Uni5 Query Builder

# Overview

The Query Builder (hereby called as QB) is a new feature of the HivePro Uni5 application. It allows the user to create custom queries for viewing specific information of assets, threats, tickets and other “entities” managed by the system.

The creation of a query in QB is based on Entities, which are the main information objects of the HivePro application, and their fields, which are called Projections in the context of QB. A query lists several Projections from one or more Entities that the user has selected, including any custom conditions to filter the results, and it is run to retrieve the result set of the selected Projections in an order specified by the user.

The QB features the Visual Builder, an interface for selecting the list of Entities and Projections and for creating the custom conditions which are called Rules for filtering the results. A Rule may be as simple as comparing against value or checking for a field not being null, but it can combine more than one Rules with logical operators to form a Group Rule which applies a complex logic.

The user may run a custom query that they have created in QB, see the results, and save it for future use. They may also edit a saved query or use it as a base for another query. Several characteristics like Category and Tags can be assigned to the query. Except from the queries created by the user, there are also “canned queries”, which are predefined and offered as default by the system.

When a query is executed and the result set is shown, the user may export it to a csv file, or in the format of a pdf file report. There is also a set of default charts that highlight specific information drawn from the result set, and the user may also create their own charts.

# Structured Query

A Structured Query is a Java class for modelling a query in QB. It is the central object of the QB application, as all functionalities use it in some way. The members that belong to a Structured Query object are inspired by the structure of a SQL query. For example,

**select** assets.id, assets.name, assets.ip\_address, assets.description, assets.criticality, tickets.id, tickets.name, tickets.assigned\_to, tickets.priority

**from** assets

**left outer join** tickets on tickets.asset\_id = assets.id

**where** assets.criticality = ‘CRITICAL’ **and** tickets.ticket\_patch\_status = ‘FAILED’

**order by** assets.id

**limit** 100

This way, the members of a Structured Query are:

* **projections** : the list of Projections that are shown in the result set
* **base entity** : the main Entity
* **joined entities** : the secondary Entities that are used in the result set and/or the rule
* **rule** : the object that models the where condition
* **order by** : one or Projections that are used to sort the result set
* **limit** : the maximum number of rows in the result set

The user will have to fill in the above by combining Entities and Projections with the help of the Visual Builder.

# Definitions of building elements

## QB Views

Specific database tables and columns have been selected as meaningful to be used in the context of QB. A database view is created for every entity, to gather the associated columns under a single name; the prefix “qb\_” is used for every view name. After a view has been created, it can be used in SQL queries just like a table, and no additional data are created.

## Entities and Projections

A query in QB is built around entities and projections that correspond to views – tables, and columns in the database respectively. For the sake of abstraction, the names of tables and columns are not shown in the user, but the entities and projections are used to build the query instead. The QB translates entities into views and projections into columns when a query is to be executed, using the mappings table, which is described later.

## Rules and Parameters

A Structured Query contains a Rule object that models a WHERE condition. A Rule might be as simple as a single condition on a field, but it can combine more than one field conditions with logical operators, thus forming a Group Rule.

A field that can be used in a condition is called a Parameter; it is one of the Projections whose value or content provides meaningful information. In other words, a Parameter is a Projection (and it may appear on the selected Projections in the result set), but not all Projections are Parameters. The mappings table contains a set of operators that can be applied on the Parameter, and some other columns with information on how the Parameter can be used in a Rule.

A Rule object can be one of the following.

* Field Operation Rule: A condition on a field. It contains the Entity and Field names, a operator and a value.
* Not Operation Rule: A Rule whose only member is another Rule, and it reverses the logic result. It is equivalent to the NOT( ) operator.
* Group Rule: A Rule which contains multiple objects, them being Rules joined with logical operators OR / AND. It is equivalent to a parenthesis. A valid Group Rule is in fact a list, which must have an odd number of members; the ones in the odd places must be Rules, and the ones in the even places must be “OR” or “AND”.

## Mappings

The mappings table contains all necessary information on Entities and Projections – Parameters, and how to use them to create Structured Queries. Each row in this table contains a mapping, which is a correspondence between a Projection name and the actual column name of a view, including the correspondence between the Entity and the view that it belongs to. The table contains additional fields, the full list of which is:

* column\_name
* projection\_name
* view\_name
* entity\_name
* input\_data\_type
  + a string to note the data type of the column. Typical values are Number, String, Timestamp, Boolean. Columns that are of array type must have the “\_Array” suffix.
* ordinal\_position
  + An arbitrary order of the Projection in the Entity it belongs to. It is useful to the UI, as it may automatically pre-select the top projections of an Entity.
* is\_id
  + true only for the ID field of the Entity. This can be checked when building the query, to make sure that the ID field of the base Entity (“from X”) exists in the selected projections.
* is\_parameter
  + is true for parameters
* range
  + For Parameters only: an optional field for keeping the range of values that this column should contain. Examples are
  + RANGE:0,10 (suitable for numeric range with lowest and highest values)
  + DISTINCT:AMBER,RED,WHITE,GREEN (for creating a drop down list of all possible values)
* Operators
  + For Parameters only: A list of the suitable operators that a Rule may apply on the Parameter.
* allow\_non\_suggested\_input
  + For Parameters only: Useful to the UI. If true, the UI may allow the user to enter any input value to be compared in the Rule.
* is\_queryable
  + For Parameters only: Useful to the UI. If true, a list of the distinct values of this column may be retrieved by running a relevant query.
* is\_hidden
  + If true, the Projection/Parameter is not available to be selected when a user is building a query. However, it may still be used in queries for any means necessary.
* is\_deprecated
  + If true, the Projection/Parameter is no longer valid due to other changes in the database which may have removed or altered the corresponding column. It is recommended not to use it in queries anymore, as it may return invalid results.

## Joined Entities

Many Entities contain columns that are pointing to other Entities. These columns can be used in queries to join them and to provide some common information in the result set.

As part of the mappings defined in the database, there is another table that contains the possible joins between Entities. Using these mappings, the application can show additional options for Projections/Parameters from Entities other than the main Entity selected.

The Joined Entities mappings table has the following columns:

* j\_ent\_str
  + the primary key of the table, a string that contains the names of the entities separated by underscore.
* column\_a and view\_a
  + that correspond to Entity A
* column\_b and view\_b
  + that correspond to Entity B
* join\_expression
  + an additional field that shows the relationship between the columns or the entities; specific annotations are used in the case of
    - one-to-many relationship, eg. *qb\_actors.id = any(qb\_alerts.actor\_ids)*
    - many-to-many relationship, which is noted by *array2array*
    - An extra special annotation is made when the join is only possible via a third Entity (indirect join), but it is desirable to have the join noted as it is commonly used. Eg. *via:qb\_vulnerabilities*.

Note: The selection of an Entity to be placed as A or B does not matter, and there is no need to have the reverse entry where the entities A-B are interchanged, even in the case where the relationship is one to many. Just make sure that the expression is like the example above, where the column of array type is on the right-hand side of the expression inside the any operator.

## Maintenance of Mappings Tables

The Project team is responsible for keeping track of all the mappings, including the joined entities, in the shared document.

<https://docs.google.com/spreadsheets/d/1Y24Mq3c6mqSZ1r8nfU22GIrU78ic9LyB/edit?usp=sharing&ouid=101993893717846806744&rtpof=true&sd=true>

With any change requested and/or reported, this document should be updated.

As for updating the mapping tables, their contents are handled by an R\_ script in the cloud\_synced\_data resource directory. For any change that alters the table itself, a new V\_ script should be created in the same directory.

# GetMappingsService

This is the backbone of the query builder services as it is used in almost all of them, from creating queries in the Visual Builder, validating them, and converting names for the actual SQL script to be executed. Since the mappings do not change while the HivePro application is running, they are loaded once from the database, and they are stored in ConcurrentHashMap objects that are private members of the service. Then they are retrieved without the need to query the database.