# UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

## FACULTAD DE INGENIERIA ELECTRICA, ELECTRONICA, INFORMATICA Y MECANICA

**ESCUELA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS**

****

**INVESTIGACIÓN FORMATIVA**

FUNDAMENTO Y DISEÑO DE BASE DE DATOS DOCENTE: CARLOS RAMON QUISPE ONOFRE ALUMNO: RONALDO TICONA JANCCO

CODIGO: 211816

CUSCO-PERU 2025

# Índice

1. Introducción
   1. Concepto de información semi-estructurada
   2. Importancia del almacenamiento de datos semi-estructurados en bases de datos relacionales
   3. Objetivo de la monografía
2. Información Semi-estructurada en SQL Server
   1. Definición y características
   2. Soporte JSON en SQL Server
   3. Ventajas de utilizar JSON en SQL Server
3. Almacenamiento de Información Semi-estructurada
   1. Tipo de columna para almacenar JSON
   2. Validación de datos JSON
   3. Inserción y almacenamiento de datos
4. Manipulación de Información JSON en SQL Server
   1. Extracción de datos
      1. Uso de JSON\_VALUE
      2. Uso de JSON\_QUERY
      3. Conversión de JSON a tablas con OPENJSON
   2. Generación de JSON desde datos relacionales
      1. Función FOR JSON PATH
      2. Función FOR JSON AUTO
5. Prácticas Recomendadas
   1. Validación de JSON al insertar
   2. Uso de índices para mejorar el rendimiento
   3. Decisiones estratégicas entre JSON y columnas tradicionales
6. Estudio de Caso: Gestión de Productos con JSON
   1. Escenario
   2. Implementación técnica
   3. Resultados obtenidos
7. Conclusiones
   1. Beneficios del uso de JSON en SQL Server
   2. Desafíos en la gestión de datos semi-estructurados
8. Bibliografía

# Introducción

## Concepto de información semi-estructurada

La información semi-estructurada se refiere a un tipo de datos que no se organiza de manera estricta en tablas y columnas, como sucede en las bases de datos relacionales, pero que tampoco carece de estructura como los datos completamente no estructurados. Este formato incluye etiquetas o delimitadores que describen las relaciones jerárquicas y la organización de los datos. Por ejemplo, un archivo JSON o XML permite representar datos como un conjunto de objetos anidados, con propiedades que pueden variar entre registros. Esto lo hace ideal para almacenar información heterogénea o con esquemas dinámicos. En un mundo donde las aplicaciones generan grandes volúmenes de datos en múltiples formatos, la información

semi-estructurada se ha convertido en un componente esencial para el intercambio y almacenamiento eficiente de datos.

## Importancia del almacenamiento de datos semi-estructurados en bases de datos relacionales

La capacidad de almacenar datos semi-estructurados en bases de datos relacionales ha ganado relevancia en los últimos años debido a la creciente complejidad y diversidad de los datos que las empresas manejan. Los sistemas relacionales, diseñados tradicionalmente para datos altamente estructurados, han evolucionado para integrar formatos más flexibles como JSON, permitiendo almacenar datos jerárquicos sin comprometer el rendimiento ni la integridad.

Esto ofrece una solución versátil que combina la robustez de un modelo relacional con la adaptabilidad necesaria para manejar datos de aplicaciones web, servicios en la nube y sistemas móviles. Además, esta integración facilita la interoperabilidad entre sistemas heterogéneos, optimizando el flujo de información en organizaciones de cualquier tamaño.

## Objetivo de la monografía

El principal objetivo de esta monografía es analizar y comprender cómo SQL Server ha incorporado el manejo de información semi-estructurada mediante el soporte nativo para JSON. A lo largo del documento, se explorarán las características de este formato, las funcionalidades que SQL Server ofrece para su almacenamiento y manipulación, y las prácticas recomendadas para optimizar su uso. Finalmente, se presenta un estudio de caso que ilustra cómo una empresa puede beneficiarse del uso de JSON para gestionar productos, resaltando tanto los beneficios como los desafíos asociados a esta tecnología.

# Información Semi-estructurada en SQL Server

## Definición y características

En el contexto de SQL Server, la información semi-estructurada se refiere a datos que tienen una organización interna mediante etiquetas, claves y valores, pero que no necesariamente se ajustan a las filas y columnas tradicionales de las bases de datos relacionales. Este tipo de datos, como los formatos JSON y XML, se almacena como texto en columnas de tipo NVARCHAR o similares, permitiendo conservar su estructura jerárquica. Entre las características más importantes se encuentran la capacidad de anidación, el soporte para datos heterogéneos y la flexibilidad para representar relaciones complejas entre atributos. SQL Server ha implementado un conjunto robusto de herramientas para trabajar con JSON, permitiendo a los desarrolladores realizar tareas como validación, extracción y generación de datos con facilidad.

## Soporte JSON en SQL Server

SQL Server incluye soporte nativo para JSON desde su versión 2016, marcando un hito en la evolución del gestor hacia la integración de datos semi-estructurados. Con JSON, es posible almacenar datos en columnas de texto, validar su formato y realizar consultas directamente sobre su contenido utilizando funciones específicas. Por ejemplo, las funciones JSON\_VALUE y JSON\_QUERY permiten extraer valores escalares y objetos completos, respectivamente. Además, la función OPENJSON convierte datos JSON en filas y columnas, lo que facilita su manipulación en un contexto relacional. Esta capacidad ha ampliado el alcance de SQL Server, haciéndolo una herramienta más poderosa para las aplicaciones modernas.

## Ventajas de utilizar JSON en SQL Server

El uso de JSON en SQL Server presenta varias ventajas. En primer lugar, permite manejar datos jerárquicos de forma eficiente dentro de un sistema relacional, eliminando la necesidad de diseñar esquemas complejos para datos anidados. En segundo lugar, mejora la interoperabilidad con APIs y aplicaciones que utilizan JSON como formato estándar de intercambio, reduciendo la necesidad de transformaciones adicionales. Finalmente, proporciona flexibilidad para gestionar datos con esquemas cambiantes, lo que es especialmente útil en aplicaciones ágiles y en evolución constante. Estos beneficios hacen de JSON una herramienta esencial para las organizaciones que buscan combinar lo mejor de los modelos relacionales y los no relacionales.

# Almacenamiento de Información Semi-estructurada

## Tipo de columna para almacenar JSON

En SQL Server, la información JSON se almacena típicamente en columnas de tipo NVARCHAR. Este tipo de datos es ideal porque permite representar texto con codificación Unicode, asegurando la compatibilidad con caracteres de diferentes idiomas y scripts.

Además, SQL Server no requiere que las columnas que almacenan JSON tengan un tamaño fijo, lo que proporciona flexibilidad al manejar datos de longitud variable. Aunque SQL Server no tiene un tipo de datos dedicado exclusivamente a JSON, el uso de NVARCHAR permite almacenar estos datos sin restricciones, combinando el formato JSON con la potencia de un sistema relacional.

## Validación de datos JSON

La validación de datos JSON es crucial para garantizar la integridad y consistencia de la información almacenada. SQL Server proporciona la función ISJSON, que verifica si un valor cumple con el formato JSON. Esto es especialmente útil al insertar datos, ya que permite prevenir errores en el almacenamiento. Por ejemplo, si una columna está diseñada para almacenar datos JSON relacionados con productos, se puede agregar una restricción CHECK que utilice ISJSON para validar los datos antes de su inserción. Esto asegura que solo se almacenen datos válidos y estructurados correctamente.

ALTER TABLE Productos

ADD CONSTRAINT chk\_json\_validacion CHECK (ISJSON(Detalles) = 1);

## Inserción y almacenamiento de datos

El proceso de inserción de datos JSON en SQL Server es similar al de cualquier otro tipo de datos, con la diferencia de que los valores deben representarse en formato JSON válido. Los desarrolladores pueden insertar datos directamente en columnas de tipo NVARCHAR utilizando sentencias SQL estándar. Por ejemplo:

INSERT INTO Productos (Id, Detalles)

VALUES (1, '{"nombre": "Laptop", "precio": 1500, "stock": 25}');

SQL Server también permite trabajar con datos JSON dinámicos, lo que simplifica la actualización de propiedades específicas sin necesidad de modificar toda la estructura.

# Manipulación de Información JSON en SQL Server

## Extracción de datos

SQL Server proporciona herramientas poderosas para extraer datos específicos de documentos JSON almacenados en columnas de texto. Estas herramientas permiten acceder a valores escalares, objetos anidados o convertir estructuras JSON completas en un formato relacional.

## Uso de JSON\_VALUE

La función JSON\_VALUE se utiliza para extraer valores escalares, como cadenas de texto o números, de un documento JSON. Su sintaxis básica incluye el nombre de la columna y la ruta al elemento que se desea extraer. Por ejemplo, si se tiene una columna Detalles con el siguiente JSON:

{"nombre": "Laptop", "precio": 1500, "stock": 25}

Se puede extraer el valor del precio así:

SELECT JSON\_VALUE(Detalles, '$.precio') AS Precio FROM Productos;

Esto devolverá el valor 1500, permitiendo utilizarlo en cálculos o filtrados.

## Uso de JSON\_QUERY

Cuando se necesita extraer un objeto o arreglo completo dentro de un JSON, se utiliza JSON\_QUERY. A diferencia de JSON\_VALUE, esta función no está limitada a valores escalares. Por ejemplo:

SELECT JSON\_QUERY(Detalles, '$') AS ObjetoCompleto FROM Productos;

Esto devolverá todo el contenido del JSON en la columna Detalles. También es útil para trabajar con estructuras jerárquicas más complejas.

## Conversión de JSON a tablas con OPENJSON

La función OPENJSON convierte datos JSON en filas y columnas, haciendo posible su manipulación como si se tratara de datos relacionales. Por ejemplo, un JSON con información de productos:

[

{"nombre": "Laptop", "precio": 1500},

{"nombre": "Tablet", "precio": 800}

]

Puede transformarse en una tabla con la siguiente consulta:

SELECT \*

FROM OPENJSON('[{"nombre": "Laptop", "precio": 1500},

{"nombre": "Tablet", "precio": 800}]') WITH (

nombre NVARCHAR(50), precio INT

);

Esto generará una tabla con las columnas nombre y precio, facilitando su integración en otras consultas.

## Generación de JSON desde datos relacionales

SQL Server permite convertir datos de tablas relacionales en estructuras JSON utilizando funciones específicas.

## Función FOR JSON PATH

La función FOR JSON PATH genera un documento JSON con un control granular sobre la estructura jerárquica. Por ejemplo:

SELECT nombre, precio FROM Productos

FOR JSON PATH;

Esto produce un JSON como:

[

{"nombre": "Laptop", "precio": 1500},

{"nombre": "Tablet", "precio": 800}

]

Si se desea personalizar aún más, se pueden especificar rutas explícitas para los atributos.

## Función FOR JSON AUTO

La función FOR JSON AUTO genera un JSON automáticamente basado en la estructura de las tablas y las relaciones entre ellas. Por ejemplo:

SELECT \*

FROM Productos FOR JSON AUTO;

Esto produce un JSON donde cada fila se convierte en un objeto. Aunque es menos flexible que FOR JSON PATH, es más rápida de implementar en casos simples.

# Prácticas Recomendadas

## Validación de JSON al insertar

Para garantizar la integridad de los datos, siempre es recomendable validar el formato JSON antes de almacenarlo. Esto puede lograrse utilizando la función ISJSON en combinaciones con restricciones o triggers en la base de datos.

## Uso de índices para mejorar el rendimiento

SQL Server permite crear índices sobre propiedades específicas dentro de un JSON utilizando columnas calculadas. Por ejemplo, si se accede frecuentemente al atributo precio de un JSON, se puede crear una columna calculada e indexarla:

ALTER TABLE Productos

ADD PrecioPersisted AS JSON\_VALUE(Detalles, '$.precio') PERSISTED;

CREATE INDEX idx\_precio ON Productos (PrecioPersisted);

Esto mejora significativamente el rendimiento en consultas.

## Decisiones estratégicas entre JSON y columnas tradicionales

Es importante evaluar cuándo usar JSON en lugar de columnas tradicionales. JSON es ideal para datos con esquemas dinámicos, pero para datos altamente estructurados y accesados frecuentemente, las columnas tradicionales suelen ser más eficientes.

# Estudio de Caso: Gestión de Productos con JSON

## Escenario

Una empresa gestiona su inventario de productos electrónicos. Cada producto tiene atributos comunes como nombre y precio, pero también características específicas que varían según el tipo de producto (por ejemplo, resolución para televisores, autonomía para laptops). Para evitar diseñar un esquema rígido, la empresa opta por almacenar las especificaciones en formato JSON dentro de una columna.

## Implementación técnica

La empresa crea una tabla Productos con una columna Especificaciones para almacenar los datos JSON. El siguiente script ilustra el diseño e inserción de datos:

CREATE TABLE Productos ( Id INT PRIMARY KEY, Nombre NVARCHAR(50),

Especificaciones NVARCHAR(MAX)

);

INSERT INTO Productos (Id, Nombre, Especificaciones) VALUES

(1, 'Laptop', '{"procesador": "Intel i7", "ram": "16GB", "autonomia": "10 horas"}'),

(2, 'Televisor', '{"resolucion": "4K", "tamaño": "55 pulgadas"}');

Además, la empresa utiliza JSON\_VALUE y OPENJSON para realizar consultas sobre atributos específicos.

## Resultados obtenidos

La implementación demostró ser eficiente para manejar datos heterogéneos, permitiendo añadir nuevos atributos sin modificar la estructura de la tabla. Las consultas específicas sobre propiedades JSON se realizaron con un rendimiento adecuado gracias al uso de índices. Sin embargo, se identificaron desafíos, como la necesidad de validar y optimizar consultas sobre datos anidados.

# Conclusiones

## Beneficios del uso de JSON en SQL Server

El soporte para JSON en SQL Server brinda flexibilidad y capacidad para manejar datos semi-estructurados sin sacrificar la robustez de un sistema relacional. Es especialmente útil para aplicaciones modernas que necesitan integrar datos de diferentes fuentes.

## Desafíos en la gestión de datos semi-estructurados

A pesar de sus beneficios, el uso de JSON en SQL Server presenta desafíos como la validación de datos y el rendimiento en consultas complejas. Una implementación exitosa requiere un diseño cuidadoso y el uso de prácticas recomendadas para garantizar la eficiencia y consistencia de los datos.

# Bibliografía

* Microsoft. (2023). *JSON Functions in SQL Server*. Recuperado de [https://learn.microsoft.com](https://learn.microsoft.com/)
* Hernández, C. (2021). *Bases de Datos: Fundamentos y Aplicaciones*. Editorial Alfaomega.
* Ben-Gan, I. (2022). *T-SQL Querying*. Microsoft Press.
* Date, C. J. (2020). *An Introduction to Database Systems*. Pearson Education.
* Rob, P., & Coronel, C. (2021). *Database Systems: Design, Implementation, and Management*. Cengage Learning.
* Stack Overflow. (2023). *Best Practices for Using JSON in SQL Server*. Recuperado de [https://stackoverflow.com](https://stackoverflow.com/)