Article
Rubens Silva · Sep 27, 2017 4m read

Frontier: An abstraction layer for rapid REST development - Part 3 - Using the SQL API

Hello again and welcome to the **Part 3 - Using the SQL API!**

If you have been wondering about how to use SQL along with Frontier, you came to the right place. That's because since Frontier wraps the common Caché SQL API within it's own, you need to use the API provided from it. But you don't need to worry about its learning curve, because the Frontier SQL API is really simple.

If you arrived here without checking the Part 1, I'd recommend you doing so, because Part 1 covers the essential for getting started with Frontier routers.

This part covers the topic 6.

6. Using the SQL API

The SQL API is similar to what %SQL.Statement provides, with a single restriction: you can NOT use it to execute a query manually.

It's designed this way to make sure that each row is processed by Frontier's SQL serializer and also to prevent the developer from using different SQL APIs (%ResultSet) or restricting them to use queries only.

So keep in mind, that whenever you want to return a collection of data using SQL you MUST use the Frontier's SQL API.

However, if you only need to do some internal operation using SQL, you can use %ResultSet, %SQL.Statement or embedded SQL without issues.

Creating a simple dynamic query

In order to demonstrate what is possible to do with it, let's create a simple method that returns a TOP 3 list of Persons using a dynamic query.

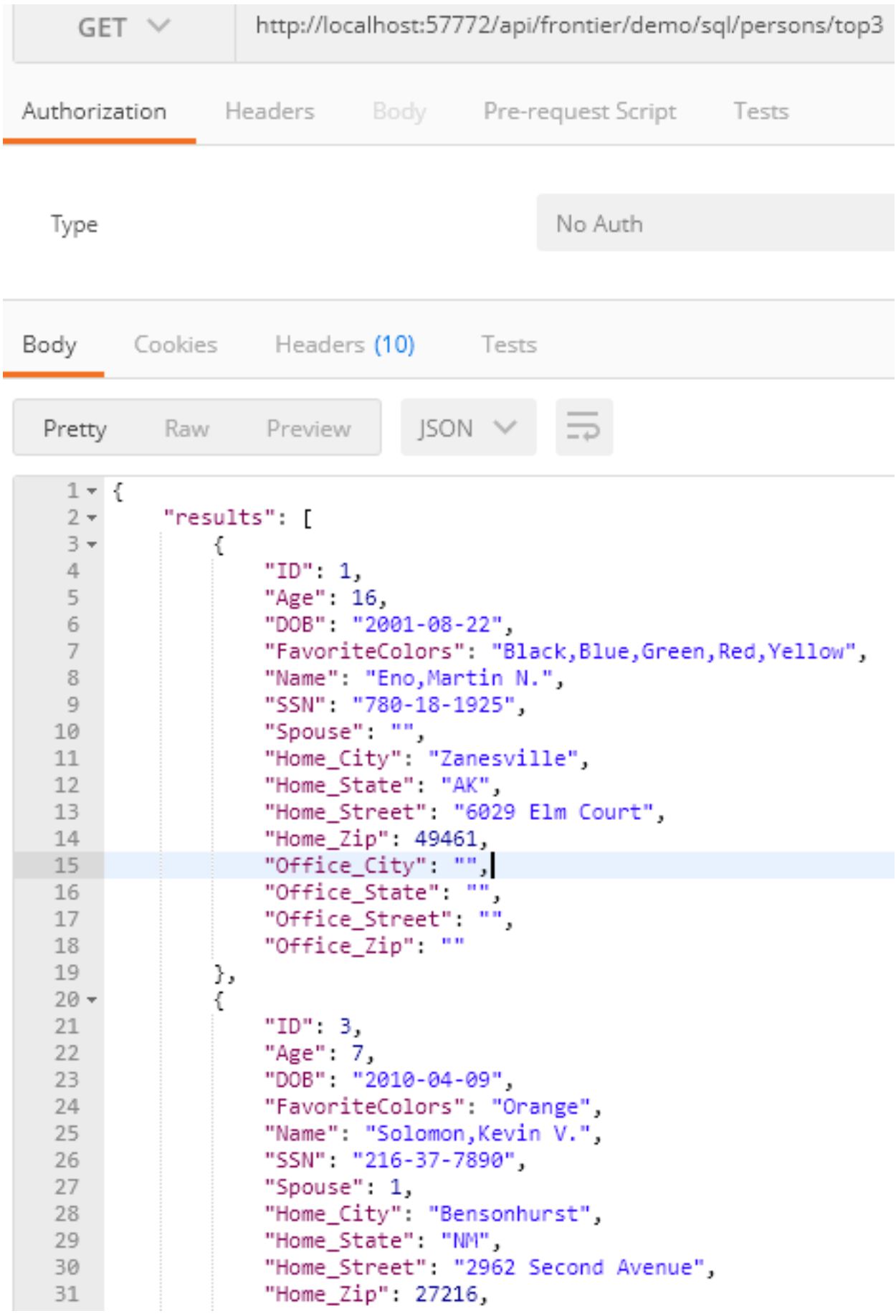
Add the route:

```
<Route Url="/sql/persons/top3" Method="GET" Call="GetPersons"></Route>
```

And the method for the Call.

```
ClassMethod GetPersons() As Frontier.SQL.Provider
{
  return %frontier.SQL.Prepare("SELECT TOP 3 * FROM SAMPLE.PERSON")
}
```

As we execute it, we can notice the result:



Very easy, right?

Overwriting the default container property

Notice that SQL results inserted are binded to that "results" property, by default this property serves as a container for SQL providers. If you want to override this behavior, simply put the result inside a %DynamicObject instance.

To see this feature in action, update the method to look like this:

```
ClassMethod GetPersons() As Frontier.SQL.Provider
{
  return {
    "persons": (%frontier.SQL.Prepare("SELECT TOP 3 * FROM SAMPLE.PERSON"))
  }
}
```

This way you can use a custom JSON structure.

Using cached queries

If you want to use a cached query, Frontier provides a way similar to the format used with %ResultSet. That is:

classname:query

Add a new route like this:

```
<Route Url="/sql/persons/namedquery" Method="GET" Call="GetPersonsUsingNamedQuery"></Route>
```

Then copy the method we created earlier and modify to use the syntax above.

```
ClassMethod GetPersonsUsingNamedQuery() As Frontier.SQL.Provider
{
  return {
    "persons": (%frontier.SQL.Prepare($classname()_":Top3Persons"))
  }
}
```

Now create the query, remember to flag it with SqlProc or it won't be available to use with the SQL engine.

```
Query Top3Persons() As %SQLQuery [ SqlProc ]
{
  SELECT TOP 3 * FROM SAMPLE.PERSON
}
```

Passing parameters to queries

Until this point, queries are not that much useful because they don't accept parameters. So to demonstrate how we can pass query parameters we will simulate a pagination mechanism using %VID.

In order to do that, create a new route:

```
<Route Url="/sql/persons/paginated" Method="GET" Call="GetPaginatedPersons"></Route>
```

And a method that receives arguments. We're going to use query parameters to delimit how much entries we want to fetch.

```
ClassMethod GetPaginatedPersons(
page As %Integer = 1,
rows As %Integer = 5) As Frontier.SQL.Provider
{
  set offset = (page * rows) - (rows - 1)
  set limit = page * rows

  return %frontier.SQL.Prepare("Frontier.Demo:PaginatedPersons").Parameters(offset, limit).Mode(2)
}
```

Notice that Parameters method, it's signature is similar to what %Execute does, but without actually executing it.

Mode is what we would call SelectMode or DisplayMode. You'll notice that the DOB field is formated.

And finally, we define a query to support it:

```
Query PaginatedPersons(
offset As %Integer = 1,
limit As %Integer = 5) As %SQLQuery [ SqlProc ]
{
  SELECT %VID as RowIndex, * from (SELECT * FROM SAMPLE.PERSON) WHERE %VID
BETWEEN :offset AND :limit
}
```

And we have a paginated query:

GET http://localhost:57772/api/frontier/demo/sql/persons/paginated?page=2&rows=3

Body Cookies Headers (10) Tests

Pretty Raw Preview JSON


```
{
    "results": [
        {
            "RowIndex": 4,
            "ID": 5,
            "Age": 3,
            "DOB": "03/24/2014",
            "FavoriteColors": "",
            "Name": "King,Stavros U.",
            "SSN": "873-14-3190",
            "Spouse": 1,
            "Home_City": "Xavier",
            "Home_State": "NJ",
            "Home_Street": "3349 Main Blvd",
            "Home_Zip": 18355,
            "Office_City": "Elmhurst",
            "Office_State": "ME",
            "Office_Street": "2318 Oak Place",
            "Office_Zip": 31899
        },
        {
            "RowIndex": 5,
            "ID": 6,
            "Age": 92,
            "DOB": "02/27/1925",
            "FavoriteColors": "",
            "Name": "Ott,Alexandra I.",
            "SSN": "516-98-8766",
            "Spouse": 1,
            "Home_City": "Fargo",
            "Home_State": "RI",
            "Home_Street": "4491 Second Place",
            "Home_Zip": 63827,
            "Office_City": "Ukiah",
            "Office_State": "CO",
            "Office_Street": "1192 Second Street",
            "Office_Zip": 32430
        },
        {
            "RowIndex": 6,
            "ID": 7,
```


Naturally, you can do the same using a dynamic query, just like before, but use ? instead of :.

This concludes the part 3, if you have any doubts please don't be afraid to post them below.

Next part we are going to learn how to share data between methods. Stay in touch!

#ObjectScript #REST API #Tutorial #Caché

Source URL:https://community.intersystems.com/post/frontier-abstraction-layer-rapid-rest-development-part-3-using-sql-api