BS ISO/IEC 19075-6:2021

BSI Standards Publication

Information technology — Guidance for the use of database language SQL

Part 6: Support for JSON

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

National foreword

This British Standard is the UK implementation of ISO/IEC 19075-6:2021. It supersedes PD ISO/IEC TR 19075-6:2017, which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee IST/40, Data management and interchange.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2021
Published by BSI Standards Limited 2021

ISBN 978 0 539 13083 6

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 September 2021.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

INTERNATIONAL STANDARD

ISO/IEC 19075-6

First edition
2021-08

Information technology — Guidance for the use of database language SQL —

Part 6: Support for JSON

Technologies de l'information — Recommandations pour l'utilisation du langage de base de données SQL —

Partie 6: Support pour JSON

Reference number
ISO/IEC 19075-6:2021(E)

Documents
Document Sharing Hub

© ISO/IEC 2021

Full document details: BS ISO/IEC 19075-6:2021

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

Contents

Page

Foreword. ix
Introduction. xi
1 Scope. 1
2 Normative references. 2
3 Terms and definitions. 3
4 JavaScript Object Notation (JSON). 6
4.1 Context for JSON. 6
4.2 What is JSON?. 6
4.3 Representations of JSON data. 7
4.3.1 Introduction to representations of JSON data. 7
4.3.2 Avro. 7
4.3.3 BSON. 8
4.4 Schemas. 8
4.4.1 JSON schemata and validity. 8
4.4.2 Avro schemata. 9
4.4.3 BSON schemata. 9
4.5 Why does JSON matter in the context of SQL? What is JSON's relationship to NoSQL? 10
4.6 Use cases for JSON support in SQL. 11
4.6.1 Introduction to the use cases. 11
4.6.2 JSON data ingestion and storage. 11
4.6.3 JSON data generation from relational data. 11
4.6.4 Querying JSON as persistent semi-structured data model instances. 12
4.7 What features address those use cases?. 12
4.7.1 Addressing the use cases. 12
4.7.2 Storing JSON data in an SQL table. 12
4.7.3 Generating JSON in an SQL query. 12
4.7.4 Querying JSON data in SQL tables using SQL. 13
5 The SQL/JSON data model. 14
5.1 Introduction to the SQL/JSON data model. 14
5.2 SQL/JSON items. 15
5.2.1 Definition of SQL/JSON items. 15
5.2.2 Atomic values. 16
5.2.3 SQL/JSON arrays. 17
5.2.4 SQL/JSON objects. 17
5.3 SQL/JSON sequences. 17
5.4 Parsing JSON. 18
5.5 Serializing JSON. 18
6 SQL/JSON functions. 19

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

6.1 Introduction to SQL/JSON functions. . . . 19
6.2 Handle JSON using built-in functions. . . . 19
6.3 JSON API common syntax. . . . 19
6.3.1 Introduction to JSON API common syntax. . . . 19
6.3.2 JSON value expression. . . . 20
6.3.3 Path expression. . . . 20
6.3.4 PASSING clause. . . . 20
6.3.5 JSON output clause. . . . 21
6.3.6 ON ERROR and ON EMPTY syntax. . . . 21
6.4 Query functions. . . . 22
6.4.1 The four query functions. . . . 22
6.4.2 JSON_EXISTS. . . . 22
6.4.3 JSON_VALUE. . . . 25
6.4.4 JSON_QUERY. . . . 29
6.4.5 JSON_TABLE. . . . 33
6.4.5.1 Introduction to JSON_TABLE. . . . 33
6.4.5.2 COLUMNS clause that is not nested. . . . 34
6.4.5.3 Nested COLUMNS clause. . . . 36
6.4.5.4 PLAN clause. . . . 37
6.4.6 Conformance features for query operators. . . . 41
6.5 Constructor functions and IS JSON predicate. . . . 43
6.5.1 Tables used to illustrate constructor functions and the IS JSON predicate. . . . 43
6.5.2 JSON_OBJECT. . . . 44
6.5.3 JSON_OBJECTAGG. . . . 45
6.5.4 JSON_ARRAY. . . . 46
6.5.5 JSON_ARRAYAGG. . . . 47
6.5.6 IS JSON predicate. . . . 49
6.5.7 Handling of JSON nulls and SQL nulls. . . . 49
6.5.8 Conformance features for constructor functions. . . . 50
7 SQL/JSON path language. . . . 51
7.1 Overview of SQL/JSON path language. . . . 51
7.2 Objectives for the SQL/JSON path language. . . . 52
7.3 Modes. . . . 53
7.3.1 Introduction to modes. . . . 53
7.3.2 Example of strict *vs* lax. . . . 54
7.4 Lexical issues. . . . 56
7.5 Syntax summary. . . . 58
7.6 Formal semantics and notational conventions. . . . 58
7.7 Primitive operations in formal semantics. . . . 59
7.7.1 Concatenation. . . . 59
7.7.2 unwrap(). . . . 59
7.7.3 wrap(). . . . 60
7.8 Mode declaration. . . . 60
7.9 <JSON path primary>. . . . 61
7.9.1 Introduction to JSON path primaries. . . . 61
7.9.2 Literals. . . . 61
7.9.3 Variables. . . . 62

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

7.9.4 Parentheses. . . . 63
7.10 Accessors. . . . 63
7.10.1 Introduction to accessors. . . . 63
7.10.2 Member accessor. . . . 64
7.10.3 Member wildcard accessor. . . . 67
7.10.4 Element accessor. . . . 68
7.10.5 Element wildcard accessor. . . . 70
7.10.6 Sequence semantics of the accessors. . . . 71
7.11 Item methods. . . . 71
7.11.1 Introduction to item methods. . . . 71
7.11.2 type(). . . . 72
7.11.3 size(). . . . 72
7.11.4 Numeric item methods (double, ceiling, floor, abs). . . . 73
7.11.5 datetime(). . . . 73
7.11.6 keyvalue(). . . . 73
7.12 Arithmetic expressions. . . . 75
7.12.1 Introduction to arithmetic expressions. . . . 75
7.12.2 Unary plus and minus. . . . 75
7.12.3 Binary operations. . . . 76
7.13 Filter expression. . . . 76
7.13.1 Introduction to filter expressions. . . . 76
7.13.2 `true`/`false` and *True*/*False* . . . 77
7.13.3 `null` and *Unknown*. . . . 78
7.13.4 Error handling in filters. . . . 78
7.13.5 Truth tables. . . . 81
7.13.6 Comparison predicates. . . . 82
7.13.7 `like_regex` predicate. . . . 83
7.13.8 `starts with` predicate. . . . 84
7.13.9 `exists` predicate. . . . 84
7.13.10 `is unknown` predicate. . . . 86
7.14 Conformance features for SQL/JSON path language. . . . 87

Bibliography. . . . 88

Index. . . . 89

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

Tables

Table

Page

1 JSON, SQL/JSON, and SQL (other than SQL/JSON values and counterparts). . . . 14
2 Parallels between JSON text and SQL/JSON data model. . . . 18
3 JSON_EXISTS sample data. . . . 22
4 Result of the sample query. . . . 23
5 Accessor example. . . . 24
6 Result 1. . . . 26
7 Result 2. . . . 26
8 Result 3. . . . 27
9 Result 4. . . . 27
10 Result 5. . . . 28
11 Result 6. . . . 29
12 JSON_EXISTS results. . . . 30
13 ON EMPTY results. . . . 31
14 ARRAY WRAPPER results. . . . 31
15 Illustrating differences. . . . 32
16 Comparison of wrapper options. . . . 32
17 JSON_TABLE sample data in a book recommendation table. . . . 33
18 Unnested query result. . . . 35
19 Nested query result. . . . 36
20 PLAN query result. . . . 39
21 Second PLAN query result. . . . 39
22 Third PLAN query result. . . . 40
23 DEPTS table. . . . 43
24 JOBS table. . . . 43
25 EMPLOYEES table. . . . 44
26 The JSON object returned. . . . 45
27 Returned JSON object with the corresponding job sequence number. . . . 46
28 ARRAYAGG query result. . . . 48
29 Second ARRAYAGG query result. . . . 48
30 Third ARRAYAGG query result. . . . 49
31 Three aspects of path evaluation governed by modes. . . . 54
32 Example of strict *vs* lax. . . . 55
33 Features of the SQL/JSON path language. . . . 58
34 Data used by unwrap() example. . . . 59
35 Data used by wrap() example. . . . 60
36 Examples of atomic values in the SQL/JSON path language 61
37 Examples of the escaping rules. . . . 61
38 Evaluation of `'$.phones.type'` in lax mode. . . . 65
39 Intermediate step. . . . 65
40 Evaluation of`'$.phones[*].type'` 66
41 Evaluation of `'$.phones[*] ? (exists (@.type)).type'` 66
42 Evaluation of `'$.phones.*'` in lax mode. . . . 67
43 Evaluation of `'$.phones[*].*'`. . . . 67
44 Evaluation of `'lax $.sensors.*[0, last, 2]'`. . . . 69
45 The step in the evaluation 71
46 Result of the query with the sample data 74
47 Evaluation of `'lax -$.readings.floor()'`. . . . 75

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

48 Evaluation of `'lax (-$.readings).floor()'` . . . 76
49 Table T with two rows . . . 78
50 Computation on row K=102 . . . 79
51 Modified table T . . . 79
52 Computation on row K=102 in modified table T. . . . 79
53 Computation of `'lax $ ? (@.hours > 9)'` on row K=102 . . . 80
54 Computation of `'strict $ ? (@.hours > 9)'` in strict mode . . . 80
55 Result of `&&`. . . . 81
56 Result of `||`. . . . 81
57 Result of `!`. . . . 81
58 Supported comparisons . . . 82
59 Final result . . . 83
60 A table with JSON column . . . 85
61 Evaluation of `'strict $ ? (exists (@.name)).name'` on row K=201 . . . 85
62 Evaluation of `'strict $ ? (exists (@.name)).name'` on row K=202 . . . 85

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

Figures

Figure

Page

1 Relationships between “JSON” and “SQL/JSON”. 14
2 The SQL/JSON path language architecture. 51

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents), or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This first edition of ISO/IEC 19075-6 cancels and replaces ISO/IEC TR 19075-6:2017. Several small technical errors have been corrected,

This document is intended to be used in conjunction with the following editions of the parts of the ISO/IEC 9075 series:

— ISO/IEC 9075-1, sixth edition or later;

— ISO/IEC 9075-2, sixth edition or later;

— ISO/IEC 9075-3, sixth edition or later;

— ISO/IEC 9075-4, seventh edition or later;

— ISO/IEC 9075-9, fifth edition or later;

— ISO/IEC 9075-10, fifth edition or later;

— ISO/IEC 9075-11, fifth edition or later;

— ISO/IEC 9075-13, fifth edition or later;

— ISO/IEC 9075-14, sixth edition or later;

— ISO/IEC 9075-15, second edition or later;

— ISO/IEC 9075-16, first edition or later.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

A list of all parts in the ISO/IEC 19075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/-national-committees.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

Introduction

The organization of this document is as follows:

1) Clause 1, "Scope", specifies the scope of this document.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this document, constitute provisions of this document.

3) Clause 3, "Terms and definitions", defines the terms and definitions used in this document.

4) Clause 4, "JavaScript Object Notation (JSON)", introduces what is JSON.

5) Clause 5, "The SQL/JSON data model", introduces the data model that is used by the SQL/JSON functions and the SQL/JSON path language.

6) Clause 6, "SQL/JSON functions", introduces the SQL/JSON functions to query and construct JSON.

7) Clause 7, "SQL/JSON path language", introduces the SQL/JSON path language.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19075-6:2021

Information technology — Guidance for the use of database language SQL —

Part 6:
Support for JSON

1 Scope

This document describes the support in SQL for JavaScript Object Notation.

This document discusses the following features of the SQL language:

— Storing JSON data.

— Publishing JSON data.

— Querying JSON data.

— SQL/JSON data model and path language.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9075-1, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*

ISO/IEC 9075-2, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1 and the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at `http://www.electropedia.org/`

— ISO Online browsing platform: available at `http://www.iso.org/obp`

3.1
JSON text
sequence of tokens

Note 1 to entry: JSON text must be encoded in Unicode (UTF-8 by default); insignificant whitespace may be used anywhere in JSON text except within strings (where all whitespace is significant), numbers, and literals. JSON text is a single object or array

3.2
token
strings, numbers, literals, or one of six structural characters

Note 1 to entry: The six structural characters are "`{`", "`}`", "`[`", "`]`", "`:`", "`,`"

3.3
value
object, array, number, string, or one of three literals

3.4
type
primitive type or structured type

3.5
primitive type
string, number, Boolean, or null

3.6
primitive value
value that is a string, number, Boolean, or null

3.7
structured type
object or array

3.8
structured value
value that is an object or an array

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

3.9
string
sequence of Unicode characters

Note 1 to entry: Some characters must be "escaped" by preceding them with a reverse solidus ("\"), while any or all characters can be represented in "Unicode notation" comprising the string '\u' followed by four hexadecimal digits or two such strings representing the UTF-16 surrogate pairs representing characters not on the Basic Multilingual Plane.

Note 2 to entry: Strings are surrounded by double-quote characters, which are not part of the value of the strings.

3.10
number
sequence comprising an integer part, optionally followed by a fractional part and/or an exponent part

Note 1 to entry: Non-numeric values, such as infinity and NaN are not permitted.

3.11
Boolean
literal "`true`" or literal "`false`"

3.12
null
literal "`null`"

3.13
object
structure represented by "`{`", zero or more members separated by "`,`", and "`}`"

3.14
member
string followed by a colon followed by a value in an object

Note 1 to entry: A member is also known as a "name-value pair"; the name is sometimes called a "key" and the second value is sometimes called a "bound value".

3.15
array
structure represented by a "`[`", zero or more elements separated by "`,`", and "`]`"

3.16
element
value in an array

3.17
field
member in an object, element in an array, name in a member, or value in a member

3.18
data model
<general> definition of what kinds of data belong to a particular universe of discourse, including the operations on those kinds of data

3.19
JSON data model
(implicit) data model associated with JSON

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

3.20
SQL/JSON data model
data model created for operating on JSON data within the SQL language

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

4 JavaScript Object Notation (JSON)

4.1 Context for JSON

The requirements for the material discussed in this document shall be as specified in ISO/IEC 9075-1 and ISO/IEC 9075-2.

4.2 What is JSON?

JSON (an acronym for "JavaScript Object Notation") is both a notation (that is, a syntax) for representing data and an implied data model. JSON is not an object-oriented data model; that is, it does not define sets of classes and methods, type inheritance, or data abstraction. Instead, JSON "objects" are simple data structures, including arrays. RFC 8259 says that JSON is a text format for the serialization of structured data. Its initial intended use was as a data transfer syntax.

The complete syntax of JSON is specified in RFC 8259.

The first-class components of the JSON data model are JSON values. A JSON value is one of the following: JSON object, JSON array, JSON string, JSON number, or one of the JSON literals: true, false, and null. A JSON object is zero or more name-value pairs and is enclosed in curly braces — `{ }`. A JSON array is an ordered sequence of zero or more values and is enclosed in square brackets — `[ ]`.

Here is an example of a JSON object:

```
{ "Name" : "Isaac Newton",
  "Weight" : 80,
  "Famous" : true,
  "Phone" : null }
```

The name-value pairs are separated by commas, and the names are separated from the values by colons. The names are always strings and are enclosed in (double) quotation marks.

Here is an example of a JSON array:

```
[ "Robert J. Oppenheimer", 98, false, "Beechwood 45789" ]
```

In a JSON array, the values are separated by commas. JSON arrays and objects are fully nestable. That is, values in both JSON objects and JSON arrays may be JSON strings, JSON numbers, JSON Booleans (represented by the JSON literals true and false), JSON nulls (represented by the JSON literal null), JSON objects, or JSON arrays.

JSON can be used to represent associative arrays — arrays whose elements are addressed by content, not by position. An associative array can be represented in JSON as a JSON object whose members are name-value pairs; the name is used as the "index" into the "array" — that is, to locate the appropriate member in the object — and the value is used as the content of the appropriate member. Here is such an associative array:

```
{ "Isaac Newton" : "apple harvester",
  "Robert J. Oppenheimer": "security risk",
```


© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

```
  "Albert Einstein" : "patent clerk",
  "Stephen Hawking" : "inspiration" }
```

An extremely important part of JSON's design is that it is inherently schema-less. Any JSON object can be modified by adding new name-value pairs, even with names that were never considered when the object was initially created or designed. Similarly, any JSON array can be modified by changing the number of values in the array.

4.3 Representations of JSON data

4.3.1 Introduction to representations of JSON data

Before delving much deeper into the primary topic of this document, readers should understand that JSON data can be represented in several widely-acknowledged and -used forms. The most obvious and most easily recognizable is its "character string" representation, in which JSON data is represented in Unicode characters as plain text. More specifically, explicit characters such as the left square brace, comma, right curly brace, quotation mark, and letters and digits are all used in their native Unicode representation (UTF-8, UTF-16, UTF-32).

However, for a variety of reasons, JSON data is sometimes stored and exchanged in one of several binary representations. For example, a binary representation of JSON data may be significantly smaller (fewer octets) than the character representation of the same data, so a reduction in network bandwidth and or storage media can be achieved by transferring or storing that data in a binary representation.

Readers should note that there are no published standards, either from traditional *de jure* standards organizations nor from consortia or other *de facto* standards groups, for any binary representations of JSON. The two described in this document are frequently used and may be representative of binary JSON representations generally. The following discussion is intended only to illustrate the use of and issues with binary serializations of JSON. The SQL standard leaves it implementation-defined whether or not such representations are supported in any particular SQL-implementation.

4.3.2 Avro

Avro is described as a "data serialization system". As such, its use is not limited to a binary representation or as a compression representation of JSON data. However, a number of JSON environments have chosen Avro as their preferred binary, compressed representation.

Avro has a number of important characteristics that affect its choice as a JSON representation.

— Data is represented in a variable-length, "shortest" form; e.g.,, the integer 2 and the integer 2000 occupy a different number of octets in an Avro string.

— Numbers are represented using a rather arcane "zig-zag" encoding (this notation "moves" the sign bit from its normal position as the first bit to the last bit; doing so permits removing leading zeros from the numbers, thus making their representation occupy fewer octets).

— There is not a one-to-one mapping between JSON atomic types and Avro atomic types.

— Avro data is always associated with an Avro schema. An Avro schema describes the physical structure of the corresponding Avro data and is needed merely to "unpack" Avro data because of the variable-length fields and the various encoding rules. An Avro schema may accompany each individual Avro

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

data string, or it may be specified separately and applied to all Avro data strings in, say, an Avro data file. Avro schemas tell almost nothing about the data other than how it is packed into a data string.

— Avro strings can be encoded using JSON notation (which sort of contradicts its use as a different representation for JSON data) or using a binary notation.

Readers should recognize that Avro is not a different kind of data at all. It is, instead, merely another way of representing the same data that the JSON character string format represents. (It should be noted that not all possible Avro strings can be treated as JSON data; similarly, not every character string is a valid bit of JSON data.) In this document, Avro is referenced as one possible *serialization* of JSON data; the character string format is another serialization of the same data.

4.3.3 BSON

BSON (variously pronounced as though it were spelled "bison" or as "beeson") is another data serialization system. BSON says that it is "a binary format in which zero or more key/value pairs are stored as a single entity," called a "document". BSON is used by — and apparently was created for — a specific commercial product.

BSON strings are no less difficult for human beings to read than Avro strings are, but the design of BSON is significantly different than Avro's. A BSON document is roughly a sequence of elements, each of which is introduced by a single-octet code (for example, the hexadecimal value '01' identifies the element as a double precision floating point value, which is always eight octets in length, and '0D' identifies the element as a string containing JavaScript code), followed by an optional element name, followed by the (mandatory) element value.

BSON, like Avro, is not a different kind of data, but merely provides yet another way of representing JSON data. (Also like Avro, not all BSON strings represent valid JSON data.)

4.4 Schemas

4.4.1 JSON schemata and validity

Neither RFC 8259 nor ISO/IEC 22275:2018 provide any mechanism by which JSON values can be tested for validity other than strict syntactic validity.

JSON text is sufficiently "self-describing" that data encoded in JSON is easily parsed and can often be used in application components that have no specific knowledge of the data contained therein. This last fact explains why JSON is so successful in the broader data management community, in spite of the lack of a standard way to document its structure.

There is at least one effort to define a schema for JSON JSONschema that would describe the structure of JSON data to be considered "valid" for some given application. However, there does not appear to be any significant amount of interest from the JSON community for the rapid development of a schema definition language for JSON.

While a schema language for JSON data could be useful in some circumstances, it does not appear that such a language is widely used, and ISO/IEC 9075-2 neither relies on nor creates such a language.

ISO/IEC 9075-2 uses the word "valid" to describe data instances that satisfy all JSON syntactic requirements. ISO/IEC 9075-2 specifies a new SQL predicate, IS JSON, to test the (syntactic) validity of JSON data instances.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

4.4.2 Avro schemata

Because Avro is a representation in which each "field" (a bit of JSON data) occupies no more octets than is required, using the particular encoding method for data of each type, it is not possible to simply index to a specific field in each Avro string. In fact, because of the way that Avro encodes its fields, it is not possible to scan an Avro string and identify the start of the second, third, or twenty-fifth field in that string.

Consequently, Avro specifies that each Avro value be described by an Avro schema. Avro schemata are expressed in the character representation of JSON. The schema that represents an entire Avro string is composed of "smaller" schemata that represent each field in the Avro string. The schema describes each field by its name, data type, and (if not already unambiguous) length. Thus, an application wishing to access the fields in an Avro string must first parse the schema of that string, then use that information to locate the desired fields in the string and to "decode" the field contents into a value of a JSON data type.

Because each JSON text can be of a different size or contain different components, one might wish to provide a different schema for each JSON text...a schema that uniquely describes that text and not (necessarily) any other JSON text. This approach necessarily creates an increase in size of the JSON texts associated with those schemata. For small JSON texts and/or JSON texts with a great many fields, the overhead (in octets) of providing a schema for each such text in the Avro representation becomes unacceptable very quickly. For very large JSON texts, particularly with small numbers of (very large) fields, the presence of a per-text schema may be perfectly reasonable.

Avro does not require that a schema be provided along with each JSON text, individually. It does permit that approach, but it also allows for a single schema to describe all of the Avro-represented JSON texts in a "container file." As long as all of the texts in that container file are sufficiently alike, a single schema is adequate — and, of course, produces much-reduced overhead. In the context of this document, an entire column containing the Avro representation of JSON texts acts as the "container file". The Avro schema associated with such a column is part of the metadata describing the column.

In the preceding paragraph, the phrase "sufficiently alike" was used. That phrase means that each object/array in the rows of a column contain the same number of members/elements, each having the same name (for objects) and data type. It also means that the lengths (number of octets) of each member/element must be respectively the same, and that is not always easy to ensure.

NOTE 1 — Therefore, the JSON objects "**{ "name" : "Joe", "salary" : 85000 }**" and "**{ "name" : "Bob", "salary" : 78000 }**" are "sufficiently alike", but "**{ "name" : "Ann", "bonus" : 85000 }**" is not.

4.4.3 BSON schemata

Because the BSON representation of JSON contains a one-octet code as the first octet of every field, it is possible to scan a BSON value and uniquely identify each field and its data type. Consequently, no sort of schema is required for BSON-represented JSON data.

However, BSON — like Avro — uses variable-length fields, so that corresponding members/elements in different objects/arrays can have significantly different lengths. Scanning BSON strings to locate those field codes (and thus the fields themselves) requires CPU cycles. BSON might benefit from having a schema capability somewhat similar to Avro's, but it is certainly not necessary.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

4.5 Why does JSON matter in the context of SQL? What is JSON's relationship to NoSQL?

It is unclear that JSON and SQL ISO/IEC 9075-2 have any inherent relationship, but it is equally clear that the technical, business, and government worlds are increasingly using both kinds of data in their environments. Individual applications are required to access and manipulate all of these kinds of data, often concurrently. There are great benefits when a single data management system can concurrently handle all of the data. Among the benefits are: reduced administrative costs, improved security and transaction management, better performance and greater possibilities for optimization, better resource allocation, and increased developer productivity.

Incorporation of JSON into the SQL umbrella offers implementers and users alike the benefits described above. That fact easily justifies the relatively small increase in size and complexity of the SQL standard, especially when the approach described in this document is used.

NoSQL database systems NoSQLDB are generally characterized by the following attributes:

— They do not use the relational model (they also do not use a number of other data models).

— They tend to be focused on "big data" and on applications for which "approximate" answers are often adequate.

— They are optimized for data retrieval, not for data creation or update, nor on the relationships between data.

— They usually do not use ACID Distributed Transaction Processing transactions; instead, they may offer transactional models that result in "eventual consistency".

— They tend to be designed using distributed, fault-tolerant, highly parallel hardware and software architectures.

NoSQL database systems come in a very wide variety of kinds, based on their targeted marketplaces, data models, and application requirements. They have been crudely taxonomized into:

— Key-value stores

— "BigTable" stores

— Document stores

— Graph databases

Key-value stores provide exactly the capability that the name implies: applications provide a key and are given a value in return. Key-value stores may manage only "flat" pairs, or they may manage hierarchical pairs.

BigTable stores implement multi-dimensional arrays such that applications provide one or more index values (often strings used as key values, instead of numeric indexes) that together provide the location of a cell, the value of which is returned to the application.

Document stores, contrary to what the name many suggest to many, do not necessarily store textual documents such as books, specifications, or magazines; instead, they store data that may be traditional textual documents or organized collections of structured (and semi-structured) data.

Graph databases provide a way to store data that is generally linked together into graphs (often directed graphs, sometimes trees in particular).

Many, but hardly all, NoSQL database systems manage data represented in JSON, especially key-value stores and document stores.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

The rapidly increasing use of JSON to interchange data between Web applications has caught the attention of academics, technologists, developers, and even enterprise management. SQL database implementers are increasingly convinced that they must support JSON data "natively". This document describes the approach taken in ISO/IEC 9075-2, which allows such implementers to provide that support in a common manner.

4.6 Use cases for JSON support in SQL

4.6.1 Introduction to the use cases

There are three primary use cases:

— JSON data ingestion and storage

— JSON data generation from relational data

— Querying JSON as persistent semi-structured data model instances

The following sections discuss these use cases in greater detail.

4.6.2 JSON data ingestion and storage

Defining a new native data type is both costly to implement for SQL-implementations and costly to adopt by database tools and applications alike. Thus, ISO/IEC 9075-2 took the approach to ingest JSON data as character strings or binary strings that are then stored in ordinary SQL columns of some existing string type. When such data is retrieved from those columns for use in JSON-based functions, it is transformed (parsed) into instances of an internal SQL/JSON data model that is never directly exposed to the application author.

4.6.3 JSON data generation from relational data

This use case asks "How can JSON data instances be (declaratively) generated from relational tables for data export?" Applications that are based on JSON data not only want to store and retrieve such data upon demand, but they typically want their queries against such data to provide results in the same form — JSON. Although it is trivial to provide procedural mechanisms by which applications can laboriously (and with many likely errors) construct JSON data, SQL's declarative nature suggests that JSON objects and arrays should be generated instead of potentially lengthy character strings that represent such objects and arrays. (Readers are cautioned not to misinterpret this use case as requiring provision of a "bulk JSON data export" facility.)

ISO/IEC 9075-2 addresses this use case by providing several functions that transform the data stored in SQL tables into instances of the internal SQL/JSON data model that can, if needed, be serialized back into character string form. This document provides several examples for those functions.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

4.6.4 Querying JSON as persistent semi-structured data model instances

This use case explores how JSON data that is stored directly in SQL tables can be queried. Direct mapping of entire SQL tables into single (or, necessarily, multiple) JSON objects or arrays has not been specified in ISO/IEC 9075-2, although support is provided for such mappings when needed by applications. Instead, JSON data is stored within an opaque data type (specifically an SQL string or Large Object) that can be manipulated through the functional interface specified in ISO/IEC 9075-2, as illustrated by examples in this document.

4.7 What features address those use cases?

4.7.1 Addressing the use cases

The use cases are addressed as follows:

— SQL query expressions can access JSON data according to its structure (e.g.,, using the names of key-value pairs in JSON objects, positions in JSON arrays, *etc.*).

— SQL queries can generate JSON data directly for return to invokers of those queries.

— SQL tables can store JSON data persistently.

In the next sections, each of these “macro-features” are explored in turn.

4.7.2 Storing JSON data in an SQL table

The approach taken by ISO/IEC 9075-2 and described in this document is to store JSON data into character string columns or binary string columns that are defined within ordinary SQL tables. That permits those JSON data to participate in SQL queries (and, importantly, SQL-transactions) in the same manner as the data stored in other columns of the same tables. By choosing to use columns whose declared types are string types, the standardization (and implementation) overhead of creating a new built-in SQL data type is avoided without losing any significant advantages of a built-in type.

Applications, however, are not expected to provide detailed code to manipulate JSON data in those strings directly in the form of character string operations. ISO/IEC 9075-2 provides a number of built-in SQL functions that access (query) JSON data stored in such columns. These functions are described in Clause 6, “SQL/JSON functions”.

4.7.3 Generating JSON in an SQL query

ISO/IEC 9075-2 provides built-in functions that generate JSON objects and JSON arrays as the results of SQL queries, whether the source of the data queried is JSON data or ordinary SQL data. These functions are also described in Clause 6, “SQL/JSON functions”.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

4.7.4 Querying JSON data in SQL tables using SQL

Since there is no well-known, standardized, and universally accepted language that queries data represented in the JSON data model, ISO/IEC 9075-2 defines the SQL/JSON path language, which is embedded in SQL operators that address the use cases outlined in Subclause 4.6, "Use cases for JSON support in SQL".

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

5 The SQL/JSON data model

5.1 Introduction to the SQL/JSON data model

The SQL/JSON data model used by SQL/JSON path language can be summarized as "sequences of items". The items are SQL scalar values with an additional SQL/JSON null value, and composite data structures using JSON arrays and objects.

To clearly distinguish between JSON values "outside" the SQL-environment and their analogs "inside" the SQL-environment, the following conventions are adopted:

— The modifier "JSON" refers to constructs within a character or binary string that conforms to RFC 8259.

— The modifier "SQL/JSON" refers to JSON constructs within the SQL-environment.

The relationship between "JSON" outside of and "SQL/JSON" within the SQL-environment is crudely illustrated in Figure 1, "Relationships between "JSON" and "SQL/JSON"".

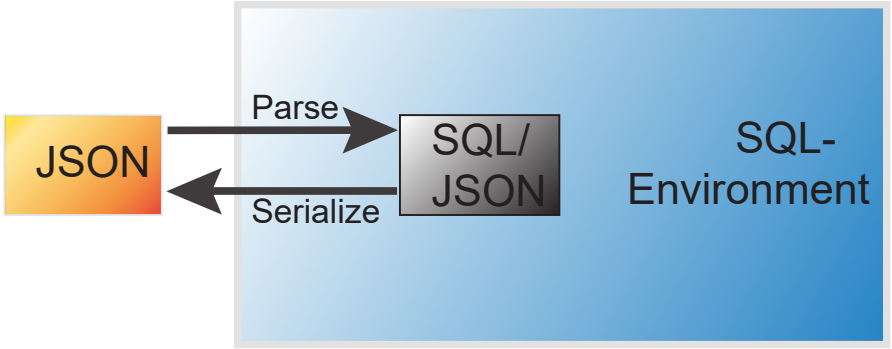

Figure 1 — Relationships between "JSON" and "SQL/JSON"

JSON data is parsed into SQL/JSON values, which can then be serialized back into JSON. All JSON data can be parsed into SQL/JSON values; however, not all SQL/JSON values can be serialized into JSON data. Parsing and serializing are addressed in more detail in Subclause 5.4, "Parsing JSON", and Subclause 5.5, "Serializing JSON", respectively.

Table 1, "JSON, SQL/JSON, and SQL (other than SQL/JSON values and counterparts)", lists various terms used in ISO/IEC 9075-2 and this document related to JSON, SQL/JSON, and SQL (other than SQL/JSON values and their corresponding counterparts).

Table 1 — JSON, SQL/JSON, and SQL (other than SQL/JSON values and counterparts)

JSON	SQL/JSON	SQL (other than SQL/JSON)
JSON array	SQL/JSON array	
JSON object	SQL/JSON object	

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

JSON	SQL/JSON	SQL (other than SQL/JSON)
JSON member	SQL/JSON member	
JSON literal `null`	SQL/JSON null	typed nulls
JSON literal `true`	*True*	*True*
JSON literal `false`	*False*	*False*
JSON number	number	non-null number
JSON string	character string	non-null character string
	datetime	non-null datetime
	SQL/JSON item	
	SQL/JSON sequence	

NOTE 2 — SQL/JSON and SQL other than SQL/JSON have precisely the same value spaces in the case of non-null scalars of Boolean, numeric, string, or datetime types.

5.2 SQL/JSON items

5.2.1 Definition of SQL/JSON items

An SQL/JSON item is defined recursively as any of the following:

1) An SQL/JSON scalar, defined as a non-null value of any of the following predefined (SQL) types: character string with character set Unicode, numeric, Boolean, or datetime.

2) An SQL/JSON null, defined as a value that is distinct from any value of any SQL type.

   NOTE 3 — An SQL/JSON null is distinct from the null value of any SQL type.

3) An SQL/JSON array, defined as an ordered list of zero or more SQL/JSON items, called the SQL/JSON elements of the SQL/JSON array.

4) An SQL/JSON object, defined as an unordered collection of zero or more SQL/JSON members, where an SQL/JSON member is a pair whose first value is a character string with character set Unicode and whose second value is an SQL/ JSON item. The first value of an SQL/JSON member is called the *key* and the second value is called the *bound value*.

   NOTE 4 — RFC 8259, section 4, "Objects", says "The names within an object SHOULD be unique". Thus, non-unique keys are permitted but not advised. The user may use the WITH UNIQUE KEYS clause in the <JSON predicate> to check for uniqueness if desired.

Two SQL/JSON items are comparable if one of them is an SQL/JSON null, or if both are in one of these types: character string, numeric, Boolean, DATE, TIME, TIMESTAMP.

Two SQL/JSON items *SJI1* and *SJI2* are said to be equivalent, defined recursively as follows:

1) If *SJI1* and *SJI2* are non-null values of a predefined type, then *SJI1* and *SJI2* are equivalent if they are equal.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

2) If *SJI1* and *SJI2* are the SQL/JSON null, then *SJI1* and *SJI2* are equivalent.

3) If *SJI1* and *SJI2* are both SQL/JSON arrays of the same length *N* and corresponding elements of *SJI1* and *SJI2* are equivalent, then *SJI1* and *SJI2* are equivalent.

4) If *SJI1* and *SJI2* are SQL/JSON objects with the same number of members, and there exists a bijection *B* from *SJI1* to *SJI2* mapping each SQL/JSON member *M* of *SJI1* to an SQL/JSON member *B*(*M*) of *SJI2* such that the key and bound value of *M* are equivalent to the key and bound value of *B*(*M*), respectively, for all members *M* of *SJI1* then SJI1 and SJI2 are equivalent.

An SQL/JSON sequence is an ordered list of zero or more SQL/JSON items.

NOTE 5 — There is no SQL <data type> whose value space is SQL/JSON items or SQL/JSON sequences.

SQL/JSON items are typed values in the following categories:

1) Atomic values:

   Atomic values are the non-null SQL values of the following types:

   a) Strings in a Unicode character set.

   b) Numeric values:

      i) Exact numeric values.

      ii) Approximate numeric values.

   c) Boolean values.

   d) Datetime values.

   e) The SQL/JSON null value, which is distinct from all SQL values, including the SQL null.

2) SQL/JSON arrays.

3) SQL/JSON objects.

These categories are discussed in the following subsections.

5.2.2 Atomic values

Atomic values in the SQL/JSON data model are virtually a subset of the values of <predefined type> in ISO/IEC 9075-2, Subclause 6.1, "<data type>". To recap the choices in Foundation:

```
<predefined type> ::=
    <character string type> [ CHARACTER SET <character set specification> ]
        [ <collate clause> ]
  | <national character string type> [ <collate clause> ]
  | <binary string type>
  | <numeric type>
  | <boolean type>
  | <datetime type>
  | <interval type>
```

The SQL/JSON data model differs from Foundation predefined types as follows:

1) The data model does not support <binary string type> and <interval type>.

2) Only character strings of Unicode characters are supported. The only collation is the binary (codepoint) collation of Unicode.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

3) The SQL/JSON null value is regarded as the sole value of its own type. That is, there is no null character string value, null numeric value, *etc*. In addition, JavaScript semantics are used for nulls in comparisons — the SQL/JSON null is equal to the SQL/JSON null. (See ISO/IEC 22275:2018, section 11.9.6, "The strict equality comparison algorithm", and similar sections.)

In general, operations in the path language operate on atomic values in the SQL/JSON data model with the same semantics as the corresponding operation in SQL, but note that null semantics follows ISO/IEC 22275:2018.

Datetimes have no serialized representation in JSON. They are part of the SQL/JSON data model to support comparison predicates after converting JSON strings to datetime.

5.2.3 SQL/JSON arrays

An SQL/JSON array is an ordered list of zero or more SQL/JSON items in the SQL/JSON data model. When serialized, the list is separated by commas and enclosed in square brackets, for example:

```
[ 2.3, "bye bye" ]
```

SQL arrays are 1-relative, whereas ISO/IEC 22275:2018 arrays are 0-relative. SQL/JSON arrays are also 0-relative. This allows some path expression to be used in either the SQL/JSON path language or in JavaScript.

5.2.4 SQL/JSON objects

An SQL-JSON object is an unordered set of zero or more members.

NOTE 6 — Empty objects are permitted; see RFC 8259, section 4, "Objects", and ISO/IEC 22275:2018, section 15.12.1.2, "The JSON syntactic grammar".

A member is a pair of values:

1) The first value is a character string and is called the key of the member.

2) The second value is any SQL/JSON item in the SQL/JSON data model and is called the bound value of the member.

An SQL/JSON object is serialized enclosed in curly braces, with the members listed in a non-deterministic order separated by commas. Each member is serialized as its key (a character string, therefore enclosed in double quotes), a colon, and then the serialization of the (second) value of the member. For example:

```
{ "name": "Fido", "tag": 12345 }
```

5.3 SQL/JSON sequences

An SQL/JSON sequence is an ordered list of zero or more SQL/JSON items. SQL/JSON sequences do not nest; they can only be concatenated. One may think of an SQL/JSON item as equivalent to an SQL/JSON sequence of that one item, which is an acceptable mental model, but this document endeavors to always view an SQL/JSON sequence as a container of zero or more SQL/JSON items.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021

5.4 Parsing JSON

Parsing refers to the process of importing from some storage format into the SQL/JSON data model. The storage format may be JSON text stored in a Unicode character string, or it may be some binary format such as AVRO or BSON. Since no specifications of AVRO and BSON are formally referenceable, they have been left as implementation extensions in ISO/IEC 9075-2.

The conversion from JSON is straightforward, because SQL/JSON data model is basically a superset of JSON, as illustrated in Table 2, "Parallels between JSON text and SQL/JSON data model".

Table 2 — Parallels between JSON text and SQL/JSON data model

JSON text	SQL/JSON data model
true	*true*
false	*false*
null	null
string, e.g.,, "hello dolly"	string, e.g.,, 'hello dolly'
number	The format for numbers in RFC 8259 is the same as numeric literals in SQL. Consequently, a number from JSON can be transferred into SQL by simply parsing it as a <signed numeric literal>.
array	SQL/JSON array
object	SQL/JSON object

5.5 Serializing JSON

Serializing JSON refers to the process of exporting a value from the SQL/JSON data model back to some storage format. Serialization is specified only to JSON text; conversion to some other format, such as AVRO or BSON, is left as an implementation-defined extension in ISO/IEC 9075-2.

SQL/JSON datetimes cannot be serialized; neither can SQL/JSON sequences of length greater than one.

The result of serialization is an implementation-dependent string of Unicode characters that, if parsed, would restore the original value in the SQL/JSON data model.

The precise result of serialization is implementation-dependent because:

— Meaningless whitespace is not specified.

— Because of escape sequences, there are multiple ways to serialize a character string.

— ISO/IEC 22275:2018, Sections 5.1.3, "The numeric string grammar", and 9.3.1, "ToNumber applied to the string type", provide some discretion in the formatting of numbers.

— Members of an SQL/JSON object are unordered, so there are many possible permutations of the members of an object.

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: BS ISO/IEC 19075-6:2021