

# CartoDB

[Visualize](#) [Analyze](#) [Develop](#) [Case studies](#) [Documentation](#) [Pricing](#) [Sign in](#)

*API reference*

## CartoDB.js - API reference

CartoDB offers a simple unified JavaScript library called CartoDB.js that let you interact with the CartoDB service. This library allows you to connect to your stored visualizations, create new visualizations, add custom interaction, or access and query your raw data from a web browser; meaning, your applications just got a whole lot more powerful with a lot less code.

When you add CartoDB.js to your websites you get some great new tools to make maps or power your content with data. Let's take a look.

### Getting started

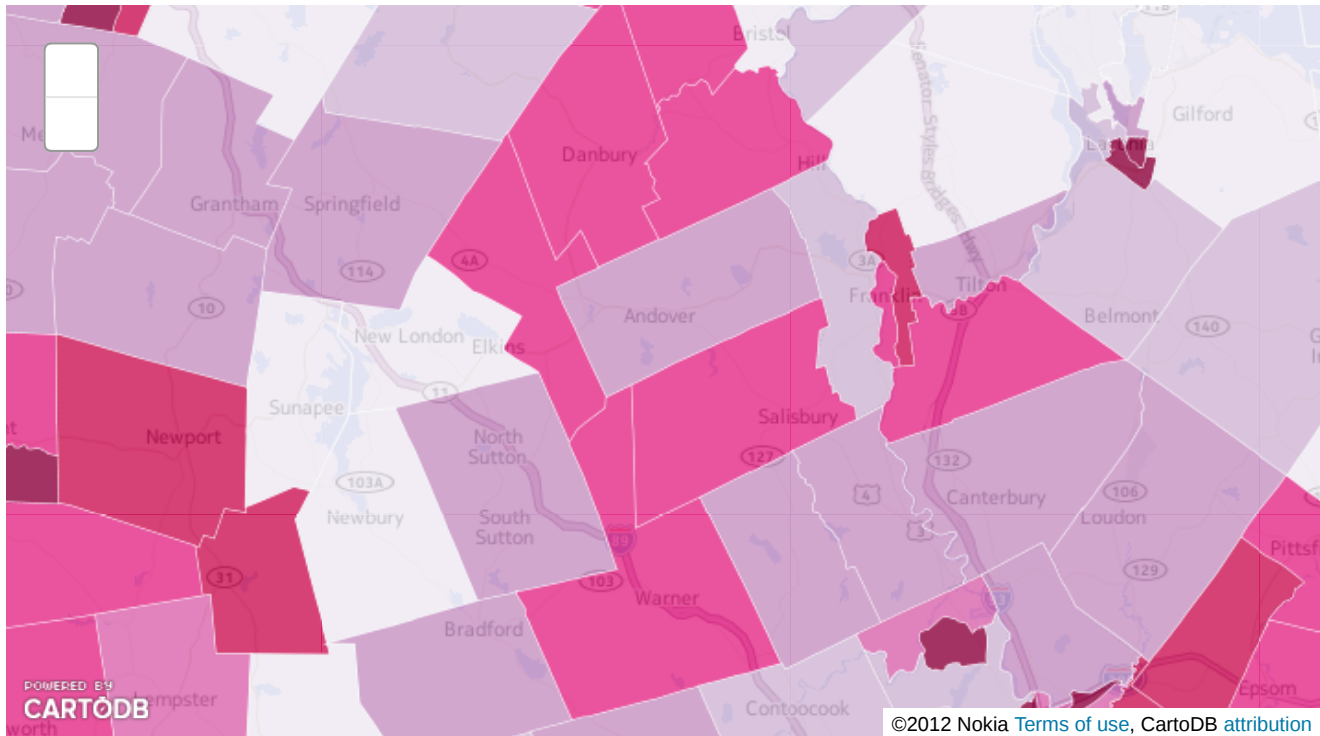
The simplest way to use a visualization created in CartoDB on an external site is at follows...

Create a simple visualization

```
...
<body>
  <div id="map"></div>
</body>
...
<script>
  // get the viz.json url from the CartoDB UI
  // - click on visualize
  // - create new visualization
  // - make visualization public
  // - click on publish
  // - go to API tab
  window.onload = function() {
    cartodb.createVis('map', 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-54f
  }
```

```
</script>
```

With a similar source code you can create a visualization like this one:



[Grab the complete example source code](#)

## Using the library

CartoDB.js can be used when you want to embed and use a visualization you have designed using CartoDB user interface, or to create visualizations from scratch dynamically using your data. If you want to create new maps on your webpage, jump to “using CartoDB visualizations in your webpage”. If you already have maps on your webpage and want to add CartoDB visualizations to them, read “Add CartoDB layer to an existing map”.

You can also use CartoDB API to create visualization without having to define them using the UI. This can be useful when the visualizations react to user interactions. To read more about it jump to, create [create visualizations at runtime](#).

We’ve also made it easier than ever for you to build maps using the mapping library of your choice. Whether you are using Leaflet or Google Maps your CartoDB.js code remains the same. This makes our API documentation simple and straightforward. It also makes it easy for you to remember and keep consistent if you development or maintain multiple maps online.

To start using CartoDB.js just paste this piece of code within the HEAD tags of your HTML:

## Linking cartodb.js on your html file

```
<link rel="stylesheet" href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.css" />
<!--[if lte IE 8]>
  <link rel="stylesheet" href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.ie.css" />
<![endif]-->
<script src="http://libs.cartocdn.com/cartodb.js/v3/cartodb.js"></script>
```

## Using CartoDB visualizations in your webpage

There are two ways to include a visualization you have done using the CartoDB UI in your webpage:

### Create a visualization from scratch

The easiest way to quickly get a CartoDB map onto your webpage. Use this when there is no map in you application and you want to add the visualization to hack over it. With this method, CartoDB.js handles all the details of loading a map interface, basemap, and your CartoDB visualization.

You can start by giving cartodb.js the DIV ID from your HTML where you want to place your map, and the viz.json URL of your visualization, which you can get from the publish window.

Simplest way to add your map to a webpage ever!

```
cartodb.createVis('map', 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-5404a6a6e...
```

That's it! No need to create the map instance, insert controls, or load layers, it handles it all for you. If you want to modify the result after instantiating your map with this method, take a look at the CartoDB.js API [available methods](#). For example, you can also use the returned layer to build more functionality (show/hide, click, hover, custom infowindows):

Simplest way to add your map to a webpage ever!

```
cartodb.createVis('map', 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-5404a6a6e...
  .done(function(vis, layers) {
    // layer 0 is the base layer, layer 1 is cartodb layer
    // when setInteraction is disabled featureOver is triggered
```

```

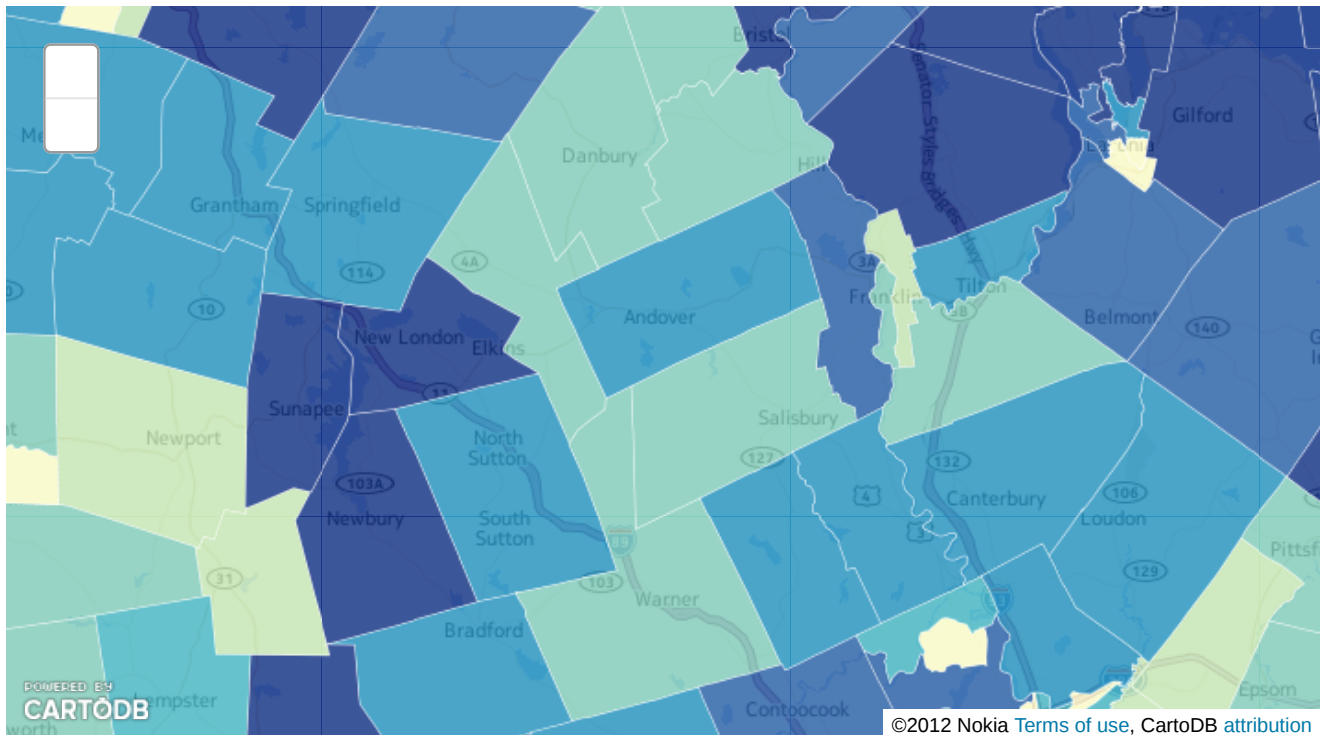
layers[1].setInteraction(true);

layers[1].on('featureOver', function(e, latlng, pos, data, layerNumber) {
  cartodb.log.log(e, latlng, pos, data, layerNumber);
});

// you can get the native map to work with it
// depending if you use google maps or leaflet
map = vis.getNativeMap();

// now, perform any operations you need
// map.setZoom(3)
// map.setCenter(new google.maps.LatLng(...))
});

```



If you are using Google Maps for your basemap in your CartoDB account, using createViz requires that you load the **Google Maps V3** JavaScript library in the HEAD of your HTML. If you use other basemaps, cartodb.js will load the Leaflet library for you automatically.

#### Adding cartodb layers to an existing map

In case you already have a map instantiated on your page, you can simply use the

[createLayer](#) method to add new CartoDB layers to it. This is particularly useful when you have more things on your map apart from CartoDB layers or you have an application where you want to integrate CartoDB layers.

Below, you have an example using a previously instantiated leaflet map.

Adding cartodb layers to an existing map

```
<div id="map_canvas"></div>

<script>

  var map = new L.Map('map_canvas', {
    center: [0,0],
    zoom: 2
  });

  cartodb.createLayer(map, 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-5404e
    .addTo(map)
    .on('done', function(layer) {
      //do stuff
    })
    .on('error', function(err) {
      alert("some error occurred: " + err);
    });
</script>
```

[Grab the complete example source code](#)

### Creating visualizations at runtime

All CartoDB services are available through the API, which basically means that you can create a new visualization without doing it before through the CartoDB UI. This is particularly useful when you are modifying the visualization depending on user interactions that change the SQL to get the data or CartoCSS to style it. This method, although needs more programming skills, provides all the flexibility you might need to create more dynamic visualizations.

When you create a visualization using the CartoDB website, you get automatically a viz.json URL defining it. When you want to create the visualization via JS, obviously you don't have it, so you will pass all the required parameters to the library so that it can

create the visualization at runtime and display it on your map. It is pretty simple.

### Creating visualizations at runtime

```
// create a layer with 1 sublayer
cartodb.createLayer(map, {
  user_name: 'mycartodbuser',
  type: 'cartodb',
  sublayers: [{
    sql: "SELECT * FROM table_name",
    cartocss: '#table_name {marker-fill: #F0F0F0;}'
  }]
})

.addTo(map) // add the layer to our map which already contains 1 sublayer
.done(function(layer) {

  // create and add a new sublayer
  layer.createSubLayer({
    sql: "SELECT * FROM table_name limit 200",
    cartocss: '#table_name {marker-fill: #F0F0F0;}'
  });

  // change the query for the first layer
  layer.getSubLayer(0).setSQL("SELECT * FROM table_name limit 10");
});
```

Want further information? [Check out the complete API method list.](#)

## Usage examples

If you want to start playing with the library, the best way to do it might be to take a look to some of the examples below:

- An easy example using the library - ([view live](#) / [source code](#)).
- Leaflet integration - ([view live](#) / [source code](#)).
- Google Maps V3 integration - ([view live](#) / [source code](#)).
- Customizing infowindow data - ([view live](#) / [source code](#)).
- An example using a layer selector - ([view live](#) / [source code](#)).

- The Hobbit map done with the library - ([view live](#) / [source code](#)).

## API methods

The documentation below reflects CartoDB.js for the v3 library versions. For major changes in the library we will update the documentation here. This documentation is meant to help developers find specific methods for using the CartoDB.js library.

For deprecated versions of this library, refer to [this documentation](#).

## Visualization

**cartodb.createVis**(map\_id, vizjson\_url[, options] [, callback])

Creates a visualization inside the map\_id DOM object.

cartodb.createVis

```
var url = 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-5404a6a683d5/viz.json'
```

```
cartodb.createVis('map', url)
  .done(function(vis, layers) {
  });
```

### Arguments

- **map\_id**: a DOM object, for example \$('#map') or a DOM id.
- **vizjson\_url**: url of the vizjson object.
- **options**:
  - **shareable**: add facebook and twitter share buttons.
  - **title**: adds a header with the title of the visualization.
  - **description**: adds description to the header (as you set in the UI).
  - **searchControl**: adds a search control (default: false).
  - **zoomControl**: adds zoom control (default: true).
  - **loaderControl**: adds loading control (default: true).
  - **center\_lat**: latitude where the map is initialized.
  - **center\_lon**: longitude where the map is initialized.
  - **zoom**: initial zoom.
  - **cartodb\_logo**: default to true, set to false if you want to remove the cartodb logo.
  - **infowindow**: set to false if you want to disable the infowindow (enabled by default).
  - **time\_slider**: show time slider with torque layers (enabled by default)
  - **layer\_selector**: show layer selector (default: false)

- **legends**: if it's true legends are shown in the map.
- **https**: if true forces tiles to be fetched using https. If false it uses the predefined method
- **scrollwheel**: enable/disable the ability of zooming using scrollwheel (default enabled)
- **fullscreen**: if true adds a button to toggle the map fullscreen

## **cartodb.Vis**

### **vis.getLayers()**

Returns an array of layers in the map. The first is the base layer.

### **vis.addOverlay(options)**

Adds an overlay to the map that can be either a zoom control, a tooltip or an infobox.

#### **Arguments**

- **options**:
  - **layer**: layer from the visualization where apply the overlay (optional)
  - **type**: 'zoom' | 'tooltip' | 'infobox'
  - Extra options are available depending on the UI component selected before

#### **Returns**

An overlay object, see [vis.Overlays](#)

### **vis.getOverlay(type)**

Return the first overlay with the specified **type**.

`vis.getOverlay`

```
var zoom = vis.getOverlay('zoom');
```

```
zoom.clean() // remove it from the screen
```

### **vis.getOverlays()**

Returns a list of overlays currently on the screen (see overlays description).

### **vis.getNativeMap()**



Returns the native map object being used. It can be `google.maps.Map` or `L.Map` depending on the provider you setup in the UI.

### **vis.Overlays**

An overlay is a control shown on top of the map.

Overlay objects are always created using method **addOverlay** of `cartodb.Vis` object.

An overlay is internally a **Backbone.View** so if you know how backbone works you can use it. If you want to use plain DOM objects you can access to **overlay.el** (**overlay.\$el** for jQuery object).

**cartodb.createLayer**(map, layerSource [, options] [, callback])

With visualizations already created through the CartoDB console, you can simply use the **createLayer** function to add them into your web pages. Unlike **createVis**, this method requires an already activated **map** object and it does not load a basemap for you. The method works the same whether your map object is [Google Maps](#) or [Leaflet](#).

#### **Arguments**

- **map**: Leaflet `L.Map` or Google Maps `google.maps.Map` object. The map should be initialized before calling this function.
- **layerSource**: contains information about the layer. It can be specified in 2 ways:

- passing the url where the layer data is located:

```
cartodb.createLayer
```

```
cartodb.createLayer(map, 'http://myserver.com/layerdata.json')
```

- passing the data directly:

```
cartodb.createLayer
```

```
cartodb.createLayer(map, { ... layer metadata ... });
```

Layer metadata is always in the form: `{ type: 'LAYER_TYPE_NAME', options: {...} }`

See [cartodb.CartoDBLayer](#) too see an example.

- **options**:

- **https**: force https
- **refreshTime**: if is set, the layer is refreshed each refreshTime milliseconds.
- **infowindow**: set to false if you want to disable the infowindow (enabled by default).
- **legends**: if it's true legends are shown in the map.
- **time\_slider**: show time slider with torque layers (enabled by default)
- **layerIndex**: when the visualization contains more than one layer this index allow to select what layer is created. Take into account that

```
layerIndex == 0
```

is the base layer and that all the tiled layers (non animated ones) are merged into a single one. The default value for this option is 1 (usually tiled layers).

- **callback(layer)**: if a function is specified is called when the layer is created passing it as argument.

#### Returns

Promise object. You can listen for the following events:

- **done**: triggered when the layer is created, the layer is passed as first argument. Each layer type has different options, see layers section.
- **error**: triggered when the layer couldn't be created. The error string is the first argument.

You can call to `addTo(map[, position])` in the promise so when the layer is ready it will be added to the map.

#### Example

```
cartodb.createLayer
```

```
var map;
var mapOptions = {
  zoom: 5,
  center: new google.maps.LatLng(43, 0),
  mapTypeId: google.maps.MapTypeId.ROADMAP
};

map = new google.maps.Map(document.getElementById('map'), mapOptions);
```

```
cartodb.createLayer(map, 'http://documentation.cartodb.com/api/v2/viz/2b13c956-e7c1-11e2-806b-5404a6e
.addTo(map)
.on('done', function(layer) {
  layer
    .on('featureOver', function(e, latlng, pos, data) {
      console.log(e, latlng, pos, data);
    })
    .on('error', function(err) {
      console.log('error: ' + err);
    });
}).on('error', function(err) {
  console.log("some error occurred: " + err);
});
```

## **cartodb.CartoDBLayer**

CartoDBLayer allows you to manage tiled layers from CartoDB. It manages the sublayers.

### **layer.clear()**

Should be called after removing the layer from the map.

### **layer.hide()**

Hides the cartodb layer from the map.

### **layer.show()**

Show the cartodb layer in the map if it was previously added.

### **layer.setOpacity(opacity)**

Change the opacity of the layer.

### **Arguments**

- **opacity**: value in range [0, 1].

### **layer.getSubLayer(layerIndex)**

Get a previously created sublayer. And exception is raised if not exists

#### Arguments

- **layerIndex**: 0 based index of the sublayer to get. Should be within [0, getSubLayerCount())

#### Returns

SubLayer object

#### Example

layer.getSubLayer

```
layer.getSubLayer(1).hide();
```

```
var sublayer = layer.getSubLayer(0);
```

```
sublayer.setSQL('SELECT * FROM table_name limit 10');
```

#### **layer.createSubLayer(layerDefinition)**

Adds a new data to the current layer. With this method data from multiple tables can be easily visualized. New in V3.

#### Arguments

- **layerDefinition**: an object with the sql and cartocss that defines the data, should be like:

layer.createSubLayer

```
sql: "SELECT * FROM table_name",  
cartocss: "#layer { marker-fill: red; }",  
interactivity: 'cartodb_id, area, column' // optional
```

sql and cartocss are mandatory, an exception is raised if any of them are not present. If the interactivity is not set, there is no interactivity enabled for that layer (better performance). SQL and CartoCSS syntax should be correct, see Postgres and CartoCSS reference. There are some restrictions in the SQL queries:

- must not write. INSERT, DELETE, UPDATE, ALTER and so on are not allowed (the query will fail)
- must not contain trailing semicolon

**Returns**

SubLayer object

**Example**

layer.createSubLayer

```
cartodb.createLayer(map, 'http://examples.cartodb.com/api/v2/viz/european_countries_e/viz.json', func
// add populated places points over the countries layer
layer.createSubLayer({
  sql: 'SELECT * FROM ne_10m_populated_places_simple',
  cartocss: '#layer { marker-fill: red; }'
});
}).addTo(map);
```

**layer.invalidate()**

Refresh the data. If the data has been changed in CartoDB server it is displayed. If not nothing happens. Every time a parameter is changed in a sublayer the layer is refreshed so this method don't need to be called manually. New in V3.

**layer.setAuthToken(auth\_token)**

Sets the auth token to create the layer. Only available for private visualizations. An exception is raised if the layer is not being loaded with HTTPS.

**Returns**

the layer itself

**Arguments**

- auth\_token: string

**layer.setParams(key, value)**

```
...  
  
    layer.setParams('test', 10); // sets test = 10  
    layer.setParams('test', null); // unset test  
    layer.setParams({'test': 1, 'color': '#F00'}); // unset test  
...
```

- key: string
- value: string or number

### Returns

the layer itself

## cartodb.CartoDBLayer.SubLayer

### sublayer.set(layerDefinition)

Sets sublayer parameters. Useful when more than one parameter need to be changed. See setSQL and setCartoCSS

### Arguments

- **layerDefinition**: an object with the sql and cartocss that defines the data, should be like

sublayer.set

```
sql: "SELECT * FROM table_name",  
cartocss: "#layer { marker-fill: red; }",  
interactivity: 'cartodb_id, area, column' // optional
```

### Return

self object

### Example

sublayer.set

```
sublayer.set({  
  sql: "SELECT * FROM table_name WHERE cartodb_id < 100",  
  cartocss: "#layer { marker-fill: red }",  
  interactivity: "cartodb_id,the_geom,magnitude"  
});
```

### **sublayer.get(attr)**

Gets the attribute for the sublayer, for example 'sql', 'cartocss'.

#### **Returns**

The requested attribute or undefined if it's not present.

### **sublayer.getSQL()**

Shortcut for get('sql')

### **sublayer.getCartoCSS()**

Shortcut for get('cartocss')

### **sublayer.setSQL(sql)**

Shortcut for set({'sql': 'SELECT \* FROM table\_name'})

### **sublayer.setCartoCSS(sql)**

Shortcut for set({'cartocss': '#layer {...}' });

### **sublayer.remove**

Remove the sublayer. If a method is called after removing it an exception is thrown.

### **sublayer.setInteraction(true)**

Sets the interaction of your layer to true (enabled) or false (disabled). When is disabled **featureOver**, **featureClick** and **featureOut** are **not** triggered.

#### **Arguments**

- **enable**: true if the interaction needs to be enabled.

### **sublayer.show**

Show a previously hidden sublayer. The layer is refreshed after calling this function.

### **sublayer.hide**

Remove temporally the sublayer from the layer. The layer is refreshed after calling this function.

### **sublayer.infowindow**

**sublayer.infowindow** is a Backbone model where we modify the parameters of the infowindow

#### **Arguments**

- **template**: Set the custom infowindow template defined on the html. You can write simple html or use [Mustache templates](#)

sublayer.infowindow.set

```
<div id="map"></div>
```

```
<script>
```

```
  sublayer.infowindow.set('template', $('#infowindow_template').html());
```

```
</script>
```

```
<script type="infowindow/html" id="infowindow_template">
```

```
  <span> custom </span>
```

```
  <div class="cartodb-popup">
```

```
    <a href="#close" class="cartodb-popup-close-button close">x</a>
```

```
    <div class="cartodb-popup-content-wrapper">
```

```
      <div class="cartodb-popup-content">
```

```
        </src>
```

```
        <!-- content.data contains the field info -->
```

```
        <h4>{{content.data.name}}</h4>
```

```
      </div>
```

```
    </div>
```

```
  <div class="cartodb-popup-tip-container"></div>
```



```
</div>  
</script>
```

[Grab the complete example source code](#)

## Events

You can add custom functions to layer events. This is useful for integrating your website with your maps, adding events for mouseovers and click events.

**sublayer.featureOver** -> (event, latlng, pos, data, layerIndex)

A callback when hovers in a feature.

### Callback arguments

- **event**: Browser mouse event object.
- **latlng**: The LatLng in an array [lat,lng] where was clicked.
- **pos**: Object with x and y position in the DOM map element.
- **data**: The CartoDB data of the clicked feature with the **interactivity** param.
- **layerIndex**: the layerIndex where the event happened

### Example

sublayer.on

```
sublayer.on('featureOver', function(e, latlng, pos, data, subLayerIndex) {  
  console.log("mouse over polygon with data: " + data);  
});
```

**layer.featureOut** -> (layerIndex)

A callback when hovers out any feature.

**layer.featureClick** -> (event, latlng, pos, data, layerIndex)

A callback when clicks in a feature.

### callback arguments

Same as **featureOver**.

## Specific UI functions

There are a few functions in CartoDB.js for creating, enabling, and disabling pieces of the user-interface.

### **cartodb.geo.ui.Tooltip**

Shows a small tooltip on hover:

cartodb.geo.ui.Tooltip

```
var tooltip = vis.addOverlay({  
  type: 'tooltip'  
  template: '<p>{{variable}}</p>' // mustache template  
});
```

### **cartodb.geo.ui.Tooltip.enable**

The tooltip is shown when hover on feature when is called.

### **cartodb.geo.ui.Tooltip.disable**

The tooltip is not shown when hover on feature.

### **cartodb.geo.ui.InfoBox**

Show an small box when the user hovers on a map feature. The position is fixed:

cartodb.geo.ui.InfoBox

```
var box = vis.addOverlay({  
  type: 'infobox',  
  template: '<p>{{name_to_display}}</p>'  
  width: 200, // width of the box  
  position: 'bottom|right' // top, bottom, left and right are available  
});
```

### **cartodb.geo.ui.InfoBox.enable**

The tooltip is shown when hover on feature when is called.

### **cartodb.geo.ui.InfoBox.disable**

The tooltip is not shown when hover on feature.

### **cartodb.geo.ui.Zoom**

Shows the zoom control:

cartodb.geo.ui.Zoom

```
vis.addOverlay({ type: 'zoom' });
```

**cartodb.geo.ui.Zoom.show()**

**cartodb.geo.ui.Zoom.hide()**

## **Getting data with SQL**

CartoDB offers a powerful SQL API for you to query and retrieve data from your CartoDB tables. The CartoDB.js offers a simple to use wrapper for sending those requests and using the results.

### **cartodb.SQL**

**cartodb.SQL** is the tool you will use to access data you store in your CartoDB tables. This is a really powerful technique for returning things like: **items closest to a point**, **items ordered by date**, or **GeoJSON vector geometries**. It's all powered with SQL and our tutorials will show you how easy it is to begin with SQL.

cartodb.SQL

```
var sql = new cartodb.SQL({ user: 'cartodb_user' });
sql.execute("SELECT * FROM table_name WHERE id > {{id}}", { id: 3 })
  .done(function(data) {
    console.log(data.rows);
  })
  .error(function(errors) {
    // errors contains a list of errors
    console.log("errors:" + errors);
  })
```

It accepts the following options:

- **format**: should be geoJSON.
- **dp**: float precision.
- **jsonp**: if jsonp should be used instead of CORS. This param is enabled if the browser does not support CORS.

These arguments will be applied for all the queries performed by this object, if you want to override them for one query see **execute** options.

**sql.execute**(sql [,vars][, options][, callback])

It executes a sql query.

#### Arguments

- **sql**: a string with the sql query to be executed. You can specify template variables like `{{variable}}` which will be filled with **vars** object.
- **vars**: a map with the variables to be interpolated in the sql query.
- **options**: accepts **format**, **dp** and **jsonp**. This object also overrides the params passed to \$.ajax.

#### Returns

Promise object. You can listen for the following events:

- **done**: triggered when the data arrives.
- **error**: triggered when something failed.

You can also use done and error methods:

sql.execute

```
sql.execute('SELECT * FROM table_name')  
  .done(fn)  
  .error(fnError)
```

**sql.getBounds**(sql [,vars][, options][, callback])

Return the bounds [ [sw\_lat, sw\_lon], [ne\_lat, ne\_lon ] ] for the geometry resulting of specified query.

sql.getBounds

```
sql.getBounds('select * from table').done(function(bounds) {
```

```
    console.log(bounds);  
  });
```

#### Arguments

- **sql**: a string with the sql query to calculate the bounds from.

### Application of getBounds in Leaflet and GMaps

You can use the getBounds results to center data on your maps using Leaflet and GMaps.

#### getBounds and Leaflet

sql.getBounds

```
sql.getBounds('select * from table').done(function(bounds) {  
    map.fitBounds(bounds);  
});
```

#### getBounds and GMaps V3

sql.getBounds

```
sql.getBounds('select * from table').done(function(bounds) {  
    var google_bounds = new google.maps.LatLngBounds();  
    google_bounds.extend(new google.maps.LatLng(bounds[0][0], bounds[0][1]));  
    google_bounds.extend(new google.maps.LatLng(bounds[1][0], bounds[1][1]));  
    map.fitBounds(google_bounds);  
});
```

### Core API functionallity

In case you are not using Google Maps or Leaflet or you want to implement your own layer object cartodb provide a way to get the tiles url for a layer definition.

If you want to use this functionallity you only need to load cartodb.core.js from our cdn, no css is needed:

Core API functionallity

```
<script src="http://libs.cartocdn.com/cartodb.js/v3/cartodb.core.js"></script>
```

An example using this functionality can be found in modestmaps example: [view live / source code](#).

Notice that cartodb.SQL is also included in that javascript file

## **cartodb.Tiles**

### **cartodb.Tiles.getTiles(layerOptions, callback)**

Fetch the tile template for the layerdefinition.

#### **Arguments**

- **layerOptions**: the data that defines the layer, it should contain at least user\_name and sublayer list. There are the available options:

cartodb.Tiles.getTiles

```
user_name: 'mycartodbuser',
sublayers: [{
  sql: "SELECT * FROM table_name";
  cartocss: '#layer { marker-fill: #F0F0F0; }'
}],
tiler_protocol: 'https', // not required
tiler_host: 'cartodb.com', // not required
tiler_port: 80 // not required
```

- **callback(tilesUrl, error)**: a function that receives the tiles templates. In case of an error the first param is null and second one is an object with errors attribute which is a list of errors. The tilesUrl object contains url template for tiles and for interactivity grids:

cartodb.Tiles.getTiles

```
{
  tiles: [
```

```

    "http://{s}.cartodb.com/HASH/{z}/{x}/{y}.png",
    ...
  ],
  grids: [
    // for each sublayer there is one entry on this array
    [
      "http://{s}.cartodb.com/HASH/0/{z}/{x}/{y}.grid.json"
    ],
    [
      "http://{s}.cartodb.com/HASH/1/{z}/{x}/{y}.grid.json"
    ],
    ...
  ]
}

```

**Example**

In this example a layer is created with one sublayer with renders all the content from table.

`cartodb.Tiles.getTiles`

```

var layerData = {
  user_name: 'mycartodbuser',
  sublayers: [{
    sql: "SELECT * FROM table_name";
    cartocss: '#layer { marker-fill: #F0F0F0; }'
  }]
};

cartodb.Tiles.getTiles(layerData, function(tiles, err) {
  if(tiler == null) {
    console.log("error: ", err.errors.join('\n'));
    return;
  }
  console.log("url template is ", tiles.tiles[0]);
}

```

```
}
```

## Versions

Keep in mind the version of CartoDB.js you are using for development. For any live code, we recommend you link directly to the tested CartoDB.js version from your development. You can find the version at anytime as follows:

### **cartodb.VERSION**

Contains the library version, should be something like '3.0.1'.

## Other important stuff

The CartoDB.js has many great features for you to use in your applications. Let's take a look at the most important for your application development.

### **Viz JSON support**

The Viz.JSON document tells CartoDB.js all the information about your map, including the style you want to use for your data and the filters you want to apply with SQL. The Viz JSON file is served with each map you create in your CartoDB account.

Although the Viz JSON file stores all your map settings, all the values are also easy to customize with CartoDB.js if you want to do something completely different than what you designed in your console. Loading the Viz JSON is as simple as:

Viz JSON support

```
cartodb.createVis('map', 'http://examples.cartodb.com/api/v2/viz/ne_10m_populated_p_1/viz.json')
```

### **Bounds wrapper**

We have added easy method to get the bounding box for any dataset or filtered query using the CartoDB.js library. The **getBounds** function can be useful for guiding users to the right location on a map or for loading only the right data at the right time based on user actions.

Bounds wrapper

```
var sql = new cartodb.SQL({ user: 'cartodb_user' });
```

```
sql.getBounds('SELECT * FROM table_name').done(function(bounds) {
```



```
console.log(bounds);  
});
```

### Event listener support

The CartoDB.js is highly asynchronous, meaning your application can get on with what it needs to do while the library efficiently does what you request in the background. This is useful for loading maps or getting query results. At the same time, we have made it very simple to add listeners and callbacks to the async portions of the library.

#### Loading events

The **createLayer** and **createVis** functions returns two important events for you to take advantage of: the first is **done**, which will let your code know that the library has successfully read the information from the Viz JSON and loaded the layer you requested. The second is 'error', which lets you know something did not go as expected when loading a requested layer:

#### Loading events

```
cartodb.createLayer(map, 'http://examples.cartodb.com/api/v1/viz/0001/viz.json')  
  .addTo(map)  
  .on('done', function(layer) {  
    alert('CartoDB layer loaded!');  
  }).on('error', function(err) {  
    alert("some error occurred: " + err);  
  });
```

#### Active layer events

The next important set of events for you to use happen on those layers that are already loaded (returned by the **done** event above). Three events are triggered by layers on your webpage, each requires the layer to include an **interactivity** layer. The first event is **featureClick**, which lets you set up events after the user clicks anything that you have mapped.

#### featureClick

```
layer.on('featureClick', function(e, latlng, pos, data, layer) {  
  console.log("mouse clicked polygon with data: " + data);
```

```
});
```

The second event is the **featureOver** event, which lets you listen for when the user's mouse is over a feature. Be careful, as these functions can get costly if you have a lot of features on a map.

featureOver

```
layer.on('featureOver', function(e, latlng, pos, data, layer) {  
  console.log("mouse over polygon with data: " + data);  
});
```

Similarly, there is the **featureOut** event. This is best used if you do things like highlighting polygons on mouseover and need a way to know when to remove the highlighting after the mouse has left.

featureOut

```
layer.on('featureOut', function(e, latlng, pos, data, layer) {  
  console.log("mouse left polygon with data: " + data);  
});
```

## Leaflet integration

If you want to use [Leaflet](#) it gets even easier, CartoDB.js handles loading all the necessary libraries for you! just include CartoDB.js and CartoDB.css in the HEAD of your website and you are ready to go! The CartoDB.css document isn't mandatory, however if you are making a map and are not familiar with writing your own CSS for the various needed elements, it can greatly help to jumpstart the process. Adding it is as simple as adding the main JavaScript library:

Leaflet integration

```
<link rel="stylesheet" href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.css" />  
<script src="http://libs.cartocdn.com/cartodb.js/v3/cartodb.js"></script>
```

## IE support

We have worked hard to support Internet Explorer with CartoDB.js. It currently works for IE7 through IE10. The biggest change you should note is that for the CSS you will

need to include an additional IE CSS document we have made available. Your tag should now house links to three documents, as follows:

### IE support

```
<link rel="stylesheet" href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.css" />
<!--[if lte IE 8]>
  <link rel="stylesheet" href="http://libs.cartocdn.com/cartodb.js/v3/themes/css/cartodb.ie.css" />
<![endif]->
<script src="http://libs.cartocdn.com/cartodb.js/v3/cartodb.js"></script>
```

### HTTPS support

You can use all the functionality of cartodb.js with HTTPs support. Be sure to add use https when importing both the JS library and the CSS file. Next, you will specify HTTPs for your Viz.JSON URL and as a parameter when you initialize your visualizaiton.

### HTTPS support

```
<div id="map"></div>
<script>
  var map = new L.Map('map', {
    center: [0,0],
    zoom: 2
  })
  cartodb.createLayer(map, 'http://examples.cartodb.com/api/v1/viz/15589/viz.json', { https: true })
    .addTo(map)
    .on('error', function(err) {
      alert("some error occurred: " + err);
    });
</script>
```

### Persistent version hosting

We are committed to making sure your website works as intended no matter what changes in the future. While we may find more efficient or more useful features to add to the library as time progresses. We never want to break things you have already

developed, for this reason, we make versioned CartoDB.js libraries available to you, meaning that the way they function will never unexpectedly change on you.

We recommend that you always develop against the most recent version of CartoDB.js, always found at:

Persistent version hosting

<http://libs.cartocdn.com/cartodb.js/v3/cartodb.js>

Anytime you wish to push a stable version of your site to the web though, you can find the version of CartoDB.js you are using by looking at the first line of the library, here:

Persistent version hosting

<http://libs.cartocdn.com/cartodb.js/v3/cartodb.js>

Or, by running the following in your code:

Persistent version hosting

```
alert(cartodb.VERSION)
```

Now, that you have your CartoDB.js version, you can point your site at that release. If the current version of CartoDB.js is 2.0.11, the URL would be:

Persistent version hosting

<http://libs.cartocdn.com/cartodb.js/v3/3.0.01/cartodb.js>

You can do the same for the CSS documents we provide:

Persistent version hosting

<http://libs.cartocdn.com/cartodb.js/v3/3.0.01/themes/css/cartodb.css>

- [Using CartoDB](#)
- [API overview](#)
- [CartoDB.js](#)

- [Getting started](#)
- [Using the library](#)
  - [Using CartoDB visualizations in your webpage](#)
  - [Creating visualizations at runtime](#)
- [Usage examples](#)
- [API methods](#)
  - [Visualization](#)
  - [cartodb.createVis](#)
  - [cartodb.Vis](#)
  - [vis.getLayers](#)
  - [vis.addOverlay](#)
  - [vis.getOverlay](#)
  - [vis.getOverlays](#)
  - [vis.getNativeMap](#)
  - [vis.Overlays](#)
  - [cartodb.createLayer](#)
  - [cartodb.CartoDBLayer](#)
  - [layer.clear](#)
  - [layer.hide](#)
  - [layer.show](#)
  - [layer.setOpacity](#)
  - [layer.getSubLayer](#)
  - [layer.createSubLayer](#)
  - [layer.invalidate](#)
  - [layer.setAuthToken](#)
  - [Returns](#)
  - [layer.setParams](#)
  - [Returns](#)
  - [cartodb.CartoDBLayer.SubLayer](#)
  - [sublayer.set](#)
  - [sublayer.get](#)
  - [sublayer.getSQL](#)
  - [sublayer.getCartoCSS](#)
  - [sublayer.setSQL](#)
  - [sublayer.setCartoCSS](#)
  - [sublayer.remove](#)
  - [sublayer.setInteraction](#)
  - [sublayer.show](#)
  - [sublayer.hide](#)
  - [sublayer.infowindow](#)
- [Events](#)
  - [sublayer.featureOver](#)
  - [layer.featureOut](#)
  - [layer.featureClick](#)
- [Specific UI functions](#)
  - [cartodb.geo.ui.Tooltip](#)
  - [cartodb.geo.ui.Tooltip.enable](#)
  - [cartodb.geo.ui.Tooltip.disable](#)

- [cartodb.geo.ui.InfoBox](#)
- [cartodb.geo.ui.InfoBox.enable](#)
- [cartodb.geo.ui.InfoBox.disable](#)
- [cartodb.geo.ui.Zoom](#)
- [cartodb.geo.ui.Zoom.show](#)
- [cartodb.geo.ui.Zoom.hide](#)
- [Getting data with SQL](#)
  - [cartodb.SQL](#)
  - [sql.execute](#)
  - [sql.getBounds](#)
  - [Application of getBounds in Leaflet and GMaps](#)
  - [getBounds and Leaflet](#)
  - [getBounds and GMaps V3](#)
- [Core API functionality](#)
  - [cartodb.Tiles](#)
  - [cartodb.Tiles.getTiles](#)
- [Versions](#)
  - [cartodb.VERSION](#)
- [Other important stuff](#)
  - [Viz JSON support](#)
  - [Bounds wrapper](#)
  - [Event listener support](#)
  - [Leaflet integration](#)
  - [IE support](#)
  - [HTTPS support](#)
  - [Persistent version hosting](#)
- [SQL API](#)
- [Advanced concepts](#)































































[Back](#)

## Further info? Take a look a the docs

Visit our support area and get some help from the community.

[Go to documentation](#)

### CartoDB is also used by

- 
- 
- 
- 
- 



- **Common questions**

- **Do you have any educational plans?**

Yes we have. [Contact us](#) for getting more information. We are quite friends of academics so, you will get a lot of benefits.

- **Need something extra? Let us know.**

Need us to help you with your visualization or application? Does your organization have unique requirements that don't quite fit our plans? [Contact us](#).

## CartoDB

- - CartoDB
  - [Visualize](#)
  - [Analyze](#)
  - [Develop](#)
  - [Pricing](#)
- - Enterprise
  - [Partners](#)
  - [Case studies](#)
  - [Contact sales](#)
- - Developers
  - [Support](#)
  - [Documentation](#)
  - [Tutorials](#)
  - [CartoDB at GitHub](#)
  - [Release notes](#)
- - More
  - [Blog](#)
  - [Attributions](#)
  - [About](#)
  - [Contact](#)

© 2014 CartoDB.

[Terms of service](#)