SQL KONFERENZ 2026

JSON in SQL Server

Building Data Like LEGO

Erik Esmaty – March 4th, 2026



Who am I?



```
{
    "speaker": {
        "name": "Erik Esmaty",
        "role": "Team Lead",
        "company": "DYMATRIX GmbH",
        "location": "Munich, Germany",
        "profiles": {
            "linkedin": "https://www.linkedin.com/in/esmaty",
            "x": "@erikesmaty"
        },
        "funFact": "Almost traveled to the wrong city for a SQL Saturday",
        "sessionGoal": "Show how JSON becomes a practical everyday tool in SQL Server"
    }
}
```

SQL KONFERENZ 2026

Thank you to the sponsors!



Event feedback

SQL KONFERENZ 2026

Agenda

- Why are we here?
- JSON Building Blocks
- How SQL Server Slowly Learned to Love JSON
- Demos
- Tools



Why are we here?





- JSON is everywhere: logs, APIs, microservices, events
- SQL Server handles JSON better than many expect
- Session goal: understand JSON, query it, combine it, enjoy it
- Don’t worry, JSON is just text with brackets



What does JSON stand for?

JavaScript Object Notation

JSON was created in the early 2000s by Douglas Crockford as a lightweight, language-independent data format derived from JavaScript syntax to make web data exchange simpler and more efficient.

JSON Building Blocks

Objects:

```
{
    "name": "Erik",
    "city": "Munich"
}
```

Arrays:

```
[
    "SQL",
    "JSON",
    "Munich"
]
```

JSON is flexible but **not forgiving**.

String, Number, Boolean, Null, Object, Array

JSON is strict!

- Quotes must match
- Commas must be correct
- Brackets must close
- No trailing commas
- Whitespace doesn't matter

Example

```
{
    "userId": 42,
    "active": true,
    "roles": [
        "admin",
        "editor"
    ],
    "profile": {
        "name": "Erik",
        "city": "Munich"
    }
}
```

- Start at the top level
- Identify objects vs. arrays
- Follow the nesting
- Think in “paths”:
  profile.city
  roles[0]

Array examples

```
{
    "cities": [
        "Munich",
        "Berlin",
        "Hamburg",
        "Cologne"
    ],
    "temperatures": [3,7,12,18],
    "events": [
        {
            "name": "SQL Konferenz",
            "city": "Hanau"
        },
        {
            "name": "Data Saturday",
            "city": "St. Augustin"
        },
        {
            "name": "Data Saturday",
            "city": "Munich"
        }
    ]
}
```



But why are we not just storing the data in columns?

- **Schema flexibility**
  JSON allows evolving structures without schema migrations.
- **Reduced impedance mismatch**
  Applications already work with JSON. Storing it avoids repetitive serialization/deserialization.
- **Partial indexing is possible**
  Frequently accessed JSON paths can be indexed (computed columns or JSON indexes in 2025).
- **Hybrid modeling**
  Combine relational structure for core data with JSON for variable attributes.
- **Better performance than file-based storage**
  Querying JSON inside the database is faster and safer than reading external files.



SQL KONFERENZ 2026

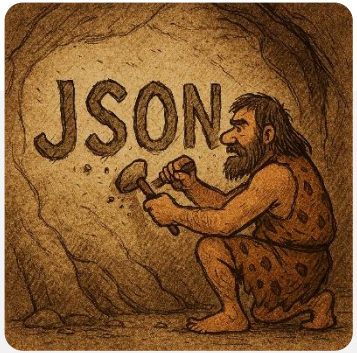
SQL
KONFERENZ
2026



Now that we know how to read JSON like ~~data~~ ~~monsters~~ humans, let's see how SQL Server uses it!

SQL 2016

Where it all began



- JSON text field
  Plain text in an nvarchar(max)-field.
- ISJSON()
  Checks whether a string contains valid JSON. Returns 1 (valid) or 0 (invalid).
- JSON_QUERY()
  Extracts a JSON object or array, when the result is structured, e.g. "roles" or "items".
- JSON_VALUE()
  Extracts a single scalar value (string, number, boolean) from JSON.
- OPENJSON()
  Turns a JSON array into rows. Anything with multiple elements.
- FOR JSON
  Converts SQL query results into JSON output. Useful for APIs or exporting data.

SQL KONFERENZ 2026



SQL KONFERENZ 2026

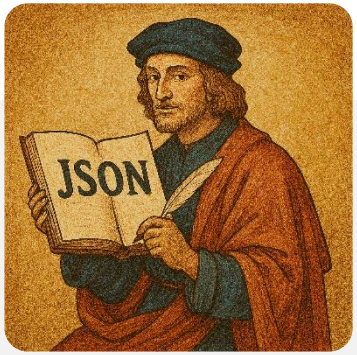
SQL 2017

Where it continued

- No new JSON functions introduced
- Improved performance for OPENJSON, JSON_VALUE, JSON_QUERY
- More stable execution plans for JSON-heavy workloads
- Reduced CPU usage when parsing large JSON documents
- Better memory handling for large NVARCHAR(MAX) JSON payloads
- Foundation work for later UTF-8 improvements (fully delivered in SQL 2019)

SQL 2019

… and further continued



- Full UTF-8 support, enabling smaller JSON storage and faster I/O
- Improved performance for OPENJSON on large arrays
- More stable execution plans for JSON-heavy workloads
- Reduced memory pressure when parsing NVARCHAR(MAX) JSON documents
- Better integration with UTF-8 external systems (APIs, microservices, message queues)
- Still **no new JSON functions**. Same feature set as 2016/2017, but faster and more efficient

SQL KONFERENZ 2026

… and further continued again



SQL 2022

- Significant performance improvements for OPENJSON, especially on large arrays
- Faster execution of JSON_VALUE and JSON_QUERY through engine-level optimizations
- More stable and predictable execution plans for JSON-heavy workloads
- Improved performance for JSON-based computed columns
- Better indexing behavior when indexing JSON values via computed columns
- Reduced CPU overhead when parsing large NVARCHAR(MAX) JSON documents
- Still no new JSON functions, but the fastest and most stable JSON engine so far

SQL KONFERENZ 2026



SQL KONFERENZ 2026

… ending up in SQL 2025

- Native JSON data type (binary, compressed, indexable)
- JSON_MODIFY method for the new type
- JSON_CONTAINS for containment checks
- ANSI SQL JSON path support + array wildcards
- JSON_QUERY … WITH ARRAY WRAPPER
- CREATE JSON INDEX for fast JSON path lookups

SQL 2025

SQL
KONFERENZ
2026



Demo-Time



Index on JSON Columns – What works today?

- JSON indexes can be created on a JSON column and a specific JSON path
- Useful for equality checks and existence checks on the indexed path
- Good for scenarios where the JSON path is accessed frequently
- Index creation is supported in SQL Server 2025 (Preview)

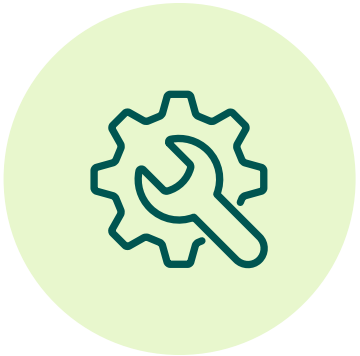
SQL KONFERENZ 2026



Index on JSON Columns – What doesn't work today

- JSON_VALUE() and JSON_QUERY() do not trigger JSON-index seeks
- Range predicates (>, <, BETWEEN) do not use JSON indexes
- The optimizer does not rewrite JSON_VALUE() to a JSON-indexable predicate
- Many queries still fall back to scans despite a JSON index

SQL KONFERENZ 2026



Online JSON Tools

- **JSONLint:**
  Validate & format JSON
- **JSON Formatter & Validator**
  (jsonformatter.org):
  Pretty-print, minify, tree view
- **JSON Editor Online** (jsoneditoronline.org):
  Visual editor with tree + code view
- **QuickType.io**:
  Generate strongly typed classes from JSON
- **BeautifyTools JSON Formatter:**
  Simple formatter/minifier



Editors with JSON Support

VS Code

- Built-in JSON schema validation
- IntelliSense for JSON
- Extensions: “Prettify JSON”, “JSON Crack”, “JSON Tools”

Notepad++

- Plugin: JSTool (format, minify, tree view)
- Plugin: JSON Viewer

Sublime Text (Free evaluation)

- Package: “Pretty JSON”

Session Eval



```
{
    "speaker": {
        "name": "Erik Esmaty",
        "role": "Team Lead",
        "company": "DYMATRIX GmbH",
        "location": "Munich, Germany",
        "profiles": {
            "linkedin": "https://www.linkedin.com/in/esmaty",
            "x": "@erikesmaty"
        },
        "funFact": "Almost traveled to the wrong city for a SQL Saturday",
        "sessionGoal": "Show how JSON becomes a practical everyday tool in SQL Server"
    }
}
```