

Default Ratings

Table of Contents

- Document Info..... 2
- Overview..... 3
 - Rating Scheme recap..... 3
- Challenges..... 4
 - Constraint Relationship 4
 - Ordering..... 4
 - Constraint Modification 4
- Proposal 6
 - Core Waltz changes 6
 - (Optional) Client-site batch job changes..... 6
- Appendices..... 7
 - Appendix A: Moving overall investment status to assessments..... 7
 - Appendix B: Original ask from Architects 9

Document Info

Attribute	Value
Status	DRAFT v0.2 (2020-08-14)
Target Version	1.27
Lead	Dave Watkins

Overview

This design doc discusses approaches to implementing default / constraining value in Waltz. The driver behind this enhancement is to ensure data consistency in Waltz. For example if an app is rated as '*Disinvest*' then it should have should no individual features rated as '*Invest*'.

Rating Scheme recap

The table below shows a typical rating scheme, ordered by the `position` database column (ascending, not shown):

Table 1. Example Rating Scheme

Rating	Code	Notes
Invest	G	-
Maintain	A	-
Disinvest	R	-
Unknown	Z	Not user selectable
Not Applicable	X	This is a waltz special code which means the X is treated as a removal

NOTE

Usage of special character X to indicate deletion should probably be examined to see if it is still needed. Suspect it no longer serves a required purpose, especially with formal app-fn decommms.

Challenges

This section outlines some challenges we face in implementing a solution for this feature.

- **Constraint Relationship**, how do we tell Waltz that there is a constraining relationship between a *measurable rating* and some other value in Waltz (for example the app's overall *investment rating*) ?
- **Ordering**, other than a position value (used for display positioning), there is no ordering semantics within a *rating scheme*. Therefore, how do we know how values are restricted (i.e. if the constraint is 'Maintain' it means we cannot set a rating to 'Invest') ?
- **Constraint Modification**, if the constraining value is altered, how does this impact existing *measurable ratings* ?

Constraint Relationship

Each app needs to (optionally) have a constraining rating per *rating scheme* / *measurable category*. This may require a new table. An alternative may be to reuse assessments and link a *measurable category* to a constraining *assessment rating*. In this case the assessment and category would need to share a common *rating scheme*. We assume that if an application has no constraining value then users are free to assign any *measurable ratings* to that application.

NOTE

we need to be careful about using the overall investment rating as a constraint since we do not have a clean way of referencing it, unlike `assessment_definitions`. However moving overall rating to become an *assessment* may be problematic due to it's widespread usage in exporters etc. Possible solutions are discussed in the section: 'Moving overall investment status to assessments' below.

Ordering

Easiest to simply use position for now and say a rating is '*more restrictive*' if appears later in the list (has a higher position value).

Table 2. Constraints, moving **down** the table is more restrictive, **up** is less restrictive

Constraint Rating	Permissible Measurable Ratings
Invest	Invest, Maintain, Disinvest
Maintain	Maintain, Disinvest
Disinvest	Disinvest

Constraint Modification

The constraining value may not be static. It may be altered by user interaction, or via a periodic batch job. When the values are changed we have two fundamental choices:

- **Hard limit**, the new constraining value becomes a *hard* upper limit for current measurable

ratings.

- If the new constraint is more restrictive any current *measurable rating* which breaches that limit will be reset to the new limit
- If the new constraint is less restrictive the more restrictive *measurable ratings* are left as is.
- **Soft limit**, the new constraining value becomes a *soft* upper limit. Breaches are flagged but not auto corrected.
- **Ratchet limit**, the new constraining value becomes a *hard* upper limit when editing *measurable ratings*. However breaches are **not** auto corrected, but **are** flagged instead.

NOTE

I think the *ratchet limit* is the most useful of the options presented above. *Hard limits* can be enforced via the batch loaders.

Proposal

Core Waltz changes

- ☐ Duplicate Investment status as a read-only assessment detail
- ☐ Update `measurable_category` to have an (optional) ref to a constraining assessment
- ☐ Implement ratchet limit

(Optional) Client-site batch job changes

- ☐ Update importers to populate new assessment
- ☐ Implement *hard limit* logic based on the assessments

Appendices

Appendix A: Moving overall investment status to assessments

Approaches to moving the fixed overall investment status app field into a more generic assessment rating:

- Duplicate investment status as an *assessment rating*
- Remove investment status from application and *only* have it as an *assessment rating*

If we remove investment status from application we will need to rework the exporters (and importers). This needs prototyping (see below for initial vendor spikes) but does give us a very useful set of data to include in our exports (e.g. all assessments for an application).

IMPORTANT

moving investment status will impact batch jobs as inserts/updates will fail unless modified.

Example exporters using pivot functions

Database vendors have differing ways of supporting pivoting. The examples given both use `cross join` to produce a *sparse* result set ensuring each application has rows for each `assessment_definition`, regardless of any provided values.

Postgres

This example utilises the `crosstab` function, which must be enabled using: `CREATE EXTENSION tablefunc;`

Postgres example

```
select a.name, f.*
from application a
inner join (
  select *
  from crosstab('
    select app.id, ad.name, rsi.name
    from application app
    cross join assessment_definition ad
    left join assessment_rating ar on ar.assessment_definition_id = ad.id and app.id =
ar.entity_id
    left join rating_scheme_item rsi on ar.rating_id = rsi.id
    where ad.entity_kind = 'APPLICATION'
    order by app.id, ad.name ')
AS assessment_rating(entity_id bigint, c1 varchar, c2 varchar, c3 varchar, c4 varchar
)) f on f.entity_id = a.id
```

NOTE

The Postgres example needs to have a fixed ordering of assessments and a matching (hard-coded) column list for the output. This should be fairly straightforward to implement using jOOQ.

MSSQL Server

This example utilises the `pivot` function.

MSSQL example

```
select a.name, p.[Sensitive Data], p.[Sanctions Apply]
from application a
inner join (
    select id, [Sensitive Data], [Sanctions Apply] from
        (select app.id, ad.name as adname, rsi.name as rsiname
         from application app
         cross join assessment_definition ad
         left join assessment_rating ar on ar.assessment_definition_id = ad.id and
         app.id = ar.entity_id
         left join rating_scheme_item rsi on ar.rating_id = rsi.id
         where ad.entity_kind = 'APPLICATION') unpvt
        pivot
            (max (rsiname)
             for adname in ( [Sensitive Data], [Sanctions Apply]))
         ) as pvt
) p on p.id = a.id;
```

NOTE

The demo data used in this example has two assessment definitions defined: 'Sensitive Data' and 'Sanctions Apply'. Similar to the Postgres example these columns need to be specified at query construction time.

MySQL example to be completed later.

Appendix B: Original ask from Architects

Below is a (lightly edited) copy of the request. It is included here as it provides the driving rationale behind the design options outlined in the main sections above.

Invest Status:

- If *App Investment Status* is **Invest**, Architect can update *Waltz* function status to **Maintain** or **Disinvest** as an override.
- If *App Investment Status* is **Maintain**, Architect can update *Waltz* function status to **Disinvest** as an override.
- If *App Investment Status* is **Disinvest** or **Decom**, Architect cannot override status in *Waltz*.

Nightly job

- If previous App Investment Status equals current WALTZ status, then no overrides exist and App Investment status is synced to Waltz.
- If Previous App Investment status is not equal to current WALTZ status, then override exists:
 - If updated App Investment status is more restrictive than Waltz, the override is removed and App Investment status is synced to Waltz from now onwards:
 - If updated App Investment status is less restrictive than Waltz, the override is kept and App Investment status is NOT synced to Waltz. (invest does not replace Maintain/Disinvest and Maintain does not replace Disinvest)