developerWorks®

Join IBM DB2 for i table with IBM Bluemix Weather web service record set

Use SQL to get weather data for each line of your DB2 for i table

Christophe Lalevée February 22, 2017

This article explains how to create a join between REST API (web service) and an IBM DB2 for i table. Examples provided in this article show how to get weather data for geolocalized stores from a retail database. IBM Bluemix Weather Company Data service provides APIs to retrieve weather data from The Weather Company (TWC).

Introduction

In today's world, APIs provide access to more and more data and capabilities beyond the firewall. APIs are sources of strategic value in today's digital economy.

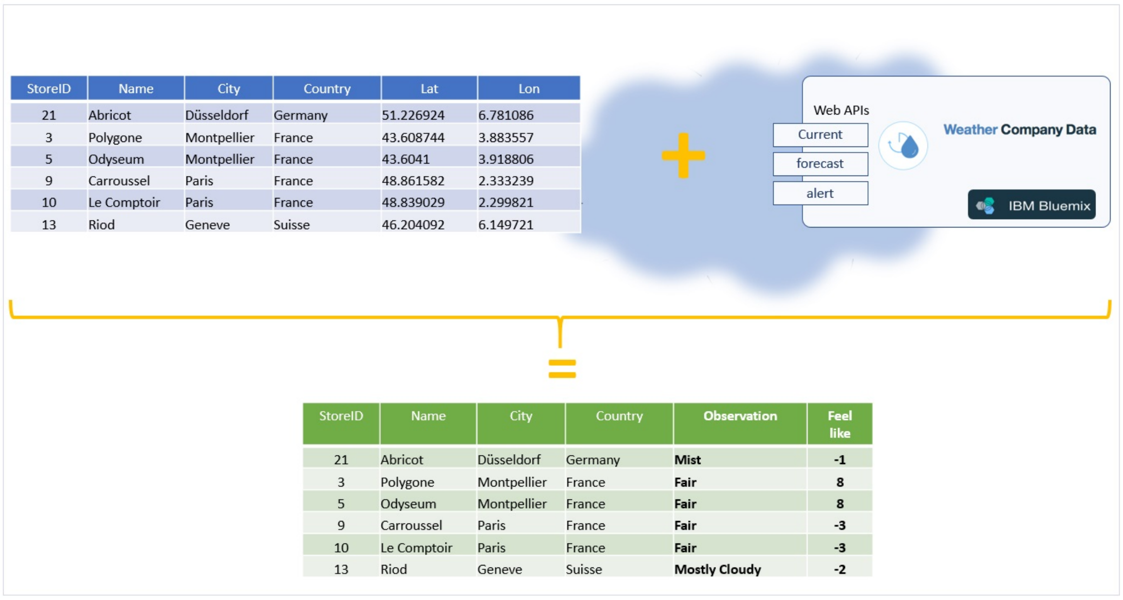
Weather is a good illustration of the added value of API to business: building a weather-driven solution for better decision making can solve business problems where weather has a significant impact on the outcome. For instance, a retail store can sell more on a sunny day, and so sales targets should be changed accordingly.

But, how difficult is it to integrate these web services with existing business data and applications? How can you join data from core business relational databases with data returned by web services?

IBM® DB2® for i provides all capabilities to easily develop a data centric solution to get access to APIs, and to make join between data returned as JSON and DB2 tables.

This article explains how to create a Weather Company Data service in IBM Bluemix® and provides examples to use it directly from DB2 for i, using `JSON_TABLE` and `httpGetCLOB` functions.

Figure 1. Overview of join between DB2 for i and Bluemix service



StoreID	Name	City	Country	Lat	Lon
21	Abricot	Düsseldorf	Germany	51.226924	6.781086
3	Polygone	Montpellier	France	43.608744	3.883557
5	Odyseum	Montpellier	France	43.6041	3.918806
9	Carroussel	Paris	France	48.861582	2.333239
10	Le Comptoir	Paris	France	48.839029	2.299821
13	Riod	Geneve	Suisse	46.204092	6.149721

StoreID	Name	City	Country	Observation	Feel like
21	Abricot	Düsseldorf	Germany	**Mist**	**-1**
3	Polygone	Montpellier	France	**Fair**	**8**
5	Odyseum	Montpellier	France	**Fair**	**8**
9	Carroussel	Paris	France	**Fair**	**-3**
10	Le Comptoir	Paris	France	**Fair**	**-3**
13	Riod	Geneve	Suisse	**Mostly Cloudy**	**-2**

What you'll need

To try out the examples and to create and run the DB2 for i SQL statements provided, you'll need:

- A Bluemix account (register for your free trial account or log in to Bluemix if you already have an account)
- A DB2 for i SQL editor connected to an IBM i release 7.2 minimum.
  For example, you can use IBM i Access Client Solutions (ACS) and SQL editor embedded.

Create Weather Company Data service instance

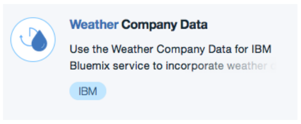
To be able to get weather data from DB2 for i, you need to create an instance of the Weather Company Data service in Bluemix. This service enables you to integrate weather data from The Weather Company® into your application.

Free plan allows you to make a maximum of 10 calls to The Weather Company per minute, and up to a maximum of 10,000 API calls for each Bluemix account. You can test the data in your applications without restrictions to geography supported (check Bluemix service to know countries or regions supported), forecast type or time series observations, with a restriction only on the number of calls. If you need more, you can subscribe to Base, Standard, or Premium plans.

Perform the following steps to create a Weather Company Data service instance in Bluemix:

1. Log in to Bluemix, https://console.ng.bluemix.net/
2. Select the region and space you want to work in.
3. Navigate to the catalog and click **Weather Company Data** (**Data & Analytics** category)

Figure 2. Weather Company Data service in Bluemix catalog



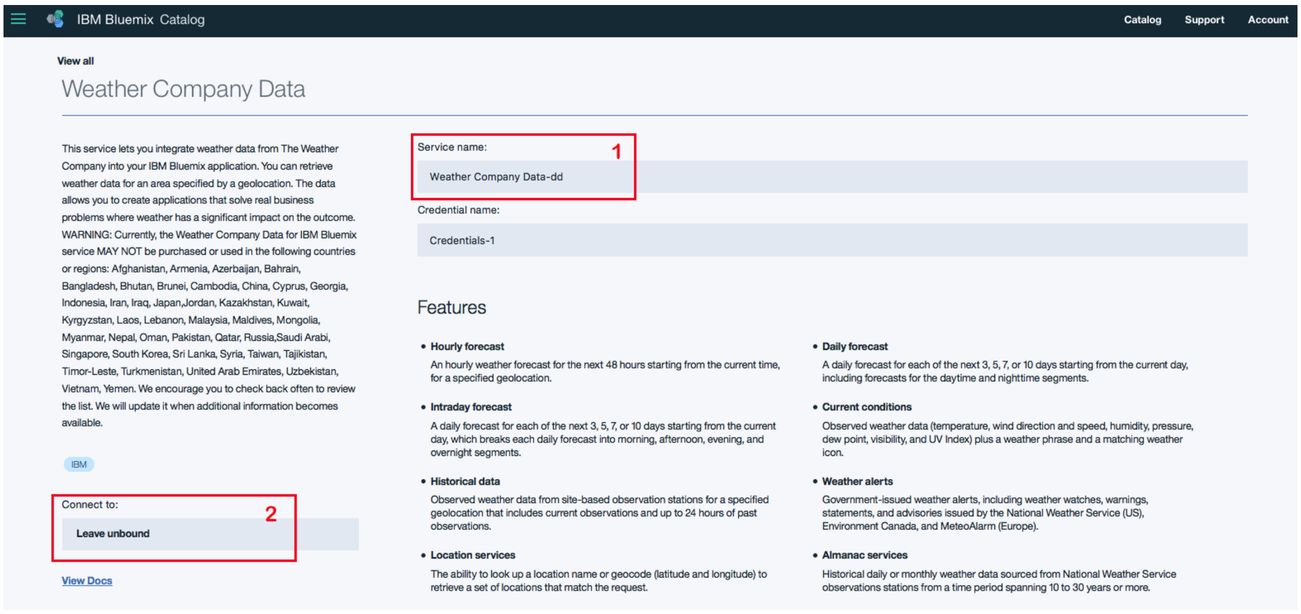
Weather Company Data

Use the Weather Company Data for IBM Bluemix service to incorporate weather

IBM

4. Create an instance.
   In the **Service Name** field, enter a new service name or keep the one proposed, like **Weather Company Data-dd** as in this example. In the **Connect to** field, retain the text, **Leave Unbound**, as we will not bind this service to a Bluemix application.

Figure 3. Weather Company Data service creation page



Weather Company Data

This service lets you integrate weather data from The Weather Company into your IBM Bluemix application. You can retrieve weather data for an area specified by a geolocation. The data allows you to create applications that solve real business problems where weather has a significant impact on the outcome. WARNING: Currently, the Weather Company Data for IBM Bluemix service MAY NOT be purchased or used in the following countries or regions: Afghanistan, Armenia, Azerbaijan, Bahrain, Bangladesh, Bhutan, Brunei, Cambodia, China, Cyprus, Georgia, Indonesia, Iran, Iraq, Japan,Jordan, Kazakhstan, Kuwait, Kyrgyzstan, Laos, Lebanon, Malaysia, Maldives, Mongolia, Myanmar, Nepal, Oman, Pakistan, Qatar, Russia,Saudi Arabi, Singapore, South Korea, Sri Lanka, Syria, Taiwan, Tajikistan, Timor-Leste, Turkmenistan, United Arab Emirates, Uzbekistan, Vietnam, Yemen. We encourage you to check back often to review the list. We will update it when additional information becomes available.

IBM

Connect to: Leave unbound 2

View Docs

Service name: Weather Company Data-dd 1

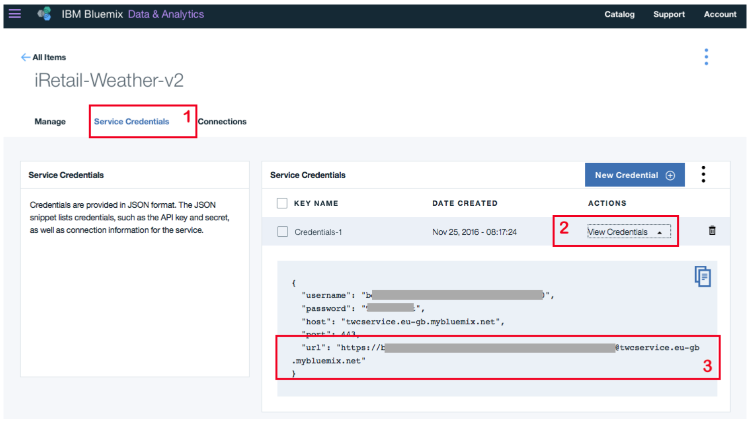
Credential name: Credentials-1

Features

- **Hourly forecast**
  An hourly weather forecast for the next 48 hours starting from the current time, for a specified geolocation.
- **Intraday forecast**
  A daily forecast for each of the next 3, 5, 7, or 10 days starting from the current day, which breaks each daily forecast into morning, afternoon, evening, and overnight segments.
- **Historical data**
  Observed weather data from site-based observation stations for a specified geolocation that includes current observations and up to 24 hours of past observations.
- **Location services**
  The ability to look up a location name or geocode (latitude and longitude) to retrieve a set of locations that match the request.
- **Daily forecast**
  A daily forecast for each of the next 3, 5, 7, or 10 days starting from the current day, including forecasts for the daytime and nighttime segments.
- **Current conditions**
  Observed weather data (temperature, wind direction and speed, humidity, pressure, dew point, visibility, and UV Index) plus a weather phrase and a matching weather icon.
- **Weather alerts**
  Government-issued weather alerts, including weather watches, warnings, statements, and advisories issued by the National Weather Service (US), Environment Canada, and MeteoAlarm (Europe).
- **Almanac services**
  Historical daily or monthly weather data sourced from National Weather Service observations stations from a time period spanning 10 to 30 years or more.

5. Click **Create**.
6. Retrieve the credentials.
   In the Bluemix dashboard, retrieve the service created, and click it. Next, click **ServiceCredentials** and then click **ViewCredentials**. Then, copy the URL parameter as it will be used later to call your own instance of weather from SQL statements.

Figure 4. Weather Company Data service credentials



Service Credentials

Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

Service Credentials

KEY NAME	DATE CREATED	ACTIONS
Credentials-1	Nov 25, 2016 - 08:17:24	View Credentials

```
{
  "username": "b[redacted]",
  "password": "[redacted]",
  "host": "twcservice.eu-gb.mybluemix.net",
  "port": 443,
  "url": "https://b[redacted]@twcservice.eu-gb
.mybluemix.net"
}
```

Also, remember the user name and password as you might need it to test online Weather Company Data APIs.

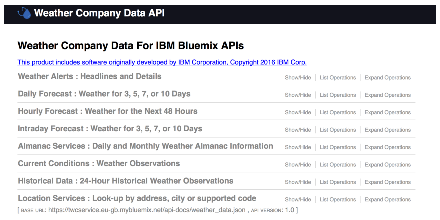
Weather Company Data APIs

Weather data are available through Representational State Transfer (REST) APIs.

You can have access to current conditions, forecasts (daily, hourly, intraday, and so on), historical data, or weather alert.

All APIs are documented and can be tested in the https://twcservice.mybluemix.net/rest-api/ website if you have created your Weather Company Data service in the US south region (otherwise, use https://twcservice.eu-gb.mybluemix.net/rest-api or https://twcservice.au-syd.mybluemix.net/rest-api).

Figure 5. Weather Company Data service list of APIs



Weather Company Data For IBM Bluemix APIs

This product includes software originally developed by IBM Corporation, Copyright 2016 IBM Corp.

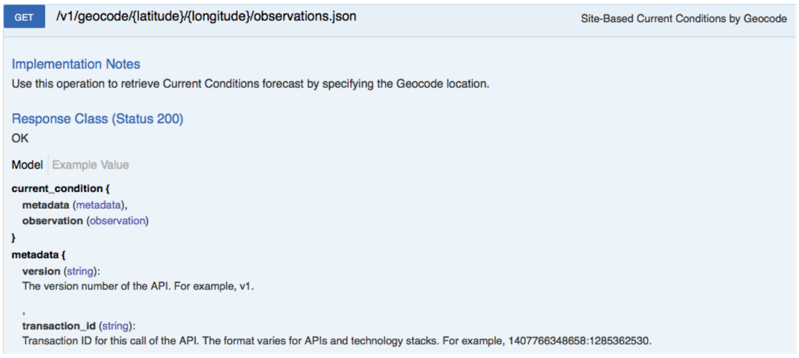
Weather Alerts : Headlines and Details	Show/Hide	List Operations	Expand Operations
Daily Forecast : Weather for 3, 5, 7, or 10 Days	Show/Hide	List Operations	Expand Operations
Hourly Forecast : Weather for the Next 48 Hours	Show/Hide	List Operations	Expand Operations
Intraday Forecast : Weather for 3, 5, 7, or 10 Days	Show/Hide	List Operations	Expand Operations
Almanac Services : Daily and Monthly Weather Almanac Information	Show/Hide	List Operations	Expand Operations
Current Conditions : Weather Observations	Show/Hide	List Operations	Expand Operations
Historical Data : 24-Hour Historical Weather Observations	Show/Hide	List Operations	Expand Operations
Location Services : Look-up by address, city or supported code	Show/Hide	List Operations	Expand Operations

[BASE URL: https://twcservice.eu-gb.mybluemix.net/api-docs/weather_data.json , API VERSION: 1.0]

The APIs are well documented online and include information you might need about the method, path, parameters, data returned, response message, and so on.

In the following figure, you can see the API path and model of data returned, if return code = 200 (OK).

Figure 6. Current conditions API documentation



GET /v1/geocode/{latitude}/{longitude}/observations.json Site-Based Current Conditions by Geocode

Implementation Notes

Use this operation to retrieve Current Conditions forecast by specifying the Geocode location.

Response Class (Status 200)

OK

Model | Example Value

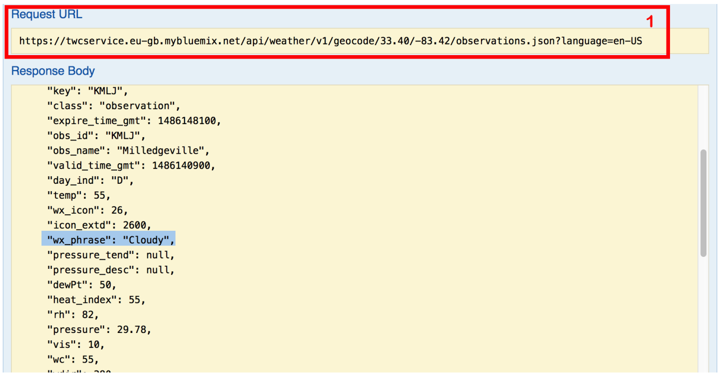
```
current_condition {
  metadata (metadata),
  observation (observation)
}
metadata {
  version (string):
  The version number of the API. For example, v1.

  ,
  transaction_id (string):
  Transaction ID for this call of the API. The format varies for APIs and technology stacks. For example, 1407766348658:1285362530.
```

Using the user name and password from your Weather Company Data service credentials, you can test all the APIs and see the data returned.

Here is the result of the online test, providing the current weather observations for a given latitude and longitude.

Figure 7. Example of current conditions API online test



Request URL

1

```
https://twcservice.eu-gb.mybluemix.net/api/weather/v1/geocode/33.40/-83.42/observations.json?language=en-US
```

Response Body

```
     "key": "KMLJ",
     "class": "observation",
     "expire_time_gmt": 1486148100,
     "obs_id": "KMLJ",
     "obs_name": "Milledgeville",
     "valid_time_gmt": 1486140900,
     "day_ind": "D",
     "temp": 55,
     "wx_icon": 26,
     "icon_extd": 2600,
     "wx_phrase": "Cloudy",
     "pressure_tend": null,
     "pressure_desc": null,
     "dewPt": 50,
     "heat_index": 55,
     "rh": 82,
     "pressure": 29.78,
     "vis": 10,
     "wc": 55,
```

We will use the URL in the **Request URL** field in the SQL requests of examples 1 and 2 later in this article.

```
        https://twcservice.eu-gb.mybluemix.net/api/weather/v1/geocode/<LAT>/<LON>/observations.json?
language=en-US
```

Weather Company Data service is now instanced and we can call its APIs using our own credentials. We are ready to create the first SQL request to call APIs from DB2 for i.

Calling REST web service using the httpGetCLOB function

To retrieve data from the Weather Company Data web service, use the DB2 for i `httpGetCLOB` SQL function. `httpGetCLOB` is a SYSTOOLS scalar function that uses the `GET` method to retrieve a representation of a resource as a character large object (CLOB).

Table 1. httpGetCLOB function parameters

Name	HTTP method	Return type	Parameters	Parameter type
httpGetCLOB	GET	CLOB(2G)	URL	VARCHAR(2048)

			HTTPHEADER	CLOB(10K)

The *httpGetCLOB* function admits two input parameters, **URL** and **HTTPHEADER**, and returns one value.

URL (input)

The `URL` parameter contains the full path to the weather data, including authentication data (your user name and password) and weather parameters.

For instance:

```
        https://<username>:<password>@twcservice.eu-gb.mybluemix.net/api/weather/v1/geocode/<LAT>/<LON>/
observations.json?language=en-US
```

This URL permits to get the current weather conditions for a location identified by <LAT>/<LON>, from Weather Company Data service hosted in Bluemix Europe (London = eu-gb.mybluemix.net) for user <username>.

HTTPHEADER (input)

The value of the `HTTPHEADER` parameter can be `NULL` or an empty string, which causes the default properties to be used. An explicit header needs to be supplied only when non-default properties must be used or when the header fields must be sent to the server.

To query the Weather Company Data API, use the default property, that is, `NULL`).

Return value

The Weather Company Data API returns weather data in a JSON object. A JSON object is represented as a list of key-value pairs. This object format depends on the APIs called. Refer to the online documentation to know about the JSON format you will get for each API.

If you need additional information about how to use this function, you can read the IBM DeveloperWorks® article "Incorporate web services into your SQL queries, Introducing HTTP functions on IBM DB2® for i", by Yi Yuan and Nick Lawrence.

DB2 for i HTTP functions are available on IBM DB2 for i since release 7.1 with DB2 PTF Group SF99701Level 23 and Java™ 1.6 or later (5761-JV1 Option 11, 12, 14, or 15).

Parsing JSON object using JSON_TABLE function

The JSON object retrieved with `httpGetCLOB` needs to be parsed to be used in SQL. This is done using the `JSON_TABLE` function.

The `JSON_TABLE` table function returns a result table from the evaluation of SQL/JSON path expressions. Each item in the result sequence of the row SQL/JSON path expression represents one or many rows in the result table.

`JSON_TABLE` is available on IBM DB2 for i for releases 7.2 and 7.3. It is provided with the 7.2 Database Group PTF SI99702 level 14 and the 7.3 Database Group PTF SI99703 level 3, which were released in November 2016.

If you need additional information about how to use this function, you can read the IBM DeveloperWorks article "The powerful `JSON_TABLE` function: Utilize JSON information with IBM DB2 for i" by John Eberhard.

Examples

In the following examples, we will use a sample DB2 table (named, `storebot.store`) from a retail demo.

If needed, you can create it with the following SQL statement:

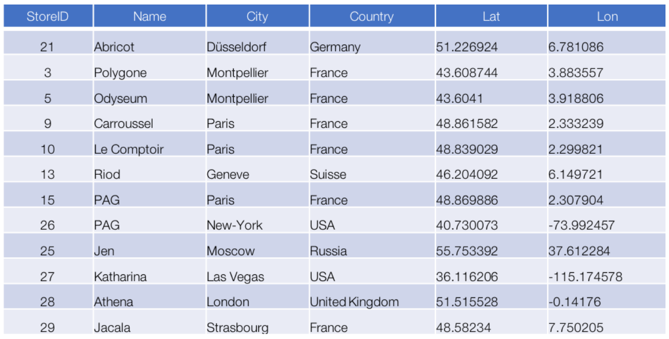
```
          CREATE TABLE STOREBOT.STORE (
            STOREID INTEGER GENERATED ALWAYS AS IDENTITY (
               START WITH 1 INCREMENT BY 1
               NO MINVALUE NO MAXVALUE
               NO CYCLE NO ORDER
               CACHE 20),
            NAME VARCHAR(50) DEFAULT NULL,
            ADDRESS1 VARCHAR(50) DEFAULT NULL,
            ADDRESS2 VARCHAR(50) DEFAULT NULL,
            POSTALCODE VARCHAR(10) DEFAULT NULL,
            CITY VARCHAR(50) DEFAULT NULL,
            COUNTRY VARCHAR(50) DEFAULT NULL,
            LAT DOUBLE PRECISION DEFAULT NULL,
            LON DOUBLE PRECISION DEFAULT NULL
        )
```

You can add rows in this table using SQL INSERT statements. For example, if you want to insert two stores, located in Montpellier (France) and Rochester (MN, USA):

```
        INSERT INTO STOREBOT.STORE
            (NAME, ADDRESS1, ADDRESS2, POSTALCODE, CITY, COUNTRY, LAT, LON)
            VALUES('Store_1','Rue de la Vieille Poste', NULL, '34000', 'Montpellier',
                   'France', 43.614934, 3.908162);
        INSERT INTO STOREBOT.STORE
            (NAME, ADDRESS1, ADDRESS2, POSTALCODE, CITY, COUNTRY, LAT, LON)
            VALUES('Store_2', '37th ST NW', NULL, 'MN 55901', ' Rochester',
                   USA, 44.061604, -92.504532);
        COMMIT;
```

Browsing table, we can see that for each store, we have latitude and longitude, as requested by the Weather Company Data API we want to use.

Figure 8. storebot.store table content (sample)



StoreID	Name	City	Country	Lat	Lon
21	Abricot	Düsseldorf	Germany	51.226924	6.781086
3	Polygone	Montpellier	France	43.608744	3.883557
5	Odyseum	Montpellier	France	43.6041	3.918806
9	Carroussel	Paris	France	48.861582	2.333239
10	Le Comptoir	Paris	France	48.839029	2.299821
13	Riod	Geneve	Suisse	46.204092	6.149721
15	PAG	Paris	France	48.869886	2.307904
26	PAG	New-York	USA	40.730073	-73.992457
25	Jen	Moscow	Russia	55.753392	37.612284
27	Katharina	Las Vegas	USA	36.116206	-115.174578
28	Athena	London	United Kingdom	51.515528	-0.14176
29	Jacala	Strasbourg	France	48.58234	7.750205

Let's start with the ACS SQL editor. Of course, you can use any other DB2 for i compatible SQL editor.

Example 1: Getting current weather condition from DB2 for i (no join with DB2 table)

In this first example, we will call the Weather Company Data API to get the current conditions for a given latitude and longitude. No join yet with a DB2 table.

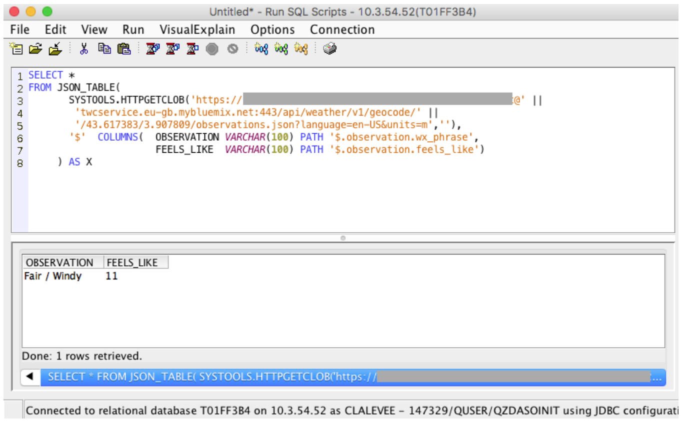
We can get the current weather condition using the SQL statement shown in Listing 1.

Listing 1. Current weather condition with given latitude and longitude

```
        SELECT *
        FROM JSON_TABLE(
                 SYSTOOLS.HTTPGETCLOB('https://<username>:<password>@' ||
                         'twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode/' ||
                         '/43.617383/3.907809/observations.json?language=en-US&units=m',''),
                 '$'
                 COLUMNS( OBSERVATION VARCHAR(100) PATH '$.observation.wx_phrase',
                                   FEELS_LIKE VARCHAR(100) PATH '$.observation.feels_like')
             ) AS X
```

In this statement, the `SYSTOOLS.httpGetCLOB` function is used to retrieve the JSON object from the Weather Company Data service API, for one geolocation only, specified directly in the service URI (Latitude=43.617383, Longitude=3.907809). This object is then used as the input in the `JSON_TABLE` function.

Figure 9. ACS SQL editor: get current conditions for a given latitude and longitude



Untitled* - Run SQL Scripts - 10.3.54.52(T01FF3B4)

```
SELECT *
FROM JSON_TABLE(
       SYSTOOLS.HTTPGETCLOB('https://[redacted]@' ||
        'twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode/' ||
        '/43.617383/3.907809/observations.json?language=en-US&units=m',''),
       '$'  COLUMNS(  OBSERVATION VARCHAR(100) PATH '$.observation.wx_phrase',
                      FEELS_LIKE  VARCHAR(100) PATH '$.observation.feels_like')
     ) AS X
```

```
OBSERVATION   FEELS_LIKE
Fair / Windy  11
```

Done: 1 rows retrieved.

SELECT * FROM JSON_TABLE(SYSTOOLS.HTTPGETCLOB('https://[redacted]...

Connected to relational database T01FF3B4 on 10.3.54.52 as CLALEVEE - 147329/QUSER/QZDASOINIT using JDBC configurati

`JSON_TABLE` admits the following three parameters:

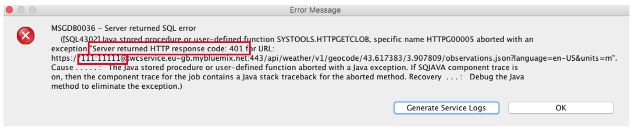
- A JSON object: In this example, the JSON object can be retrieved using the `SYSTOOLS.httpGetCLOB` function calling Weather Company Data service API
- A path: It is the expression used to locate information within the JSON object.
- Column list: It includes the column names, the data types, and the SQL/JSON path to the JSON values we want to extract from the JSON object. In this example, we retrieve only two columns, `Observation` and `Feels_like` (*perceived* temperature).

In case of an error during the call of API, you will get a service error message from `SYSTOOLS.httpGetCLOB`.

Examples:

- HTTP return code = 401: Authentication error (wrong ID and password)

Figure 10. Weather Company Data API return code 401

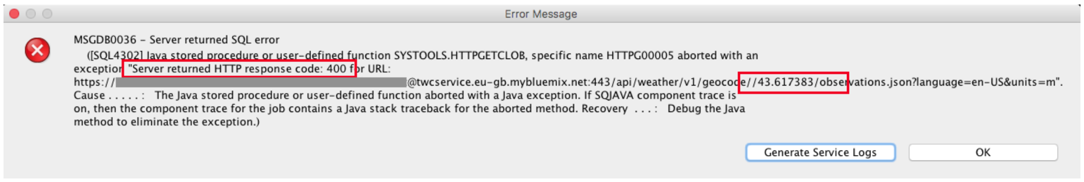


Error Message

```
MSGDB0036 - Server returned SQL error
  ([SQL4302] Java stored procedure or user-defined function SYSTOOLS.HTTPGETCLOB, specific name HTTPG00005 aborted with an
exception "Server returned HTTP response code: 401 for URL:
https://111:11111@twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode/43.617383/3.907809/observations.json?language=en-US&units=m".
Cause . . . . . :   The Java stored procedure or user-defined function aborted with a Java exception. If SQJAVA component trace is
on, then the component trace for the job contains a Java stack traceback for the aborted method. Recovery  . . . :   Debug the Java
method to eliminate the exception.)
```

- HTTP return code = 400: Bad Request, syntax error (longitude parameter is missing)

Figure 11. Weather Company Data API return code 400

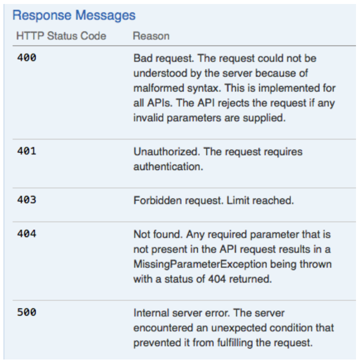


Error Message

```
MSGDB0036 - Server returned SQL error
  ([SQL4302] Java stored procedure or user-defined function SYSTOOLS.HTTPGETCLOB, specific name HTTPG00005 aborted with an
exception "Server returned HTTP response code: 400 for URL:
https://[redacted]@twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode//43.617383/observations.json?language=en-US&units=m".
Cause . . . . . :   The Java stored procedure or user-defined function aborted with a Java exception. If SQJAVA component trace is
on, then the component trace for the job contains a Java stack traceback for the aborted method. Recovery  . . . :   Debug the Java
method to eliminate the exception.)
```

You can find the list of APIs error codes on the Weather Company Data API website.

Figure 12. List of Weather Company Data API return codes



Response Messages

HTTP Status Code	Reason
400	Bad request. The request could not be understood by the server because of malformed syntax. This is implemented for all APIs. The API rejects the request if any invalid parameters are supplied.
401	Unauthorized. The request requires authentication.
403	Forbidden request. Limit reached.
404	Not found. Any required parameter that is not present in the API request results in a MissingParameterException being thrown with a status of 404 returned.
500	Internal server error. The server encountered an unexpected condition that prevented it from fulfilling the request.

Example 2: Getting current weather condition from DB2 for i (join with DB2 table)

In the previous example, latitude and longitude parameters are constants.

In this second example, we want to call this Weather Company Data API for each selected row from a SQL request, something like a join between a DB2 for i table and the web service.

We must write a join SQL request using `STOREBOT.STORE` as the primary table, and the same `JSON_TABLE` that we mentioned before (Example 1) as the secondary table.

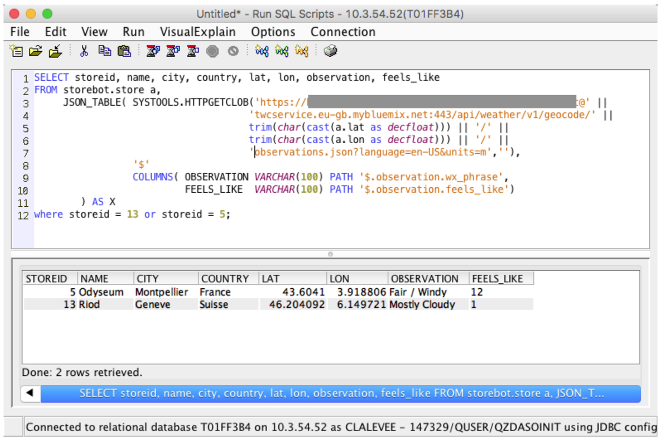
But, where is the join condition? In the Weather Company Data API call, we need to replace latitude and longitude constants by columns `lat` and `lon` (as shown in Listing 2).

Listing 2. Current weather conditions for stores in DB2 table

```
SELECT storeid, name, city, country, lat, lon, observation, feels_like
FROM storebot.store A,
     JSON_TABLE (
              SYSTOOLS.HTTPGETCLOB('https://<username>:<password>@' ||
                                   'twcservice.eu-gb.mybluemix.net:443' ||
                                   '/api/weather/v1/geocode/' ||
                                   trim(char(cast(a.lat as decfloat))) || '/' ||
                                   trim(char(cast(a.lon as decfloat))) || '/' ||
                                   'observations.json?language=en-US&units=m',''),
             '$'
             COLUMNS( OBSERVATION VARCHAR(100) PATH '$.observation.wx_phrase',
                      FEELS_LIKE VARCHAR(100) PATH '$.observation.feels_like')
        ) AS X
WHERE storeid = 13
OR storeid = 5;
```

You may notice that we have to cast these columns (`trim(char(cast(a.lat as decfloat)))`) to match the format and data type required by the web service, regardless of the format and data type of the columns in DB2. In the API call, it will be used as character (URI), included in path.

Figure 13. ACS SQL editor: get current conditions for each selected row from a SQL request


Untitled* - Run SQL Scripts - 10.3.54.52(T01FF3B4)
File Edit View Run VisualExplain Options Connection

```
SELECT storeid, name, city, country, lat, lon, observation, feels_like
FROM storebot.store a,
     JSON_TABLE( SYSTOOLS.HTTPGETCLOB('https://[redacted]@' ||
                                      'twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode/' ||
                                      trim(char(cast(a.lat as decfloat))) || '/' ||
                                      trim(char(cast(a.lon as decfloat))) || '/' ||
                                      'observations.json?language=en-US&units=m',''),
                  '$'
                  COLUMNS( OBSERVATION VARCHAR(100) PATH '$.observation.wx_phrase',
                           FEELS_LIKE  VARCHAR(100) PATH '$.observation.feels_like')
          ) AS X
where storeid = 13 or storeid = 5;
```

```
STOREID NAME     CITY        COUNTRY LAT       LON      OBSERVATION   FEELS_LIKE
      5 Odyseum  Montpellier France    43.6041 3.918806 Fair / Windy  12
     13 Riod     Geneve      Suisse  46.204092 6.149721 Mostly Cloudy 1

Done: 2 rows retrieved.
SELECT storeid, name, city, country, lat, lon, observation, feels_like FROM storebot.store a, JSON_T...
Connected to relational database T01FF3B4 on 10.3.54.52 as CLALEVEE - 147329/QUSER/QZDASOINIT using JDBC config
```

Be aware that the plan (Free, Standard, or Premium) you subscribed for the Weather Company Data service in Bluemix has limitations in terms of the number of calls. Depending on the number of rows returned by the `SELECT` statement and the number of requests you run, you might need to switch plans.

Example 3: Getting 3 days forecast weather conditions from DB2 for i (inner join with DB2 table)

The `JSON_TABLE` function allows you to retrieve multiple rows using the array elements embedded in the JSON object: a nested column definition is then used in the `JSON_TABLE` function.

We can use it to retrieve the weather for the next three days (current day + three days).

If we look at the Weather Company Data APIs documentation, we can see the JSON structure for the current day + three days forecast:

```
{
 "metadata": {
      . . .
 },
 "forecasts": [                                          <- “forecasts” = Table of Days
   {                                                    <- Day 1 (current)
     "class": "fod_long_range_daily",
     "expire_time_gmt": 1486490521,
     "fcst_valid": 1486468800,
     "fcst_valid_local": "2017-02-07T07:00:00-0500",
     "num": 1,
     "max_temp": 64,
     "min_temp": 54,
     "torcon": null,
     "stormcon": null,
     "blurb": null,
     "blurb_author": null,
     "lunar_phase_day": 11,
     "dow": "Tuesday",
     "lunar_phase": "Waxing Gibbous",
     "lunar_phase_code": "WXG",
     "sunrise": "2017-02-07T07:29:26-0500",
     "sunset": "2017-02-07T18:14:58-0500",
     "moonrise": "2017-02-07T15:05:27-0500",
     "moonset": "2017-02-07T04:25:08-0500",
     "qualifier_code": null,
     "qualifier": null,

 "narrative": "Thunderstorms, some may contain heavy rain.",
     "qpf": 0.66,
     "snow_qpf": 0,
     "snow_range": "",
     "snow_phrase": "",
     "snow_code": "",
     "night": {. . .}                                  <- “night” = night data of day 1
     "day": {. . .}                                       <- “day” = day data of day 1
    }
   {}                                                  <- Day 2
   {}                                                  <- Day 3
   {}                                                  <- Day 4
 ]
}
```

Looking at the day data of a day, we can see the data we want to display for each store, and we must declare in the `COLUMNS` section of `JSON_TABLE`.

```
  . . .
  "night": {. . .}
  "day": {
        "fcst_valid": 1486468800,
        "fcst_valid_local": "2017-02-07T07:00:00-0500",                       <- Date
        "day_ind": "D",
        "thunder_enum": 2,
        "daypart_name": "Today",
        "long_daypart_name": "Tuesday",
        "alt_daypart_name": "Today",
        "thunder_enum_phrase": "Thunder expected",
        "num": 1,
        "temp": 64,                                                            <- Temperature
        "hi": 63,
        "wc": 60,
        "pop": 90,
        "icon_extd": 402,
        "icon_code": 4,
        "wxman": "wx6500",
        "phrase_12char": "Hvy T-Storms",
        "phrase_22char": "Heavy Thunderstorms",
        "phrase_32char": "Heavy Thunderstorms",                                <- Observation
        "subphrase_pt1": "Heavy",
        "subphrase_pt2": "T-Storms",
        "subphrase_pt3": "",
        "precip_type": "rain",
        "rh": 74,
        . . .
  }
  . . .
```

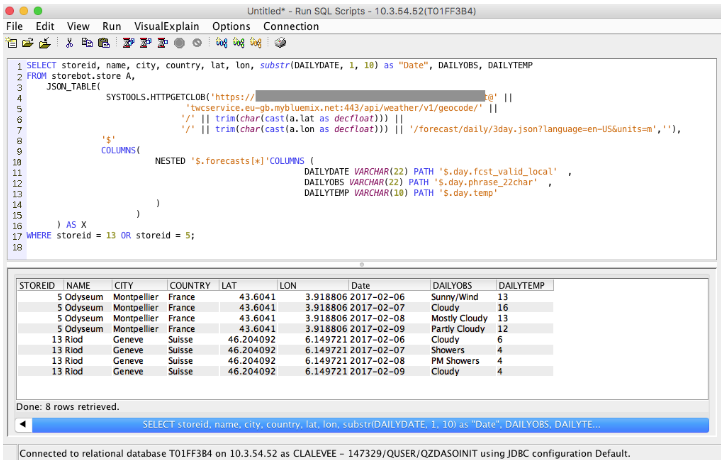
In Listing 3, `JSON_TABLE` part, `COLUMNS` / `NESTED` section, you can see that we declare the JSON table `forecast`. The `JSON_TABLE` function will return one row by element in the JSON `forecast` table. Each row will contain three fields: `day.fcst_valid_local`, `day.phrase_22char`, and `day.temp`.

Listing 3. Weather forecast for 3 days for stores in DB2 table

```
        SELECT storeid, name, city, country, lat, lon,
               SUBSTR(dailydate, 1, 10) as "Date", daylyobs, dailytemp
        FROM storebot.store A,
            JSON_TABLE(
                       SYSTOOLS.HTTPGETCLOB('https://<username>:<password>@' ||
                                            'twcservice.eu-gb.mybluemix.net:443' ||
                                            '/api/weather/v1/geocode/' ||
                                            '/' || trim(char(cast(a.lat as decfloat))) ||
                                            '/' || trim(char(cast(a.lon as decfloat))) ||
                                            '/forecast/daily/3day.json?language=en-US&units=m',''),
                       '$'
                       COLUMNS(
                               NESTED '$.forecasts[*]'
                                      COLUMNS ( DAILYDATE VARCHAR(22)PATH '$.day.fcst_valid_local' ,
                                                DAILYOBS VARCHAR(22) PATH '$.day.phrase_22char' ,
                                                DAILYTEMP VARCHAR(10) PATH '$.day.temp'
                                      )
                       )
               ) AS X
        WHERE storeid = 13 OR storeid = 5;
```

As we have four elements in the `forecast` table (today' conditions + three days), we will get four rows to be joined with each store.

Figure 14. ACS SQL editor: get weather forecasts for three days for each selected row from a SQL request


Untitled* - Run SQL Scripts - 10.3.54.52(T01FF3B4)
File Edit View Run VisualExplain Options Connection

```
SELECT storeid, name, city, country, lat, lon, substr(DAILYDATE, 1, 10) as "Date", DAILYOBS, DAILYTEMP
FROM storebot.store A,
    JSON_TABLE(
                SYSTOOLS.HTTPGETCLOB('https://[redacted]@' ||
                                'twcservice.eu-gb.mybluemix.net:443/api/weather/v1/geocode/' ||
                               '/' || trim(char(cast(a.lat as decfloat))) ||
                               '/' || trim(char(cast(a.lon as decfloat))) || '/forecast/daily/3day.json?language=en-US&units=m',''),
               '$'
               COLUMNS(
                          NESTED '$.forecasts[*]'COLUMNS (
                                                 DAILYDATE VARCHAR(22) PATH '$.day.fcst_valid_local'  ,
                                                 DAILYOBS VARCHAR(22) PATH '$.day.phrase_22char'  ,
                                                 DAILYTEMP VARCHAR(10) PATH '$.day.temp'
                          )
               )
    ) AS X
WHERE storeid = 13 OR storeid = 5;
```

STOREID	NAME	CITY	COUNTRY	LAT	LON	Date	DAILYOBS	DAILYTEMP
5	Odyseum	Montpellier	France	43.6041	3.918806	2017-02-06	Sunny/Wind	13
5	Odyseum	Montpellier	France	43.6041	3.918806	2017-02-07	Cloudy	16
5	Odyseum	Montpellier	France	43.6041	3.918806	2017-02-08	Mostly Cloudy	13
5	Odyseum	Montpellier	France	43.6041	3.918806	2017-02-09	Partly Cloudy	12
13	Riod	Geneve	Suisse	46.204092	6.149721	2017-02-06	Cloudy	6
13	Riod	Geneve	Suisse	46.204092	6.149721	2017-02-07	Showers	4
13	Riod	Geneve	Suisse	46.204092	6.149721	2017-02-08	PM Showers	4
13	Riod	Geneve	Suisse	46.204092	6.149721	2017-02-09	Cloudy	4

```
Done: 8 rows retrieved.
SELECT storeid, name, city, country, lat, lon, substr(DAILYDATE, 1, 10) as "Date", DAILYOBS, DAILYTE...
Connected to relational database T01FF3B4 on 10.3.54.52 as CLALEVEE - 147329/QUSER/QZDASOINIT using JDBC configuration Default.
```

The result shows that for each store, we have the condition date, observation, and temperature columns from Weather Company Data services, plus additional columns from the `storebot.store` table.

Summary

The power of the `httpGetCLOB` SQL function combined with the `JSON_TABLE` function provides an easy way to retrieve and parse data from REST web services, and to join JSON data with traditional DB2 for i relational data. As SQL is the standard database query language, you can then use this statement in your programs (such as RPG, PHP, Node.js, and so on) or in your WebQuery IBM DB2 Web Query for i reports.

So, let's start modernizing your IBM i applications and accelerating your digital transformation with APIs integration!

References

Bluemix: Weather Company Data service

- Getting started with Weather Company Data
- Weather Company Data For IBM Bluemix APIs

httpGetCLOB function

- Accessing web services: Using IBM DB2 for i HTTP UDFs and UDTFs

JSON_TABLE function

- The powerful JSON_TABLE function: Utilize JSON information with IBM DB2 for i
- JSON_TABLE in DB2 for i SQL Reference