Reporting and Analytics Aggregates User's Guide

How Do I Customize Queries and Hierarchies?

5/3/2025

How Do I Customize Queries and Hierarchies?

Contents

This page demonstrates how to customize the existing Reporting and Analytics Aggregates (RAA) queries and hierarchies to:

- Replace the definitions of existing measures with custom definitions.
- Aggregate additional measures.
- Define other joins to Info Mart tables.
- Add a `WHERE` qualification to RAA queries.
- Rename the aggregation tables.
- Define the aggregation levels that RAA will aggregate.
- Define which aggregation levels should be tables and which should be views within Info Mart.

Building RAA Queries

The RAA aggregation archive (`GIMAgg.jar`) includes several data-definition files that create the aggregation queries—language files that define the identifiers and macros that are used, templates, and other supporting files—all of which have been created using the Scheme programming language (a high-level dialect of the Lisp programming language). To customize aggregation, you must have a clear understanding of Scheme:

Using Scheme to Build RAA Queries

Understand that editing the files in the aggregation archive is not a supported feature. Nonetheless, this page provides some fundamental information on how the RAA queries are written. From there, you can extrapolate how to tailor them—at your own discretion—to better meet a specific business need.

Out of the box, RAA includes the following query families:

- AGENT
- AGENT_GRP
- AGENT_CAMPAIGN
- AGENT_QUEUE
- BGS_SESSION
- CALLBACK
- CAMPAIGN
- CHAT_STATS
- I_AGENT
- I_SESS_STATE
- I_STATE_RSN
- ID
- QUEUE
- QUEUE_GRP
- QUEUE_ABN
- QUEUE_ACC_AGENT
- SDR_SESS_MILESTONE
- SDR_SESSION
- SDR_SESS_BLOCK
- SDR_SURVEY
- SDR_SURVEY_ANS
- ID_FCR

These query families correspond directly to the hierarchies that are described in Disposition Measure Hierarchies, and share similar names. (Hierarchies, however, are all prefaced with “H_”.)

Internal coding for each query family further differentiates family members along the seven time intervals (see Aggregation Intervals).

To modify a query's family, you should be familiar, at minimum, with the query's top-level definition, which is stored in a `select-*.ss` Scheme file. Depending on the release of RAA installed in your environment, `select-*.ss` is found either inside the aggregation Java archive, or in the `\agg\lib\meta*.jar` file. There is one such file for each query family; for example:

- `select-AGENT.ss` stores the top-level definition for the AGENT query.
- `select-AGENT_CAMPAIGN.ss` stores the top-level definition for the AGENT_CAMPAIGN query.

These files reference macros that are defined in other, lower levels of query definition that reside in language and other ss files. RAA queries are complex.

Customize Aggregation Using the Patch-Aggregation File

For aggregation that operates in integrated mode, you can customize aggregation by creating / editing a Patch-Aggregation file (`patch-agg.ss`), which you must place in the Genesys Info Mart folder in which the `gim_etl_server` batch script is located. For aggregation that operates in autonomous mode, place the `patch-agg.ss` file in the Genesys Info Mart working directory of the java process that runs standalone aggregation—the Genesys Info Mart root directory.

See the following sections for more information:

- Loading the Patch
- Editing the patch-agg.ss file
- Format of the patch-agg.ss file

Loading the patch-agg.ss file

The following procedure describes the steps you must follow to load the patch-agg.ss file.

Procedure: Loading the Patch

Purpose: Use this procedure to load the patch.

Steps

1. Create (or update) the patch file—**patch-agg.ss**— and specify the desired changes to one or more queries. Create the file either in ASCII format, or UTF8, as follows:
   - **UTF-8 format** — UTF-8 specific characters are defined in `patch_agg.ss` (or in another custom schema file). Create or update the file in UTF-8 format and define JVM option `-Dfile.encoding=UTF-8` during Genesys Info Mart start up, as follows:
     - When running RAA in a standalone mode, execute the following command:{

       ```
       java -Dfile.encoding=UTF-8 -jar GIMAgg.jar
       ```

       OR
     - When running with Genesys Info Mart, follow the instructions for modifying the JVM Startup Parameters in the Genesys Info Mart Deployment Guide when adding the `-Dfile.encoding=UTF-8` option, and then start Genesys Info Mart and RAA.
   - **ASCII format** — use the syntax described in Format of the patch-agg.ss file, below.
2. Place the patch file in the Genesys Info Mart root directory.
3. Restart aggregation. (The aggregation engine reads the patch file only upon start.)
4. Use Genesys Info Mart Manager to verify that the aggregation process is running successfully. Alternatively, check for the following message in the Genesys Info Mart log:

   ```
   Agg.Main     processing: patch-agg.ss
   ```

   > **Tip**
   > The aggregation process will not run at all if `patch-agg.ss` contains errors.
5. For multi-tenant environments, run an update of tenant aliases as discussed in What Do I need to Know About Managing Multi-Tenant Environments?.

Editing the patch-agg.ss file

In the `patch-agg.ss` file, you specify the query modifications that will override or add to the existing Genesys-provided query definitions. Use the instructions in this patch file to add, customize, or correct metric definitions.

- **Sample Patch** The following code in the `patch-agg.ss` file provides one example that updates the ID query. This example relies on a contact-center configuration that attaches agent cost to each interaction and records this information in the AGENT_COST column of the IRF_USER_DATA_GEN_1 table. AGENT_COST is a custom attached-data measure that you must configure before you can add it

to the aggregation query.

```
(alter-query ID
          (add                                    (~metric AGENT_COST
 "max(irfug.AGENT_COST)"))
          (replace                                    (~metric INVITE
 "sum(irf.CUSTOMER_RING_COUNT)"))
          (add                                    (inner-join IRF_USER_DATA_GEN_1 irfug
                                                      "irf.INTERACTION_RESOURCE_ID =
 irfug.INTERACTION_RESOURCE_ID"))
 )
```

This code accomplishes the following:

- Adds the AGENT_COST metric to the ID query.
- Replaces the definition of the existing INVITE metric to a new definition that tallies ringing interactions only (instead of ringing and dialing interactions).
- Adds a join to a user-defined, attached-data Info Mart table that is named IRF_USER_DATA_GEN_1 (from which the AGENT_COST metric is sourced in this example).

When this file is loaded, the aggregation engine will follow these instructions and regularly populate the new and changed columns in the AGT_ID_* group of tables and views, as notifications of newly transformed data are sent to the RAA internal queue.

- **Special Runtime Parameters for Customization** The parameters that are listed in the Table *Special Runtime Parameters for Customization and Debugging* are not described elsewhere within the RAA documentation. They are provided to aid in the construction and debugging of your customized aggregation queries.

Special Runtime Parameters for Customization and Debugging

Runtime Parameter	Description
allowDestructiveDDL	Specifies whether the aggregation engine can use destructive data-definition language (DDL)—such as drop table—to delete and recreate database objects as needed. This parameter uses a date range to limit RAA issuance of destructive operations to two days before and after the date that you specify with this parameter. You must specify the date value in YYYY-MM-DD format. For example: `java -jar ./agg/GIMAgg.jar -user=<GIMDBUser> -pass=<passwd> -jdbcurl=<GIMDBURL> -allowDestructiveDDL=2013-07-31` Dropping and recreating aggregation tables would be necessary, for instance, if you customized a hierarchy to add new dimensions in a database created with release 8.1.103 or earlier. Setting this option is unnecessary for standard aggregation operations.

checkQuery	Validates the specified query against Info Mart. The query can be a Genesys-provided query or a custom query. For example: `java -jar ./agg/GIMAgg.jar -user=<GIMDBUser> -pass=<passwd> -jdbcurl=<GIMDBURL> -checkQuery AGENT_GRP`
evaluateFile	Evaluates the specified Scheme file within the context of aggregation, enabling you to perform debugging without the use of any tools and without connection to Info Mart. For example: `java -jar ./agg/GIMAgg.jar -evaluateFile my-Agg.ss` Tip You can also use Scheme `printf` and `display` statements to output results for debugging purposes.

- **Limitations of the patch-agg File** By following this format, you can add additional alter-query statements to the patch file. If you include more than one alter-query statement for the same query, make sure that their instructions do not conflict; otherwise, RAA will use the last statement's definition. Also, you can include as many add and replace statements as needed within an alter-query statement to attain the desired level of query modification. Lastly, at this time, you *cannot* use this file to accomplish any of the following:
  - Remove metrics from the query (although you can replace their definition with a constant, such as "0", or build a new query altogether).
  - Restrict changes only to a particular member of a query family. For instance, you cannot specify to update the query definition for the AGT_I_SESS_STATE_SUBHR table without also simultaneously affecting the query definition for all other family members (AGT_I_SESS_STATE_HOUR and AGT_I_SESS_STATE_DAY).
  - Add another join to the DATE_TIME dimension; only one join is permitted and this one join is already used.

Format of the patch-agg.ss file

The commands that you write in the `patch-agg.ss` file must obey syntax rules and be written by

using the identifiers that RAA recognizes. This section provides syntax for the following types of customizations, including examples for each type:

Adding measures to query definitions — Syntax

```
(alter-query queryName1
     (add     (~metric mNameX (mDefX)))
     (add     (~metric mNameY “expr1“ (expr2) “expr3“))
     ...
)

(alter-query queryName2
     ... )
```

where:

- queryName is the name of any RAA-defined query.
- mNameX and mNameY are the names of measures that you want to add to the query definition.
- mDefX and mDefY (defined in the preceding syntax by the concatenation of three expressions: expr1, expr2, and expr3) are the measures’ definitions, written in the SQL format, and bound by double quotation marks or parentheses. In this example, expr2 is a call to a procedure that returns a string value.

> **Tip**
> You can also use macros and functions as part of the SQL definition; however, this release does not support their use for customization purposes.

Example

```
(alter-query ID
     (add     (~metric OTHER_COST (max(ext.COLUMN))))
)
```

- **Joining other tables to those within the query definition — Syntax**

```
(alter-query queryName
     (add joinType TableX aliasX (joinCondition1)))
     (add joinType TableY aliasY "expr1“ (expr2) “expr3“)
)
```

Where:

- `joinType` describes the type of table join—either of the following:
  - `inner-join`
  - `left-outer-join`
- `TableX` and `TableY` are the names of the tables that are to be joined. These tables could be custom tables. For performance reasons, Genesys recommends that they exist within Info Mart.
- `aliasX` and `aliasY` are the aliases for TableX and TableY, respectively

> **Tip**
>
> You must provide an alias. This alias must differ from the reserved aliases that are used within the `select-*.ss` files if you are adding a new table join to the query or defining a second (or third) join to a table that already exists within the query.

- `joinCondition` describes how two tables are joined, written in SQL format and bound by double quotation marks or parenthesis.

Example

```
(alter-query ID
      (add (inner-join IRF_USER_DATA_GEN_1 irfug
         "irf.INTERACTION_RESOURCE_ID = irfug.INTERACTION_RESOURCE_ID"))
 )
```

> **Tip**
>
> The irf alias that is used in this example is already defined in the `select-ID.ss` file.

- **Adding WHERE qualifications to queries — Syntax**

```
(alter-query queryName
      (add-predicate ( whereQualification ))
 )
```

Where:

- `whereQualification` is the expression to be added to the WHERE clause of the query. Place the expression in double quotation marks to code advanced SQL in free form.

You cannot delete or modify expressions in the WHERE clause using add-predicate.

Example

```
(alter-query ID
     (add-predicate (itf.tenant_key=1))
)
```

- **Altering the definitions of measures within queries — Syntax**

```
(alter-query queryName
        (replace (~metric mName1 “mDefX”))
        (replace (~metric mName2 “expr1“ (expr2) “expr3“))
        ...
)
```

Example

```
(alter-query ID
     (replace (~metric AGENT_COST "sum(case when " (is-agent?) " then
              irf.CUSTOMER_HANDLE_DURATION*r_.AGENT_HOURLY_RATE else 0 end)"))
)
```

- **Editing joins within queries — Syntax**

```
(alter-query queryName
        (replace joinType Table alias (joinCondition))
        (replace joinType Table alias "joinCondition"))
        ...
)
```

Example

```
(alter-query CAMPAIGN
     (replace (left-outer-join IRF_USER_DATA_KEYS irfud
         (irf.INTERACTION_RESOURCE_ID = irfud.INTERACTION_RESOURCE_ID
          and irf.START_DATE_TIME_KEY = irfud.START_DATE_TIME_KEY)))
)
```

- **Adding dimensions to queries — Syntax** Exercise caution when you add new dimensions to an existing aggregation hierarchy that is populated. To keep the hierarchy’s data homogeneous, you must delete and reaggregate all of the data for all aggregation levels of the hierarchy, thereby ensuring that existing records are dimensioned by the new addition. RAA requires that you specify special-permission within the Scheme code to perform this destructive operation. In addition, consider setting the allow **DestructiveDDL** runtime parameter (described in Special Runtime Parameters for Customization) to `true`.
  The syntax for adding dimensions to a query is:

```
(special-permission ‘add-dimension)
...
(alter-query queryName
     (add     (~dimension dimNameX))
     (add     (~dimension dimNameY Tbl.Col))
        ...
     ...
)
```

where:

- `dimNameX` and `dimNameY` are two dimensions that you want to add to the query definition.
- `Tbl.Col` are the table (or alias) and column, respectively, that hold the dimension. If the name of the dimension and its schema location are identical, you do not have to include `Tbl.Col`.
- `TableX` and `TableY` are the names of the tables that are to be joined. These can be custom tables. For performance reasons, Genesys recommends that you enter names of tables that exist within Info Mart.

Example

```
(special-permission ‘add-dimension)
...
(alter-query AGENT
     (add     (~dimension r_.CREATE_AUDIT_KEY))
     (add     (~dimension r_.Updated  “r_.UPDATE_AUDIT_KEY”))
)
```

- **Modifying hierarchies — Syntax**
  The following syntax provides only some of the identifiers that are used with the alter-hierarchy command:

```
(alter-hierarchy hierName1
     (agg-levels:  AgLv1 AgLv2 .. AgLvX)
     (materialize: AgTb1 AgTb2 .. AgTbX)
     (query:        baseQuery)
     (template:    “CustomText”))

(alter-hierarchy hierName2
     ... )
```

where:

- `hierName` is the name of the aggregation hierarchy that you want to alter. For information about the Genesys-provided hierarchies, see What is an Aggregation Hierarchy?.
- `AgLv1 AgLv2 .. AgLvX` are the levels that you want RAA to aggregate. For example, HOUR, DAY, MONTH.
- `AgTb1 AgTb2 .. AgTbX` are those aggregation levels that RAA will materialize as table objects in Info Mart, instead of views. For example, `(materialize: HOUR DAY)` indicates that only the HOUR

and DAY levels will exist as tables. All other levels will exist as views. You can also specify none.

Not all combinations of views and tables will work within the materialize statement. Within a hierarchy, table creation cannot be dependent on a view; instead, table creation must be based directly from data that is pulled from FACT tables. Otherwise dependencies (not covered in this document) must also be updated. So, for instance, the following alter-hierarchy statement will not yield results, if this is the only change that is made to Scheme files:

```
(alter-hierarchy H_I_SESS_STATE
     (agg-levels:   SUBHOUR, HOUR, DAY)
     (materialize: HOUR DAY)
```

For the interval-based hierarchies, hour results are derived from subhour results. In this example, the SUBHOUR aggregation level is defined to exist as a view to the detriment of this customization example.

- baseQuery defines what entity the hierarchy is based on—either data from an Info Mart FACT table (in which case you specify fact) or the name of a defined query. The H_AGENT_GRP hierarchy, for example, is based on the AGENT query. CustomText defines the template by which RAA names the aggregation tables and views. Genesys recommends that this template include %QUERY% and %LEVEL% somewhere within the custom text. For example: KJM_%QUERY%_%LEVEL%_72099

Example

Views typically have slower performance than tables. Perhaps you would prefer that the subhour views within Info Mart be tables. The following example makes the subhour entity for the H_AGENT hierarchy a table that contains subhour aggregations. This coding also changes the template by which aggregation entities are named and reduces the number of aggregation levels processed. The subhour table that RAA creates and populates, for example, is AG3_AGENT_SUBHR_TEST, instead of AG2_AGENT_SUBHR. Notice also that WEEK is omitted from agg-levels.

```
(alter-hierarchy H_AGENT
 (agg-levels: SUBHOUR HOUR DAY MONTH QUARTER YEAR)
      (materialize: SUBHOUR HOUR DAY MONTH)
      (template:    “AG3_%QUERY%_%LEVEL%_TEST”)
 )
```

- **Adding SQL strings to Scheme aggregate definition — Syntax** The following syntax examples illustrate how you can use the Scheme macro `inline-sql` to add, drop, or replace specific SQL statements in Scheme aggregate definitions.
  - To specify an aggregator Scheme file description without alias:
    `(inline-sql <SQL statement>)`
  - To specify an aggregator Scheme file description with alias:
    `(inline-sql left-outer-join-alias <SQL statement>)`
  - To specify a drop statement in the patch-agg.ss Scheme patch file (if you've specified an alias):
    `(drop (inline-sql left-outer-join-alias))`
  - To specify a replace statement in the patch-agg.ss Scheme patch file (if you've specified an alias):
    `(replace (inline-sql left-outer-join-alias <SQL statement>))`

Examples

```
(inline-sql "inner join media_type mt2 on mt2.MEDIA_TYPE_KEY = irf.MEDIA_TYPE_KEY")
 (inline-sql testtenant "inner join tenant t2 on t2.tenant_key = irf.tenant_key")
```

```
(inline-sql media3 "inner join media_type mt3 on mt3.MEDIA_TYPE_KEY =
irf.MEDIA_TYPE_KEY")
(inline-sql tenant3 "inner join tenant t3 on t3.tenant_key = irf.tenant_key")
```

```
(alter-query MYAGENT
        (drop (inline-sql media3))
        (replace (inline-sql tenant3 "inner join tenant t3 on t3.tenant_key =
irf.tenant_key and t3.tenant_key > 0"))
```

Important

General Syntax Note: Any commas within the measure definitions or join conditions must be escaped by using a backward slash (\) or bound by double quotation marks (”). When more than one element makes up the expression, use double quotation marks. Single quotation marks (’) must be bound by double quotation marks. For example: `it.INTERACTION_SUBTYPE_CODE in ('INTERNALCOLLABORATIONREPLY','INBOUNDCOLLABORATIONREPLY')` can be presented as any of the following in Scheme code:

- `it.INTERACTION_SUBTYPE_CODE in "('INTERNALCOLLABORATIONREPLY', 'INBOUNDCOLLABORATIONREPLY')"`
- `it.INTERACTION_SUBTYPE_CODE in (\'INTERNALCOLLABORATIONREPLY\'\, \'INBOUNDCOLLABORATIONREPLY\')`
- `it.INTERACTION_SUBTYPE_CODE in ("'INTERNALCOLLABORATIONREPLY'"\, "'INBOUNDCOLLABORATIONREPLY'`