VDI.js

Data Set Schemata

```
var color_palette = // various shades of blue for the charts
[
  '#00b0f0', '#1f497d', '#3e6ea7', '#6994c7', '#8db3e2',
  '#b8cce4', '#31859b', '#4bacc6', '#92cddc', '#b7dde8',
  '#0082ba', '#004499', '#0062dd', '#ccccee'
]

var schemata = // describe the data - column names and ordering
{
  supply:     [ 'datacenter', 'region', 'machine_type' ],
  history:    [ null, 'datacenter', 'region', 'machine_type' ],
  sessions:   [ null, 'region', 'datacenter', 'cluster', 'user_type', 'machine_type' ],
  datastores: [ 'region', 'datacenter', 'machine_type' ],
  dstrend:    [ 'region', 'datacenter', 'machine_type' ]
}

var aggregates = // which columns get summed when we render each chart?
{
  supply:     [ 3, 4 ],
  history:    [ 4, 5 ],
  sessions:   [ 6, 7, 8 ],
  datastores: [],
  dstrend:    []
}

var chart_opts = // chart.js options for each chart
{
  supply:
  {
    title: 'VDI Supply Available for Use',
    yAxis: 'User Sessions Available',
    labels: [ 'Internal', 'Third Party Access' ], // these need to correspond (in order) to the numeric colums in
the JSON
    type: 'bar'
  },
  history:
  {
    title: 'VDI Supply History',
    yAxis: 'User Sessions Available',
    labels: [ 'Internal', 'Third Party Access' ],
    type: 'line'
  },
  sessions:
  {
    title: 'Historical Number of VDI Sessions',
    yAxis: 'User Sessions',
    labels: [ '# of Free Machines', '# of Users Logged On', 'Total # of Machines Running' ],
    type: 'line'
  },
```

```
  },
  datastores:
  {
    title: 'VDI Datastores',
    yAxis: '[Placeholder]',
    labels: [ 'Category 1', 'Category 2' ],
    type:  'bar'
  },
  dstrend:
  {
    title: 'VDI Storage Utiliztion Trends',
    yAxis: '[Placeholder]',
    labels: [ 'Category 1', 'Category 2' ],
    type:  'line'
  },
}

var predictions = [] // holds all data, all the time.  not mutable
var selections  = [] // holds the subset of data that the user has selected
var possibles   =    // all possible unique values for each filter will go here
{
  supply:
  {
    region:       {},
    datacenter:   {},
    machine_type:  {}
  },
  history:
  {
    region:       {},
    datacenter:   {},
    machine_type:  {}
  },
  sessions:
  {
    region:       {},
    datacenter:   {},
    cluster:      {},
    user_type:    {},
    machine_type:  {}
  },
  datastores:
  {
    region:       {},
    datacenter:   {},
    machine_type:  {}
  },
  dstrend:
  {
    region:       {},
    datacenter:   {},
    machine_type:  {}
  }
}

var filters = {} // this will be the subset of possible values that the user has selected in the filters
```

```
$.extend( true, filters, possibles ) // deep-copy the easy way
```

Filter Functions

Handle Filter

```
function handle_filter ( ev )
{
  let filter    = $( ev.target ).attr( 'chartfilter' )
  let chart_key = $( ev.target ).attr( 'chartkey' )

  reset_filter( chart_key, filter )

  let filter_vals   = Object.keys( filters[ chart_key ][ filter ] )
  let selected_opts = $( ev.target ).find( 'option:selected' )
  let selected_vals = []
  let delete_these  = []

  for ( i = 0; i < selected_opts.length; i++ )
  {
    selected_vals.push( $( selected_opts[i] ).val() )
  }

  // console.log( 'need to search ' + filter + ' for ' + selected_vals )

  for ( i = 0 ; i < filter_vals.length; i++ )
  {
    let filter_value = filter_vals[i]

    if ( selected_vals.indexOf( filter_value ) == -1 )
    {
      delete_these.push( filter_value )
    }
  }

  for ( i = 0; i < delete_these.length; i++ )
  {
    delete filters[ chart_key ][ filter ][ delete_these[i] ]
  }

  //console.log( 'we deleted these non-matching items from filters.' + filter )
  //console.log( delete_these )
  //console.log( filters[ filter ] )

  reset_selections( chart_key ) // start from zero.  get all the data and work from there.
  apply_filters( chart_key ) // scan through the data and only select the data that matches the filters
  // ...which calls: update_widgets() // scan through the widgets and enable/disable options based on user
choice
  // ...and then calls: render_chart() // show the chart with filters applied
}
```

Initialize All

```
function init_all ()
{
  // create tabs for content
  $( 'div#vdi-tabs' ).tabs()

  let chart_keys = Object.keys( possibles )

  for ( i = 0 ; i < chart_keys.length; i++ )
  {
    let key = chart_keys[i]

    // get json from server for the visuals...
    $.getJSON( '/wimi/kabiyesi/' + key, function ( json ) { setup_chart( key, json ) } )
  }

  // JSON payload will be a list of records, where each record is a list itself
  // wherein the column order for each record will be as follows:
  // datacenter  region   machine_type   INT_supply  TPA_supply

  $( 'select.filter' ).change( handle_filter )

  $( 'input.filter-reset' ).click( reset_all )
}
```

Reset All

```
function reset_all ( ev )
{
  let chart_key = $( ev.target ).attr( 'chartkey' )

  init_filters( chart_key ) // reset filters back to default state (all-selected)
  reset_selections( chart_key ) // reset selections back to default state (all data)
  populate_filter_widgets( chart_key ) // put back all the options in the filters
  update_widgets( chart_key ) // make all options selectable now
  render_chart( chart_key ) // show the chart with all options selected
}
```

Initialize Filters

```
function init_filters ( chart_key )
{
  let chart_columns = schemata[ chart_key ]

  // initialize or reset filter state-tracking variable - by default, all data
  // is selected (filter state variable should include all possible choices)
  for ( i = 0; i < predictions[ chart_key ].length; i++ )
  {
    for ( j = 0; j < chart_columns.length; j++ )
    {
      if ( chart_columns[j] == null ) { continue }

      possibles[ chart_key ][ chart_columns[j] ][ predictions[ chart_key ][i][j] ] = predictions[ chart_key ][i][j]
    }
  }

  select_all_filters( chart_key )

  return filters[ chart_key ]
}
```

Select All Filters

```
function select_all_filters ( chart_key )
{
  let possible_categories = Object.keys( possibles[ chart_key ] )

  for ( i = 0; i < possible_categories.length; i++ )
  {
    let column = possible_categories[i]
    let values = Object.keys( possibles[ chart_key ][ column ] )

    for ( j = 0; j < values.length; j++ )
    {
      filters[ chart_key ][ column ][ values[j] ] = values[j]
    }
  }

  return filters
}
```

Reset Filter

```
function reset_filter ( chart_key, filter )
{
  //console.log( 'resetting filters.' + filter + ' to ... ' )
  //console.log( possibles[ filter ] )

  let restore_vals = Object.keys( possibles[ chart_key ][ filter ] )

  for ( i = 0; i < restore_vals.length; i++ )
  {
    filters[ chart_key ][ filter ][ restore_vals[i] ] = restore_vals[i]
  }
}
```

Populate Filter Widgets

```
function populate_filter_widgets ( chart_key )
{
  // first empty any non-header options from all filter widgets
  $( 'select.filter[chartkey="' + chart_key + '"] option' ).each
  (
    function ()
    {
      if ( $( this ).val() != '' ) // header options have no value
      {
        $( this ).remove()
      }
    }
  )

  let filter_names = Object.keys( possibles[ chart_key ] )

  // now iterate over predictions data and populate filters
  for ( i = 0; i < filter_names.length; i++ )
  {
    let filter = filter_names[i]
    let values = Object.keys( possibles[ chart_key ][ filter ] ).sort()
    let widget = $( 'select.filter[chartkey="' + chart_key + '"][chartfilter="' + filter + '"]' )

    //console.log( 'restoring all possible options to filter: ' + filter + ' for chart key: ' + chart_key )
    //console.log( { restoring_values: values, to_select_widget: widget } )

    // then stuff the widget with the options that it should have, based on the data (nod to Marta ;-))
    for ( j = 0; j < values.length; j++ )
    {
      widget.append( '<option value="' + values[j] + '">' + values[j] + '</option>'  )
    }
  }
}
```

Update Filter Widgets

```
function update_widgets ( chart_key )
{
  // based on user selections, update filter widgets to reflect what choices
  // have been made, and what choices are/are not possible
  let filter_names = Object.keys( filters[ chart_key ] )

  filter_scan: for ( i = 0; i < filter_names.length; i++ ) // loop through each filter widget
  {
    let filter_name = filter_names[i]
    let col_index   = schemata[ chart_key ].indexOf( filter_name )
    let options     = $( 'select.filter[chartkey="' + chart_key + '"][chartfilter="' + filter_name + '"] option' ) // get
filter widget options

    options.attr( { disabled: true } ) // disable all options in the widget

    opt_scan: for ( j = 0; j < options.length; j++ ) // scan through the widget options and...
    {
      let opt_val = $( options ).eq( j ).val()

      data_scan: for ( k = 0; k < selections[ chart_key ].length; k++ ) // check selections list for matching
records
      {
        if ( selections[ chart_key ][k][ col_index ] == opt_val ) // enable the option if it matches user choices
in "selections" variable
        {
          $( options ).eq( j ).attr( { disabled: false } )

          break data_scan // bail on first match (no need to keep pounding)
        }
      }
    }
  }

  // delete the disabled options unless it's the header opt with a none-value
  $( 'select.filter[chartkey="' + chart_key + '"] option:disabled' ).each
  (
    function ()
    {
      if ( $( this ).val() != '' )
      {
        $( this ).remove()
      }
    }
  )
}
```

Apply Filters

```
function apply_filters ( chart_key )
{
  filter_selections( chart_key )

  update_widgets( chart_key )

  render_chart( chart_key )
}
```

Reset Selections

```
function reset_selections ( chart_key ) // reset selections to 'everything' selected
{
  selections[ chart_key ] = predictions[ chart_key ] // for now, it's really not more complicated than this ;-)
}
```

Filter Selections

```
function filter_selections ( chart_key )
{
  let results = []
  let filter_names = Object.keys( filters[ chart_key ] )

  // iterate over records and check if each one matches users filters (choices)
  scan_records: for ( i = 0; i < selections[ chart_key ].length; i++ )
  {
    let record = selections[ chart_key ][i]

    // start checking this record, filter by filter, and skip over it at
    // the first instance where it fails to match
    check_record: for ( j = 0; j < filter_names.length; j++ )
    {
      let filter_name   = filter_names[j] // name of the filter we're dealing with
      let filter_column = schemata[ chart_key ].indexOf( filter_name ) // db record column index to which this
filter pertains
      let column_value  = record[ filter_column ] // value of the db record in the column to which this filter
pertains

      if ( filters[ chart_key ][ filter_name ][ column_value ] == undefined ) // e.g - filters.supply.region.NAM
      {
        continue scan_records // quit checking for matches on this record at the first sign of failure
      }
    }

    results.push( record ) // if we got here, all filters matched, so save result
  }

  selections[ chart_key ] = results

  return results
}
```

In Filter

```
// check if given value exists in given filter using dictionary key lookup

function in_filter ( value, filter ) { return filter[ value ] != null }
```

Chart Functions

Setup Chart

```
function setup_chart ( chart_key, json )
{
  // instantiate data
  predictions[ chart_key ] = json // predictions shouldn't be messed with - holds all data
  selections[ chart_key ]  = predictions[ chart_key ] // default to all records selected

  // the selections variable should always reflect the predictions data after
  // all user filters have been applied.

  // initialize filter state-tracking variable - by default, all data is
  // selected (filter state variable should include all possible choices)
  init_filters( chart_key )

  // populate filter widgets
  populate_filter_widgets( chart_key )

  render_chart( chart_key )
}
```

Render Chart

```
function render_chart ( chart_key )
{
  // render chart.js chart when selections change
  //let match_count = $( 'select.filter[chartkey="' + chart_key + '"][chartindicator] option' ).length - 1
  //let match_indic = $( 'select.filter[chartkey="' + chart_key + '"][chartindicator]' ).attr( 'chartindicator' ) // e.g.-
"VMs"

  //$( 'span.status[chartkey="' + chart_key + '"]' ).text( 'showing ' + match_count + ' matching ' + match_indic
)

  let chart_data = make_chartjs_datasets( chart_key )
  let my_opts    = chart_opts[ chart_key ]

  $( 'div.chart-div[chartkey="' + chart_key + '"]' ).html( '<canvas class="chart-canvas" chartkey="' +
chart_key + '"></canvas>' )

  let vdi_ctx = $( 'canvas.chart-canvas[chartkey="' + chart_key + '"]' )

  // colors and other per-dataset options
  for ( i = 0; i < chart_data.datasets.length; i++ )
  {
    chart_data.datasets[i].backgroundColor = color_palette[i]
```

```
    chart_data.datasets[i].borderColor = color_palette[i]
    chart_data.datasets[i].fill = my_opts.type == 'line' ? false : undefined
  }

  var vdi_chart = new Chart
  (
    vdi_ctx,
    {
      type: my_opts.type,
      data: chart_data,
      options:
      {
        title: { display: true, text: my_opts.title },
        legend: { display: true, position: 'top' },
        responsive: true,
        maintainAspectRatio: false,
        hoverMode: 'index',
        scales:
        {
          yAxes:
          [
            {
              scaleLabel: { display: true, labelString: my_opts.yAxis },
              ticks:
              {
                beginAtZero: true,
                min: 0,
                callback: function ( label, index, labels )
                {
                  return label.toLocaleString()
                }
              }
            }
          ]
        },
        tooltips:
        {
          callbacks:
          {
            label: function ( tooltipItem, data )
            {
              return data.datasets[ tooltipItem.datasetIndex ].data[ tooltipItem.index ].toLocaleString()
            }
          }
        }
      }
    }
  )
}
```

Make Chart.js Datasets

```
function make_chartjs_datasets ( chart_key )
{
```

```
// make a data structure based on the current contents of the selections
// variable.  the data structure will be fed to chart.js to create a new
// chart, so it needs to be in a certain format that follows certain rules...

let chart_struct =
{
  labels: [], // labels for x-axis
  datasets: [] // values from numeric columns of data, after aggregation
}

// each chart.js dataset needs a label.  we define these in the chart_opts
// global var at the top of this file, and reference it here, by chart key

for ( i = 0; i < chart_opts[ chart_key ].labels.length; i++ ) // line/bar labels
{
  chart_struct.datasets[i] = { label: chart_opts[ chart_key ].labels[i], data: [] }
}

// part of this process will be aggregation of the data.  we will group
// the data by the unique values from the first column of data in all the
// db records.  Then we will sum up the numeric data in each column that
// is marked for aggregation in the global "aggreagetes" variable, by chart key.

let group_by = {} // keys will be the unique values from the first column of db records for the chart

// first we need to determine what our groupings are.  We let the data tell
// us its own story.  Specifically, we scan through the data to identify the
// unique instances of whatever is getting grouped.  Since we always group by
// the data in the first column, we can take that as a given and hard-code
// the zeroth item in each record we scan.  since dictionary keys can't be
// duplicated, we leverage that feature in order to create our distinct groups

for ( i = 0; i < selections[ chart_key ].length; i++ )
{
  let group_by_value = selections[ chart_key ][i][0] // get the group-by column value

  // the column value is used as the dictionary key, and the value is just
  // an empty list that will later hold the summed values for that column

  group_by[ group_by_value ] = []
}

// by virtue of the loop above, we now have a distinct list of values from
// the first column of data for this chart.  It is contained in the keys
// of the group_by dictionary.  In addition to needing these values to label
// the x-axis of the chart (which we do by saving them in the
// chart_struct.labels list variable), but we're going to need to refer to
// this list of values several more times before we're done with the
// aggregation we need to do

chart_struct.labels = Object.keys( group_by ).sort()

// now we need to start summing things up.  Our aggregates global variable
// contains the column index offset of the column(s) in each DB record that
// need to be SUM()'med, so we use that below in order to avoid asking JS
```

```
  // to add non-numeric columns of data

  for ( i = 0; i < selections[ chart_key ].length; i++ ) // loop through all records
  {
    let group_by_value = selections[ chart_key ][i][0]

    // for every record, loop through columns getting SUM()'med

    for ( j = 0 ; j < aggregates[ chart_key ].length; j++ )
    {
      // line below is so we don't get NaN's in the output
      group_by[ group_by_value ][j] = group_by[ group_by_value ][j] ? group_by[ group_by_value ][j] : 0

      // this actually SUM()'s the column, grouped by whatever the value is
      // in the first column of the DB record we are looking at in this
      // iteration of the loop...

      group_by[ group_by_value ][j] += parseFloat( selections[ chart_key ][i][ aggregates[ chart_key ][j] ] )
    }
  }

  // and now we need to populate the chart.js data structure with the data
  // we just aggregated (grouped and summed) in the loop above

  for ( i = 0 ; i < aggregates[ chart_key ].length; i++ ) // loop through columns that got SUM()'med
  {
    for ( j = 0; j < chart_struct.labels.length; j++ ) // loop through each of the distinct values in each grouping
    {
      let group_by_value = chart_struct.labels[j] // this is the group-by data category

      // for each of the groupings of data we have, shove the data for that
      // rouping into the chart.js dataset pertaining to whatever column
      // we're currently iterating over in the outer loop that encloses
      // this nested loop...

      chart_struct.datasets[i].data.push( group_by[ group_by_value ][i] )
    }
  }

  // console.log( { group_by: group_by, chart_struct: chart_struct, chart_key: chart_key } )

  return chart_struct
}
```

Init_all

```
$( init_all )
```