# NOMENCLATURE

cube – a data structure with edges

edge – defines how the data will be grouped

domain – every edge has a domain which defines it’s valid values

partition – every domain is partitioned into parts, internally this is an ordered array of mutually exclusive parts.

part – one part of a partition

part objects - Partitions are often an array of objects (with a name, value, and other attributes). These objects usually represent the values along the axis of a chart.

cell – a unique tuple representing one part from each edge

value -

object -

attribute -

record/row – anaglous to a database row. In the case of a cube, there is one record for every cell: which is an object with having the attributes of the

column – anagolous to a database column: a common attribute definition found on all objects in a table

# Facets, Edges, GroupBy and Joins

ES facets a simple group-by operation; without allowing group-by on multiple attributes, and only aggregating attributes from the root document.

The ES facet is distinctly different from an edge

A facet collapses all but one dimension, it can not handle more than one dimension

Edges are a convenient mix of join and group-by. Technically edges are an outer join (<http://en.wikipedia.org/wiki/Relational_algebra#Full_outer_join>) combined with an aggregate. The motivation behind outer joins is to ensure both sets are covered in the result. Furthermore, aggregates are limited to aggregating records once and only once.

There are cases, when dealing with normalized data, and in ETL situations, where counting a record more than once, or not at all, is preferred. In this case, use the test property along with allowNulls=false to get the effect of an inner join.

# ORDER OF OPERATIONS

Each of the clauses are executed in a particular order, irrespective of their order in the JSON structure. This is most limiting in the case of the **where** clause. Use sub queries to get around this limitation for now.

**from** – the array, or list, to operate on. Can also be the results of a query, or an in-lined subquery.

**edges** – definition of the edge names and their domains

**where** – early in the processing to limit rows and aggregation: has access to domain names

**select** – additional aggregate columns added

**analytic** – analytic columns added

**sort** – run at end, but only if output to a list.

# QUERY STRUCTURE

Queries are in a JSON structure which can be interpreted by ESQuery.js (for ES requests, limited by ES’s functionality) and by CUBE.js (for local processing with Javascript).

# from

The table, index, or relation that is being processed by the query. In Javascript this can be an array of objects, a cube, or an in-lined query. In the case of ES, this is the name of the index being scanned. Nested ES documents can be pulled by using a dots (.) as a path separator to nested property. Example: Patches are pulled from the BZ

**{**

**"**from":"bugs.attachments**",**

**"where": {"term":{"bugs.attachments[\"attachments.ispatch\"]":"1"}}**

**}**

Example: Pull review requests from BZ:

**{**

**"from":"bugs.attachments.flags",**

**"where": {"term" : {"bugs.attachments.flags.request\_status" : "?"}}**

**}**

ESQuery.js can pull individual nested documents from ES. ES on it’s own can only return a document once. Aggregation over nested documents is not supported.

# select

Can be a single attribute definition, or an array of attribute definitions. The former will result in nameless value in each data element of the resulting cube. The latter will result in an object, with given attributes, in each data element

**name** – The name of the attribute. Optional if value is a simple variable name.

**value** – Code to generate the attribute value (MVEL for ES, Javascript otherwise)

**aggregate** – one of many aggregate operations

**none** – when expecting only one value

**one** – when expecting all values to be identical

**binary** – returns 1 if value found, 0 for no value

**exists** – same as binary

**count** – count number of values

**sum** – mathematical summation of values

**average** – mathematical average of values

* **select.default** to replace null in the event there is no data

**minimum** – return minimum value observed

**maximum** – return maximum value observed

**percentile** – return given percentile

* **select.percentile** defined from 0.0 to 1.0 (required)
* **select.default** to replace null in the event there is no data

**join** – concatenate all values to a single string (optional **select.separator** to use a separator)

**sort** – one of “increasing”, “decreasing” or “none” (default is “none”). Only meaningful when the output of the query is a list, not a cube.

# where

Where clause is code to return true/false or whether the data will be included in the aggregate. This does not impact the edges; every edge is restricted to it’s own domain.

# esfilter

Similar to the where clause, but used by ES to filter the top-level documents only. The where clause can filter out nested documents, **esfilter** can not. **esfilter** is very fast and should be used whenever possible to restrict the data being processed by scripts and facets.

# edges

The edges clause is an array of edge definitions. Each edge is a column which SQL group-by will be applied; with the additional stipulation that all parts of all domains will have values, even if null.

**name** – The name given to the resulting edge (optional, if the value is a simple attribute name)

**value** – The code to generate the edge value before grouping

**range** – Can be used instead of value, but only for algebraic fields: In which case, if the minimum of a domain part is in the range, it will be used in the aggregate.

**min** – The code that defined the minimum value

**max** – The code defining the supremum (of all values greater than the range, pick the smallest)

**mode** – “inclusive” will ensure any domain part that intersects with the range will be used in the aggregate. “snapshot” (default) will only count ranges that contain the domain part key value.

**test** – Can be used instead of value: Code that is responsible for returning true/false on whether the data will match the domain parts. Use this to simulate a SQL join.

**domain** – The range of values to be part of the aggregation

**allowNulls** – Set to true if you want to aggregate all values outside the domain

# edges[].domain

The domain is defined as an attribute of every edge. Each domain defines a covering partition.

**name** – Name given to this domain definition, for use in other code in the query (default to type name).

**type** – One of a few predefined types (Default {“type”:”default”})

**value** – Domain partitions are technically Javascript objects with descriptive attributes (name, value, max, min, etc). The value attribute is code that will extract the value of the domain after aggregation is complete.

**key** – Code to extract the unique key value from any part object in a partition. This is important so a 1-1 relationship can be established – mapping fast string hashes to slow object comparisons.

**isFacet** – for ES queries: Will force each part of the domain to have it’s own facet. Each part of the domain must be explicit, and define **edges[].domain.partition.esfilter** as the facet filter. Avoid using “script” filters in facets because they are WAY slow.

# edges[].domain.type

Every edge must be limited to one of a few basic domain types. Which further defines the other domain attributes which can be assigned.

default- For when the type parameter is missing: Defines parts of domain as an unlimited set of unique values. Useful for numbers and strings, but can be used on objects in general.

time – Defines parts of a time domain.

**edge.domain.min** – Minimum value of domain (optional)

**edge.domain.max** – Supremum of domain (optional)

**edge.domain.interval** – The size of each time part. (max-min)/interval must be an integer

duration – Defines an time interval

**edge.domain.min** – Minimum value of domain (optional)

**edge.domain.max** – Supremum of domain (optional)

**edge.domain.interval** – The size of each time part. (max-min)/interval must be an integer

numeric – Defines a unit-less range of values

**edge.domain.min** – Minimum value of domain (optional)

**edge.domain.max** – Supremum of domain (optional)

**edge.domain.interval** – The size of each time part. (max-min)/interval must be an integer

count – just like numeric, but limited to integers >= 0

set – An explicit set of unique values

**edge.domain.partitions** – the set of values allowed. These can be compound objects, but **edge.test** and **edge.domain.value** need to be defined.

# analytic

Each analytic column defines an additional attribute for the result set. An analytic does not change the number of rows returned. For each analytic, the data is grouped, sorted and assigned a “rownum” attribute that can be used to calculate the attribute value.

**name** – name given to resulting attribute

**value** – code to determine the attribute value. It has access to two special variables:

**“rownum”** – which is integer, starting at zero for the first row

**“rows”** – an array of all data in the group.

**edges** – an array of column names used to determine the groups

**where** – code that returns true/false to indicate if a record is a member of any group. This will not affect the number of rows returned, only how the analytic is calculated. If where returns false then **rownum** and **rows** will both be null: Be sure to properly handle those values in your code.

**sort** – a single attribute name, or array of attribute names, used to sort the members of each group

# DREADFUL BITS

There are some parts of CUBE that have not been fully thought out and refactored to match.

## CUBE.merge

What does it mean to match edges of two, or more, cubes? If the cubes match on all edges, then their elements can be merged to a single element. But in the general case it is not clear to me what it means to match on an edge. In some of the simpler cases, we may merge summary data (which are smaller dimensional cubes). Maybe disjoint edges declare a hyper cube?

Until this is resolved we can only merge all edges, which must all match.

CUBE.merge({"cubes":[

{"from":s0, "edges":["test\_name", "date"]},

{"from":s1, "edges":["test\_name", "date"]}

]})