PROYECTO JSON

### **Generación de JSON desde Consultas SQL**

**Generación básica de JSON**: Utilizando FOR JSON PATH, podemos convertir una consulta en formato JSON. Por ejemplo:



Este código produce un JSON estructurado que contiene todas las bases de datos en SQL Server, organizadas bajo la raíz 'Bases'.

**Inclusión de valores nulos**: Mediante FOR JSON AUTO, INCLUDE\_NULL\_VALUES, es posible incluir valores NULL en la salida JSON, asegurando que no se omitan campos vacíos en el resultado.

**Eliminación de arreglos en el JSON**: Utilizando WITHOUT\_ARRAY\_WRAPPER junto con FOR JSON PATH, se elimina el uso de corchetes en el JSON generado. Esto es útil cuando solo se requiere un objeto JSON sin una estructura de arreglo.

### **Validación y Extracción de Datos JSON**

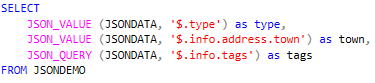
**Validación con ISJSON**: La función ISJSON verifica si una cadena es un JSON válido. Por ejemplo:

DECLARE @JSON VARCHAR(MAX) = '{"nombre": "TriggerDB.com", "Especialidad": "SQL Server"}'

SELECT CASE WHEN ISJSON(@JSON) > 0 THEN 'JSON Válido' ELSE 'JSON No Válido' END

Esta función es útil para asegurarse de que los datos cumplen con la estructura JSON antes de su procesamiento.

**Extracción de valores específicos**: Mediante JSON\_VALUE, se pueden extraer valores específicos de una cadena JSON. Por e

jemplo:

Aquí, se obtiene el valor de un campo específico y un arreglo completo de tags, lo cual es fundamental para analizar contenido JSON almacenado en una base de datos.

**Uso de OPENJSON**: La función OPENJSON descompone el JSON en filas y columnas, permitiendo su integración en tablas. Este ejemplo convierte una lista de etiquetas en filas:



### 

### **Carga de JSON en Tablas SQL**

**Insertar JSON en una tabla**: Con OPENJSON y la cláusula WITH, es posible mapear campos JSON a columnas de una tabla:

DECLARE @person NVARCHAR(MAX) = '{ "id": 2, "firstName": "John", "lastName": "Smith" }';

INSERT INTO Person

SELECT \*

FROM OPENJSON(@person)

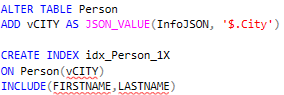
WITH (id INT, firstName NVARCHAR(50), lastName NVARCHAR(50))

Este método permite cargar datos JSON en estructuras relacionales.

**Guardar JSON en columnas**: Para almacenar datos JSON directamente, podemos definir columnas NVARCHAR(MAX) y aplicar un CHECK que valide la estructura JSON:



**Creación de índices en datos JSON**: Los índices en columnas derivadas de JSON pueden mejorar el rendimiento de consultas. Con JSON\_VALUE, se define una columna virtual sobre la cual aplicar el índice:



Este índice facilita la búsqueda rápida de registros en función de datos JSON.

### **Conclusión**

El manejo de JSON en SQL Server es una herramienta versátil que permite almacenar y manipular datos semiestructurados. Las funciones de SQL Server permiten validar, extraer y manipular datos JSON, integrándose con facilidad en bases de datos relacionales y ofreciendo nuevas capacidades para aplicaciones modernas.