The new Report Security functionality is primarily configured through the addition of XML attributes to elements in the Fieldmapper XML file, fm_IDL.xml. These new attributes fall into three categories:

- Field value redaction
- Core class row restriction
- Joined class row restriction

These attributes, explained below, are defined within a new XML namespace with the URI **`http://open-ils.org/spec/opensrf/IDL/reporter/v1/security`**, and the common namespace prefix of **`repsec`**. An accompanying XSD is provided to confirm that the locations and values of the attributes, when supplied, do not fall outside their defined scope and that the fm_IDL.xml file remains both well-formed and valid.

Field value redaction

Fields used in the SELECT and ORDER BY SQL clauses can be redacted, either outputting NULL or, optionally, a data type compatible alternate literal value. This is configured through the addition of several XML attributes to the fields to which redaction should be applied.

XML Attributes involved:

- repsec:redact and repsec:redact_default
    - Type: XML Schema boolean
    - Valid values: true or false
- repsec:redact_with and repsec:redact_with_default
    - Type: string
    - Valid values: any string literal that can be cast to the same datatype as the field naturally holds
- repsec:redact_skip_function and repsec:redact_skip_function_default
    - Type: string
    - Valid values: a fully-qualified database function that returns a boolean value. When the function returns TRUE the redaction of the field will not be performed, otherwise the field will be redacted.
- repsec:redact_skip_function_parameters and repsec:redact_skip_function_parameters_default
    - Type: string
    - Valid values: a colon-separated list of parameters to pass to the function named in the repsec:redact_skip_function. Each parameter in the list can have one of the following three forms
        - The string **`$runner`**, which will cause the id of the staff user that scheduled the run of the report to be passed to the function.
        - The name of a field defined for the class, which will cause the value of that field from the row being tested for field redaction to be passed to the function.

    - Any other string that does not contain a colon (:) character, which will be passed as a dollar-quoted literal to the function. There is no provision for escaping an embedded colon, so special consideration must be given to the string literals that need to be passed into the function, and function may need to perform text manipulation in order to interpret some other string as a colon before operating on the string as passed.

Field values can be marked for redaction in one of two ways: adding the **repsec:redact** attribute with a boolean value of **TRUE** directly to the **<field>** element of an IDL class; or by adding a **repsec:redact_default** attribute of **TRUE** to the **<fields>** container element for all fields of an IDL class.

Any default set on the **<fields>** element by applying a **repsec:redact_default** attribute can be overridden on individual **<field>** elements by applying the **repsec:redact** attribute. If not supplied at all, the default is to NOT redact a field. When generating SQL, redaction settings are only tested at the field level, so each field will have its own "calculated redact attribute" which will be truthy or falsy.

If the "calculated redact attribute" is falsy, all other redaction-related attributes are ignored for the field.

A **repsec:redact_skip_function** attribute on the **<field>** element, or **repsec:redact_skip_function_default** on the **<fields>** element, is not required for redaction. The contents of the **repsec:redact_skip_function_default** on the **<fields>** element is used as the default for all fields that do not carry their own **repsec:redact_skip_function** attribute.

The **repsec:redact_skip_function_parameters** and **repsec:redact_skip_function_parameters_default** values are interpreted as described above, and are used to pass row level, reporting staff, and global contextual information to the function used to decide if the redaction request should be skipped or completed for each marked field.

Finally, all fields are redacted with an SQL **NULL** value by default. Supplying a **repsec:redact_with** attribute on the **<field>** element, or a **repsec:redact_with_default** on the **<fields>** container element, allows the administrator to supply a non-**NULL** replacement value for the database column contents. This value must be a string literal that, when cast, is type-compatible with the column's natural data type. For instance, an INT column cannot be redacted with a string that does not consist entirely of numeric characters, and once cast to an INT must not over- or under-flow the INT data type's boundaries.

Two new, stock functions are supplied for use as **redact_skip_function** values:

- evergreen.direct_opt_in_check

        - Purpose: confirm that the patron either has a home library at which the staff has at least one of the permissions from the Required Permission List, or that the patron has opted in at of those libraries.
        - Parameters
            - Patron ID
            - Staff ID
            - Required Permission List
        - Returns TRUE if the patron is opt-in visible to the staff member, and FALSE otherwise.
        - Example:

```
<class id="au" ...>
  <fields ...
    repsec:redact_default="true"
    repsec:redact_skip_function_default="evergreen.direct_opt_in_check"
    repsec:redact_skip_function_parameters_default="id:$runner:{VIEW_USER}">
      <field name="id" ... repsec:redact="false"/>
        ...
  </fields>
</class>
```

- evergreen.hint_opt_in_check
    - Purpose: Indirectly confirm that a patron related to a row from another linked table either has a home library at which the staff has at least one of the permissions from the Required Permission List, or that the patron has opted in at of those libraries.
    - Parameters
        - Patron ID
        - Staff ID
        - Required Permission List
    - Applicable classes:
        - aua (table: actor.usr_address)
        - auact (table: actor.usr_activity)
        - aus (table: actor.usr_setting
        - actscecm (table: actor.stat_cat_entry_usr_map)
        - ateo (table: action_trigger.event_output)
    - Returns TRUE if the patron is opt-in visible to the staff member, and FALSE otherwise.
    - Example:

```
<class id="ateo" ...>
  <fields ...
    repsec:redact_skip_function_default="evergreen.hint_opt_in_check"
    repsec:redact_skip_function_parameters_default="ateo:id:$runner:{VIEW_USER}">
      <field name="data" ... repsec:redact="true"/>
        ...
  </fields>
```

```
</class>
```

In addition to these new functions, the preexisting stock permission test functions, such as **permission.usr_has_perm** and **permission.usr_has_work_perm** can be used as **redact_skip_function** values. These can be used in this way:

```
<class id="circ" ...>
 <fields ...
 repsec:redact_skip_function_default="permission.usr_has_work_perm"
 repsec:redact_skip_function_parameters_default="$runner:VIEW_CIRCULATIONS:circ_lib">
   <field name="due_date" ... repsec:redact="true"/>
        ...
 </fields>
</class>
```

Core class row restriction

Two new attributes are available for defining row-level restrictions on the core class of a report template:

- repsec:restriction_function
- repsec:restriction_function_parameters

From a configuration perspective, these work in exactly the same way as the **repsec:redact_skip_function** and **repsec:redact_skip_function_parameters** attributes described above, but are applied to the **<class>** element instead of **<field>**. Indeed, the two stock functions described above, **evergreen.direct_opt_in_check** and **evergreen.hint_opt_in_check**, can be used as restriction functions.

When added, these functions generate a WHERE-clause condition that, when TRUE, allow a tested row to be included in the report output, all else being equal. This attribute pair is only applicable to the core class of a report.

For example:

```
<class id="au" ...
  repsec:restriction_function="evergreen.direct_opt_in_check"
  repsec:restriction_function_parameters="id:$runner:{VIEW_USER}">
```

will cause any report that uses the au class as the core source to add a WHERE-clause condition that restricts the inclusion of rows from actor.usr in the final report output by applying the **evergreen.direct_opt_in_check** function.

Joined class row restriction

Two additional new attributes are available to define JOIN-clause restrictions that should be applied to each row in order to include the tested table row in the final report output:

- repsec:projection_function
- repsec:projection_function_parameters

As with the core class restrictions, the above-described functions can be used for this purpose.

There are two locations for these attributes, and where they are attached defines whether they will be applied to the relevant underlying table when it is on the left side of a join, or the right side.

Put another way, an administrator can decide if restrictions are added when a table is joined from, joined to, or both.

To restrict access to rows of a table being joined to -- that is, when a "child relation" is joined into a report -- the attributes are applied to the **<class>** element. For example:

```
<class id="actscecm" ...
  repsec:projection_function="evergreen.direct_opt_in_check"
  repsec:projection_function_parameters="target_usr:$runner:{VIEW_USER}">
```

will cause any report that links from the au class to the **actscecm** class to restrict the visibility of rows from **actscecm** by applying the **evergreen.direct_opt_in_check** function.

If the projection function attributes are instead applied to the **<link>** element, the restriction will be applied only if the link is followed in the report. This allows a report creator to build a template that does not limit output when it does not link through to tables that should be restricted, while enforcing the desired visibility limits when a joined table should be restricted due to relationship with the linking table, even if that joined table is not itself restricted at the **<class>** level. This may be the case when a table contains classifying data that is useful and safe when queried by itself in aggregate, but provides a channel for leaking personal information when joined to, within a report, from another user-centered table.

For a somewhat contrived example, consider this configuration that allows core and linked use of ancihu (Non-cataloged In House Use) while redacting the **staff** field based on the **VIEW_USER** permission, but restricts rows based on the opt-in visibility of the staff that recorded the in-house use based on the **VIEW_CIRC** permission of the report-running staff member if that field is linked in the template:

```
<class id="ancihu" ...>
       ...
  <field
```

```
    name="staff"
    repsec:redact="true"
    repsec:redact_skip_function="evergreen.direct_opt_in_check"
    repsec:redact_skip_function_parameters="staff:$runner:{VIEW_USER}"/>
       ...
  <link ...
    field="staff"
    repsec:projection_function="evergreen.direct_opt_in_check"
    repsec:projection_function_parameters="staff:$runner:{VIEW_CIRC}"/>
```

NOTE: Any **<class>**-level restrictions on the linked table, in this case the **actor.usr** table, will also be enforced.

As with Field Value Redaction described above, the existing stock permission functions can be used in all cases for both Core class and Join class row restriction if there is a relevant Org Unit column available.