Wgn3ReviewMetrics (see step 4).

   Assuming that the ADV group has values 1 and 104 for GroupIDM and GroupID respectively, you must supply these necessary parameters to the procedure:

   ```
   1,104,0,5,0,'P','P'
   1,104,1,10,1,null,'A'
   1,104,2,50,1,null,null
   ```

   Where the ordered parameters to the procedure are:
   ```
   GroupIDM,
   GroupID,
   Category (0,1,2,3),
   Value,
   ValueType (0=percentage, 1=absolute),
   MetricTarget ('A'=All,'P'=Prime,'X'=Excluded),
   DataTarget ('A'=All,'P'=Prime,'X'=Excluded),
   DataGrouping(0 or Null=no grouping, 1=grouping, defaults to NULL if not
   specified),
   Process (defaults to 1 if not specified)
   ```

   Although this customization can issue SQL Insert statements directly against this table, we strongly recommend that you use this stored procedure. Using this stored procedure prevents future schema changes causing customizations to fail.
3. Repeat steps 1 and 2 for MGR group.

   Assuming the MGR group has 1 and 105 for GroupIDM and GroupID respectively, supply these parameters to the procedure:

   ```
   1,105,0,2,0,'P','P'
   ```

4. Create a custom stored procedure. This stored procedure must implement these metrics and deploy them in the database.

   The simplest method to create a custom stored procedure is to edit the CA-supplied customizable package body **WGN_RQ_CUST** as shown below The physical source file is WGN_RQ_CUST_BODY.sql located in the Support\CustomSQLTemplates' directory on the installation image.

   Oracle

```
PROCEDURE Set_Metrics
IS
BEGIN

  -- Default is to insert metric parameters for top-level group only.
  -- All other sub-groups inherit this.

  -- Wgn_RQ.Set_Metrics_Def_1;

  -- To use custom code the default call above must be commented out.
  -- Provide custom code as follows by always using the specific
  -- procedure/function calls where stated:

  -- purge existing metrics
  -- <<MANDATORY>>
  WGN_RQ_EXEC_DDL.Truncate_Table('Wgn3ReviewMetrics');

  -- insert new metrics
  -- <<OPTIONAL>>
  -- WGN_RQ.METRIC_INSERT_1(1,100,0,5,0,'P','P',
  -- <<p_DGrouping>>,<<p_Process>>);
  -- WGN_RQ.METRIC_INSERT_1(1,100,1,10,1,null,'A');
  -- WGN_RQ.METRIC_INSERT_1(1,100,2,50,1,null,null);
  -- WGN_RQ.METRIC_INSERT_1(1,101,0,2,0,'P','P');
  -- WGN_RQ.METRIC_INSERT_1(1,123,0,50,0,'P','P',1,1);
  -- WGN_RQ.METRIC_INSERT_1(1,103,0,75,0,'P','P',
  -- p_DGrouping=>1,p_Process=>1);
  -- WGN_RQ.METRIC_INSERT_1(1,118,3,75,0,'P','P');
  -- ...
  -- ... where <<p_DGrouping>> defaults to null
  -- ...   and <<p_Process>> defaults to 1 if omitted

  WGN_RQ.METRIC_INSERT_1(1,104,0,5,0,'P','P');
  WGN_RQ.METRIC_INSERT_1(1,104,1,10,1,null,'A');
  WGN_RQ.METRIC_INSERT_1(1,104,2,50,1,null,null);
  WGN_RQ.METRIC_INSERT_1(1,105,0,2,0,'P','P');

END Set_Metrics;
```

SQL Server

```
CREATE PROC WGN_RQ_SET_METRICS_CUSTOM_1
AS
BEGIN
      truncate table wgn3reviewmetrics

      EXEC WGN_RQ_METRIC_INSERT_1 1,104,0,5,0,'P','P'
      EXEC WGN_RQ_METRIC_INSERT_1 1,104,1,10,1,null,'A'
      EXEC WGN_RQ_METRIC_INSERT_1 1,104,2,50,1,null,null
      EXEC WGN_RQ_METRIC_INSERT_1 1,105,0,2,0,'P','P'
END
```

More information:

Metric Definitions (see page 19)

Other Customizations

For a quick setup, you may not need any other customizations. However, there are two others that you need to be aware of.

The default implementation of what constitutes queue entry of having a state of 'Closed' is that it has been 'Approved' in the iConsole. The underlying values for what constitutes 'Approved' can vary depending on customer configuration. The default value for this is defined as the event having a current audit record with an Audit Type of 1 and a corresponding value of '1'.

The default implementation assumes that the process that builds the queues will run once daily, and will sample 1 day's worth of events each time it runs.

More information:

Customizations (see page 25)
GET_SAMPLE_SIZE_STUB_1 (see page 29)
WGN_V_RQ_CRITERIA_CL_1 View (see page 26)

Miscellaneous Issues

The default implementation of a 'Closed' queue entry (and therefore removed from a reviewer's queue) is that all the participants that the reviewer manages are associated with an Audit Issue that has a status of 'Closed'. However, the default iConsole behavior is to only associate a new issue with the 'sender' of the event. You can avoid this potential confusion by configuring the iConsole to assign audit issues to 'All managed users'.

The default definition of the Prime (type 'P') data set is 'events that trigger policy'. The default implementation does this by identifying events that have associated triggers in the database. It does not distinguish between capture and control triggers. By default, this can cause all events to be considered as Prime data. You can avoid this potential confusion by configuring policy so that capture triggers are not stored in the database.

Chapter 3: Manage the Review Queue

This section outlines the key interfaces and tables for managing the Review Queue and populating it with events that need to be reviewed.

This section refers to the logical names of the key routines. The actual naming of procedures or functions in SQL Server and Oracle differ. Logical names are prefixed with 'WGN_RQ_' for SQL Server and with 'WGN_RQ.' for Oracle.

This section contains the following topics:

Populate_Queue

This is the main procedure for populating the queue using the customizations and metrics previously outlined. To automate the Review Queue building (to run every 24 hours, for example), you can set up a task to run every 24 hours with a single call to this procedure.

By default, this procedure takes no parameters and the events that are considered candidates for Review Queue are those determined by DERIVE_UPPER_BOUND_STUB_1.

This procedure has three optional arguments: the process, and the upper and lower bound of events to consider for Review Queue. The last two override those determined by DERIVE_UPPER_BOUND_STUB_1.

The first parameter 'process' allows discrete sets of metrics and customizations to be identified and run independently, for example, daily versus monthly metrics. The upper and lower bound parameters are supported mainly for testing purposes; they are not intended for production use because the upper and lower bounds should be automatically derived using the appropriate customizations.

Populate_Queue_From_Date

This procedure is used when initially setting up Review Queue on an existing system that already contains event data that needs to be queued retrospectively as a one off exercise.

It takes as a parameter the date and time from which events are to be queued. It repeatedly calls POPULATE_QUEUE() to queue events from the specified date. The date and time are based on UpdateTimeStamps (that is, database load dates and **not** EventTimeStamps!). The number of events processed per invocation is determined by DERIVE_UPPER_BOUND_STUB_1() and GET_SAMPLE_SIZE_1() respectively.

For Oracle, the date must be specified in the standard format for the target database. This is normally 'DD-MMM-YY' but you can check this by running SELECT sysdate FROM dual;

For example. from SQL*Plus:

```
SQL> select sysdate from dual;
SYSDATE
---------
09-SEP-07

SQL> exec wgn_rq.populate_queue_from_date(1,'09-SEP-07');
```

If you need to specify a time portion for greater control, then specify the parameter as follows:

```
SQL> exec wgn_rq.populate_queue_from_date(1,to_date('09-SEP-07
02:15:30','DD-MON-YYYY HH24:MI:SS'));
```

Populate_Queue_Batch

This procedure is an alternative to POPULATE_QUEUE() and can be scheduled to run automatically using the SQL Server or Oracle job schedulers. The key difference between POPULATE_QUEUE() and this procedure is that this one makes repeated attempts to populate the queue if one or more invocations are missed due to any form of outage; conversely, POPULATE_QUEUE() runs just once.

Clear_Queue

This procedure is a tool to completely clear down the Review Queue, in order to restart. It is intended for use only during initial setup and prototyping.

Important! Never use this procedure on a production system!

Restart_Queue

If the Review Queue process encounters an unexpected failure and has to be restarted, use this procedure to perform the restart. The Review Queue process is self-checkpointing, and will only resume those steps necessary in order to complete.

Abandon_Run

Use this procedure if you need to remove all queue entries for a particular run of Review Queue population. It removes all corresponding Review Queue entries and marks the Run as abandoned.

Wgn3QueueHistory and Wgn3JobHistory

On SQL Server, the Wgn3QueueHistory table records key metrics about each run of the Review Queue process.

On Oracle, the actual data is stored in a generic table called WGN3JobHistory along with other job types. Review Queue entries can be identified as having a JOBTYPE = 1. Wgn3QueueHistory is actually a view that returns only those entries where JOBTYPE =1. This provides compatibility with older versions, where WGN3QueueHistory existed as a table on Oracle.

Note: In this release there is an anomaly between the Oracle and SQL Server implementations, because WGN3QueueHistory is still a table in SQL Server databases.

Wgn3QueueState and Wgn3JobState

This database table records detailed information about each step performed in the last (current) run of the Review Queue process.

Note: If the Review Queue process fails with an error that can be trapped, the error text is written to this table as the failing statement.

On Oracle, the actual data is stored in a generic table called WGN3JobState along with other job types. Review Queue entries can be identified as having a JOBTYPE = 1. On Oracle, Wgn3QueueState is actually a view that returns only those entries where JOBTYPE =1. This provides compatibility with versions where WGN3QueueState existed as a table on Oracle.

Also, WGN3JobState now holds details of each step performed for multiple invocations of the Review Queue, and they can identified as such by the Run column. This is an enhancement to WGNQueueState, which previously only held information for the last run.

Note: In this release there is an anomaly between the Oracle and SQL Server implementations, because WGN3QueueState is still a table in SQL Server databases.

Chapter 4: Metric Definitions

This chapter provides more detail and examples of metric definition.

This section contains the following topics:

Input Data Classification

This section introduces the concept of categorization of the input data. Categorization is crucial to the definition of the metrics.

The process is attempting to select events for review over an externally specified time frame. There will be a prime driver that determines which events are considered highest priority for selection. Although the actual definition of the prime target is customizable, the concept of having a prime target remains. The most common prime target is 'events that have triggered policy'.

To define the metrics, we formally adopt an enumeration of the categories of input data:

Type P

Prime data, as just described. The Prime target.

Type A

All data. The entire set of events being considered over the specified time period. This is fully inclusive of Type P.

Type X

Data that is normally excluded from review. In simple set algebra, the Type A set is the union of Type P and Type X sets.

Note: For all three categories of data, it is implicit that an event which has already been assigned to a queue for a specified group *is excluded from the above sets*. This includes Type A.

Metric Definition

There are four distinct metric categories, defined on a per group basis, with subgroups overriding or inheriting the metric from the parent.

The purpose of the metric is to calculate an absolute number to be applied to the specified **Data Target**. This number is calculated using combination(s) of **Metric Value, Metric Type** and **Metric Target**. See below for attribute definitions.

The following attributes define a metric:

Metric Category

The category of the metric. The allowable values are:

0 Main Selection metric

1 Under Subscription metric

2 Over Subscription metric

3 Mandatory metric

Metric Value

The numeric value of the metric. The allowable values are:

Any positive number

Metric Type

Used to determine if the metric is to be interpreted as an absolute value or percentage. The allowable values are:

0 Percentage

1 Absolute

Metric Target

Used to determine which data set classification is used to calculate the runtime value of the metric. The allowable values are:

A Type A data

P Type P data

X Type X data

Data Target

Used to define which data set the metric is applied to at runtime. The allowable values are:

A Type A data

P Type P data

X Type X data

Data Grouping

Flag to indicate whether the metric should be calculated at data grouping level.

Group Identifier(GroupIDM,GroupID)

The CA DataMinder user group to which the metric is applied.

Process

The Review Queue process number to which the metric belongs, the default is process number 1.

Main Selection Metric Examples

Metric Definition: Example 1

'Retrieve all events that triggered policy.'

- **Metric Category:** 0
- **Metric Value:** 100
- **Metric Type:** 0
- **Metric Target:** P
- **Data Target:** P
- **Data Group:** -

This can be interpreted as 'retrieve n events from the Prime (Type **P**) data, where n is calculated as being **100 percent** of the Prime (Type **P**) data.'

Metric Definition: Example 2

'Retrieve n events that triggered policy, where n is defined as 5% of total events.'

- **Metric Category:** 0
- **Metric Value:** 5
- **Metric Type:** 0
- **Metric Target:** A
- **Data Target:** P
- **Data Group:** -

This can be interpreted as 'retrieve n events from the Prime (Type P) data, where n is calculated as being 5 percent of All (Type A) data.'

Under Subscription Metric Examples

Metric Definition: Example 1

'Ensure at least 5% of all emails are reviewed (aka 'top-up').'

- **Metric Category:** 1
- **Metric Value:** 5
- **Metric Type:** 0
- **Metric Target:** A
- **Data Target:** A
- **Data Group:** -

This can be interpreted as 'retrieve n events from All data (Type A)' where n is defined as 5 percent of All (Type A) data minus the main selection metric.

Metric Definition: Example 2

'Ensure at least 20% of incoming and outgoing events are reviewed.'

- **Metric Category:** 1
- **Metric Value:** 20
- **Metric Type:** 0
- **Metric Target:** A
- **Data Target:** A
- **Data Group:** 1

This can be interpreted as 'retrieve n events from All data (Type A)' where n is defined as 20 percent of All (Type A) data per Data Grouping minus the main selection metric.

Note: This example assumes that customized function is implemented to implement the grouping, which in this instance derives the Direction (Incoming/Outgoing) of the event.

Metric Definition: Example 3

'Ensure a minimum of 10 events are returned.'

- **Metric Category:** 1
- **Metric Value:**10
- **Metric Type:** 1
- **Metric Target:** -
- **Data Target:** A
- **Data Group:** -

This can be interpreted as 'retrieve n events from All data (Type **A**)', where n is defined as **10** minus the main selection metric.

Over Subscription Metric Example

'Ensure a maximum of 50 events that triggered policy are returned.'

- **Metric Category:** 2
- **Metric Value:** 50
- **Metric Type:** 1
- **Metric Target:** -
- **Data Target:** P
- **Data Group:** -

This can be interpreted as 'restrict the Events from the Prime (Type P) data to 50.'

Mandatory Metric Example

'Ensure that a minimum of 1 event per user is returned.'

There are two distinct configuration steps that must to be performed to implement this metric.

1. Define a customized function that derives the grouping value. In this example, the grouping value is a value that uniquely identifies a CA DataMinder user for a particular event participant. This customized function is invoked for each candidate event participant to derive the CA DataMinder user.
2. Define the metric that defines which CA DataMinder groups this mandatory selection '1 per user' metric is to be applied to.

More information:

Customizations (see page 25)

Chapter 5: Customizations

To configure the Review Queue for specific customer configurations there are various elements that can be customized. Not every element has to be customized, and each has a default implementation.

This section covers what each of the customization points are, what the purpose of each is, and what the default is.

This section contains the following topics:

Customizable Views

General

To customize a view, a new view must be created with the same interface as the distributed view but with a name that has an additional suffix of CUST.

For example, to customize this view:

```
WGN_V_RQ_CRITERIA_CLOSED_1,
```

You must create a new view called:

```
WGN_V_RQ_CRITERIA_CLOSED_1_CUST
```

This new view has the same columns defined as the original view:

```
WGN_V_RQ_CRITERIA_CLOSED_1.
```

WGN_V_RQ_CRITERIA_CL_1 View

Purpose

A view that returns event participant references for all participants that meet the criteria for removal from the Review Queue. This view is used to determine which queue entries are to be purged and will no longer be presented for review.

Default

Returns participant references for all events that have an audit issue with current audit entry with audit type of 1 and a value of '1'. This is the default configuration that defines an 'Approved' audit issue.

Note: In the case of this view WGN_V_RQ_CRITIERIA_CL_1, it is necessary to manually update this view to select from the customized version WGN_V_RQ_CRITERIA_CL_1_CUST by simply replacing the reference to WGN_V_RQ_CRITERIA_CL_1_DEF with WGN_V_RQ_CRITERIA_CL_1_CUST.

Note: For SQL Server versions prior to 12.0, the SQL Server version of this view is named WGN_V_RQ_CRITERIA_CLOSED_1, WGN_V_RQ_CRITERIA_CLOSED_1_DEF and WGN_V_RQ_CRITERIA_CLOSED_1_CUST respectively

WGN_V_RQ_EVENT_SLICE_1 View

Purpose

A view that returns a portion of the total events that will be considered as candidates for the Review Queue as part of the current run.

Default

Returns ALL events from the Wgn3RTEventSlice table.

Note: In the case of this view WGN_V_RQ_EVENT_SLICE_1, it is necessary to manually update this view to select from the customized version WGN_V_RQ_EVENT_SLICE_1_CUST by simply replacing the reference to WGN_V_RQ_EVENT_SLICE_1_DEF with WGN_V_RQ_EVENT_SLICE_1_CUST.

Customizable Stored Procedures and Functions

Note: This section refers to the logical names of customizable routines. However, the actual naming of procedures or functions in SQL Server and Oracle differ and will have the logical names prefixed with 'WGN_RQ_' and 'WGN_RQ.' respectively.

General

To customize a stored procedure, a new stored procedure needs to be created, that has the same interface as the distributed stored procedure, but with a name that replaces the characters STUB_x. For:

SQL Server

The suffix STUB_x must be **replaced** with CUSTOM_x, where 'x' is the version number.

Oracle

The STUB_x suffix must be **removed**.

For example in SQL Server, to customize the stored procedure WGN_RQ_SET_METRICS_STUB_1, a new stored procedure must be created called WGN_RQ_SET_METRICS_CUSTOM_1 that has the same interface defined as WGN_RQ_SET_METRICS_STUB_1.

The above applies to functions as well as to stored procedures.

SET_METRICS_STUB_1

Purpose

Allows the metrics that defined on a per group basis how the queues are to be built. These metric are stored in a database table called WGN3REVIEWMETRICS, so for example if there is a group that has a unique identifier of groupidm=1 and groupid=104, and the metrics required for this group are:

Retrieve 5% of triggered events with a minimum of 10 events and a maximum 50 events then the implementation of these metrics would be as follows.

Default

Defines a single metric for all groups to retrieve all triggered events.

SQL Server

```
EXEC WGN_RQ_METRIC_INSERT_1 1,104,0,5,0,'P','P'

EXEC WGN_RQ_METRIC_INSERT_1 1,104,1,10,1,null,'A'

EXEC WGN_RQ_METRIC_INSERT_1 1,104,2,50,1,null,null
```

Oracle

```
EXEC WGN_RQ.METRIC_INSERT_1(1,104,0,5,0,'P','P');

EXEC WGN_RQ.METRIC_INSERT_1(1,104,1,10,1,null,'A');

EXEC WGN_RQ.METRIC_INSERT_1(1,104,2,50,1,null,null);
```

The parameters to METRIC_INSERT_1() correspond to the metric attributes of the same name, including two final optional parameters, DGrouping and Process.

The DGrouping parameter defaults to 'null' which is interpreted as 'no data grouping' applied' setting to a value of '1' means that 'data groupng is applied' if applicable.

The final parameter to METRIC_INSERT_1() which defaults to a value of 1 and corresponds to the process metric attribute. Non default values, define discrete sets of review metrics identified by that number, for example a weekly versus monthly set. To invoke with a non default (1) set of metrics is done by specifying as the process argument to POPULATE_QUEUE().

Note: For SQL Server the process id is also used in determining the names of the customizations below, where the value of <n> is mapped to the process argument supplied to POPULATE_QUEUE:

```
GET_TYPE_A_PRED_CUSTOM_n_1
GET_TYPE_P_PRED_CUSTOM_n_1
GROUPING_MAIN_CUSTOM_n_1
GROUPING_MAND_CUSTOM_n_1
```

For Oracle, these per process customizations are done by passing the Process Id to the routines.

DERIVE_UPPER_BOUND_STUB_1

Purpose

When the Review Queues are populated a range of Events between a lower and upper bound is considered as candidates for the Review Queue. The lower bound is always defined as the upper bound from the previous run (except on the first run or if specific bounds are specified). This procedure allows the derivation of the upper bound to be customized.

Default

Derives the upper bound as being one day's worth of events since the previous run, so the default assumes that the process runs once per day.

GET_SAMPLE_SIZE_STUB_1

Purpose

This function returns the number of DAYS that will be used to determine the upper bound of events. It is important to note that this sample offset is based on UpdateTimeStamp, which is the creation date of the event record in the database. An example of a customization may be a requirement to use a sample size of 1 hour's worth of events. In this case the custom function would merely specify 'RETURN (1/24)'.

Default

1 (DAY)

DAY_ALIGN_STUB

Function Name

The name of this function varies by DBMS.

On Oracle CMSs, the function is:
`GET_DAY_ALIGN_STUB`

On SQL Server CMSs, the function is:
`DAY_ALIGN_STUB`

Purpose

This function acts as a switch, and can be customized to return either true or false. The default to retain compatibility with older versions is to return false. So by default, Review Queue does not operate in 'day align mode'. The sections below illustrates the behavioral difference between Review Queue operating in 'day align mode' and 'non day align mode'.

Default

false/0

On Oracle, this function returns a Boolean value of true or false, so it needs to return true to switch to 'day align mode'.

However, on SQL Server there is no support for Boolean datatypes, so it needs to return a non zero (1 by convention) to switch to 'day align mode'.

Day Align Mode

1. When the Review Queue selects the events that are candidates to be placed onto the queue, it attempts to select every event from the calendar day on which the upper bound of the next run would occur. This means that, by default, if the sample size is set to 1 day then each run of Review Queue will select all the events from the next calendar day as candidates, and not just those that are within the 24 hour window that follows chronologically from the highest event of the previous run. This means that the event window is used to select events from the Review Queue will not 'slip' over time.
2. When there are insufficient events to satisfy the sample size (default 1 day), the Review Queue will simply process those events that are available. It will not abandon without processing.

Non Day Align Mode

1. When the Review Queue selects the events that are candidates to be placed onto the Review Queue, it attempts to select every event that lies within the window defined by the last event processed by the previous run *plus* the sample size. It makes no attempt to align this window on any boundary, and simply selects the sample size as defined by GET_SAMPLE_SIZE_STUB.
2. When there are insufficient events to satisfy the sample size (default 1 day), the Review Queue will ABANDON, and not process any events until there is are sufficient events available to satisfy the sample size. Note that from release 12.0, when this occurs an entry gets written to WGN3QUEUESTATE to indicate that the Review Queue was abandoned due to insufficient events being available to process.

Examples

- Using the default sample size of 1 day, consider that on Day*x* the Review Queue ran and processed events up to and including an event that occurred at 23:45:20pm. When the Review Queue runs on Dayx+1, the highest event that occurred on that day occurred is at 23:58:00pm (slightly more than 24 hrs ahead of the highest from Day*x*), and on Day*x*+1 there also exists an event that occurred at 23:45:20pm (exactly 24 hrs of the highest from Day*x*). If the Review Queue is operating in 'day align mode' then it will process events on Day*x*+1 up to and including the one that occurred at 23:58:00pm. Conversely, in 'non align mode' it will process events on Day*x*+1 only up to and including the one that occurred at 23:45:20pm, but those that occurred between 23:45:20pm and 23:58:00pm will not be processed until the Review Queue runs on Day*x*+2.
- Using the default sample size of 1 day, consider that on Dayx the Review Queue ran and processed events up to and including an event that occurred at 23:45:20pm. When the Review Queue runs on Day*x*+1, the highest event that occurred in the system occurred at 09:00:00am on Day*x*+1 (so there is only around 9:15 hrs worth of candidate events, which is less than the 1 day as required by the sample size). If the Review Queue is operating in 'day align mode' then it will successfully process events on Day*x*+1 up to and including the one that occurred at 09:00:00am. Conversely, in 'non align mode' it will *abandon* and process no events because it insists on processing a full sample of events.

GET_CTAS_LIMIT_STUB_1

Purpose

Part of the queue management process requires that entries be physically removed (purged) from the queue once they meet the criteria that defines they are closed. When purging the queue, it can be more efficient to create a new queue that just contains the 'Open' entries rather than deleting every entry that is 'Closed', if the number of entries to delete is a high percentage of the total. This method allows that that threshold to be customized.

Default

80%

GET_SORT_PRECEDENCE_STUB_1

Purpose

Allows optimization if the order in which the results are to be displayed in the search is more than simple FIFO or LIFO (such as when the triggered events must be displayed before non-triggered events).

Note: This function is **not** relevant in Oracle.

Default

Triggered event are given higher precedence than non-triggered.

GET_TS_LOWER_BOUND_STUB_1

Purpose

Allows an additional restriction to be applied when choosing events as candidates for Review Queue by specifying an additional lower bound on EventTimeStamp. This will help performance on partitioned systems.

SQL Server default

None. No additional filter on EventTimeStamp is added.

Oracle default

In Oracle the EventTimeStamp predicate has already been considered by default to help query performance, particularly where partitioning is present, which should normally be the case. If there are no bounds input to the process or bounds are not available from any previous runs, then the **minimum** EventTimeStamp value is used as a lower bound.

GET_TYPE_A_PRED_STUB_1

Purpose

The Review Queue is built from events between an upper and lower bound (see DERIVE_UPPER_BOUND_STUB_1) within these bounds there are conceptually two distinct sets of data. These are:

Type A

All data. This is the full set of events within the upper and lower bound that are to be considered when building the Review Queue. An example of Type A predicate would be to only include e-mail and IM events but exclude Web events.

Type P

Prime data. This is the prime target for the main selection that drives the queue building.

If this procedure is customized, it must return an SQL subquery that projects the eventuid and EventTimeStamp pairs of events that are considered Type A.

Default

Null. All events within the lower and upper bounds are candidates for review.

SQL Server custom implementation

In SQL Server, the custom function must always be named WGN_RQ_GET_TYPE_A_PRED_CUSTOM_n_1 where 'n' is the RQ Process number for which this function applies. If the name of the custom function(s) deviates from that stipulated above then the RQ process to populate the Review Queue will merely ignore it and subsequently it will fail to implement the intended custom behavior.

Oracle custom implementation

In Oracle, the custom function is contained within the stored package WGN_RQ_CUST and must **always** be named WGN_RQ_CUST.GET_TYPE_A_PRED. Additionally, the process number is passed to this function such that multiple implementations can be coded on a per process basis by using a PL/SQL CASE construct. The custom package body is contained within the supplied file WGN_RQ_BODY.sql and can be modified accordingly.

GET_TYPE_P_PRED_STUB_1

Purpose

See the previous description.

Type P data is the prime target for the main selection that drives the building of the queue. An example of Type P predicate could be to define that the prime target for selection is events that have triggered policy.

If this procedure is customized then it has to return an SQL sub query, that projects the eventuid and eventtimestamp pairs of events that are to be considered Type P data.

Default implementation

Select only those events that have triggered policy.

SQL Server custom implementation

In SQL Server, the custom function must **always** be named WGN_RQ_GET_TYPE_P_PRED_CUSTOM_n_1 where 'n' is the RQ Process number for which this function applies. If the name of the custom function(s) deviates from that stipulated above then the RQ process to populate the Review Queue will merely ignore it and subsequently it will fail to implement the intended custom behavior.

Oracle custom implementation

In Oracle, the custom function is contained within the stored package WGN_RQ_CUST and must **always** be named WGN_RQ_CUST.GET_TYPE_P_PRED. Additionally, the process number is passed to this function such that multiple implementations can be coded on a per process basis by using a PL/SQL CASE construct. The custom package body is contained within the supplied file WGN_RQ_BODY.sql and can be modified accordingly.

VALIDATE_BOUNDS_STUB_1

Purpose

As described earlier, the events that are considered candidates for placing on Review Queue are between a lower and upper bound (see DERIVE_UPPER_BOUND_STUB_1). Due to operational procedure, it may be necessary to determine whether the bounds are valid.

For example there may be an operational constraint that dictates that a minimum of 1000 events must be eligible for selection before the process can run.

Customizing this allows those operational constraints to be enforced.

If the result of this validation proves false, then the Review Queue population is abandoned, and can be viewed as such.

Default

Returns false if the upper bound is greater than the highest event in the system.

MAIN_GROUPING_STUB_1

Note: In SQL Server this function is standalone and named differently as described further down. There is **no** Stub version.

Purpose

Function that returns the user defined grouping to be defined for metric categories 0,1 and 2. Note as the actual type of the grouping is unknown it must be returned as a character string. Note as per definition of metrics, only groups that have the appropriate DGrouping setting of 1 will be subject to this level of grouping. This function is called to attach a user defined value to each event that will allow totals to be computed grouped by that value, examples are 'user' 'direction' 'hour' etc.

The function takes the arguments listed below. These are the respective attributes from the event to which the user defined value is to be attached.

```
eventuid
eventtimestamp
ParticipantIndex
groupidm
groupid
useridm
userid
AddressUID1
AddressUID2
ParticipantNodeUID
InterventionNodeUID
eventmajortype
```

Default Implementation

Returns NULL.

SQL Server custom implementation

In SQL Server, the custom function must **always** be named WGN_RQ_GROUPING_MAIN_CUSTOM_n_1 where 'n' is the RQ Process number for which this function applies. If the name of the custom function(s) deviates from that stipulated above then the RQ process to populate the Review Queue will merely ignore it and subsequently it will fail to implement the intended custom grouping behavior.

Oracle custom implementation

In Oracle, the custom function is contained within the stored package WGN_RQ_CUST and must **always** be named WGN_RQ_CUST.GROUPING_MAIN. Additionally, the process number is passed to this function such that multiple implementations can be coded on a per process basis by using a PL/SQL CASE construct. The custom package body is contained within the supplied file WGN_RQ_BODY.sql and can be modified accordingly.

MAND_GROUPING_STUB_1

Note: In SQL Server this function is standalone and named differently as described further down. There is **no** Stub version.

Purpose

Same purpose as for section MAIN_GROUPING_STUB_1 *(see page 35)* function described previously, but used to derive user defined data grouping for the category 3 metric -- *Mandatory Selection*.

Default implementation

Returns NULL.

SQL Server custom implementation

In SQL Server, the database stored function must always be named WGN_RQ_GROUPING_MAND_CUSTOM_n_1 where 'n' is the RQ Process number for which this function applies. If the name of the custom functions deviates from that stipulated above then the RQ process to populate the Review Queue will merely ignore them and it will subsequently fail to implement the intended custom grouping behavior.

The following code is an example of how to implement such a function for a mandatory selection of '**at least 1 event per user**' and it assumes it applies for the default RQ process "1". It makes the assumption that the input parameter @userid can be guaranteed to uniquely identify a CA DataMinder user. This is an assumption that may or may not be valid depending on how CA DataMinder is deployed. If this is not true, then a combination of @useridm and @userid must be used (as they comprise the formal unique identifier of a CA DataMinder user in the database)> But to retain the simplicity of the example, that assumption is made here.

This function is invoked for each event participant who is a potential candidate for Review Queue. The parameters that are passed are the respective values for each event participant. Because it is generic in nature, it must return a value of type nvarchar. This is why the @userid must be cast.

SQL Server

```
CREATE FUNCTION WGN_RQ_GROUPING_MAND_CUSTOM_1_1 (
@eventuid bigint ,
@eventtimestamp datetime ,
@ParticipantIndex int ,
@groupidm int ,
@groupid int ,
@useridm int ,
@userid int ,
@AddressUID1 bigint ,
@AddressUID2 bigint ,
@ParticipantNodeUID int  ,
@InterventionNodeUID int  ,
@eventmajortype int
)
RETURNS nVarchar(512)
AS
BEGIN
    return cast(@userid as nvarchar(512))
END
```

To implement '**at least 1 event per user**', this requires a metric of category '3' which is the Mandatory selection metric. The interpretation of the metric attributes is identical to the other metric categories, so the example below is interpreted as 'Include a minimum of **1** event per **data grouping** from type **A** data'. However, the metric does not define how the data group is implemented (in this instance it is per user), that is done by the function above. Setting the value of Data Grouping to '1' is merely stating that the metric is to be applied using the defined data grouping, but not how that grouping is implemented.

- **Metric Category:** 3
- **Metric Value:**1
- **Metric Type:** 1
- **Metric Target:** -
- **Data Target:** A
- **Data Group:** 1

Assuming that for the Group (1,105) we wish to apply the above metric then the implementation of that additional metric is shown below:

SQL Server

```
CREATE PROC WGN_RQ_SET_METRICS_CUSTOM_1
AS
BEGIN
      truncate table wgn3reviewmetrics

    /*
    Other metrics from previous example omitted
    */

-- The metric below defines 1 per user for group 1,105
 EXEC WGN_RQ_METRIC_INSERT_1 1,105,3,1,1,null,'A',1,1

END
```

Oracle custom implementation

In Oracle, the custom function is contained within the stored package WGN_RQ_CUST and must **always** be named WGN_RQ_CUST.GROUPING_MAND. Additionally, the process number is passed to this function such that multiple implementations can be coded on a per process basis by using a PL/SQL CASE construct. The custom package body is contained within the supplied file WGN_RQ_BODY.sql and can be modified accordingly.

The following code is an example of how to implement such a function for a mandatory selection of '**at least 1 event per user**' and it assumes it applies for the default RQ process '1'. It makes the assumption that the input parameter p_UserID can be guaranteed to uniquely identify a CA DataMinder user. This is an assumption that may or may not be valid, depending on how CA DataMinder is deployed. (If this were not true then a combination of p_UserIDM and p_UserID would need to be used because they comprise the formal unique identifier of a CA DataMinder user in the database, but to retain the simplicity of the example that assumption is made here.)

This function is invoked for each event participant who is a potential candidate for the Review Queue, and the parameters that are passed are the respective values for each event participant. Because it is generic, it has to return a value of type VARCHAR2, hence the reason why the p_UserID has to be cast.

Oracle

```
FUNCTION Grouping_Mand   ( p_EventUID            IN NUMBER,
                           p_EventTimeStamp      IN DATE,
                           p_ParticipantIndex    IN NUMBER,
                           p_GroupIDM            IN NUMBER,
                           p_GroupID             IN NUMBER,
                           p_UserIDM             IN NUMBER,
                           p_UserID              IN NUMBER,
                           p_AddressUID1         IN NUMBER,
                           p_AddressUID2         IN NUMBER,
                           p_ParticipantNodeUID  IN NUMBER,
                           p_InterventionNodeUID IN NUMBER,
                           p_EventMajorType      IN NUMBER,
                           p_Process             IN NUMBER DEFAULT 1 )
RETURN VARCHAR2
IS
BEGIN

  -- This procedure is customizable per process and therefore the following
  -- constructs must be strictly adhered to.

  CASE p_Process

     WHEN 1 THEN

       RETURN TO_CHAR(p_UserID);

     WHEN 2 THEN

       -- To use custom code the default call below must be commented out
       -- and additional custom code provided in it's place.

       -- invoke default routine
       RETURN
WGN_RQ.Grouping_Mand_Def_1(p_EventUID,p_EventTimeStamp,p_ParticipantIndex,
                       p_GroupIDM,p_GroupID,p_UserIDM,p_UserID,
                       p_AddressUID1,p_AddressUID2,
p_ParticipantNodeUID,p_InterventionNodeUID,p_EventMajorType);

     ELSE

       -- DO NOT CUSTOMIZE

       -- invoke default routine
       RETURN
WGN_RQ.Grouping_Mand_Def_1(p_EventUID,p_EventTimeStamp,p_ParticipantIndex,
                       p_GroupIDM,p_GroupID,p_UserIDM,p_UserID,
                       p_AddressUID1,p_AddressUID2,
p_ParticipantNodeUID,p_InterventionNodeUID,p_EventMajorType);
```

```
  END CASE;

END Grouping_Mand;
```

To implement '**at least 1 event per user**' requires a metric of category '3' which is the Mandatory selection metric. The interpretation of the metric attributes is identical to the other metric categories, so the example below is interpreted as 'Include a minimum of **1** event per **data grouping** from type **A** data'. However the metric does not define how the data group is implemented (in this instance it is per user), that is done by the function above. Setting the value of Data Grouping to '1' is merely stating that the metric is to be applied using the defined data grouping, but not how that grouping is implemented.

- **Metric Category:** 3
- **Metric Value:** 1
- **Metric Type:** 1
- **Metric Target:** -
- **Data Target:** A
- **Data Group:** 1

Assuming that for the Group (1,105) we wish to apply the above metric then the implementation of that additional metric is shown below:

Oracle

```
PROCEDURE Set_Metrics
IS
BEGIN

  -- Default is to insert metric parameters for top-level group only.
  -- All other sub-groups inherit this.

  -- Wgn_RQ.Set_Metrics_Def_1;

  -- To use custom code the default call above must be commented out.
  -- Provide custom code as follows by always using the specific
  -- procedure/function calls where stated:

  -- purge existing metrics
  -- <<MANDATORY>>
  WGN_RQ_EXEC_DDL.Truncate_Table('Wgn3ReviewMetrics');

  -- insert new metrics
  -- <<OPTIONAL>>

 -- Other metrics from previous example omitted

-- The metric below defines 1 per user for group 1,105
 WGN_RQ.METRIC_INSERT_1(1,105,3,1,1,null,'A',1,1);

END Set_Metrics;
```

More information:

MAIN_GROUPING_STUB_1 (see page 35)