



Instituto Tecnológico de Costa Rica
Centro Académico de Alajuela
Bases de datos 2

Investigación II Bases de Datos 2

Antonio Enrique Fernández García – Carné: 2022075006
Josué David Mena González – Carné: 2022138381
Marco Vinicio Rodríguez Vargas - Carné: 2022149445
Maximilian Latysh – Carné: 2022091544

Profesor:
Alberto Shum Chan

II Semestre, 2023

Definición y características principales de JSON

JavaScript Object Notation, conocido por su acrónimo JSON, es un formato de intercambio de datos ligero y de fácil lectura, adecuado tanto para seres humanos como para sistemas informáticos. Se utiliza para transmitir información entre un servidor y un cliente, así como entre los diferentes módulos de un sistema. A pesar de que este formato tuvo su origen como un subconjunto de la notación literal de objetos de JavaScript, su amplia adopción lo ha posicionado como un formato de datos independiente. De acuerdo con Mihai (2021), “The growing popularity of this form of data organization is due to the adoption of application development techniques without the use of a rigid scheme of data representation, which are considered by developers much more flexible, allowing them to react much faster to the changes that may occur. Representing data in JSON documents offers the advantage of encapsulating their complexity”. Su versatilidad ha permitido su implementación en una amplia variedad de plataformas y entornos de desarrollo.

Dentro de las principales características de este formato, se pueden identificar:

- Los datos se presentan en pares de clave-valor
- Permite tipos de datos simples, como números, strings, booleanos, nulos, arreglos y objetos, gracias a esto se puede representar de manera sencilla cualquier tipo de información
- Brinda la posibilidad de anidar objetos y arreglos, lo que significa que se puede representar datos jerárquicos
- Es independiente del lenguaje de programación
- La mayoría de lenguajes de programación poseen bibliotecas que permiten analizar y crear JSON, esto facilita su implementación y lo convierte en un formato fácil de parsear

- Es un formato de datos ligero, no agrega una sobrecarga al tamaño de datos, esto permite que puede ser transmitido de manera sencilla por la red
- No existen los comentarios, esto quiere decir que la documentación debe hacerse en otro lugar.
- No admite estructuras de datos cíclicas (que se referencian a ellas mismas), esto provocaría problemas a la hora de procesar y parsear datos.

Ventajas de JSON frente a otros formatos de intercambio de datos

JSON vs. XML

El lenguaje de marcado extensible (XML) permite definir y almacenar datos de forma compatible, admite el intercambio de información entre sistemas de computación, como sitios web, bases de datos y aplicaciones de terceros. Cuando se compara con JSON se puede observar este último presenta ciertas ventajas frente al XML, entre ellas se encuentran:

- JSON es más legible para los humanos que XML, la razón de esto es la sintaxis, principalmente porque XML utiliza etiquetas que en algunos momentos puede dificultar la comprensión.
- Al tener una sintaxis compacta los archivos JSON tienden a pesar menos que los XML
- Actualmente JSON es más utilizado en la web debido a la eficiencia y simplicidad
- El uso de etiquetas en XML hace más complicado la representación de datos jerárquicos mientras que JSON ofrece una estructura más flexible.
- La forma de representar datos en el formato JSON es muy parecida a la forma en que los hacen los lenguajes de manera nativa, esto permite un análisis más fácil.

JSON vs. CSV:

Los archivos CSV son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. Algunas ventajas que presenta JSON con respecto a CSV son:

- JSON permite la anidación para representar datos jerárquicos mientras que CSV al representar los datos de forma tabular, no.
- JSON brinda una gama amplia de representación de datos mientras que CSV se limita a datos simples (números, cadenas de caracteres...)
- JSON es más legible pues en CSV la única forma de interpretar los datos es conociendo con anterioridad las estructuras de datos que están representando.

- CSV se puede utilizar simplemente para presentar datos tabulares u hojas de cálculo mientras que JSON permite una mayor libertad, siendo utilizado en aplicaciones web, APIs e inclusive en bases de datos.

JSON vs. BSON

BSON es un formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos MongoDB. Es una representación binaria de estructuras de datos y mapas. El nombre BSON está basado en el término JSON y significa Binary JSON (JSON Binario). Ambos son dos buenos formatos y dependen del contexto para saber cuál es más conveniente, sin embargo, algunos aspectos en los que JSON es mejor son:

- JSON es más amigable para la depuración y el desarrollo, ya que los datos son fácilmente legibles y editables.
- Si bien los archivos BSON son más eficientes en término de espacio, la compresión de datos JSON en la red reduce significativamente el tamaño de los datos antes de la transmisión lo que lo hace más eficiente para transmisión de datos web
- Existen más herramientas y bibliotecas para trabajar con JSON, lo que permite trabajar más rápido y cómodo

JSON vs. protobuf:

Los Protocol Buffers, también conocidos como protobuf, son un formato binario que facilita el almacenamiento e intercambio de datos en aplicaciones. JSON presenta las siguientes ventajas sobre protobuf:

- JSON es más útil para realizar soluciones rápidas sin necesidad de planear la forma en que se van a transmitir los datos
- JSON no tiene un esquema formal, esto da la libertad de editar y hacer los cambios necesarios sin miedo a afectar la compatibilidad con versiones anteriores

- Es más sencillo editar la representación de datos en formato JSON que protobuf

JSON vs. YAML:

El fichero YAML es un lenguaje de declaración de datos que facilita la legibilidad y la capacidad de escritura del usuario. Este formato de serialización de datos se encarga de almacenar archivos de configuración y se puede usar junto con todos los lenguajes de programación. Algunas ventajas de JSON son:

- JSON es mejor en plataformas, lenguajes, comunicaciones de software distribuido, aplicaciones web, archivos de configuración y API.
- JSON se acopla mejor a la multiplataforma pues muchas de estas parsean de manera nativa los archivos de este tipo
- JSON es generalmente más rápido de parsear que YAML

Soporte de JSON en Oracle

A lo largo de las versiones de Oracle SQL se han introducido características muy importantes que cambian hasta cierto punto de gran forma el motor de bases de datos. El caso de la introducción de JSON a este motor no se queda atrás y para muchos ha sido una gran implementación. En el caso de esta adición se fue mejorando a lo largo de las nuevas versiones y estos detalles lo podemos ver en la documentación de cada versión. Más adelante se especificará más a fondo esta parte.

¿A partir de qué versión está soportado?

Todo empieza a partir de la versión 12c de oracle en su release 1. Esta versión fue lanzada en julio del año 2014. Esta versión introdujo por primera vez el soporte a JSON para bases de datos en Oracle SQL. En esta versión destacan elementos como la creación de tablas capaces de almacenar JSON, hacer consultas a datos de tipo JSON, sacar valores de JSON, entre otros. Como se puede ver se introdujeron muchos elementos llamativos e importantes para la manipulación de JSON, pero este apenas fue el inicio.

Resumen de mejoras y cambios en el soporte de JSON a lo largo de las diferentes versiones de Oracle

Esta parte se puede resumir por versión, destacando los puntos más importantes añadidos en cada una hasta la actualidad y así poder comparar cada actualización.

Versión 12c release 2:

- Se incluyó el manual en la documentación.
- Se añadieron soportes para las funciones JSON en PL/SQL.
- Se incluye una función para generar JSON directamente de SQL.
- Se simplificó la búsqueda de JSON.

Versión 18c:

- Se incluye la función TREAT que cambia la declaración de una expresión a tipo JSON.
- Mejoras a la función JSON_TABLE.
- Se introduce la condición JSON_EQUAL.

Versión 19c:

- Se mejora la función JSON_OBJECT.
- Se añade la función JSON_SERIALIZE.
- Se permite el mapeo de JSON a objetos SQL.

Versión 21c:

- Muy importante, se introduce el tipo de dato JSON.
- Se crea la función JSON_SCALAR.
- Se añaden otras funciones.

Versión 23c:

- Esta versión permite hacer y validar estructuras de JSON a partir de un esquema JSON.
- Se añadió la keyword ORDERED a la función JSON_SERIALIZE.
- Se añadió la vista dual a datos relacionales.
- CREATE SEARCH INDEX ayuda a crear índices en XML, JSON y texto.

En general han habido muchísimas novedades e implementaciones y novedades en cada versión a lo largo de los años. Esto es solo una pequeña parte de todo lo que se ha dado. Para consultar más información al respecto, el siguiente [enlace](#) permite ver más a fondo lo mencionado.

Operaciones básicas con JSON

Insertar, actualizar, y borrar datos en formato JSON

```
-- tabla categoria
DROP TABLE CATEGORIA;

CREATE TABLE CATEGORIA(
  id VARCHAR2(40) NOT NULL,
  descripcion VARCHAR2(100) NOT NULL,
  CONSTRAINT pk_categoria PRIMARY KEY(id)
)

-- tabla proveedor
DROP TABLE PROVEEDOR;

CREATE TABLE PROVEEDOR(
  id NUMBER(10) NOT NULL,
  nombre VARCHAR2(50) NOT NULL,
  descripcion VARCHAR2(100) NOT NULL,
  direccion VARCHAR2(100) NOT NULL,
  CONSTRAINT pk_proveedor PRIMARY KEY(id)
);
```

a. Insertar datos en formato JSON de la tabla categoria

```
INSERT INTO CATEGORIA (id, descripcion)
VALUES ('Bebida_energetica1', '{"nombre": "Powerade", "detalles":
"Bebida energética con electrolitos"}');

INSERT INTO CATEGORIA (id, descripcion)
VALUES ('Bebida_energetica2', '{"nombre": "Gatorade", "detalles":
"Bebida energética para rehidratación"}');
```

ID	DESCRIPCION
1 Bebida_energetica1	{"nombre": "Powerade", "detalles": "Bebida energética con electrolitos"}
2 Bebida_energetica2	{"nombre": "Gatorade", "detalles": "Bebida energética para rehidratación"}

b. Insertar datos en formato JSON de la tabla proveedor

```
INSERT INTO PROVEEDOR (id, nombre, descripcion, direccion)
```

```
VALUES ('proveedor1', 'Distribuidora Powerade CR',
      '{"tipo": "distribuidora de bebidas energéticas",
"productos_destacados": ["Powerade Fruit Punch", "Powerade Mixed Berry"]}',
      '{"calle": "Calle Los laureles", "ciudad": "San Jose", "pais": "Costa Rica"}'
);

INSERT INTO PROVEEDOR (id, nombre, descripcion, direccion)
VALUES ('proveedor2', 'Distribuidora Gatorade CR',
      '{"tipo": "distribuidora de bebidas energéticas",
"productos_destacados": ["Gatorade Tropical Fruit", "Gatorade Cool Blue"]}',
      '{"calle": "Calle Los Santos", "ciudad": "Heredia", "pais": "Costa Rica"}'
);
```

Consulta completa

ID	NOMBRE	DESCRIPCION	DIRECCION
1	proveedor1	Distribuidora Powerade CR	{ "tipo": "distribuidora de bebidas energéticas", "productos_destacados": ["Powerade Fruit Punch", "Powerade Mixed Berry"] }
2	proveedor2	Distribuidora Gatorade CR	{ "tipo": "distribuidora de bebidas energéticas", "productos_destacados": ["Gatorade Tropical Fruit", "Gatorade Cool Blue"] }

Consulta con mejor calidad

ID	NOMBRE	DESCRIPCION
1	proveedor1	Distribuidora Powerade CR
2	proveedor2	Distribuidora Gatorade CR

DIRECCION
{ "calle": "Calle Los laureles", "ciudad": "San Jose", "pais": "Costa Rica" }
{ "calle": "Calle Los Santos", "ciudad": "Heredia", "pais": "Costa Rica" }

c. Actualizar datos en formato JSON de la tabla categoria

```
UPDATE CATEGORIA
SET descripcion = '{"nombre": "Prime", "detalles": "Bebida energética para deportistas de alto rendimiento"}'
WHERE id = 'Bebida_energetica1';
```

ID	DESCRIPCION
1	Bebida_energetica1
2	Bebida_energetica2

d. Actualizar datos en formato JSON de la tabla proveedor

```
UPDATE PROVEEDOR
SET descripcion = '{"tipo": "distribuidora de bebidas
```

```
energéticas", "pais": "Costa Rica", "productos_destacados":
["Powerade Orange", "Powerade Strawberry"]}',
    direccion = '{"calle": "Calle La Arboleda", "ciudad":
"Heredia", "pais": "Costa Rica"}'
WHERE id = 'proveedor1';
```

Consulta completa

ID	NOMBRE	DESCRIPCION	DIRECCION
1 proveedor1	Distribuidora Powerade CR	{"tipo": "distribuidora de bebidas energéticas", "pais": "Costa Rica", "productos_destacados": ["Powerade Orange", "Powerade Strawberry"]}	{"calle": "Calle La Arboleda", "ciudad": "Heredia", "pais": "Costa Rica"}
2 proveedor2	Distribuidora Gatorade CR	{"tipo": "distribuidora de bebidas energéticas", "productos_destacados": ["Gatorade Tropical Fruit", "Gatorade Cool Blue"]}	{"calle": "Calle Los Santos", "ciudad": "Heredia", "pais": "Costa Rica"}

Consulta con mejor calidad

1 proveedor1	Distribuidora Powerade CR	{"tipo": "distribuidora de bebidas energéticas", "pais": "Costa Rica", "productos_destacados": ["Powerade Orange", "Powerade Strawberry"]}
2 proveedor2	Distribuidora Gatorade CR	{"tipo": "distribuidora de bebidas energéticas", "productos_destacados": ["Gatorade Tropical Fruit", "Gatorade Cool Blue"]}

DIRECCION
{"calle": "Calle La Arboleda", "ciudad": "Heredia", "pais": "Costa Rica"}
{"calle": "Calle Los Santos", "ciudad": "Heredia", "pais": "Costa Rica"}

e. Borrar datos en formato JSON de la tabla categoria

```
DELETE FROM CATEGORIA
WHERE id = ('Bebida_energetica2');
```

ID	DESCRIPCION
1 Bebida_energetical	{"nombre": "Prime", "detalles": "Bebida energética para deportistas de alto rendimiento"}

f. Borrar datos en formato JSON de la tabla proveedor

```
DELETE FROM PROVEEDOR
WHERE id = 'proveedor2';
```

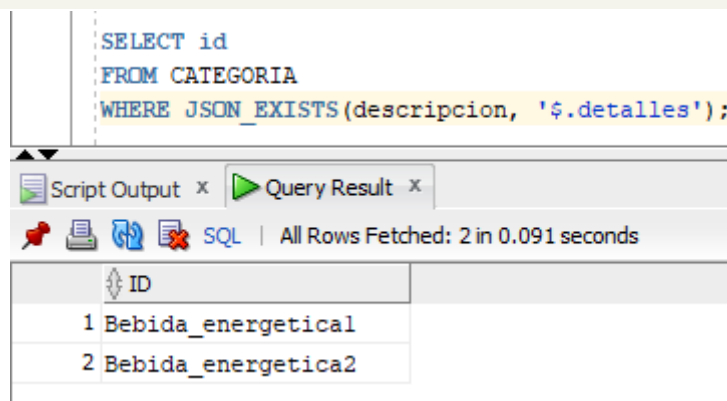
Consulta completa

ID	NOMBRE	DESCRIPCION	DIRECCION
1 proveedor1	Distribuidora Powerade CR	{"tipo": "distribuidora de bebidas energéticas", "pais": "Costa Rica", "productos_destacados": ["Powerade Orange", "Powerade Strawberry"]}	{"calle": "Calle La Arboleda", "ciudad": "Heredia", "pais": "Costa Rica"}

Consulta de datos almacenados en formato JSON utilizando SQL y funciones específicas de Oracle

- a. **JSON_EXISTS**: Esta función determina si un objeto JSON incluye una estructura específica. En este caso, se utiliza para buscar si cumple con cierta condición en el campo de “descripcion”. La condición que se evalúa es si el objeto JSON tiene en el campo “descripcion” contiene en su estructura “detalles”.

```
SELECT id
FROM CATEGORIA
WHERE JSON_EXISTS(descripcion, '$.detalles');
```



The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

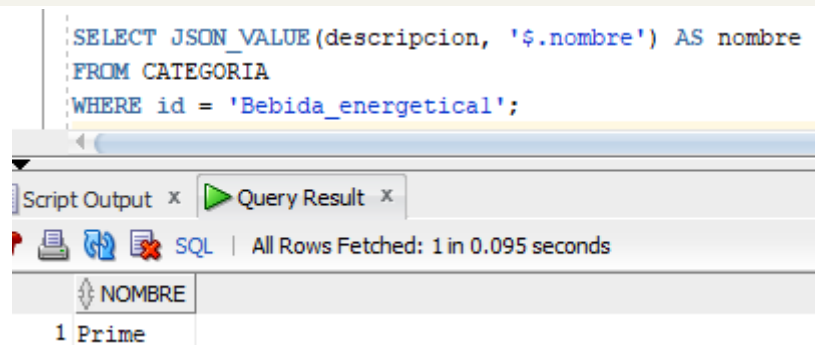
```
SELECT id
FROM CATEGORIA
WHERE JSON_EXISTS(descripcion, '$.detalles');
```

Below the editor, the 'Query Result' tab is active, displaying the results of the query. The status bar indicates 'All Rows Fetched: 2 in 0.091 seconds'.

ID
1 Bebida_energetical
2 Bebida_energetica2

- b. **JSON_VALUE**: Se utiliza para consultar un valor específico de un objeto JSON. En este caso, se utiliza para extraer el valor asociado a “nombre” del objeto JSON almacenado en el campo “descripcion”.

```
SELECT JSON_VALUE(descripcion, '$.nombre') AS nombre
FROM CATEGORIA
WHERE id = 'Bebida_energetical1';
```



The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

```
SELECT JSON_VALUE(descripcion, '$.nombre') AS nombre
FROM CATEGORIA
WHERE id = 'Bebida_energetical1';
```

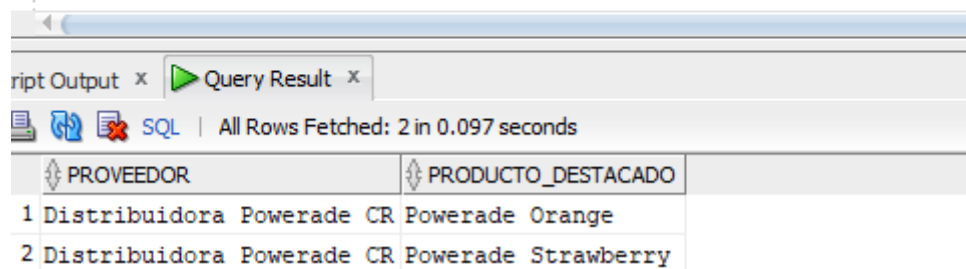
Below the editor, the 'Query Result' tab is active, displaying the results of the query. The status bar indicates 'All Rows Fetched: 1 in 0.095 seconds'.

NOMBRE
1 Prime

- c. JSON_TABLE: Se utiliza para convertir un objeto JSON a una tabla relacional. En este caso, se transforma el objeto JSON de proveedor para crear una tabla con su nombre y sus productos destacados.

```
SELECT PROVEEDOR.nombre AS PROVEEDOR, JSON.producto_destacado
FROM PROVEEDOR,
     JSON_TABLE(PROVEEDOR.descripcion, '$.productos_destacados[*]'
        COLUMNS (
            producto_destacado VARCHAR2(50) PATH '$'
        )
    ) JSON;
```

```
SELECT PROVEEDOR.nombre AS PROVEEDOR, JSON.producto_destacado
FROM PROVEEDOR,
     JSON_TABLE(PROVEEDOR.descripcion, '$.productos_destacados[*]'
        COLUMNS (
            producto_destacado VARCHAR2(50) PATH '$'
        )
    ) JSON;
```



The screenshot shows a database query result window with two tabs: "Script Output" and "Query Result". The "Query Result" tab is active, displaying the results of the SQL query. The status bar indicates "All Rows Fetched: 2 in 0.097 seconds". The results are presented in a table with two columns: "PROVEEDOR" and "PRODUCTO_DESTACADO".

	PROVEEDOR	PRODUCTO_DESTACADO
1	Distribuidora Powerade CR	Powerade Orange
2	Distribuidora Powerade CR	Powerade Strawberry

Validación y conversión de datos JSON

Saber cuándo y de qué manera se puede procesar la información de los JSONs es un asunto meramente importante. Por lo tanto, lógicamente la página oficial de Oracle tiene mucha documentación que contiene información explicativa ([ver esta página](#)).

Herramientas y funciones en Oracle para validar la estructura y contenido de datos JSON

Existen varios métodos en Oracle SQL para determinar si un JSON está correctamente formateado. Hace falta destacar que pueden existir casos cuando no hay errores gramaticales en el JSON pero igualmente no aplican para un cierto caso de uso principalmente debido a que posiblemente la estructura del JSON para una entrada no correspondía a la forma esperada (Oracle Help Center, 2023).

1. Función **JSON_EXISTS**: Esta función permite determinar si una expresión de JSON arbitraria contiene una cierta estructura esperada o no. Esta verificación se hace tomando información JSON y haciendo un query por medio de la función utilizando un “path expression” para determinar si lo esperado sí está presente.
2. Función **JSON_TEXTCONTAINS**: Permite analizar no solamente la estructura sino necesariamente la información asociada a las llaves que se buscaron por medio del “path expression”. Esto permite buscar los JSON que poseen una cierta llave y un cierto valor asociado a esta llave. Sin embargo, solamente funciona para columnas de tablas con un índice de contexto JSON (ver el ejemplo).
3. Las expresiones **IS JSON** y **IS NOT JSON**: Contrario a las funciones anteriores, estas expresiones no analizan la estructura para buscar componentes, sino simplemente validan si un texto es o no es una hilera de JSON válida. Generalmente, se deberían de utilizar antes de hacer cualquier otro procedimiento con información textual que representa un JSON.

```

-- Tira tura porque sí se encontró el wikir.
SELECT CASE WHEN json_exists('{"wikir" : 123}', '$.wikir') THEN
'true' ELSE 'false' END AS ttt FROM DUAL;

-- Tira false porque tamar no es wikir.
SELECT CASE WHEN json_exists('{"tamar" : 123}', '$.wikir') THEN
'true' ELSE 'false' END AS ttt FROM DUAL;

-- Preparación para la prueba 2
CREATE TABLE tartar (
    jalor VARCHAR2(100)
);

-- Índice para la columna
CREATE INDEX ix
    ON tartar(jalor)
    INDEXTYPE IS CTXSYS.CONTEXT
    PARAMETERS ('SECTION GROUP CTXSYS.JSON_SECTION_GROUP SYNC (ON
COMMIT)');

INSERT INTO tartar VALUES('{"wikir" : 123}');
INSERT INTO tartar VALUES('{"wikir" : "Wakitaki"}');
INSERT INTO tartar VALUES('{"timar" : "Wakitaki"}');

COMMIT;
-- Devuelve {"wikir" : "Wakitaki"}
SELECT * FROM tartar WHERE json_textcontains(jalor, '$.wikir',
'Wakitaki');

-- Tira true porque la estructura en sí del texto sí es correcta.
SELECT CASE WHEN '{"hola" : [213, 32432, 23432423, 4]}' IS JSON
THEN 'true' ELSE 'false' END AS ttt FROM DUAL;

-- Tira false porque se quitó una doble comilla
SELECT CASE WHEN '{"hola : [213, 32432, 23432423, 4]}' IS JSON
THEN 'true' ELSE 'false' END AS ttt FROM DUAL;

```

Conversión entre datos JSON y otros tipos de datos en Oracle

El principio de la conversión de datos JSON a otros tipos de datos en Oracle SQL y viceversa consiste principalmente en el empleo de dos funciones: **JSON_VALUE** y **JSON_OBJECT**.

Table 18-1 Compatible Scalar Data Types: Converting JSON to SQL

JSON Type (Source)	SQL Type (Destination)	Notes
string	VARCHAR2	None
string	CLOB	None
string	NUMBER	The JSON string must be numeric.
string	DATE	The JSON string must have a supported ISO 8601 format.
string	TIMESTAMP	The JSON string must have a supported ISO 8601 format.
number	NUMBER	None
number	VARCHAR2	None
number	CLOB	None
boolean	VARCHAR2	The instance value is the SQL string "true" or "false".
boolean	CLOB	The instance value is the SQL string "true" or "false".
null	Any SQL data type.	The instance value is SQL NULL.

Figura 4. Una tabla indicando las conversiones estándar (es decir, cómo se convierten los tipos más comunes). Obtenida de [esta página](#).

1. **JSON_VALUE**: Permite convertir información en formato JSON (ya sea como texto, blob o las demás representaciones de JSON en Oracle SQL) a un tipo específico escogiendo, por medio de un “path expression”, cuáles campos del JSON se van a “consumir” y cuáles no. Este tipo puede ser basado en los tipos estándares; sin embargo, también es capaz de manejar tipos creados por el usuario. Por ende, existe una deserealización del JSON con estilo recursivo (es decir, puede hacer la conversión esperada por lo tanto que los datos originales pueden ser convertidos). Aquí se muestra una tabla de “conversiones” obtenida de la documentación de la misma función en Oracle.

2. **JSON_OBJECT**: Permite convertir una serie de datos en JSON. Es capaz de manejar los tipos estándar de Oracle SQL. Además, también puede manejar los tipos creados por los usuarios. Por ende, se hace una serialización de los contenidos de los tipos creados por los usuarios de forma recursiva.

Ejemplo: Se puede crear tipos anidados que se pueden procesar a la hora serializarlos a un JSON o deserializarlos.

```
-- Preparación del ejemplo
CREATE TYPE t01 AS OBJECT
  (valor1 VARCHAR2(10),
   valor2 int);

CREATE TYPE t02 AS OBJECT
  (valor VARCHAR2(10),
   tipo t01);

CREATE TABLE tamino (
  valor1 t01,
  valor2 t02,
  valor3 int
);

INSERT INTO tamino VALUES(
  t01('hola', 5), t02('adios', t01('no', 4)), 101
);

-- Serealizamos a "tamino"
SELECT json_object(valor1, valor2, valor3) from tamino;

-- Deserealizamos un JSON en forma textual al tipo t02
SELECT
  json_value('{ "VALOR": "adios", "TIPO": { "VALOR1": "no", "VALOR2": 4 } }',
  '$' RETURNING t02).valor FROM DUAL;
```

Referencias

- MIHAI, Gianina. (2021). *JSON Data and Relational Data Integration in Oracle*. Annals of Dunarea de Jos University of Galati. Fascicle I. Economics and Applied Informatics. 27. 37-43. doi: 10.35219/eai15840409221. Recuperado de:
https://web.archive.org/web/20220621150849id_/http://www.eia.feaa.ugal.ro/images/eia/2021_3/Mihai.pdf
- Friesen, Jeff. (2016). *Introducing JSON*. doi: 10.1007/978-1-4842-1916-4_7. Recuperado de:
https://books.google.co.cr/books?id=9dh6DAAQBAJ&pg=PA133&lpg=PA133&dq=10.1007/978-1-4842-1916-4_7&source=bl&ots=6OBge9tHht&sig=ACfU3U3mE-u-o-loDHZbFGw3Nf9eHTX5hOg&hl=en&sa=X&ved=2ahUKewjV8-v8k5WCAxXyr-okEHZ20BokQ6AF6BAgIEAM#v=onepage&q=10.1007%2F978-1-4842-1916-4_7&f=false
- Petkovic, Dusan. (2023). *Comparison of Techniques for Storing JSON Data in Relational Form*. International Journal of Computer Applications. 185. 9-12.
doi:10.5120/ijca2023922962. Recuperado de:
<https://www.ijcaonline.org/archives/volume185/number22/petkovi%C4%87-2023-ijca-922962.pdf>
- Šimec, Alen & Magličić,. (2014). *Comparison of JSON and XML Data Formats*.
https://www.researchgate.net/publication/329707959_Comparison_of_JSON_and_XML_Data_Formats
- Amazon Web Services, Inc. (s. f.). *YAML frente a JSON: Diferencia entre los formatos de serialización de datos*.
<https://aws.amazon.com/es/compare/the-difference-between-yaml-and-json/#:~:text=While%20YAML%20may%20appear%20to,already%20parse%20JSON%20data%20format.>

JSON. (s. f.). *Introducing JSON*. Recuperado de: <https://www.json.org/json-en.html>

Oracle Base. (2016). *An Introduction to JSON Support in the Oracle Database*.

<https://oracle-base.com/articles/misc/an-introduction-to-json-support-in-the-oracle-database>

September2023. (2023b, septiembre 16). *SQL/JSON Function JSON_VALUE*. Oracle Help Center.

https://docs.oracle.com/en/database/oracle/oracle-database/23/adjsn/function-JSON_VALUE.html

JSON in Oracle database. (2022, 24 abril). Oracle Help Center.

<https://docs.oracle.com/en/database/oracle/oracle-database/19/adjsn/json-in-oracle-database.html>