How Well Do Relational Database Engines Support JSON?

Christian Antognini

@ChrisAntognini # ITOUGTD19

trivadis
makes IT easier.

@ChrisAntognini

Senior principal consultant, trainer and partner at Trivadis

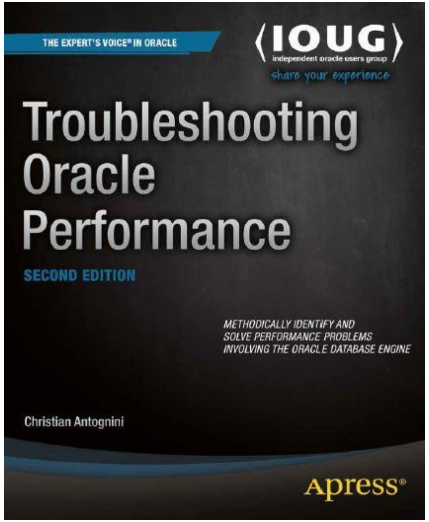
- christian.antognini@trivadis.com

Focus: get the most out of database engines

- Logical and physical database design
- Query optimizer
- Application performance management

Author of *Troubleshooting Oracle Performance* (Apress, 2008/14)

OakTable Network, Oracle ACE Director



THE EXPERT'S VOICE® IN ORACLE

IOUG independent oracle users group share your experience

Troubleshooting Oracle Performance

SECOND EDITION

METHODICALLY IDENTIFY AND SOLVE PERFORMANCE PROBLEMS INVOLVING THE ORACLE DATABASE ENGINE

Christian Antognini

Apress®

Agenda

1. Introduction
2. Generating JSON Data
3. Storing JSON Data
4. Querying JSON Data
5. Summary
6. Bibliography

Introduction

Multi-Model Database Engines

A single data model does not fit all requirements

Most of the time the effort of using several database engines is too high

Most leading database engines (incl. RDBMS) are *de facto* multi model

Even though data is stored in a relation model, it might be required to provide it in another format

JSON Use Cases

Export data in JSON format

- Predefined JSON schema
- Plain format instead of a CSV file

Import JSON data

- Convert data to relational schema
- Store data as JSON

Implement a JSON document store

Extend a relational store

- Support an extensible schema
- Handle a high number of (sparse) columns

SQL/JSON

SQL:2016 introduces 44 new optional features; 22 of them are related to JSON

The new features cover the following use cases:

- With SQL queries declaratively generate JSON data from relation data
- Persistently store JSON data into a database
- Use SQL queries to access JSON data according to its structure

What the standard does not cover are features to modify a JSON document

Database Engines Under Investigation

MySQL® Community Server 8.0.14

ORACLE® DATABASE Enterprise Edition 18.5

PostgreSQL 11.1

Microsoft® SQL Server® Enterprise Edition 2017 (14.0.3048.4)

Generating JSON Data

Constructor Functions

SQL:2016 specifies four constructor functions to generate JSON data from relational data

Construct a JSON object

- JSON_OBJECT (T811, T814, T830)
- JSON_OBJECTAGG (T812, T814, T830)

Construct a JSON array

- JSON_ARRAY (T811)
- JSON_ARRAYAGG (T811, T813)

f10
delete
+
=
)
0
—
-
D
E
M
O
"
'
:
;
retu

Constructor Functions – Implementations

MySQL

It provides the functions, but the implementation differs from SQL:2016 and provides less control of the output; a number of additional functions are available

ORACLE DATABASE

It supports most of the functionality; some “advanced” functionality is missing

PostgreSQL

It provides functions with similar names, the implementation almost matches the one of MySQL; a number of additional functions are available

Microsoft SQL Server

It uses a completely different approach based on the FOR JSON clause

Storing JSON Data

trivadis
makes IT easier.

Digression – Designing JSON Documents

Representing data as JSON can be considerably more flexible than the traditional relational data model

Both approaches can co-exist and complement each other within the same application

If the JSON documents do not have a somewhat fixed structure, it is challenging to query their contents!

JSON Data Type

SQL:2016 does *not* specify a new native data type for JSON data

The proposed approach is to ingest character or binary strings that are stored in ordinary data types

Pros

- Easier to implement by both database engines and tools

Cons

- No automatic validation
- Not optimized storage format that could lead to suboptimal access performance

JSON Data Type – Implementations

MySQL Implements the data type JSON

ORACLE DATABASE Implements no data type; VARCHAR2 or CLOB are used

PostgreSQL Implements two data types, JSON and JSONB

Microsoft SQL Server Implements no data type; NVARCHAR is used

IS JSON Predicate

SQL:2016 specifies a predicate to test the validity of a JSON document

- Without uniqueness constraint (T821)
  - No type constraint

```
CHECK ( <column_name> IS JSON )
```

  - With type constraint

```
CHECK ( <column_name> IS JSON [ VALUE | ARRAY | OBJECT | SCALAR ] )
```

- With uniqueness constraint (T822)

```
CHECK ( <column_name> IS JSON WITH UNIQUE [ KEYS ] )
```

IS JSON Predicate – Implementations

MySQL Data type validates content; no type and uniqueness constraints; uniqueness is forced (last duplicate key wins)

ORACLE DATABASE Implements predicate; no type constraint

PostgreSQL Data types validate content; no type and uniqueness constraints; uniqueness is forced (last duplicate key wins) with JSONB only

Microsoft SQL Server Implements ISJSON function; no type and uniqueness constraints

Querying JSON Data

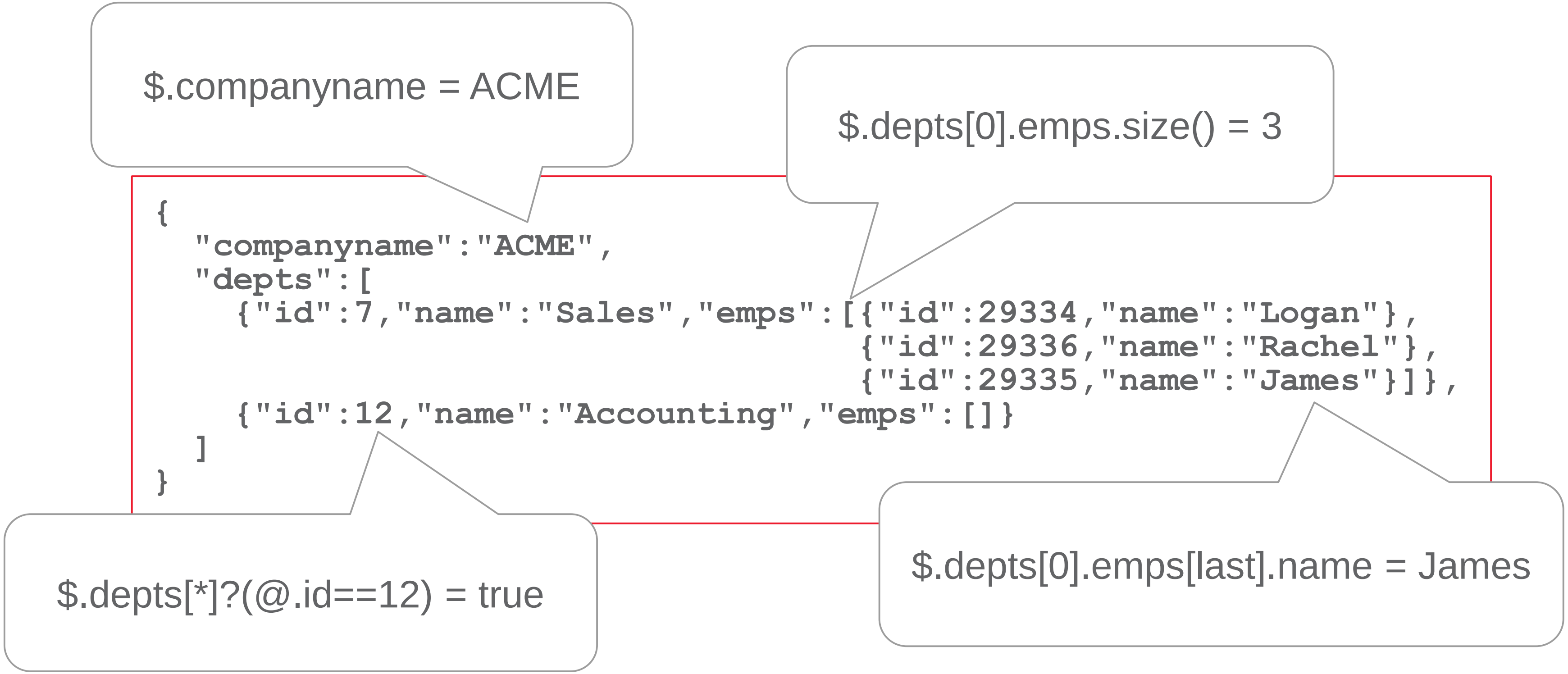
SQL/JSON Path Language

It is a language to query JSON data

SQL:2016 specifies it (T831-T837)

Lexically and syntactically, it adopts many features of ECMAscript, though it is neither a subset nor a superset of ECMAscript

SQL/JSON Path Language – Examples



$.companyname = ACME

$.depts[0].emps.size() = 3

```
{
  "companyname":"ACME",
  "depts":[
    {"id":7,"name":"Sales","emps":[{"id":29334,"name":"Logan"},
                                   {"id":29336,"name":"Rachel"},
                                   {"id":29335,"name":"James"}]},
    {"id":12,"name":"Accounting","emps":[]}
  ]
}
```

$.depts[*]?(@.id==12) = true

$.depts[0].emps[last].name = James

Query Operators

SQL:2016 specifies four query operators to evaluate SQL/JSON path language expressions (T831-T837) against JSON data

- JSON_EXISTS – determine whether a path expression is satisfied (T821, T823, T825)
- JSON_VALUE – extract a scalar value (T821, T823, T825, T826)
- JSON_QUERY – extract a non-scalar value (T828, T823, T825, T829)
- JSON_TABLE – generate relational data (T821, T823-T827, T838)

f10
delete
+
=
—
-
)
0
D
E
M
O
"
;
:
retu

Query Functions – Implementations

MySQL®	It provides only JSON_TABLE; a number of additional functions are available
ORACLE® DATABASE	It supports the basic functionality for all four functions; some “advanced” functionality is missing
PostgreSQL	A number of functions and operators are available
Microsoft® SQL Server®	It provides only JSON_VALUE and JSON_QUERY; two additional functions are available

Indexing JSON Data

All four database engines supports indexes created on scalar values extracted via, for example, the JSON_VALUE function

```
CREATE INDEX idx ON company (json_value(json, '$.companyname'))
```

In PostgreSQL indexes are supported for JSONB only

Obviously, if the structure of a JSON document is not known, it cannot be easily indexed and queried!

- For a number of use cased inverted indexes are not good enough

Summary

Support of SQL/JSON

MySQL®	Limited
ORACLE® DATABASE	It fully supports 5 features and partially supports 10 others
PostgreSQL	Limited (patches that are in review in the current CommitFest exist)
Microsoft® SQL Server®	Limited

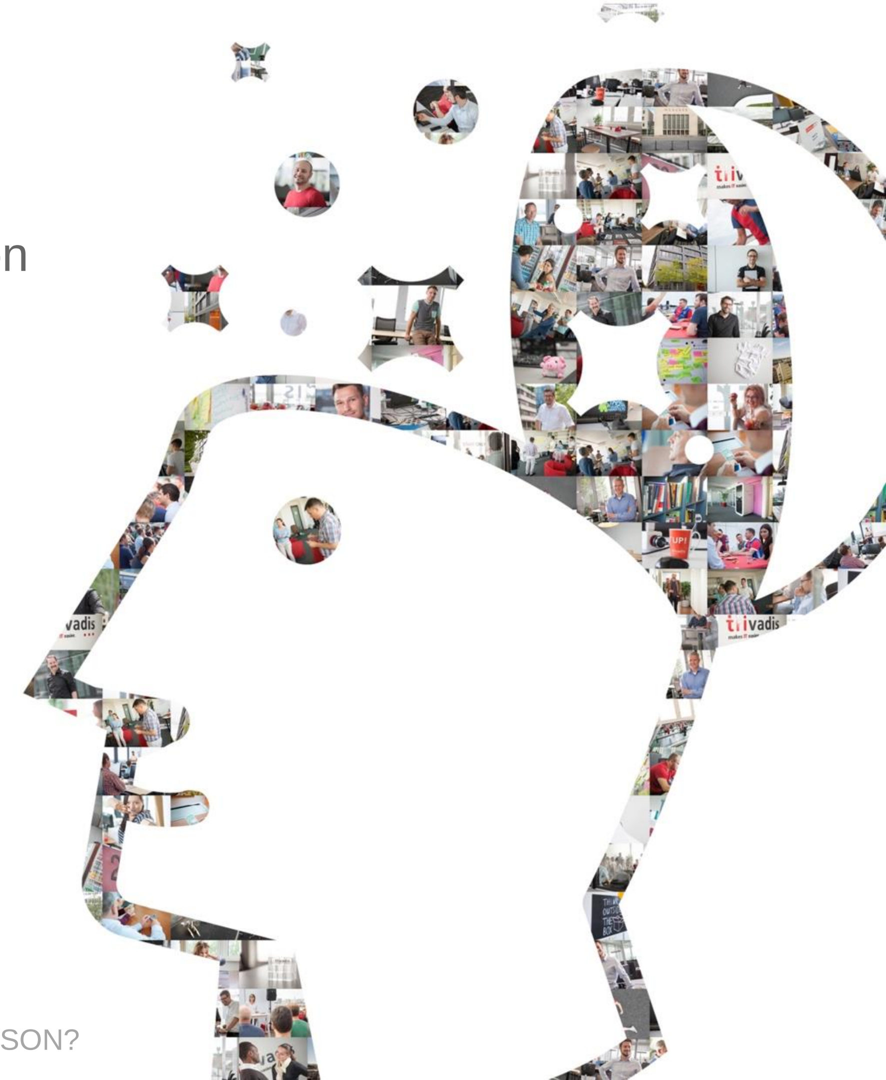
Summary

The four database engines under investigation provides good JSON support

- If only part of the data has to be stored as JSON, reducing the number of involved database engines is a real possibility
- They should get better and better

In general, the SQL/JSON support is weak

- Because it was introduced in SQL:2016, it is not surprising
- Oracle Database has the best support



Bibliography

Bibliography

ISO/IEC, *9075-2:2016 – Part 2: Foundation (SQL/Foundation)*. 2016

ISO/IEC, *SQL Technical Reports – Part 6: SQL support for JavaScript Object Notation (JSON)*. 2017

Ecma International, *ECMAScript 2018 Language Specification. 2018*

Hammerschmidt, Beda, *JSON in der Oracle Datenbank*. DOAG Konferenz, 2018

Microsoft Corporation, *JSON data in SQL Server*. 2018

Oracle Corporation, *MySQL 8.0 Reference Manual*. 2018

Oracle Corporation, *Oracle Database 18c – JSON Developer's Guide*. 2018

The PostgreSQL Global Development Group, *PostgreSQL 11 Documentation*. 2018

Winand, Markus, *What's New in SQL:2016*. Modern SQL, 2017

Questions and Answers

Christian Antognini
Senior Principal Consultant

christian.antognini@trivadis.com

@ChrisAntognini https://antognini.ch/blog/