Article
Dmitrii Kuznetsov · Feb 15, 2021 17m read

Four Database APIs

A concurrent session in IRIS: SQL, Objects, REST, and GraphQL


Kazimir Malevich, "Athletes" (1932)

> *"But of course you don't understand! How can a person who has always traveled in a horse-drawn carriage understand the feelings and impressions of the express traveler or the pilot in the air?"*
>
> Kazimir Malevich (1916)

Introduction

We ' ve already addressed the topic of why object/type representation is superior to SQL for implementing subject area models. And those conclusions and facts are as true now as they have ever been. So why should we take a step back and discuss technologies that drag abstractions back to the global level, where they had been in the pre-object and pre-type era? Why should we encourage the use of spaghetti code, which results in bugs that are hard to track down, and which relies only on virtuoso developer skills?

There are several arguments in favor of transmitting data via SQL/REST/GraphQL-based APIs as opposed to representing them as types/objects:

- These technologies are very well-studied and fairly easy to deploy.
- They enjoy incredible popularity, and have been widely implemented in accessible, open-source software.
- You often have no choice but to use these technologies, especially on the web and in databases.
- Most importantly, APIs still use objects, since they provide the most appropriate way of implementing the APIs in code.

Before we talk about implementing APIs, let's first take a look at the underlying abstraction layers. The diagram below shows how data moves between the place where it is permanently stored and the place where it is processed and presented to the user of our applications.

Today, data is stored either on spinning hard-disk drives (HDD) or, to use a more modern technology, in flash memory chips on an SSD. Data is written and read using a stream that consists of separate storage blocks on the HDD/SSD.

The division into blocks is not random. Rather, it is determined by the physics, mechanics, and electronics of the data storage medium. In an HDD, these are the tracks/sectors on a rotating

magnetic disk. In an SSD, these are the memory segments in a rewritable silicon chip.

The essence is the same: these are blocks of information, which we must find and put together in order to retrieve the pieces of data we need. Data must be assembled into structures that match our data model/type with the values that correspond to the time of the query. The DBMS bundled with the file subsystem in the operating system is responsible for the process of assembling and retrieving data.

We can bypass the DBMS by directly addressing to the file system or even the HDD/SSD. But then we lose two super important bridges to data: between the storage blocks and the file streams, and between the files and the ordered structure in the database model. In other words, we take responsibility for the development of all the code for processing blocks, files, and models, including all optimizations, careful debugging, and long-term reliability tests.

The DBMS gives us an excellent opportunity to deal with data in a high-level language, using understandable models and representations. This is one of the great advantages of these systems. DBMS and data platforms, such as InterSystems IRIS, offer even more: the ability to simultaneously access ordered data in a wide variety of ways. And it is up to you which to use in your project.

Let's take advantage of the variety of tools that IRIS gives us. Let's make the code prettier and cleaner. We will make direct use of the ObjectScript object-oriented language for utilizing and developing APIs. In other words, for example, we will call the SQL code directly from inside the ObjectScript software. For other APIs, we will use ready-made libraries and built-in ObjectScript tools.

We ' ll take our examples from the SQLZoo Internet project, which offers learning resources for SQL. We will use the same data in our other API examples.

> If you would like to get an overview of the variety of approaches to API design and take advantage of ready-made solutions, here is an interesting and useful collection of public APIs, which have been gathered into a single project on GitHub.

SQL

There is no more natural way to start than with SQL. Who here isn't familiar with it?

There is a huge volume of tutorials and books on SQL. We'll be relying on SQLZoo. This is a good beginner's SQL course with examples, walkthroughs, and a language reference.

Let's carry over some tasks from SQLZoo to the IRIS platform and solve them using a variety of methods.

> How quickly can you access InterSystems IRIS on your computer? One of the fastest options is to deploy a container in Docker from a ready-made InterSystems IRIS Community Edition image. InterSystems IRIS Community Edition is a free developer version of the InterSystems IRIS Data Platform.
> Other ways to access InterSystems IRIS Community Edition in the Learning Portal

To move the data from SQLZoo to our own IRIS instance storage.

To do this:

1. Open the management portal (mine, for example, at http://localhost:52773/csp/sys/UtilHome.csp),
2. Switch to the USER area - in Namespace %SYS click the “ Switch ” link and select USER
3. Go to System > SQL - open System Explorer, then SQL and click the “ Go ” button.
4. On the right side will open the tab "Execute query" with the button "Execute" - that's what we need.

To learn more about working with SQL through the management portal, see the documentation.

Have a look at ready-made scripts for deploying the database and SQLZoo test data set in the description in the Data section.

Here are a couple of direct links for the world table:

- A script that is used to create the world database
- The data that goes into that table

The script for creating the database can be executed in the Query Executor form in the IRIS management portal.

```
CREATE TABLE world(
   name VARCHAR(50) NOT NULL
   ,continent VARCHAR(60)
   ,area DECIMAL(10)
   ,population DECIMAL(11)
   ,gdp DECIMAL(14)
   ,capital VARCHAR(60)
   ,tld VARCHAR(5)
   ,flag VARCHAR(255)
   ,PRIMARY KEY (name)
)
```

To load the test set for the Query Executor form, go to the Wizards > Data Import menu. Note that the directory with the test data file must be added in advance, when you create the container, or loaded from your computer through the browser. This option is available in the control panel in the data import wizard.

Check whether the table with the data is present by running this script in the Query Executor form:

```
SELECT * FROM world
```

Now we can access the examples and tasks from the SQLZoo website. All of the examples below require that you implement an SQL query in the first assignment:

```
SELECT population
  FROM world
 WHERE name = 'France'
```

That way, you will be able to keep seamlessly working with the API by transferring tasks from SQLZoo to the IRIS platform.

Please note: as I ’ ve discovered, the data at the SQLZoo site interface is different from the exported data. At least in the first example, the values for the population of France and Germany differ. Don't get hung up about it. Use the Eurostat data for reference.

> Another convenient way to gain SQL access to the database in IRIS is the Visual Studio Code editor with the SQLTools plugin and the SQLTools Driver for InterSystems IRIS. This

solution is popular with developers — give it a spin.

In order to proceed smoothly to the next step and gain object access to our database, let's take a small detour from "pure" SQL queries to SQL queries embedded in the application code in ObjectScript, which is an object-oriented language built into IRIS.

How to set up IRIS access and develop in ObjectScript in VSCode.

```
Class User.worldquery
{
ClassMethod WhereName(name As %String)
{
    &sql(
        SELECT population INTO :population
          FROM world
         WHERE name = :name
    )

    IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg  QUIT}
    ELSEIF SQLCODE=100 {WRITE "Query returns no results"  QUIT}
    WRITE name, " ", population
}
}
```

Let's check the result in the terminal:

```
do ##class(User.worldquery).WhereName("France")
```

You should receive the name of the country and the number of inhabitants as a response.

Objects/Types

So, on to the REST/GraphQL story. We are implementing an API for web protocols. More often than not, we would have source code under the hood on the server side in a language that has good support for types, or even an entirely object-oriented paradigm. Here are some of the languages we are talking about: Spring in Java/Kotlin, Django in Python, Ruby on Rails, ASP.NET in C#, or Angular in TypeScript. And, needless to say, objects in ObjectScript, which is native to the IRIS platform.

Why is this important? The types and objects in your code will be simplified to data structures when they are sent out. You need to consider how models are simplified in the program, which is similar to taking into account the losses in relational models. You also need to make sure that, on the other side of the API, the models are adequately restored and can be used without any distortions. This presents an additional burden: an additional responsibility for you as a programmer. Outside the code and beyond the help of translators, compilers, and other automatic tools, you need to continuously ensure that models are correctly transferred.

If we look at the above issue from a different perspective, we don ’ t yet see any technologies

and tools on the horizon that can be used to easily transfer types/objects from a program in one language to a program in another. What is left? There are simplified implementations of SQL/REST/GraphQL, and an ocean of documentation describing the API in human-friendly language. Informal (from the computer ' s perspective) documentation for developers describes exactly what should be translated into formal code using all the available means, so the computer can process it.

Programmers constantly develop different approaches to solving the above problems. One of the successful approaches is the cross-language paradigm in the object DBMS of the IRIS platform.

The following table should help you understand the relationship between the OPP and SQL models in IRIS:

Object-oriented programming (OOP)	Structured query language (SQL)
Package	Schema
Class	Table
Property	Column
Method	Stored procedure
Relationship between two classes	Foreign key constraint, built-in join
Object (in memory or on disk)	Row (on disk)

You can learn more about displaying object and relational models from the IRIS documentation.

When executing our SQL query to create the world table from the example above, IRIS will automatically generate descriptions of the corresponding object in the class named User.world.

```
Class User.world Extends %Persistent [ ClassType = persistent, DdlAllowed, Final, Own
er = {_SYSTEM}, ProcedureBlock, SqlRowIdPrivate, SqlTableName = world ]

{

Property name As %Library.String(MAXLEN = 50) [ Required, SqlColumnNumber = 2 ];

Property continent As %Library.String(MAXLEN = 60) [ SqlColumnNumber = 3 ];

Property area As %Library.Numeric(MAXVAL = 9999999999, MINVAL = -9999999999, SCALE =
0) [ SqlColumnNumber = 4 ];

Property population As %Library.Numeric(MAXVAL = 99999999999, MINVAL = -99999999999,
SCALE = 0) [ SqlColumnNumber = 5 ];

Property gdp As %Library.Numeric(MAXVAL = 99999999999999, MINVAL = -99999999999999, S
CALE = 0) [ SqlColumnNumber = 6 ];

Property capital As %Library.String(MAXLEN = 60) [ SqlColumnNumber = 7 ];
```

```
Property tld As %Library.String(MAXLEN = 5) [ SqlColumnNumber = 8 ];

Property flag As %Library.String(MAXLEN = 255) [ SqlColumnNumber = 9 ];

Parameter USEEXTENTSET = 1;

/// Bitmap Extent Index auto-
generated by DDL CREATE TABLE statement.  Do not edit the SqlName of this index.

Index DDLBEIndex [ Extent, SqlName = "%%DDLBEIndex", Type = bitmap ];

/// DDL Primary Key Specification

Index WORLDPKey2 On name [ PrimaryKey, Type = index, Unique ];
}
```

This is a template you can use to develop your application in an object-oriented style. All you need to do is add methods to the class in ObjectScript, which has ready-made bundles for the database. In fact, the methods for this class are "stored procedures," to borrow the SQL terminology.

Let's try to implement the same example that we completed before using SQL. Add the WhereName method to the User.world class, which will play the role of the "Country information" object designer for the entered country name:

```
ClassMethod WhereName(name As %String) As User.world
{
    Set id = 1
    While ( ..%ExistsId(id) ) {
        Set countryInfo = ..%OpenId(id)
        if ( countryInfo.name = name ) { Return countryInfo }
        Set id = id + 1
    }
    Return countryInfo = ""
}
```

Check the following in the terminal:

```
set countryInfo = ##class(User.world).WhereName("France")


write countryInfo.name


write countryInfo.population
```

We can see from this example that, in order to find the desired object by country name, unlike in an SQL query, we need to manually sort through the database records one by one. In the worst-case scenario, if our object is at the end of the list (or is not there at all), we ' ll have to sort through all the records. There is a separate discussion about how you can speed up the search process by indexing object fields and autogenerating class methods in IRIS. You can read more in the documentation and in the posts in the developer community portal.

For example, for our class, knowing the IRIS generated index name from the WORLDPKey2 country name, you can initialize/design an object from the database using a single rapid query:

```
set countryInfo = ##class(User.world).WORLDPKey2Open("France")
```

Also check:

```
write countryInfo.name

write countryInfo.population
```

You can find some guidelines for deciding whether to use object or SQL access for stored objects in this documentation.

Of course, you should always keep in mind that you may only fully utilize one of them for your tasks.

Furthermore, thanks to the availability of ready-made binary bundles in IRIS that support common OOP languages, such as Java, Python, C, C# (.Net), JavaScript, and even Julia (see GitHub and OpenExchange), which is rapidly gaining popularity, you ' ll always be able to choose whatever language development tools that are most convenient for you.

Now let's dive into the discussion on data in the web API.

REST, or the RESTful Web API

Let's step outside the boundaries of the server and the familiar terminal and utilize some more mainstream interfaces: the browser and similar applications. These applications rely on the hypertext protocols from the HTTP family to manage interactions between systems. IRIS comes with a bunch of tools that are suited for this purpose, including an actual database server and the Apache HTTP server.

> Representational state transfer (REST) is an architectural style for designing distributed applications and, in particular, web applications. Though popular, REST is just a set of architectural principles, while SOAP is a standard protocol maintained by the World Wide Web Consortium (W3C), and thus technologies based on SOAP are backed up by a standard.

The global ID in REST is a URL, and it defines each successive information unit when exchanged with a database or a back-end application. See documentation for developing REST services in IRIS.

In our example, the base identifier will be something like the base from the address of the IRIS server, http://localhost:52773, and the /world/ path to our data that is a subdirectory of it. In particular, we are talking about our /world/France country directory.

It will look something like the following in a Docker container:

http://localhost:52773/world/France

If you are developing a complete application, be sure to check out the IRIS documentation recommendations. One of them is based on the REST API description in accordance with the OpenAPI 2.0 specification.

Let's do this the easy way, by implementing the API manually. In our example, we will create the simplest REST solution that requires just two steps in IRIS:

1. Create a class path monitor in the URL, which will inherit from the %CSP.REST system class
2. Add a call to our monitor class when configuring the IRIS web application

Step 1: Class Monitor

It should be clear how you can implement a class. Follow instructions in the documentation for creating a "manual" REST.

```
/// Description

Class User.worldrest Extends %CSP.REST

{

Parameter UseSession As Integer = 1;

Parameter CHARSET = "utf-8";

XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]

{

<Routes>

    <Route Url="/:name" Method="GET" Call="countryInfo" />

</Routes>

}

}
```

Make sure to include a handler method. It should perform exactly the same function as the calls in the terminal from the previous example:

```
ClassMethod countryInfo(name As %String) As %Status

{

    set countryInfo = ##class(User.world).WhereName(name)

    write "Country: ", countryInfo.name

    write "<br>"

    write "Population: ", countryInfo.population

    return $$$OK

}
```

As you can see, the parameter that starts with a colon, :name, is indicated to pass the name of the called handler method in the monitor from the incoming REST query to the parameters.

Step 2: Configuring the IRIS Web Application

Under System Administration > Security > Applications > Web Applications,

add a new web application with a URL entry address at /world and the following handler: our worldrest monitor class.

After it is configured, the web application should immediately respond when you go to http://localhost:52773/world/France. Keep in mind that the database is case sensitive, so the correct letter case must be used when transmitting the request data to the method parameter.

Lifehacks:

- Use the debugging tools if necessary. You can find a good description in this two-part article (check out the comments as well).
- If the error "401 Unauthorized" appears, and you are sure that the monitor class is on the server and that there are no errors in the link, try adding the %All role on the Application Roles tab in the web application settings. This is not an entirely secure method, and you need to understand the possible implications of allowing access for all roles, but it is acceptable for a local installation.

GraphQL

This is new territory in the sense that you won't find anything in the current IRIS documentation about APIs using GraphQL. However, this shouldn't stop us from using this wonderful tool.

> It's only five years since GraphQL has gone public. Developed by the Linux Foundation, GraphQL is a query language for APIs. And it is probably safe to say that this is the best technology that resulted from improvement of the REST architecture and the various web APIs. Here is a short introductory article about it for beginners. And, thanks to the efforts of InterSystems enthusiasts and engineers, IRIS has offered support for GraphQL since 2018.

Here is the related article "Implementing GraphQL for InterSystems Platforms". And here is GraphQL understood, explained, and implemented.

The GraphQL application consists of two modules: the back end of the application on the IRIS side and the front end part that runs in the browser. In other words, you need to configure it according to the instructions for the GraphQL and GraphiQL web application.

For example, this is how the application configuration looks for me in IRIS inside a Docker container. These are the settings for a GraphQL web application that acts as a REST monitor and a database schema handler:

And the second GraphQL application is a user interface for the browser, written in HTML and JavaScript:

It can be run by going to http://localhost:52773/graphiql/index.html.

Without any additional restrictive settings, the application will immediately pick up all the

database schemas it can find in the installation area. This means our examples will start working right away. Plus, the front end provides a wonderful organized array of hints from the available objects.

Here is an example of a GraphQL query for our database:

```
{
  User_world ( name: France ) {
    name
    population
  }
}
```

And here is the matching response:

```
{
  "data": {
    "User_world": [
      {
        "name": "France",
        "population": 65906000
      }
    ]
  }
}
```

This is what it looks like in the browser:

Summary

Technology name	Technology age	Query example
SQL	50 years	`SELECT population FROM world WHERE name = 'France'`

	Edgar F. Codd	
OOP	40 years Alan Kay and Dan Ingalls	`set countryInfo = ##class(User.world).WhereName("Fran`
REST	20 years Roy Thomas Fielding	http://localhost:52773/world/France
GraphQL	5 years Lee Byron	`{ User_world ( name: France ) {  name  population`

- There is a lot of energy being invested in such technologies as SQL, REST, and probably also GraphQL. There is also a lot of history behind them. They all play well with each other, within the IRIS platform, to create programs that handle data.
- Although it was not mentioned in this article, IRIS also supports other APIs, based on XML (SOAP) and JSON, that are well implemented.
- Unless you specifically take care of, for example, marshaling your objects, remember that data exchanged via API still represents an incomplete, stripped-down version of an object transfer. You as a developer (not the code) are responsible for ensuring the correct transfer of information about the data type of an object .

A Question for You, Dear Readers

The purpose of this article was not to just compare modern APIs, and not even to review the basic capabilities of IRIS. It was to help you see how easy it is to switch between APIs when accessing a database, take your first steps in IRIS, and obtain rapid results from your task.

That is why I would be very interested to know what you think:

- Does this type of approach help you get up and running with the software?
- Which process steps make it difficult to master the tools for working with the API in IRIS?
- Could you name an obstacle that you would not have been able to foresee?

If you know a user who is still learning how to use IRIS, please ask them to leave a comment below. The resulting discussion will be useful for all involved.

#API #Beginner #Data Model #Object Data Model #InterSystems IRIS