Manejando JSON en Oracle Database 12c

Por Francisco Riccio 

Introducción

A partir de la versión Oracle Database 12.1.0.2 es posible trabajar con el formato JSON dentro de nuestra base de datos relacional Oracle.

JSON es un formato para la notación de objetos de JavaScript pero también permite mantener un formato para estructurar datos no estructurados tal como hacemos con XML, siendo fácil su lectura, escritura y parseo.

JSON permite almacenar los siguientes tipos de datos:

- Cadena.
- Objetos (Representado por {}).
- Arreglos (Representado por []).
- NULL
- Caracteres Unicode.
- Números.
- Boolean.

Ventajas y Desventajas de utilizar JSON sobre XML	
Ventajas	**Desventajas**
Utiliza menos bytes para estructurar los datos (recordemos que XML utiliza espacio de nombres) y por lo tanto es el ideal para implementar servicios web (RESTful).	No se integra con todos los tipos de datos posibles. Con XML es posible incluir archivos multimedia.
Simplicidad para su construcción. Se puede estructurar objetos, arreglos de manera más sencilla.	Mayor soporte y funcionalidades por tener más año en el mercado.

Oracle Database 12.1.0.2 no ha implementado un tipo de dato especial para JSON como si existe para XML (XMLType). Su implementación a la fecha se realiza utilizando los tipos de datos caracteres que conocemos como: CLOB, VARCHAR/CHAR y una serie de funcionalidades que darán el soporte respectivo.

Con la finalidad de aprender cómo estructurar documentos JSON se recomienda la revisión del siguiente tutorial: http://www.w3schools.com/json/

Implementación

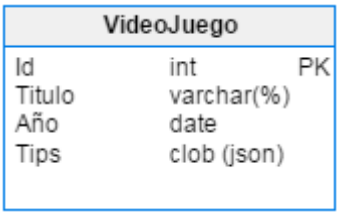
a) Escenario Propuesto

Para comprender como utilizar JSON en nuestra base de datos Oracle 12c, primero vamos a introducir un escenario y luego se realizarán variantes del caso.

El escenario será implementar una base de datos que permita registrar un catálogo de videojuegos para una tienda de alquiler. Se deberá almacenar la información de los empleados, clientes y los alquileres respectivos.

Adicional es importante que los clientes puedan registrar tips, secretos o comentarios de algunos de los videojuegos registrados en la tienda; con la finalidad de ayudar a otros videos jugadores a superar dificultades. Este último requerimiento es complejo, debido a que cada videojuego es diferente, algunos podrán contar con tips por niveles y por tipo de complejidad, mientras otros únicamente cuentan con un tipo de complejidad sin niveles y en otros escenarios, para un mismo nivel de un mismo videojuego, pudieran existir más de un tip, secreto o comentario que un cliente quisiera registrar. Como podemos apreciar, este último requerimiento sería un dolor de cabeza tratarlo de modelar bajo un modelo relacional debido a que los datos no guardan una misma estructura en cada escenario y es donde JSON podría ayudarnos a contar con un modelo más simple y flexible.

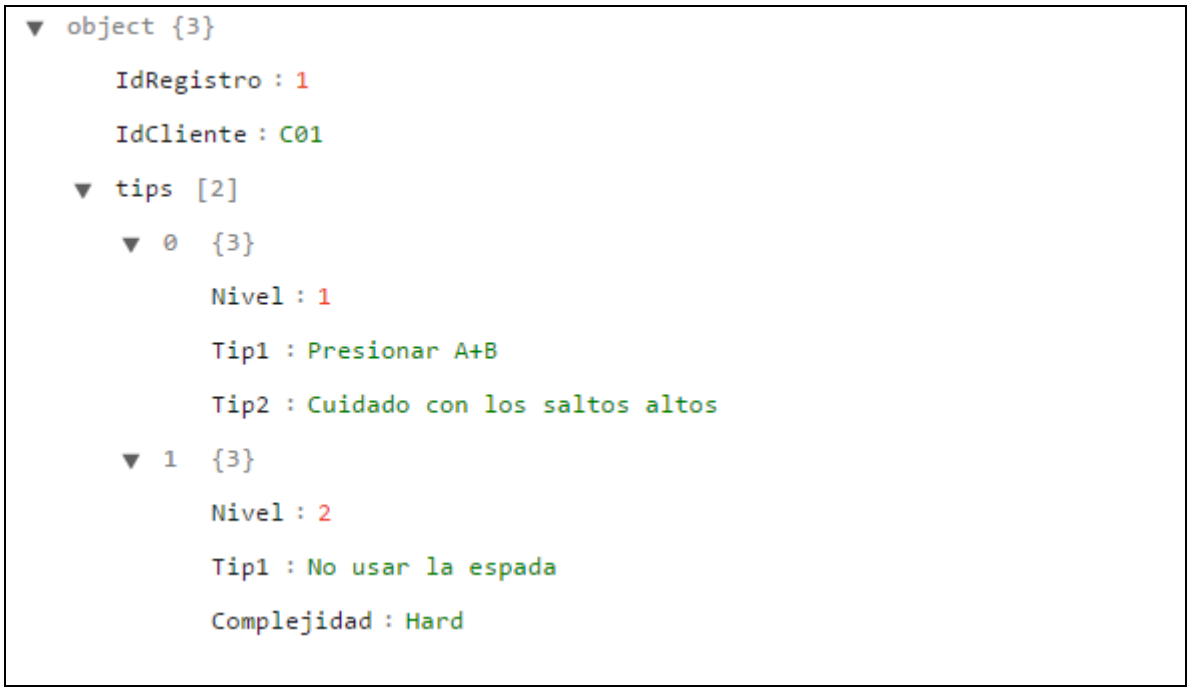
A continuación se representa la tabla VideoJuego (se han omitido algunos campos para su simplicidad) y su estructura JSON que podría utilizarse:



Se muestra una posible entrada en el campo Tips en formato JSON:

{"IdRegistro":1,IdCliente:"C01",tips:[{"Nivel": 1,Tip1:"Presionar A+B",Tip2:"Cuidado con los saltos altos"}, {"Nivel": 2,Tip1:"No utilizar la espada","Complejidad":"Hard"}]}

Gráficamente:



b) <u>Creación de la tabla</u>

```
SQL> create table VideoJuego
(
 id number generated always as identity,
 titulo varchar(20),
 anio date,
 tips clob constraint CC JSON CLOB check (tips is JSON)
);

Table created.
```

Para indicar que un campo almacenará un dato en formato JSON debemos crear un check constraint sobre un campo de tipo CLOB o VARCHAR/CHAR. Los documentos JSON suelen ser extensos por lo que se recomienda el uso de CLOB como tipo de dato.

Si deseamos que no se permitan ingresar documentos JSON repetidos podemos agregarle la opción (WITH UNIQUE KEYS) como se muestra a continuación:

```
SQL> create table VideoJuego
(
 id number generated always as identity,
 titulo varchar(20),
 anio date,
 tips clob constraint CC JSON CLOB check (tips is JSON (with unique keys))
);

Table created.
```

Un documento repetido se considera cuando existen dos documentos con los mismos atributos y valores.

Nota: Si deseamos obligar que los nombres de los atributos estén encerrados en doble comillas al ingresarlo en el documento JSON debemos añadir la siguiente extensión: JSON (STRICT).

c) Registro de Datos

```
SQL> insert into VideoJuego(titulo,anio,tips) values ('Super Mario Bros',to_date('13/09/1985','DD/MM/YYYY'),
 '{IdRegistro:1,IdCliente:"C01",tips:[{Nivel: "1-2",Tip1:"Al final de la fase hay tubos secretos"}]}'
);

1 row created.

SQL> insert into VideoJuego(titulo,anio,tips) values ('Contra',to_date('20/02/1987','DD/MM/YYYY'),
      '{IdRegistro:1,IdCliente:"C08",
       tips:[{Nivel: "1",Tip1:"Es importante no perder vidas"},
             {Nivel: "7",Tip1:"Es necesario contar con 3 vidas"}
            ]}'
     );

1 row created.

SQL> commit;

Commit complete.
```

Nota: En esta versión de Oracle Database (12.1.0.2) si deseamos modificar el valor de un atributo o agregar más atributos a un documento ya registrado debemos reemplazar todo el documento.

d) Realizar consultas

- Consulta básica.

```
SQL> select v.titulo,v.tips.IdCliente
     from VideoJuego v;

TITULO
--------------------
IDCLIENTE
----------------------------------------
Super Mario Bros
C01

Contra
C08
```

En este caso solo debemos especificar el nombre de la columna y el atributo del documento JSON que deseamos listar.

- Consulta básica utilizando la función **JSON_VALUE**.

La función JSON_VALUE recibe 2 parámetros: un campo o una cadena de caracteres donde este almacenado el documento JSON y el PATH. El símbolo $ indica inicio del documento.

```
SQL> select v.titulo,
       JSON_VALUE(tips,'$.IdCliente')
from VideoJuego v;

TITULO
--------------------
JSON_VALUE(TIPS,'$.IDCLIENTE')
----------------------------------------
Super Mario Bros
C01

Contra
C08
```

Como podemos apreciar, llegamos al mismo resultado pero la función permite utilizar opciones adicionales como se verán a continuación:

```
SQL> select v.titulo,
       JSON_VALUE(tips,'$.IdCliente' returning VARCHAR2(3)) as IdCliente
from VideoJuego v;

TITULO               IDC
-------------------- ---
Super Mario Bros     C01
Contra               C08
```

Por defecto, los valores de los atributos son devueltos como un VARCHAR2(4000). Con la opción RETURNING podemos cambiar la precisión o inclusive cambiar a otro tipo de dato, como: NUMBER, TIMESTAMP, etc.

En caso lo limitemos a 3 caracteres y existan valores por encima de esa longitud dará un error a menos que sea truncado con la siguiente opción:

RETURNING VARCHAR2(xxx) TRUNCATE.

Nota 1: La opción RETURNING no acepta el valor VARCHAR sino VARCHAR2.

Nota 2: La notación JSON no tiene definido un formato para fechas por lo cual se recomienda utilizar el formato ISO 8601 para evitar problemas en la conversión.

Ejemplo de una fecha en formato ISO 8601: 2016-11-20T23:00:55.

Nota 3: Es posible consultar el valor de 1 elemento de un arreglo contenido en un documento JSON mediante la función JSON_VALUE, ejemplo:

```
SQL> select v.titulo,
     JSON_VALUE(v.tips,'$.tips[1].Nivel' returning VARCHAR2(3)) as NIV
from VideoJuego v;

TITULO               NIV
-------------------- ---
Super Mario Bros
Contra               7
```

Aquí se lista aquellos títulos con el segundo tip registrado (tips[1]).

- Consulta con el operador WHERE.

  Para filtrar aquellas filas que cumplan con una condición para el valor de un atributo del documento JSON podemos usar JSON_VALUE o JSON_EXISTS.

El siguiente ejemplo se implementará utilizando ambas funciones para filtrar aquellos documentos que cuenten con el atributo IdCliente y su valor sea C08.

```
SQL> select v.titulo
from VideoJuego v
where JSON_EXISTS(tips,'$.IdCliente')
      and JSON_VALUE(tips,'$.IdCliente')='C08';

TITULO
--------------------
Contra
```

Nota 1: Es posible utilizar la función JSON_EXISTS en un Check Constraint para forzar que un atributo siempre exista en un documento JSON, ejemplo:

ADD CONSTRAINT xxx (JSON_EXISTS(campoJSON, '$.atributo'));

Nota 2: Existe además la función JSON_TEXTCONTAINS que permite buscar un texto en todos los atributos del documento JSON. Para utilizar está función se requiere previamente contar con un índice JSON el cual será mostrado más adelante.

- Consulta utilizando la función **JSON_TABLE**

  La función JSON_TABLE permite mostrar los valores de un arreglo almacenado en un documento JSON.

  En el siguiente ejemplo se lista por cada título los diferentes tips almacenados.

  La función recibe el documento JSON como primer parámetro y luego recibe el arreglo a recorrer. La sección COLUMNS indicará los atributos que serán mostrados mediante la sentencia SELECT además de ofrecernos la opción de precisar el tipo de dato más exacto que corresponde.

```
SQL> select v.titulo,t.*
from VideoJuego v,
     JSON_TABLE(tips,'$.tips[*]'
      COLUMNS (
        "Nivel" VARCHAR(5) PATH '$.Nivel',
        "Tip1" VARCHAR(40) PATH '$.Tip1',
        "Tip2" VARCHAR(10) EXISTS PATH '$.Tip2'
      )) t;

TITULO               NIVEL TIP1                                      TIP2
-------------------- ----- ----------------------------------------- ------
Super Mario Bros     1-2   Al final de la fase hay tubos secretos    false
Contra               1     Es importante no perder vidas             false
Contra               7     Es necesario contar con 3 vidas           false
```

e) Índices en documentos JSON

Los campos que almacenan documentos JSON pueden ser indexados por índices de tipo B-TREE o BITMAP, inclusive pueden contar con la cláusula UNIQUE INDEX.

Ejemplo:

```
SQL> create index idx_VideoJuego_01
  on VideoJuego(JSON_VALUE(tips,'$.IdCliente'));

Index created.
```

Aquí se logra indexar por el atributo IdCliente.

Si realizamos una consulta filtrando por el atributo IdCliente veremos que el optimizador utilizará el índice creado:

```
SQL> explain plan for
 select v.titulo
 from VideoJuego v
where JSON_VALUE(tips,'$.IdCliente')='C08';

Explained.

SQL> set linesize 500
select *
from table(dbms_xplan.display());
SQL>
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
Plan hash value: 3086889993

------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                   |     1 |  2014 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| VIDEOJUEGO        |     1 |  2014 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_VIDEOJUEGO_01 |     1 |       |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------------------------
```

Se mencionó que es posible realizar búsquedas de un texto en un documento JSON, para lograr esto, primero debemos crear un Índice JSON, el cual se apoyará de Oracle Text para realizar la búsqueda.

Ejemplo de la creación del índice:

```
SQL> create index idx_videojuego_02
  ON VideoJuego(tips)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('SECTION GROUP CTXSYS.JSON_SECTION_GROUP SYNC (ON COMMIT)');


Index created.
```

Posteriormente realizamos la consulta y validamos que el índice fue utilizado por el optimizador.

```
SQL> explain plan for
 select tips
 from VideoJuego v
 where JSON_TEXTCONTAINS(tips,'$','tubos secretos');

Explained.

SQL> set linesize 500
select *
from table(dbms_xplan.display());
SQL>
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan hash value: 3122988714

-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                   |     1 |  2014 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| VIDEOJUEGO        |     1 |  2014 |     4   (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX               | IDX_VIDEOJUEGO_02 |       |       |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Conclusión

Concluimos que Oracle Database 12c (12.1.0.2) provee un conjunto de funcionalidades y soporte para almacenar datos en formato JSON sobre los tipos de datos caracteres. Asimismo podemos aprovechar este formato para generar modelos relacionales/orientado a documentos más eficientes, eliminando extensivos niveles de normalización en nuestro diseño generando un mejor tiempo de respuesta en nuestros programas.

Nota: Se Recomienda revisar la siguiente lectura para contar con un mejor criterio de decisión al elegir XML o JSON para representar nuestros datos no estructurados.

http://www.oracle.com/technetwork/es/articles/sql/xmltype-en-database11g-a-traves-xdb-1931103-esa.html

Publicado por Ing. Francisco Riccio. Es un IT Architect en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: francisco@friccio.com

web: www.friccio.com