February 26, 2018

Tips for parsing JSON in APEX

Carsten Czarski
CONSULTING MEMBER OF TECHNICAL STAFF

Working with JSON is daily business for Application Express developers. RESTful services typically return JSON, JSON is frequently used to store configuration or other kinds of flexible data, and JSON also becomes more and more a data exchange format. So developers frequently encounter the requirement to parse JSON documents and process data.

Based on the database version, there are multiple alternatives to work with JSON documents in a PL/SQL or SQL context.

- On 11.2 or 12.1.0.**1** databases, no native JSON functionality is present in SQL or PL/SQL. However, APEX provides the **APEX_JSON** package since version 5.0.
- In 12.1.0.2, the **JSON_TABLE**, **JSON_QUERY** or **JSON_VALUE** SQL functions were introduced. These allow to parse JSON within a SQL query or a SQL DML statement.
- Starting with 12.2, additional SQL functions for JSON generation as well as a PL/SQL based JSON parser is available.

When using 12.2 database (or even 18.1), there are a few alternatives to choose from. Many APEX developers just continue using APEX_JSON - because it's known - and probably also because of the **APEX_** prefix. But having a closer look into the native JSON functionality is absolutely well-invested time: it can lead to massively improved performance thus lowering general load on the database. This article will show a few comparisons between APEX_JSON and native functionality.

Don't stop reading when you're still on 11.2 or have to support 11.2 databases as well. It might be a good idea to use PL/SQL conditional compilation and add APEX_JSON as well as native PL/SQL JSON parsing to your code. On a more recent database, your application will perform better out-of-the-box. And as soon as you discontinue your 11.2 support, you can simply remove the old code. And the nice side-effect is, that you already trained yourself on the new functionality.

Prerequisites

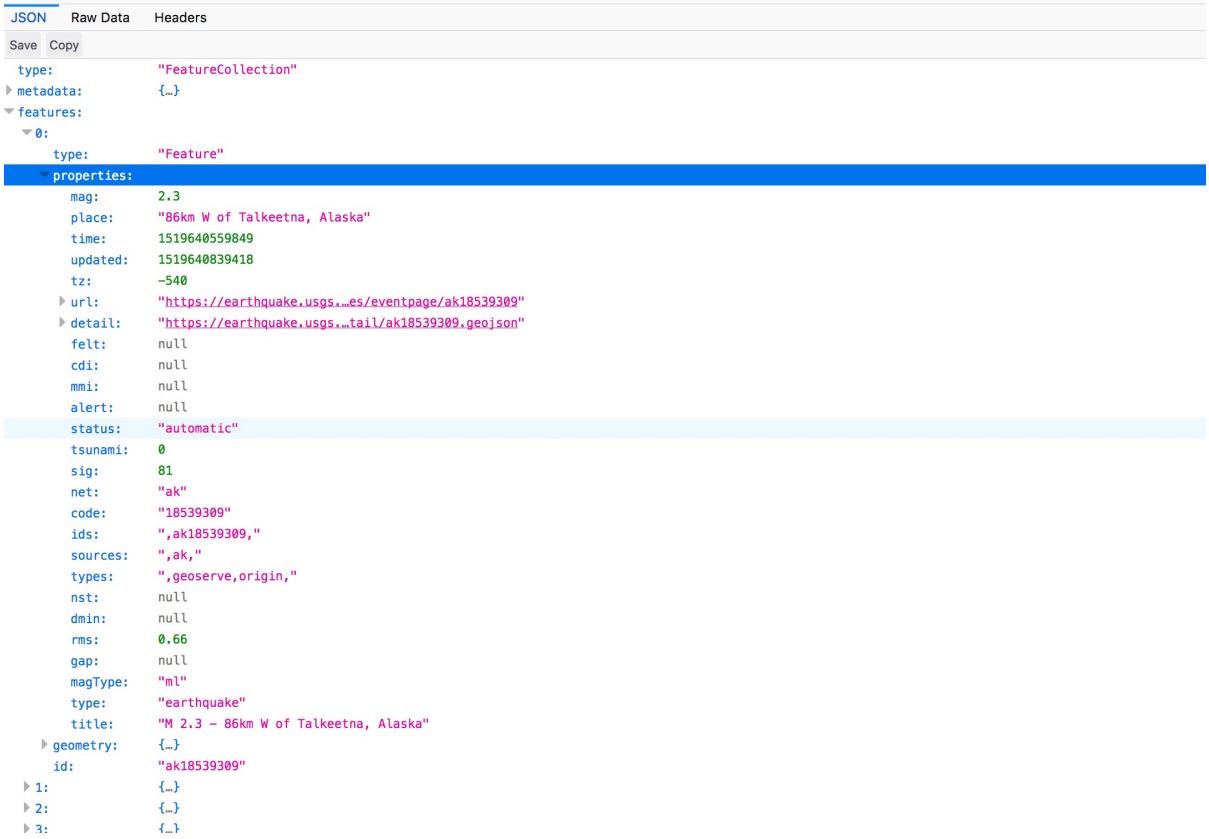
Let's now start with an example to compare the various approaches of JSON parsing in the database. We first need a JSON document to play with - and since the internet is full of them, we simply grab it from

there. As for earlier articles and how tos, we'll use Earthquake JSON feeds provided by the US Geological Survey (USGS). With the **APEX_WEB_SERVICE** package we'll grab them and because we don't want to execute HTTP requests all the time, we'll store them as a CLOB into a table:

```
create table earthquake_json(
    id          number    generated always as identity,
    fetched_at  timestamp default systimestamp,
    document    clob,
    constraint document_isjson check (document is json)
);

insert into earthquake_json(
    document
) values (
    apex_web_service.make_rest_request(
        p_url         => 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.geojson',
        p_http_method => 'GET' )
);

select id, fetched_at, dbms_lob.getlength(document) from earthquake_json;

ID  FETCHED_AT                    DBMS_LOB.GETLENGTH(DOCUMENT)
--- ----------------------------- ----------------------------
  1 26.02.18 02:29:59,988083000                        6743973
```

Depending on the URL being invoked, the JSON contains earthquakes for a day, a week or a month. The JSON document structure is outlined in the screen shot below. The **features** attribute is an array - each element contains data for one earthquake.

> When working on a 12c or 11g database, you might encounter an ORA-28860 message ("Fatal SSL error") when invoking APEX_WEB_SERVICE.MAKE_REST_REQUEST. This is due to a bug in the UTL_HTTP package (bugs 25734963 and 26040483). Navigate to MyOracleSupport and request or download a patch for your database version. For instance, download patch number **27551077** for Oracle Database 12.2.



For the below examples, we invoked the URL returning earthquakes for a *month*. That leads to somewhat larger JSON documents; which is intentional. On a larger scale, we'll clearly see how the different JSON parsers will perform.

Parse JSON within PL/SQL

First, we want to know, how many array elements are present in the **features** attribute. First attempt is with **APEX_JSON**:

```
declare
    l_clob           clob;
    l_feature_count  pls_integer;
    l_time           timestamp;
begin
    select document into l_clob
      from earthquake_json
     where id = 1;

    l_time := systimestamp;

    apex_json.parse(
        p_source => l_clob );

    dbms_output.put_line( 'Parsing Time: ' || extract( second from ( systimestamp - l_time ) ) );
    l_time := systimestamp;

    l_feature_count := apex_json.get_count( 'features' );
    dbms_output.put_line( 'Array Count: ' || l_feature_count );

    dbms_output.put_line( 'Get Array Count Time: ' || extract( second from ( systimestamp - l_time ) ) );
end;

Parsing Time:         12,293681
Array Count:               9456
Get Array Count Time:  0,000039
```

As the output indicates, the call to **APEX_JSON.PARSE** needed about 12 seconds; that is the time APEX_JSON needed to actually parse the JSON document. Counting the elements of the **features** array is rather quick; it needed only a very small fraction of a second. During the PARSE call, APEX_JSON built an internal representation of the JSON document; that required all the time. Retrieving information from that internal memory structure is actually pretty cheap. So, when working with APEX_JSON, try to call PARSE only once and to reuse the parsed JSON as often as possible.

Let's now have a look into the *PL/SQL Object Types* for JSON parsing, introduced in Oracle Database 12.2. These types are **JSON_OBJECT_T**, **JSON_ARRAY_T**, **JSON_ELEMENT_T**, **JSON_SCALAR_T** and **JSON_KEY_LIST**. JSON_ELEMENT_T is a generalization of JSON_OBJECT_T, JSON_ARRAY_T or JSON_SCALAR_T, representing either a JSON object, an array or a scalar attribute value. JSON_ELEMENT_T, JSON_OBJECT_T or JSON_ARRAY_T contain a static PARSE method in order to parse a JSON document. When we change the above PL/SQL block in order to use the new PL/SQL object types, the result doesn't look so different ...

```
declare
    l_clob           clob;
    l_feature_count  pls_integer;
    l_time           timestamp;

    l_json           json_object_t;
    l_features       json_array_t;
begin
    select document into l_clob
      from earthquake_json
     where id = 1;

    l_time := systimestamp;

    l_json := json_object_t.parse( l_clob );

    dbms_output.put_line( 'Parsing Time: ' || extract( second from ( systimestamp - l_time ) ) );
    l_time := systimestamp;

    l_features := l_json.get_array( 'features' );
    dbms_output.put_line( 'Array Count: ' || l_features.get_size );

    dbms_output.put_line( 'Get Array Count Time: ' || extract( second from ( systimestamp - l_time ) ) );
end;

Parsing Time:         0,124148
Array Count:          9456
Get Array Count Time: 0,000083
```

... but there is a huge difference regarding the consumed time. 0.12 seconds vs 12 seconds - that is about 100 times. For smaller documents, the performance difference is smaller, but it's always there. So JSON parsing with APEX_JSON is much more expensive than with the native functions - which is obvious, since APEX_JSON is a PL/SQL implementation whereas the PL/SQL object types have a C implementation as

part of the database itself. And such a thing like JSON parsing (which is mostly string operations) will gain huge benefits from a native implementation.

Parse JSON with a SQL query

Now we want to look into the earthquake data in more detail; we want to get some information about the weakest and the strongest earthquake contained in the JSON document. We *could* do this the same way as we did above: implementing a procedure using APEX_JSON (or better: JSON_OBJECT_T), parse the JSON, then retrieve the data of interest using PL/SQL loops and calling the various APEX_JSON getter functions or JSON_OBJECT_T methods.

But we're in a database, aren't we? The most powerful way to deal with data is to use SQL - and to use SQL functionality with JSON data, we need to treat a JSON document as it was a table. The **JSON_TABLE** SQL function will come in very handy now ...

```
with eqdata as (
    select e.id,
           e.title,
           e.mag
    from earthquake_json j, json_table(
        document,
        '$.features[*]'
        columns(
            id    varchar2(20)  path '$.id',
            mag   number        path '$.properties.mag',
            title varchar2(200) path '$.properties.title' ) ) e
), minmax as (
    select min(e.mag) minmag, max(e.mag) maxmag
      from eqdata e
)
select e.id,
       e.title,
       e.mag
  from eqdata e, minmax m
 where e.mag in ( m.minmag, m.maxmag )
/

ID          TITLE                                          MAG
----------- ---------------------------------------------- -----
us2000d7q6  M 7.5 - 89km SSW of Porgera, Papua New Guinea    7.5
uw61366531  M -0.8 - 36km NNE of Amboy, Washington          -0.8

2 rows selected.

Elapsed: 00:00:01.545
```

JSON_TABLE allows to project attributes of an array within a JSON document as a table - with rows and columns. That table can then be used like a normal table - so within a SQL query we can build subqueries, apply aggregations or all other SQL functionality to the data.

Isn't that an elegant way to parse JSON? The usage of subquery factoring and SQL aggregate functions leads to a clear, structured and very maintainable SQL query. Even more: if, at some day, the data will be present as a normal relational table, it will be super-easy to adopt to this. Most of the query wouldn't even have to be changed. JSON_TABLE is available in 12.1.0.2 or later.

When you're still on 11.2, there's no JSON_TABLE. But does that mean, you cannot use this elegant SQL-centric JSON parsing ...?

No, **APEX_JSON** will help. It provides the **TO_XMLTYPE** function which converts a JSON to an XML document. And for XML documents, the SQL function XMLTABLE is available since Oracle 9.2. So, combining APEX_JSON and XMLTABLE will give you the same power as JSON_TABLE does. Here's the SQL query doing the same as above - but that one will run on 11.2 as well.

```
with eqdata as (
    select e.id,
           e.title,
           e.mag
    from earthquake_json j, xmltable(
        '/json/features/row'
        passing apex_json.to_xmltype( j.document )
        columns
            id    varchar2(20)  path 'id/text()',
            mag   number        path 'properties/mag/text()',
            title varchar2(200) path 'properties/title/text()' ) e
), minmax as (
```

```
    select min(e.mag) minmag, max(e.mag) maxmag
      from eqdata e
)
select e.id,
       e.title,
       e.mag
  from eqdata e, minmax m
 where e.mag in ( m.minmag, m.maxmag )
/

ID          TITLE                                          MAG
----------- ---------------------------------------------- -----
us2000d7q6  M 7.5 - 89km SSW of Porgera, Papua New Guinea    7.5
uw61366531  M -0.8 - 36km NNE of Amboy, Washington          -0.8

2 rows selected.

Elapsed: 00:00:27.152
```

The results are the same. As we can see in the timing - there is no free lunch: We got the power of SQL to work on the JSON document in 11.2. But we first had to convert to XML with APEX_JSON; and that comes at a cost. 27 seconds is way more than the 1.4 seconds JSON_TABLE consumed by JSON_TABLE. A significant part of this was spent for the JSON to XML conversion ...

```
select apex_json.to_xmltype(document)
  from earthquake_json
 where id = 1;

APEX_JSON.TO_XMLTYPE(DOCUMENT)
------------------------------
<json>...

Elapsed: 00:00:12.711
```

... which pretty nicely matches the time consumption of APEX_JSON.PARSE in the first example.

From the results of these simple tests, we can derive some rules for working with JSON in APEX (and also outside of APEX) ...

1. APEX_JSON introduces JSON functionality to 11.2 and 12.1.0.1 databases, which don't have any native JSON support. So before having nothing, use APEX_JSON. But it comes at a price. Since APEX_JSON is implemented in PL/SQL, parsing is much more expensive compared to the native JSON functionality introduced in the upcoming database releases. Thus, on the other side: *use(!) native JSON functionality when available in your database.*

2. The JSON_TABLE SQL function (12.1.0.2) or APEX_JSON.TO_XMLTYPE / XMLTYPE (on 11.2) allow to work on JSON documents with SQL. That allows not only to apply aggregate or other SQL functions, it also allows to drive APEX components like reports or charts with JSON data. Some tasks are much more efficient to implement in SQL - so when you have such a task: Use JSON_TABLE instead of procedural PL/SQL.

3. As your database instance and APEX applications move forward, consider replacing APEX_JSON calls with JSON_TABLE or JSON_OBJECT_T.PARSE, depending on the concrete task.

More Information

- Documentation on APEX_JSON (APEX 5.0)
  https://docs.oracle.com/cd/E59726_01/doc.50/e39149/apex_json.htm#AEAPI29635

- Documentation on JSON_TABLE (Database 12.1.0.2)
  https://docs.oracle.com/database/121/SQLRF/functions092.htm#SQLRF56973

- Documentation on JSON_OBJECT_T (Database 12.2.0.1)
  https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adjsn/using-PLSQL-object-types-for-JSON.html#GUID-F0561593-D0B9-44EA-9C8C-ACB6AA9474EE

back to blogs.oracle.com/apex

Recent Content

APPLICATION EXPRESS

Custom Authentication and Authorization using built in APEX Access Control - A How To.

Its pretty well known in the APEX world that built-in security comes in two main flavors: Authentication (can I get in to the app at all)...

APPLICATION EXPRESS

Extending interaction to Interactive Grids

Interactive Grids were introduced in Oracle APEX 5.1. Interactive Grid is the evolution of the legacy tabular forms, and introduced the...